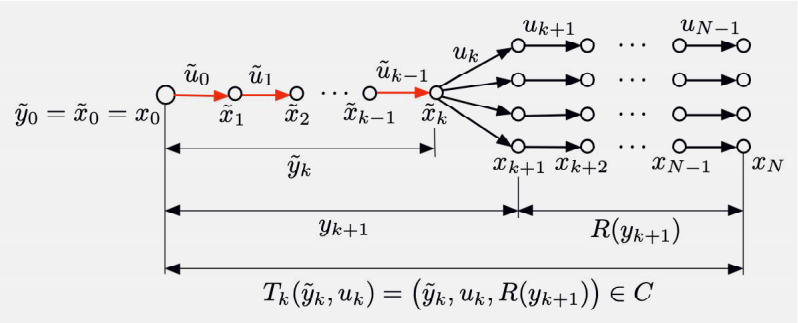

Figure 2.5.2 The trajectory generation mechanism of the rollout algorithm. At stage k, and given the current partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k),$$

which starts at $\tilde{x}_0$ and ends at $\tilde{x}_k$, we consider all possible next states $x_{k+1} = f_k(\tilde{x}_k, u_k)$, run the base heuristic starting at $y_{k+1} = (\tilde{y}_k, u_k, x_{k+1})$, and form the complete trajectory $T_k(\tilde{y}_k, u_k)$. Then the rollout algorithm:

(a) Finds $\tilde{u}_k$, the control that minimizes the cost $G\big(T_k(\tilde{y}_k, u_k)\big)$ over all u_k for which the complete trajectory $T_k(\tilde{y}_k, u_k)$ is feasible.

(b) Extends $\tilde{y}_k$ by $\big(\tilde{u}_k, f_k(\tilde{x}_k, \tilde{u}_k)\big)$ to form $\tilde{y}_{k+1}$.

This process is illustrated in Fig. 2.5.2. Note that the partial trajectory $R(y_{k+1})$ produced by the base heuristic depends on the entire partial trajectory y_{k+1}, not just the state x_{k+1}.

A complete trajectory $T_k(y_k, u_k)$ of the form (2.39) is generally feasible for only the subset $\hat{U}_k(y_k)$ of controls u_k that maintain feasibility:

$$\hat{U}_k(y_k) = \big\{u_k \mid T_k(y_k, u_k) \in C\big\}. \tag{2.40}$$

Our rollout algorithm starts from a given initial state $\tilde{y}_0 = \tilde{x}_0$, and generates successive partial trajectories $\tilde{y}_1, \ldots, \tilde{y}_N$, of the form

$$\tilde{y}_{k+1} = \big(\tilde{y}_k, \tilde{u}_k, f_k(\tilde{x}_k, \tilde{u}_k)\big), \qquad k = 0, \ldots, N-1, \tag{2.41}$$

where $\tilde{x}_k$ is the last state component of $\tilde{y}_k$, and $\tilde{u}_k$ is a control that minimizes the heuristic cost $G\big(T_k(\tilde{y}_k, u_k)\big)$ over all u_k for which $T_k(\tilde{y}_k, u_k)$ is feasible. Thus at stage k, the algorithm forms the set $U_k(\tilde{y}_k)$ [cf. Eq. (2.40)] and selects from $U_k(\tilde{y}_k)$ a control $\tilde{u}_k$ that minimizes the cost of the complete trajectory $T_k(\tilde{y}_k, u_k)$:

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big); \tag{2.42}$$

see Fig. 2.5.2. The objective is to produce a feasible final complete trajectory $\tilde{y}_N$, which has a cost $G(\tilde{y}_N)$ that is no larger than the cost of $R(\tilde{y}_0)$

produced by the base heuristic starting from $\tilde{y}_0$, i.e.,

$$G(\tilde{y}_N) \leq G\big(R(\tilde{y}_0)\big).$$

Note that $T_k(\tilde{y}_k, u_k)$ is not guaranteed to be feasible for any given u_k (i.e., may not belong to C), but we will assume that the constraint set $U_k(\tilde{y}_k)$ of problem (2.42) is nonempty, so that our rollout algorithm is well-defined. We will later modify our algorithm so that it is well-defined under the weaker assumption that just *the complete trajectory generated by the base heuristic starting from the initial state $\tilde{y}_0$ is feasible*, i.e., $R(\tilde{y}_0) \in C$.

Constrained Rollout Algorithm

The algorithm starts at stage 0 and sequentially proceeds to the last stage. At the typical stage k, it has constructed a partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k) \tag{2.43}$$

that starts at the given initial state $\tilde{y}_0 = \tilde{x}_0$, and is such that

$$\tilde{x}_{t+1} = f_t(\tilde{x}_t, \tilde{u}_t), \qquad t = 0, 1, \ldots, k-1.$$

The algorithm then forms the set of controls

$$U_k(\tilde{y}_k) = \big\{u_k \mid T_k(\tilde{y}_k, u_k) \in C\big\}$$

that is consistent with feasibility [cf. Eq. (2.40)], and chooses a control $\tilde{u}_k \in U_k(\tilde{y}_k)$ according to the minimization

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big), \tag{2.44}$$

[cf. Eq. (2.42)], where

$$T_k(\tilde{y}_k, u_k) = \Big(\tilde{y}_k, u_k, R\big(\tilde{y}_k, u_k, f_k(\tilde{x}_k, u_k)\big)\Big);$$

[cf. Eq. (2.39)]. Finally, the algorithm sets

$$\tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k), \qquad \tilde{y}_{k+1} = (\tilde{y}_k, \tilde{u}_k, \tilde{x}_{k+1}),$$

[cf. Eq. (2.41)], thus obtaining the partial trajectory $\tilde{y}_{k+1}$ to start the next stage.

It can be seen that our constrained rollout algorithm is not much more complicated or computationally demanding than its unconstrained

version where the constraint $T \in C$ is not present (as long as checking feasibility of a complete trajectory T is not computationally demanding). Note, however, that our algorithm makes essential use of the deterministic character of the problem, and does not admit a straightforward extension to stochastic problems, since checking feasibility of a complete trajectory is typically difficult in the context of these problems.

The rollout algorithm just described is illustrated in Fig. 2.5.3 for our earlier traveling salesman Example 2.5.1. Here we want to find a minimum travel cost tour that additionally satisfies a safety constraint, namely that the "safety cost" of the tour should be less than a certain threshold. Note that the minimum cost tour, ABDCA, in this example does not satisfy the safety constraint. Moreover, the tour ABCDA obtained by the rollout algorithm has barely smaller cost than the tour ACDBA generated by the base heuristic starting from A. In fact if the travel cost D→A were larger, say 25, the tour produced by constrained rollout would be more costly than the one produced by the base heuristic starting from A. This points to the need for a constrained version of the notion of sequential improvement and for a fortified variant of the algorithm, which we discuss next.

Sequential Consistency, Sequential Improvement, and the Cost Improvement Property

We will now introduce sequential consistency and sequential improvement conditions guaranteeing that the control set $U_k(\tilde{y}_k)$ in the minimization (2.44) is nonempty, and that the costs of the complete trajectories $T_k(\tilde{y}_k, \tilde{u}_k)$ are improving with each k in the sense that

$$G\big(T_{k+1}(\tilde{y}_{k+1}, \tilde{u}_{k+1})\big) \leq G\big(T_k(\tilde{y}_k, \tilde{u}_k)\big), \qquad k = 0, 1, \ldots, N-1,$$

while at the first step of the algorithm we have

$$G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big) \leq G\big(R(\tilde{y}_0)\big).$$

It will then follow that the cost improvement property

$$G(\tilde{y}_N) \leq G\big(R(\tilde{y}_0)\big)$$

holds.

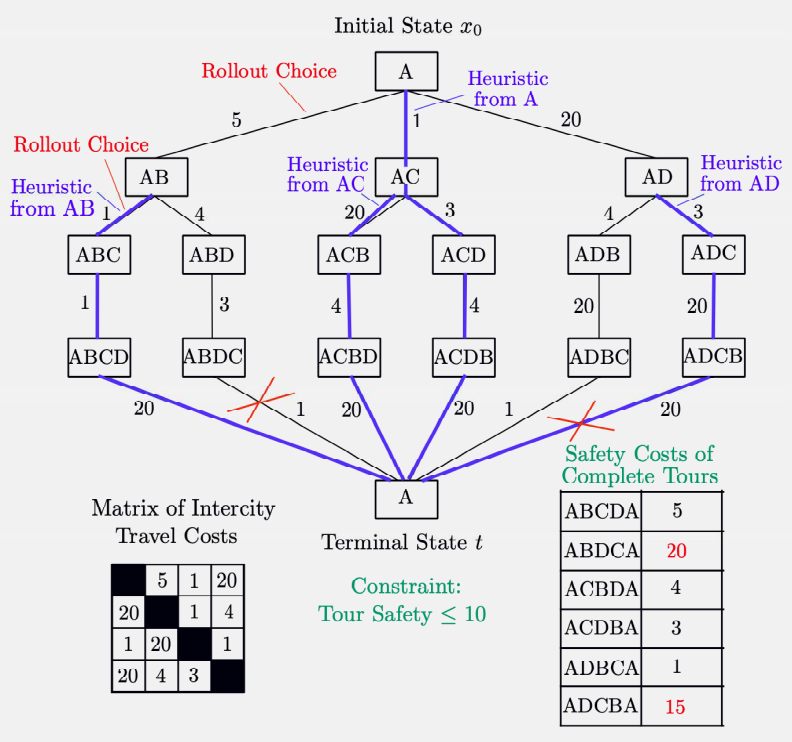

Figure 2.5.3 The constrained traveling salesman problem; cf. Example 2.5.1, and its rollout solution using the base heuristic shown, which completes a partial tour as follows:

At A it yields ACDBA.
At AB it yields ABCDA.
At AC it yields ACBDA.
At AD it yields ADCBA.

This base heuristic is not assumed to have any special structure. It is just capable of completing every partial tour without regard to any additional considerations. Thus for example the heuristic generates at A the complete tour ACDBA, and it switches to the tour ACBDA once the salesman moves to AC.

At city A, the rollout algorithm:

(a) Considers the partial tours AB, AC, and AD.

(b) Uses the base heuristic to obtain the corresponding complete tours ABCDA, ACBDA, and ADCBA.

(c) Discards ADCBA as being infeasible.

(d) Compares the other two tours, ABCDA and ACBDA, finds ABCDA to have smaller cost, and selects the partial tour AB.

(e) At AB, it considers the partial tours ABC and ABD.

(f) It uses the base heuristic to obtain the corresponding complete tours ABCDA and ABDCA, and discards ABDCA as being infeasible.

(g) It finally selects the complete tour ABCDA.

Definition 2.5.1: We say that the base heuristic is *sequentially consistent* if whenever it generates a partial trajectory

$$(x_k, u_k, x_{k+1}, u_{k+1}, \ldots, u_{N-1}, x_N),$$

starting from a partial trajectory y_k, it also generates the partial trajectory

$$(x_{k+1}, u_{k+1}, x_{k+2}, u_{k+2}, \ldots, u_{N-1}, x_N),$$

starting from the partial trajectory $y_{k+1} = \big(y_k, u_k, x_{k+1}\big)$.

As we have noted in the context of unconstrained rollout, greedy heuristics tend to be sequentially consistent. Also any policy [a sequence of feedback control functions $\mu_k(y_k)$, $k = 0, 1, \ldots, N-1$] for the DP problem of minimizing the terminal cost $G(y_N)$ subject to the system equation

$$y_{k+1} = \big(y_k, u_k, f_k(x_k, u_k)\big)$$

and the feasibility constraint $y_N \in C$ can be seen to be sequentially consistent. For an example where sequential consistency is violated, consider the base heuristic of the traveling salesman Example 2.5.1. From Fig. 2.5.3, it can be seen that the base heuristic at A generates ACDBA, but from AC it generates ACBDA, thus violating sequential consistency.

For a given partial trajectory y_k, let us denote by $y_k \cup R(y_k)$ the complete trajectory obtained by joining y_k with the partial trajectory generated by the base heuristic starting from y_k. Thus if

$$y_k = (x_0, u_0, \ldots, u_{k-1}, x_k)$$

and

$$R(y_k) = (x_k, u_k, \ldots, u_{N-1}, x_N),$$

we have

$$y_k \cup R(y_k) = (x_0, u_0, \ldots, u_{k-1}, x_k, u_k, \ldots, u_{N-1}, x_N).$$

Definition 2.5.2: We say that the base heuristic is *sequentially improving* if for every $k = 0, 1, \ldots, N-1$ and partial trajectory y_k for which $y_k \cup R(y_k) \in C$, the set $\hat{U}_k(y_k)$ is nonempty, and we have

$$G\big(y_k \cup R(y_k)\big) \geq \min_{u_k \in \hat{U}_k(y_k)} G\big(T_k(y_k, u_k)\big). \tag{2.45}$$

Note that for a base heuristic that is not sequentially consistent, the condition $y_k \cup R(y_k) \in C$ does not imply that the set $\hat{U}_k(y_k)$ is nonempty. The reason is that starting from the next state

$$y_{k+1} = \big(y_k, u_k, f_k(x_k, u_k)\big),$$

the base heuristic may generate a different trajectory than from y_k, even if it applies u_k at y_k. Thus we need to include nonemptiness of $\hat{U}_k(y_k)$ as a requirement in the preceding definition of sequential improvement (in the fortified version of the algorithm to be discussed shortly, this requirement will be removed).

On the other hand, if the base heuristic is sequentially consistent, it is also sequentially improving. The reason is that for a sequentially consistent heuristic, $y_k \cup R(y_k)$ is equal to one of the trajectories contained in the set

$$\big\{T_k(y_k, u_k) \mid u_k \in \hat{U}_k(y_k)\big\}.$$

Our main result is contained in the following proposition.

Proposition 2.5.1: (Cost Improvement for Constrained Rollout) Assume that the base heuristic is sequentially improving and generates a feasible complete trajectory starting from the initial state $\tilde{y}_0 = \tilde{x}_0$, i.e., $R(\tilde{y}_0) \in C$. Then for each k, the set $U_k(\tilde{y}_k)$ is nonempty, and we have

$$\begin{aligned} G\big(R(\tilde{y}_0)\big) &\geq G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big) \\ &\geq G\big(T_1(\tilde{y}_1, \tilde{u}_1)\big) \\ &\geq \cdots \\ &\geq G\big(T_{N-1}(\tilde{y}_{N-1}, \tilde{u}_{N-1})\big) \\ &= G(\tilde{y}_N), \end{aligned}$$

where

$$T_k(\tilde{y}_k, \tilde{u}_k) = \big(\tilde{y}_k, \tilde{u}_k, R(\tilde{y}_{k+1})\big);$$

cf. Eq. (2.39). In particular, the final trajectory $\tilde{y}_N$ generated by the constrained rollout algorithm is feasible and has no larger cost than the trajectory $R(\tilde{y}_0)$ generated by the base heuristic starting from the initial state.

Proof: Consider $R(\tilde{y}_0)$, the complete trajectory generated by the base heuristic starting from $\tilde{y}_0$. Since $\tilde{y}_0 \cup R(\tilde{y}_0) = R(\tilde{y}_0) \in C$ by assumption, it follows from the sequential improvement definition, that the set $U_0(\tilde{y}_0)$ is nonempty and we have

$$G\big(R(\tilde{y}_0)\big) \geq G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big),$$

[cf. Eq. (2.45)], while $T_0(\tilde{y}_0, \tilde{u}_0) \in C$.

The preceding argument can be repeated for the next stage, by replacing $\tilde{y}_0$ with $\tilde{y}_1$, and $R(\tilde{y}_0)$ with $T_0(\tilde{y}_0, \tilde{u}_0)$. Since $\tilde{y}_1 \cup R(\tilde{y}_1) = T_0(\tilde{y}_0, \tilde{u}_0) \in C$, from the sequential improvement definition, the set $U_1(\tilde{y}_1)$ is nonempty and we have

$$G\big(T_0(\tilde{y}_0, \tilde{u}_0)\big) = G\big(\tilde{y}_1 \cup R(\tilde{y}_1)\big) \geq G\big(T_1(\tilde{y}_1, \tilde{u}_1)\big),$$

[cf. Eq. (2.45)], while $T_1(\tilde{y}_1, \tilde{u}_1) \in C$. Similarly, the argument can be successively repeated for every k, to verify that $U_k(\tilde{y}_k)$ is nonempty and that $G\big(T_k(\tilde{y}_k, \tilde{u}_k)\big) \geq G\big(T_{k+1}(\tilde{y}_{k+1}, \tilde{u}_{k+1})\big)$ for all k. **Q.E.D.**

Proposition 2.5.1 establishes the fundamental cost improvement property for constrained rollout under the sequential improvement condition. On the other hand we may construct examples where the sequential improvement condition (2.45) is violated and the cost of the solution produced by rollout is larger than the cost of the solution produced by the base heuristic starting from the initial state (cf. the unconstrained rollout Example 2.3.3).

In the case of the traveling salesman Example 2.5.1, it can be verified that the base heuristic specified in Fig. 2.5.3 is sequentially improving. However, if the travel cost D→A were larger, say 25, then it can be verified that the definition of sequential improvement would be violated at A, and the tour produced by constrained rollout would be more costly than the one produced by the base heuristic starting from A.

The Fortified Rollout Algorithm and Other Variations

We will now discuss some variations and extensions of the constrained rollout algorithm. Let us first consider the case where the sequential improvement assumption is not satisfied. Then it may happen that given the current partial trajectory $\tilde{y}_k$, the set of controls $U_k(\tilde{y}_k)$ that corresponds to feasible trajectories $T_k(\tilde{y}_k, u_k)$ [cf. Eq. (2.40)] is empty, in which case the rollout algorithm cannot extend the partial trajectory $\tilde{y}_k$ further. To bypass this difficulty, we introduce a *fortified constrained rollout algorithm*, patterned after the fortified algorithm given earlier. For validity of this algorithm, *we require that the base heuristic generates a feasible complete trajectory $R(\tilde{y}_0)$ starting from the initial state $\tilde{y}_0$.*

The fortified constrained rollout algorithm, in addition to the current partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k),$$

maintains a complete trajectory $\hat{T}_k$, called *tentative best trajectory*, which is feasible (i.e., $\hat{T}_k \in C$) and agrees with $\tilde{y}_k$ up to state $\tilde{x}_k$, i.e., $\hat{T}_k$ has the form

$$\hat{T}_k = (\tilde{x}_0, \tilde{u}_0, \tilde{x}_1, \ldots, \tilde{u}_{k-1}, \tilde{x}_k, \overline{u}_k, \overline{x}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N), \tag{2.46}$$

for some $\overline{u}_k, \overline{x}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N$ such that

$$\overline{x}_{k+1} = f_k(\tilde{x}_k, \overline{u}_k), \qquad \overline{x}_{t+1} = f_t(\overline{x}_t, \overline{u}_t), \quad t = k+1, \ldots, N-1.$$

Initially, $\hat{T}_0$ is the complete trajectory $R(\tilde{y}_0)$, generated by the base heuristic starting from $\tilde{y}_0$, which is assumed to be feasible. At stage k, the algorithm forms the subset $\hat{U}_k(\tilde{y}_k)$ of controls $u_k \in U_k(\tilde{y}_k)$ such that the corresponding $T_k(\tilde{y}_k, u_k)$ is not only feasible, but also has cost that is no larger than the one of the current tentative best trajectory:

$$\hat{U}_k(\tilde{y}_k) = \Big\{ u_k \in U_k(\tilde{y}_k) \mid G\big(T_k(\tilde{y}_k, u_k)\big) \leq G(\hat{T}_k) \Big\}.$$

There are two cases to consider at state k:

(1) *The set $\hat{U}_k(\tilde{y}_k)$ is nonempty.* Then the algorithm forms the partial trajectory $\tilde{y}_{k+1} = (\tilde{y}_k, \tilde{u}_k, \tilde{x}_{k+1})$, where

$$\tilde{u}_k \in \arg \min_{u_k \in \hat{U}_k(\tilde{y}_k)} G\big(T_k(\tilde{y}_k, u_k)\big), \qquad \tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k),$$

and sets $T_k(\tilde{y}_k, \tilde{u}_k)$ as the new tentative best trajectory, i.e.,

$$\hat{T}_{k+1} = T_k(\tilde{y}_k, \tilde{u}_k).$$

(2) *The set $\hat{U}_k(\tilde{y}_k)$ is empty.* Then, the algorithm forms the partial trajectory $\tilde{y}_{k+1} = \big(\tilde{y}_k, \tilde{u}_k, \tilde{x}_{k+1}\big)$, where

$$\tilde{u}_k = \overline{u}_k, \qquad \tilde{x}_{k+1} = \overline{x}_{k+1},$$

and $\overline{u}_k, \overline{x}_{k+1}$ are the control and state subsequent to $\tilde{x}_k$ in the current tentative best trajectory $\hat{T}_k$ [cf. Eq. (2.46)], and leaves $\hat{T}_k$ unchanged, i.e.,

$$\hat{T}_{k+1} = \hat{T}_k.$$

It can be seen that the fortified constrained rollout algorithm will follow the initial complete trajectory $\hat{T}_0$, the one generated by the base heuristic starting from $\tilde{y}_0$, up to a stage k where it will discover a new feasible complete trajectory with smaller cost to replace $\hat{T}_0$ as the tentative best trajectory. Similarly, the new tentative best trajectory $\hat{T}_k$ may be subsequently replaced by another feasible trajectory with smaller cost, etc.

Note that if the base heuristic is sequentially improving, and the fortified rollout algorithm will generate the same complete trajectory as the (nonfortified) rollout algorithm given earlier, with the tentative best trajectory $\hat{T}_{k+1}$ being equal to the complete trajectory $T_k(\tilde{y}_k, \tilde{u}_k)$ for all k. The reason is that if the base heuristic is sequentially improving, the

controls $\tilde{u}_k$ generated by the nonfortified algorithm belong to the set $\hat{U}_k(\tilde{y}_k)$ [by Prop. 2.5.1, case (1) above will hold].

However, it can be verified that even when the base heuristic is not sequentially improving, the fortified rollout algorithm will generate a complete trajectory that is feasible and has cost that is no worse than the cost of the complete trajectory generated by the base heuristic starting from $\tilde{y}_0$. This is because each tentative best trajectory has a cost that is no worse than the one of its predecessor, and the initial tentative best trajectory is just the trajectory generated by the base heuristic starting from the initial condition $\tilde{y}_0$.

Tree-Based Constrained Rollout Algorithms

It is possible to improve the performance of the rollout algorithm at the expense of maintaining more than one partial trajectory. In particular, instead of the partial trajectory $\tilde{y}_k$ of Eq. (2.43), we can maintain a *tree* of partial trajectories that is rooted at $\tilde{y}_0$. These trajectories need not have equal length, i.e., they need not involve the same number of stages. At each step of the algorithm, we select a single partial trajectory from this tree, and execute the rollout algorithm's step as if this partial trajectory were the only one. Let this partial trajectory have k stages and denote it by $\tilde{y}_k$. Then we extend $\tilde{y}_k$ similar to our earlier rollout algorithm, with possibly multiple feasible trajectories. There is also a fortified version of this algorithm where a tentative best trajectory is maintained, which is the minimum cost complete trajectory generated thus far.

The aim of the tree-based algorithm is to obtain improved performance, essentially because it can go back and extend partial trajectories that were generated and temporarily abandoned at previous stages. The net result is a more flexible algorithm that is capable of examining more alternative trajectories. Note also that there is considerable freedom to select the number of partial trajectories maintained in the tree.

We finally mention a drawback of the tree-based algorithm: it is suitable for off-line computation, but it cannot be applied in an on-line context, where the rollout control selection is made after the current state becomes known as the system evolves in real-time .

2.5.1 Constrained Rollout for Discrete Optimization and Integer Programming

As noted in Section 2.1, general discrete optimization problems may be formulated as DP problems, which in turn can be addressed with rollout. The following is an example of a classical problem that involves both discrete and continuous variables. It can also be viewed as an instance of a 0-1 integer programming problem, and in fact this is the way it is usually addressed in the literature; see e.g., the book [DrH01]. The author's rollout book [Ber20a] contains additional examples.

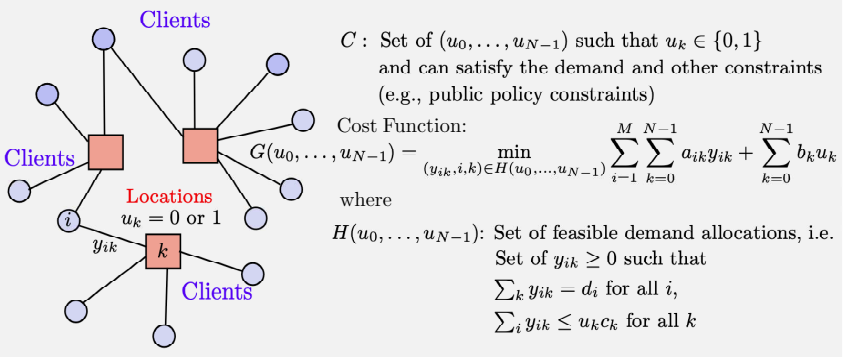

Figure 2.5.4 Schematic illustration of the facility location problem; cf. Example 2.5.2. Clients are matched to facilities, and the locations of the facilities are subject to optimization.

Example 2.5.2 (Facility Location)

We are given a candidate set of N locations, and we want to place in some of these locations a "facility" that will serve the needs of a total of M "clients." Each client $i = 1, \ldots, M$ has a demand d_i for services that may be satisfied at a location $k = 0, \ldots, N-1$ at a cost a_{ik} per unit. If a facility is placed at location k, it has capacity to serve demand up to a known level c_k.

We introduce a 0-1 integer variable u_k to indicate with $u_k = 1$ that a facility is placed at location k at a cost b_k and with $u_k = 0$ that a facility is not placed at location k. Thus if y_{ik} denotes the amount of demand of client i to be served at facility k, the constraints are

$$\sum_{k=0}^{N-1} y_{ik} = d_i, \qquad i = 1, \ldots, M, \tag{2.47}$$

$$\sum_{i=1}^{M} y_{ik} \leq c_k u_k, \qquad k = 0, \ldots, N-1, \tag{2.48}$$

together with

$$y_{ik} \geq 0, \quad u_k \in \{0, 1\}, \qquad i = 1, \ldots, M, \ k = 0, \ldots, N-1. \tag{2.49}$$

We wish to minimize the cost

$$\sum_{i=1}^{M} \sum_{k=0}^{N-1} a_{ik} y_{ik} + \sum_{k=0}^{N-1} b_k u_k \tag{2.50}$$

subject to the preceding constraints; see Fig. 2.5.4. The essence of the problem is to place enough facilities at favorable locations to satisfy the clients' demand

at minimum cost. This can be a very difficult mixed integer programming problem.

On the other hand, when all the variables u_k are fixed at some 0 or 1 values, the problem belongs to the class of *linear transportation* problems (see e.g., [Ber98]), and can be solved by fast polynomial algorithms. Thus the essential difficulty of the problem is how to select the integer variables u_k, $k = 0, \ldots, N-1$. This can be viewed as a discrete optimization problem of the type discussed in Section 1.6.3 (cf. Fig. 1.6.2). In terms of the notation of this figure, the control components are $u_0, \ldots, u_{N-1}$, where u_k can take the values 0 or 1.

To address the problem suboptimally by rollout, we must define a base heuristic at a "state" $(u_0, \ldots, u_k)$, where $u_j = 1$ or $u_j = 0$ specifies that a facility is or is not placed at location j, respectively. A suitable base heuristic at that state is to place a facility at all of the remaining locations (i.e., $u_j = 1$ for $j = k+1, \ldots, N-1$), and its cost is obtained by solving the corresponding linear transportation problem of minimizing the cost (2.50) subject to the constraints (2.47)-(2.49), with the variables u_j, $j = 0, \ldots, k$, fixed at the previously chosen values, and the variables u_j, $j = k+1, \ldots, N$, fixed at 1.

To illustrate, at the initial state where no placement decision has been made, we set $u_0 = 1$ (a facility is placed at location 0) or $u_0 = 0$ (a facility is not placed at location 0), we solve the two corresponding transportation problems, and we fix u_0, depending on which of the two resulting costs is smallest. Having fixed the status of location 0, we repeat with location 1: set the variable u_1 to 1 and to 0, solve the corresponding two transportation problems, and fix u_1, depending on which of the two resulting costs is smallest, etc.

It is easily seen that if the initial base heuristic choice (placing a facility at every candidate location) is feasible, i.e.,

$$\sum_{i=1}^{M} d_i \leq \sum_{k=0}^{N-1} c_k,$$

the rollout algorithm will yield a feasible solution with cost that is no larger than the cost corresponding to the initial application of the base heuristic. In fact it can be verified that the base heuristic here is sequentially consistent, so it is not necessary to use the fortified version of the algorithm. Regarding computational costs, the number of transportation problems to be solved is at first count $2N$, but it can be reduced to $N+1$ by exploiting the fact that one of the two transportation problems at each stage after the first has been solved at an earlier stage.

It is worth noting, for readers that are familiar with the integer programming method of branch-and-bound, that the graph of Fig. 2.1.4 corresponds to the branch-and-bound tree for the problem, so the rollout algorithm amounts to a quick (and imperfect) method to traverse the branch-and-bound tree. This observation may be useful if we wish to use integer programming techniques to add improvements to the rollout algorithm.

We finally note that the rollout algorithm requires the solution of many linear transportation problems, which are defined by fairly similar data. It is thus important to use an algorithm that is capable of using effectively the final

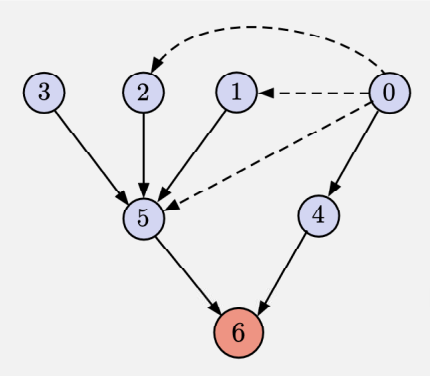

Figure 2.5.5 Schematic illustration of a constrained shortest path problem with root node $N = 6$. Given the current feasible spanning tree solution (indicated with solid line arcs), the rollout algorithm, considers a node k (in the figure $k = 0$) and the spanning tree arcs $\{u_i \mid i \neq k\}$ that are outgoing from the nodes $i \neq k$. It then considers the spanning trees that correspond to the outgoing arcs u_k from k that do not close a cycle with the set $\{u_i \mid i \neq k\}$ and are feasible [in the figure, these are the arcs indicated with broken lines, plus the arc (0,4)], and selects the arc that forms a spanning tree solution of minimum cost.

solution of one transportation problem as a starting point for the solution of the next. The auction algorithm for transportation problems (Bertsekas and Castañon [BeC89]) is particularly well-suited for this purpose.

Example 2.5.3 (Constrained Shortest Paths and Directed Spanning Trees)

Let us consider a spanning tree-type problem involving a directed graph with nodes $0, 1, \ldots, N$. At each node $k \in \{0, \ldots, N-1\}$ there is a set of outgoing arcs $u_k \in U_k$. Node N is special: it is viewed as a "root" node and has no outgoing arc. We are interested in collections of arcs involving a single outgoing arc per node,

$$u = (u_0, \ldots, u_{N-1})$$

with $u_k \in U_k$, $k = 0, \ldots, N-1$. We require that these arcs do not form a cycle, so that u specifies a directed spanning tree that is rooted at node N. Note that for every node k, such a spanning tree specifies a unique path that starts at k, lies on the spanning tree, and ends at node N. We wish to find u that minimizes a given cost function $G(u)$ subject to certain additional constraints, which we do not specify further. The set of all constraints on u (including the constraint that the arcs form a directed spanning tree) is denoted abstractly as $u \in U$, so the problem comes within our constrained optimization framework of this section.

Note that this problem contains as a special case the classical shortest path problem, where we have a length for every arc and the objective is

to find a tree of shortest paths to node N from all the nodes $0, \ldots, N-1$. Here U is just the constraint that the set of arcs $u = (u_0, \ldots, u_{N-1})$ form a directed spanning tree that is rooted at node N, and $G(u)$ is the sum of the lengths of all the paths specified by u, summed over all the start nodes $k = 0, \ldots, N-1$. Other shortest path-type problems, involving constraints, are included as special cases. For example, there may be a constraint that all the paths to N that are specified by the spanning tree corresponding to u contain a number of arcs that does not exceed a given upper bound.

Suppose that we have an initial solution/directed spanning tree

$$\bar{u} = (\bar{u}_0, \ldots, \bar{u}_{N-1}),$$

which is feasible (note here that finding such an initial solution may be a challenge). Let us apply the constrained rollout algorithm with a base heuristic that operates as follows: given a partial trajectory

$$y_k = (u_0, \ldots, u_{k-1}),$$

i.e., a sequence of k arcs, each outgoing from one of the nodes $0, \ldots, k-1$, it generates the complete trajectory/directed spanning tree

$$(u_0, \ldots, u_{k-1}, \bar{u}_k, \ldots, \bar{u}_{N-1}).$$

Thus the rollout algorithm, given a partial trajectory

$$\tilde{y}_k = (\tilde{u}_0, \ldots, \tilde{u}_{k-1}),$$

considers the set $\hat{U}_k(\tilde{y}_k)$ of all outgoing arcs u_k from node k, such that the complete trajectory

$$(\tilde{y}_k, u_k, \bar{u}_{k+1}, \ldots, \bar{u}_{N-1})$$

is feasible. It then selects the arc $u_k \in \hat{U}_k(\tilde{y}_k)$ that minimizes the cost

$$G(\tilde{y}_k, u_k, \bar{u}_{k+1}, \ldots, \bar{u}_{N-1});$$

see Fig. 2.5.5. It can be seen by induction, starting from $\bar{u}$, that the set of arcs $\hat{U}_k(\tilde{y}_k)$ is nonempty, and that the algorithm generates a sequence of feasible solutions/spanning trees, each with cost no worse than the preceding one.

Note that throughout the rollout process, a rooted spanning tree is maintained, and at each stage k, a single arc $\bar{u}_k$ that is outgoing from node k is replaced by the outgoing arc $\tilde{u}_k$. Thus two successive rooted spanning trees generated by the algorithm, differ by at most a single arc.

An interesting aspect of this rollout algorithm is that it can be applied multiple times with the final solution of one rollout application used to specify the base heuristic of the next rollout application. Moreover, a different order of nodes may be used in each rollout application. This can be viewed as a form of policy iteration, of the type that we have discussed. The algorithm will eventually terminate, in the sense that it can make no further progress. More irregular/heuristic orders of node selections are also possible; for example some nodes may be selected multiple times before others will be selected for the first time. However, there is no guarantee that the final solution thus obtained will be optimal.

2.6 SMALL STAGE COSTS AND LONG HORIZON - CONTINUOUS-TIME ROLLOUT

Let us consider the deterministic one-step approximation in value space scheme

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k)\big) \Big]. \tag{2.51}$$

In the context of rollout, $\tilde{J}_{k+1}\big(f_k(x_k, u_k)\big)$ is either the cost of the trajectory generated by the base heuristic starting from the next state $f_k(x_k, u_k)$, or some approximation that may involve truncation and terminal cost function approximation, as in the truncated rollout scheme of Section 2.3.5.

There is a special difficulty within this context, which is often encountered in practice. It arises when the cost per stage $g_k(x_k, u_k)$ is either 0 or is small relative to the cost-to-go approximation $\tilde{J}_{k+1}\big(f_k(x_k, u_k)\big)$. Then there is a potential pitfall to contend with: *the cost approximation errors that are inherent in the term* $\tilde{J}_{k+1}\big(f_k(x_k, u_k)\big)$ *may overwhelm the first stage cost term* $g_k(x_k, u_k)$, with unpredictable consequences for the quality of the one-step-lookahead policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$.

The most straightforward way to address this issue is to use longer lookahead; this is typically what is done in the context of MPC (cf. Section 1.6.7). The difficulty here is that long lookahead minimization may require extensive on-line computation. In this case, creative application of the lookahead tree pruning and incremental rollout ideas, discussed in Section 2.4.2, may be helpful.

We will next discuss the difficulty with small stage costs for an alternative context, which arises from discretization of a continuous-time optimal control problem.

Continuous-Time Optimal Control and Approximation in Value Space

Consider a problem that involves a vector differential equation of the form

$$\dot{x}(t) = h\big(x(t), u(t), t\big), \qquad 0 \leq t \leq T, \tag{2.52}$$

where $x(t) \in \Re^n$ is the state vector at time t, $\dot{x}(t) \in \Re^n$ is the vector of first order time derivatives of the state at time t, $u(t) \in U \subset \Re^m$ is the control vector at time t, where U is the control constraint set, and T is a given terminal time. Starting from a given initial state $x(0)$, we want to find a feasible control trajectory $\big\{u(t) \mid t \in [0, T]\big\}$, which together with its corresponding state trajectory $\big\{x(t) \mid t \in [0, T]\big\}$, minimizes a cost function of the form

$$G\big(x(T)\big) + \int_0^T g\big(x(t), u(t), t\big)\, dt, \tag{2.53}$$

where g represents cost per unit time, and G is a terminal cost function. This is a classical problem with a long history.

Let us consider a simple conversion of the preceding continuous-time problem to a discrete-time problem, while treading lightly over some of the associated mathematical fine points. We introduce a small discretization increment $\delta > 0$, such that $T = \delta N$ where N is a large integer, and we replace the differential equation (2.52) by

$$x_{k+1} = x_k + \delta \cdot h_k(x_k, u_k), \qquad k = 0, \ldots, N-1.$$

Here the function h_k is given by

$$h_k(x_k, u_k) = h\big(x(k\delta), u(k\delta), k\delta\big),$$

where we view $\{x_k \mid k = 0, \ldots, N-1\}$ and $\{u_k \mid k = 0, \ldots, N-1\}$ as state and control trajectories, respectively, which approximate the corresponding continuous-time trajectories:

$$x_k \approx x(k\delta), \qquad u_k \approx u(k\delta).$$

We also replace the cost function (2.53) by

$$g_N(x_N) + \sum_{k=0}^{N-1} \delta \cdot g_k(x_k, u_k),$$

where

$$g_N(x_N) = G\big(x(N\delta)\big), \qquad g_k(x_k, u_k) = g\big(x(k\delta), u(k\delta), k\delta\big).$$

Thus the approximation in value space scheme with time discretization takes the form

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U} \Big[\delta \cdot g_k(x_k, u_k) + \tilde{J}_{k+1}\big(x_k + \delta \cdot h_k(x_k, u_k)\big)\Big]; \qquad (2.54)$$

where $\tilde{J}_{k+1}$ is the function that approximates the cost-to-go starting from a state at time $k+1$. We note here that the ratio of the terms $\delta \cdot g_k(x_k, u_k)$ and $\tilde{J}_{k+1}\big(x_k + \delta \cdot h_k(x_k, u_k)\big)$ is likely to tend to 0 as $\delta \to 0$, since $\tilde{J}_{k+1}\big(x_k + \delta \cdot h_k(x_k, u_k)\big)$ ordinarily stays roughly constant at a nonzero level as $\delta \to 0$. This suggests that the one-step lookahead minimization may be degraded substantially by discretization, and other errors, including rollout truncation and terminal cost approximation. Note that a similar sensitivity to errors may occur in other discrete-time models that involve frequent selection of decisions, with cost per stage that is very small relative to the cumulative cost over many stages and/or the terminal cost.

To deal with this difficulty, we subtract the constant $\tilde{J}_k(x_k)$ in the one-step-lookahead minimization (2.54), and write

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U} \Big[\delta \cdot g_k(x_k, u_k) + \Big(\tilde{J}_{k+1}\big(x_k + \delta \cdot h_k(x_k, u_k)\big) - \tilde{J}_k(x_k)\Big)\Big]; \tag{2.55}$$

since $\tilde{J}_k(x_k)$ does not depend on u_k, the results of the minimization are not affected. Assuming $\tilde{J}_k$ is differentiable with respect to its argument, we can write

$$\tilde{J}_{k+1}\big(x_k + \delta \cdot h_k(x_k, u_k)\big) - \tilde{J}_k(x_k) \approx \delta \cdot \nabla_x \tilde{J}_k(x_k)' h_k(x_k, u_k),$$

where $\nabla_x \tilde{J}_k$ denotes the gradient of J_k (a column vector), and prime denotes transposition. By dividing with δ, and taking informally the limit as $\delta \to 0$, we can write the one-step lookahead minimization (2.55) as

$$\tilde{\mu}(t) \in \arg\min_{u(t) \in U} \Big[g\big(x(t), u(t), t\big) + \nabla_x \tilde{J}_t\big(x(t)\big)' h\big(x(t), u(t), t\big)\Big], \tag{2.56}$$

where $\tilde{J}_t(x)$ is the continuous-time cost function approximation and $\nabla_x \tilde{J}_t(x)$ is its gradient with respect to x. This is the correct analog of the approximation in value space scheme (2.51) for continuous-time problems.

Rollout for Continuous-Time Optimal Control

In view of the value approximation scheme of Eq. (2.56), it is natural to speculate that the continuous-time analog of rollout with a base policy of the form

$$\pi = \Big\{\mu_t\big(x(t)\big) \mid 0 \le t \le T\Big\}, \tag{2.57}$$

where $\mu_t\big(x(t)\big) \in U$ for all $x(t)$ and t, has the form

$$\tilde{\mu}_t\big(x(t)\big) \in \arg\min_{u(t) \in U} \Big[g\big(x(t), u(t), t\big) + \nabla_x J_{\pi,t}\big(x(t)\big)' h\big(x(t), u(t), t\big)\Big]. \tag{2.58}$$

Here $J_{\pi,t}\big(x(t)\big)$ is the cost of the base policy π starting from state $x(t)$ at time t, and satisfies the terminal condition

$$J_{\pi,T}\big(x(T)\big) = G\big(x(T)\big).$$

Computationally, the inner product in the right-hand side of the above minimization can be approximated using the finite difference formula

$$\nabla_x J_{\pi,t}\big(x(t)\big)' h\big(x(t), u(t), t\big) \approx \frac{J_{\pi,t}\Big(x(t) + \delta \cdot h\big(x(t), u(t), t\big)\Big) - J_{\pi,t}\big(x(t)\big)}{\delta},$$

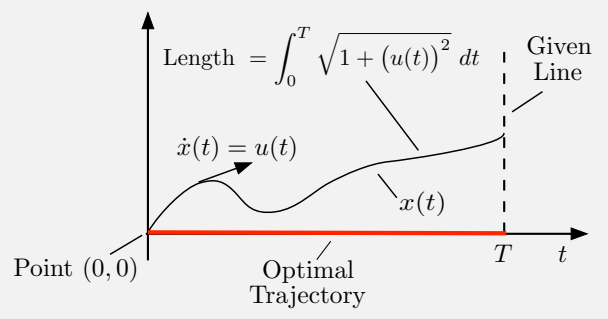

Figure 2.6.1 Problem of finding a curve of minimum length from a given point to a given line, and its formulation as a calculus of variations problem.

which can be calculated by running the base policy π starting from $x(t)$ and from $x(t) + \delta \cdot h\big(x(t), u(t), t\big)$. (This finite differencing operation may involve tricky computational issues, but we will not get into this.)

An important question is how to select the base policy π. A choice that is often sensible and convenient is to choose π to be a "short-sighted" policy, which takes into account the "short term" cost from the current state (say for a very small horizon starting from the current time t), but ignores the remaining cost. An extreme case is the *myopic* policy, given by

$$\mu_t\big(x(t)\big) \in \arg\min_{u \in U} g\big(x(t), u(t), t\big).$$

This policy is the continuous-time analog of the greedy policy that we discussed in the context of discrete-time problems, and the traveling salesman Example 1.2.3 in particular.

The following example illustrates the rollout algorithm (2.58) with a problem that has a special property: the base policy cost $J_{\pi,t}\big(x(t)\big)$ is independent of $x(t)$ (it depends only on t), so that $\nabla_x J_{\pi,t}\big(x(t)\big) \equiv 0$. In this case, in view of Eq. (2.56), the rollout policy is myopic. It turns out that the optimal policy in this example is also myopic, so that the rollout policy is optimal, even though the base policy is very poor.

Example 2.6.1 (A Calculus of Variations Problem)

This is a simple example from the classical context of calculus of variations (see [Ber17a], Example 7.1.3). The problem is to find a minimum length curve that starts at a given point and ends at a given line. Without loss of generality, let $(0, 0)$ be the given point, and let the given line be the vertical line that passes through $(T, 0)$, as shown in Fig. 2.6.1.

Let $\big(t, x(t)\big)$ be the points of the curve, where $0 \leq t \leq T$. The portion of the curve joining the points $\big(t, x(t)\big)$ and $\big(t + dt, x(t + dt)\big)$ can be approximated, for small dt, by the hypotenuse of a right triangle with sides dt and

$\dot{x}(t)dt$. Thus the length of this portion is

$$\sqrt{(dt)^2 + \big(\dot{x}(t)\big)^2 (dt)^2},$$

which is equal to

$$\sqrt{1 + \big(\dot{x}(t)\big)^2}\, dt.$$

The length of the entire curve is the integral over $[0, T]$ of this expression, so the problem is to

$$\begin{aligned} &\text{minimize} \quad \int_0^T \sqrt{1 + \big(\dot{x}(t)\big)^2}\, dt \\ &\text{subject to} \quad x(0) = 0. \end{aligned}$$

To reformulate the problem as a continuous-time optimal control problem, we introduce a control u and the system equation

$$\dot{x}(t) = u(t), \qquad x(0) = 0.$$

Our problem then takes the form

$$\text{minimize} \quad \int_0^T \sqrt{1 + \big(u(t)\big)^2}\, dt.$$

This is a problem that fits our continuous-time optimal control framework, with

$$h\big(x(t), u(t), t\big) = u(t), \quad g\big(x(t), u(t), t\big) = \sqrt{1 + \big(u(t)\big)^2}, \quad G\big(x(T)\big) = 0.$$

Consider now a base policy π whereby the control depends only on t and not on x. Such a policy has the form

$$\mu_t\big(x(t)\big) = \beta(t), \qquad \text{for all } x(t),$$

where $\beta(t)$ is some scalar function. For example, $\beta(t)$ may be constant, $\beta(t) \equiv \bar{\beta}$ for some scalar $\bar{\beta}$, which yields a straight line trajectory that starts at $(0, 0)$ and makes an angle ϕ with the horizontal with $\tan(\phi) = \bar{\beta}$. The cost function of the base policy is

$$J_{\pi,t}\big(x(t)\big) = \int_t^T \sqrt{1 + \beta(\tau)^2}\, d\tau,$$

which is independent of $x(t)$, so that $\nabla_x J_{\pi,t}\big(x(t)\big) \equiv 0$. Thus, from the minimization of Eq. (2.58), we have

$$\tilde{\mu}_t\big(x(t)\big) \in \arg\min_{u(t)\in\Re} \sqrt{1 + \big(u(t)\big)^2},$$

and the rollout policy is

$$\tilde{\mu}_t\big(x(t)\big) \equiv 0.$$

This is the optimal policy: it corresponds to the horizontal straight line that starts at $(0, 0)$ and ends at $(T, 0)$.

Rollout with General Base Heuristics - Sequential Improvement

An extension of the rollout algorithm (2.58) is to use a more general base heuristic whose cost function $H_t\big(x(t)\big)$ can be evaluated by simulation. This rollout algorithm has the form

$$\tilde{\mu}(t) \in \arg\min_{u(t)\in U} \Big[g\big(x(t),u(t),t\big) + \nabla_x H_t\big(x(t)\big)' h\big(x(t),u(t),t\big)\Big].$$

Here the policy cost function $J_{\pi,t}$ is replaced by a more general differentiable function H_t, obtainable through a base heuristic, which may lack the sequential consistency property that is inherent in policies.

We will now show a cost improvement property of the rollout algorithm based on the natural condition

$$H_T\big(\tilde{x}(T)\big) = G\big(\tilde{x}(T)\big), \tag{2.59}$$

and the assumption

$$\min_{u(t)\in U} \Big[g\big(x(t),u(t),t\big) + \nabla_t H_t\big(x(t)\big) + \nabla_x H_t\big(x(t)\big)' h\big(x(t),u(t),t\big)\Big] \le 0, \tag{2.60}$$

for all $\big(x(t),t\big)$, where $\nabla_x H_t$ denotes gradient with respect to x, and $\nabla_t H_t$ denotes gradient with respect to t. This assumption is the continuous-time analog of the sequential improvement condition of Definition 2.3.2 [cf. Eq. (2.13)]. Under this assumption, we will show that

$$J_{\tilde{\pi},0}\big(x(0)\big) \le H_0\big(x(0)\big), \tag{2.61}$$

i.e., the cost of the rollout policy starting from the initial state $x(0)$ is no worse than the base heuristic cost starting from the same initial state.

Indeed, let $\big\{\tilde{x}(t) \mid t \in [0,T]\big\}$ and $\big\{\tilde{u}(t) \mid t \in [0,T]\big\}$ be the state and control trajectories generated by the rollout policy starting from $x(0)$. Then the sequential improvement condition (2.60) yields

$$g\big(\tilde{x}(t),\tilde{u}(t),t\big) + \nabla_t H_t\big(\tilde{x}(t)\big) + \nabla_x H_t\big(\tilde{x}(t)\big)' h\big(\tilde{x}(t),\tilde{u}(t),t\big) \le 0$$

for all t, and by integration over $[0,T]$, we obtain

$$\int_0^T g\big(\tilde{x}(t),\tilde{u}(t),t\big)\,dt + \int_0^T \Big(\nabla_t H_t\big(\tilde{x}(t)\big) + \nabla_x H_t\big(\tilde{x}(t)\big)' h\big(\tilde{x}(t),\tilde{u}(t),t\big)\Big)\,dt \le 0. \tag{2.62}$$

The second integral above can be written as

$$\int_0^T \Big(\nabla_t H_t\big(\tilde{x}(t)\big) + \nabla_x H_t\big(\tilde{x}(t)\big)' h\big(\tilde{x}(t),\tilde{u}(t),t\big)\Big)\,dt = \int_0^T \Big(\nabla_t H_t\big(\tilde{x}(t)\big) + \nabla_x H_t\big(\tilde{x}(t)\big)' \frac{d\tilde{x}(t)}{dt}\Big)\,dt,$$

and its integrand is the total differential with respect to time: $\frac{d}{dt}\Big(H_t\big(\tilde{x}(t)\big)\Big)$. Thus we obtain from Eq. (2.62)

$$\begin{aligned}\int_0^T g\big(\tilde{x}(t),\tilde{u}(t),t\big)\,dt + \int_0^T \frac{d}{dt}\Big(H_t\big(\tilde{x}(t)\big)\Big)\,dt \\ = \int_0^T g\big(\tilde{x}(t),\tilde{u}(t),t\big)\,dt + H_T\big(\tilde{x}(T)\big) - H_0\big(\tilde{x}(0)\big) \leq 0.\end{aligned} \tag{2.63}$$

Since $H_T\big(\tilde{x}(T)\big) = G\big(\tilde{x}(T)\big)$ [cf. Eq. (2.59)] and $\tilde{x}(0) = x(0)$, from Eq. (2.63) [which is a direct consequence of the sequential improvement condition (2.60)], it follows that

$$J_{\tilde{\pi},0}\big(x(0)\big) = \int_0^T g\big(\tilde{x}(t),\tilde{u}(t),t\big)\,dt + G\big(\tilde{x}(T)\big) \leq H_0\big(x(0)\big),$$

thus proving the cost improvement property (2.61).

Note that the sequential improvement condition (2.60) is satisfied if H_t is the cost function $J_{\pi,t}$ corresponding to a base policy π. The reason is that for any policy $\pi = \big\{\mu_t(x(t)) \mid 0 \leq t \leq T\big\}$ [cf. Eq. (2.57)], the analog of the DP algorithm (under the requisite mathematical conditions) is

$$0 = g\big(x(t),\mu_t(x(t)),t\big) + \nabla_t J_{\pi,t}\big(x(t)\big) + \nabla_x J_{\pi,t}\big(x(t)\big)' h\big(x(t),\mu_t(x(t)),t\big). \tag{2.64}$$

In continuous-time optimal control theory, this is known as the *Hamilton-Jacobi-Bellman equation*. It is a partial differential equation, which may be viewed as the continuous-time analog of the DP algorithm for a single policy; there is also a Hamilton-Jacobi-Bellman equation for the optimal cost function $J_t^*\big(x(t)\big)$ (see optimal control textbook accounts, such as [Ber17a], Section 7.2, and the references cited there). As illustration, the reader may verify that the cost function of the base policy used in the calculus of variations problem of Example 2.6.1 satisfies this equation. It can be seen from the Hamilton-Jacobi-Bellman Eq. (2.64) that when $H_t = J_{\pi,t}$, the sequential improvement condition (2.60) and the cost improvement property (2.61) hold.

Approximating Cost Function Differences

The preceding analysis suggests that when dealing with a discrete-time problem with a long horizon N, a system equation $x_{k+1} = f_k(x_k,u_k)$, and a small cost per stage $g_k(x_k,u_k)$ relative to the optimal cost-to-go function $J_{k+1}^*\big(f_k(x_k,u_k)\big)$, it is worth considering an alternative implementation of the approximation in value space scheme. In particular, we should consider approximating the cost differences

$$D_k^*(x_k,u_k) = J_{k+1}^*\big(f_k(x_k,u_k)\big) - J_k^*(x_k)$$

instead of approximating the optimal cost-to-go functions $J^*_{k+1}\big(f_k(x_k,u_k)\big)$. The one-step-lookahead minimization (2.51) should then be replaced by

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \big[g_k(x_k,u_k) + \tilde{D}_k(x_k,u_k)\big],$$

where $\tilde{D}_k$ is the approximation to D^*_k.

Note also that while for continuous-time problems, the idea of approximating the gradient of the optimal cost function is essential and comes out naturally from the analysis, for discrete-time problems, approximating cost-to-go differences rather than cost functions is optional and should be considered in the context of a given problem, possibly in conjunction with increased lookahead. Methods along this line include advantage updating, cost shaping, biased aggregation, and the use of baselines, for which we refer to the books [BeT96], [Ber19a], and [Ber20a]. A special method to explicitly approximate cost function differences is *differential training*, which was proposed in the author's paper [Ber97b], and was also discussed in Section 4.3.4 of the book [Ber20a].

The Case of Zero Cost per Stage

The most extreme and challenging case of small stage costs arises when the cost per stage is zero for all states, while a nonzero cost may be incurred only at termination. This type of cost structure occurs, among others, in games such as chess and backgammon. It also occurs in several other contexts, including constraint programming problems (Section 2.1), where there is not even a terminal cost, just constraints to be satisfied.

Under these circumstances, the idea of approximating cost-to-go differences that we have just discussed may not be effective, and applying approximation in value space may involve serious challenges. An advisable remedy is to resort to long lookahead, either through multistep lookahead minimization (possibly augmented by pruning or incremental rollout ideas; cf. Section 2.4), or through some form of truncated rollout, as in the TD-Gammon program.

It may also be important to introduce a terminal cost function approximation by using problem simplification (solving a simpler problem, in place of the original), or neural network training. Aggregation, discussed in Section 3.6, is another possibility along this line.

2.7 STOCHASTIC ROLLOUT AND MONTE CARLO TREE SEARCH

We will now discuss the extension of the rollout algorithm to stochastic DP problems with a finite number of control and disturbances at every stage. We will restrict ourselves to the case where the base heuristic is a policy

$\pi = \{\mu_0, \ldots, \mu_{N-1}\}$. The rollout policy applies at state x_k the control $\tilde{\mu}_k(x_k)$ given by the minimization

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

Equivalently, the rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ is obtained by minimization over the Q-factors $Q_{k,\pi}(x_k, u_k)$ of the base policy:

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

where

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

We first establish that the cost improvement property that we showed for deterministic problems under the sequential consistency condition carries through for stochastic problems. In particular, let us 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 cost corresponding to starting the rollout algorithm at state x_k. We claim that

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k. \tag{2.65}$$

We prove this inequality by induction similar to the deterministic case [cf. Eq. (2.12)]. 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,

$$\begin{aligned} J_{k,\tilde{\pi}}(x_k) &= E\Big\{ g_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big) + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big)\Big)\Big\} \\ &\leq E\Big\{ g_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big) + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k), w_k\big)\Big)\Big\} \\ &= \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big)\Big\} \\ &\leq E\Big\{ g_k\big(x_k, \mu_k(x_k), w_k\big) + J_{k+1,\pi}\Big(f_k\big(x_k, \mu_k(x_k), w_k\big)\Big)\Big\} \\ &= J_{k,\pi}(x_k), \end{aligned} \tag{2.66}$$

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 final equality is the DP equation for the base policy π.

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

The preceding cost improvement argument assumes that the cost functions $J_{k+1,\pi}$ of the base policy are calculated exactly. In practice, truncated rollout with terminal cost function approximation and limited simulation may be used to approximate $J_{k+1,\pi}$. In this case the cost function of the rollout policy can still be viewed as the result of a Newton step in the context of an approximation in value space scheme. Moreover, the cost improvement property can still be proved under some conditions that we will not discuss in this book; see the books [Ber12], [Ber19a], and [Ber20a].

Some Rollout Examples

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, thanks to its underlying Newton step; see also the case studies referenced at the end of the chapter. To emphasize this point, we provide here an example of an nontrivial optimal stopping problem where the rollout policy is actually optimal, despite the fact that the base policy is rather naive. Such behavior is of course special and nontypical, but highlights the nature of the cost improvement property of rollout.

Example 2.7.1 (Optimal Stopping and Rollout Optimality)

Optimal stopping problems are characterized by the availability, at each state, of a control that stops the evolution of the system. We will consider a problem with two control choices: at each stage we observe the current state of the system and decide whether to continue or to stop the process. We formulate this as an N-stage problem where stopping is mandatory at or before stage N.

Consider a stationary version of the problem (state and disturbance spaces, disturbance distribution, control constraint set, and cost per stage are the same for all times). At each state x_k and at time k, if we stop, the system moves to a termination state at a cost $C(x_k)$ and subsequently remains there at no cost. If we do not stop, the system moves to state $x_{k+1} = f(x_k, w_k)$ at cost $g(x_k, w_k)$. The terminal cost, assuming stopping has not occurred by the last stage, is $C(x_N)$. An example is a problem of optimal exercise of a financial option where x is the asset's price, $C(x) = x$, and $g(x, w) \equiv 0$.

The DP algorithm (for states other than the termination state) is given by

$$J_N^*(x_N) = C(x_N), \tag{2.67}$$

$$J_k^*(x_k) = \min\Big[C(x_k),\, E\Big\{g(x_k, w_k) + J_{k+1}^*\big(f(x_k, w_k)\big)\Big\}\Big], \tag{2.68}$$

and it is optimal to stop at time k for states x in the set

$$S_k = \Big\{ x \;\Big|\; C(x) \leq E\Big\{ g(x,w) + J^*_{k+1}\big(f(x,w)\big) \Big\} \Big\}.$$

Consider now the rather primitive base policy π, whereby *we stop at every state* x. Thus we have for all x_k and k,

$$J_{k,\pi}(x_k) = C(x_k).$$

The rollout policy is stationary and can be computed on-line relatively easily, since $J_{k,\pi}$ is available in closed form. In particular, the rollout policy is to stop at x_k if

$$C(x_k) \leq E\Big\{ g(x_k,w_k) + C\big(f(x_k,w_k)\big) \Big\},$$

i.e., if x_k is in the set S_{N-1}, and otherwise to continue.

The rollout policy also has an intuitive interpretation: it stops at the states for which it is better to stop rather than continue for one more stage and then stop. A policy of this type turns out to be optimal in several types of stopping applications. Let us provide a condition that guarantees its optimality.

We have from the DP Eqs. (2.67)-(2.68),

$$J^*_{N-1}(x) \leq J^*_N(x), \qquad \text{for all } x,$$

and using this fact in the DP equation (2.68), we obtain inductively

$$J^*_k(x) \leq J^*_{k+1}(x), \qquad \text{for all } x \text{ and } k.$$

Using this fact and the definition of S_k we see that

$$S_0 \subset \cdots \subset S_k \subset S_{k+1} \subset \cdots \subset S_{N-1}. \tag{2.69}$$

We will now consider a condition guaranteeing that all the stopping sets S_k are equal. Suppose that the set S_{N-1} is *absorbing* in the sense that if a state belongs to S_{N-1} and we decide to continue, the next state will also be in S_{N-1}:

$$f(x,w) \in S_{N-1}, \qquad \text{for all } x \in S_{N-1},\ w. \tag{2.70}$$

We will show that equality holds in Eq. (2.69) and for all k we have

$$S_k = S_{N-1} = \Big\{ x \in S \;\Big|\; C(x) \leq E\Big\{ g(x,w) + C\big(f(x,w)\big) \Big\} \Big\}.$$

Indeed, by the definition of S_{N-1}, we have

$$J^*_{N-1}(x) = C(x), \qquad \text{for all } x \in S_{N-1},$$

and using Eq. (2.70) we obtain for $x \in S_{N-1}$

$$E\Big\{g(x,w) + J^*_{N-1}\big(f(x,w)\big)\Big\} = E\Big\{g(x,w) + C\big(f(x,w)\big)\Big\} \geq C(x).$$

Therefore, stopping is optimal for all $x_{N-2} \in S_{N-1}$ or equivalently $S_{N-1} \subset S_{N-2}$. This together with Eq. (2.69) implies $S_{N-2} = S_{N-1}$. Proceeding similarly, we obtain $S_k = S_{N-1}$ for all k. Thus the optimal policy is to stop if and only if the state is within the set S_{N-1}, which is precisely the set of states where the rollout policy stops.

In conclusion, if condition (2.70) holds (the one-step stopping set S_{N-1} is absorbing), the rollout policy is optimal. Moreover, the preceding analysis [cf. Eq. (2.69)] can be used to show that even if the one-step stopping set S_{N-1} is not absorbing, the rollout policy stops and is optimal within the set of states $x \in \cap_k S_k$, and correctly continues within the set of states $x \notin S_{N-1}$. Contrary to the optimal policy, it also stops within the subset of states $x \in S_{N-1}$ that are not in $\cap_k S_k$. Thus, even in the absence of condition (2.70), the rollout policy is quite sensible even though the base policy is not.

We next discuss a special case of the preceding example. Again the one-step lookahead/rollout policy is optimal, despite the fact that the base policy is poor. Related examples can be found in Chapter 3 of the DP textbook [Ber17a].

Example 2.7.2 (The Rational Burglar)

A burglar may at any night k choose to retire with his accumulated earnings x_k or enter a house and bring home a random amount w_k. However, in the latter case he gets caught with probability p, and then he is forced to terminate his activities and forfeit all of his earnings thus far. The amounts w_k are independent, identically distributed with mean $\overline{w}$. The problem is to find a policy that maximizes the burglar's expected earnings over N nights.

We can formulate this problem as a stopping problem with two actions (retire or continue) and a state space consisting of the real line, the retirement state, and a special state corresponding to the burglar getting caught. The DP algorithm is given by

$$J^*_N(x_N) = x_N,$$

$$J^*_k(x_k) = \max\Big[x_k,\, (1-p)E\big\{J^*_{k+1}(x_k + w_k)\big\}\Big].$$

The one-step stopping set is

$$S_{N-1} = \big\{x \mid x \geq (1-p)(x+\overline{w})\big\} = \left\{x \;\middle|\; x \geq \frac{(1-p)\overline{w}}{p}\right\},$$

(more accurately this set together with the special state corresponding to the burglar's arrest). Since this set is absorbing in the sense of Eq. (2.70), we see that the one-step lookahead/rollout policy by which the burglar retires when his earnings reach or exceed $(1-p)\overline{w}/p$ is optimal. Note that the base policy of the burglar is the "timid" policy of always retiring, regardless of his accumulated earnings, which is far from optimal.

2.7.1 Simplified Rollout and Policy Iteration

The cost improvement property (2.65) also holds for the simplified version of the rollout algorithm (cf. Section 2.3.4) where the rollout policy is defined by

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in \overline{U}_k(x_k)} Q_{k,\pi}(x_k, u_k), \tag{2.71}$$

for a subset $\overline{U}_k(x_k) \subset U_k(x_k)$ that contains the base policy control $\mu_k(x_k)$. The proof is obtained by replacing the last inequality in the argument of Eq. (2.66),

$$\min_{u_k \in U_k(x_k)} Q_{k,\pi}(x_k, u_k) \leq Q_{k,\pi}\big(x_k, \mu_k(x_k)\big),$$

with the inequality

$$\min_{u_k \in \overline{U}_k(x_k)} Q_{k,\pi}(x_k, u_k) \leq Q_{k,\pi}\big(x_k, \mu_k(x_k)\big).$$

The simplified rollout algorithm (2.71) may be implemented in a number of ways, including control constraint discretization/approximation, a random search algorithm, or a one-agent-at-a-time minimization process, as in multiagent rollout.

The simplified rollout idea can also be used within the infinite horizon policy iteration (PI) context. In particular, instead of the minimization

$$\tilde{\mu}(x) \in \arg \min_{u \in U(x)} E_w\Big\{g(x,u,w) + \alpha J_\mu\big(f(x,u,w)\big)\Big\}, \qquad \text{for all } x, \tag{2.72}$$

in the policy improvement operation, it is sufficient for cost improvement to generate a new policy $\tilde{\mu}$ that satisfies for all x,

$$E_w\Big\{g\big(x,\tilde{\mu}(x),w\big)+\alpha J_\mu\big(f(x,\tilde{\mu}(x),w)\big)\Big\} \leq E_w\Big\{g(x,u,w)+\alpha J_\mu\big(f(x,u,w)\big)\Big\}.$$

This cost improvement property is the critical argument for proving convergence of the PI algorithm and its variations to the optimal cost function and policy; see the corresponding proofs in the books [Ber17a] and [Ber19a].

2.7.2 Certainty Equivalence Approximations

As in the case of deterministic DP problems, it is possible to use ℓ-step lookahead, with the aim to improve the performance of the policy obtained through approximation in value space. This, however, can be computationally expensive, because the lookahead graph expands fast as ℓ increases, due to the stochastic character of the problem. Using *certainty equivalence* (CE for short) is an important approximation approach for dealing with this difficulty, as it reduces the size of the ℓ-step lookahead graph. Moreover, CE mitigates the potentially excessive simulation because it reduces the stochastic variance of the Q-factors calculated at each stage.

In the pure but somewhat flawed version of this approach, when solving the ℓ-step lookahead minimization problem, we simply replace *all* of the uncertain quantities $w_k, w_{k+1}, \ldots, w_{k+\ell-1}, \ldots, w_{N-1}$ by some nominal value $\overline{w}$, thus making that problem fully deterministic. Unfortunately, this affects significantly the character of the approximation: when w_k is replaced by a deterministic quantity the Newton step interpretation of the underlying approximation in value space scheme is lost to a great extent.

Still, we may largely correct this difficulty, while retaining substantial simplification, by using CE for *only after the first stage* of the ℓ-step lookahead. We can do this with a CE scheme whereby only the uncertain quantities $w_{k+1}, \ldots, w_{N-1}$ are replaced by a deterministic value $\overline{w}$, while w_k is treated as a stochastic quantity.†

The CE approach, first proposed in the paper by Bertsekas and Castañon [BeC99], has an important property: *it maintains the Newton step character of the approximation in value space scheme*. In particular, the function $J_{\tilde{\mu}}$ of the ℓ-step lookahead policy $\tilde{\mu}$ obtained is generated by a Newton step, applied to the function obtained by the last $\ell-1$ minimization steps (modified by CE, and applied to the terminal cost function approximation); see the monograph [Ber20a] for a discussion. Thus the benefit of the fast convergence of Newton's method is restored. In fact based on insights derived from this Newton step interpretation, it appears that the performance penalty for the CE approximation is typically small. At the same time the ℓ-step lookahead minimization involves only one stochastic step, the first one, and hence potentially a much "thinner" lookahead graph, than the ℓ-step minimization that does not involve any CE-type approximations; see Fig. 2.7.1. Moreover, the ideas of tree pruning and iterative deepening, which we have discussed in Section 2.4 for deterministic multistep lookahead, come into play when the CE approximation is used.

2.7.3 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 is obtained from

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

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

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

† Variants of the CE approach, based on less drastic simplifications of the probability distributions of the uncertain quantities, are given in the books [Ber17a], Section 6.2.2 and [Ber19a], Section 2.3.2.

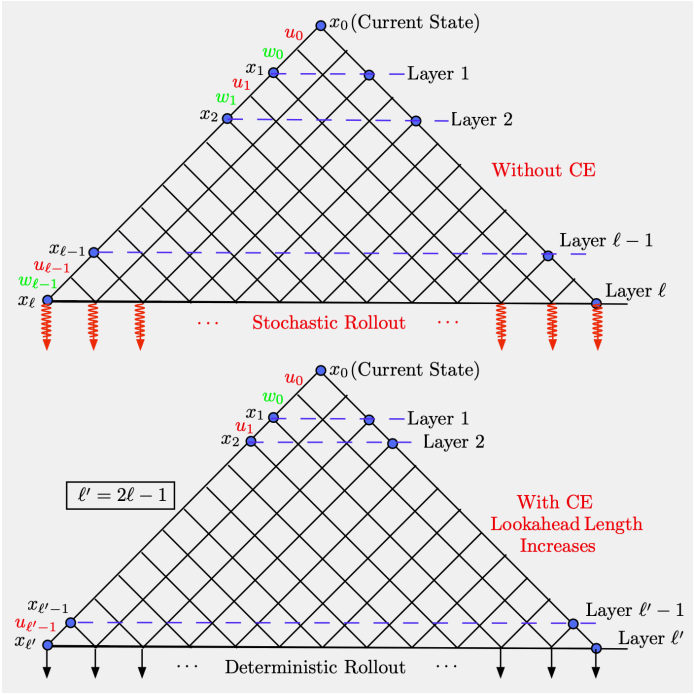

Figure 2.7.1 Illustration of multistep lookahead for stochastic problems with the CE approximation, applied at the states after the first layer of states of the multistep lookahead tree. The figure on the top (or the bottom) illustrates the lookahead tree without (or with, respectively) CE. It can be seen that with CE, the lookahead tree grows much faster (the layers contain more states). In particular, the "height" of the ℓ-step lookahead graph without CE is the same as the "height" of a ℓ'-step lookahead graph with CE, where $\ell' = 2\ell - 1$. Moreover, with a number m of controls per state, and a number n of disturbances per state-control pair, the number of leaves of the ℓ-step lookahead tree is estimated as $O(mn\ell)$ without CE and $O\big(m(n+\ell)\big)$ with CE.

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

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

where $J_{k+1,\pi}$ is the cost-to-go function of the base policy, i.e., $J_{k+1,\pi}(x_{k+1})$ is the cost of using the base policy starting from x_{k+1}. For problems with a large number of stages, it is also common to truncate the rollout trajectories and add a terminal cost function approximation as compensation for the resulting error.

By Monte Carlo averaging of the costs of the sample trajectories plus the terminal cost (if any), we obtain an approximation to the Q-factor

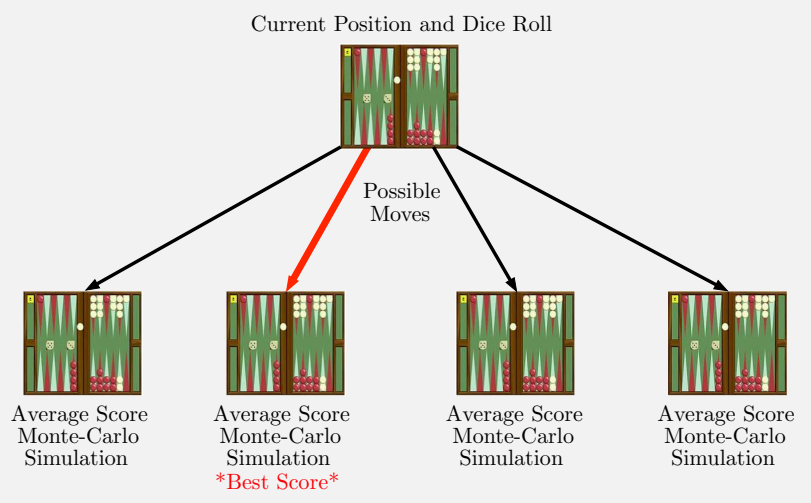

Figure 2.7.2 Illustration of rollout for backgammon. At a given position and roll of the dice, the set of all possible moves is generated, and the outcome of the game for each move is evaluated by "rolling out" (simulating to the end) many games using a suboptimal/heuristic backgammon player (the TD-Gammon player was used for this purpose in [TeG96]), and by Monte Carlo averaging the scores. The move that results in the best average score is selected for play.

$Q_{k,\pi}(x_k, u_k)$ for each $u_k \in U_k(x_k)$, denoted by $\tilde{Q}_{k,\pi}(x_k, u_k)$. We then compute the (approximate) rollout control $\tilde{\mu}_k(x_k)$ with the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_{k,\pi}(x_k, u_k). \tag{2.73}$$

Example 2.7.3 (Backgammon)

The first impressive application of rollout was given for the ancient two-player game of backgammon, in the paper by Tesauro and Galperin [TeG96]; see Fig. 2.7.2. 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 one-step and two-step lookahead with lookahead cost function approximation provided by a neural network, resulting in a backgammon program called TD-Gammon [Tes89a], [Tes89b], [Tes92], [Tes94], [Tes95], [Tes02]. TD-Gammon was trained with an approximate policy iteration method, and was used as the base policy (for each of the two players) to simulate game trajectories. The rollout algorithm also involved truncation of long game trajectories, using a terminal cost function approximation based on TD-Gammon's position evaluation. Game trajectories are of course random, since they involve the use of dice at each player's turn. Thus the scores of many trajectories have to be generated and averaged with the Monte Carlo method to assess the probability of a win from a given position.

An important issue to consider here is that backgammon is a two-player game and not an optimal control problem that involves a single decision maker. While there is a DP theory for sequential zero-sum games, this theory has not been covered in this book. Thus how are we to interpret rollout algorithms in the context of two-player games, with both players using some base policy? The answer is to view the game as a (one-player) optimal control problem, where one of the two players passively uses the base policy exclusively (TD-Gammon in the present example). The other player takes the role of the optimizer, and actively tries to improve on his base policy (TD-Gammon) by using rollout. Thus "policy improvement" in the context of the present example means that when playing against a TD-Gammon opponent, the rollout player achieves a better score on the average than if he/she were to play with the TD-Gammon strategy. In particular, the theory does not guarantee that a rollout player that is trained using TD-Gammon for both players will do better than TD-Gammon would against a non-TD-Gammon opponent. While this is a plausible practical hypothesis, it is one that can only be tested empirically. In fact relevant counterexamples have been constructed for the game of Go using "adversarial" optimization techniques; see Wang et al. [WGB22], and also our discussion on minimax problems in Section 2.12.

Most of the currently existing 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, they are too time-consuming for real-time play (without parallel computing hardware), 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 (many articles and empirical works on computer backgammon are posted on-line; see e.g., http://www.bkgm.com/articles/page07.html).

2.7.4 Variance Reduction in Rollout - Comparing Advantages

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], and Rubinstein and Kroese [RuK1]).

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

$$\tilde{Q}_{k,\pi}(x_k, u_k) - \tilde{Q}_{k,\pi}(x_k, \hat{u}_k)$$

† The situation in backgammon is exacerbated by its high branching factor, i.e., for a given position, the number of possible successor positions is quite large, as compared for example with chess.

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 $\tilde{Q}_{k,\pi}(x_k, u_k)$ may be magnified through the preceding differencing operation.

An approach to counteract this type of simulation error magnification is to approximate the Q-factor difference $\tilde{Q}_{k,\pi}(x_k, u_k) - \tilde{Q}_{k,\pi}(x_k, \hat{u}_k)$ by sampling the difference

$$C_k(x_k, u_k, \mathbf{w}_k) - C_k(x_k, \hat{u}_k, \mathbf{w}_k), \tag{2.74}$$

where $\mathbf{w}_k = (w_k, w_{k+1}, \ldots, w_{N-1})$ is the *same disturbance sequence* for the two controls u_k and $\hat{u}_k$, and

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

with $\{\mu_{k+1}, \ldots, \mu_{N-1}\}$ being the tail portion of the base policy. †

For a simple example that illustrates how this form of variance reduction works, suppose we want to calculate the difference $q_1 - q_2$ of two numbers q_1 and q_2 by subtracting two simulation samples $s_1 = q_1 + w_1$ and $s_2 = q_2 + w_2$, where w_1 and w_2 are zero mean random variables. Then $s_1 - s_2$ is unbiased in the sense that its mean is equal to $q_1 - q_2$. However, the variance of $s_1 - s_2$ decreases as the correlation of w_1 and w_2 increases. It is maximized when w_1 and w_2 are uncorrelated, and it is minimized (it is equal to 0) when w_1 and w_2 are equal.

The preceding example suggests a simulation scheme that is based on the difference (2.74) and involves a common disturbance $\mathbf{w}_k$ for u_k and $\hat{u}_k$. In particular, it may be far more accurate than the one obtained by differencing samples of $C_k(x_k, u_k, \mathbf{w}_k)$ and $C_k(x_k, \hat{u}_k, \hat{\mathbf{w}}_k)$, which involve two different disturbances $\mathbf{w}_k$ and $\hat{\mathbf{w}}_k$. Indeed, by introducing the zero mean sample errors

$$D_k(x_k, u_k, \mathbf{w}_k) = C_k(x_k, u_k, \mathbf{w}_k) - \tilde{Q}_{k,\pi}(x_k, u_k),$$

it can be seen that the variance of the error in estimating $\tilde{Q}_{k,\pi}(x_k, u_k) - \tilde{Q}_{k,\pi}(x_k, \hat{u}_k)$ with the former method will be no larger than with the latter method if and only if

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

† For this to be possible, we need to assume that the probability distribution of each disturbance w_i does not depend on x_i and u_i.

By expanding the quadratic forms and using the fact $E\{D_k(x_k, u_k, \mathbf{w}_k)\} = 0$, we see that this condition is equivalent to

$$E\{D_k(x_k, u_k, \mathbf{w}_k)D_k(x_k, \hat{u}_k, \mathbf{w}_k)\} \geq 0; \tag{2.75}$$

i.e., the errors $D_k(x_k, u_k, \mathbf{w}_k)$ and $D_k(x_k, \hat{u}_k, \mathbf{w}_k)$ being nonnegatively correlated. A little thought should convince the reader that this property is likely to hold in many types of problems.

Roughly speaking, the relation (2.75) 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

$$E\left\{\left|D_k(x_k, u_k, \mathbf{w}_k) - D_k(x_k, \hat{u}_k, \mathbf{w}_k)\right|^2\right\} \leq \gamma E\left\{\left|D_k(x_k, u_k, \mathbf{w}_k)\right|^2\right\}. \tag{2.76}$$

Then we have, by using the generic relation $ab \geq a^2 - |a| \cdot |b - a|$ for two scalars a and b,

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

from which we obtain

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

where 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. (2.76).

Thus, under the assumption (2.76), the condition (2.75) holds and guarantees that by averaging cost difference samples rather than differencing (independently obtained) averages of cost samples, the simulation error variance does not increase.

Let us finally note the potential benefit of using Q-factor differences in contexts other than rollout. In particular when approximating Q-factors $Q_{k,\pi}(x_k, u_k)$ using parametric architectures (Section 3.3 in the next chapter), it may be important to approximate and compare instead the differences

$$A_{k,\pi}(x_k, u_k) = Q_{k,\pi}(x_k, u_k) - \min_{u_k \in U_k(x_k)} Q_{k,\pi}(x_k, u_k).$$

The function $A_{k,\pi}(x_k, u_k)$ is also known as the *advantage of the pair* (x_k, u_k), and can serve just as well as $Q_{k,\pi}(x_k, u_k)$ for the purpose of comparing controls, but may work better in the presence of approximation errors. The use of advantages will be discussed further in Chapter 3.

2.7.5 Monte Carlo Tree Search

In our earlier discussion of simulation-based rollout implementation, 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:

(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 multistep lookahead 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. Such variants 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).

A simple remedy for (a) above is to use rollout trajectories of reasonably limited length, with some terminal cost approximation at the end (in an extreme case, the rollout may be skipped altogether for some states, i.e., rollout trajectories have zero length). The terminal cost function may be very simple (such as zero) or may be obtained through some auxiliary calculation. In fact the base policy used for rollout may be used to construct the terminal cost function approximation, as noted for the rollout-based backgammon algorithm of Example 2.7.3. In particular, an approximation to the cost function of the base policy may be obtained by training some approximation architecture, such as a neural network (see Chapter 3), and may be used as a terminal cost function.

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 incremental multistep rollout scheme for deterministic problems that we discussed in Section 2.4.3; see Fig. 2.4.6.

The MCTS approach can be based on sophisticated procedures for implementing and combining the ideas just described. The general idea is to use the interim results of the computation and statistical tests to focus the simulation effort along the most promising directions. Thus to implement MCTS with multistep lookahead, one needs to maintain a lookahead tree, which is expanded as the relevant Q-factors are evaluated by simulation, and which balances *the competing desires of exploitation and exploration* (generate and evaluate controls that seem most promising in terms of performance versus assessing the potential of inadequately explored controls). Ideas that were developed in the context of multiarmed bandit problems have played an important role in the construction of this type of MCTS procedures (see the end-of-chapter references).

In the simple case of one-step lookahead, with Q-factors calculated by Monte Carlo simulation, MCTS fundamentally aims to find efficiently the minimum of the expected values of a finite number of random variables. This is illustrated in the following example.

Example 2.7.4 (Statistical Tests for Adaptive Sampling with One-Step Lookahead)

Let us consider a typical one-step lookahead selection strategy that is based on adaptive sampling. We are at a state x_k and we try to find a control $\tilde{u}_k$ that minimizes an approximate Q-factor

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

over $u_k \in U_k(x_k)$, with $\tilde{Q}_k(x_k, u_k)$ computed by averaging samples of the expression within braces. We assume that $U_k(x_k)$ contains m elements, which for simplicity are denoted $1, \ldots, m$. At the ℓth sampling period, knowing the outcomes of the preceding sampling periods, we select one of the m controls, say i_ℓ, and we draw a sample of $\tilde{Q}_k(x_k, i_\ell)$, whose value is denoted by S_{i_ℓ}. Thus after the nth sampling period we have an estimate $Q_{i,n}$ of the Q-factor of each control $i = 1, \ldots, m$ that has been sampled at least once, given by

$$Q_{i,n} = \frac{\sum_{\ell=1}^{n} \delta(i_\ell = i) S_{i_\ell}}{\sum_{\ell=1}^{n} \delta(i_\ell = i)},$$

where

$$\delta(i_\ell = i) = \begin{cases} 1 & \text{if } i_\ell = i, \\ 0 & \text{if } i_\ell \neq i. \end{cases}$$

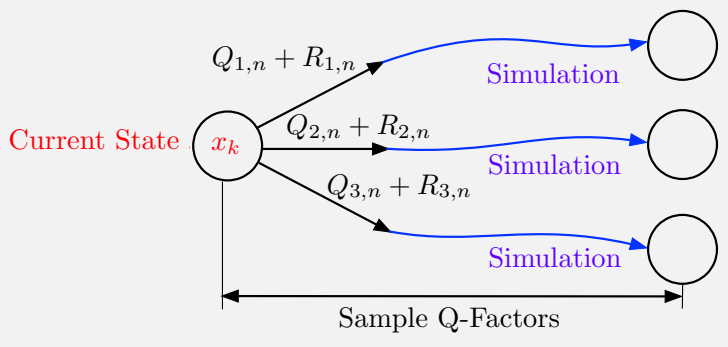

Figure 2.7.3 Illustration of one-step lookahead MCTS at a state x_k. The Q-factor sampled next corresponds to the control i with minimum sum of exploitation index (here taken to be the running average $Q_{i,n}$) and exploration index ($R_{i,n}$, possibly given by the UCB rule).

Thus $Q_{i,n}$ is the *empirical mean* of the Q-factor of control i (total sample value divided by total number of samples), assuming that i has been sampled at least once.

After n samples have been collected, with each control sampled at least once, we may declare the control i that minimizes $Q_{i,n}$ as the "best" one, i.e., the one that truly minimizes the Q-factor $Q_k(x_k, i)$. However, there is a positive probability that there is an error: the selected control may not minimize the true Q-factor. In adaptive sampling, roughly speaking, we want to design the sample selection strategy and the criterion to stop the sampling, in a way that keeps the probability of error small (by allocating some sampling effort to all controls), and the number of samples limited (by not wasting samples on controls i that appear inferior based on their empirical mean $Q_{i,n}$).

Intuitively, a good sampling policy will balance at time n the desires of exploitation and exploration (i.e., sampling controls that seem most promising, in the sense that they have a small empirical mean $Q_{i,n}$, versus assessing the potential of inadequately explored controls, those i that have been sampled a small number of times). Thus it makes sense to sample next the control i that minimizes the sum

$$T_{i,n} + R_{i,n}$$

of two indexes: an *exploitation index* $T_{i,n}$ and an *exploration index* $R_{i,n}$. Usually the exploitation index is chosen to be the empirical mean $Q_{i,n}$; see Fig. 2.7.3. The exploration index is based on a confidence interval formula and depends on the sample count

$$s_i = \sum_{\ell=1}^{n} \delta(i_\ell = i)$$

of control i. A frequently suggested choice is the UCB rule (upper confidence

bound), which sets

$$R_{i,n} = -c\sqrt{\frac{\log n}{s_i}},$$

where c is a positive constant that is selected empirically (some analysis suggests values near $c = \sqrt{2}$, assuming that $Q_{i,n}$ is normalized to take values in the range $[-1, 0]$). The UCB rule, first proposed in the paper by Auer, Cesa-Bianchi, and Fischer [ACF02], has been extensively discussed in the literature both for one-step and for multistep lookahead [where it is called UCT (UCB applied to trees; see Kocsis and Szepesvari [KoS06])].†

Its justification is based on probabilistic analyses that relate to the multiarmed bandit problem, and is beyond our scope. Alternatives to the UCB formula have been suggested, and in fact in the AlphaZero program, the exploitation term has a different form than the one above, and depends on the depth of lookahead (see Silver et al. [SHS17]).

Sampling policies for MCTS with multistep lookahead are based on similar sampling ideas to the case of one-step lookahead. A simulated trajectory is run from a node i of the lookahead tree that minimizes the sum $T_{i,n} + R_{i,n}$ of an exploitation index and an exploration index. There are several schemes of this type, but the details are beyond our scope and are often problem-dependent (see the end-of-chapter references).

A major success has been the use of MCTS in two-player game contexts, such as the AlphaGo program (Silver et al. [SHM16]), which performs better than the best humans in the game of Go. This program integrates several of the techniques discussed in this book, including MCTS and rollout using a base policy that is trained off-line using a deep neural network. The AlphaZero program, which has performed spectacularly well against humans and other programs in the games of Go and chess (Silver et al. [SHS17]), bears some similarity with AlphaGo, and critically relies on MCTS, but does not use rollout in its on-line playing mode (it relies primarily on very long lookahead).

2.7.6 Randomized Policy Improvement by Monte Carlo Tree Search

We have described rollout and MCTS as schemes for policy improvement: start with a base policy, and compute an improved policy based on the results of one-step lookahead or multistep lookahead followed by simulation with the base policy. We have implicitly assumed that both the base policy and the rollout policy are deterministic in the sense that they map each state x_k into a unique control $\tilde{\mu}_k(x_k)$ [cf. Eq. (2.73)]. In some (even

† The paper [ACF02] refers to the rule given here as UCB1 and credits its motivation to the paper by Agrawal [Agr95]. The book by Lattimore and Szepesvari [LaS20] provides an extensive discussion of the UCB rule and its generalizations.

nonstochastic) contexts, success has been achieved with *randomized policies*, which map a state x_k to a probability distribution over the set of controls $U_k(x_k)$, rather than mapping onto a single control. In particular, the AlphaGo and AlphaZero programs use MCTS to generate and use for training purposes randomized policies, which specify at each board position the probabilities with which the various moves are selected.

A randomized policy can be used as a base policy in a rollout context in exactly the same way as a deterministic policy: for a given state x_k, we just generate sample trajectories and associated sample Q-factors, using probabilistically selected controls, starting from each leaf-state of the lookahead tree that is rooted at x_k. We then average the corresponding Q-factor samples. The rollout/improved policy, as described here, is a deterministic policy, i.e., it applies at x_k the control $\tilde{\mu}_k(x_k)$ that is "best" according to the results of the rollout [cf. Eq. (2.73)]. Still, however, if we wish to generate an improved policy that is randomized, we can simply change the probabilities of different controls in the direction of the deterministic rollout policy. This can be done by increasing by some amount the probability of the "best" control $\tilde{\mu}_k(x_k)$ from its base policy level, while proportionally decreasing the probabilities of the other controls.

The use of MCTS provides a related method to "improve" a randomized policy. In the process of the adaptive simulation that is used in MCTS, we generate *frequency counts* of the different controls in $U_k(x_k)$, i.e., the proportion of rollout trajectories associated with each $u_k \in U_k(x_k)$. We can then obtain the rollout randomized policy by moving the probabilities of the base policy in the direction suggested by the frequency counts, i.e., increase the probability of high-count controls and reduce the probability of the others. This type of policy improvement is reminiscent of gradient-type methods, and has been successful in some contexts; see the end-of-chapter references for such policy improvement implementations in AlphaGo, AlphaZero, and other applications.

2.8 ROLLOUT FOR INFINITE-SPACES PROBLEMS - OPTIMIZATION HEURISTICS

We have considered so far finite control space applications of rollout, so there is a finite number of relevant Q-factors at each state x_k, which are evaluated by simulation and are exhaustively compared. When the control constraint set is infinite, to implement this approach the constraint set must be replaced by a finite set, obtained by some form of discretization or random sampling, which can be inconvenient and ineffective. In this section we will discuss an alternative approach to deal with an infinite number of controls and Q-factors at x_k. The idea is to *use a base heuristic that involves a continuous optimization*, and to rely on a linear or nonlinear programming method to solve the corresponding lookahead optimization problem.

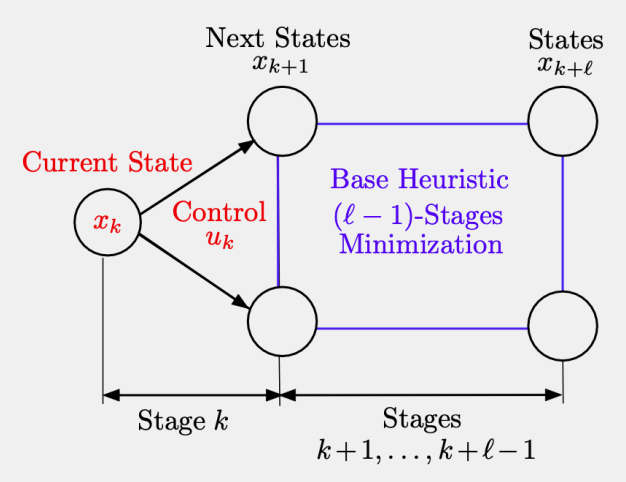

Figure 2.8.1 Schematic illustration of rollout for a deterministic problem with infinite control spaces. The base heuristic is to solve an $(\ell-1)$-stage deterministic optimal control problem, which together with the kth stage minimization over $u_k \in U_k(x_k)$, seamlessly forms an ℓ-stage continuous spaces optimal control/nonlinear programming problem that starts at state x_k.

2.8.1 Rollout for Infinite-Spaces Deterministic Problems

To develop the basic idea of how to deal with infinite control spaces, we first consider deterministic problems, involving a system $x_{k+1} = f_k(x_k, u_k)$, and a cost per stage $g_k(x_k, u_k)$. The rollout minimization is

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k), \tag{2.77}$$

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

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big), \tag{2.78}$$

with $H_{k+1}(x_{k+1})$ being the cost of the base heuristic starting from state x_{k+1} [cf. Eq. (2.10)]. Suppose that we have a differentiable closed-form expression for H_{k+1}, and the functions g_k and f_k are known and are differentiable with respect to u_k. Then the Q-factor $\tilde{Q}_k(x_k, u_k)$ of Eq. (2.78) is also differentiable with respect to u_k, and its minimization (2.77) may be addressed with one of the many gradient-based methods that are available for differentiable unconstrained and constrained optimization.

The preceding approach requires that the heuristic cost $H_{k+1}(x_{k+1})$ can be differentiated, so it should either be available in closed form, which is quite restrictive, or that it can be differentiated numerically, which may be inconvenient and/or unreliable. These difficulties can be circumvented by *using a base heuristic that is itself based on multistep optimization*. In

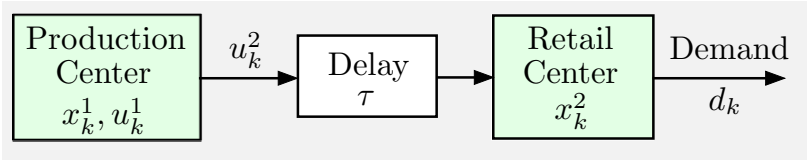

Figure 2.8.2. Illustration of a simple supply chain system for Example 2.8.1.

particular, suppose that $H_{k+1}(x_{k+1})$ is the optimal cost of some $(\ell-1)$-stage deterministic optimal control problem that is related to the original problem. Then the rollout algorithm (2.77)-(2.78) can be implemented by solving the *ℓ-stage deterministic optimal control problem, which seamlessly concatenates the first stage minimization over u_k [cf. Eq. (2.77)], with the $(\ell-1)$-stage minimization of the base heuristic*; see Fig. 2.8.1. This ℓ-stage problem may be solvable on-line by standard continuous spaces nonlinear programming or optimal control methods.† A major paradigm of methods of this type is model predictive control, which we have discussed in Chapter 1 (cf. Section 1.6.7). In the present section we will discuss a few other possibilities. The following is a simple example of an important class of inventory storage and supply chain management processes.

Example 2.8.1 (Supply Chain Management)

Let us consider a supply chain system, where a certain item is produced at a production center and fulfilled at a retail center. Stock of the item is shipped from the production center to the retail center, where it arrives with a delay of $\tau \geq 1$ time units, and is used to fulfill a known stream of demands d_k over an N-stage horizon; see Fig. 2.8.2. We denote:

x_k^1: The stock at hand at the production center at time k.

x_k^2: The stock at hand at the retail center at time k, and used to fulfill demand (both positive and negative x_k^2 are allowed; a negative value indicates that there is backordered demand).

u_k^1: The amount produced at time k.

u_k^2: The amount shipped at time k (and arriving at the retail center τ time units later).

The state at time k is the stock available at the production and retail centers, x_k^1, x_k^2, plus the stock amounts that are in transit and have not yet arrived at the retail center $u_{k-\tau-1}^2, \ldots, u_{k-1}^2$. The control $u_k = (u_k^1, u_k^2)$ is

† Note, however, that for this to be possible, it is necessary to have a mathematical model of the system; a simulator is not sufficient. Another difficulty occurs when the control space is the union of a discrete set and a continuous set. Then it may be necessary to use some type of mixed integer programming technique to solve the ℓ-stage problem. Alternatively, it may be possible to handle the discrete part by brute force enumeration, followed by continuous optimization.

chosen from some constraint set that may depend on the current state, and is subject to production capacity and transport availability constraints. The system equation is

$$x^1_{k+1} = x^1_k + u^1_k - u^2_k, \qquad x^2_{k+1} = x^2_k + u^2_{k-\tau} - d_k,$$

and involves the delayed control component $u^2_{k-\tau}$. Thus the exact DP algorithm involves state augmentation as introduced in Section 1.6.3, and may thus be much more complicated than in the case where there are no delays.†

The cost at time k consists of three components: a production cost that depends on x^1_k and u^1_k, a transportation cost that depends on u^2_k, and a fulfillment cost that depends on x^2_k [which includes positive costs for both excess inventory (i.e., $x^2_k > d_k$) and for backordered demand (i.e., $x^2_k < d_k$)]. The precise forms of these cost components are immaterial for the purposes of this example.

Here the control vector u_k is often continuous (or a mixture of discrete and continuous components), so it may be essential for the purposes of rollout to use the continuous optimization framework of this section. In particular, at the current stage k, we know the current state, which includes x^1_k, x^2_k, and the amounts of stock in transit together with their scheduled arrival times at the retail center. We then apply some heuristic optimization to determine the stream of future production and shipment levels over ℓ steps, and use the first component of this stream as the control applied by rollout. As an example we may use as base policy one that brings the retail inventory to some target value ℓ stages ahead, and possibly keep it at that value for a portion of the remaining periods. This is a nonlinear programming or mixed integer programming problem that may be solvable with available software far more efficiently than by a discretized form of DP.

A major benefit of rollout in the supply chain context is that it can readily incorporate on-line replanning. This is necessary when unexpected demand changes, production or transport equipment failures occur, or updated forecasts become available.

The following example deals with a common class of problems of resource allocation over time.

Example 2.8.2 (Multistage Linear and Mixed Integer Programming)

Let us consider a deterministic optimal control problem with linear system equation

$$x_{k+1} = A_k x_k + B_k u_k + d_k, \qquad k = 0, \ldots, N-1,$$

† Despite the fact that with large delays, the size of the augmented state space can become very large (cf. Section 1.6.3), the implementation of rollout schemes is not affected much by this increase in size. For this reason, rollout can be very well suited for problems involving delayed effects of past states and controls.

where A_k and B_k are known matrices of appropriate dimension, d_k is a known vector, and x_k and u_k are column vectors. The cost function is linear of the form

$$c_N{}'x_N + \sum_{k=0}^{N-1}(c_k{}'x_k + d_k{}'u_k),$$

where c_k and d_k are known column vectors of appropriate dimension, and a prime denotes transpose. The terminal state and state-control pairs (x_k, u_k) are constrained by

$$x_N \in T, \qquad (x_k, u_k) \in P_k, \qquad k = 0, \ldots, N-1,$$

where T and P_k, $k = 0, \ldots, N-1$, are given sets, which are specified by linear and possibly integer constraints.

As an example, consider a multi-item production system, where the state is $x_k = (x_k^1, \ldots, x_k^n)$ and x_k^i represents stock of item i available at the start of period k. The state evolves according to the system equation

$$x_{k+1}^i = \sum_{j=1}^{n} a_k^{ij} u_k^{ij} - d_k^i, \qquad i = 1, \ldots, n,$$

where u_k^{ij} is the amount of product i that is used during time k for the manufacture of product j, a_k^{ij} are known scalars that are related to the underlying production process, and d_k^i is a deterministic demand of product i that is fulfilled at time k. One constraint here is that

$$\sum_{j=1}^{n} u_k^{ij} \leq x_k^i, \qquad i = 1, \ldots, n,$$

and there are additional linear and integer constraints on (x_k, u_k), which are collected in a general constraint of the form $(x_k, u_k) \in P_k$ (e.g., nonnegativity, production capacity, storage constraints, etc). Note that the problem may be further complicated by production delays, as in the preceding supply chain Example 2.8.1. Moreover, while in this section we focus on deterministic problems, we may envision a stochastic version of the problem where the demands d_k^i are random with given probability distributions, which are subject to revisions based on randomly received forecasts.

The problem may be solved using a linear or mixed integer programming algorithm, but this may be very time-consuming when N is large. Moreover, the problem will need to be resolved on-line if some of the problem data changes and replanning is necessary. A suboptimal alternative is to use truncated rollout with an ℓ-stage mixed integer optimization, and a polyhedral terminal cost function $\tilde{J}_{k+\ell}$ to provide a terminal cost optimization. A simple possibility is no terminal cost [$\tilde{J}_{k+\ell}(x_{k+\ell}) \equiv 0$], and another possibility is a polyhedral lower bound approximation that can be based on relaxing the integer constraints after stage $k + \ell$, or some kind of training approach that uses data.

We will next discuss how rollout can accommodate stochastic disturbances by using deterministic optimization ideas based on certainty equivalence (cf. Section 2.7.4) and the methodology of stochastic programming.

2.8.2 Rollout Based on Stochastic Programming

We have focused so far in this section on rollout that relies on deterministic continuous optimization. There is an important class of methods, known as *stochastic programming*, which can be used for stochastic optimal control, but bears a close connection to continuous spaces deterministic optimization. We will first describe this connection for two-stage problems, then discuss extensions to many-stages problems, and finally show how rollout can be brought to bear for their approximate solution.

Example 2.8.3 (Two-Stage Stochastic Programming)

Consider a stochastic problem of optimal decision making over two stages: In the first stage we will choose a finite-dimensional vector u_0 from a subset U_0 with cost $g_0(u_0)$. Then an uncertain event represented by a random variable w_0 will occur, whereby w_0 will take one of the values $w^1, \ldots, w^m$ with corresponding probabilities $p^1, \ldots, p^m$. Once w_0 occurs, we will know its value w^i, and we must then choose at the second stage a vector u_1^i from a subset $U_1(u_0, w^i)$ at a cost $g_1(u_1^i, w^i)$. The objective is to minimize the expected cost

$$g_0(u_0) + \sum_{i=1}^{m} p^i g_1(u_1^i, w^i),$$

subject to

$$u_0 \in U_0, \qquad u_1^i \in U_1(u_0, w^i), \quad i = 1, \ldots, m.$$

We can view this problem as a two-stage DP problem, where $x_1 = w_0$ is the system equation, the disturbance w_0 can take the values $w^1, \ldots, w^m$ with probabilities $p^1, \ldots, p^m$, the cost of the first stage is $g_0(u_0)$, the cost of the second stage is $g_1(x_1, u_1)$, and the terminal cost is 0. The intuitive meaning is that since at time 0 we don't know yet which of the m values w^i of w_0 will occur, we must calculate (in addition to u_0) a separate second stage decision u_1^i for each i, which will be used after we know that the value of w_0 is w^i.

However, if u_0 and u_1 take values in a continuous space such as the Euclidean spaces $\Re^{d_0}$ and $\Re^{d_1}$, respectively, we can also equivalently view the problem as a nonlinear programming problem of dimension $(d_0 + md_1)$ (the optimization variables are u_0 and u_1^i, $i = 1, \ldots, m$).

For a generalization of the preceding example, consider the stochastic DP problem of Section 1.3 for the case where there are only two stages, and the disturbances w_0 and w_1 can independently take one of the m values $w^1, \ldots, w^m$ with corresponding probabilities $p_0^1, \ldots, p_0^m$ and $p_1^1, \ldots, p_1^m$, respectively. The optimal cost function $J_0(x_0)$ is given by the two-stage DP algorithm

$$J_0(x_0) = \min_{u_0 \in U_0(x_0)} \left[\sum_{i=1}^{m} p_0^i \Big\{ g_0(x_0, u_0, w^i) \right.$$

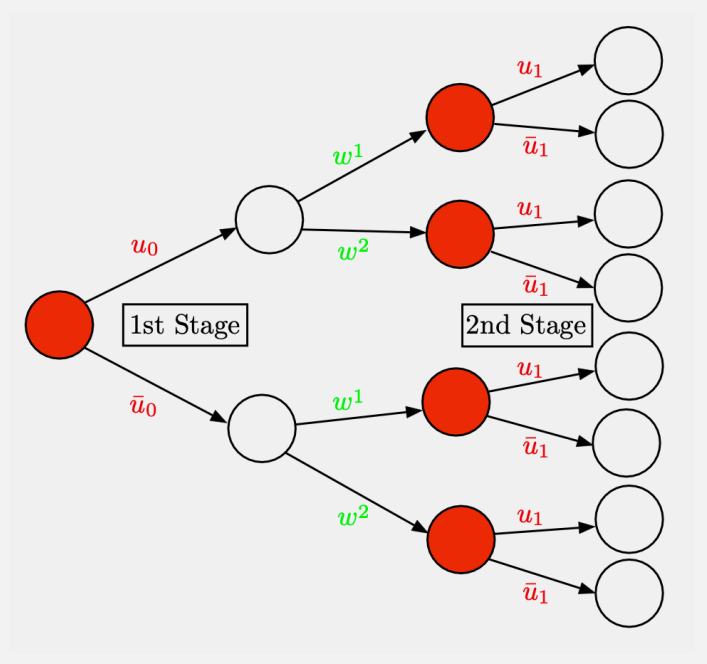

Figure 2.8.3. Illustration of the DP problem associated with two-stage stochastic programming; cf. Example 2.8.3. The figure depicts the case where each variable u_0, w_0, and u_1 can take only two values. A similar conversion to a DP problem is possible for a multistage stochastic programming problem, involving multiple choices of decisions, each followed by an uncertain event whose outcome is perfectly observed by the decision maker.

$$+ \min_{u_1^i \in U_1(f_0(x_0,u_0,w^i))} \left[\sum_{j=1}^{m} p_1^j \Big\{ g_1\big(f_0(x_0,u_0,w^i),u_1^i,w^j\big) + g_2\big(f_1\big(f_0(x_0,u_0,w^i),u_1^i,w^j\big)\big) \Big\} \right] \Bigg\} \Bigg].$$

By bringing the inner minimization outside the inner brackets, we see that this DP algorithm is equivalent to solving the nonlinear programming problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} p_0^i \Bigg\{ g_0(x_0,u_0,w^i) + \sum_{j=1}^{m} p_1^j \Big\{ g_1\big(f_0(x_0,u_0,w^i),u_1^i,w^j\big) \\ & \qquad + g_2\big(f_1\big(f_0(x_0,u_0,w^i),u_1^i,w^j\big)\big) \Big\} \Bigg\} \\ \text{subject to} \quad & u_0 \in U_0(x_0), \qquad u_1^i \in U_1\big(f_0(x_0,u_0,w^i)\big), \quad i = 1,\ldots,m. \end{aligned} \tag{2.79}$$

If the controls u_0 and u_1^i are elements of $\Re^d$, this problem involves $d(1+m)$ scalar variables. An example is the multi-item production problem described in Example 2.8.2 in the case where the demands w_k^i and/or the production coefficients a_k^{ij} are stochastic.

We can also consider an N-stage stochastic optimal control problem. A similar reformulation as a nonlinear programming problem is possible. It converts the N-stage stochastic problem into a deterministic optimization problem of dimension that grows exponentially with the number of stages N. In particular, for an N-stage problem, the number of control variables expands by a factor m with each additional stage. The total number of variables is bounded by

$$d\big(1+m+m^2+\cdots+m^{N-1}\big),$$

where m is the maximum number of values that a disturbance can take at each stage and d is the dimension of the control vector.

2.8.3 Stochastic Rollout with Certainty Equivalence

The dimension of the preceding nonlinear programming formulation of the multistage stochastic optimal control problem with continuous control spaces can be very large. This motivates a variant of a rollout algorithm that relies on a stochastic optimization for the current stage, and a deterministic optimization that relies on (assumed) certainty equivalence for the remaining stages, where the base policy is used. In this way, the dimension of the nonlinear programming problem to be solved by rollout is drastically reduced.

This rollout algorithm operates as follows: Given a state x_k and control $u_k \in U_k(x_k)$, we consider the next states x_{k+1}^i that correspond to the m possible values w_k^i, $i=1,\ldots,m$, which occur with the known probabilities p_k^i, $i=1,\ldots,m$. We then consider the approximate Q-factors

$$\tilde{Q}_k(x_k,u_k)=\sum_{i=1}^{m} p_k^i\big(g_k(x_k,u_k,w_k^i)+\tilde{H}_{k+1}(x_{k+1}^i)\big), \tag{2.80}$$

where $\tilde{H}_{k+1}(x_{k+1}^i)$ is the cost of a base policy, which starting at stage $k+1$ from

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

optimizes the cost-to-go starting from x_{k+1}^i, while assuming that the future disturbances $w_{k+1},\ldots,w_{N-1}$, will take some nominal (nonrandom) values $\bar{w}_{k+1},\ldots,\bar{w}_{N-1}$. The rollout control $\tilde{\mu}_k(x_k)$ computed by this algorithm is

$$\tilde{\mu}_k(x_k)\in \arg\min_{u_k\in U_k(x_k)} \tilde{Q}_k(x_k,u_k). \tag{2.81}$$

Note that this rollout algorithm does not have the cost improvement property, because it involves an approximation: the cost $\tilde{H}_{k+1}(x_{k+1}^i)$ used in Eq. (2.80) is an approximation to the cost of a policy. It is the cost of a policy applied to the certainty equivalent version of the original stochastic problem.

The key fact now is that the problem (2.81) can be viewed as a seamless $(N-k)$-stage deterministic optimization, which involves the control u_0, and for each value w_k^i of the disturbance w_k, the sequence of controls $(u_{k+1}^i, \ldots, u_{N-1}^i)$. If the controls are elements of $\Re^d$, this deterministic optimization involves a total of

$$d\big(1 + (N-k-1)m\big) \tag{2.82}$$

scalar variables. Currently available deterministic optimization software can deal with quite large numbers of variables, particularly in the context of linear programming, so by using rollout in combination with certainty equivalence, very large problems with continuous state and control variables may be addressed.

Another possibility is to use multistep lookahead that aims to represent better the stochastic character of the uncertainty. Here at state x_k we solve an $(N-k)$-stage optimal control problem, where the uncertainty is fully taken into account in the first ℓ stages, similar to stochastic programming, and in the remaining $N-k-\ell$ stages, the uncertainty is dealt with by certainty equivalence, by fixing the disturbances $w_{k+\ell}, \ldots, w_{N-1}$ at some nominal values (we assume here for simplicity that $\ell < N-k$). If the controls are elements of $\Re^d$, and the number of values that the disturbances $w_0, \ldots, w_{N-1}$ can take is m, the total number of control variables of this problem is

$$d\big(1 + m + \cdots + m^{\ell-1} + (N-k-\ell)m^\ell\big),$$

[this is the ℓ-step lookahead generalization of the formula (2.82)]. Once the optimal policy $\{\tilde{u}_k, \tilde{\mu}_{k+1}, \tilde{\mu}_{k+2}, \ldots\}$ for this problem is obtained, the first control component $\tilde{u}_k$ is applied at x_k and the remaining components $\{\tilde{\mu}_{k+1}, \tilde{\mu}_{k+2}, \ldots\}$ are discarded. Note also that this multistep lookahead approach may be combined with the ideas of multiagent rollout, which will be discussed in the next section.

2.9 MULTIAGENT ROLLOUT

We will now consider a special structure of the control space, whereby the control u_k consists of m components, $u_k = (u_k^1, \ldots, u_k^m)$, with a separable control constraint structure $u_k^\ell \in U_k^\ell(x_k)$, $\ell = 1, \ldots, m$. The control constraint set is the Cartesian product

$$U_k(x_k) = U_k^1(x_k) \times \cdots \times U_k^m(x_k). \tag{2.83}$$

Conceptually, each component u_k^ℓ, $\ell = 1, \ldots, m$, is chosen at stage k by a separate "agent" (a decision making entity), and for the sake of the following discussion, we assume that each set $U_k^\ell(x_k)$ is finite. We discussed this type of problem briefly in Section 1.6.5, and we will discuss it in this section in greater detail.

The one-step lookahead minimization

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1,\pi}\big(f_k(x_k, u_k, w_k)\big) \Big\}, \quad (2.84)$$

where π is a base policy, involves as many as n^m Q-factors, where n is the maximum number of elements of the sets $U_k^\ell(x_k)$ [so that n^m is an upper bound to the number of controls in $U_k(x_k)$, in view of the Cartesian product structure (2.83)]. As a result, the standard rollout algorithm requires an exponential [order $O(n^m)$] number of base policy cost computations per stage, which can be overwhelming even for moderate values of m.

This motivates an alternative and more efficient rollout algorithm, called *multiagent rollout* also referred to as *agent-by-agent rollout*, that still achieves the cost improvement property

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \forall\ x_k, k, \quad (2.85)$$

where $J_{k,\tilde{\pi}}(x_k)$, $k = 0, \ldots, N$, is the cost-to-go of the rollout policy $\tilde{\pi}$ starting from state x_k. Indeed we will exploit the multiagent structure to construct an algorithm that maintains the cost improvement property at much smaller computational cost, namely requiring order $O(nm)$ base policy cost computations per stage.

A key idea here is that the computational requirements of the rollout one-step minimization (2.84) are proportional to the size of the control space and are independent of the size of the state space. We consequently reformulate the problem so that control space complexity is traded off with state space complexity, as discussed in Section 1.6.5. This is done by "unfolding" the control u_k into its m components $u_k^1, u_k^2, \ldots, u_k^m$. At the same time, between x_k and the next state $x_{k+1} = f_k(x_k, u_k, w_k)$, we introduce artificial intermediate "states" and corresponding transitions; see Fig. 2.9.1, given in Section 1.6.5 and repeated here for convenience.

It can be seen that this reformulated problem is equivalent to the original, since any control choice that is possible in one problem is also possible in the other problem, while the cost structure of the two problems is the same: each policy of the reformulated problem corresponds to a policy of the original problem, with the same cost function, and reversely.†

† A fine point here is that policies of the original problem involve functions of x_k, while policies of the reformulated problem involve functions of the choices of the preceding agents, as well as x_k. However, by successive substitution of

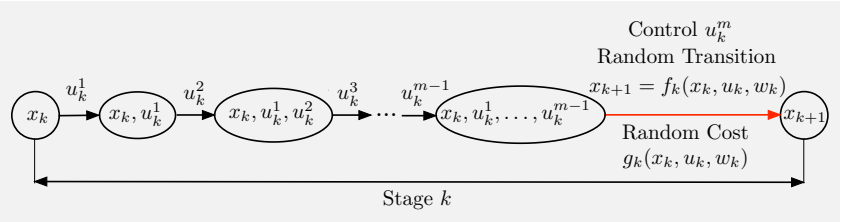

Figure 2.9.1 Equivalent formulation of the N-stage stochastic optimal control problem for the case where the control u_k consists of m components $u_k^1, u_k^2, \ldots, u_k^m$:

$$u_k = (u_k^1, \ldots, u_k^m) \in U_k^1(x_k) \times \cdots \times U_k^m(x_k);$$

cf. Section 1.6.5. The figure depicts the kth stage transitions. Starting from state x_k, we generate the intermediate states

$$(x_k, u_k^1), (x_k, u_k^1, u_k^2), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1}),$$

using the respective controls $u_k^1, \ldots, u_k^{m-1}$. The final control u_k^m leads from $(x_k, u_k^1, \ldots, u_k^{m-1})$ to $x_{k+1} = f_k(x_k, u_k, w_k)$, and a stage cost $g_k(x_k, u_k, w_k)$ is incurred. All of the preceding transitions, which involve the controls $u_k^1, \ldots, u_k^{m-1}$, incur zero cost.

Multiagent Rollout

Consider now the standard rollout algorithm applied to the reformulated problem of Fig. 2.9.1, with a given base policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, which is also a policy of the original problem [so that $\mu_k = (\mu_k^1, \ldots, \mu_k^m)$, with each μ_k^ℓ, $\ell = 1, \ldots, m$, being a function of just x_k]. The algorithm involves a minimization over only one control component at the states x_k and at the intermediate states

$$(x_k, u_k^1), (x_k, u_k^1, u_k^2), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1}).$$

In particular, *for each stage k, the algorithm requires a sequence of m minimizations, once over each of the agent controls $u_k^1, \ldots, u_k^m$, with the past controls determined by the rollout policy, and the future controls determined by the base policy.* Assuming a maximum of n elements in the constraint sets $U_k^\ell(x_k)$, the computation required at each stage k is of order $O(n)$ for each of the "states"

$$x_k, (x_k, u_k^1), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1}),$$

---

the control functions of the preceding agents, we can view control functions of each agent as depending exclusively on x_k. It follows that the multi-transition structure of the reformulated problem cannot be exploited to reduce the cost function beyond what can be achieved with a single-transition structure.

for a total of order $O(nm)$ computation.

To elaborate, at $(x_k, u_k^1, \ldots, u_k^{\ell-1})$ with $\ell \leq m$, and for each of the controls $u_k^\ell \in U_k^\ell(x_k)$, we generate by simulation a number of system trajectories up to stage N, with all future controls determined by the base policy. We average the costs of these trajectories, thereby obtaining the Q-factors corresponding to $(x_k, u_k^1, \ldots, u_k^{\ell-1}, u_k^\ell)$, for all values $u_k^\ell \in U_k^\ell(x_k)$ (with the preceding controls $u_k^1, \ldots, u_k^{\ell-1}$ held at the values computed earlier, and the future controls $u_k^{\ell+1}, \ldots, u_k^m, u_{k+1}, \ldots, u_{N-1}$ determined by the base policy). We then select the control $u_k^\ell \in U_k^\ell(x_k)$ that corresponds to the minimal Q-factor.

Prerequisite assumptions for the preceding algorithm to work in an on-line multiagent setting are:

(a) All agents have access to the current state x_k as well as the base policy (including the control functions μ_n^ℓ, $\ell = 1, \ldots, m$, $n = 0, \ldots, N-1$ of all agents).

(b) There is an order in which agents compute and apply their local controls.

(c) The agents share their information, so agent ℓ knows the local controls $u_k^1, \ldots, u_k^{\ell-1}$ computed by the predecessor agents $1, \ldots, \ell-1$ in the given order.

Note that the rollout policy obtained from the reformulated problem may be different from the rollout policy obtained from the original problem. However, the former rollout algorithm is far more efficient than the latter in terms of required computation, while still maintaining the cost improvement property (2.85).

The following spiders-and-flies example illustrates how multiagent rollout may exhibit intelligence and agent coordination that is totally lacking from the base policy. This behavior has been supported by computational experiments and analysis with larger (two-dimensional) spiders-and-flies problems.

Example 2.9.1 (Spiders and Flies)

We have two spiders and two flies moving along integer locations on a straight line. For simplicity we assume that the flies' positions are fixed at some integer locations, although the problem is qualitatively similar when the flies move randomly. The spiders have the option of moving either left or right by one unit; see Fig. 2.9.2. The objective is to minimize the time to capture both flies. The problem has essentially a finite horizon since the spiders can force the capture of the flies within a known number of steps.

The salient feature of the optimal policy here is to move the two spiders towards different flies. The minimal time to capture is the maximum of the initial distances of the two spider-fly pairs of the optimal policy.

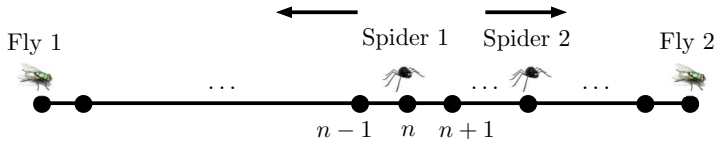

Figure 2.9.2 Illustration of the two-spiders and two-flies problem. The spiders move along integer points of a line. The two flies stay still at some integer locations. The character of the optimal policy is to move the two spiders towards two different flies.

Multiagent rollout with the given base policy starts with spider 1 at location n, and calculates the two Q-factors of moving to locations $n-1$ and $n+1$, assuming that the remaining moves of the two spiders will be made using the go-towards-the-nearest-fly base policy. The Q-factor of going to $n-1$ is smallest because it saves in unnecessary moves of spider 1 towards fly 2, so spider 1 will move towards fly 1. The trajectory generated by multiagent rollout is to move spiders 1 and 2 towards flies 1 and 2, respectively, then spider 2 first captures fly 2, and then spider 1 captures fly 1.

Let us apply multiagent rollout with the base policy that directs each spider to move one unit towards the closest fly position (a tie is broken by moving towards the right-side fly). The base policy is poor because it may unnecessarily move both spiders in the same direction, when in fact only one is needed to capture the fly. This limitation is due to the lack of coordination between the spiders: each acts selfishly, ignoring the presence of the other. We will see that rollout restores a significant degree of coordination between the spiders through an optimization that takes into account the long-term consequences of the spider moves.

According to the multiagent rollout mechanism, the spiders choose their moves one-at-a-time, optimizing over the two Q-factors corresponding to the right and left moves, while assuming that future moves will be chosen according to the base policy. Let us consider a stage, where the two flies are alive, while both spiders are closest to fly 2, as in Fig. 2.9.2. Then the rollout algorithm will start with spider 1 and calculate two Q-factors corresponding to the right and left moves, while using the base heuristic to obtain the next move of spider 2, and the remaining moves of the two spiders. Depending on the values of the two Q-factors, spider 1 will move to the right or to the left, and it can be seen that it will choose to *move away from spider 2* even if doing so increases its distance to its closest fly *contrary to what the base heuristic will do*. Then spider 2 will act similarly and the process will continue. Intuitively, at the state of Fig. 2.9.2, spider 1 moves away from spider 2 and fly 2, because it recognizes that spider 2 will capture earlier fly 2, so it might as well move towards the other fly.

Thus *the multiagent rollout algorithm induces implicit move coordination*, i.e., each spider moves in a way that takes into account future moves of the other spider. In fact it can be verified that the algorithm will produce an optimal sequence of moves starting from any initial spider positions. It can also be seen that ordinary rollout (both flies move at once) will also produce an optimal move sequence.

The example illustrates how a poor base heuristic can produce an ex-

cellent rollout solution, something that can be observed frequently in many other problems. Intuitively, the key fact is that rollout is "farsighted" in the sense that it can benefit from control calculations that reach far into future stages.

A two-dimensional generalization of the example is also interesting. Here the flies are at two corners of a square in the plane. It can be shown that the two spiders, starting from the same position within the square, will separate under the rollout policy, with each moving towards a different spider, while under the base policy, they will move in unison along the shortest path to the closest surviving fly. Again this will happen for both standard and multiagent rollout.

Let us consider another example of a discrete optimization problem that can be solved efficiently with multiagent rollout.

Example 2.9.2 (Multi-Vehicle Routing)

Consider a multi-vehicle routing problem, whereby m vehicles move along the arcs of a given graph, aiming to perform tasks located at the nodes of the graph; see Fig. 2.9.3. When a vehicle reaches a task, it performs it, and can move on to perform another task. We wish to perform the taks in a minimum number of individual vehicle moves.

For a large number of vehicles and a complicated graph, this is a non-trivial combinatorial problem. The problem can be formulated as a discrete deterministic optimization problem, and addressed by approximate DP methods. The state at a given stage is the m-tuple of current positions of the vehicles together with the list of pending tasks, but the number of these states can be enormous (it increases exponentially with the number of nodes and the number of vehicles). Moreover the number of joint move choices by the vehicles also increases exponentially with the number of vehicles.

We are thus motivated to use a multiagent rollout approach. We define a base heuristic as follows: at a given stage and state (vehicle positions and pending tasks), it finds the closest pending task (in terms of number of moves needed to reach it) for each of the vehicles and moves each vehicle one step towards the corresponding closest pending task (this is a legitimate base heuristic: it assigns to each state a vehicle move for every vehicle).†

† There is an alternative version of the base heuristic, which makes selections one-vehicle-at-a-time: at a given stage and state (vehicle positions and pending tasks), it finds the closest pending task (in terms of number of moves needed to reach it) for vehicle 1 and moves this vehicle one step towards this closest pending task. Then it finds the closest pending task for vehicle 2 (the pending status of the tasks, however, may have been affected by the move of vehicle 1) and moves this vehicle one step towards this closest pending task, and continues similarly for vehicles $3, \ldots, n$. There is a subtle difference between the two base heuristics: for example they may make different choices when vehicle 1 reaches a pending task in a single move, thereby changing the status of that task, and affecting the choice of the base heuristic for vehicle 2, etc.

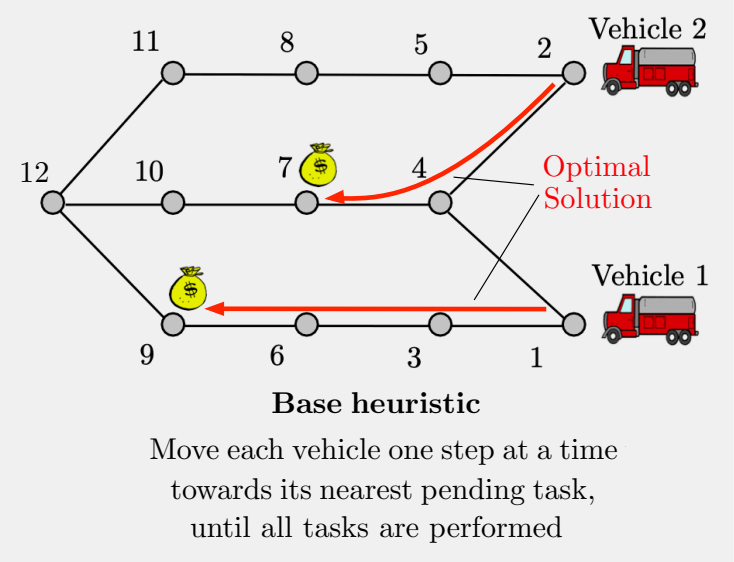

Figure 2.9.3 An instance of the vehicle routing problem of Example 2.9.2, and the multiagent rollout approach. The two vehicles aim to collectively perform the two tasks as fast as possible. Here, we should avoid sending both vehicles to node 4, towards the task at node 7; sending only vehicle 2 towards that task, while sending vehicle 1 towards the task at node 9 is clearly optimal. However, the base heuristic has "limited vision" and does not perceive this. By contrast the standard and the one-vehicle-at-a-time rollout algorithms look beyond the first move and avoid this inefficiency: they examine both moves of vehicle 1 to nodes 3 and 4, and use the base heuristic to explore the corresponding trajectories to the end of the horizon, and discover that vehicle 2 can reach quickly node 7, and that it is best to send vehicle 1 towards node 9.

In particular, the one-vehicle-at-a-time rollout algorithm will operate as follows: given the starting position pair $(1, 2)$ of the vehicles and the current pending tasks at nodes 7 and 9, we first compare the Q-factors of the two possible moves of vehicle 1 (to nodes 3 and 4), assuming that all the remaining moves will be selected by the base heuristic at the beginning of each stage. Thus vehicle 1 will choose to move to node 3. Then with knowledge of the move of vehicle 1 from 1 to 3, we select the move of vehicle 2 by comparing the Q-factors of its two possible moves (to nodes 4 and 5), taking also into account the fact that the remaining moves will be made according to the base heuristic. Thus vehicle 2 will choose to move to node 4.

We then continue at the next state [vehicle positions at (3,4) and pending tasks at nodes 7 and 9], select the base heuristic moves of vehicles 1 and 2 on the path to the closest pending tasks [(9 and 7), respectively], etc. Eventually the rollout finds the optimal solution (move vehicle 1 to node 9 in three moves and move vehicle 2 to node 7 in two moves), which has a total cost of 5. By contrast it can be seen that the base heuristic at the initial state will move both vehicles to node 4 (towards the closest pending task), and generate a trajectory that moves vehicle 1 along the path $1 \to 4 \to 7$ and vehicle 2 along the path $2 \to 4 \to 7 \to 10 \to 12 \to 9$, while incurring a total cost of 7.

In the multiagent rollout algorithm, at a given stage and state, we take up each vehicle in the order $1, \ldots, n$, and we compare the Q-factors of the available moves to that vehicle while assuming that all the remaining moves will be made according to the base heuristic, and taking into account the moves that have been already made and the tasks that have already been performed; see the illustration of Fig. 2.9.3. In contrast to all-vehicles-at-once rollout, the one-vehicle-at-a-time rollout algorithm considers a polynomial (in m) number of moves and corresponding shortest path problems at each stage. In the example of Fig. 2.9.3, the one-vehicle-at-a-time rollout finds the optimal solution, while the base heuristic starting from the initial state does not.

The Cost Improvement Property

Generally, it is unclear how the two rollout policies (standard/all-agents-at-once and agent-by-agent) perform relative to each other in terms of attained cost.† On the other hand, both rollout policies perform no worse than the base policy, since the performance of the base policy is identical for both the reformulated and the original problems. This cost improvement property can also be shown analytically as follows by induction, by modifying the standard rollout cost improvement proof; cf. Section 2.7.

Proposition 2.9.1: (Cost Improvement for Multiagent Rollout) The rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ obtained by multiagent rollout satisfies

$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k, \tag{2.86}$$

where π is the base policy.

† For an example where the standard rollout algorithm works better, consider a single-stage problem, where the objective is to minimize the first stage cost $g_0(u_0^1, \ldots, u_0^m)$. Let $\overline{u}_0 = (\overline{u}_0^1, \ldots, \overline{u}_0^m)$ be the control applied by the base policy, and assume that $\overline{u}_0$ is not optimal. Suppose that starting at $\overline{u}_0$, the cost cannot be improved by varying any single control component. Then the multiagent rollout algorithm stays at the suboptimal $\overline{u}_0$, while the standard rollout algorithm finds an optimal control. Thus, for one-stage problems, the standard rollout algorithm will perform no worse than the multiagent rollout algorithm.

The example just given is best seen within the framework of the classical coordinate descent method for minimizing a function of m components. This method can get stuck at a nonoptimal point in the absence of appropriate conditions on the cost function, such as differentiability and/or convexity. However, within our context of multistage rollout and possibly stochastic disturbances, it appears that the consequences of such a phenomenon may not be serious. In fact, one can construct multi-stage examples where multiagent rollout performs better than the standard rollout.

Proof: We will show the inequality (2.86) by induction, but for simplicity, we will give the proof for the case of just two agents, i.e., $m = 2$. Clearly the inequality 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,

$$\begin{aligned}
J_{k,\tilde{\pi}}(x_k) &= E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big) \\
&\qquad + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big)\Big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big) \\
&\qquad + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \tilde{\mu}_k^2(x_k), w_k\big)\Big)\Big\} \\
&= \min_{u_k^2 \in U_k^2(x_k)} E\Big\{g_k(x_k, \tilde{\mu}_k^1(x_k), u_k^2, w_k) \\
&\qquad + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), u_k^2, w_k\big)\Big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \tilde{\mu}_k^1(x_k), \mu_k^2(x_k), w_k\big) \\
&\qquad + J_{k+1,\pi}\Big(f_k\big(x_k, \tilde{\mu}_k^1(x_k), \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&= \min_{u_k^1 \in U_k^1(x_k)} E\Big\{g_k(x_k, u_k^1, \mu_k^2(x_k), w_k) \\
&\qquad + J_{k+1,\pi}\Big(f_k\big(x_k, u_k^1, \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&\leq E\Big\{g_k\big(x_k, \mu_k^1(x_k), \mu_k^2(x_k), w_k\big) \\
&\qquad + J_{k+1,\pi}\Big(f_k\big(x_k, \mu_k^1(x_k), \mu_k^2(x_k), w_k\big)\Big)\Big\} \\
&= J_{k,\pi}(x_k),
\end{aligned}$$

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 as it pertains to agent 2.

(d) The third equality holds by the definition of the rollout algorithm as it pertains to agent 1.

(e) The fourth equality is the DP equation for the base policy π.

The induction proof of the cost improvement property (2.86) is thus complete for the case $m = 2$. The proof for an arbitrary number of agents m is entirely similar. **Q.E.D.**

Optimizing the Agent Order in Agent-by-Agent Rollout - Multiagent Parallelization

In the multiagent rollout algorithm described so far, the agents optimize the control components sequentially in a fixed order. It is possible to improve performance by trying to optimize at each stage k the order of the agents.

An efficient way to do this is to first optimize over all single agent Q-factors, by solving the m minimization problems that correspond to each of the agents $\ell = 1, \ldots, m$ being first in the multiagent rollout order. If ℓ_1 is the agent that produces the minimal Q-factor, we fix ℓ_1 to be the first agent in the multiagent rollout order. Then we optimize over all single agent Q-factors, by solving the $m-1$ minimization problems that correspond to each of the agents $\ell \neq \ell_1$ being second in the multiagent rollout order. Let ℓ_2 be the agent that produces the minimal Q-factor, fix ℓ_2 to be the second agent in the multiagent rollout order, and continue in this manner. In the end, after

$$m + (m-1) + \cdots + 1 = \frac{m(m+1)}{2} \tag{2.87}$$

minimizations, we obtain an agent order $\ell_1, \ldots, \ell_m$ that produces a potentially much reduced Q-factor value, as well as the corresponding rollout control component selections.

The method just described likely produces substantially better performance, and eliminates the need for guessing a good agent order, but it increases the number of Q-factor calculations needed per stage roughly by a factor $(m+1)/2$. Still this is much better than the all-agents-at-once approach, which requires an exponential number of Q-factor calculations. Moreover, the Q-factor minimizations of the above process can be parallelized, so with m parallel processors, we can perform the number of $m(m+1)/2$ minimizations derived above in just m batches of parallel minimizations, which require about the same time as in the case where the agents are selected for Q-factor minimization in a fixed order. We finally note that our earlier cost improvement proof goes through again by induction, when the order of agent selection is variable at each stage k.

Multiagent Rollout Variants

The agent-by-agent rollout algorithm admits several variants. We describe briefly a few of these variants.

(a) We may use rollout with multistep lookahead, truncated rollout, and terminal cost function approximation, as described earlier. Of course, in such cases the cost improvement property need not hold.

(b) When the control constraint sets $U_k^\ell(x_k)$ are infinite, multiagent rollout still applies, based on the tradeoff between control and state space

complexity, cf. Fig. 2.9.1. In particular, when the sets $U_k^\ell(x_k)$ are intervals of the real line, each agent's lookahead minimization problem can be performed with the aid of one-dimensional search methods.

(c) When the problem is deterministic there are additional possible variants of the multiagent rollout algorithm. In particular, for deterministic problems, we may use a more general base policy, i.e., a heuristic that is not defined by an underlying policy; cf. Section 2.3.1. In this case, if the sequential improvement assumption for the modified problem of Fig. 2.9.1 is not satisfied, then the cost improvement property may not hold. However, cost improvement may be restored by introducing fortification, as discussed in Section 2.3.2.

(d) The multiagent rollout algorithm can be simply modified to apply to infinite horizon problems. In this context, we may also consider policy iteration methods, which can be viewed as repeated rollout. These methods may involve agent-by-agent policy improvement, and value and policy approximations of intermediately generated policies (see the RL book [Ber19a], Section 5.7.3).

(e) The multiagent rollout algorithm can be simply modified to apply to deterministic continuous-time optimal control problems; cf. Section 2.6. The idea is again to simplify the minimization over $u(t)$ in the case where $u(t)$ consists of multiple components $u^1(t), \ldots, u^m(t)$.

(f) We can implement within the agent-by-agent rollout context the use of Q-factor differences. The motivation is similar: deal with the approximation errors that are inherent in the estimated cost of the base policy, $\tilde{J}_{k+1,\pi}\big(f_k(x_k, u_k)\big)$, and may overwhelm the current stage cost term $g_k(x_k, u_k)$. As noted in Section 2.3.7, this may seriously degrade the quality of the rollout policy; see also the discussion of advantage updating and differential training in Chapter 3.

Constrained Multiagent Rollout

Let us consider a special structure of the control space, where the control u_k consists of m components, $u_k = (u_k^1, \ldots, u_k^m)$, each belonging to a corresponding set $U_k^\ell(x_k)$, $\ell = 1, \ldots, m$. Thus the control space at stage k is the Cartesian product

$$U_k(x_k) = U_k^1(x_k) \times \cdots \times U_k^m(x_k).$$

We refer to this as the *multiagent case*, motivated by the special case where each component u_k^ℓ, $\ell = 1, \ldots, m$, is chosen by a separate agent ℓ at stage k.

Similar to the unconstrained case, we can introduce a modified but equivalent problem, involving one-at-a-time agent control selection. In particular, at the generic state x_k, we break down the control u_k into the se-

quence of the m controls $u_k^1, u_k^2, \ldots, u_k^m$, and between x_k and the next state $x_{k+1} = f_k(x_k, u_k)$, we introduce artificial intermediate "states"

$$(x_k, u_k^1), (x_k, u_k^1, u_k^2), \ldots, (x_k, u_k^1, \ldots, u_k^{m-1}),$$

and corresponding transitions. The choice of the last control component u_k^m at "state" $(x_k, u_k^1, \ldots, u_k^{m-1})$ marks the transition at cost $g_k(x_k, u_k)$ to the next state $x_{k+1} = f_k(x_k, u_k)$ according to the system equation. It is evident that this reformulated problem is equivalent to the original, since any control choice that is possible in one problem is also possible in the other problem, with the same cost.

By working with the reformulated problem, we can consider a rollout algorithm requires a sequence of m minimizations per stage, one over each of the control components $u_k^1, \ldots, u_k^m$, with the past controls already determined by the rollout algorithm, and the future controls determined by running the base heuristic. Assuming a maximum of n elements in the control component spaces $U_k^\ell(x_k)$, $\ell = 1, \ldots, m$, the computation required for the m single control component minimizations is of order $O(nm)$ per stage. By contrast the standard rollout minimization (2.44) involves the computation of as many as n^m terms $G\big(T_k(\tilde{y}_k, u_k)\big)$ per stage.

2.9.1 Asynchronous and Autonomous Multiagent Rollout

In this section we consider multiagent rollout algorithms that are distributed and asynchronous in the sense that the agents may compute their rollout controls in parallel rather than in sequence, aiming at computational speedup. An example of such an algorithm is obtained when at a given stage, agent ℓ computes the rollout control $\tilde{u}_k^\ell$ before knowing the rollout controls of some of the agents $1, \ldots, \ell - 1$, and uses the controls $\mu_k^1(x_k), \ldots, \mu_k^{\ell-1}(x_k)$ of the base policy in their place.

This algorithm may work well for some problems, but it does not possess the cost improvement property, and may not work well for other problems. In fact we can construct a simple example involving a single state, two agents, and two controls per agent, where the second agent does not take into account the control applied by the first agent, and as a result the rollout policy performs worse than the base policy for some initial states.

Example 2.9.3 (Cost Deterioration in the Absence of Adequate Agent Coordination)

Consider a problem with two agents ($m = 2$) and a single state. Thus the state does not change and the costs of different stages are decoupled (the problem is essentially static). Each of the two agents has two controls: $u_k^1 \in \{0, 1\}$ and $u_k^2 \in \{0, 1\}$. The cost per stage g_k is equal to 0 if $u_k^1 \neq u_k^2$, is equal to 1 if $u_k^1 = u_k^2 = 0$, and is equal to 2 if $u_k^1 = u_k^2 = 1$. Suppose that the base

policy applies $u_k^1 = u_k^2 = 0$. Then it can be seen that when executing rollout, the first agent applies $u_k^1 = 1$, and in the absence of knowledge of this choice, the second agent also applies $u_k^2 = 1$ (thinking that the first agent will use the base policy control $u_k^1 = 0$). Thus the cost of the rollout policy is 2 per stage, while the cost of the base policy is 1 per stage. By contrast the rollout algorithm that takes into account the first agent's control when selecting the second agent's control applies $u_k^1 = 1$ and $u_k^2 = 0$, thus resulting in a rollout policy with the optimal cost of 0 per stage.

The difficulty here is inadequate coordination between the two agents. In particular, each agent uses rollout to compute the local control, thinking that the other will use the base policy control. If instead the two agents coordinated their control choices, they would have applied an optimal policy.

The simplicity of the preceding example raises serious questions as to whether the cost improvement property (2.86) can be easily maintained by a distributed rollout algorithm where the agents do not know the controls applied by the preceding agents in the given order of local control selection, and use instead the controls of the base policy. One may speculate that if the agents are naturally "weakly coupled" in the sense that their choice of control has little impact on the desirability of various controls of other agents, then a more flexible inter-agent communication pattern may be sufficient for cost improvement.†

An important question is to clarify the extent to which agent coordination is essential. In what follows in this section, we will discuss a distributed asynchronous multiagent rollout scheme, which is based on the use of a signaling policy that provides estimates of coordinating information once the current state is known.

Autonomous Multiagent Rollout - Signaling Policies

An interesting possibility for autonomous control selection by the agents is to use a distributed rollout algorithm, which is augmented by a precomputed signaling policy that embodies agent coordination.‡ The idea is to assume that the agents do not communicate their computed rollout control

† In particular, one may divide the agents in "coupled" groups, and require coordination of control selection only within each group, while the computation of different groups may proceed in parallel. Note that the "coupled" group formations may change over time, depending on the current state. For example, in applications where the agents' locations are distributed within some geographical area, it may make sense to form agent groups on the basis of geographic proximity, i.e., one may require that agents that are geographically near each other (and hence are more coupled) coordinate their control selections, while agents that are geographically far apart (and hence are less coupled) forego any coordination.

‡ The general idea of coordination by sharing information about the agents' policies arises also in other multiagent algorithmic contexts, including some that involve forms of policy gradient methods and Q-learning; see the surveys of the

components to the subsequent agents in the given order of local control selection. Instead, *once the agents know the state, they use precomputed (or easily computed) approximations to the control components of the preceding agents*, and compute their own control components in parallel and asynchronously. We call this algorithm *autonomous multiagent rollout*. While this type of algorithm involves a form of redundant computation, it allows for additional speedup through parallelization.

The algorithm at the kth stage uses a base policy $\mu_k = \{\mu_k^1, \ldots, \mu_k^{m-1}\}$, but also uses *a second policy* $\widehat{\mu}_k = \{\widehat{\mu}_k^1, \ldots, \widehat{\mu}_k^{m-1}\}$, called the *signaling policy*, which is computed off-line, is known to all the agents for on-line use, and is designed to play an agent coordination role. Intuitively, $\widehat{\mu}_k^\ell(x_k)$ *provides an intelligent "guess" about what agent ℓ will do at state x_k*. This is used in turn by all other agents $i \neq \ell$ to compute asynchronously their own rollout control components on-line.

More precisely, the autonomous multiagent rollout algorithm uses the base and signaling policies to generate a rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ as follows. At stage k and state x_k, $\tilde{\mu}_k(x_k) = \big(\tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^m(x_k)\big)$, is obtained according to

$$\begin{aligned}
\tilde{\mu}_k^1(x_k) \in \arg\min_{u_k^1 \in U_k^1(x_k)} & E\Big\{ g_k\big(x_k, u_k^1, \mu_k^2(x_k), \ldots, \mu_k^m(x_k), w_k\big) \\
& \qquad + J_{k+1,\pi}\Big(f_k\big(x_k, u_k^1, \mu_k^2(x_k), \ldots, \mu_k^m(x_k), w_k\big)\Big)\Big\}, \\
\tilde{\mu}_k^2(x_k) \in \arg\min_{u_k^2 \in U_k^2(x_k)} & E\Big\{ g_k\big(x_k, \widehat{\mu}_k^1(x_k), u_k^2, \ldots, \mu_k^m(x_k), w_k\big) \\
& \qquad + J_{k+1,\pi}\Big(f_k\big(x_k, \widehat{\mu}_k^1(x_k), u_k^2, \ldots, \mu_k^m(x_k), w_k\big)\Big)\Big\}, \\
& \qquad \cdots \qquad\qquad \cdots \qquad\qquad \cdots \\
\tilde{\mu}_k^m(x_k) \in \arg\min_{u_k^m \in U_k^m(x_k)} & E\Big\{ g_k\big(x_k, \widehat{\mu}_k^1(x_k), \ldots, \widehat{\mu}_k^{m-1}(x_k), u_k^m, w_k\big) \\
& \qquad + J_{k+1,\pi}\Big(f_k\big(x_k, \widehat{\mu}_k^1(x_k), \ldots, \widehat{\mu}_k^{m-1}(x_k), u_k^m, w_k\big)\Big)\Big\}.
\end{aligned} \tag{2.88}$$

Note that the preceding computation of the controls $\tilde{\mu}_k^1(x_k), \ldots, \tilde{\mu}_k^m(x_k)$ can be done asynchronously and in parallel, and without direct agent coordination, since the signaling policy values $\widehat{\mu}_k^1(x_k), \ldots, \widehat{\mu}_k^{m-1}(x_k)$ are precomputed and are known to all the agents.

The simplest choice is to *use as signaling policy $\widehat{\mu}$ the base policy μ*. However, this choice does not guarantee policy improvement as evidenced by Example 2.9.3. In fact performance deterioration with this choice is not uncommon, and can be observed in more complicated examples, including the following.

---

relevant research cited earlier. The survey by Matignon, Laurent, and Le Fort-Piat [MLL12] focuses on coordination problems from an RL point of view.

Example 2.9.4 (Spiders and Flies - Use of the Base Policy for Signaling)

Consider the problem of Example 2.9.1, which involves two spiders and two flies on a line, and the base policy μ that moves a spider towards the closest surviving fly (and in case where a spider starts at the midpoint between the two flies, moves the spider to the right). Assume that we use as signaling policy $\widehat{\mu}$ the base policy μ. It can then be verified that if the spiders start from different positions, the rollout policy will be optimal (will move the spiders in opposite directions). If, however, the spiders start *from the same position*, a completely symmetric situation is created, whereby the rollout controls move both flies in the direction of the fly *furthest away* from the spiders' position (or to the left in the case where the spiders start at the midpoint between the two flies). Thus, the flies end up oscillating around the middle of the interval between the flies and never catch the flies!

The preceding example is representative of a broad class of counterexamples that involve multiple identical agents. If the agents start at the same initial state, with a base policy that has identical components, and use the base policy for signaling, the agents will select identical controls under the corresponding multiagent rollout policy, ending up with a potentially serious cost deterioration.

This example also highlights an effect of the sequential choice of the control components $u_k^1, \ldots, u_k^m$, based on the reformulated problem of Fig. 2.9.1: it tends to break symmetries and "group think" that guides the agents towards selecting the same controls under identical conditions. Generally, any sensible multiagent policy must be able to deal in some way with this "group think" issue. One simple possibility is for each agent ℓ to randomize somehow the control choices of other agents $j \neq \ell$ when choosing its own control, particularly in "tightly coupled" cases where the choice of agent ℓ is "strongly" affected by the choices of the agents $j \neq \ell$.

An alternative idea is to choose the signaling policy $\widehat{\mu}_k$ to approximate the sequential multiagent rollout policy (the one computed with each agent knowing the controls applied by the preceding agents), or some other policy that is known to embody coordination between the agents. In particular, we may obtain $\widehat{\mu}_k$ as the multiagent rollout policy for a related but simpler problem, such as a certainty equivalent version of the original problem, whereby the stochastic system is replaced by a deterministic one.

Another interesting possibility is to compute $\widehat{\mu}_k = (\widehat{\mu}_k^1, \ldots, \widehat{\mu}_k^m)$ by off-line training of a neural network (or m networks, one per agent) with training samples generated through the sequential multiagent rollout policy. We intuitively expect that if the neural network provides a signaling policy that approximates well the sequential multiagent rollout policy, we would obtain better performance than the base policy. This expectation was confirmed in a case study involving a large-scale multi-robot repair application (see [BKB20]).

The advantage of autonomous multiagent rollout with neural network or other approximations is that it may lead to approximate policy improvement, while at the same time allowing asynchronous agent operation without coordination through communication of their rollout control values (but still assuming knowledge of the exact state by all agents).

2.10 ROLLOUT FOR BAYESIAN OPTIMIZATION AND SEQUENTIAL ESTIMATION

In this section, we discuss a wide class of problems that has been studied intensively in statistics and related fields since the 1940s. Roughly speaking, in these problems we use observations and sampling for the purpose of inference, but the number and the type of observations are not fixed in advance. Instead, the outcomes of the observations are sequentially evaluated on-line with a view towards stopping or modifying the observation process. This involves sequential decision making, thus bringing to bear exact and approximate DP. A central issue here is to estimate an m-dimensional random vector θ, using optimal sequential selection of observations, which are based on feedback from preceding observations; see Fig. 2.10.1. Here is a simple but historically important illustrative example, where θ represents a binary hypothesis.

Example 2.10.1 (Hypothesis Testing - Sequential Probability Ratio Test)

Consider a hypothesis testing problem whereby we can make observations, at a cost C each, relating to two hypotheses. Given a new observation, we can either accept one of the hypotheses or delay the decision for one more period, pay the cost C, and obtain a new observation. At issue is trading off the cost of observation with the higher probability of accepting the wrong hypothesis. As an example, in a quality control setting, the two hypotheses may be that a certain product meets or does not meet a certain level of quality, while the observations may consist of quantitative tests of the quality of the product.

Intuitively, one expects that once the conditional probability of one of the hypotheses, given the observations thus far, gets sufficiently close to 1, we should stop the observations. Indeed classical DP analyses bear this out; see e.g., the books by Chernoff [Che72], DeGroot [DeG70], Whittle [Whi82], and the references quoted therein. In particular, the simple version of the hypothesis testing problem just described admits a simple and elegant optimal solution, known as the *sequential probability ratio test*. On the other hand more complex versions of the problem, involving for example multiple hypotheses and/or multiple types of observations, are computationally intractable, thus necessitating the use of suboptimal approaches.

An important distinction in sequential estimation problems is whether the current choice of observation affects the cost and the availability of

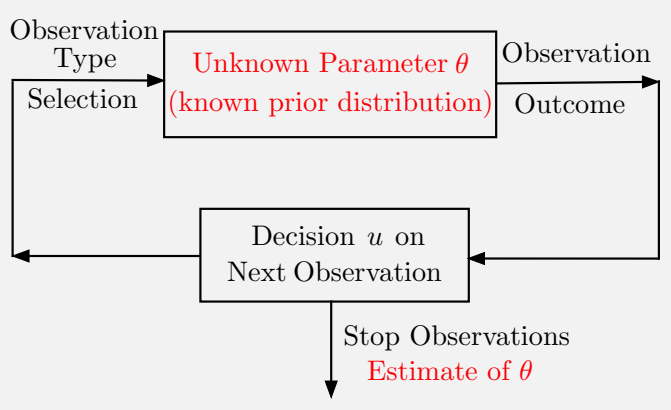

Figure 2.10.1 Illustration of sequential estimation of a parameter θ. At each time a decision is made to select one of several observation types relating to θ, each of different cost, or stop the observations and provide a final estimate of θ.

future observations. If this is so, the problem can often be viewed most fruitfully as a *combined estimation and control problem*, and is related to a type of *adaptive control* problem that we will discuss in the next section. As an example we will consider there sequential decoding, whereby we search for a hidden code word by using a sequence of queries, in the spirit of the Wordle puzzle and the family of Mastermind games [see, e.g., the Wikipedia page for "Mastermind (board game)"].

If the observation choices are "independent" and do not affect the cost or availability of future observations, the problem is substantially simplified. We will discuss problems of this type in the present section, starting with the case of surrogate and Bayesian optimization.

Surrogate Optimization

Surrogate optimization refers to a collection of methods, which address suboptimally a broad range of minimization problems, beyond the realm of DP. The problem is to minimize approximately a function that is given as a "black box." By this we mean a function whose analytical expression is unknown, and whose values at any one point may be hard-to-compute, e.g., may requite costly simulation or experimentation. The idea is to replace such a cost function with a "surrogate" whose values are easier to compute.

Here we introduce a model of the cost function that is parametrized by a parameter θ; see Fig. 2.10.2. We observe sequentially the cost function at a few observation points, construct a model of the cost function (the surrogate) by estimating θ based on the results of the observations, and minimize the surrogate to obtain a suboptimal solution. The question is how to select observation points sequentially, using feedback from previous observations. This selection process often embodies an *exploration-*

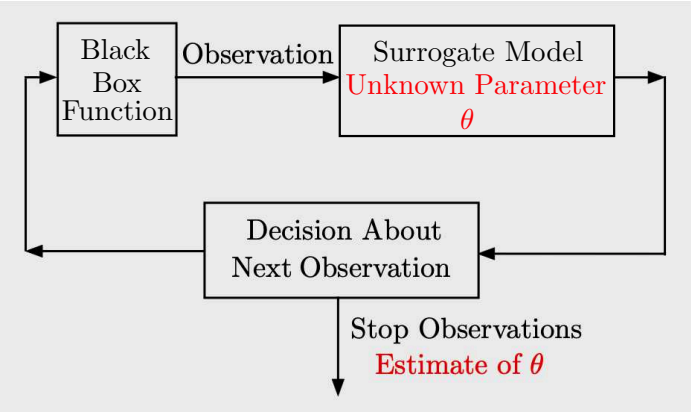

Figure 2.10.2 Illustration of construction of a surrogate for a "black box" function f whose values are hard-to-compute. We replace f with a parametric model that involves a parameter θ to be estimated by using observations at some points. The points are selected sequentially, using the results of earlier observations. Eventually, the observation process is stopped (often when an observation/computation budget limit is reached), and the final estimate of θ is used to construct the surrogate to be minimized in place of f.

exploitation tradeoff: Observing at points likely to have near-optimal value vs observing at points in relatively unexplored areas of the search space.

Surrogate optimization at its core involves construction from data of functions of interest. Thus the ideas to be presented apply to other domains, e.g., the construction of probability density functions from data.

Bayesian Optimization

Bayesian optimization (BO) has been used widely for the approximate optimization of functions whose values at given points can only be obtained through time-consuming calculation, simulation, or experimentation. A classical application from geostatistical interpolation, pioneered by the statisticians Matheron and Krige, was to identify locations of high gold distribution in South Africa based on samples from a few boreholes (the name "kriging" is often used to refer to this type of application; see the review by Kleijnen [Kle09]). As another example, BO has been used to select the hyperparameters of machine-learning models, including the architectural parameters of the deep neural network of AlphaZero; see [SHS17].

In this section, we will focus on a relatively simple BO formulation that can be viewed as the special case of surrogate optimization. In particular, we will discuss the case where the surrogate function is parametrized by the collection of its values at the points where it is defined.† See the references cited later in this section. Formally, we want to minimize a

† More complex forms of surrogates are obtained through linear combinations

real-valued function f, defined over a set of m points, which we denote by $1,\ldots,m$. These m points lie in some space, which we leave unspecified for the moment.† The values of the function are not readily available, but can be estimated with observations that may be imperfect. However, the observations are so costly that we can only hope to observe the function at a limited number of points. Once the function has been estimated with this type of observation process, we obtain a surrogate cost function, which may be minimized to obtain an approximately optimal solution.

We denote the value of f at a point u by θ_u:

$$\theta_u = f(u), \qquad \text{for all } u = 1,\ldots,m.$$

Thus the m-dimensional vector $\theta = (\theta_1,\ldots,\theta_m)$ belongs to $\Re^m$ and represents the function f. We assume that we obtain sequentially noisy observations of values $f(u) = \theta_u$ at suitably selected points u. These values are used to estimate the vector θ (i.e., the function f), and to ultimately minimize (approximately) f over the m points $u = 1,\ldots,m$. The essence of the problem is to select points for observation based on an exploration-exploitation tradeoff (exploring the potential of relatively unexplored candidate solutions and improving the estimate of promising candidate solutions). The fundamental idea of the BO methodology is that the function value changes relatively slowly, so that observing the function value at some point provides information about the function values at neighboring points. Thus a limited number of strategically chosen observations can provide reasonable approximation to the true cost function over a large portion of the search space.

For a mathematical formulation of a BO framework, we assume that at each of N successive times $k = 1,\ldots,N$, we select a single point $u_k \in \{1,\ldots,m\}$, and observe the corresponding component θ_{u_k} of θ (i.e., the function value at u_k) with some noise w_{u_k}, i.e.,

$$z_{u_k} = \theta_{u_k} + w_{u_k}; \tag{2.89}$$

see Fig. 2.10.3. We view the observation points $u_1,\ldots,u_N$ as the optimization variables (or controls/actions in a DP/RL context), and consider policies for selecting u_k with knowledge of the preceding observations $z_{u_1},\ldots,z_{u_{k-1}}$ that have resulted from the selections $u_1,\ldots,u_{k-1}$. We assume that the noise random variables w_u, $u \in \{1,\ldots,m\}$ are independent

---

of some basis functions, with the parameter vector θ consisting of the weights of the basis functions.

† We restrict the domain of definition of f to be the finite set $\{1,\ldots,m\}$ in order to facilitate the implementation of the rollout algorithm to be discussed in what follows. However, in a more general and sometimes more convenient formulation, the domain of f can be an infinite set, such as a subset of a finite-dimensional Euclidean space.

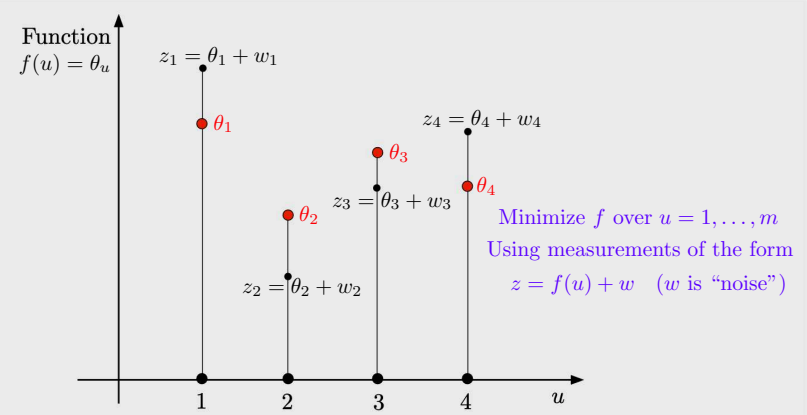

Figure 2.10.3 Illustration of a function f that we wish to estimate. The function is defined at the points $u = 1, 2, 3, 4$, and is represented by a vector $\theta = (\theta_1, \theta_2, \theta_3, \theta_4) \in \Re^4$, in the sense that $f(u) = \theta_u$ for all u. The prior distribution of θ is given, and is used to construct the posterior distribution of θ given noisy observations $z_u = \theta_u + w_u$ at some of the points u.

and that their distributions are given. Moreover, we assume that θ has a given a priori distribution on the space of m-dimensional vectors $\Re^m$, which we denote by b_0. The posterior distribution of θ, given any subset of observations

$$\{z_{u_1}, \ldots, z_{u_k}\},$$

is denoted by b_k.

An important special case arises when b_0 and the distributions of w_u, $u \in \{1, \ldots, m\}$, are Gaussian. In this case b_0 is a multidimensional Gaussian distribution, defined by its mean (based on prior knowledge, or an equal value for all $u = 1, \ldots, m$ in case of absence of such knowledge) and its covariance matrix [implying greater correlation for pairs (u, u') that are "close" to each other in some problem-specific sense, e.g., exponentially decreasing with the Euclidean distance between u and u']. A key consequence of this assumption is that the posterior distribution b_k is multidimensional Gaussian, and can be calculated in closed form by using well-known formulas.

More generally, b_k evolves according to an equation of the form

$$b_{k+1} = B_k(b_k, u_{k+1}, z_{u_{k+1}}), \qquad k = 0, \ldots, N-1. \tag{2.90}$$

Thus given the set of observations up to time k, and the next choice u_{k+1}, resulting in an observation value $z_{u_{k+1}}$, the function B_k gives the formula for updating b_k to b_{k+1}, and may be viewed as a recursive estimator of b_k. In the Gaussian case, the function B_k can be written in closed form, using standard formulas for Gaussian random vector estimation. In other

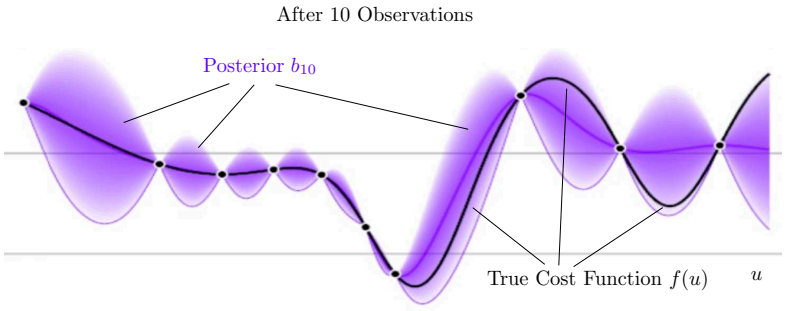

Figure 2.10.4 Illustration of the true cost function f, defined over an interval of the real line, and the posterior distribution b_{10} after noise-free measurements at 10 points. The shaded area represents the interval of the mean plus/minus the standard deviation of the posterior b_{10} at the points u. The mean of the finally obtained posterior, as a function of u, may be viewed as a surrogate cost function that can be minimized in place of f. Note that since the observations are assumed noise-free, the mean of the posterior is exact at the observation points.

cases where no closed form expression is possible, B_k can be implemented through simulation that computes (approximately) the new posterior b_{k+1} using samples generated from the current posterior b_k.

At the end of the sequential estimation process, after the complete observation set

$$\{z_{u_1}, \ldots, z_{u_N}\}$$

has been obtained, we have the posterior distribution b_N of θ, which we can use to compute a surrogate of f. As an example we may use as surrogate the posterior mean $\hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_m)$, and declare as minimizer of f over u the point u^* with minimum posterior mean:

$$u^* \in \arg\min\{\hat{\theta}_u \mid u = 1, \ldots, m\};$$

see Fig. 2.10.4.

There is a large literature relating to the surrogate and Bayesian optimization methodology and its applications, particularly for the Gaussian case. We refer to the books by Rasmussen and Williams [RaW06], Powell and Ryzhov [PoR12], the highly cited papers by Saks et al. [SWM89], Jones, Schonlau, and Welch [JSW98], and Queipo et al. [QHS05], the reviews by Sasena [Sas02], Powell and Frazier [PoF08], Forrester and Keane [FoK09], Kleijnen [Kle09], Brochu, Cora, and De Freitas [BCD10], Ryzhov, Powell, and Frazier [RPF12], Ghavamzadeh, Mannor, Pineau, and Tamar [GMP15], Shahriari et al. [SSW16], and Frazier [Fra18], and the references quoted there. Our purpose here is to focus on the aspects of the subject that are most closely connected to exact and approximate DP.

A Dynamic Programming Formulation

The sequential estimation problem just described, viewed as a DP problem, involves a state at time k, which is the posterior (or belief state) b_k, and a control/action at time k, which is the point index u_{k+1} selected for observation. The transition equation according to which the state evolves, is

$$b_{k+1} = B_k(b_k, u_{k+1}, z_{u_{k+1}}), \qquad k = 0, \ldots, N-1;$$

cf. Eq. (2.90). To complete the DP formulation, we need to introduce a cost structure. To this end, we assume that observing θ_u, as per Eq. (2.89), incurs a cost $c(u)$, and that there is a terminal cost $G(b_N)$ that depends of the final posterior distribution; as an example, the function G may involve the mean and covariance corresponding to b_N.

The corresponding DP algorithm is given by

$$J_k^*(b_k) = \min_{u_{k+1} \in \{1,\ldots,m\}} \Big[c(u_{k+1}) + E_{z_{u_{k+1}}} \Big\{ J_{k+1}^*\big(B_k(b_k, u_{k+1}, z_{u_{k+1}})\big) \mid b_k, u_{k+1} \Big\} \Big], \tag{2.91}$$

and proceeds backwards from the terminal condition

$$J_N^*(b_N) = G(b_N). \tag{2.92}$$

The expected value in the right side of the DP equation (2.91) is taken with respect to the conditional distribution of $z_{u_{k+1}}$, given b_k and the choice u_{k+1}. The observation cost $c(u)$ may be 0 or a constant for all u, but it can also have a more complicated dependence on u. The terminal cost $G(b_N)$ may be a suitable measure of surrogate "fidelity" that depends on the posterior mean and covariance of θ corresponding to b_N.

Generally, executing the DP algorithm (2.91) is practically infeasible, because the space of posterior distributions is infinite-dimensional. In the Gaussian case where the a priori distribution b_0 is Gaussian and the noise variables w_u are Gaussian, the posterior b_k is m-dimensional Gaussian, so it is characterized by its mean and covariance, and can be specified by a finite set of numbers. Despite this simplification, the DP algorithm (2.91) is prohibitively time-consuming even under Gaussian assumptions, except for simple special cases. We consequently resort to approximation in value space, whereby the function J_{k+1}^* in the right side of Eq. (2.91) is replaced by an approximation $\tilde{J}_{k+1}$.

Approximation in Value Space

The most popular BO methodology makes use of a myopic/greedy policy μ_{k+1}, which at each time k and given b_k, selects a point $\hat{u}_{k+1} = \mu_{k+1}(b_k)$

for the next observation, using some calculation involving an *acquisition function*. This function, denoted $A_k(b_k, u_{k+1})$, quantifies some form of "expected benefit" for an observation at u_{k+1}, given the current posterior b_k.† The myopic policy selects the next point at which to observe, $\hat{u}_{k+1}$, by maximizing the acquisition function:

$$\hat{u}_{k+1} \in \arg \max_{u_{k+1} \in \{1,\ldots,m\}} A_k(b_k, u_{k+1}). \tag{2.93}$$

Several ways to define suitable acquisition functions have been proposed, and an important issue is to be able to calculate economically its values $A_k(b_k, u_{k+1})$ for the purposes of the maximization in Eq. (2.93). Another important issue of course is to be able to calculate the posterior b_k economically.

Approximation in value space is an alternative approach, which is based on the DP formulation of the preceding section. In particular, in this approach we approximate the DP algorithm (2.91) by replacing J^*_{k+1} with an approximation $\tilde{J}_{k+1}$ in the minimization of the right side. Thus we select the next observation at point $\tilde{u}_{k+1}$ according to

$$\tilde{u}_{k+1} \in \arg \min_{u_{k+1} \in \{1,\ldots,m\}} Q_k(b_k, u_{k+1}), \qquad k = 0, \ldots, N-1, \tag{2.94}$$

where $Q_k(b_k, u_{k+1})$ is the Q-factor corresponding to the pair (b_k, u_{k+1}), given by

$$Q_k(b_k, u_{k+1}) = c(u_{k+1}) + E_{z_{u_{k+1}}} \Big\{ \tilde{J}_{k+1}\big(B_k(b_k, u_{k+1}, z_{u_{k+1}})\big) \;\Big|\; b_k, u_{k+1} \Big\}. \tag{2.95}$$

The expected value in the preceding equation is taken with respect to the conditional probability distribution of $z_{u_{k+1}}$ given (b_k, u_{k+1}), which can be

† A common type of acquisition function is the *upper confidence bound*, which has the form

$$A_k(b_k, u) = T_k(b_k, u) + \beta R_k(b_k, u),$$

where $T_k(b_k, u)$ is the negative of the mean of $f(u)$ under the posterior distribution b_k, $R_k(b_k, u)$ is the standard deviation of $f(u)$ under the posterior distribution b_k, and β is a tunable positive scalar parameter. Thus $T_k(b_k, u)$ can be viewed as an *exploitation index* (encoding our desire to search within parts of the space where f takes low value), while $R_k(b_k, u)$ can be viewed as an *exploration index* (encoding our desire to search within parts of the space that are relatively unexplored). There are several other popular acquisition functions, which directly or indirectly embody a tradeoff between exploitation and exploration. A popular example is the *expected improvement* acquisition function, which is equal to the expected value of the reduction of $f(u)$ relative to the minimal value of f obtained up to time k (under the posterior distribution b_k).

computed using b_k and the given distribution of the noise $w_{u_{k+1}}$. Thus if b_k and $\tilde{J}_{k+1}$ are available, we may use Monte Carlo simulation to determine the Q-factors $Q_k(b_k, u_{k+1})$ for all $u_{k+1} \in \{1, \ldots, m\}$, and select as next point for observation the one that corresponds to the minimal Q-factor [cf. Eq. (2.94)].

Rollout Algorithms for Bayesian Optimization

A special case of approximation in value space is the rollout algorithm, whereby the function J^*_{k+1} in the right side of the DP Eq. (2.91) is replaced by the cost function of some base policy $\mu_{k+1}(b_k)$, $k = 0, \ldots, N-1$. Thus, given a base policy the rollout algorithm uses the cost function of this policy as the function $\tilde{J}_{k+1}$ in the approximation in value space scheme (2.94)-(2.95). The values of $\tilde{J}_{k+1}$ needed for the Q-factor calculations in Eq. (2.95) can be computed or approximated by simulation. Greedy/myopic policies based on an acquisition function [cf. Eq. (2.93)] have been suggested as base policies in various rollout proposals.†

In particular, given b_k, the rollout algorithm computes for each $u_{k+1} \in \{1, \ldots, m\}$ a Q-factor value $Q_k(b_k, u_{k+1})$ by simulating the base policy for multiple time steps starting from all possible posteriors b_{k+1} that can be generated from (b_k, u_{k+1}), and by accumulating the corresponding cost [including a terminal cost such as $G(b_N)$]; see Fig. 2.10.5. It then selects the next point $\tilde{u}_{k+1}$ for observation by using the Q-factor minimization of Eq. (2.94).

Note that the equation

$$b_{k+1} = B_k(b_k, u_{k+1}, z_{u_{k+1}}), \qquad k = 0, \ldots, N-1,$$

which governs the evolution of the posterior distribution (or belief state), is stochastic because $z_{u_{k+1}}$ involves the stochastic noise $w_{u_{k+1}}$. Thus some Monte Carlo simulation is unavoidable in the calculation of the Q-factors $Q_k(b_k, u_{k+1})$. On the other hand, one may greatly reduce the Monte Carlo computational burden by employing a certainty equivalence approximation, which at stage k, treats only the noise $w_{u_{k+1}}$ as stochastic, and replaces the noise variables $w_{u_{k+2}}, w_{u_{k+3}}, \ldots$, after the first stage of the calculation, by deterministic quantities such as their means $\hat{w}_{u_{k+2}}, \hat{w}_{u_{k+3}}, \ldots$.

† The rollout algorithm for BO was first proposed under Gaussian assumptions by Lam, Wilcox, and Wolpert [LWW16]. It was further discussed by Jiang et al. [JJB20], [JCG20], Lee at al. [LEC20], Lee [Lee20], Yue and Kontar [YuK20], Lee et al. [LEP21], Paulson, Sorouifar, and Chakrabarty [PSC22], where it is also referred to as "nonmyopic BO" or "nonmyopic sequential experimental design." For related work, see Gerlach, Hoffmann, and Charlish [GHC21]. These papers also discuss various approximations to the rollout approach, and generally report encouraging computational results. Section 3.5 of the author's book [Ber20a] focuses on rollout algorithms for surrogate and Bayesian optimization.

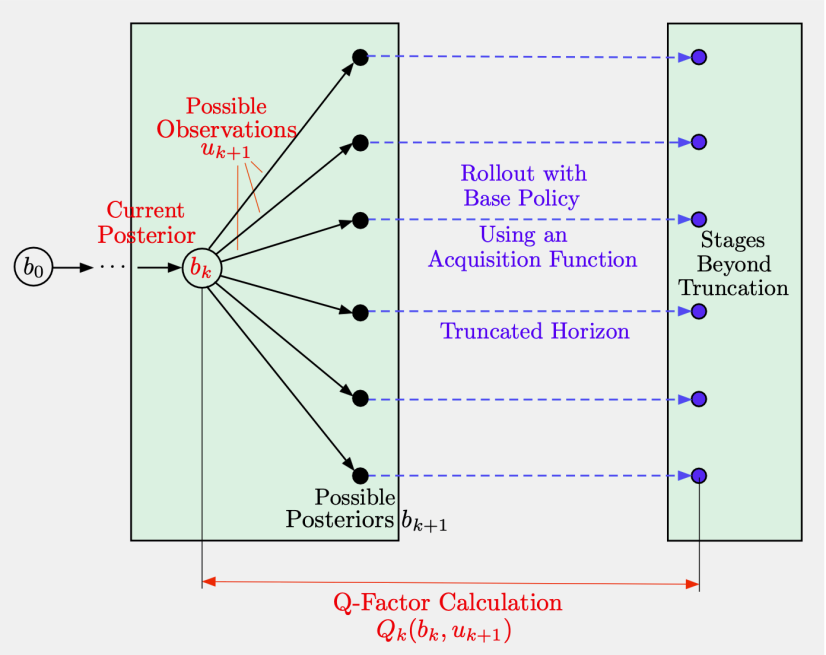

Figure 2.10.5 Illustration of rollout at the current posterior b_k. For each $u_{k+1} \in \{1, \ldots, m\}$, we compute the Q-factor $Q_k(b_k, u_{k+1})$ by using Monte-Carlo simulation with samples from $w_{u_{k+1}}$ and a base heuristic that uses an acquisition function starting from each possible posterior b_{k+1}. The rollout may extend to the end of the horizon N, or it may be truncated after a few steps.

The simulation of the Q-factor values may also involve other approximations, some of which have been suggested in various proposals for rollout-based BO. For example, if the number of possible observations m is very large, we may compute and compare the Q-factors of only a subset. In particular, at a given time k, we may rank the observations by using an acquisition function, select a subset U_{k+1} of most promising observations, compute their Q-factors $Q_k(b_k, u_{k+1})$, $u_{k+1} \in U_{k+1}$, and select the observation whose Q-factor is minimal; this idea has been used in the case of the Wordle puzzle in the papers by Bhambri, Bhattacharjee, and Bertsekas [BBB22], [BBB23], which will be discussed in the next section.

Multiagent Rollout for Bayesian Optimization

In some BO applications there arises the possibility of simultaneously performing multiple observations before receiving feedback about the corresponding observation outcomes. This occurs, among others, in two important contexts:

(a) In parallel computation settings, where multiple processors are used to perform simultaneously expensive evaluations of the function f at multiple points u. These evaluations may involve some form of truncated simulation, so they yield evaluations of the form $z_u = \theta_u + w_u$, where w_u is the simulation noise.

(b) In distributed sensor systems, where a number of sensors provide in parallel relevant information about the random vector θ that we want to estimate; see e.g., the recent paper by Li, Krakow, and Gopalswamy [LKG21], which describes related multisensor estimation problems, based on the multiagent rollout methodology of Section 2.9.

Of course in such cases we may treat the entire set of simultaneous observations as a single observation within an enlarged Cartesian product space of observations, but there is a fundamental difficulty: the size of the observation space (and hence the number of Q-factors to be calculated by rollout at each time step) grows exponentially with the number of simultaneous observations. This in turn greatly increases the computational requirements of the rollout algorithm.

To address this difficulty, we may employ the methodology of multiagent rollout whereby the policy improvement is done one-agent-at-a-time in a given order, with (possibly partial) knowledge of the choices of the preceding agents in the order. As a result, the amount of computation for each policy improvement grows linearly with the number of agents, as opposed to exponentially for the standard all-agents-at-once method. At the same time the theoretical cost improvement property of the rollout algorithm can be shown to be preserved, while the empirical evidence suggests that great computational savings are achieved with hardly any performance degradation.

Generalization to Sequential Estimation of Random Vectors

Aside from BO, there are several other types of simple sequential estimation problems, which involve "independent sampling," i.e., problems where the choice of an observation type does not affect the quality, cost, or availability of observations of other types. A common class of problems that contains BO as a special case and admits a similar treatment, is to sequentially estimate an m-dimensional random vector $\theta = (\theta_1, \ldots, \theta_m)$ by using N linear observations of θ of the form

$$z_u = a_u' \theta + w_u, \qquad u \in \{1, \ldots, n\},$$

where n is some integer. Here w_u are independent random variables with given probability distributions, the m-dimensional vectors a_u are known, and $a_u'\theta$ denotes the inner product of a_u and θ. Similar to the case of BO, the problem simplifies if the given a priori distribution of θ is Gaussian,

and the random variables w_u are independent and Gaussian. Then, the posterior distribution of θ, given any subset of observations, is Gaussian (thanks to the linearity of the observations), and can be calculated in closed form.

Observations are generated sequentially at times $1, \ldots, N$, one at a time and with knowledge of the outcomes of the preceding observations, by choosing an index $u_k \in \{1, \ldots, n\}$ at time k, at a cost $c(u_k)$. Thus u_k are the optimization variables, and affect both the quality of estimation of θ and the observation cost. The objective, roughly speaking, is to select N observations to estimate θ in a way that minimizes an appropriate cost function; for example, one that penalizes some form of estimation error plus the cost of the observations. We can similarly formulate the corresponding optimization problem in terms of N-stage DP, and develop rollout algorithms for its approximate solution.

2.11 ADAPTIVE CONTROL BY ROLLOUT WITH A POMDP FORMULATION

In this section, we discuss various approaches for the approximate solution of Partially Observed Markovian Decision Problems (POMDP) with a special structure, which is well-suited for adaptive control, as well as other contexts that involve search for a hidden object.† It is well known that POMDP are among the most challenging DP problems, and nearly always require the use of approximations for (suboptimal) solution.

The application and implementation of rollout and approximate PI methods to general finite-state POMDP is described in the author's RL book [Ber19a] (Section 5.7.3). Here we will focus attention on a special class of POMDP where the state consists of two components:

(a) A perfectly observed component x_k that evolves over time according to a discrete-time equation.

(b) An unobserved component θ that stays constant and is estimated through the perfect observations of the component x_k.

We view θ as a parameter in the system equation that governs the evolution of x_k, hence the connection with adaptive control. Thus we have

$$x_{k+1} = f_k(x_k, \theta, u_k, w_k), \tag{2.96}$$

where u_k is the control at time k, selected from a set $U_k(x_k)$, and w_k is a random disturbance with given probability distribution that depends

† In Section 1.6.6, we discussed the indirect adaptive control approach, which enforces a separation of the controller into a system identification algorithm and a policy reoptimization algorithm. The POMDP approach of this section (also summarized in Section 1.6.6), does not assume such an a priori separation, and is thus founded on a more principled algorithmic framework.

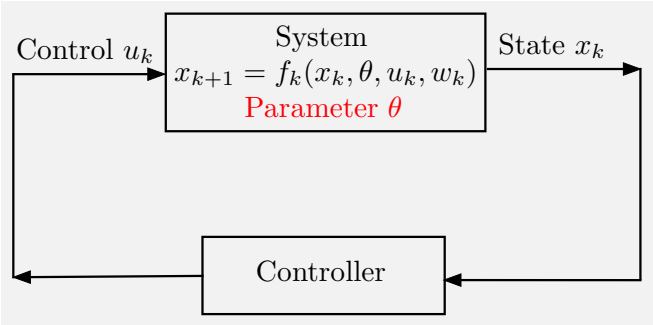

Figure 2.11.1 Illustration of an adaptive control scheme involving perfect state observation of a system with an unknown parameter θ. At each time a decision is made to select a control and (possibly) one of several observation types, each of different cost.

on (x_k, θ, u_k). We will assume that θ can take one of m known values $\theta^1, \ldots, \theta^m$:

$$\theta \in \{\theta^1, \ldots, \theta^m\},$$

see Fig. 2.11.1.

The a priori probability distribution of θ is given and is updated based on the observed values of the state components x_k and the applied controls u_k. In particular, we assume that the information vector

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

is available at time k, and is used to compute the conditional probabilities

$$b_{k,i} = P\{\theta = \theta^i \mid I_k\}, \qquad i = 1, \ldots, m.$$

These probabilities form a vector

$$b_k = (b_{k,1}, \ldots, b_{k,m}),$$

which together with the perfectly observed state x_k, form the pair (x_k, b_k) that is commonly called the *belief state* of the POMDP at time k.

Note that according to the classical methodology of POMDP (see e.g., [Ber17a], Chapter 4), the belief component b_{k+1} is determined by the belief state (x_k, b_k), the control u_k, and the observation obtained at time $k+1$, i.e., x_{k+1}. Thus b_k can be updated according to an equation of the form

$$b_{k+1} = B_k(x_k, b_k, u_k, x_{k+1}),$$

where B_k is an appropriate function, which can be viewed as a recursive estimator of θ. There are several approaches to implement this estimator (perhaps with some approximation error), including the use of Bayes' rule and the simulation-based method of particle filtering.

The preceding mathematical model forms the basis for a classical adaptive control formulation, where each θ^i represents a set of system parameters, and the computation of the belief probabilities $b_{k,i}$ can be viewed as the outcome of a system identification algorithm. In this context, the problem becomes one of *dual control*, a classical type of combined identification and control problem, whose optimal solution is notoriously difficult.

Another interesting context arises in search problems, where θ specifies the locations of one or more objects of interest within a given space. Some puzzles, including the popular Wordle game, fall within this category, as we will discuss briefly later in this section.

The Exact DP Algorithm - Approximation in Value Space

We will now describe an exact DP algorithm that operates in the space of information vectors I_k. To describe this algorithm, let us denote by $J_k(I_k)$ the optimal cost starting at information vector I_k at time k. We can view I_k as a state of the POMDP, which evolves over time according to the equation

$$I_{k+1} = (I_k, x_{k+1}, u_k) = \big(I_k, f_k(x_k, \theta, u_k, w_k), u_k\big).$$

Viewing this as a system equation, whose right hand side involves the state I_k, the control u_k, and the disturbance w_k, the DP algorithm takes the form

$$\begin{aligned} J_k^*(I_k) &= \min_{u_k \in U_k(x_k)} E_{\theta, w_k}\Big\{ g_k(x_k, \theta, u_k, w_k) + J_{k+1}^*(I_{k+1}) \mid I_k, u_k \Big\} \\ &= \min_{u_k \in U_k(x_k)} E_{\theta, w_k}\Big\{ g_k(x_k, \theta, u_k, w_k) + \\ &\qquad\qquad J_{k+1}^*\big(I_k, f_k(x_k, \theta, u_k, w_k), u_k\big) \mid I_k, u_k \Big\}, \end{aligned} \tag{2.97}$$

for $k = 0, \ldots, N-1$, with $J_N(I_N) = g_N(x_N)$; see e.g., the DP textbook [Ber17a], Section 4.1.

The algorithm (2.97) is typically very hard to implement, in part because of the dependence of J_{k+1}^* on the entire information vector I_{k+1}, which expands in size according to

$$I_{k+1} = (I_k, x_{k+1}, u_k).$$

To address this difficulty, we may use approximation in value space, based on replacing $J_{k+1}^*(I_{k+1})$ in the DP algorithm (2.97) with some function $\tilde{J}_{k+1}(I_{k+1})$ such that the expected value

$$E_{\theta, w_k}\Big\{ \tilde{J}_{k+1}(I_{k+1}) \mid I_k, u_k \Big\} \tag{2.98}$$

can be obtained with a tractable computation for any (I_k, u_k). A useful possibility arises when the cost function approximations

$$E_{w_k}\Big\{\tilde{J}_{k+1}(I_{k+1}) \mid I_k, \theta^i, u_k\Big\}$$

can be obtained for each *fixed value of* θ^i with a tractable computation. In this case, we may compute the cost function approximation (2.98) by using the formula

$$E_{\theta,w_k}\Big\{\tilde{J}_{k+1}(I_{k+1}) \mid I_k, u_k\Big\} = \sum_{i=1}^{m} b_{k,i} E_{w_k}\big\{\tilde{J}_{k+1}(I_{k+1}) \mid I_k, \theta, u_k\big\}$$

which follows from the law of iterated expectations,

$$E_{\theta,w_k}\{\cdot \mid I_k, u_k\} = E_{\theta}\big\{E_{w_k}\{\cdot \mid I_k, \theta, u_k\} \mid I_k, u_k\big\}.$$

We will now discuss some choices of functions $\tilde{J}_{k+1}$ with a structure that facilitates the implementation of the corresponding approximation in value space scheme. One possibility is to use the optimal cost functions corresponding to the m parameters θ^i,

$$\hat{J}^i_{k+1}(x_{k+1}), \qquad i = 1, \ldots, m. \tag{2.99}$$

In particular, $\hat{J}^i_{k+1}(x_{k+1})$ is the optimal cost that would be obtained starting from state x_{k+1} under the assumption that $\theta = \theta^i$; this corresponds to a perfect state information problem. Then an approximation in value space scheme with one-step lookahead minimization is given by

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(x_k)} \sum_{i=1}^{m} b_{k,i} E_{w_k}\Big\{g_k(x_k, \theta^i, u_k, w_k) + \hat{J}^i_{k+1}\big(f_k(x_k, \theta^i, u_k, w_k)\big) \mid x_k, \theta^i, u_k\Big\}. \tag{2.100}$$

In particular, instead of the optimal control, which minimizes the optimal Q-factor of (I_k, u_k) appearing in the right side of Eq. (2.97), we apply control $\tilde{u}_k$ that minimizes the expected value over θ of the optimal Q-factors that correspond to *fixed values of* θ.

In the case where the horizon is infinite, it is reasonable to expect that an improving estimate of the parameter θ can be obtained over time, and that with a suitable estimation scheme, it converges asymptotically to the correct value of θ, call it θ^*, i.e.,

$$\lim_{k\to\infty} b_{k,i} = \begin{cases} 1 & \text{if } \theta^i = \theta^*, \\ 0 & \text{if } \theta^i \neq \theta^*. \end{cases}$$

Then it can be seen that the generated one-step lookahead controls $\tilde{u}_k$ are asymptotically obtained from the Bellman equation that corresponds to the correct parameter θ^*, and are typically optimal in some asymptotic sense. Schemes of this type have been extensively discussed in the adaptive control literature since the 70s; see the end-of-chapter references and discussion.

Generally, the optimal costs $\hat{J}^i_{k+1}(x_{k+1})$ of Eq. (2.99), which correspond to the different parameter values θ^i, may be hard to compute, despite the fact that they correspond to perfect state information problems.† An alternative possibility is to use off-line trained approximations to $\hat{J}^i_{k+1}(x_{k+1})$ involving neural networks or other approximation architectures. Still another possibility, described next, is to use a rollout approach.

Rollout and Cost Improvement

A simpler possibility for approximation in value space is to use the cost of a given policy π^i in place of the optimal cost $\hat{J}^i_{k+1}(x_{k+1})$ of Eq. (2.99) that corresponds to θ^i. In this case the one-step lookahead scheme (2.100) takes the form

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} \sum_{i=1}^{m} b_{k,i} E_{w_k} \Big\{ g_k(x_k, \theta^i, u_k, w_k) + \hat{J}^i_{k+1,\pi^i}\big(f_k(x_k, \theta^i, u_k, w_k)\big) \mid x_k, \theta^i, u_k \Big\}, \tag{2.101}$$

with $\pi^i = \{\mu^i_0, \ldots, \mu^i_{N-1}\}$, $i = 1, \ldots, m$, being known policies, *with components μ^i_k that depend only on x_k*. Here, the term

$$\hat{J}^i_{k+1,\pi^i}\big(f_k(x_k, \theta^i, u_k, w_k)\big)$$

in Eq. (2.101) is the cost of the base policy π^i, calculated starting from the next state

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

under the assumption that θ will stay fixed at the value $\theta = \theta^i$ until the end of the horizon. Note that the cost function of π^i, conditioned on $\theta = \theta^i$, x_k, and u_k, which is needed in Eq. (2.101), can be calculated by Monte Carlo simulation. This is made possible by the fact that the components μ^i_k of π^i depend only on x_k [rather than I_k or the belief state (x_k, b_k)].

The preceding scheme has the character of a rollout algorithm, but strictly speaking, it does not qualify as a rollout algorithm because the

† In favorable special cases, such as linear quadratic problems, the optimal costs $\hat{J}^i_{k+1}(x_{k+1})$ may be easily calculated in closed form. Still, however, even in such cases the calculation of the belief probabilities $b_{k,i}$ may not be simple, and may require the use of a system identification algorithm.

policy components μ_k^i involve a dependence on i in addition to the dependence on x_k. On the other hand if we restrict all the policies π^i to be the same for all i, the corresponding functions μ_k depend only on x_k and not on i, thus defining a legitimate base policy. In this case the rollout scheme (2.101) amounts to replacing

$$E_{w_k}\big\{J_{k+1}^*(I_{k+1}) \mid I_k, u_k\big\}$$

in the DP algorithm (2.97) with

$$E_{w_k}\big\{J_{k+1,\pi}(I_{k+1}) \mid I_k, u_k\big\}.$$

Similar to Section 2.7, a cost improvement property can then be shown.

Within our rollout context, a policy π such that $\pi^i = \pi$ for all i should be a *robust* policy, in the sense that it should work adequately well for all parameter values θ^i. The method to obtain such a policy is likely problem-dependent. On the other hand robust policies have a long history in the context of adaptive control, and have been discussed widely (see e.g., the book by Jiang and Jiang [JiJ17], and the references quoted therein).

The Case of a Deterministic System

Let us now consider the case where the system (2.96) is deterministic of the form

$$x_{k+1} = f_k(x_k, \theta, u_k). \tag{2.102}$$

Then, while the problem still has a stochastic character due to the uncertainty about the value of θ, the DP algorithm (2.97) and its approximation in value space counterparts are greatly simplified because there is no expectation over w_k to contend with. Indeed, given a state x_k, a parameter θ^i, and a control u_k, the on-line computation of the control of the rollout-like algorithm (2.101), takes the form

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(x_k)} \sum_{i=1}^m b_{k,i}\Big(g_k(x_k, \theta^i, u_k) + \hat{J}^i_{k+1,\pi^i}\big(f_k(x_k, \theta^i, u_k)\big)\Big). \tag{2.103}$$

The computation of $\hat{J}^i_{k+1,\pi^i}\big(f_k(x_k, \theta^i, u_k)\big)$ involves a deterministic propagation from the state x_{k+1} of Eq. (2.102) up to the end of the horizon, using the base policy π^i, while assuming that θ is fixed at the value θ^i.

In particular, the term

$$Q_k(x_k, u_k, \theta^i) = g_k(x_k, \theta^i, u_k) + \hat{J}^i_{k+1,\pi^i}\big(f_k(x_k, \theta^i, u_k)\big) \tag{2.104}$$

appearing on the right side of Eq. (2.103) is viewed as a Q-factor that must be computed for every pair (u_k, θ^i), $u_k \in U_k(x_k)$, $i = 1, \ldots, m$, using the base policy π^i. The expected value of this Q-factor,

$$\hat{Q}_k(x_k, u_k) = \sum_{i=1}^m b_{k,i} Q_k(x_k, u_k, \theta^i),$$

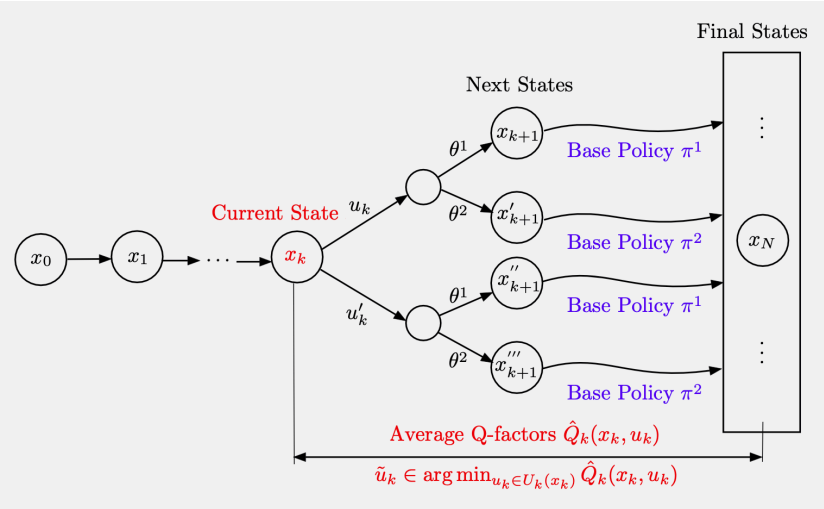

Figure 2.11.2 Schematic illustration of adaptive control by rollout for deterministic systems; cf. Eqs. (2.104) and (2.105). The Q-factors $Q_k(x_k, u_k, \theta^i)$ are averaged over θ^i, using the current belief distribution b_k, and the control applied is the one that minimizes over $u_k \in U_k(x_k)$ the averaged Q-factor

$$\hat{Q}_k(x_k, u_k) = \sum_{i=1}^{m} b_{k,i} Q_k(x_k, u_k, \theta^i).$$

must then be calculated for every $u_k \in U_k(x_k)$, and the computation of the rollout control $\tilde{u}_k$ is obtained from the minimization

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} \hat{Q}_k(x_k, u_k); \tag{2.105}$$

cf. Eq. (2.103). This computation is illustrated in Fig. 2.11.2.

The case of a deterministic system is particularly interesting because we can typically expect that the true parameter θ^* is identified in a finite number of stages, since at each stage k, we are receiving a noiseless measurement relating to θ, namely the state x_k. Once this happens, the problem becomes one of perfect state information.

An illustration similar to the one of Fig. 2.11.2 applies to the rollout scheme (2.101) for the case of a stochastic system. In this case, a Q-factor

$$Q_k(x_k, u_k, \theta^i, w_k) = g_k(x_k, \theta^i, u_k, w_k) + \hat{J}^i_{k+1,\pi^i}\big(f_k(x_k, \theta^i, u_k, w_k)\big)$$

must be calculated for every triplet (u_k, θ^i, w_k), using the base policy π^i. The rollout control $\tilde{u}_k$ is obtained by minimizing the expected value of this Q-factor [averaged using the distribution of (θ, w_k)]; cf. Eq. (2.101).

An interesting and intuitive example that demonstrates the deterministic system case is the popular Wordle puzzle.

Example 2.11.1 (The Wordle Puzzle)

In the classical form of this puzzle, we try to guess a mystery word θ^* out of a known finite collection of 5-letter words. This is done with sequential guesses each of which provides additional information on the correct word θ^*, by using certain given rules to shrink the current mystery list (the smallest list that contains θ^*, based on the currently available information). The objective is to minimize the number of guesses to find θ^* (using more than 6 guesses is considered to be a loss). This type of puzzle descends from the classical family of Mastermind puzzles that centers around decoding a secret sequence of objects (e.g., letters or colors) using partial observations.

The rules for shrinking the mystery list relate to the common letters between the word guesses and the mystery word θ^*, and they will not be described here (there is a large literature regarding the Wordle puzzle). Moreover, θ^* is assumed to be chosen from the initial collection of 5-letter words according to a uniform distribution. Under this assumption, it can be shown that the belief distribution b_k at stage k continues to be uniform over the mystery list. As a result, we may use as state x_k the mystery list at stage k, which evolves deterministically according to an equation of the form (2.102), where u_k is the guess word at stage k. There are several base policies to use in the rollout-like algorithm (2.103), which are described in the papers by Bhambri, Bhattacharjee, and Bertsekas [BBB22], [BBB23], together with computational results, which show that the corresponding rollout algorithm (2.103) performs remarkably close to the optimal policy (first obtained with a very computationally intensive exact DP calculation by Selby in 2022).

The rollout approach also applies to several variations of the Wordle puzzle. Such variations may include for example a larger length $\ell > 5$ of mystery words, and/or a known nonuniform distribution over the initial collection of ℓ-letter words; see [BBB22].

The Case of Sequential Estimation - Alternative Base Policies

We finally note that the adaptive control framework of this section contains as a special case the sequential estimation framework of the preceding section. Here the problem formulation involves a dynamic system of the form

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

where the state x_{k+1} is the observation at time $k+1$ and exhibits no explicit dependence on the preceding observation x_k, but depends on the stochastic disturbance w_k, and on the decision u_k; cf. Figs. 2.11.1 and 2.11.3. This decision may involve a cost and determines the type of next observation out of a collection of possible types.

While the rollout methodology of the present section applies to sequential estimation problems, other rollout algorithms may also be used, depending on the problem's detailed structure. In particular, the rollout algorithms for Bayesian optimization of the works noted in Section 2.10 involve base policies that depend on the current belief state b_k, rather

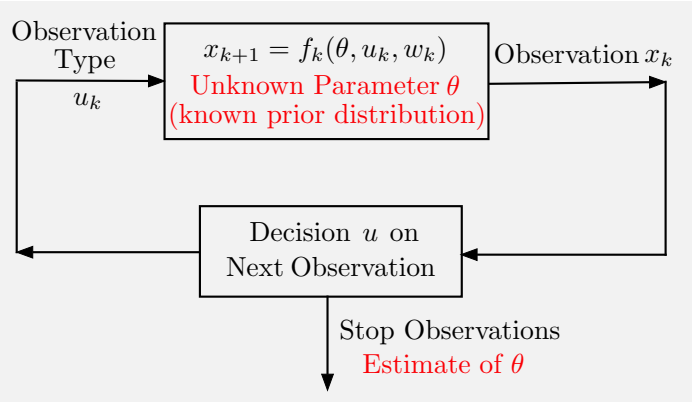

Figure 2.11.3 A view of sequential estimation as an adaptive control problem. The system function f_k does not depend on the current state x_k, so the system provides a decision-dependent noisy observation of θ.

than the current state x_k. Another example of rollout for adaptive control, which uses a base policy that depends on the current belief state is given in Section 6.7 of the book [Ber22a]. For work on related stochastic optimal control problems that involve observation costs and the rollout approach, see Antunes and Heemels [AnH14], and Khashooei, Antunes, and Heemels [KAH15].

2.12 ROLLOUT FOR MINIMAX CONTROL

The problem of optimal control of uncertain systems is usually treated within a stochastic framework, whereby all disturbances $w_0, \ldots, w_{N-1}$ are described by probability distributions, and the expected value of the cost is minimized. However, in many practical situations a stochastic description of the disturbances may not be available, but one may have information with less detailed structure, such as bounds on their magnitude. In other words, one may know a set within which the disturbances are known to lie, but may not know the corresponding probability distribution. Under these circumstances one may use a minimax approach, whereby the worst possible values of the disturbances within the given set are assumed to occur. Within this context, we take the view that the disturbances are chosen by an antagonistic opponent. The minimax approach is also connected with two-player games, when in lack of information about the opponent, we adopt a worst case viewpoint during on-line play, as well as with contexts where we wish to guard against adversarial attacks.†

† The minimax approach to decision and control has its origins in the 50s and 60s. It is also referred to by other names, depending on the underlying

To be specific, consider a finite horizon context, and assume that the disturbances $w_0, w_1, \ldots, w_{N-1}$ do not have a probabilistic description but rather are known to belong to corresponding given sets $W_k(x_k, u_k) \subset D_k$, $k = 0, 1, \ldots, N-1$, which may depend on the current state x_k and control u_k. The minimax control problem is to find a policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ with $\mu_k(x_k) \in U_k(x_k)$ for all x_k and k, which minimizes the cost function

$$J_\pi(x_0) = \max_{\substack{w_k \in W_k(x_k, \mu_k(x_k)) \\ k=0,1,\ldots,N-1}} \left[g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), w_k\big) \right].$$

The DP algorithm for this problem takes the following form, which resembles the one corresponding to the stochastic DP problem (maximization is used in place of expectation):

$$J_N^*(x_N) = g_N(x_N), \tag{2.106}$$

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \max_{w_k \in W_k(x_k, u_k)} \Big[g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big) \Big]. \tag{2.107}$$

This algorithm can be explained by using a principle of optimality type of argument. In particular, we consider the tail subproblem whereby we are at state x_k at time k, and we wish to minimize the "cost-to-go"

$$\max_{\substack{w_t \in W_t(x_t, \mu_t(x_t)) \\ t=k,k+1,\ldots,N-1}} \left[g_N(x_N) + \sum_{t=k}^{N-1} g_t\big(x_t, \mu_t(x_t), w_t\big) \right].$$

We argue that if $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$ is an optimal policy for the minimax problem, then the tail of the policy $\{\mu_k^*, \mu_{k+1}^*, \ldots, \mu_{N-1}^*\}$ is optimal for the tail subproblem. The optimal cost of this subproblem is $J_k^*(x_k)$, as given by the DP algorithm (2.106)-(2.107). The algorithm expresses the intuitive fact that when at state x_k at time k, then regardless of what happened in the past, we should choose u_k that minimizes the worst/maximum value over w_k of the sum of the current stage cost plus the optimal cost of the tail subproblem that starts from the next state. This argument requires a mathematical proof, which turns out to involve a few fine points. For a detailed mathematical derivation, we refer to the author's textbook [Ber17a], Section 1.6. However, the DP algorithm (2.106)-(2.107) is correct assuming finite state and control spaces, among other cases.

---

context, such as *robust control*, *robust optimization*, *control with a set membership description of the uncertainty*, and *games against nature*. In this book, we will be using the minimax control name.

Approximation in Value Space and Minimax Rollout

The approximation ideas for stochastic optimal control are also relevant within the minimax context. In particular, approximation in value space with one-step lookahead applies at state x_k a control

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} \max_{w_k \in W_k(x_k,u_k)} \Big[g_k(x_k,u_k,w_k) + \tilde{J}_{k+1}\big(f_k(x_k,u_k,w_k)\big) \Big], \tag{2.108}$$

where $\tilde{J}_{k+1}(x_{k+1})$ is an approximation to the optimal cost-to-go $J^*_{k+1}(x_{k+1})$ from state x_{k+1}.

Rollout is obtained when this approximation is the tail cost of some base policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$:

$$\tilde{J}_{k+1}(x_{k+1}) = J_{k+1,\pi}(x_{k+1}).$$

Given π, we can compute $J_{k+1,\pi}(x_{k+1})$ by solving a *deterministic maximization* DP problem with the disturbances $w_{k+1}, \ldots, w_{N-1}$ playing the role of "optimization variables/controls." For finite state, control, and disturbance spaces, this is a longest path problem defined on an acyclic graph, since the control variables $u_{k+1}, \ldots, u_{N-1}$ are determined by the base policy. It is then straightforward to implement rollout: at x_k we generate all next states of the form

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

corresponding to all possible values of $u_k \in U_k(x_k)$ and $w_k \in W_k(x_k, u_k)$. We then run the maximization/longest path problem described above to compute $\tilde{J}_{k+1}(x_{k+1})$ from each of these possible next states x_{k+1}. Finally, we obtain the rollout control $\tilde{u}_k$ by solving the minimax problem in Eq. (2.108). Moreover, it is possible to use truncated rollout to approximate the tail cost of the base policy.†

Note that like all rollout algorithms, the minimax rollout algorithm is well-suited for on-line replanning in problems where data may be changing or may be revealed during the process of control selection.

We mentioned earlier that deterministic problems allow a more general form of rollout, whereby we may use a base heuristic that need not be a legitimate policy, i.e., it need not be sequentially consistent. For cost improvement it is sufficient that the heuristic be sequentially improving. A similarly more general view of rollout is not easily constructed for stochastic problems, but is possible for minimax control.

In particular, suppose that at any state x_k there is a heuristic that generates a sequence of feasible controls and disturbances, and corresponding states,

$$\{u_k, w_k, x_{k+1}, u_{k+1}, w_{k+1}, x_{k+2}, \ldots, u_{N-1}, w_{N-1}, x_N\},$$

† For a more detailed discussion of this implementation, see the author's paper [Ber19b] (Section 5.4).

with corresponding cost

$$H_k(x_k) = g_k(x_k, u_k, w_k) + \cdots + g_{N-1}(x_{N-1}, u_{N-1}, w_{N-1}) + g_N(x_N).$$

Then the rollout algorithm applies at state x_k a control

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(x_k)} \max_{w_k \in W_k(x_k, u_k)} \Big[g_k(x_k, u_k, w_k) + H_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big].$$

This does not preclude the possibility that the disturbances $w_k, \ldots, w_{N-1}$ are chosen by an antagonistic opponent, but allows more general choices of disturbances, obtained for example, by some form of approximate maximization. For example, when the disturbance involves multiple components, $w_k = (w_k^1, \ldots, w_k^m)$, corresponding to multiple opponent agents, *the heuristic may involve an agent-by-agent maximization strategy*.

The sequential improvement condition, similar to the deterministic case, is that for all x_k and k,

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

It guarantees cost improvement, i.e., that for all x_k and k, the rollout policy

$$\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$$

satisfies

$$J_{k,\tilde{\pi}}(x_k) \leq H_k(x_k).$$

Thus, generally speaking, minimax rollout is fairly similar to rollout for deterministic as well as stochastic DP problems. The main difference with deterministic (or stochastic) problems is that to compute the Q-factor of a control u_k, we need to solve a maximization problem, rather than carry out a deterministic (or Monte-Carlo, respectively) simulation with the given base policy.

Example 2.12.1 (Pursuit-Evasion Problems)

Consider a pursuit-evasion problem with state $x_k = (x_k^1, x_k^2)$, where x_k^1 is the location of the minimizer/pursuer and x_k^2 is the location of the maximizer/evader, at stage k, in a (finite node) graph defined in two- or three-dimensional space. There is also a cost-free and absorbing termination state that consists of a subset of pairs (x^1, x^2) that includes all pairs with $x^1 = x^2$. The pursuer chooses one out of a finite number of actions $u_k \in U_k(x_k)$ at each stage k, when at state x_k, and if the state is x_k and the pursuer selects u_k, the evader may choose from a known set $X_{k+1}(x_k, u_k)$ of next states x_{k+1}, which depends on (x_k, u_k). The objective of the pursuer is to minimize a nonnegative terminal cost $g(x_N^1, x_N^2)$ at the end of N stages (or reach the

termination state, which has cost 0 by assumption). A reasonable base policy for the pursuer can be precomputed by DP as follows: given the current (nontermination) state $x_k = (x_k^1, x_k^2)$, make a move along the path that starts from x_k^1 and minimizes the terminal cost after $N - k$ stages, under the assumption that the evader will stay motionless at his current location x_k^2. (In a variation of this policy, the DP computation is done under the assumption that the evader will follow some nominal sequence of moves.)

For the on-line computation of the rollout control, we need the maximal value of the terminal cost that the evader can achieve starting from every $x_{k+1} \in X_{k+1}(x_k, u_k)$, assuming that the pursuer will follow the base policy (which has already been computed). We denote this maximal value by $\tilde{J}_{k+1}(x_{k+1})$. The required values $\tilde{J}_{k+1}(x_{k+1})$ can be computed by an $(N-k)$-stage DP computation involving the optimal choices of the evader, while assuming the pursuer uses the (already computed) base policy. Then the rollout control for the pursuer is obtained from the minimization

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} \max_{x_{k+1} \in X_{k+1}(x_k, u_k)} \tilde{J}_{k+1}(x_{k+1}).$$

Note that the preceding algorithm can be adapted for the imperfect information case where the pursuer knows x_k^2 imperfectly. This is possible by using a form of assumed certainty equivalence: the pursuer's base policy and the evader's maximization can be computed by using an estimate of the current location x_k^2 instead of the unknown true location.

In the preceding pursuit-evasion example, the choice of the base policy was facilitated by the special structure of the problem. Generally, however, finding a suitable base policy that can be conveniently implemented is an important problem-dependent issue.

Variants of Minimax Rollout

Several of the variants of rollout discussed earlier have analogs in the minimax context, e.g., truncation with terminal cost approximation, multistep and selective step lookahead, and multiagent rollout. In particular, in the ℓ-step lookahead variant, we solve the ℓ-stage problem

$$\min_{u_k, \mu_{k+1}, \ldots, \mu_{k+\ell-1}} \max_{\substack{w_k \in W_k(x_k, u_k) \\ w_t \in W_t(x_t, \mu_t(x_t)) \\ t=k+1, \ldots, N-1}} \left\{ g_k(x_k, u_k, w_k) + \sum_{t=k+1}^{k+\ell-1} g_t\big(x_t, \mu_t(x_t), w_t\big) + H_{k+\ell}(x_{k+\ell}) \right\},$$

we find an optimal solution $\tilde{u}_k, \tilde{\mu}_{k+1}, \ldots, \tilde{\mu}_{k+\ell-1}$, and we apply the first component $\tilde{u}_k$ of that solution. As an example, this type of problem is solved at each move of chess programs like AlphaZero, where the terminal cost function is encoded through a position evaluator. In fact when multistep lookahead is used, special techniques such as *alpha-beta pruning* may

be used to accelerate the computations by eliminating unnecessary portions of the lookahead graph. These techniques are well-known in the context of the two-person computer game methodology, and are used widely in games such as chess.

It is interesting to note that, contrary to the case of stochastic optimal control, there is an on-line *constrained form of rollout* for minimax control. Here there are some additional trajectory constraints of the form

$$(x_0, u_0, \ldots, u_{N-1}, x_N) \in C,$$

where C is an arbitrary set. The modification needed is similar to the one of Section 6.6: at partial trajectory

$$\tilde{y}_k = (\tilde{x}_0, \tilde{u}_0, \ldots, \tilde{u}_{k-1}, \tilde{x}_k),$$

generated by rollout, we use a heuristic with cost function H_{k+1} to compute the Q-factor

$$\tilde{Q}_k(\tilde{x}_k, u_k) = \max_{w_k, \ldots, w_{N-1}} \Big[g_k(\tilde{x}_k, u_k, w_k) + H_{k+1}\big(f_k(\tilde{x}_k, u_k, w_k), w_{k+1}, \ldots, w_{N-1}\big) \Big]$$

for each u_k in the set $\tilde{U}_k(\tilde{y}_k)$ that guarantee feasibility [we can check feasibility here by running some algorithm that verifies whether the future disturbances $w_k, \ldots, w_{N-1}$ can be chosen to violate the constraint under the base policy, starting from $(\tilde{y}_k, u_k)$]. Once the set of "feasible controls" $\tilde{U}_k(\tilde{y}_k)$ is computed, we can obtain the rollout control by the Q-factor minimization:

$$\tilde{u}_k \in \arg \min_{u_k \in \tilde{U}_k(\tilde{y}_k)} \tilde{Q}_k(\tilde{x}_k, u_k).$$

We may also use fortified versions of the unconstrained and constrained rollout algorithms, which guarantee a feasible cost-improved rollout policy. This requires the assumption that the base heuristic at the initial state produces a trajectory that is feasible for all possible disturbance sequences. Similar to the deterministic case, there are also truncated and multiagent versions of the minimax rollout algorithm.

Example 2.12.2 (Multiagent Minimax Rollout)

Let us consider a minimax problem where the minimizer's choice involves the collective decision of m agents, $u = (u^1, \ldots, u^m)$, with u^ℓ corresponding to agent ℓ, and constrained to lie within a finite set U^ℓ. Thus u must be chosen from within the set

$$U = U^1 \times \ldots \times U^m,$$

which is finite but grows exponentially in size with m. The maximizer's choice w is constrained to belong to a finite set W. We consider multiagent

rollout for the minimizer, and for simplicity, we focus on a two-stage problem. However, there are straightforward extensions to a more general multistage framework.

In particular, we assume that the minimizer knowing an initial state x_0, chooses $u = (u^1, \ldots, u^m)$, with $u^\ell \in U^\ell$, $\ell = 1, \ldots, m$, and a state transition

$$x_1 = f_0(x_0, u)$$

occurs with cost $g_0(x_0, u)$. Then the maximizer, knowing x_1, chooses $w \in W$, and a terminal state

$$x_2 = f_1(x_1, w)$$

is generated with cost

$$g_1(x_1, w) + g_2(x_2).$$

The problem is to select $u \in U$, to minimize

$$g_0(x_0, u) + \max_{w \in W} \big[g(x_1, w) + g_2(x_2)\big].$$

The exact DP algorithm for this problem is given by

$$J_1^*(x_1) = \max_{w \in W} \Big[g_1(x_1, w) + g_2\big(f_1(x_1, w)\big)\Big],$$

$$J_0^*(x_0) = \min_{u \in U} \Big[g_0(x_0, u) + J_1^*\big(f_0(x_0, u)\big)\Big].$$

This DP algorithm is computationally intractable for large m. The reason is that the set of possible minimizer choices u grows exponentially with m, and for each of these choices the value of $J_1^*\big(f_0(x_0, u)\big)$ must be computed.

However, the problem can be solved approximately with multiagent rollout, using a base policy $\mu = (\mu^1, \ldots, \mu^m)$. Then the number of times $J_1^*\big(f_0(x_0, u)\big)$ needs to be computed is dramatically reduced. This computation is done sequentially, one-agent-at-a-time, as follows:

$$\tilde{u}^1 \in \arg \min_{u^1 \in U^1} \Big[g_0\big(x_0, u^1, \mu^2(x_0), \ldots, \mu^m(x_0)\big) + J_1^*\Big(f_0\big(x_0, u^1, \mu^2(x_0), \ldots, \mu^m(x_0)\big)\Big)\Big],$$

$$\tilde{u}^2 \in \arg \min_{u^2 \in U^2} \Big[g_0\big(x, \tilde{u}^1, u^2, \mu^3(x_0), \ldots, \mu^m(x_0)\big) + J_1^*\Big(f_0\big(x_0, \tilde{u}^1, u^2, \mu^3(x_0), \ldots, \mu^m(x_0)\big)\Big)\Big],$$

$$\cdots \qquad \cdots \qquad \cdots \qquad \cdots$$

$$\tilde{u}^m \in \arg \min_{u^m \in U^m} \Big[g_0\big(x_0, \tilde{u}^1, \tilde{u}^2, \ldots, \tilde{u}^{m-1}, u^m\big) + J_1^*\Big(f_0\big(x_0, \tilde{u}^1, \tilde{u}^2, \ldots, \tilde{u}^{m-1}, u^m\big)\Big)\Big].$$

In this algorithm, the number of times for which $J_1^*\big(f_0(x_0,u)\big)$ must be computed grows linearly with m.

When the number of stages is larger than two, a similar algorithm can be used. Essentially, the one-stage maximizer's cost function J_1^* must be replaced by the optimal cost function of a multistage maximization problem, where the minimizer is constrained to use the base policy (see also the paper [Ber19b], Section 5.4).

An interesting question is how do various algorithms work when approximations are used in the min-max and max-min problems? We can certainly improve the minimizer's policy *assuming a fixed policy for the maximizer*. However, it is unclear how to improve both the minimizer's and the maximizer's policies simultaneously. In practice, in *symmetric games*, like chess, a common policy is trained for both players. In particular, in the AlphaZero and TD-Gammon programs this strategy is computationally expedient and has worked well. However, there is no reliable theory to guide the simultaneous training of policies for both maximizer and minimizer, and it is quite plausible that unusual behavior may arise in exceptional cases.† Even *exact* policy iteration methods for Markov games encounter serious convergence difficulties, and need to be modified for reliable behavior. The author's paper [Ber21c] and book [Ber22b] (Chapter 5) address these convergence issues with modified versions of the policy iteration method, and give many earlier references.

We finally note another source of difficulty in minimax control: Newton's method applied to solution of the Bellman equation for minimax problems exhibits more complex behavior than its expected value counterpart. The reason is that the Bellman operator T for infinite horizon problems, given by

$$(TJ)(x) = \min_{u\in U(x)} \max_{w\in W(x,u)} \Big[g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big], \qquad \text{for all } x,$$

is neither convex nor concave as a function of J. To see this, note that the function

$$\max_{w\in W(x,u)} \Big[g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big],$$

viewed as a function of J [for fixed (x,u)], is convex, and when minimized over $u \in U(x)$, it becomes neither convex nor concave. As a result there are special difficulties in connection with convergence of Newton's method and the natural form of policy iteration, given by Pollatschek and Avi-Itzhak [PoA69]; see also Chapter 5 of the author's abstract DP book [Ber22a].

† Indeed such exceptional cases have been reported for the AlphaGo program in late 2022, when humans defeated an AlphaGo look-alike, KataGo, "by using adversarial techniques that take advantage of KataGo's blind spots" (according to the reports); see Wang et al. [WGB22].

Minimax Control and Zero-Sum Game Theory

Zero-sum game problems are viewed as fundamental in the field of economics, and there is an extensive and time-honored theory around them. In the case where the game involves a dynamic system

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

and a cost function

$$g_k(x_k, u_k, w_k),$$

there are two players, the minimizer choosing $u_k \in U_k(x_k)$, and the maximizer choosing $w_k \in W_k(x_k)$, at each stage k. Such zero-sum games involve *two* minimax control problems:

(a) The *min-max problem*, where the minimizer chooses a policy first and the maximizer chooses a policy second with knowledge of the minimizer's policy. The DP algorithm for this problem has the form

$$J_N^*(x_N) = g_N(x_N),$$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \max_{w_k \in W_k(x_k)} \Big[g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big) \Big].$$

(b) The *max-min problem*, where the maximizer chooses policy first and the minimizer chooses policy second with knowledge of the maximizer's policy. The DP algorithm for this problem has the form

$$\hat{J}_N(x_N) = g_N(x_N),$$

$$\hat{J}_k(x_k) = \max_{w_k \in W_k(x_k)} \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k, w_k) + \hat{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big].$$

A basic and easily seen fact is that

$$\text{Max-Min optimal value} \leq \text{Min-Max optimal value}.$$

Game theory is particularly interested on conditions that guarantee that

$$\text{Max-Min optimal value} = \text{Min-Max optimal value}. \tag{2.109}$$

However, this question is of limited interest in engineering contexts that involve worst case design. Moreover, the validity of the minimax equality (2.109) is beyond the range of practical RL. This is so primarily because once approximations are introduced, the delicate assumptions that guarantee this equality are disrupted.

2.13 NOTES, SOURCES, AND EXERCISES

Section 2.1: In this chapter, we have placed emphasis on finite horizon problems, possibly involving a nonstationary system and cost per stage. However, the insights that can be obtained from the infinite horizon/stationary context fully apply. These include the interpretation of approximation in value space as a Newton step, and of rollout as a single step of the policy iteration method. The reason is that an N-step finite horizon/nonstationary problem can be converted to an infinite horizon/stationary problem with a termination state to which the system moves at the Nth stage; see Section 1.6.2.

Section 2.2: Approximation in value space has been considered in an ad hoc manner since the early days of DP, motivated by the curse of dimensionality. Moreover, the idea of ℓ-step lookahead minimization with horizon truncation beyond the ℓ steps has a long history and is often referred to as "rolling horizon" or "receding horizon" optimization. Approximation in value space was reframed in the late 80s and was coupled with model-free simulation methods that originated in artificial intelligence.

Section 2.3: The main idea of rollout algorithms, obtaining an improved policy starting from some other suboptimal policy, has appeared in several DP contexts, including games; see e.g., Abramson [Abr90], and Tesauro and Galperin [TeG96]. The name "rollout" was coined by Tesauro [TeG96] in the context of backgammon; see Example 2.7.3. The use of the name "rollout" has gradually expanded beyond its original context; for example the samples collected through trajectory simulation are referred to as "rollouts" by some authors.

In this book, we will adopt the original intended meaning: rollout is an algorithm that provides policy improvement starting from a base policy, which is evaluated with some form of Monte Carlo simulation, perhaps augmented by some other calculation that may include a terminal cost function approximation. The author's rollout book [Ber20a] provides a more extensive discussion of rollout algorithms and their applications.

There has been a lot of research on rollout algorithms, which we list selectively in chronological order: Christodouleas [Chr97], Bertsekas and Castañon [BeC99], Duin and Voss [DuV99], Secomandi [Sec00], [Sec01], [Sec03], Ferris and Voelker [FeV02], [FeV04], McGovern, Moss, and Barto [MMB02], Savagaonkar, Givan, and Chong [SGC02], Wu, Chong, and Givan [WCG02], [WCG03], Bertsimas and Popescu [BeP03], Guerriero and Mancini [GuM03], Tu and Pattipati [TuP03], Meloni, Pacciarelli, and Pranzo [MPP04], Yan et al. [YDR04], Han, Lai, and Spivakovsky [HLS06], Besse and Chaib-draa [BeC08], Novoa and Storer [NoS09], Sun et al. [SZL08], Mishra et al. [MCT10], Bertazzi et al. [BBG13], Sun et al. [SLJ13], Tesauro et al. [TGL13], Antunes and Heemels [AnH14], Beyme and Leung [BeL14], Goodson, Thomas, and Ohlmann [GTO15], [GTO17], Khashooei, Antunes,

and Heemels [KAH15], Li and Womer [LiW15], Mastin and Jaillet [MaJ15], Huang, Jia, and Guan [HJG16], Simroth, Holfeld, and Brunsch [SHB15], Lan, Guan, and Wu [LGW16], Lam, Willcox, and Wolpert [LWW16], Gommans et al. [GTA17], Lam and Willcox [LaW17], Ulmer [Ulm17], Bertazzi and Secomandi [BeS18], Zhang, Ohlmann, and Thomas [ZOT18], Sarkale et al. [SNC18], Ulmer at al. [UGM18], Arcari, Hewing, and Zeilinger [AHZ19], Chu, Xu, and Li [CXL19], Goodson, Bertazzi, and Levary [GBL19], Guerriero, Di Puglia, and Macrina [GDM19], Ho, Liu, and Zabinsky [HLZ19], Liu et al. [LLL19], Nozhati et al. [NSE19], Singh and Kumar [SiK19], Yu et al. [YYM19], Yuanhong [Yua19], Andersen, Stidsen, and Reinhardt [ASR20], Durasevic and Jakobovic [DuJ20], Issakkimuthu, Fern, and Tadepalli [IFT20], Lee et al. [LEC20], Li et al. [LZS20], Lee [Lee20], Montenegro et al. [MLM20], Meshram and Kaza [MeK20], Schope, Driessen, and Yarovoy [SDY20], Yan, Wang, and Xu [YWX20], Yue and Kontar [YuK20], Zhang, Kafouros, and Yu [ZKY20], Hoffman et al. [HCR21], Houy and Flaig [HoF21], Li, Krakow, and Gopalswamy [LKG21], Liu et al. [LPS21], Nozhati [Noz21], Rimélé et al. [RGG21], Tuncel et al. [TBP21], Xie, Li, and Xu [XLX21], Bertsekas [Ber22d], Paulson, Sonouifar, and Chakrabarty [PSC22], Bai et al. [BLJ23], Rusmevichientong et al. [RST23], Yilmaz, Xiang, and Klein [YXK24].

These references collectively include a large number of computational studies, discuss variants and problem-specific adaptations of rollout algorithms for a broad variety of practical problems, and consistently report favorable computational experience. The size of the cost improvement over the base policy is often impressive, evidently owing to the fast convergence rate of Newton's method that underlies rollout. Moreover these works illustrate some of the other important advantages of rollout: reliability, simplicity, suitability for on-line replanning, and the ability to interface with other RL techniques, such as neural network training, which can be used to provide suitable base policies and/or approximations to their cost functions.

The adaptation of rollout algorithms to discrete deterministic optimization problems, the notions of sequential consistency, sequential improvement, fortified rollout, and the use of multiple heuristics for parallel rollout were first given in the paper by Bertsekas, Tsitsiklis, and Wu [BTW97], and were also discussed in the neuro-dynamic programming book [BeT96]. Rollout algorithms for stochastic problems were further formalized in the papers by Bertsekas [Ber97b], and Bertsekas and Castañon [BeC99]. Extensions to constrained rollout were first given in the author's papers [Ber05a], [Ber05b]. A survey of rollout in discrete optimization was given by the author in [Ber13a].

The model-free rollout algorithm, in the form given here, was first discussed in the RL book [Ber19a]. It is inspired by the method of comparison training, proposed by Tesauro [Tes89a], [Tes89b], [Tes01], and subsequently used by several other authors (see [DNW16], [TCW19]). This is a general

method for training an approximation architecture to choose between two alternatives, using a dataset of expert choices in place of an explicit cost function.

The material on most likely sequence generation for n-grams, HMMs, and Markov Chains is recent, and was developed in the paper by Li and Bertsekas [LiB24].

Section 2.4: Our discussion of rollout, iterative deepening, and pruning in the context of multistep approximation in value space for deterministic problems contains some original ideas. In particular, the incremental multistep rollout algorithm and variations of Section 2.4.2 are presented here for the first time.

Note also that the multistep lookahead approximations described in Section 2.4 can be used more broadly within algorithms that employ forms of multistep lookahead search as subroutines. In particular, local search algorithms, such as tabu search, genetic algorithms, and others, which are commonly used for discrete and combinatorial optimization, may be modified along the lines of Section 2.4 to incorporate RL and approximate DP ideas.

Section 2.5: Constrained forms of rollout were introduced in the author's papers [Ber05a] and [Ber05b]. The paper [Ber05a] also discusses rollout and approximation in value space for stochastic problems in the context of so-called *restricted structure policies*. The idea here is to simplify the problem by selectively restricting the information and/or the controls available to the controller, thereby obtaining a restricted but more tractable problem structure, which can be used conveniently in a one-step lookahead context. An example of such a structure is one where fewer observations are obtained, or one where the control constraint set is restricted to a single or a small number of given controls at each state.

Section 2.6: Rollout for continuous-time optimal control was first discussed in the author's rollout book [Ber20a]. A related discussion of policy iteration, including the motivation for approximating the gradient of the optimal cost-to-go $\nabla_x J_t$ rather than the optimal cost-to-go J_t, has been given in Section 6.11 of the neuro-dynamic programming book [BeT96]. This discussion also includes the use of value and policy networks for approximate policy evaluation and policy improvement for continuous-time optimal control. The underlying ideas have long historical roots, which are recounted in detail in the book [BeT96].

Section 2.7: The idea of the certainty equivalence approximation in the context of rollout for stochastic systems (Section 2.7.3) was proposed in the paper by Bertsekas and Castañon [BeC99], together with extensive empirical justification. However, the associated theoretical insight into this idea was established more recently, through the interpretation of approximation in value space as a Newton step, which suggests that the lookahead min-

imization after the first step can be approximated with small degradation of performance.

The idea of variance reduction in the context of rollout (Section 2.7.4) was proposed by the author in the paper [Ber97b]. See also the DP textbook [Ber17a], Section 6.5.2.

The paper by Chang, Hu, Fu, and Marcus [CHF05], and the 2007 first edition of their monograph proposed and analyzed adaptive sampling in connection with DP, as well as early forms of Monte Carlo tree search, including statistical tests to control the sampling process (a second edition, [CHF13], appeared in 2013). The name "Monte Carlo tree search" has become popular, and in its current use, it encompasses a variety of methods that involve adaptive sampling, rollout, and extensions to sequential games. We refer to the papers by Coulom [Cou06], and Chang et al. [CHF13], the discussion by Fu [Fu17], and the survey by Browne et al. [BPW12].

Statistical tests for adaptive sampling has been inspired by works on multiarmed bandit problems; see Lai and Robbins [LaR85], Agrawal [Agr95], Burnetas and Katehakis [BuK97], Meuleau and Bourgine [MeB99], Auer, Cesa-Bianchi, and Fischer [ACF02], Kocsis and Szepesvari [KoS06], Dimitrakakis and Lagoudakis [DiL08], Audibert, Munos, and Szepesvari [AMS09], and Munos [Mun14]. The book by Lattimore and Szepesvari [LaS20] focuses on multiarmed bandit methods, and provides an extensive account of the UCB rule. For recent work on the theoretical properties of the UCB and UCT rules, see Shah, Xie, and Xu [SXX22], and Chang [Cha24].

Adaptive sampling and MCTS may be viewed within the context of a broader class of on-line lookahead minimization techniques, sometimes called *on-line search* methods. These techniques are based on a variety of ideas, such as random search and intelligent pruning of the lookahead tree. One may naturally combine them with approximation in value space and (possibly) rollout, although it is not necessary to do so (the multistep minimization horizon may extend to the terminal time N). For representative works, some of which apply to continuous spaces problems, including POMDP, see Hansen and Zilberstein [HaZ01], Kearns, Mansour, and Ng [KMN02], Peret and Garcia [PeG04], Ross et al. [RPP08], Silver and Veness [SiV10], Hostetler, Fern, and Dietterich [HFD17], and Ye et al. [YSH17]. The multistep lookahead approximation ideas of Section 2.4 may also be viewed within the context of on-line search methods.

Another rollout idea for stochastic problems, which we have not discussed in this book, is the *open-loop feedback controller* (OLFC), a suboptimal control scheme that dates to the 60s; see Dreyfus [Dre65]. The OLFC applies to POMDP as well, and uses an open-loop optimization over the future evolution of the system. In particular, it uses the current information vector I_k to determine the belief state b_k. It then solves the open-loop

problem of minimizing

$$E\left\{g_N(x_N)+\sum_{i=k}^{N-1}g_i(x_i,u_i,w_i)\;\Big|\;I_k\right\}$$

subject to the constraints

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

and applies the first control $\overline{u}_k$ in the optimal open-loop control sequence $\{\overline{u}_k,\overline{u}_{k+1},\ldots,\overline{u}_{N-1}\}$. It is easily seen that the OLFC is a rollout algorithm that uses as base policy the optimal open-loop policy for the problem (the one that ignores any state or observation feedback).

For a detailed discussion of the OLFC, we refer to the author's survey paper [Ber05a] (Section 4) and DP textbook [Ber17a] (Section 6.4.4). The survey [Ber05a] discusses also a generalization of the OLFC, called *partial open-loop-feedback-control*, which calculates the control input on the basis that *some* (but not necessarily all) of the observations will in fact be taken in the future, and the remaining observations will not be taken. This method often allows one to deal with those observations that are troublesome and complicate the solution, while taking into account the future availability of other observations that can be reasonably dealt with. A computational case study for hydrothermal power system scheduling is given by Martinez and Soares [MaS02]. A variant of the OLFC, which also applies to minimax control problems, is given in the author's paper [Ber72b], together with a proof of a cost improvement property over the optimal open-loop policy.

Section 2.8: The role of stochastic programming in providing a link between stochastic DP and continuous spaces deterministic optimization (cf. Section 2.8) is well known; see the texts by Birge and Louveaux [BiL97], Kall and Wallace [KaW94], and Prekopa [Pre95], and the survey by Ruszczynski and Shapiro [RuS03]. Stochastic programming has been applied widely, and there is much to be gained from its combination with RL. The material of this section comes from the author's rollout book [Ber20a], Section 2.5.2. For a computational study that has tested the ideas of this section on a problem of maintenance scheduling, see Hu et al. [HWP22].

Section 2.9: The multiagent rollout algorithm was proposed in the author's papers [Ber19c], [Ber20b]. The paper [Ber21a] provides an extensive overview of this research. See also the notes and sources for Chapter 1.

Section 2.10: The material on rollout for Bayesian optimization and sequential estimation comes from a recent paper by the author [Ber22d]. This paper is also the basis for the adaptive control material of Section 2.11, and has been included in the book [Ber22a]. The paper by Bhambri, Bhattacharjee, and Bertsekas [BBB22] discusses this material for the

case of a deterministic system, applies rollout to sequential decoding in the context of the challenging Wordle puzzle, and provides an implementation using some popular base heuristics, with performance that is very close to optimal. For related work see Loxley and Cheung [LoC23].

Section 2.11: The POMDP framework for adaptive control dates to the 60s, and has stimulated substantial theoretical investigations; see Mandl [Man74], Borkar and Varaiya [BoV79], Doshi and Shreve [DoS80], Kumar and Lin [KuL82], and the survey by Kumar [Kum85]. Some of the pitfalls of performing parameter estimation while simultaneously applying adaptive control have been described by Borkar and Varaiya [BoV79], and by Kumar [Kum83]; see [Ber17a], Section 6.8 for a related discussion.

The papers just mentioned have proposed on-line estimation of the unknown parameter θ and the use at each time period of a policy that is optimal for the current estimate. The papers provide nontrivial analyses that assert asymptotic optimality of the resulting adaptive control schemes under appropriate conditions. If parameter estimation schemes are similarly used in conjunction with rollout, as suggested in Section 2.11 [cf. Eq. (2.101)], one may conjecture that an asymptotic cost improvement property can be proved for the rollout policy, again under appropriate conditions.

Section 2.12: The treatment of sequential minimax problems by DP (cf. Section 2.12) has a long history. For some early influential works, see Blackwell and Girshick [BlG54], Shapley [Sha53], and Witsenhausen [Wit66]. In minimax control problems, the maximizer is assumed to make choices with perfect knowledge of the minimizer's policy. If the roles of maximizer and minimizer are reversed, i.e., the maximizer has a policy (a sequence of functions of the current state) and the minimizer makes choices with perfect knowledge of that policy, the minimizer gains an advantage, the problem may genuinely change, and the optimal value may be reduced. Thus "min-max" and "max-min" are generally two different problems. In classical two-person zero-sum game theory, however, the main focus is on situations where the min-max and max-min are equal. By contrast, in engineering worst case design contexts, the min-max and max-min values are typically unequal.

There is substantial literature on sequential zero-sum games in the context of DP, often called *Markov games*. The classical paper by Shapley [Sha53] addresses discounted infinite horizon games. A PI algorithm for finite-state Markov games was proposed by Pollatschek and Avi-Itzhak [PoA69], and was interpreted as a Newton method for solving the associated Bellman equation. They have also shown that the algorithm may not converge to the optimal cost function. Computational studies have verified that the Pollatschek and Avi-Itzhak algorithm converges much faster than its competitors, *when it converges* (see Breton et al. [BFH86], and also Filar and Tolwinski [FiT91], who proposed a modification of the algorithm). Related methods have been discussed for Markov games by van der Wal

[Van78] and Tolwinski [Tol89]. The paper by Raghavan and Filar [RaF91], and the textbook by Filar and Vrieze [FiV96] provide extensive surveys of the research up to that time.

The author's paper [Ber21b] has explained the reason behind the unreliable behavior of the Pollatschek and Avi-Itzhak algorithm. This explanation relies on the Newton step interpretation of PI given in Chapter 1: in the case of Markov games, the Bellman operator does not have the concavity property that is typical of one-player games. The paper [Ber21b] has also provided a modified algorithm with solid convergence properties under a totally asynchronous implementation, which applies to very general types of sequential zero-sum games and minimax control. Related aggregation-based RL algorithms were also given. The algorithms, variations, and analysis of the paper [Ber21b] were incorporated as Chapter 5 in the 3rd edition of the author's abstract DP book [Ber22b].

The paper by Yu [Yu14] provides an analysis of stochastic shortest path games, where the termination state may not be reachable under some policies, following the earlier paper by Patek and Bertsekas [PaB99]. The paper [Yu14] also includes a rigorous analysis of the Q-learning algorithm for stochastic shortest path games (without any cost function approximation). The papers by Perolat et al. [PSP15], [PPG16], and the survey by Zhang, Yang, and Basar [ZYB21] discuss alternative RL methods for games. The author's paper [Ber19b] develops VI, PI, and Dijkstra-like finitely terminating algorithms for exact solution of shortest path minimax problems. It also discusses related rollout algorithms for approximate solution.

E X E R C I S E S

2.1 (A Traveling Salesman Rollout Example with a Sequentially Improving Heuristic)

Consider the traveling salesman problem of Example 1.2.3 and Fig. 1.2.11, and the rollout algorithm starting from city A.

(a) Assume that the base heuristic is chosen to be the farthest neighbor heuristic, which completes a partial tour by successively moving to the farthest neighbor city not visited thus far. Show that this base heuristic is sequentially consistent. What are the tours produced by this base heuristic and the corresponding rollout algorithm? *Answer*: The base heuristic will produce the tour A→AD→ADB→ADBC→A with cost 45. The rollout algorithm will produce the tour A→AB→ABD→ABDC→A with cost 13.

(b) Assume that the base heuristic at city A is the nearest neighbor heuristic, while at the partial tours AB, AC, and AD it is the farthest neighbor heuristic. Show that this base heuristic is sequentially improving but not sequentially consistent. Compute the final tour generated by rollout.

Solution of part (b): Clearly the base heuristic is not sequentially consistent, since from A it generates

$$\text{A}\rightarrow\text{AC}\rightarrow\text{ACD}\rightarrow\text{ACDB}\rightarrow\text{A},$$

but from AC it generates

$$\text{AC}\rightarrow\text{ACB}\rightarrow\text{ACBD}\rightarrow\text{A}.$$

However, it is seen that the sequential improvement criterion (2.13) holds at each of the states A, AB, AC, and AD (and also trivially for the remaining states).

The base heuristic at A is the nearest neighbor heuristic so it generates

$$\text{A}\rightarrow\text{AC}\rightarrow\text{ACD}\rightarrow\text{ACDB}\rightarrow\text{A with cost 28.}$$

The rollout algorithm at state A looks at the three successor states AB, AC, AD, and runs the farthest neighbor heuristic from each, and generates:

$$\text{A}\rightarrow\text{AB}\rightarrow\text{ABD}\rightarrow\text{ABDC}\rightarrow\text{A with cost 13,}$$

$$\text{A}\rightarrow\text{AC}\rightarrow\text{ACB}\rightarrow\text{ACBD}\rightarrow\text{A with cost 45,}$$

$$\text{A}\rightarrow\text{AD}\rightarrow\text{ADB}\rightarrow\text{ADBC}\rightarrow\text{A with cost 45,}$$

so the rollout algorithm will move from A to AB.

Then the rollout algorithm looks at the two successor states ABC, ABD, and runs the base heuristic (whatever that may be; it does not matter) from each. The paths generated are:

$$\text{AB}\rightarrow\text{ABC}\rightarrow\text{ABCD}\rightarrow\text{A with cost 26,}$$

AB→ABD→ABDC→A with cost 13,

so the rollout algorithm will move from AB to ABD.

Thus the final tour generated by the rollout algorithm is

A→AB→ABD→ABDC→A, with cost 13.

2.2 (A Generic Example of a Base Heuristic that is not Sequentially Improving)

Consider a rollout algorithm for a deterministic problem with a base heuristic that produces an optimal control sequence at the initial state x_0, and uses the (optimal) first control u_0 of this sequence to move to the (optimal) next state x_1. Suppose that the base heuristic produces a strictly suboptimal sequence from every successor state $x_2 = f_1(x_1, u_1)$, $u_1 \in U_1(x_1)$, so that the rollout yields a control u_1 that is strictly suboptimal. Show that the trajectory produced by the rollout algorithm starting from the initial state x_0 is strictly inferior to the one produced by the base heuristic starting from x_0, while the sequential improvement condition does not hold.

2.3 (Computational Exercise - Parking with Problem Approximation and Rollout)

In this computational exercise we consider a more complex, imperfect state information version of the one-directional parking problem of Example 1.6.1. Recall that in this problem a driver is looking for a free parking space in an area consisting of N spaces arranged in a line, with a garage at the end of the line (space N). The driver starts at space 0 and traverses the parking spaces sequentially, i.e., from each space he/she goes to the next space, up to when he/she decides to park in space k at cost $c(k)$, if space k is free. Upon reaching the garage, parking is mandatory at cost C.

In Example 1.6.1, we assumed that the driver knows the probabilities $p(k+1), \ldots, p(N-1)$ of the parking spaces $(k+1), \ldots, (N-1)$, respectively, being free. Under this assumption, the state at stage k is either the termination state t (if already parked), or it is F (location k free), or it is $\overline{F}$ (location k taken), and the DP algorithm has the form

$$J_k^*(F) = \begin{cases} \min\Big[c(k),\, p(k+1)J_{k+1}^*(F) + \big(1-p(k+1)\big)J_{k+1}^*(\overline{F})\Big] & \text{if } k < N-1, \\ \min\big[c(N-1),\, C\big] & \text{if } k = N-1, \end{cases} \tag{2.110}$$

$$J_k^*(\overline{F}) = \begin{cases} p(k+1)J_{k+1}^*(F) + \big(1-p(k+1)\big)J_{k+1}^*(\overline{F}) & \text{if } k < N-1, \\ C & \text{if } k = N-1, \end{cases} \tag{2.111}$$

for the states other than the termination state t, while for t we have $J_k^*(t) = 0$ for all k.

We will now consider the more complex variant of the problem where the probabilities $p(0), \ldots, p(N-1)$ do not change over time, but are unknown to

the driver, so that he/she cannot use the exact DP algorithm (2.110)-(2.111). Instead, the driver considers a one-step lookahead approximation in value space scheme, which uses empirical estimates of these probabilities that are based on the ratio $\frac{f_k}{k+1}$, where f_k is the number of free spaces seen up to space k, after the free/taken status of spaces $0,\ldots,k$ has been observed. In particular, these empirical estimates are given by

$$b_k(m,f_k)=\gamma\bar{p}(m)+(1-\gamma)\frac{f_k}{k+1},\qquad m=k+1,\ldots,N-1, \tag{2.112}$$

where f_k is the number of free spaces seen up to space k, and γ and $\bar{p}(m)$ are fixed numbers between 0 and 1. Of course the values f_k observed by the driver evolve according to the true (and unknown) probabilities $p(0),\ldots,p(N-1)$ according to

$$f_{k+1}=\begin{cases} f_k+1 & \text{with probability } p(k+1),\\ f_k & \text{with probability } 1-p(k+1).\end{cases} \tag{2.113}$$

For the solution of this exercise you may assume any reasonable values you wish for N, $p(m)$, $\bar{p}(m)$, and γ. Recommended values are $N\geq 100$, and probabilities $p(m)$ and $\bar{p}(m)$ that are nonincreasing with m.

The decision made by the approximation in value space scheme is to park at space k if and only if it is free and in addition

$$c(k)\leq b_k(k+1,f_k)\tilde{J}_{k+1}(F)+\big(1-b_k(k+1,f_k)\big)\tilde{J}_{k+1}(\overline{F}), \tag{2.114}$$

where $\tilde{J}_{k+1}(F)$ and $\tilde{J}_{k+1}(\overline{F})$ are the cost-to-go approximations from stage $k+1$. Consider the following two different methods to compute $\tilde{J}_{k+1}(F)$ and $\tilde{J}_{k+1}(\overline{F})$ for use in Eq. (2.114):

(1) Here the approximate cost function values $\tilde{J}_{k+1}(F)$ and $\tilde{J}_{k+1}(\overline{F})$ are obtained by using problem approximation, whereby at time k it is assumed that the probabilities of free/taken status at the future spaces $m=k+1,\ldots,N-1$ are $b_k(m,f_k)$, $m=k+1,\ldots,N-1$, as given by Eq. (2.112).

More specifically, $\tilde{J}_{k+1}(F)$ and $\tilde{J}_{k+1}(\overline{F})$ are obtained by solving optimally the problem whereby we use the probabilities $b_k(m,f_k)$ of Eq. (2.112) in place of the unknown $p(m)$ in the DP algorithm (2.110)-(2.111):

$$\tilde{J}_{k+1}(F)=\hat{J}_{k+1}(F),\qquad \tilde{J}_{k+1}(\overline{F})=\hat{J}_{k+1}(\overline{F}),$$

where $\hat{J}_{k+1}(F)$ and $\hat{J}_{k+1}(\overline{F})$ are given at the last step of the DP algorithm

$$\hat{J}_{N-1}(F)=\min\big[c(N-1),\,C\big],\qquad \hat{J}_{N-1}(\overline{F})=C,$$

$$\hat{J}_m(F)=\min\big[c(m),\,b_k(m+1,f_k)\hat{J}_{m+1}(F)+\big(1-b_k(m+1,f_k)\big)\hat{J}_{m+1}(\overline{F})\big],$$
$$\text{if } k<m<N-1,$$

$$\hat{J}_m(\overline{F})=b_k(m+1,f_k)\hat{J}_{m+1}(F)+\big(1-b_k(m+1,f_k)\big)\hat{J}_{m+1}(\overline{F}),$$
$$\text{if } k<m<N-1.$$

(2) Here for each k, the approximate cost function values $\tilde{J}_{k+1}(F)$ and $\tilde{J}_{k+1}(\overline{F})$ are obtained by using rollout with a greedy base heuristic (park as soon as possible), and Monte Carlo simulation. In particular, according to this greedy heuristic, we have $\tilde{J}_{k+1}(F) = c(k+1)$. To compute $\tilde{J}_{k+1}(\overline{F})$ we generate many random trajectories by running the greedy heuristic forward from space $k+1$ assuming the probabilities $b_k(m+1, f_k)$ of Eq. (2.112) in place of the unknown $p(m+1)$, $m = k+1, \ldots, N-1$, and we average the cost results obtained.

(a) Use Monte Carlo simulation to compute the expected cost from spaces $0, \ldots, N-1$, when using each of the two schemes (1) and (2).

(b) Compare the performance of the schemes of part (a) with the following:

(i) The optimal expected costs $J_k^*(F)$ and $J_k^*(\overline{F})$ from $k = 0, \ldots, N-1$, using the DP algorithm (2.110)-(2.111), and the probabilities $p(m)$, $m = 0, \ldots, N-1$, that you used for the random generation of the numbers of free spaces f_k [cf. Eq. (2.113)].

(ii) The expected costs $\hat{J}_k(F)$ and $\hat{J}_k(\overline{F})$ from $k = 0, \ldots, N-1$ that are attained by using the greedy base heuristic. Argue that these are given by

$$\hat{J}_k(F) = c(k), \qquad k = 0, \ldots, N-1,$$

$$\hat{J}_k(\overline{F}) = p(k+1)c(k+1) + \big(1 - p(k+1)\big)\hat{J}_{k+1}(\overline{F}), \qquad k = 0, \ldots, N-2,$$

$$\hat{J}_{N-1}(\overline{F}) = C.$$

(c) Argue that scheme (1) becomes superior to scheme (2) in terms of cost attained as $\gamma \approx 1$ and $\bar{p}(m) \approx p(m)$. Are your computational results in rough agreement with this assertion?

(d) Argue that as $\gamma \approx 0$ and $N >> 1$, scheme (1) becomes superior to scheme (2) in terms of cost attained from parking spaces $k >> 1$.

(e) What happens if the probabilities $p(m)$ do not change much with m?

2.4 (Breakthrough Problem with a Random Base Heuristic)

Consider the breakthrough problem of Example 2.3.2 with the difference that instead of the greedy heuristic, we use the *random* heuristic, which at a given node selects one of the two outgoing arcs with equal probability. Denote by

$$D_k = p^k$$

the probability of success of the random heuristic in a graph of k stages, and by R_k the probability of success of the corresponding rollout algorithm. Show that for all k

$$R_k = p(2-p)R_{k-1} + p^2 D_{k-1}(1 - R_{k-1}).$$

and that

$$\frac{R_k}{D_k} = (2-p)\frac{R_{k-1}}{D_{k-1}} + p(1 - R_{k-1}).$$

Conclude that R_k/D_k increases exponentially with k.

2.5 (Breakthrough Problem with Truncated Rollout)

Consider the breakthrough problem of Example 2.3.2 and consider a truncated rollout algorithm that uses a greedy base heuristic with ℓ-step lookahead. This is the same algorithm as the one described in Example 2.3.2, except that if both outgoing arcs of the current node at stage k are free, the rollout algorithm considers the two end nodes of these arcs, and from each of them it runs the greedy algorithm for $\min\{l, N-k-1\}$ steps. Consider a Markov chain with $l+1$ states, where states $i = 0, \ldots, l-1$ correspond to the path generated by the greedy algorithm being blocked after i arcs. State ℓ corresponds to the path generated by the greedy algorithm being unblocked after ℓ arcs.

(a) Derive the transition probabilities for this Markov chain so that it models the operation of the rollout algorithm.

(b) Use computer simulation to generate the probability of a breakthrough, and to demonstrate that for large values of N, the optimal value of ℓ is roughly constant and much smaller than N (this can also be justified analytically, by using properties of Markov chains).

2.6 (Incremental Truncated Rollout Algorithm for Constraint Programming)

Consider a discrete N-stage optimization problem, involving a tree with a root node s that plays the role of an artificial initial state, and N layers of states $x_1 = (u_0), x_2 = (u_0, u_1), \ldots, x_N = (u_0, \ldots, u_{N-1})$, as shown in Fig. 2.1.4. We allow deadend nodes in the graph of this problem, i.e., states that have no successor states, and thus cannot be part of any feasible solution. We also assume that all stage costs as well as the terminal cost are 0. The problem is to find a feasible solution, i.e., a sequence of N transitions through the graph that starts at the initial state s and ends at some node of the last layer of states x_N.

(a) Argue that this is a discrete spaces formulation of a constraint programming problem, such as the one described in Section 2.1.

(b) Describe in detail an incremental rollout algorithm with ℓ-step lookahead minimization and m-step rollout truncation, which is similar to the IMR algorithm of Section 2.4.2 and operates as follows: The algorithm maintains a connected subtree S that contains the initial state s. The base policy at a state x_k either generates a feasible sequence of $\overline{m}$ arcs starting at x_k, where $\overline{m}$ is an integer that satisfies $1 \leq \overline{m} \leq min\{m, N-k\}$, or determines that such a sequence does not exist. In the former case the node x_k is expanded by adding all of its neighbor nodes to S. In the latter case, the node x_k is deleted from S. The algorithm terminates once a state x_N of the last layer is added to S.

(c) Argue that since the algorithm cannot keep deleting nodes indefinitely, one of two things will eventually happen:

  (1) The graph S will be reduced to just the root node s, proving that there is no feasible solution.

  (2) The algorithm will terminate with a feasible solution.

(d) Suppose that the algorithm is operated so that the selected node x_k at each iteration is a leaf node of S, which is at maximum arc distance from s (the number of arcs of the path connecting s and x_k is maximized). Show that the subtree S always consists of just a path of nodes, together with all the neighbor nodes of the nodes of the path. Conclude that in this case, the algorithm can be implemented so that it requires $O(Nd)$ memory storage, where d is the maximum node degree. How does this algorithm compare with a depth-first search algorithm for finding a feasible solution?

(e) Describe an adaptation of the algorithm of part (d) for the case where most of the arcs have cost 0 but there are some arcs with positive cost.

2.7 (Purchasing Over Time with Multiagent Rollout)

Consider a market that makes available for purchase m products over N time periods, and a buyer that may or may not buy any one of these products subject to cash availability. For each product $i = 1, \ldots, m$ and time period $k = 0, \ldots, N-1$, we denote:

a_k^i: The asking price of product i at time k (the case where product i is unavailable for purchase is modeled by setting a_k^i to ∞).

v_k^i: The value to the buyer of product i at time k.

u_k^i: The decision to buy ($u_k^i = 1$) or not to buy ($u_k^i = 0$) product i at time k.

The conditional distributions $P(a_{k+1}^i \mid a_k^i, u_k^i = 1)$ and $P(a_{k+1}^i \mid a_k^i, u_k^i = 0)$ are given. (Thus when $u_k^i = 1$, product i will be made available at the next time period at a possibly different price a_{k+1}^i; however, it may also be unavailable, i.e., $a_{k+1}^i = \infty$.)

The amount of cash available to the buyer at time k is denoted by c_k, and evolves according to

$$c_{k+1} = c_k - \sum_{i=1}^{m} u_k^i a_k^i.$$

The initially available cash c_0 is a given positive number. Moreover, we have the constraint

$$\sum_{i=1}^{m} u_k^i a_k^i \leq c_k,$$

i.e., the buyer may not borrow to buy products. The buyer aims to maximize the total value obtained over the N time periods, plus the remaining cash at time N:

$$c_N + \sum_{k=0}^{N-1} \sum_{i=1}^{m} u_k^i a_k^i.$$

(a) Formulate the problem as a finite horizon DP problem by identifying the state, control, and disturbance spaces, the system equation, the cost function, and the probability distribution of the disturbance. Write the corresponding exact DP algorithm.

(b) Introduce a suitable base policy and formulate a corresponding multiagent rollout algorithm for addressing the problem.

2.8 (Treasure Hunting Using Adaptive Control and Rollout)

Consider a problem of sequentially searching for a treasure of known value v among n given locations. At each time period we may either select a location i to search at cost $c_i > 0$, or we may stop searching. Moreover, if the search location i is different from the location j where we currently are, we incur an additional switching cost $s_{ij} \geq 0$. If the treasure is at location i, a search at that location will find it with known probability $\beta_i < 1$. Our initial location is given, and the a priori probabilities p_i, $i = 1, \ldots, n$, that the treasure is at location i are also given. We assume that $\sum_{i=1}^{n} p_i < 0$, so there is positive probability that there is no treasure at any one of the n locations.

(a) Formulate the problem as a special case of the adaptive control problem of Section 2.11, with the parameter θ taking one of $(n + 1)$ values, $\theta^0, \theta^1, \ldots, \theta^n$, where θ^0 corresponds to the case where there is no treasure at any location, and θ^i, $i = 1, \ldots, n$, corresponds to the case where the treasure is at location i. Use as state the current location together with the current probability distribution of θ, and use as control the choice between stopping the search or continuing the search at one of the n locations.

(b) Consider the special case where there is only one location. Show that the optimal policy is to continue searching up to the point where the conditional expected benefit of the search falls below a certain threshold, and that the optimal cost can be computed very simply. (The proof is given in Example 4.3.1 of the DP textbook [Ber17a].)

(c) Formulate the rollout algorithm of Section 2.11 with two different base policies:

  (1) A policy that is optimal among the policies that never switch to another location (they continue to search the same location up to stopping).

  (2) A policy that is optimal among the policies that may stay at the current location or may switch to the location that is most likely to contain the treasure according to the current probability distribution of θ.

(d) Implement the preceding two rollout algorithms using reasonable problem data of your choice.

3

Learning Values and Policies

Contents

3.1. Parametric Approximation Architectures . . . . . p. 323
3.1.1. Cost Function Approximation . . . . . p. 324
3.1.2. Feature-Based Architectures . . . . . p. 325
3.1.3. Training of Linear and Nonlinear Architectures . p. 336
3.2. Neural Networks . . . . . p. 343
3.2.1. Training of Neural Networks . . . . . p. 348
3.2.2. Multilayer and Deep Neural Networks . . . . . p. 349
3.3. Training of Cost Functions in Approximate DP . . . . p. 351
3.3.1. Fitted Value Iteration . . . . . p. 351
3.3.2. Q-Factor Parametric Approximation - Model-Free Implementation . . . . . p. 353
3.3.3. Parametric Approximation in Infinite Horizon Problems - Approximate Policy Iteration . . . . p. 356
3.3.4. Optimistic Policy Iteration with Parametric Q-Factor Approximation - SARSA and DQN . . . . . p. 359
3.3.5. Approximate Policy Iteration for Infinite Horizon POMDP . . . . . p. 362
3.3.6. Advantage Updating - Approximating Q-Factor Differences . . . . . p. 366
3.3.7. Differential Training of Cost Differences for Rollout p. 369
3.4. Training of Policies in Approximate DP . . . . . p. 371
3.4.1. The Use of Classifiers for Approximation in Policy Space . . . . . p. 371
3.4.2. Policy Iteration with Value and Policy Networks p. 375
3.4.3. Why Use On-Line Play and not Just Train a Policy Network to Emulate the Lookahead Minimization? p. 378

3.5. Policy Gradient and Related Methods . . . . . p. 379
3.5.1. Gradient Methods for Cost Optimization . . . . p. 380
3.5.2. Random Search and Cross-Entropy Methods . . p. 388
3.6. Aggregation . . . . . p. 390
3.6.1. Aggregation with Representative States . . . . p. 391
3.6.2. Continuous Control Space Discretization . . . . p. 397
3.6.3. Continuous State Space - POMDP Discretization p. 398
3.6.4. General Aggregation . . . . . p. 400
3.6.5. Hard Aggregation and Error Bounds . . . . . p. 403
3.6.6. Aggregation Using Features . . . . . p. 405
3.6.7. Biased Aggregation . . . . . p. 408
3.6.8. Asynchronous Distributed Multiagent Aggregation p. 411
3.7. Notes, Sources, and Exercises . . . . . p. 413

In this chapter, we will discuss the methods and objectives of off-line training through the use of parametric approximation architectures such as neural networks. We begin with a general discussion of parametric architectures and their training in Section 3.1. We then consider the training of neural networks in Section 3.2, and their use in the context of finite horizon approximate DP in Section 3.3. In Sections 3.4 and 3.5, we discuss the training of policies. Finally, in Section 3.6, we discuss aggregation methods.

3.1 PARAMETRIC APPROXIMATION ARCHITECTURES

For the success of approximation in value space, it is important to select a class of lookahead function approximations $\tilde{J}_k$ that is suitable for the problem at hand. In the preceding two chapters we discussed several methods for choosing $\tilde{J}_k$, based mostly on some form of rollout. We will now discuss how $\tilde{J}_k$ can be obtained by off-line training from a parametric class of functions, possibly involving a neural network, with the parameters "optimized" with the use of some algorithm.

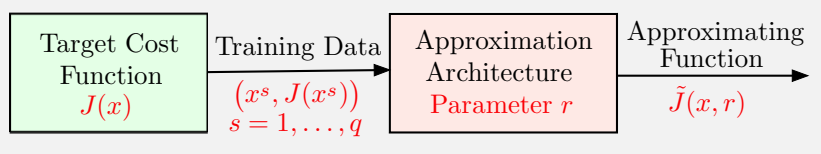

Figure 3.1.1 The general structure for parametric cost approximation. We approximate the target cost function $J(x)$ with a member from a parametric class $\tilde{J}(x, r)$ that depend on a parameter vector r. We use training data $\big(x^s, J(x^s)\big)$, $s = 1, \ldots, q$, and a form of optimization that aims to find a parameter $\hat{r}$ that "minimizes" the size of the errors $J(x^s) - \tilde{J}(x^s, \hat{r})$, $s = 1, \ldots, q$.

As we have noted in Chapter 1, the most popular structure for parametric cost function approximation involves a target function $J(x)$ that we want to approximate with a member of a parametric class of functions $\tilde{J}(x, r)$ that depend on a parameter vector r (see Fig. 3.1.1). In particular, we collect training data $\big(x^s, J(x^s)\big)$, $s = 1, \ldots, q$, which we use to determine a parameter $\hat{r}$ that leads to a good "fit" between the data $J(x^s)$ and the predictions $\tilde{J}(x^s, \hat{r})$ of the parametrized function. This is usually done through an optimization approach, aiming to minimize the size of the errors $J(x^s) - \tilde{J}(x^s, \hat{r})$, $s = 1, \ldots, q$.

Approximation of a target policy μ with a policy from a parametric class $\tilde{\mu}(x, r)$ is largely similar. Here, the training data may be obtained, for example, from rollout control calculations, thus enabling the construction of both value and policy networks that can be combined for use in a perpetual rollout scheme. An important difference, however, is that the

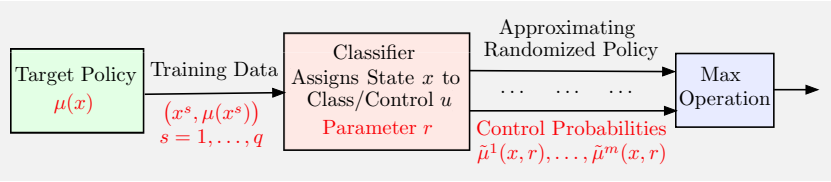

Figure 3.1.2 A general structure for parametric policy approximation for the case where the control space is finite, $U = \{u^1, \ldots, u^m\}$, and its relation to a classification scheme. It produces a randomized policy of the form (3.1), which is converted to a nonrandomized policy through the maximization operation (3.2).

approximate cost values $\tilde{J}(x, r)$ are real numbers, whereas the approximate policy values $\tilde{\mu}(x, r)$ are elements of a control space U. Thus if U consists of m dimensional vectors, $\tilde{\mu}(x, r)$ consists of m numerical components. In this case the parametric approximation problems for cost functions and for policies are fairly similar, and both involve continuous space approximations.

On the other hand, the case where the control space is finite $U = \{u^1, \ldots, u^m\}$ is quite different. In this case, for any x, $\tilde{\mu}(x, r)$ consists of one of the m possible controls $u^1, \ldots, u^m$. This connects policy space approximation with traditional classification schemes, whereby objects x are classified as belonging to one of the categories $u^1, \ldots, u^m$. In particular, $\mu(x)$ *defines the category of* x*, and can be viewed as a classifier*. Some of the most prominent classification schemes actually produce randomized outcomes, i.e., x is associated with a probability distribution

$$\{\tilde{\mu}(u^1, r), \ldots, \tilde{\mu}(u^m, r)\}, \tag{3.1}$$

which is a randomized policy in our policy approximation context; see Fig. 3.1.2. This is typically done algorithmic convenience, since many optimization methods, including least squares regression, require that the optimization variables are continuous. Then, the randomized policy (3.1) can be converted to a nonrandomized policy using a maximization operation: associate x with the control of maximum probability (cf. Fig. 3.1.2),

$$\tilde{\mu}(x, r) \in \arg \max_{i=1,\ldots,m} \tilde{\mu}^i(x, r). \tag{3.2}$$

We will discuss the use of classification methods for approximation in policy space in Section 3.4, following our discussion of parametric approximation in value space.

3.1.1 Cost Function Approximation

For the remainder of this section, as well as Sections 3.2 and 3.3, we will focus on approximation in value space schemes, where the approximate cost

functions are selected from a parametric class of functions $\tilde{J}_k(x_k, r_k)$ that for each k, depend on the current state x_k and a vector $r_k = (r_{1,k}, \ldots, r_{m_k,k})$ of m_k "tunable" scalar parameters. By adjusting the parameters, one can change the "shape" of $\tilde{J}_k$ so that it is a reasonably good approximation to some target function, usually the true optimal cost-to-go function J_k^*, or the cost-to-go function $J_{k,\pi}$ of some policy π. The class of functions $\tilde{J}_k(x_k, r_k)$ is called an *approximation architecture*, and the process of choosing the parameter vectors r_k is commonly called *training* or *tuning* the architecture. We will focus initially on approximation of cost functions, hence the use of the $\tilde{J}_k$ notation. In Section 3.4 we will consider the other major use of parametric approximation architectures, of the form $\tilde{\mu}_k(x_k, r_k)$, where the target function is a control function μ_k that is part of some policy.

The simplest training approach for parametric architectures is to do some form of semi-exhaustive or semi-random search in the space of parameter vectors and adopt the parameters that result in best performance of the associated one-step lookahead controller (according to some criterion). There are methods of this type that have been used primarily in cases where the number of parameters is relatively small.

Random search and Bayesian optimization methods have also been used to tune *hyperparameters* of an approximation architecture; for example, the number of layers in a neural network, or the number of clusters in the context of partitioning discrete spaces into clusters, etc. We refer to the research literature for further discussion.

Other systematic approaches are based on numerical optimization, such as a least squares fit that aims to match the cost approximation produced by the architecture to a "training set," i.e., a large number of pairs of state and cost values that are obtained through some form of sampling process. Throughout Sections 3.1-3.3 we will focus primarily on this approach.

3.1.2 Feature-Based Architectures

There is a large variety of approximation architectures, based for example on polynomials, wavelets, radial basis functions, discretization/interpolation schemes, neural networks, and others. A particularly interesting type of cost approximation involves *feature extraction*, a process that maps the state x_k into some vector $\phi_k(x_k)$, called the *feature vector* associated with x_k at time k. The vector $\phi_k(x_k)$ consists of scalar components

$$\phi_{1,k}(x_k), \ldots, \phi_{m_k,k}(x_k),$$

called *features*. A feature-based cost approximation has the form

$$\tilde{J}_k(x_k, r_k) = \hat{J}_k\big(\phi_k(x_k), r_k\big),$$

where r_k is a parameter vector and $\hat{J}_k$ is some function. Thus, the cost approximation depends on the state x_k through its feature vector $\phi_k(x_k)$.

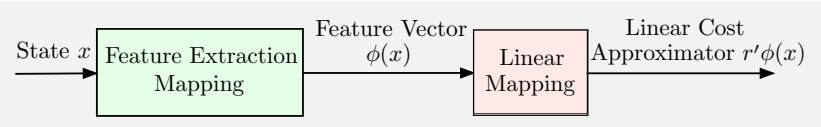

Figure 3.1.3 The structure of a linear feature-based architecture. At time k, we use a feature extraction mapping to generate an input $\phi_k(x_k)$ to a linear mapping defined by a parameter vector r_k.

Note that we are allowing for different features $\phi_k(x_k)$ and different parameter vectors r_k for each stage k. This is necessary for nonstationary problems (e.g., if the state space changes over time), and also to capture the effect of proximity to the end of the horizon. On the other hand, for stationary problems with a long or infinite horizon, where the state space does not change with k, it is common to use the same features and parameters for all stages. The subsequent discussion can easily be adapted to infinite horizon methods, as we will discuss later.

Features are often handcrafted, based on whatever human intelligence, insight, or experience is available, and are meant to capture the most important characteristics of the current state. There are also systematic ways to construct features, including the use of data and neural networks, which we will discuss shortly. In this section, we provide a brief and selective presentation of architectures.

One idea behind using features is that the optimal cost-to-go functions J_k^* may be complicated nonlinear mappings, so it is sensible to try to break their complexity into smaller, less complex pieces. In particular, if the features encode much of the nonlinearity of J_k^*, we may be able to use a relatively simple architecture $\hat{J}_k$ to approximate J_k^*. For example, with a well-chosen feature vector $\phi_k(x_k)$, a good approximation to the cost-to-go is often provided by *linearly* weighting the features, i.e.,

$$\tilde{J}_k(x_k, r_k) = \hat{J}_k\big(\phi_k(x_k), r_k\big) = \sum_{\ell=1}^{m_k} r_{\ell,k}\phi_{\ell,k}(x_k) = r_k'\phi_k(x_k), \tag{3.3}$$

where $r_{\ell,k}$ and $\phi_{\ell,k}(x_k)$ are the ℓth components of r_k and $\phi_k(x_k)$, respectively, and $r_k'\phi_k(x_k)$ denotes the inner product of r_k and $\phi_k(x_k)$, viewed as column vectors of $\Re^{m_k}$ (a prime denotes transposition, so r_k' is a row vector); see Fig. 3.1.3.

This is called a *linear feature-based architecture*, and the scalar parameters $r_{\ell,k}$ are also called *weights*. Among other advantages, these architectures admit simpler training algorithms than their nonlinear counterparts; see the NDP book [BeT96]. Mathematically, the approximating function $\tilde{J}_k(x_k, r_k)$ can be viewed as a member of the subspace spanned by the features $\phi_{\ell,k}(x_k)$, $\ell = 1, \ldots, m_k$, which for this reason are also referred to as *basis functions*. We provide a few examples, where for simplicity we drop the index k.

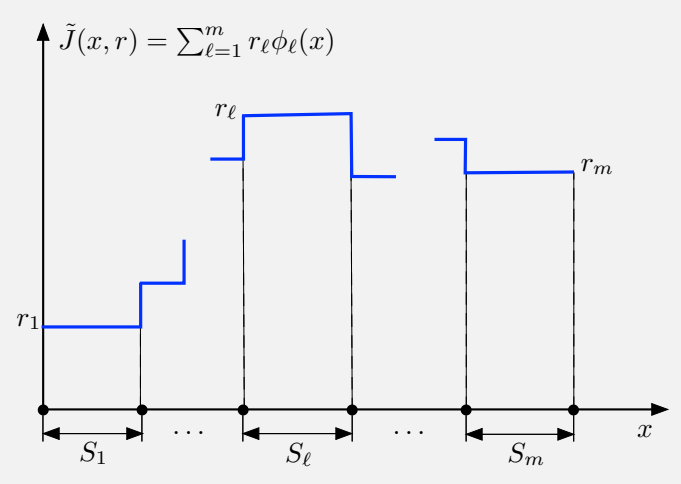

Figure 3.1.4 Illustration of a piecewise constant architecture. The state space is partitioned into subsets $S_1, \ldots, S_m$, with each subset S_ℓ defining the feature

$$\phi_\ell(x) = \begin{cases} 1 & \text{if } x \in S_\ell, \\ 0 & \text{if } x \notin S_\ell, \end{cases} \qquad \ell = 1, \ldots, m,$$

with its own weight r_ℓ.

Example 3.1.1 (Piecewise Constant Approximation)

Suppose that the state space is partitioned into subsets $S_1, \ldots, S_m$, so that every state belongs to one and only one subset. Let the ℓth feature be defined by membership to the set S_ℓ, i.e.,

$$\phi_\ell(x) = \begin{cases} 1 & \text{if } x \in S_\ell, \\ 0 & \text{if } x \notin S_\ell, \end{cases} \qquad \ell = 1, \ldots, m.$$

Consider the architecture

$$\tilde{J}(x,r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(x),$$

where r is the vector consists of the m scalar parameters $r_1, \ldots, r_m$. It can be seen that $\tilde{J}(x,r)$ is the piecewise constant function that has value r_ℓ for all states within the set S_ℓ; see Fig. 3.1.4.

The piecewise constant approximation is an example of a linear feature-based architecture that involves exclusively *local features*. These are features that take a nonzero value only for a relatively small subset of states. Thus a change of a single weight causes a change of the value of $\tilde{J}(x,r)$ for relatively few states x. At the opposite end we have linear

feature-based architectures that involve *global features*. These are features that take nonzero values for a large number of states. The following is a common example.

Example 3.1.2 (Polynomial Approximation)

An important case of linear architecture is one that uses polynomial basis functions. Suppose that the state consists of n components $x^1, \ldots, x^n$, each taking values within some range of integers. For example, in a queueing system, x^i may represent the number of customers in the ith queue, where $i = 1, \ldots, n$. Suppose that we want to use an approximating function that is quadratic in the components x^i. Then we can define a total of $1 + n + n^2$ basis functions that depend on the state $x = (x^1, \ldots, x^n)$ via

$$\phi_0(x) = 1, \qquad \phi_i(x) = x^i, \qquad \phi_{ij}(x) = x^i x^j, \quad i, j = 1, \ldots, n.$$

A linear approximation architecture that uses these functions is given by

$$\tilde{J}(x, r) = r_0 + \sum_{i=1}^{n} r_i x^i + \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij} x^i x^j,$$

where the parameter vector r has components r_0, r_i, and r_{ij}, with $i, j = 1, \ldots, n$. Indeed, any kind of approximating function that is polynomial in the components $x^1, \ldots, x^n$ can be constructed similarly.

A more general polynomial approximation may be based on some other known features of the state. For example, we may start with a feature vector

$$\phi(x) = \big(\phi_1(x), \ldots, \phi_m(x)\big)',$$

and transform it with a quadratic polynomial mapping. In this way we obtain approximating functions of the form

$$\tilde{J}(x, r) = r_0 + \sum_{i=1}^{m} r_i \phi_i(x) + \sum_{i=1}^{m} \sum_{j=1}^{m} r_{ij} \phi_i(x) \phi_j(x),$$

where the parameter r has components r_0, r_i, and r_{ij}, with $i, j = 1, \ldots, m$. This can also be viewed as a linear architecture that uses the basis functions

$$w_0(x) = 1, \qquad w_i(x) = \phi_i(x), \qquad w_{ij}(x) = \phi_i(x)\phi_j(x), \quad i, j = 1, \ldots, m.$$

The preceding example architectures are generic in the sense that they can be applied to many different types of problems. Other architectures rely on problem-specific insight to construct features, which are then combined into a relatively simple architecture. We present two examples involving games.

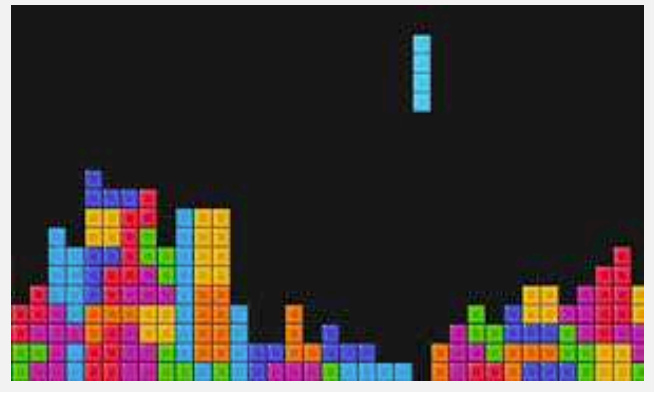

Figure 3.1.5 The board of the tetris game. The squares fill up as blocks of different shapes fall from the top of the grid and are added to the top of the wall. The shapes are generated according to some stochastic process. As a given block falls, the player can move horizontally and rotate the block in all possible ways, subject to the constraints imposed by the sides of the grid and the top of the wall. When a row of full squares is created, this row is removed, the bricks lying above this row move one row downward, and the player scores a point. The player's objective is to maximize the score attained (total number of rows removed) within N steps or up to termination of the game, whichever occurs first.

Example 3.1.3 (Tetris)

Let us consider the game of tetris, which we formulated in Example 1.6.2 as a stochastic shortest path problem with the termination state being the end of the game (see Fig. 3.1.5). The state is the pair of the board position x and the shape of the current falling block y. We viewed as control, the horizontal positioning and rotation applied to the falling block. The optimal cost-to-go function is a vector of huge dimension (there are 2^{200} board positions in a "standard" tetris board of width 10 and height 20). However, it has been successfully approximated in practice by low-dimensional linear architectures.

In particular, the following features have been proposed in the paper by Bertsekas and Ioffe [BeI96]: the heights of the columns, the height differentials of adjacent columns, the wall height (the maximum column height), the number of holes of the board, and the constant 1 (the unit is often included as a feature in cost approximation architectures, as it allows for a constant shift in the approximating function). These features are readily recognized by tetris players as capturing important aspects of the board position.† There

† The use of feature-based approximate DP methods for the game of tetris was first suggested in the paper by Tsitsiklis and Van Roy [TsV96], which introduced just two features (in addition to the constant 1): the wall height and the number of holes of the board. Most studies have used the set of features of [BeI96] described here, but other sets of features have also been used; see [ThS09] and the discussion in [GGS13].

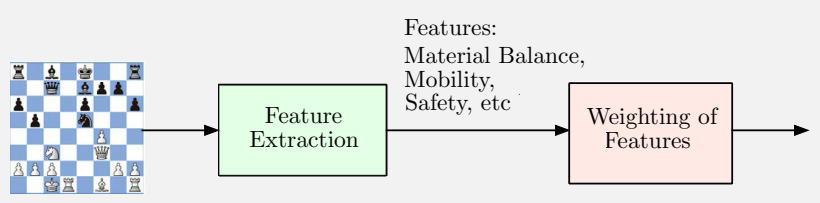

Figure 3.1.6 A feature-based architecture for computer chess.

are a total of 22 features for a "standard" board with 10 columns. Of course the $2^{200} \times 22$ matrix of feature values cannot be stored in a computer, but for any board position, the corresponding row of features can be easily generated, and this is sufficient for implementation of the associated approximate DP algorithms. For recent works involving approximate DP methods and the preceding 22 features, see [Sch13], [GGS13], and [SGG15], which reference several other related papers.

In the works mentioned above the shapes of the falling blocks are stochastically independent. In a more challenging version of the problem, which has not been considered in the literature thus far, successive shapes are correlated. Then the state of the problem would become more complex, since past shapes would be useful in predicting future shapes. As a result, we may need to introduce state estimation and additional features in order to properly deal with the effects of correlations.

Example 3.1.4 (Computer Chess)

Computer chess programs that involve feature-based architectures have been available for many years, and are still used widely (they have been upstaged in the mid-2010s by alternative types of chess programs, which use neural network techniques that will be discussed later). These programs are based on approximate DP for minimax problems, a feature-based parametric architecture, and multistep lookahead.

The fundamental principles on which all computer chess programs (as well as most two-person game programs) are based were laid out by Shannon [Sha50], before Bellman started his work on DP. Shannon proposed multistep lookahead and evaluation of the end positions by means of a "scoring function" (in our terminology this plays the role of a cost function approximation). This function may involve, for example, the calculation of a numerical value for each of a set of major features of a position that chess players easily recognize (such as material balance, mobility, pawn structure, and other positional factors), together with a method to combine these numerical values into a single score. Shannon then went on to describe various strategies of exhaustive and selective search over a multistep lookahead tree of moves.

We may view the scoring function as a feature-based architecture for evaluating a chess position/state (cf. Fig. 3.1.6). In most computer chess programs, the features are weighted linearly, i.e., the architecture $\tilde{J}(x, r)$ that is used for multistep lookahead is linear [cf. Eq. (3.3)]. In many cases, the weights have been determined manually, by trial and error based on experi-

ence. However, in some programs, the weights have been determined with supervised learning techniques that use examples of grandmaster play, i.e., by adjustment to bring the play of the program as close as possible to the play of chess grandmasters. This is a technique that applies more broadly in artificial intelligence; see Tesauro [Tes89b], [Tes01].

In a recent computer chess breakthrough, the entire idea of extracting features of a position through human expertise was abandoned in favor of feature discovery through self-play and the use of neural networks. The first program of this type to attain supremacy over humans, as well as over the best computer programs that use human expertise-based features, was AlphaZero (Silver et al. [SHS17]). This program, described in Section 1.1, is based on DP principles of approximate policy iteration and multistep lookahead based on Monte Carlo tree search.

Our next example relates to a methodology for feature construction, where the number of features may increase as more data is collected. For a simple example, consider the piecewise constant approximation of Example 3.1.1, where more pieces are progressively added based on new data, possibly using some form of exploration-exploitation tradeoff.

Example 3.1.5 (Feature Extraction from Data)

We have viewed so far feature vectors $\phi(x)$ as functions of x, obtained through some unspecified process that is based on prior knowledge about the cost function being approximated. On the other hand, features may also be extracted from data. For example suppose that with some preliminary calculation using data, we have identified some suitable states $x(\ell)$, $\ell = 1, \ldots, m$, that can serve as "anchors" for the construction of Gaussian basis functions of the form

$$\phi_\ell(x) = e^{-\frac{\|x - x(\ell)\|^2}{2\sigma^2}}, \qquad \ell = 1, \ldots, m, \tag{3.4}$$

where σ is a scalar "variance" parameter, and $\|\cdot\|$ denotes the standard Euclidean norm. This type of function is known as a *radial basis function*. It is concentrated around the state $x(\ell)$, and it is weighed with a scalar weight r_ℓ to form a parametric linear feature-based architecture, which can be trained using additional data. Several other types of data-dependent basis functions, such as support vector machines, are used in machine learning, where they are often referred to as *kernels*.

While it is possible to use a preliminary calculation to obtain the anchors $x(\ell)$ in Eq. (3.4), and then use additional data for training, one may also consider enrichment of the set of basis functions simultaneously with training. In this case the number of the basis functions increases as the training data is collected. A motivation here is that the quality of the approximation may increase with additional basis functions. This idea underlies a field of machine learning, known as *kernel methods* or sometimes *nonparametric methods*.

A further discussion is outside our scope. We refer to the literature; see e.g., books such as Cristianini and Shawe-Taylor [ChS00], [ShC04], Scholkopf and Smola [ScS02], Bishop [Bis06], Kung [Kun14], surveys such as Hofmann,

Scholkopf, and Smola [HSS08], Pillonetto et al. [PDC14], RL-related discussions such as Dietterich and Wang [DiW02], Ormoneit and Sen [OrS02], Engel, Mannor, and Meir [EMM05], Jung and Polani [JuP07], Reisinger, Stone, and Miikkulainen [RSM08], Busoniu et al. [BBD10a], Bethke [Bet10], and recent developments such as Tu et al. [TRV16], Rudi, Carratino, and Rosasco [RCR17], Belkin, Ma, and Mandal [BMM18]. In what follows, for the sake of simplicity, we will focus on parametric architectures with a fixed and given feature vector, since the choice of approximation architecture is somewhat peripheral to our main focus.

The next example considers a feature extraction strategy that is particularly relevant to problems of partial state information.

Example 3.1.6 (Feature Extraction from Sufficient Statistics)

The concept of a sufficient statistic, which originated in inference methodologies, plays an important role in DP. As discussed in Section 1.6, it refers to quantities that summarize all the essential content of the state x_k for optimal control selection at time k.

In particular, consider a partial information context where at time k we have accumulated the *information vector* (also called the *past history*)

$$I_k = (z_0, \ldots, z_k, u_0, \ldots, u_{k-1}),$$

which consists of the past controls $u_0, \ldots, u_{k-1}$ and the state-related measurements $z_0, \ldots, z_k$ obtained at the times $0, \ldots, k$. The control u_k is allowed to depend only on I_k, and the optimal policy is a sequence of the form $\big\{\mu_0^*(I_0), \ldots, \mu_{N-1}^*(I_{N-1})\big\}$. We say that a function $S_k(I_k)$ is a *sufficient statistic at time* k if the control function μ_k^* depends on I_k only through $S_k(I_k)$, i.e., for some function $\hat{\mu}_k$, we have

$$\mu_k^*(I_k) = \hat{\mu}_k\big(S_k(I_k)\big),$$

where μ_k^* is optimal.

There are several examples of sufficient statistics, and they are typically problem-dependent. A trivial possibility is to view I_k itself as a sufficient statistic, and a more sophisticated possibility is to view the *belief state* b_k as a sufficient statistic (this is the conditional probability distribution of x_k given I_k; cf. Section 1.6.4). For a proof that b_k is indeed a sufficient statistic and for a more detailed discussion of other possible sufficient statistics, see [Ber17a], Chapter 4. For a mathematically more advanced discussion, see [BeS78], Chapter 10.

Since a sufficient statistic contains all the relevant information for optimal control purposes, an idea that suggests itself is to introduce features of a given sufficient statistic and to train a corresponding approximation architecture accordingly. As examples of potentially good features, one may consider some special characteristic of I_k (such as whether some alarm-like "special" event has been observed), or a partial history (such as the last m measurements and controls in I_k, or more sophisticated versions based on the concept

of a *finite-state controller* proposed by White [Whi91], and White and Scherer [WhS94], and further discussed by Hansen [Han98], Kaelbling, Littman, and Cassandra [KLC98], Meuleau et al. [MPK99], Poupart and Boutilier [PoB04], Yu and Bertsekas [YuB08], Saldi, Yuksel, and Linder [SYL17]). In the case where the belief state b_k is used as a sufficient statistic, examples of good features may be a point estimate based on b_k, the variance of this estimate, and other quantities that can be simply extracted from b_k.

The paper by Bhattacharya et al. [BBW20] considers another type of feature vector that is related to the belief state. This is a sufficient statistic, denoted by y_k, which subsumes the belief state b_k, in the sense that b_k can be computed exactly knowing y_k. One possibility is for y_k to be the union of b_k and some identifiable characteristics of the belief state, or some compact representation of the measurement history up to the current time (such as a number of most recent measurements, or the state of a finite-state controller). Even though the information content of y_k is no different than the information content of b_k for the purposes of exact optimization, a sufficient statistic y_k that is specially designed for the problem at hand may lead to improved performance in the presence of cost and policy approximations.

We finally note a related idea, which is to supplement a sufficient statistic with features of other sufficient statistics, and thus obtain an enlarged/richer sufficient statistic. In problem-specific contexts, and in the presence of approximations, this may yield improved results.

Example 3.1.7 (Feature-Based Dimensionality Reduction by Aggregation)

The use of a feature vector $\phi(x)$ to represent the state x in an approximation architecture of the form $\tilde{J}\big(\phi(x), r\big)$ implicitly involves *state aggregation*, i.e., the grouping of states into subsets. We will discuss aggregation in some detail in Section 3.6. Here we will give a summary of a special type of aggregation architecture.

In particular, let us assume that the feature vector can take only a finite number of values, and define for each possible value v, the subset of states S_v whose feature vector is equal to v:

$$S_v = \big\{ i \mid \phi(x) = v \big\}.$$

We refer to the sets S_v as the *aggregate states* induced by the feature vector. These sets form a partition of the state space. An approximate cost-to-go function of the form $\tilde{J}\big(\phi(x), r\big)$ is piecewise constant with respect to this partition; that is, it assigns the same cost-to-go value $\tilde{J}(v, r)$ to all states in the set S_v.

An often useful approach to deal with problem complexity in DP is to introduce an "aggregate" DP problem, whose states are some suitably defined feature vectors $\phi(x)$ of the original problem. The precise form of the aggregate problem may depend on intuition and/or heuristic reasoning, based on our understanding of the original problem. Suppose now that the aggregate problem is simple enough to be solved exactly by DP, and let $\hat{J}(v)$

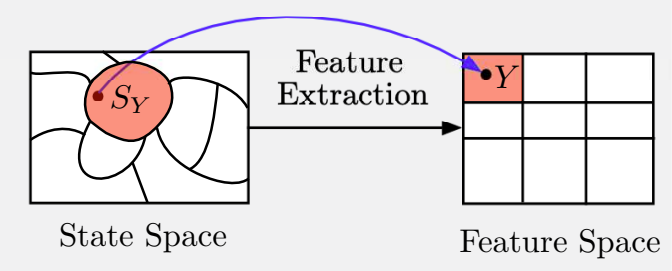

Figure 3.1.7 Feature-based state partitioning using a partition of the space of features. Each set Y of the feature space partition induces a set S_Y of the state space partition that consists of states with "similar" features, i.e., states that map into the same subset of the feature-space partition.

be its optimal cost-to-go when the initial value of the feature vector is v. Then $\hat{J}\big(\phi(x)\big)$ provides an approximation architecture for the original problem, i.e., the architecture that assigns to state x the (exactly) optimal cost-to-go $\hat{J}\big(\phi(x)\big)$ of the feature vector $\phi(x)$ in the aggregate problem. There is considerable freedom on how one formulates and solves aggregate problems. We refer to the DP textbooks [Ber12], [Ber17a], and the RL textbook [Ber19a], Chapter 6, for a detailed treatment; see also the discussion of Section 3.6.

The next example relates to an architecture that is particularly useful when parallel computation is available.

Example 3.1.8 (Feature-Based State Space Partitioning)

A simple method to construct complex and sophisticated approximation architectures, is to partition the state space into several subsets and construct a separate approximation in each subset. For example, by using a separate linear or quadratic polynomial approximation in each subset of the partition, we can construct piecewise linear or piecewise quadratic approximations over the entire state space. Similarly, we may use a separate neural network architecture on each set of the partition. An important issue here is the choice of the method for partitioning the state space. Regular partitions (e.g., grid partitions) may be used, but they often lead to a large number of subsets and very time-consuming computations.

Generally speaking, each subset of the partition should contain "similar" states so that the variation of the optimal cost-to-go over the states of the subset is relatively smooth and can be approximated with smooth functions. An interesting possibility is to use features as the basis for partition. In particular, one may use a more or less regular partition of the space of features, which induces a possibly irregular partition of the original state space. In this way, each subset of the irregular partition contains states with "similar features;" see Fig. 3.1.7.

As an illustration consider the game of chess. The state here consists of the board position, but the nature of the position progresses over time through opening, middlegame, and endgame phases. Moreover each of these

phases may be affected differently by special features of the position. For example there are several different types of endgames (rook endgames, king-and-pawn endgames, minor-piece endgames, etc), which are characterized by identifiable features and call for different playing strategies. It would thus make sense to partition the set of chess positions according to their features, and use a separate strategy on each set of the partition. Indeed this is done to some extent in a number of chess programs.

A potential difficulty with partitioned architectures is that there is discontinuity of the approximation along the boundaries of the partition. For this reason, a variant, called *soft partitioning*, is sometimes employed, whereby the subsets of the partition are allowed to overlap and the discontinuity is smoothed out over their intersection. In particular, once a function approximation is obtained in each subset, the approximate cost-to-go in the overlapping regions is taken to be a smoothly varying linear combination of the function approximations of the corresponding subsets.

Partitioning and local approximations can also be used to enhance the quality of approximation in parts of the space where the target function has some special character. For example, suppose that the state space S is partitioned in subsets $S_1, \ldots, S_M$ and consider approximations of the form

$$\tilde{J}(x,r) = \hat{J}(x,\hat{r}) + \sum_{m=1}^{M} \sum_{k=1}^{K_m} r_m(k)\phi_{k,m}(x), \tag{3.5}$$

where each $\phi_{k,m}(x)$ is a basis function which is local, in the sense that it contributes to the approximation only on the set S_m; that is, it takes the value 0 for $x \notin S_m$. Here $\hat{J}(x,\hat{r})$ is an architecture of the type discussed earlier, and the parameter vector r consists of $\hat{r}$ and the coefficients $r_m(k)$ of the basis functions. Thus the portion $\hat{J}(x,\hat{r})$ of the architecture is used to capture "global" aspects of the target function, while each portion

$$\sum_{k=1}^{K_m} r_m(k)\phi_{k,m}(i)$$

is used to capture aspects of the target function that are "local" to the subset S_m. The book [BeT96] (Section 3.1.3) discusses the training of local-global approximation architectures with methods that are tailored to their special structure.

Architectures with Automatic Feature Construction

Unfortunately, in practice we often do not know an adequate set of features, so it is important to have methods that construct features automatically, to supplement whatever features may already be available. Indeed, there are architectures that do not rely on the knowledge of good features. We have noted the kernel methods of Example 3.1.5 in this connection. Another very popular possibility is *neural networks*, which we will describe in Section 3.2.

Some of these architectures involve training that constructs simultaneously both the feature vectors $\phi(x)$ and the parameter vectors r that weigh them.

Generally, architectures that construct features automatically do not preclude the use of additional features that are based on a priori knowledge or understanding of the problem at hand. In particular these architectures may, in addition to x, use as inputs additional hand-crafted features that are relevant for the problem at hand. Another possibility is to combine automatically constructed features with other a priori known good features into a (mixed) linear architecture that involves both types of features. The weights of the latter linear architecture may be obtained with a separate second stage training process, following the first stage training process that constructs automatically suitable features using a nonlinear architecture such as a neural network.

3.1.3 Training of Linear and Nonlinear Architectures

In this section, we discuss briefly the training process of choosing the parameter vector r of a parametric architecture $\tilde{J}(x, r)$, focusing primarily on incremental gradient methods. The most common type of training is based on a least squares optimization, also known as *least squares regression*. Here a set of state-cost training pairs (x^s, β^s), $s = 1, \ldots, q$, called the *training set*, is collected and r is determined by solving the problem

$$\min_r \sum_{s=1}^{q} \big(\tilde{J}(x^s, r) - \beta^s\big)^2. \tag{3.6}$$

Thus r is chosen to minimize the sum of squared errors between the sample costs β^s and the architecture-predicted costs $\tilde{J}(x^s, r)$. Here there is some target cost function J that we aim to approximate with $\tilde{J}(\cdot, r)$, and the sample cost β^s is the value $J(x^s)$ plus perhaps some error or "noise."

The cost function of the training problem (3.6) is generally nonconvex, and can be quite complicated. This may pose challenges, since there may exist multiple local minima. However, for a linear architecture the cost function is convex quadratic, and the training problem admits a closed-form solution. In particular, for the linear architecture $\tilde{J}(x, r) = r'\phi(x)$, the problem becomes

$$\min_r \sum_{s=1}^{q} \big(r'\phi(x^s) - \beta^s\big)^2.$$

By setting the gradient of the quadratic objective to 0, we obtain

$$\sum_{s=1}^{q} \phi(x^s)\big(r'\phi(x^s) - \beta^s\big) = 0,$$

or

$$\sum_{s=1}^{q} \phi(x^s)\phi(x^s)'r = \sum_{s=1}^{q} \phi(x^s)\beta^s.$$

Thus by matrix inversion we obtain the minimizing parameter vector

$$\hat{r} = \left(\sum_{s=1}^{q} \phi(x^s)\phi(x^s)'\right)^{-1} \sum_{s=1}^{q} \phi(x^s)\beta^s. \tag{3.7}$$

If the inverse above does not exist, an additional quadratic in r, called a *regularization* function, is added to the least squares objective to deal with this, and also to help with other issues to be discussed later. A singular value decomposition approach may also be used to deal with the matrix inversion issue; see [BeT96], Section 3.2.2.

Thus a linear architecture has the important advantage that the training problem can be solved exactly and conveniently with the formula (3.7) (of course it may be solved by any other algorithm that is suitable for linear least squares problems, including iterative algorithms). By contrast, if we use a nonlinear architecture, such as a neural network, the associated least squares problem is nonquadratic and also nonconvex, so it is hard to solve in principle. Despite this fact, through a combination of sophisticated implementation of special gradient algorithms, called *incremental*, and powerful computational resources, neural network methods have been successful in practice.

Incremental Gradient Methods

We will now discuss briefly special methods for solution of the nonlinear least squares training problem (3.6), assuming a parametric architecture that is differentiable in the parameter vector. This methodology can be properly viewed as a subject in nonlinear programming and iterative algorithms, and as such it can be studied independently of the approximate DP methods of this book. Thus the reader who has already some exposure to the subject may skip to the next section. The author's nonlinear programming textbook [Ber16] and the RL book [Ber19a] provide more detailed presentations.

We view the training problem (3.6) as a special case of the minimization of a sum of component functions

$$f(y) = \sum_{i=1}^{m} f_i(y), \tag{3.8}$$

where each f_i is a differentiable scalar function of the n-dimensional column vector y (this is the parameter vector). Thus we use the more common

symbols y and m in place of r and q, respectively, and we replace the squared error terms

$$\big(\tilde{J}(x^s, r) - \beta^s\big)^2$$

in the training problem (3.6) with the generic terms $f_i(y)$.

The (ordinary) gradient method for problem (3.8) generates a sequence $\{y^k\}$ of iterates, starting from some initial guess y^0 for the minimum of the cost function f. It has the form†

$$y^{k+1} = y^k - \gamma^k \nabla f(y^k) = y^k - \gamma^k \sum_{i=1}^{m} \nabla f_i(y^k), \tag{3.9}$$

where γ^k is a positive stepsize parameter. The incremental gradient method is similar to the ordinary gradient method, but uses the gradient of a single component of f at each iteration. It has the general form

$$y^{k+1} = y^k - \gamma^k \nabla f_{i_k}(y^k), \tag{3.10}$$

where i_k is some index from the set $\{1, \ldots, m\}$, chosen by some deterministic or randomized rule. Thus a single component function f_{i_k} is used at iteration k, with great economies in gradient calculation cost over the ordinary gradient method (3.9), particularly when m is large. This is of course a radical simplification, which involves a large approximation error, yet it performs surprisingly well! The idea is to attain faster convergence when far from the solution as we will explain shortly; see the author's books [BeT96], [Ber16], and [Ber19a] for a more detailed discussion.

The method for selecting the index i_k of the component to be iterated on at iteration k is important for the performance of the method. We describe three common rules , the last two of which involve randomization:‡

(1) A *cyclic order*, the simplest rule, whereby the indexes are taken up in the fixed deterministic order $1, \ldots, m$, so that i_k is equal to (k modulo m) plus 1. A contiguous block of iterations involving the components $f_1, \ldots, f_m$ in this order and exactly once is called a *cycle*.

(2) A *uniform random order*, whereby the index i_k chosen randomly by sampling over all indexes with a uniform distribution, independently of the past history of the algorithm. This rule may perform better than the cyclic rule in some circumstances.

---

† We use standard calculus notation for gradients; see, e.g., [Ber16], Appendix A. In particular, $\nabla f(y)$ denotes the n-dimensional column vector whose components are the first partial derivatives $\partial f(y)/\partial y_i$ of f with respect to the components $y_1, \ldots, y_n$ of the column vector y.

‡ With these stepsize rules, the incremental gradient method is often called *stochastic gradient* or *stochastic gradient descent* method.

(3) A *cyclic order with random reshuffling*, whereby the indexes are taken up one by one within each cycle, but their order after each cycle is reshuffled randomly (and independently of the past). This rule is used widely in practice, particularly when the number of components m is modest, for reasons to be discussed later.

Note that in the cyclic cases, it is essential to include all components in a cycle; otherwise some components will be sampled more often than others, leading to a bias in the convergence process. Similarly, it is necessary to sample according to the uniform distribution in the random order case.

Focusing for the moment on the cyclic rule (with or without reshuffling), we note that the motivation for the incremental gradient method is faster convergence: we hope that far from the solution, a single cycle of the method will be as effective as several (as many as m) iterations of the ordinary gradient method (think of the case where the components f_i are similar in structure). Near a solution, however, the incremental method may not be as effective.

In particular, we note that there are two complementary performance issues to consider in comparing incremental and nonincremental methods:

(a) *Progress when far from convergence.* Here the incremental method can be much faster. For an extreme case take m large and all components f_i identical to each other. Then an incremental iteration requires m times less computation than a classical gradient iteration, but gives exactly the same result, when the stepsize is scaled to be m times larger. While somewhat extreme, this example reflects the essential mechanism by which incremental methods can be much superior: far from the minimum a single component gradient will point to "more or less" the right direction, at least most of the time; see the following example.

(b) *Progress when close to convergence.* Here the incremental method can be inferior. In particular, the ordinary gradient method (3.9) is convergent with a constant stepsize under reasonable assumptions, see e.g., [Ber16]. However, the incremental method requires a diminishing stepsize, and its ultimate rate of convergence can be much slower.

This type of behavior is illustrated in the following example, first given in the 1995 first edition of author's nonlinear programming book [Ber16], and the neurodynamic programming book [BeT96].

Example 3.1.9

Assume that y is a scalar, and that the problem is

$$\text{minimize} \quad f(y) = \tfrac{1}{2}\sum_{i=1}^{m}(c_i y - b_i)^2$$

$$\text{subject to} \quad y \in \Re,$$

where c_i and b_i are given scalars with $c_i \neq 0$ for all i. The minimum of each of the components $f_i(y) = \frac{1}{2}(c_i y - b_i)^2$ is

$$y_i^* = \frac{b_i}{c_i},$$

while the minimum of the least squares cost function f is

$$y^* = \frac{\sum_{i=1}^m c_i b_i}{\sum_{i=1}^m c_i^2}.$$

It can be seen that y^* lies within the range of the component minima

$$R = \left[\min_i y_i^*,\ \max_i y_i^*\right],$$

and that for all y *outside* the range R, the gradient

$$\nabla f_i(y) = c_i(c_i y - b_i)$$

has the same sign as $\nabla f(y)$ (see Fig. 3.1.8). As a result, when outside the region R, the incremental gradient method

$$y^{k+1} = y^k - \gamma^k c_{i_k}(c_{i_k} y^k - b_{i_k})$$

approaches y^* at each step, provided the stepsize γ^k is small enough. In fact it can be verified that it is sufficient that

$$\gamma^k \leq \min_i \frac{1}{c_i^2}.$$

However, for y *inside* the region R, the ith step of a cycle of the incremental gradient method need not make progress. It will approach y^* (for small enough stepsize γ^k) only if the current point y^k does not lie in the interval connecting y_i^* and y^*. This induces an oscillatory behavior within the region R, and as a result, the incremental gradient method will typically not converge to y^* unless $\gamma^k \to 0$. By contrast, the ordinary gradient method, which takes the form

$$y^{k+1} = y^k - \gamma \sum_{i=1}^m c_i(c_i y^k - b_i),$$

can be verified to converge to y^* for any constant stepsize γ with

$$0 < \gamma \leq \frac{1}{\sum_{i=1}^m c_i^2}.$$

However, for y outside the region R, a full iteration of the ordinary gradient method need not make more progress towards the solution than a single step of the incremental gradient method. In other words, with comparably intelligent stepsize choices, *far from the solution (outside R), a single pass through the entire set of cost components by incremental gradient is roughly as effective as m passes by ordinary gradient.*

The preceding example assumes that each component function f_i has a minimum, so that the range of component minima is defined. In cases where the components f_i have no minima, a similar phenomenon may occur, as illustrated by the following example (the idea here is that we may combine several components into a single component that has a minimum).

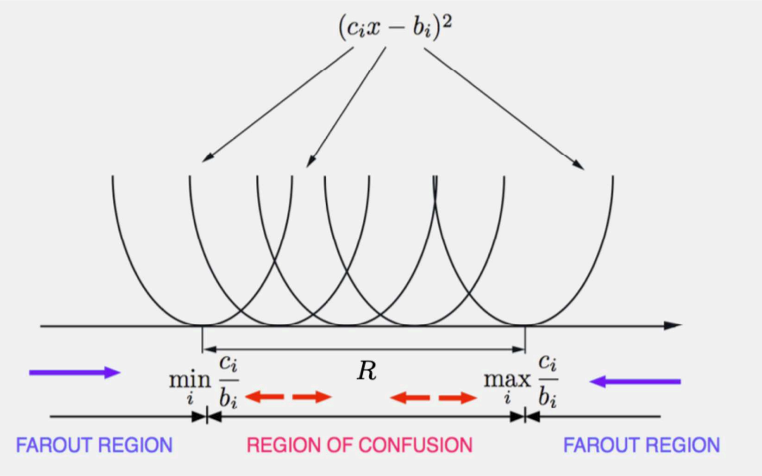

Figure 3.1.8. Illustrating the advantage of incrementalism when far from the optimal solution. The region of component minima

$$R = \left[\min_i y_i^*, \max_i y_i^*\right],$$

is labeled as the "region of confusion." It is the region where the method does not have a clear direction towards the optimum. The ith step in an incremental gradient cycle is a gradient step for minimizing $(c_i y - b_i)^2$, so if y lies outside the region of component minima $R = \left[\min_i y_i^*, \max_i y_i^*\right]$, (labeled as the "farout region") and the stepsize is small enough, progress towards the solution y^* is made.

Example 3.1.10:

Consider the case where f is the sum of increasing and decreasing convex exponentials, i.e.,

$$f_i(y) = a_i e^{b_i y}, \qquad y \in \Re,$$

where a_i and b_i are scalars with $a_i > 0$ and $b_i \neq 0$. Let

$$I^+ = \{i \mid b_i > 0\}, \qquad I^- = \{i \mid b_i < 0\},$$

and assume that I^+ and I^- have roughly equal numbers of components. Let also y^* be the minimum of $\sum_{i=1}^m f_i$.

Consider the incremental gradient method that given the current point, call it y^k, chooses some component f_{i_k} and iterates according to the incremental gradient iteration

$$y^{k+1} = y^k - \alpha^k \nabla f_{i_k}(y^k).$$

Then it can be seen that if $y^k >> y^*$, y^{k+1} will be substantially closer to y^* if $i \in I^+$, and negligibly further away than y^* if $i \in I^-$. The net effect, averaged

over many incremental iterations, is that if $y^k >> y^*$, an incremental gradient iteration makes roughly one half the progress of a full gradient iteration, with m times less overhead for calculating gradients. The same is true if $y^k << y^*$. On the other hand as y^k gets closer to y^* the advantage of incrementalism is reduced, similar to the preceding example. In fact in order for the incremental method to converge, a diminishing stepsize is necessary, which will ultimately make the convergence slower than the one of the nonincremental gradient method with a constant stepsize.

The discussion of the preceding examples relies on y being one-dimensional, but in many multidimensional problems the same qualitative behavior can be observed. In particular, a pass through the ith component f_i by the incremental gradient method can make progress towards the solution in the region where the component gradient $\nabla f_{i_k}(y^k)$ makes an angle less than 90 degrees with the cost function gradient $\nabla f(y^k)$. If the components f_i are not "too dissimilar," this is likely to happen in a region of points that are not too close to the optimal solution set. This behavior has been verified in many practical contexts, including the training of neural networks (cf. the next section), where incremental gradient methods have been used extensively, frequently under the name *backpropagation methods*.

Stepsize Choice and Diagonal Scaling

The choice of the stepsize γ^k plays an important role in the performance of incremental gradient methods. In practice, it is common to use a constant stepsize for a (possibly prespecified) number of iterations, then decrease the stepsize by a certain factor, and repeat, up to the point where the stepsize reaches a prespecified floor value. An alternative possibility is to use a diminishing stepsize rule of the form

$$\gamma^k = \min\left\{\gamma,\ \frac{\beta_1}{k+\beta_2}\right\},$$

where γ, β_1, and β_2 are some positive scalars. There are also variants of the method that use a constant stepsize throughout, and can be shown to converge to a stationary point of f under reasonable assumptions. In one type of such method the degree of incrementalism gradually diminishes as the method progresses (see [Ber97a]). Another incremental approach with similar aims, is the aggregated gradient method, which is discussed in the author's textbooks [Ber15a], [Ber16], [Ber19a].

Regardless of whether a constant or a diminishing stepsize is ultimately used, the incremental method must use a much larger stepsize than the corresponding nonincremental gradient method (as much as m times larger, so that the size of the incremental gradient step is comparable to the size of the nonincremental gradient step).

One possibility is to use an adaptive stepsize rule, whereby, roughly speaking, the stepsize is reduced (or increased) when the progress of the

method indicates that the algorithm is (or is not) oscillating. There are formal ways to implement such stepsize rules with sound convergence properties (see [Tse98], [MYF03]).

The difficulty with stepsize selection may also be addressed with *diagonal scaling*, i.e., using a stepsize γ_j^k that is different for each of the components y_j of y. Second derivatives can be very useful for this purpose. In generic nonlinear programming problems of unconstrained minimization of a function f, it is common to use diagonal scaling with stepsizes

$$\gamma_j^k = \gamma \left(\frac{\partial^2 f(y^k)}{\partial^2 y_j} \right)^{-1}, \qquad j = 1, \ldots, n,$$

where γ is a constant that is nearly equal 1 (the second derivatives may also be approximated by gradient difference approximations). However, in least squares training problems, this type of scaling is inconvenient because of the additive form of f as a sum of a large number of component functions:

$$f(y) = \sum_{i=1}^{m} f_i(y),$$

cf. Eq. (3.8). The neural network literature includes a number of practical scaling schemes, some of which have been incorporated in publicly and commercially available software; see the two influential papers, Duchi, Hazan, and Singer [DHS11], and Kingman and Ba [KiB14], as well as subsequent works that expand on the ideas of these two papers.

The RL book [Ber19a] (Section 3.1.3) describes another type method that involves second derivatives and is based on Newton's method. The idea here is to write Newton's method in a format that is well suited to the additive character of the cost function f, and involves low order matrix inversion. One can then implement diagonal scaling by setting to zero the off-diagonal terms of the inverted matrices, so that the algorithm involves no matrix inversion. There is also another related algorithm, which is based on the Gauss-Newton method and the extended Kalman filter; see the author's paper [Ber96], and the books [BeT96] and [Ber16].

3.2 NEURAL NETWORKS

There are several different types of neural networks that can be used for a variety of tasks, such as pattern recognition, classification, image and speech recognition, natural language processing, and others. In this section, we focus on our finite horizon DP context, and the role that neural networks can play in approximating the optimal cost-to-go functions J_k^*. As an example within this context, we may first use a neural network to construct an approximation to J_{N-1}^*. Then we may use this approximation to approximate J_{N-2}^*, and continue this process backwards in time, to obtain

approximations to all the optimal cost-to-go functions J_k^*, $k = 1, \ldots, N-1$, as we will discuss in more detail in Section 3.3.

Throughout this section, we will focus on the type of neural network, known as a *multilayer perceptron*, which is the one most used at present in the RL applications discussed in this book. Naturally, there are variations that are adapted to the problem at hand. For example AlphaZero uses a specialized neural network that takes advantage of the board-like structure of chess and Go to facilitate and expedite the associated computations.

To describe the use of neural networks in finite horizon DP, let us consider the typical stage k, and for convenience drop the index k; the subsequent discussion applies to each value of k separately. We consider parametric architectures $\tilde{J}(x, v, r)$ of the form

$$\tilde{J}(x, v, r) = r'\phi(x, v) \tag{3.11}$$

that depend on two parameter vectors v and r. Our objective is to select v and r so that $\tilde{J}(x, v, r)$ approximates some target cost function that can be sampled (possibly with some error). The process is to collect a training set that consists of a large number of state-cost pairs (x^s, β^s), $s = 1, \ldots, q$, and to find a function $\tilde{J}(x, v, r)$ of the form (3.11) that matches the training set in a least squares sense, i.e., (v, r) minimizes

$$\sum_{s=1}^{q} \big(\tilde{J}(x^s, v, r) - \beta^s\big)^2.$$

We postpone for later the question of how the training pairs (x^s, β^s) are generated.† Notice the different roles of the two parameter vectors here: *v parametrizes $\phi(x, v)$*, which in some interpretation may be viewed as a feature vector, and *r is a vector of linear weighting parameters for the components of $\phi(x, v)$*.

Single Layer Perceptron

A neural network architecture provides a parametric class of functions $\tilde{J}(x, v, r)$ of the form (3.11) that can be used in the optimization framework just described. The simplest type of neural network is the *single layer perceptron*; see Fig. 3.2.1. Here the state x is encoded as a vector of numerical values $y(x)$ with components $y_1(x), \ldots, y_n(x)$, which is then transformed linearly as

$$Ay(x) + b,$$

† The least squares training problem used here is based on *nonlinear regression*. This is a classical method for approximating the expected value of a function with a parametric architecture, and involves a least squares fit of the architecture to simulation-generated samples of the expected value. We refer to machine learning and statistics textbooks for more discussion.

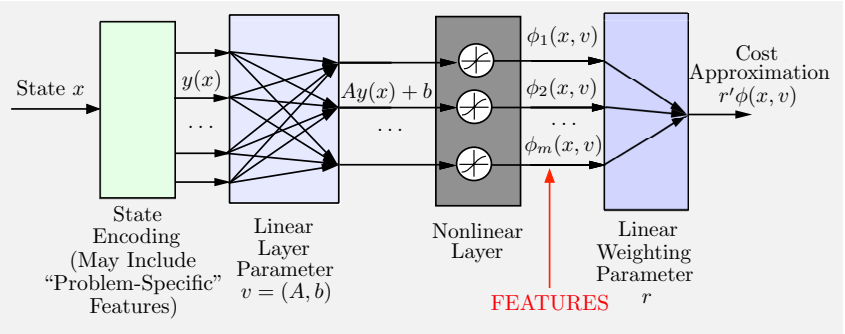

Figure 3.2.1 Schematic illustration of a single layer perceptron, a neural network consisting of a linear layer and a nonlinear layer. It provides a way to compute features of the state, which can be used for cost function approximation. The state x is encoded as a vector of numerical values $y(x)$, which is then transformed linearly as $Ay(x)+b$ in the linear layer. The m scalar output components of the linear layer, become the inputs to nonlinear one-dimensional functions $\sigma : \Re \mapsto \Re$, thus producing the m scalars

$$\phi_\ell(x,v) = \sigma\big((Ay(x)+b)_\ell\big),$$

which can be viewed as features that are in turn linearly weighted with parameters r_ℓ.

where A is an $m \times n$ matrix and b is a vector in $\Re^m$.† This transformation is called the *linear layer* of the neural network. We view the components of A and b as parameters to be determined, and we group them together into the parameter vector $v = (A, b)$.

Each of the m scalar output components of the linear layer,

$$\big(Ay(x)+b\big)_\ell, \qquad \ell = 1, \ldots, m,$$

becomes the input to a nonlinear differentiable and monotonically increasing function σ that maps scalars to scalars. A simple and popular possibility is the *rectified linear unit* (ReLU for short), which is simply the function $\max\{0, \xi\}$, approximated by a differentiable function σ by some form of smoothing operation; for example $\sigma(\xi) = \ln(1+e^\xi)$, which is illustrated in Fig. 3.2.2. Other functions, used since the early days of neural networks, have the property

$$-\infty < \lim_{\xi \to -\infty} \sigma(\xi) < \lim_{\xi \to \infty} \sigma(\xi) < \infty;$$

† The method of encoding x into the numerical vector $y(x)$ is generally problem-dependent, but it can be critical for the success of the training process. We should note also that some of the components of $y(x)$ could be known interesting features of x that can be designed based on problem-specific knowledge.

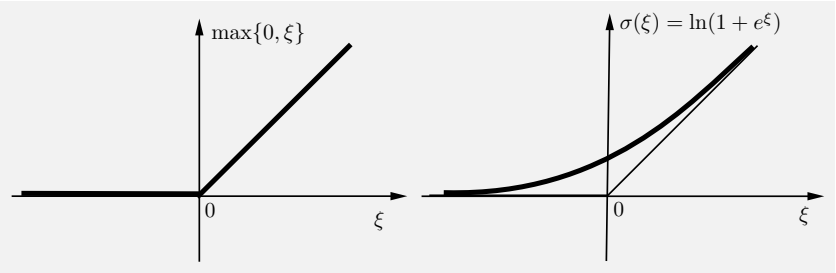

Figure 3.2.2 The rectified linear unit $\sigma(\xi) = \ln(1 + e^{\xi})$. It is the function $\max\{0, \xi\}$ with its corner "smoothed out." Its derivative is $\sigma'(\xi) = e^{\xi}/(1 + e^{\xi})$, and approaches 0 and 1 as $\xi \to -\infty$ and $\xi \to \infty$, respectively.

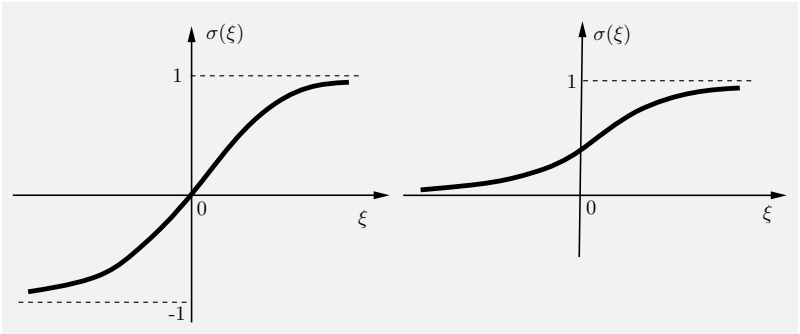

Figure 3.2.3 Some examples of sigmoid functions. The hyperbolic tangent function is on the left, while the logistic function is on the right.

see Fig. 3.2.3. Such functions are called *sigmoids*, and some common choices are the *hyperbolic tangent* function

$$\sigma(\xi) = \tanh(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}},$$

and the *logistic* function

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}}.$$

In what follows, we will ignore the character of the function σ (except for differentiability), and simply refer to it as a "nonlinear unit" and to the corresponding layer as a "nonlinear layer."

At the outputs of the nonlinear units, we obtain the scalars

$$\phi_\ell(x, v) = \sigma\big((Ay(x) + b)_\ell\big), \qquad \ell = 1, \ldots, m.$$

One possible interpretation is to view $\phi_\ell(x, v)$ as features of x, which are linearly combined using weights r_ℓ, $\ell = 1, \ldots, m$, to produce the final

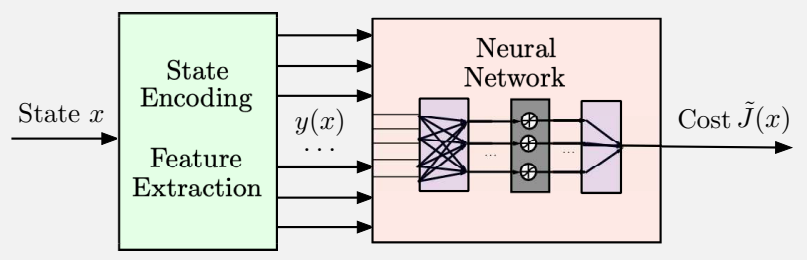

Figure 3.2.4 Nonlinear architecture with a view of the state encoding process as a feature extraction mapping preceding the neural network. The state encoder may also contain tunable parameters.

output

$$\tilde{J}(x, v, r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(x, v) = \sum_{\ell=1}^{m} r_\ell \sigma\big((Ay(x) + b)_\ell\big).$$

Note that each value $\phi_\ell(x, v)$ depends on just the ℓth row of A and the ℓth component of b, not on the entire vector v. In some cases this motivates placing some constraints on individual components of A and b to achieve special problem-dependent "handcrafted" effects.

State Encoding and Direct Feature Extraction

The state encoding operation that transforms x into the neural network input $y(x)$ can be instrumental in the success of the approximation scheme. Examples of state encodings are components of the state x, numerical representations of qualitative characteristics of x, and more generally features of x, i.e., functions of x that aim to capture "important nonlinearities" of the optimal cost-to-go function. With the latter view of state encoding, we may consider the approximation process as consisting of a feature extraction mapping, followed by a neural network with input the extracted features of x, and output the cost-to-go approximation; see Fig. 3.2.4. In a more general view of the neural network, the state encoder may involve some tunable parameters.

The idea here is that with a good feature extraction mapping, the neural network need not be very complicated (may involve few nonlinear units and corresponding parameters), and may be trained more easily. This intuition is borne out by simple examples and practical experience. However, as is often the case with neural networks, it is hard to support it with a quantitative analysis.

Universal Approximation Property of Neural Networks

An important question is how well we can approximate the target function J_k^* with a neural network architecture, assuming we can choose the number of the nonlinear units m to be as large as we want. The answer to

this question is quite favorable and is provided by the so-called *universal approximation theorem*.

Roughly, the theorem says that assuming that x is an element of a Euclidean space X and $y(x) \equiv x$, a neural network of the form described can approximate arbitrarily closely (in an appropriate mathematical sense), over a compact subset $S \subset X$, any piecewise continuous function $J : S \mapsto \Re$, provided the number m of nonlinear units is sufficiently large. For proofs of the theorem, we refer to Cybenko [Cyb89], Funahashi [Fun89], Hornik, Stinchcombe, and White [HSW89], and Leshno et al. [LLP93]. For additional sources and intuitive explanations we refer to Bishop ([Bis95], pp. 129-130), Jones [Jon90], and the RL textbook [Ber19a], Section 3.2.1.

While the universal approximation theorem provides some assurance about the adequacy of the neural network structure, it does not predict how many nonlinear units we may need for "good" performance in a given problem. Unfortunately, this is a difficult question to even pose precisely, let alone to answer adequately. In practice, one is often reduced to trying increasingly larger values of m until one is convinced that satisfactory performance has been obtained for the task at hand. One may improve on trial-and-error schemes with more systematic hyperparameter search methods, such as Bayesian optimization, and in fact this has been used to tune the parameters of the deep network used by AlphaZero.

Experience has shown that in many cases the number of required nonlinear units and corresponding dimension of A can be very large, adding significantly to the difficulty of solving the training problem. This has given rise to many suggestions for modifications of the perceptron structure. An important possibility is to concatenate multiple single layer perceptrons so that the output of the nonlinear layer of one perceptron becomes the input to the linear layer of the next, giving rise to deep neural networks, which we will discuss later.

3.2.1 Training of Neural Networks

Given a set of state-cost training pairs (x^s, β^s), $s = 1, \ldots, q$, the parameters of the neural network A, b, and r are obtained by solving the problem

$$\min_{A,b,r} \sum_{s=1}^{q} \left(\sum_{\ell=1}^{m} r_\ell \sigma\left(\big(Ay(x^s) + b\big)_\ell \right) - \beta^s \right)^2. \tag{3.12}$$

Note that the cost function of this problem is generally nonconvex, so there may exist multiple local minima.

In practice it is common to augment the cost function of this problem with a *regularization* function, such as a quadratic in the parameters A, b, and r. This is customary in least squares problems in order to make the problem easier to solve algorithmically. However, in the context of neural network training, regularization is primarily important for a different

reason: it helps to avoid *overfitting*, which occurs when the number of parameters of the neural network is relatively large (comparable to the size of the training set). In this case a neural network model matches the training data very well but may not do as well on new data. This is a known difficulty, which is the subject of much current research, particularly in the context of deep neural networks.

An important issue is to select a method to solve the training problem (3.12). While we can use any unconstrained optimization method that is based on gradients, in practice it is important to take into account the cost function structure of problem (3.12). The salient characteristic of this cost function is that it is the sum of a potentially very large number q of component functions. This makes the computation of the cost function value of the training problem and/or its gradient very costly. For this reason the incremental methods of Section 3.1.3 are universally used for training.† Experience has shown that these methods can be vastly superior to their nonincremental counterparts in the context of neural network training.

The implementation of the training process has benefited from experience that has been accumulated over time, and has provided guidelines for scaling, regularization, initial parameter selection, and other practical issues; we refer to books on neural networks such as Bishop [Bis95], Bishop and Bishop [BiB24], Goodfellow, Bengio, and Courville [GBC16], and Haykin [Hay08], as well as to the overview paper on deep neural network training by Sun [Sun19], and the references quoted therein. Still, incremental methods can be quite slow, and training may be a time-consuming process. Fortunately, the training is ordinarily done off-line, possibly using parallel computation, in which case computation time may not be a serious issue. Moreover, in practice the neural network training problem typically need not be solved with great accuracy. This is also supported by the Newton step view of approximation in value space, which suggests that great accuracy in the terminal cost function approximation is not critically important for good performance of the on-line play controller.

3.2.2 Multilayer and Deep Neural Networks

An important generalization of the single layer perceptron architecture involves a concatenation of multiple layers of linear and nonlinear functions; see Fig. 3.2.5. In particular the outputs of each nonlinear layer become the inputs of the next linear layer. In some cases it may make sense to add

† The incremental methods are valid for an arbitrary order of component selection within the cycle, but it is common to randomize the order at the beginning of each cycle. Also, in a variation of the basic method, we may operate on a batch of several components at each iteration, called a *minibatch*, rather than a single component. This has an averaging effect, which reduces the tendency of the method to oscillate and allows for the use of a larger stepsize; see the end-of-chapter references.

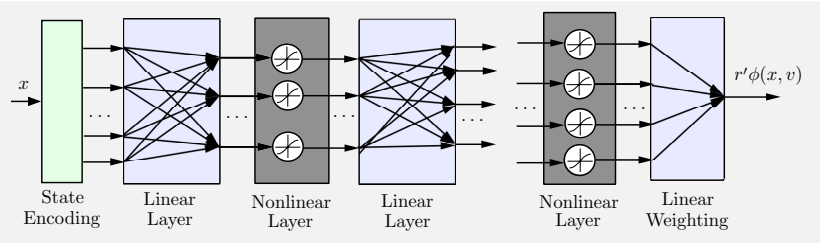

Figure 3.2.5 A deep neural network, with multiple layers. Each nonlinear layer constructs the inputs of the next linear layer.

as additional inputs some of the components of the state x or the state encoding $y(x)$.

In the early days of neural networks practitioners tended to use few nonlinear layers (say one to three). For example, Tesauro's backgammon program and its descendants have performed well with one or two nonlinear layers [PaR12]. However, more recently a lot of success in certain problem domains (including image and speech processing, large language models, as well as approximate DP) has been achieved with *deep neural networks*, which involve a considerably larger number of layers.

There are a few questions to consider here. The first has to do with the reason for having multiple nonlinear layers, when a single one is sufficient to guarantee the universal approximation property. Here are some qualitative (and somewhat speculative) explanations:

(a) If we view the outputs of each nonlinear layer as features, we see that the multilayer network produces a hierarchy of features, where each set of features is a function of the preceding set of features [except for the first set of features, which is a function of the encoding $y(x)$ of the state x]. In the context of specific applications, this hierarchical structure can be exploited to specialize the role of some of the layers and to enhance some characteristics of the state.

(b) Given the presence of multiple linear layers, one may consider the possibility of using matrices A with a particular sparsity pattern, or other structure that embodies special linear operations such as convolution, which may be well-matched to the training problem at hand. Moreover, when such structures are used, the training problem often becomes easier, because the number of parameters in the linear layers is drastically decreased.

(c) Overparametrization (more weights than data, as in a deep neural network) helps to mitigate the detrimental effects of overfitting, and the attendant need for regularization. The explanation for this fascinating phenomenon (observed as early as the late 90s) is the subject

of much current research; see [ZBH16], [BMM18], [BRT18], [SJL18], [ADH19], [BLL19], [HMR19], [MVS19], [SuY19], [Sun19], [HaR21], [VLK21], [ZBH21] for representative works.

We finally note that the use of deep neural networks has been an important factor for the success of the AlphaGo and AlphaZero programs, as well as for large language models. New developments in hardware, software, architectural structurea, and training methodology, are likely to improve the already enormous power of deep neural networks, and to allow the use of ever larger datasets, which are becoming increasingly available.

3.3 TRAINING OF COST FUNCTIONS IN APPROXIMATE DP

In the context of approximate DP/RL, architectures are mainly used to approximate either cost functions or policies. When a neural network is involved, the terms *value network* and *policy network* are commonly used, respectively.† In this section we will illustrate the use of value networks in finite horizon DP, while in the next section we will discuss the use of policy networks. We will also illustrate in Section 3.3.3 the combined use of policy and value networks within an approximate policy iteration context, whereby the policies and their cost functions are approximated by a policy and a value network, respectively, to generate a sequence of (approximately) improved policies. Finally, in Sections 3.3.4 and 3.4.5, we will describe how approximating Q-factor or cost differences (rather than Q-factors or costs) can be beneficial within our context of approximation in value space.

3.3.1 Fitted Value Iteration

Let us describe a popular approach for training an approximation architecture $\tilde{J}_k(x_k, r_k)$ for a finite horizon DP problem. The parameter vectors r_k are determined sequentially, starting from the end of the horizon, and proceeding backwards as in the DP algorithm: first r_{N-1} then r_{N-2}, and so on. The algorithm samples the state space for each stage k, and generates a large number of states x_k^s, $s = 1, \ldots, q$. It then determines sequentially the parameter vectors r_k to obtain a good "least squares fit" to the DP algorithm. The method can also be used in the infinite horizon case, in essentially identical form, and it is commonly called *fitted value iteration*.

In particular, each r_k is determined by generating a large number of sample states and solving a least squares problem that aims to minimize the error in satisfying the DP equation for these states at time k. At

† The alternative terms *critic network* and *actor network* are also used often. In this book, we will use the terms "value network" and "policy network."

the typical stage k, having obtained r_{k+1}, we determine r_k from the least squares problem

$$r_k \in \arg\min_r \sum_{s=1}^{q} \left(\tilde{J}_k(x_k^s, r) - \min_{u \in U_k(x_k^s)} E\Big\{ g_k(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k), r_{k+1}\big) \Big\} \right)^2$$

where x_k^s, $i = 1, \ldots, q$, are the sample states that have been generated for the kth stage. Since r_{k+1} is assumed to be already known, the complicated minimization term in the right side of this equation is the known scalar

$$\beta_k^s = \min_{u \in U_k(x_k^s)} E\Big\{ g_k(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k), r_{k+1}\big) \Big\}, \qquad (3.13)$$

so that r_k is obtained as

$$r_k \in \arg\min_r \sum_{s=1}^{q} \big(\tilde{J}_k(x_k^s, r) - \beta_k^s \big)^2. \qquad (3.14)$$

The algorithm starts at stage $N-1$ with the minimization

$$r_{N-1} \in \arg\min_r \sum_{s=1}^{q} \left(\tilde{J}_{N-1}(x_{N-1}^s, r) - \min_{u \in U_{N-1}(x_{N-1}^s)} E\Big\{ g_{N-1}(x_{N-1}^s, u, w_{N-1}) + g_N\big(f_{N-1}(x_{N-1}^s, u, w_{N-1})\big) \Big\} \right)^2$$

and ends with the calculation of r_0 at $k = 0$.

In the case of a linear architecture, where the approximate cost-to-go functions are

$$\tilde{J}_k(x_k, r_k) = r_k' \phi_k(x_k), \qquad k = 0, \ldots, N-1,$$

the least squares problem (3.14) greatly simplifies, and admits the closed form solution

$$r_k = \left(\sum_{s=1}^{q} \phi_k(x_k^s) \phi_k(x_k^s)' \right)^{-1} \sum_{s=1}^{q} \beta_k^s \phi_k(x_k^s);$$

cf. Eq. (3.7). For a nonlinear architecture such as a neural network, incremental gradient algorithms may be used.

An important implementation issue is how to select the sample states x_k^s, $s = 1, \ldots, q$, $k = 0, \ldots, N-1$. In practice, they are typically obtained

by some form of Monte Carlo simulation, but the distribution by which they are generated is important for the success of the method. In particular, it is important that the sample states are "representative" in the sense that they are visited often under a nearly optimal policy. More precisely, the frequencies with which various states appear in the sample should be roughly proportional to the probabilities of their occurrence under an optimal policy.

Aside from the issue of selection of the sampling distribution that we have just described, a difficulty with fitted value iteration arises when the horizon N is very long, since then the total number of parameters over the N stages may become excessive. In this case, however, the problem is often stationary, in the sense that the system and cost per stage do not change as time progresses. Then it may be possible to treat the problem as one with an infinite horizon and bring to bear additional methods for training approximation architectures; see the relevant discussions in Chapter 5 of the book [Ber19a].

We finally note an important difficulty with the training method of this section: the calculation of each sample β_k^s of Eq. (3.13) requires a minimization of an expected value, which can be very time consuming. In the next section, we describe an alternative type of fitted value iteration, which uses Q-factors, and involves a simpler minimization, whereby the order of the minimization and expectation operations in Eq. (3.13) is reversed.

3.3.2 Q-Factor Parametric Approximation - Model-Free Implementation

We will now consider an alternative form of approximation in value space and fitted value iteration, which involves approximation of the optimal Q-factors of state-control pairs (x_k, u_k) at time k, with no intermediate approximation of cost-to-go functions. An important characteristic of this algorithm is that it *allows for a model-free computation* (i.e., the use of a computer model in place of a mathematical model).

We recall that the optimal Q-factors are defined by

$$Q_k^*(x_k,u_k) = E\Big\{g_k(x_k,u_k,w_k)+J_{k+1}^*\big(f_k(x_k,u_k,w_k)\big)\Big\}, \quad k=0,\ldots,N-1, \tag{3.15}$$

where J_{k+1}^* is the optimal cost-to-go function for stage $k+1$. Thus $Q_k^*(x_k,u_k)$ is the cost attained by using u_k at state x_k, and subsequently using an optimal policy.

As noted in Section 1.3, the DP algorithm can be written as

$$J_k^*(x_k) = \min_{u\in U_k(x_k)} Q_k^*(x_k,u_k),$$

and by using this equation, we can write Eq. (3.15) in the following equivalent form that relates Q_k^* with Q_{k+1}^*:

$$Q_k^*(x_k,u_k) = E\Big\{ g_k(x_k,u_k,w_k) + \min_{u\in U_{k+1}(f_k(x_k,u_k,w_k))} Q_{k+1}^*\big(f_k(x_k,u_k,w_k),u\big)\Big\}. \tag{3.16}$$

This suggests that in place of the Q-factors $Q_k^*(x_k,u_k)$, we may use Q-factor approximations as the basis for suboptimal control.

We can obtain such approximations by using methods that are similar to the ones we have considered so far. Parametric Q-factor approximations $\tilde{Q}_k(x_k,u_k,r_k)$ may involve a neural network, or a feature-based linear architecture. The feature vector may depend on just the state, or on both the state and the control. In the former case, the architecture has the form

$$\tilde{Q}_k(x_k,u_k,r_k) = r_k(u_k)'\phi_k(x_k), \tag{3.17}$$

where $r_k(u_k)$ is a separate weight vector for each control u_k. In the latter case, the architecture has the form

$$\tilde{Q}_k(x_k,u_k,r_k) = r_k'\phi_k(x_k,u_k), \tag{3.18}$$

where r_k is a weight vector that is independent of u_k. The architecture (3.17) is suitable for problems with a relatively small number of control options at each stage. In what follows, we will focus on the architecture (3.18), but the discussion, with few modifications, also applies to the architecture (3.17) and to nonlinear architectures as well.

We may adapt the fitted value iteration approach of the preceding section to compute sequentially the parameter vectors r_k in Q-factor parametric approximations, starting from $k = N-1$. This algorithm is based on Eq. (3.16), with r_k obtained by solving least squares problems similar to the ones of the cost function approximation case [cf. Eq. (3.14)]. As an example, the parameters r_k of the architecture (3.18) are computed sequentially by collecting sample state-control pairs (x_k^s,u_k^s), $s=1,\ldots,q$, and solving the linear least squares problems

$$r_k \in \arg\min_r \sum_{s=1}^{q} \big(r'\phi_k(x_k^s,u_k^s) - \beta_k^s\big)^2, \tag{3.19}$$

where

$$\beta_k^s = E\left\{ g_k(x_k^s,u_k^s,w_k) + \min_{u\in U_{k+1}(f_k(x_k^s,u_k^s,w_k))} r_{k+1}'\phi_{k+1}\big(f_k(x_k^s,u_k^s,w_k),u\big)\right\}. \tag{3.20}$$

Thus, having obtained r_{k+1}, we obtain r_k through a least squares fit that aims to minimize the sum of the squared errors in satisfying Eq. (3.16). Note that the solution of the least squares problem (3.19) can be obtained in closed form as

$$r_k = \left(\sum_{s=1}^{q} \phi_k(x_k^s, u_k^s) \phi_k(x_k^s, u_k^s)' \right)^{-1} \sum_{s=1}^{q} \beta_k^s \phi_k(x_k^s, u_k^s);$$

[cf. Eq. (3.7)]. Once r_k has been computed, the one-step lookahead control $\tilde{\mu}_k(x_k)$ is obtained on-line as

$$\tilde{\mu}_k(x_k) \in \arg \min_{u \in U_k(x_k)} \tilde{Q}_k(x_k, u, r_k), \tag{3.21}$$

without the need to calculate any expected value. This latter property is a primary incentive for using Q-factors in approximate DP, particularly when there are tight constraints on the amount of on-line computation that is possible in the given practical setting.

The samples β_k^s of Eq. (3.20) involve the exact computation of an expected value. In an alternative implementation, we may replace β_k^s with an average of just a few samples (even a single sample) of the random variable

$$g_k(x_k^s, u_k^s, w_k) + \min_{u \in U_{k+1}(f_k(x_k^s, u_k^s, w_k))} r'_{k+1} \phi_{k+1}\big(f_k(x_k^s, u_k^s, w_k), u\big), \tag{3.22}$$

collected according to the probability distribution of w_k. This distribution may either be known explicitly, or in a model-free situation, through a computer simulator. In particular, *to implement this scheme, we only need a simulator that for any pair (x_k, u_k) generates a sample of the stage cost $g_k(x_k, u_k, w_k)$ and the next state $f_k(x_k, u_k, w_k)$ according to the distribution of w_k.*

Note that the samples of the random variable (3.22) do not require the computation of an expected value like the samples (3.13) in the cost approximation method of the preceding chapter. Moreover the samples of (3.22) involve a simpler minimization than the samples (3.13). This is an important advantage of working with Q-factors rather than state costs.

Having obtained the weight vectors $r_0, \ldots, r_{N-1}$, and hence the one-step lookahead policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ through Eq. (3.21), a further possibility is to approximate this policy with a parametric architecture. This is *approximation in policy space built on top of approximation in value space*. The idea here is to simplify even further the on-line computation of the suboptimal controls by avoiding the minimization of Eq. (3.21).

3.3.3 Parametric Approximation in Infinite Horizon Problems - Approximate Policy Iteration

In this section we will briefly discuss parametric approximation methods for infinite horizon problems, based on the policy iteration (PI) method. We will focus on the finite-state version of the α-discounted problem of Section 1.4.1, and adopt notation that is more convenient for such problems. In particular, states and successor states will be denoted by i and j, respectively. Moreover the system equation will be represented by control-dependent transition probabilities $p_{ij}(u)$ (the probability that the system will move to state j, given that it starts at state i and control u is applied). For a state-control pair (i, u), the average cost per stage is denoted by $g(i, u, j)$.

We recall that the PI algorithm in its exact form produces a sequence of stationary policies whose cost functions are progressively improving and converge in a finite number of iterations to the optimal. The corresponding convergence proof relies on the generic cost improvement property of PI, and depends on the finiteness of the state and control spaces. This proof, together with other PI-related convergence proofs, can be found in the author's textbooks [Ber17a] or [Ber19a].

Let us state the exact form of the PI algorithm in terms of Q-factors, and in a form that is suitable for the use of approximations and simulation-based implementations. Given any policy μ, it generates the next policy $\tilde{\mu}$ with a two-step process as follows (cf. Section 1.4.1):

(a) *Policy evaluation*: We compute the cost function J_μ of μ and its associated Q-factors, which are given by

$$Q_\mu(i,u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_\mu(j)\big), \qquad i = 1, \ldots, n, \ u \in U(i).$$

Thus $Q_\mu(i, u)$ is the cost of starting at i, using u at the first stage, and then using μ for the remaining stages.

(b) *Policy improvement*: We compute the new policy $\tilde{\mu}$ according to

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} Q_\mu(i,u), \qquad i = 1, \ldots, n.$$

Let us now describe one way to approximate the two steps of the preceding process.

(a) *Approximate policy evaluation*: Here we introduce a parametric architecture $\tilde{Q}_\mu(i, u, r)$ for the Q-factors of μ. We determine the value of the parameter vector r by generating (using a simulator of the system) a large number of training triplets (i^s, u^s, β^s), $s = 1, \ldots, q$, and

by using a least squares fit:

$$\overline{r} \in \arg\min_{r} \sum_{s=1}^{q} \big(\tilde{Q}_\mu(i^s, u^s, r) - \beta^s\big)^2. \tag{3.23}$$

In particular, for a given pair (i^s, u^s), the scalar β^s is generated by starting at i^s, using u^s at the first stage, and simulating a trajectory of states and controls using μ for some number k of subsequent stages. Thus, β^s is a sample of $Q^k_\mu(i^s, u^s)$, the k-stage Q-factor of μ, which in the limit as $k \to \infty$ yields the infinite horizon Q-factor of μ. The number of stages k may be either large, or fairly small. However, in the latter case some terminal cost function approximation should be added at the end of the k-stage trajectory, to compensate for the difference $\big|Q_\mu(i,u) - Q^k_\mu(i,u)\big|$, which decreases in proportion to α^k, and may be large when k is small. Such a function may be obtained with additional training or from a previous iteration.

(b) *Approximate policy improvement*: Here we compute the new policy $\tilde{\mu}$ according to

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \tilde{Q}_\mu(i, u, \overline{r}), \qquad i = 1, \ldots, n, \tag{3.24}$$

where $\overline{r}$ is the parameter vector obtained from the policy evaluation formula (3.23).

An important alternative for approximate policy improvement, is to compute a set of pairs $\big(i^s, \tilde{\mu}(i^s)\big)$, $s = 1, \ldots, q$, using Eq. (3.24), and fit these pairs with a policy approximation architecture (see the next section on approximation in policy space). The overall scheme then becomes policy iteration that is based on approximation in both value and policy spaces.

At the end of the last policy evaluation step of PI, we have obtained a final Q-factor approximation $\tilde{Q}(i, u, \tilde{r})$. Then, in on-line play mode, we may apply the policy

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \tilde{Q}(i, u, \tilde{r}),$$

i.e., use the (would be) next policy iterate. Alternatively, we may apply the one-step lookahead policy

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \left[\sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \min_{u' \in U(j)} \tilde{Q}(j, u', \tilde{r})\right)\right], \tag{3.25}$$

or its multistep lookahead version. The latter alternative implements a Newton step and will likely result in substantially better performance.

However, it is more time consuming, particularly if it is implemented by using a computer model and model-free simulation. Still another possibility, which also implements a Newton step, is to replace the function

$$\min_{u' \in U(j)} \tilde{Q}(j, u', \tilde{r})$$

in the preceding Eq. (3.25) with an off-line trained approximation.

Challenges Relating to Approximate Policy Iteration

Approximate PI in its various forms has been the subject of extensive research, both theoretical and applied. A more detailed discussion is beyond our scope, and we refer to the literature, including the DP textbook [Ber12] or the RL textbook [Ber19a], for detailed accounts. Let us provide a few comments relating to the challenges of approximate PI implementation.

(a) *Architectural issues*: The architecture $\tilde{Q}_\mu(i, u, r)$ may involve the use of features, and it could be linear, or it could be nonlinear such as a neural network. A major advantage of a linear feature-based architecture is that the policy evaluation (3.23) is a linear least squares problem, which admits a closed-form solution. Moreover, when linear architectures are used, there is a broader variety of approximate policy evaluation methods with solid theoretical performance guarantees, such as TD(λ), LSTD(λ), and LSPE(λ), which are not described in this book, but are discussed extensively in the literature. Generally, finding an architecture that matches well the problem at hand, and properly training it, can be a difficult and time-consuming task.

(b) *Exploration issues*: Generating an appropriate set of training triplets (i^s, u^s, β^s) at the policy evaluation step poses considerable challenges. A generic difficulty has to do with *inadequate exploration*, the Achilles heel of approximate PI. In particular, to evaluate a policy μ, we may need to generate Q-factor samples of μ starting from states frequently visited by μ, but this may bias the simulation by underrepresenting states that are unlikely to occur under μ. As a result, the Q-factor estimates of these underrepresented states may be highly inaccurate, causing potentially serious errors in the calculation of the improved control policy $\tilde{\mu}$ via the policy improvement Eq. (3.24).

One possibility to improve the exploration of the state space is to use a large number of initial states to form a rich and representative subset. It may then be necessary to use relatively short trajectories to keep the cost of the simulation low. However, when using short trajectories it may be important to introduce a terminal cost function approximation in the policy evaluation step in order to make the cost sample β^s more accurate. There have been other related approaches to improve exploration, such as using a so-called *off-policy*, i.e., a

policy μ' other than the currently evaluated policy μ, to visit states that are unlikely to be visited using μ. See the discussions in Section 6.4 of the DP textbook [Ber12].

(c) *Oscillation issues*: Contrary to exact PI, which is guaranteed to yield an optimal policy, approximate PI produces a sequence of policies, which are only guaranteed to lie asymptotically within a certain error bound from the optimal; see the books [BeT96], Section 6.2.2, and [Ber12], Section 2.5. Moreover, the generated policies may oscillate. By this we mean that after a few iterations, policies tend to repeat in cycles.

The associated parameter vectors r may also tend to oscillate, although it is possible that there is convergence in parameter space and oscillation in policy space. This phenomenon, known as *chattering*, is explained in the author's survey papers [Ber10c], [Ber11b], and book [Ber12] (Section 6.4.3), and can be particularly damaging, because there is no guarantee that the policies involved in the oscillation are "good" policies, and there is often no way to verify how well they perform relative to the optimal. We note, however, that oscillations can be avoided and approximate PI can be shown to converge under special conditions, which arise in particular when an aggregation approach is used; see the approximate PI survey [Ber11b].

We refer to the literature for further discussion of the preceding issues, as well as a variety of other approximate PI methods.

3.3.4 Optimistic Policy Iteration with Parametric Q-Factor Approximation - SARSA and DQN

There are also "optimistic" approximate PI methods with Q-factor approximation, and/or a few samples in between policy updates. Because of the use of Q-factors and the limited number of samples between policy updates, these schemes have the potential of on-line play implementation, but a number of difficulties must be overcome in this case, as we will explain later in this section.

As an example, let us consider an extreme version of Q-factor parametric approximation that uses *a single sample* between policy updates. At the start of iteration k, we have the current parameter vector r^k, we are at some state i^k, and we have chosen a control u^k. Then:

(1) We simulate the next transition (i^k, i^{k+1}) using the transition probabilities $p_{i^k j}(u^k)$.

(2) We generate the control u^{k+1} with the minimization

$$u^{k+1} \in \arg \min_{u \in U(i^{k+1})} \tilde{Q}(i^{k+1}, u, r^k). \tag{3.26}$$

[In some schemes, to enhance exploration, u^{k+1} is chosen with a small probability to be a random element of $U(i^{k+1})$ or one that is "ϵ-greedy," i.e., attains within some ϵ the minimum above. This is commonly referred to as the use of an *off-policy*.]

(3) We update the parameter vector via

$$r^{k+1} = r^k - \gamma^k \nabla \tilde{Q}(i^k, u^k, r^k) \cdot \big(\tilde{Q}(i^k, u^k, r^k) - g(i^k, u^k, i^{k+1}) - \alpha \tilde{Q}(i^{k+1}, u^{k+1}, r^k)\big), \tag{3.27}$$

where γ^k is a positive stepsize, and $\nabla(\cdot)$ denotes gradient with respect to r evaluated at the current parameter vector r^k. To get a sense for the rationale of this iteration, note that if $\tilde{Q}$ is a linear feature-based architecture, $\tilde{Q}(i, u, r) = \phi(i, u)'r$, then $\nabla \tilde{Q}(i^k, u^k, r^k)$ is just the feature vector $\phi(i^k, u^k)$, and iteration (3.27) becomes

$$r^{k+1} = r^k - \gamma^k \phi(i^k, u^k)\big(\phi(i^k, u^k)'r^k - g(i^k, u^k, i^{k+1}) - \alpha \phi(i^{k+1}, u^{k+1})'r^k\big).$$

Thus r^k *is changed in an incremental gradient direction*: the one opposite to the gradient (with respect to r) of the incremental error

$$\big(\phi(i^k, u^k)'r - g(i^k, u^k, i^{k+1}) - \alpha \phi(i^{k+1}, u^{k+1})'r^k\big)^2,$$

evaluated at the current iterate r^k.

The process is now repeated with r^{k+1}, i^{k+1}, and u^{k+1} replacing r^k, i^k, and u^k, respectively.

Extreme optimistic schemes of the type just described have received a lot of attention, in part because *they admit a model-free implementation* [i.e., the use of a computer simulator, which provides for each pair (i^k, u^k), the next state i^{k+1} and corresponding cost $g(i^k, u^k, i^{k+1})$ that are needed in Eq. (3.27)]. They are often referred to as SARSA (State-Action-Reward-State-Action); see e.g., the books [BeT96], [BBD10], [SuB18]. When Q-factor approximation is used, their behavior is very complex, their theoretical convergence properties are unclear, and there are no associated performance bounds in the literature. In practice, SARSA is more commonly used in a less extreme/optimistic form, whereby several (perhaps many) state-control-transition cost-next state samples are batched together and suitably averaged before updating the vector r^k.

Other variants of the method attempt to save in sampling effort by storing the generated samples in a buffer and reusing them in some randomized fashion in subsequent iterations (cf. our earlier discussion of exploration. This is also called sometimes *experience replay*, an idea that has been used since the early days of RL, both to save in sampling effort and to enhance exploration. The DQN (Deep Q Network) scheme, championed by DeepMind (see Mnih et al. [MKS15]), is based on this idea (the term "Deep" is a reference to DeepMind's affinity for deep neural networks, but experience replay does not depend on the use of a deep neural network architecture).

Q-Learning Algorithns and On-Line Play

Algorithms that approximate Q-factors, including SARSA and DQN, are fundamentally off-line training algorithms, primarily because their training process is long and requires the collection of many samples before reaching a stage that resembles parameter convergence. It can therefore be unreliable to use the interim approximate Q-factors for on-line decision making, particularly in an adaptive context that involves changing system parameters, thereby requiring on-line replanning.

On the other hand, compared to the approximate PI method of Section 3.3.3, SARSA and DQN are far better suited for on-line implementation, because the control generation process of Eq. (3.26) can also be used to select controls on-line, thereby facilitating the combination of training and on-line control selection. To this end, it is important, among others, to make sure that the parameters r_k stay at "safe" levels during the on-line control process, which can be a challenge. Still, even if this difficulty can be overcome in the context of a given problem, there are a number of other difficulties that SARSA and DQN can encounter during on-line play.

(a) *On-line exploration issues*: The need to occasionally select controls using an off-policy in order to enhance exploration. Finding an off-line policy that adequately deals with exploration in a given practical context can be a challenge. Moreover, an additional concern is that the off-policy controls may improve exploration, but may be of poor quality, and in some contexts, may induce instability.

(b) *Robustness and replanning issues*: In an adaptive control context where the problem parameters are changing, the algorithm may be too slow to adapt to the changes.

(c) *Performance degradation issues*: Similar to our earlier discussion [cf. the comparison of Eqs. (3.24) and (3.25)], the minimization of Eq. (3.26) does not implement a Newton step, thereby resulting in performance loss. The alternative implementation

$$u^{k+1} \in \arg\min_{u \in U(i^{k+1})} \left[\sum_{j=1}^{n} p_{i^{k+1}j}(u) \Big(g(i^{k+1}, u, j) + \alpha \min_{u' \in U(j)} \tilde{Q}(j, u', r^k) \Big) \right],$$

which is patterned after Eq. (3.26), is better in this regard, but is computationally more costly, and thus less suitable for on-line implementation.

Generally speaking, the combination of off-line training and on-line play with the use of SARSA and DQN involves serious challenges. However,

in some specific contexts encouraging results have been obtained, often by "manual tuning" to the problem at hand. Moreover, the methods have received a lot of attention, thanks in part to the availability of publicly available software, which also allow for a model-free implementation.

3.3.5 Approximate Policy Iteration for Infinite Horizon POMDP

In this section, we consider partial observation Markovian decision problems (POMDP) with a finite number of states and controls, and discounted additive cost over an infinite horizon. As discussed in Section 1.6.4, the optimal solution is typically intractable, so approximate DP/RL approaches must be used. In this section we focus on PI methods that are based on rollout, and approximations in policy and value space. They update a policy by using truncated rollout with that policy and a terminal cost function approximation. We focus on cost function approximation schemes, but Q-factor approximation is also possible.

Because of its simulation-based rollout character, the methodology of this section depends critically on the finiteness of the control space. It can be extended to POMDP with infinite state space but finite control space, although we will not consider this possibility in this section. In particular, we assume that there are n states denoted by $i \in \{1, \ldots, n\}$ and a finite set of controls U at each state. We denote by

$$p_{ij}(u) \quad \text{and} \quad g(i, u, j)$$

the transition probabilities and corresponding transition costs, from i to j under $u \in U$. The cost is accumulated over an infinite horizon and is discounted by $\alpha \in (0, 1)$. At each new generated state j, an observation z from a finite set Z is obtained with known probability $p(z \mid j, u)$ that depends on j and the control u that was applied prior to the generation of j. The objective is to select each control optimally as a function of the prior history of observations and controls.

A classical approach to this problem is to convert it to a perfect state information problem whose state is the current belief $b = (b_1, \ldots, b_n)$, where b_i is the conditional distribution of the state i given the prior history. As noted in Section 1.6.4, b is a sufficient statistic, which can serve as a substitute for the set of available observations, in the sense that optimal control can be achieved with knowledge of just b.

In this section, we consider a more general form of sufficient statistic, which we call the *feature state* and we denote by y. *We require that the feature state y subsumes the belief state b.* By this we mean that b can be computed exactly knowing y. One possibility is for y to be the union of b and some identifiable characteristics of the belief state, or some compact representation of the measurement history up to the current time (such as a number of most recent measurements, or the state of a finite-state controller). We also make the additional assumption that y *can be*

sequentially generated using a known feature estimator $F(y, u, z)$. By this we mean that given that the current feature state is y, control u is applied, and observation z is obtained, the next feature can be exactly predicted as $F(y, u, z)$.

Clearly, since b is a sufficient statistic, the same is true for y. Thus the optimal costs achievable by the policies that depend on y and on b are the same. However, specific suboptimal schemes may become more effective with the use of the feature state y instead of just the belief state b.

The optimal cost $J^*(y)$, as a function of the sufficient statistic/feature state y, is the unique solution of the corresponding Bellman equation

$$J^*(y) = \min_{u \in U} \left[\hat{g}(y, u) + \alpha \sum_{z \in Z} \hat{p}(z \mid b_y, u) J^*\big(F(y, u, z)\big) \right].$$

Here we use the following notation:

b_y is the belief state that corresponds to feature state y, with components denoted by $b_{y,i}$, $i = 1, \ldots, n$.

$\hat{g}(y, u)$ is the expected cost per stage

$$\hat{g}(y, u) = \sum_{i=1}^{n} b_{y,i} \sum_{j=1}^{n} p_{ij}(u) g(i, u, j).$$

$\hat{p}(z \mid b_y, u)$ is the conditional probability that the next observation will be z given the current belief state b_y and control u

F is the feature state estimator. In particular, $F(y, u, z)$ is the next feature vector, when the current feature state is y, control u is applied, and observation z is obtained.

The feature space reformulation of the problem can serve as the basis for approximation in value space, whereby J^* is replaced in Bellman's equation by some function $\tilde{J}$ after one-step or multistep lookahead. For example a one-step lookahead scheme yields the suboptimal policy $\tilde{\mu}$ given by

$$\tilde{\mu}(y) \in \arg\min_{u \in U} \left[\hat{g}(y, u) + \alpha \sum_{z \in Z} \hat{p}(z \mid b_y, u) \tilde{J}\big(F(y, u, z)\big) \right].$$

In ℓ-step lookahead schemes, $\tilde{J}$ is used as terminal cost function in an ℓ-step horizon version of the original infinite horizon problem. In the standard form of a rollout algorithm, $\tilde{J}$ is the cost function of some base policy. We will next discuss a rollout scheme with ℓ-step lookahead, which involves rollout truncation and terminal cost approximation.

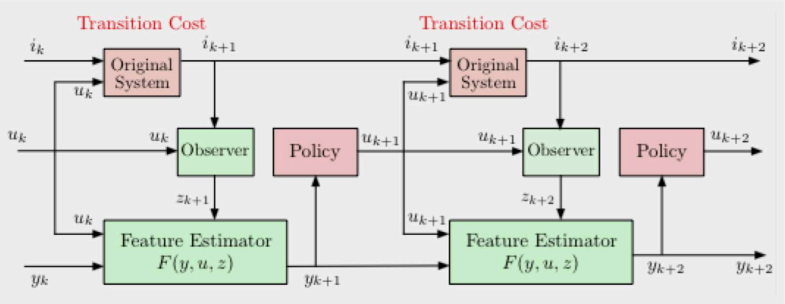

Figure 3.3.1 Composite system simulator for POMDP for a given policy. The starting state i_k at stage k of a trajectory is generated randomly using the belief state b_k, which is in turn computed from the feature state y_k.

Truncated Rollout with Terminal Cost Function Approximation

In the pure form of the rollout algorithm, the cost function approximation $\tilde{J}$ is the cost function J_μ of a known base policy μ, and its value $\tilde{J}(y) = J_\mu(y)$ at any y is obtained by first extracting b from y, and then running a simulator starting from b, and using the system model, the feature generator, and μ. In the truncated form of rollout, $\tilde{J}(y)$ is obtained by running the simulator of μ for a given number of steps m, and then adding a terminal cost approximation $\hat{J}(\bar{y})$ for each terminal feature state $\bar{y}$ that is obtained at the end of the m steps of the simulation with μ (see Fig. 3.3.1).

Thus the rollout policy is defined by the base policy μ, the terminal cost function approximation $\hat{J}$, the number of steps m after which the simulated trajectory with μ is truncated, as well as the lookahead size ℓ. The choices of m and ℓ are typically made by trial and error, based on computational tractability among other considerations, while $\hat{J}$ may be chosen on the basis of problem-dependent insight or through the use of some off-line approximation method. In variants of the method, the multistep lookahead may be implemented approximately using a Monte Carlo tree search or adaptive sampling scheme.

Using m-step rollout between the ℓ-step lookahead and the terminal cost approximation gives the method the character of a single PI. We will use repeated truncated rollout as the basis for constructing a PI algorithm, which we will discuss next.

Supervised Learning of Rollout Policies and Cost Functions - Approximate Policy Iteration

The rollout algorithm uses multistep lookahead and on-line simulation of the base policy to generate the rollout control at any feature state of interest. To avoid the cost of on-line simulation, we can approximate the rollout

policy off-line by using some approximation architecture, which may involve a neural network. This is policy approximation built on top of the rollout scheme.

To this end, we may introduce a parametric family/architecture of policies of the form $\hat{\mu}(y, r)$, where r is a parameter vector. We then construct a training set that consists of a large number of sample feature state-control pairs (y^s, u^s), $s = 1, \ldots, q$, such that for each s, u^s is the rollout control at feature state y^s. We use this data set to obtain a parameter $\bar{r}$ by solving a corresponding classification problem, which associates each feature state y with a control $\hat{\mu}(y, \bar{r})$. The parameter $\bar{r}$ defines a classifier, which given a feature state y, classifies y as requiring control $\hat{\mu}(y, \bar{r})$ (see Section 3.4).

We can also apply the rollout policy approximation to the context of PI. The idea is to view rollout as a single policy improvement, and to view the PI algorithm as a *perpetual rollout process*, which performs multiple policy improvements, using at each iteration the current policy as the base policy, and the next policy as the corresponding rollout policy.

In particular, we consider a PI algorithm where at the typical iteration we have a policy μ, which we use as the base policy to generate many feature state-control sample pairs (y^s, u^s), $s = 1, \ldots, q$, where u^s is the rollout control corresponding to feature state y^s. We then obtain an "improved" policy $\hat{\mu}(y, \overline{r})$ with an approximation architecture and a classification algorithm, as described above. The "improved" policy is then used as a base policy to generate samples of the corresponding rollout policy, which is approximated in policy space, etc.

To use truncated rollout in this PI scheme, we must also provide a terminal cost approximation, which may take a variety of forms. Using zero is a simple possibility, which may work well if either the size ℓ of multistep lookahead or the length m of the rollout is relatively large. Another possibility is to use as terminal cost in the truncated rollout an approximation of the cost function of some base policy, which may be obtained with a neural network-based approximation architecture.

In particular, at any policy iteration with a given base policy, once the rollout data is collected, one or two neural networks are constructed: A policy network that approximates the rollout policy, and (in the case of rollout with truncation) a value network that constructs a cost function approximation for that rollout policy. Thus, we may consider two types of methods:

(a) *Approximate rollout and PI with truncation*, where each generated policy as well as its cost function are approximated by a policy and a value network, respectively. The cost function approximation of the current policy is used to truncate the rollout trajectories that are used to train the next policy.

(b) *Approximate rollout and PI without truncation*, where each gener-

ated policy is approximated using a policy network, but the rollout trajectories are continued up to a large maximum number of stages (enough to make the cost of the remaining stages insignificant due to discounting) or upon reaching a termination state. The advantage of this scheme is that only a policy network is needed; a value network is unnecessary since there is no rollout truncation with cost function approximation at the end.

Note that as in all approximate PI schemes, the sampling of feature states used for training is subject to exploration concerns. In particular, for each policy approximation, it is important to include in the sample set $\{y^s \mid s = 1, \ldots, q\}$, a subset of feature states that are "favored" by the rollout trajectories; e.g., start from some initial subset of feature states y^s and selectively add to this subset feature states that are encountered along the rollout trajectories. This is a challenging issue, which must be approached with care.

An extensive case study of the methodology of this section was given in the paper by Bhattacharya et al. [BBW20], for the case of a pipeline repair problem. The implementation used there also includes the use of a partitioned state space architecture and an asynchronous distributed algorithm for off-line training; see Section 3.4.2.

3.3.6 Advantage Updating - Approximating Q-Factor Differences

Let us now focus on an important alternative to computing Q-factor approximations. It is motivated by the potential benefit of approximating Q-factor differences rather than Q-factors. In this method, called *advantage updating*, instead of computing and comparing $Q_k^*(x_k, u_k)$ for all $u_k \in U_k(x_k)$, we compute

$$A_k(x_k, u_k) = Q_k^*(x_k, u_k) - \min_{u \in U_k(x_k)} Q_k^*(x_k, u).$$

The function $A_k(x_k, u_k)$ can serve to compare controls, i.e., at state x_k select

$$\tilde{\mu}_k(x_k) \in \arg \min_{u \in U_k(x_k)} A_k(x_k, u),$$

and this can also be done when $A_k(x_k, u_k)$ is approximated with a value network.

Note that in the absence of approximations, selecting controls by advantage updating is clearly equivalent to selecting controls by comparing their Q-factors. By contrast, when approximation is involved, comparing advantages instead of Q-factors can be important, because the former may have a much smaller range of values than the latter. In particular, Q_k^* may embody sizable quantities that depend on x_k but are independent of u_k, and which may interfere with algorithms such as the fitted value iteration

(3.19)-(3.20). Thus, when training an architecture to approximate Q_k^*, the training algorithm may naturally try to capture the large scale behavior of Q_k^*, which may be irrelevant because it may not be reflected in the Q-factor differences A_k. However, with advantage updating, we may instead focus the training process on finer scale variations of Q_k^*, which may be all that matters. Here is an example (first given in the book [BeT96]) of what can happen when trained approximations of Q-factors are used.

Example 3.3.1

Consider the deterministic scalar linear system

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

and the quadratic cost per stage

$$g(x_k, u_k) = \delta(x_k^2 + u_k^2),$$

where δ is a very small positive constant [think of δ-discretization of a continuous-time problem involving the differential equation $dx(t)/dt = u(t)$]. Let us focus on the stationary policy π, which applies at state x the control

$$\mu(x) = -2x,$$

and view it as the base policy of a rollout algorithm. The Q-factors of π over an infinite number of stages can be calculated to be

$$Q_\pi(x, u) = \frac{5x^2}{4} + \delta\left(\frac{9x^2}{4} + u^2 + \frac{5}{2}xu\right) + O(\delta^2).$$

(We omit the details of this calculation, which is based on the classical analysis of linear-quadratic optimal control problems; see e.g., Section 1.5, or [Ber17a], Section 3.1.) Thus the important part of $Q_\pi(x, u)$ for the purpose of rollout policy computation is

$$\delta\left(u^2 + \frac{5}{2}xu\right). \tag{3.28}$$

However, when a value network is trained to approximate $Q_\pi(x, u)$, the approximation will be dominated by $\frac{5x^2}{4}$, and the important part (3.28) will be "lost" when δ is very small. By contract, the advantage function can be calculated to be

$$\begin{aligned} A_\mu(x, u) &= Q_\pi(x, u) - \min_v Q_\pi(x, v) + O(\delta^2) \\ &= \delta\left(u^2 + \frac{5}{2}xu - \min_v \left(v^2 + \frac{5}{2}xv\right)\right) + O(\delta^2) \\ &= \delta\left(u^2 + \frac{5}{2}xu + \frac{5^2}{4}x^2\right) + O(\delta^2), \end{aligned}$$

and when approximated with a value network, the approximation will be essentially unaffected by δ.

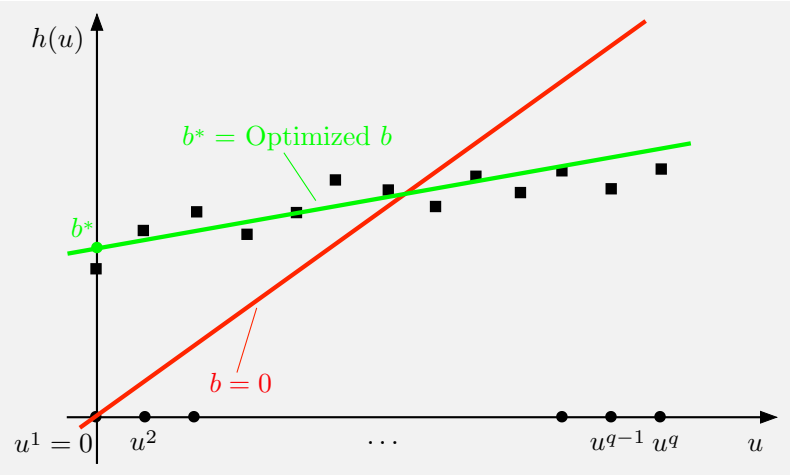

Figure 3.3.2 Illustration of the idea of subtracting a baseline constant from a cost or Q-factor approximation. Here we have samples $h(u^1), \ldots, h(u^q)$ of a scalar function $h(u)$ at sample points $u^1, \ldots, u^q$, and we want to approximate $h(u)$ with a linear function $\tilde{h}(u, r) = ru$, where r is a scalar tunable weight. We subtract a baseline constant b from the samples, and we solve the problem

$$\bar{r} \in \arg\min_r \sum_{s=1}^{q} \Big(\big(h(u^s) - b \big) - ru^s \Big)^2.$$

By properly adjusting b, we can improve the quality of the approximation, which after subtracting b from all the sample values, takes the form $\tilde{h}(u, b, r) = b + ru$. Conceptually, b serves as an additional weight (multiplying the basis function 1), which enriches the approximation architecture.

The Use of a Baseline

The idea of advantage updating is also related to the useful technique of subtracting a suitable constant (often called a *baseline*) from a quantity that is estimated; see Fig. 3.3.2 (in the case of advantage updating, the baselines depend on x_k, but the same general idea applies). This idea can also be used in the context of the fitted value iteration method given earlier, as well as in conjunction with other simulation-based methods in RL.

Example 3.1.1 also points to the connection between the ideas underlying advantage updating and the rollout methods for small stage costs relative to the cost function approximation, which we discussed in Section 2.6. In both cases it is necessary to avoid including terms of disproportionate size in the target function that is being approximated. The remedy in both cases is to subtract from the target function a suitable state-dependent baseline.

3.3.7 Differential Training of Cost Differences for Rollout

Let us now consider ways to approximate Q-factor differences (cf. our advantage updating discussion of the preceding section) by approximating cost function differences first. We recall here that given a base policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, the off-line computation of an approximate rollout policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ consists of two steps:

(1) In a preliminary phase, we compute approximations $\tilde{J}_k$ to the cost functions $J_{k,\pi}$ of the base policy π, possibly using simulation and a least squares fit from a parametrized class of functions.

(2) Given $\tilde{J}_k$ and a state x_k at time k, we compute the approximate Q-factor

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

for all $u \in U_k(x_k)$, and we obtain the (approximate) rollout control $\tilde{\mu}_k(x_k)$ from the minimization

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

Unfortunately, this method also suffers from the error magnification inherent in the Q-factor differencing operation. This motivates an alternative approach, called *differential training*, which is based on cost-to-go difference approximations. To this end, we note that to compute the rollout control $\tilde{\mu}_k(x_k)$, it is sufficient to have the differences of costs-to-go

$$\tilde{J}_{k+1}\big(f_k(x_k, u, w_k)\big) - \tilde{J}_{k+1}\big(f_k(x_k, \mu_k(x_k), w_k)\big), \tag{3.29}$$

where $\mu_k(x_k)$ is the control applied by the base policy at x_k.

We thus consider a function approximation approach, whereby given any two states x_{k+1} and $\hat{x}_{k+1}$, we obtain an approximation $\tilde{G}_{k+1}(x_{k+1}, \hat{x}_{k+1})$ of the cost difference (3.29). We then compute the rollout control by

$$\tilde{\mu}_k(x_k) \in \arg\min_{u \in U_k(x_k)} E\Big\{g_k(x_k, u, w_k) - g_k(x_k, \mu_k(x_k), w_k) + \tilde{G}_{k+1}\big(f_k(x_k, u, w_k), f_k(x_k, \mu_k(x_k), w_k)\big)\Big\}, \tag{3.30}$$

where $\mu_k(x_k)$ is the control applied by the base policy at x_k. Note that the minimization (3.30) aims to simply subtract the approximate Q-factor of the base policy control $\mu_k(x_k)$ from the approximate Q-factor of every other control $u \in U_k(x_k)$.

An important point here is that the training of an approximation architecture to obtain $\tilde{G}_{k+1}$ can be done using any of the standard training

methods, and a "differential" system, whose "states" are pairs $(x_k, \hat{x}_k)$ and will be described shortly. To see this, let us denote for all k and pair of states $(x_k, \hat{x}_k)$

$$G_k(x_k, \hat{x}_k) = J_{k,\pi}(x_k) - J_{k,\pi}(\hat{x}_k)$$

the cost function differences corresponding to the base policy π. We consider the DP equations corresponding to π, and to x_k and $\hat{x}_k$:

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

$$J_{k,\pi}(\hat{x}_k) = E\Big\{g_k\big(\hat{x}_k, \mu_k(\hat{x}_k), w_k\big) + J_{k+1,\pi}\big(f_k(\hat{x}_k, \mu_k(\hat{x}_k), w_k)\big)\Big\},$$

and we subtract these equations to obtain

$$\begin{aligned} G_k(x_k, \hat{x}_k) = E\Big\{&g_k\big(x_k, \mu_k(x_k), w_k\big) - g\big(\hat{x}_k, \mu_k(\hat{x}_k), w_k\big) \\ &+ G_{k+1}\big(f_k(x_k, \mu_k(x_k), w_k), f_k(\hat{x}_k, \mu_k(\hat{x}_k), w_k)\big)\Big\}, \end{aligned}$$

for all $(x_k, \hat{x}_k)$ and k. Therefore, G_k can be viewed as the cost-to-go function for a problem involving a fixed policy (the base policy), the state $(x_k, \hat{x}_k)$, the cost per stage

$$g_k\big(x_k, \mu_k(x_k), w_k\big) - g_k\big(\hat{x}_k, \mu_k(\hat{x}_k), w_k\big), \tag{3.31}$$

and the system equation

$$(x_{k+1}, \hat{x}_{k+1}) = \Big(f_k\big(x_k, \mu_k(x_k), w_k\big), f_k\big(\hat{x}_k, \mu_k(\hat{x}_k), w_k\big)\Big). \tag{3.32}$$

Thus, it can be seen that *any of the standard methods that can be used to train architectures that approximate $J_{k,\pi}$, can also be used for training architectures that approximate G_k*. For example, one may use simulation-based methods that generate pairs of trajectories starting at the pair of initial states $(x_k, \hat{x}_k)$, and generated according to Eq. (3.32) by using the base policy π. Note that a single random sequence $\{w_0, \ldots, w_{N-1}\}$ may be used to simultaneously generate samples of $G_k(x_k, \hat{x}_k)$ for several triples $(x_k, \hat{x}_k, k)$, and in fact this may have a substantial beneficial effect.

A special case of interest arises when a linear, feature-based architecture is used for the approximator $\tilde{G}_k$. In particular, let ϕ_k be a feature extraction mapping that associates a feature vector $\phi_k(x_k)$ with state x_k and time k, and let $\tilde{G}_k$ be of the form

$$\tilde{G}_k(x_k, \hat{x}_k) = r_k'\big(\phi_k(x_k) - \phi_k(\hat{x}_k)\big),$$

where r_k is a tunable weight vector of the same dimension as $\phi_k(x_k)$ and prime denotes transposition. The rollout policy is generated by

$$\tilde{\mu}_k(x_k) \in \arg\min_{u \in U_k(x_k)} E\Big\{g_k(x_k, u, w_k) + r_{k+1}'\phi_{k+1}\big(f_k(x_k, u, w_k)\big)\Big\},$$

which corresponds to using $r'_{k+1}\phi_{k+1}(x_{k+1})$ (plus an unknown inconsequential constant) as an approximation to $J_{k+1,\pi}(x_{k+1})$. Thus, in this approach, *we essentially use a linear feature-based architecture to approximate the cost functions $J_{k,\pi}$ of the base policy, but we train this architecture using the differential system (3.32) and the differential cost per stage of Eq. (3.31).* This is done by selecting pairs of initial states, running in parallel the corresponding trajectories using the base policy, and subtracting the resulting trajectory costs from each other.

3.4 TRAINING OF POLICIES IN APPROXIMATE DP

We have focused so far on approximation in value space using parametric architectures. In this section we will discuss briefly how the cost function approximation methods of this chapter can be suitably adapted for the purpose of approximation in policy space, whereby we select the policy by using optimization over a parametric family of some form.

In particular, suppose that for a given stage k, we have access to a dataset of sample state-control pairs (x_k^s, u_k^s), $s = 1, \ldots, q$, obtained through some unspecified process, such as rollout or problem approximation. We may then wish to "learn" this process by training the parameter vector r_k of a parametric family of policies $\tilde{\mu}_k(x_k, r_k)$, using least squares minimization/regression:

$$\overline{r}_k \in \arg\min_{r_k} \sum_{s=1}^{q} \left\| u_k^s - \tilde{\mu}_k(x_k^s, r_k) \right\|^2; \tag{3.33}$$

cf. our discussion of approximation in policy space in Section 1.3.3.

3.4.1 The Use of Classifiers for Approximation in Policy Space

As we have noted in Section 3.1, in the case of a continuous control space, training of a parametric architecture for policy approximation is similar to training for a cost approximation. In the case where the control space is finite, however, it is useful to make the connection of approximation in policy space with *classification*; cf. Fig. 3.1.2 and the discussion of Section 3.1.

Classification is an important subject in machine learning. The objective is to construct an algorithm, called a *classifier*, which assigns a given "object" to one of a finite number of "categories" based on its "characteristics." Here we use the term "object" generically. In some cases, the classification may relate to persons or situations. In other cases, an object may represent a hypothesis, and the problem is to decide which of the hypotheses is true, based on some data. In the context of approximation in policy space, *objects correspond to states, and categories correspond to*

controls to be applied at the different states. Thus in this case, we view each sample $\big(x_k^s, u_k^s\big)$ as an object-category pair.

Generally, in (multiclass) classification we assume that we have a population of objects, each belonging to one of m categories $c = 1, \ldots, m$. We want to be able to assign a category to any object that is presented to us. Mathematically, we represent an object with a vector x (e.g., some raw description or a vector of features of the object), and we aim to construct a rule that assigns to every possible object x a unique category c.

To illustrate a popular classification method, let us assume that if we draw an object x at random from this population, the conditional probability of the object being of category c is $p(c \,|\, x)$. If we know the probabilities $p(c \,|\, x)$, we can use a classical statistical approach, whereby we assign x to the category $c^*(x)$ that has maximal posterior probability, i.e.,

$$c^*(x) \in \arg \max_{c=1,\ldots,m} p(c \,|\, x). \tag{3.34}$$

This is called the Maximum a Posteriori rule (or MAP rule for short; see for example the book [BeT08], Section 8.2, for a discussion).

When the probabilities $p(c \,|\, x)$ are unknown, we may try to estimate them using a least squares optimization, based on the following property, whose proof is outlined in Exercise 3.1.

Proposition 3.4.1: (Least Squares Property of Conditional Probabilities) Let $\xi(x)$ be any prior distribution of x, so that the joint distribution of (c, x) is

$$\zeta(c, x) = \xi(x) p(c \,|\, x).$$

For a pair of classes (c, c'), define $z(c, c')$ by

$$z(c, c') = \begin{cases} 1 & \text{if } c = c', \\ 0 & \text{otherwise,} \end{cases}$$

and for a fixed class c and any function h of (c, x), consider

$$E\Big\{ \big(z(c, c') - h(c, x)\big)^2 \Big\},$$

the expected value with respect to the distribution $\zeta(c', x)$ of the random variable $\big(z(c, c') - h(c, x)\big)^2$. Then $p(c \,|\, x)$ minimizes this expected value over all functions $h(c, x)$, i.e., for all functions h and all classes c, we have

$$E\Big\{\big(z(c,c')-p(c\,|\,x)\big)^2\Big\} \le E\Big\{\big(z(c,c')-h(c,x)\big)^2\Big\}. \qquad (3.35)$$

The proposition states that $p(c\,|\,x)$ is the function of (c,x) that minimizes

$$E\Big\{\big(z(c,c')-h(c,x)\big)^2\Big\} \qquad (3.36)$$

over all functions h of (c,x), *for any prior distribution of x and class c.* This suggests that we can obtain approximations to the probabilities $p(c\,|\,x)$, $c=1,\ldots,m$, by minimizing an empirical/simulation-based approximation of the expected value (3.36).

More specifically, let us assume that we have a training set consisting of q object-category pairs (x^s,c^s), $s=1,\ldots,q$, and corresponding vectors

$$z^s(c)=\begin{cases}1 & \text{if } c^s=c,\\ 0 & \text{otherwise,}\end{cases} \qquad c=1,\ldots,m,$$

and adopt a parametric approach. In particular, for each category $c=1,\ldots,m$, we approximate the probability $p(c\,|\,x)$ with a function $\tilde h(c,x,r)$ that is parametrized by a vector r, and optimize over r the empirical approximation to the expected squared error of Eq. (3.36). Thus we can obtain r by the least squares regression:

$$\bar r \in \arg\min_r \sum_{s=1}^{q}\sum_{c=1}^{m}\big(z^s(c)-\tilde h(c,x^s,r)\big)^2, \qquad (3.37)$$

perhaps with some quadratic regularization added. The functions $\tilde h(c,x,r)$ may be provided for example by a feature-based architecture or a neural network.

Note that each training pair (x^s,c^s) is used to generate m examples for use in the regression problem (3.37): $m-1$ "negative" examples of the form $(x^s,0)$, corresponding to the $m-1$ categories $c\neq c^s$, and one "positive" example of the form $(x^s,1)$, corresponding to $c=c^s$. Note also that the incremental gradient method can be applied to the solution of this problem.

The regression problem (3.37) approximates the minimization of the expected value (3.36), so we conclude that its solution $\tilde h(c,x,\bar r)$, $c=1,\ldots,m$, approximates the probabilities $p(c\,|\,x)$. Once this solution is obtained, we may use it to classify a new object x according to the rule

$$\text{Estimated Object Category}=\tilde c(x,\bar r)\in \arg\max_{c=1,\ldots,m}\tilde h(c,x,\bar r), \qquad (3.38)$$

which approximates the MAP rule (3.34); cf. Fig. 3.4.1.

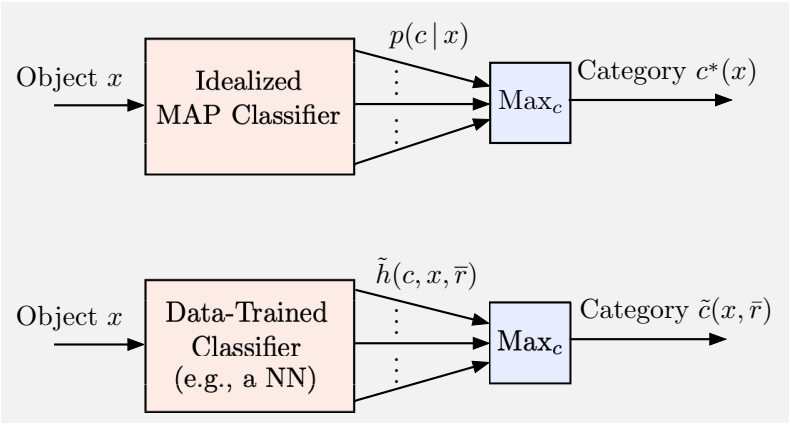

Figure 3.4.1 Illustration of the MAP classifier $c^*(x)$ for the case where the probabilities $p(c \mid x)$ are known [cf. Eq. (3.34)], and its data-trained version $\tilde{c}(x, \bar{r})$ [cf. Eq. (3.38)]. The classifier may be obtained by using the data set (x_k^s, u_k^s), $s = 1, \ldots, q$, and an approximation architecture such as a feature-based architecture or a neural network.

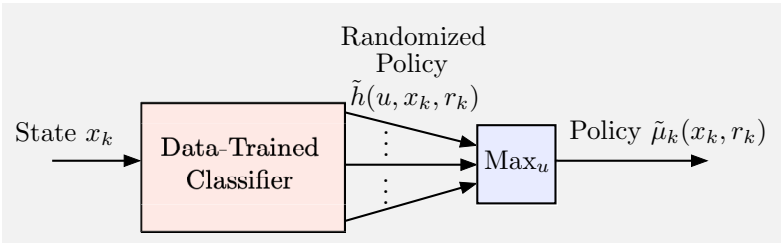

Figure 3.4.2 Illustration of classification-based approximation in policy space. The classifier, defined by the parameter r_k, is constructed by using the training set (x_k^s, u_k^s), $s = 1, \ldots, q$. It yields a randomized policy that consists of the probability $\tilde{h}(u, x_k, r_k)$ of using control $u \in U_k(x_k)$ at state x_k. This policy is approximated by the deterministic policy $\tilde{\mu}_k(x_k, r_k)$ that uses at state x_k the control that maximizes over $u \in U_k(x_k)$ the probability $\tilde{h}(u, x_k, r_k)$ [cf. Eq. (3.38)].

Returning to approximation in policy space, for a given training set (x_k^s, u_k^s), $s = 1, \ldots, q$, the classifier just described provides (approximations to) the "probabilities" of using the controls $u_k \in U_k(x_k)$ at the states x_k, so it yields a "randomized" policy $\tilde{h}(u, x_k, r_k)$ for stage k [once the values $\tilde{h}(u, x_k, r_k)$ are normalized so that, for any given x_k, they add to 1]; cf. Fig. 3.4.2. In practice, this policy is usually approximated by the deterministic policy $\tilde{\mu}_k(x_k, r_k)$ that uses at state x_k the control of maximal probability at that state; cf. Eq. (3.38).

For the simpler case of a classification problem with just two categories, say A and B, a similar formulation is to hypothesize a relation of

the following form between object x and its category:

$$\text{Object Category} = \begin{cases} A & \text{if } \tilde{h}(x,r) = 1, \\ B & \text{if } \tilde{h}(x,r) = -1, \end{cases}$$

where $\tilde{h}$ is a given function and r is the unknown parameter vector. Given a set of q object-category pairs $(x^1, z^1), \ldots, (x^q, z^q)$ where

$$z^s = \begin{cases} 1 & \text{if } x \text{ is of category } A, \\ -1 & \text{if } x \text{ is of category } B, \end{cases}$$

we obtain r by the least squares regression:

$$\bar{r} \in \arg\min_r \sum_{s=1}^{q} \big(z^s - \tilde{h}(x^s, r)\big)^2.$$

The optimal parameter vector $\bar{r}$ is used to classify a new object with data vector x according to the rule

$$\text{Estimated Object Category} = \begin{cases} A & \text{if } \tilde{h}(x,\bar{r}) > 0, \\ B & \text{if } \tilde{h}(x,\bar{r}) < 0. \end{cases}$$

In the context of DP and approximation in policy space, this classifier may be used, among others, in stopping problems where there are just two controls available at each state: stopping (i.e., moving to a termination state) and continuing (i.e., moving to some nontermination state).

There are several variations of the preceding classification schemes, for which we refer to the specialized literature. Moreover, there are several commercially and publicly available software packages for solving the associated regression problems and their variants. They can be brought to bear on the problem of parametric approximation in policy space using any training set of state-control pairs, regardless of how it was obtained.

3.4.2 Policy Iteration with Value and Policy Networks

We noted earlier that contrary to rollout, approximate policy iteration (PI) is fundamentally an off-line training algorithm, because for a large scale problem, it is necessary to represent the cost functions or Q-factors of the successively generated policies with an approximation architecture; cf. Section 3. Thus, in a typical implementation, approximate PI involves the successive use of value networks to represent the cost functions of the generated policies, and one-step or multistep lookahead minimization to implement policy improvement.

On the other hand, it is also possible to use policy networks to approximate the results of policy improvement. In particular, we can start with a

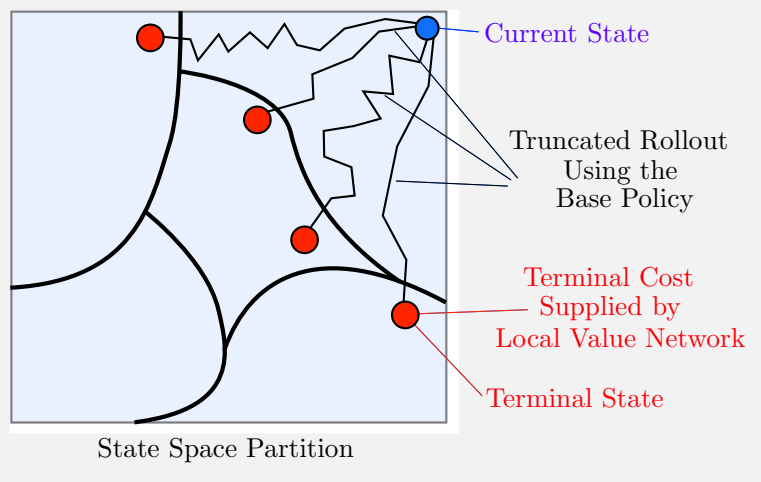

Figure 3.4.3 Illustration of a truncated rollout scheme with a partitioned architecture. A local value network is used for terminal cost function approximation for each subset of the partition.

base policy and a terminal cost approximation, and generate state-control samples of the corresponding truncated rollout policy. These samples can be used with an approximation in policy space scheme to train a policy network that approximates the truncated rollout policy.

Then the cost function of the policy network can be approximated with a value network using the cost approximation methodology that we have discussed in this chapter. This value network can be used in turn as a terminal cost approximation in a truncated rollout algorithm where the previously obtained policy network can be used as a base policy. A new policy network can then be trained using samples of this rollout policy, etc. Thus a perpetual rollout scheme is obtained, which involves a sequence of value and policy networks.

One may also consider approximate PI algorithms that do not use a value network at all. Indeed the value network is only used to provide the approximate cost function values of the current policy, which are needed to calculate samples of the improved policy and train the corresponding policy network. On the other hand the samples of the improved policy can also be computed by rollout, using simulation-generated cost function values of the current policy. If the rollout can be suitably implemented with simulation, the training of a value network may be unnecessary.

Multiprocessor Parallelization

We have noted earlier that parallelization and distributed computation can be used in several different ways in rollout and PI schemes, including Q-factor, Monte Carlo, and multiagent parallelization. It is also possible to

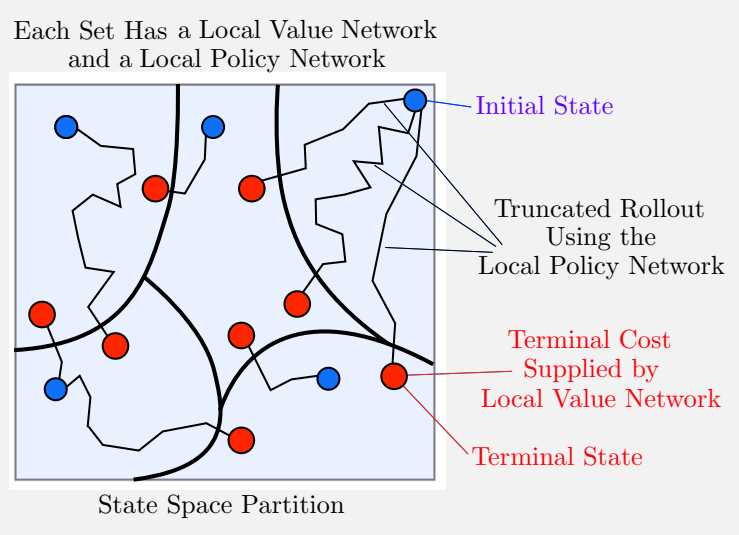

Figure 3.4.4 Illustration of a perpetual truncated rollout scheme with a partitioned architecture. A local value network and a local policy network are used for each subset of the partition. The policy network is used as the base policy and the value network is used to provide a terminal cost function approximation.

State-control training pairs for the corresponding rollout policy are obtained by starting at an initial state within some subset of the partition, generating rollout trajectories using the local policy network, which are truncated once the state enters a different subset of the partition, with the corresponding terminal cost function approximation supplied by the value network of that subset.

When a separate processor is used for each subset of partition, the corresponding value networks are communicated between processors. This can be done asynchronously, with each processor sharing its value network as it becomes available. In a variation of this scheme, the local policy networks may also be shared selectively among processors for selective use in the truncated rollout process.

consider the use of multiple neural networks in the implementation of rollout or approximate PI. For example, when feature-based partitioning of the state space is used (cf. Example 3.1.8), we may consider a multiprocessor parallelization scheme, which involves multiple local value and/or policy networks, which operate locally within a subset of the state space partition; see Figs. 3.4.3 and 3.4.4.

Let us finally note that multiprocessor parallelization leads to the idea of an approximation architecture that involves a graph. Each node of the graph consists of a neural network and each arc connecting a pair of nodes corresponds to data transfer between the corresponding neural networks. The question of how to train such an architecture is quite complex and one may think of several alternative possibilities. For example the training may be collaborative with the exchange of training results and/or training data communicated periodically or asynchronously; see the book [Ber20a], Section 5.8.

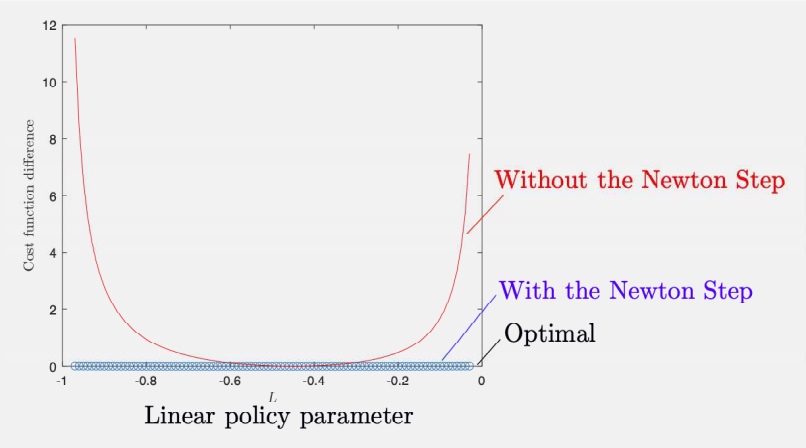

Figure 3.4.5 Illustration of the performance enhancement obtained when the Newton step/rollout is used in conjunction with an off-line trained (suboptimal) base policy for the linear quadratic problem.

3.4.3 Why Use On-Line Play and not Just Train a Policy Network to Emulate the Lookahead Minimization?

This is a sensible and common question, which stems from the mindset that neural networks have extraordinary function approximation properties. In other words, why go through the arduous and time-consuming process of on-line lookahead minimization, if we can do the same thing off-line and represent the lookahead policy with a trained policy network? In particular, we can select the policy from a suitably restricted class of policies, such as a parametric class of the form $\mu(x, r)$, where r is a parameter vector. We may then estimate r using some type of off-line training. Then the on-line computation of controls $\mu(x, r)$ can be much faster compared with on-line lookahead minimization.

On the negative side, because parametrized approximations often involve substantial calculations, they are not well suited for on-line replanning. From our point of view in this book, there is another important reason why approximation in value space is needed on top of approximation in policy space: *the off-line trained policy may not perform nearly as well as the corresponding one-step or multistep lookahead/rollout policy*, because it lacks the extra power of the associated exact Newton step (cf. our discussion of AlphaZero and TD-Gammon in Section 1.1, and linear quadratic problems in Section 1.5).

Figure 3.4.5 illustrates this fact with a one-dimensional linear-quadratic example, and compares the performance of a linear policy with its corresponding one-step lookahead policy. In this example the system equation

is

$$x_{k+1} = x_k + 2u_k,$$

and the quadratic cost function parameters are $q = 1$, $r = 0.5$. The optimal policy for this system and cost parameter values is

$$\mu^*(x) = L^*x,$$

with $L^* \approx -0.4$, and the optimal cost function is

$$J^*(x) = K^*x^2,$$

where $K^* \approx 1.1$. We want to to explore what happens when we use a policy of the form

$$\mu_L(x) = Lx,$$

where $L \neq L^*$ (e.g., a policy that is optimal for another system equation or cost function parameters). The cost function of μ_L has the form

$$J_\mu(x) = K_L x^2,$$

where K_L is obtained by using the formulas given in Section 1.5. The figure shows the quadratic cost coefficient differences $K_L - K^*$ and $K_{\tilde{L}} - K^*$ as a function of L, where K_L and $K_{\tilde{L}}$ are the quadratic cost coefficients of μ (without one-step lookahead/Newton step) and the corresponding one-step lookahead policy $\tilde{\mu}$ (with one-step lookahead/Newton step).

3.5 POLICY GRADIENT AND RELATED METHODS

In this section we focus on infinite horizon problems and we discuss an alternative training approach for approximation in policy space, which is based on controller parameter optimization: we parametrize the policies by a vector r, and we optimize the corresponding expected cost over r. In particular, we determine r through the minimization

$$\min_r E\big\{J_{\tilde{\mu}(r)}(i_0)\big\}, \tag{3.39}$$

where $J_{\tilde{\mu}(r)}(i_0)$ is the cost of the policy $\tilde{\mu}(r)$ starting from the initial state i_0, and the expected value above is taken with respect to a suitable probability distribution of the initial state i_0 (cf. Fig. 3.5.1).

Note that in the case where the initial state i_0 is known and fixed, the method involves just minimization of $J_{\tilde{\mu}(r)}(i_0)$ over r. This simplifies a great deal the minimization, particularly when the problem is deterministic.

Before delving into the details, it is worth reminding the reader that using an off-line trained policy without combining it with approximation in value space has the fundamental shortcoming, which we noted frequently in this book: *the off-line trained policy will not perform nearly as well as a scheme that uses this policy in conjunction with lookahead minimization*, because it lacks the extra power of the associated exact Newton step.

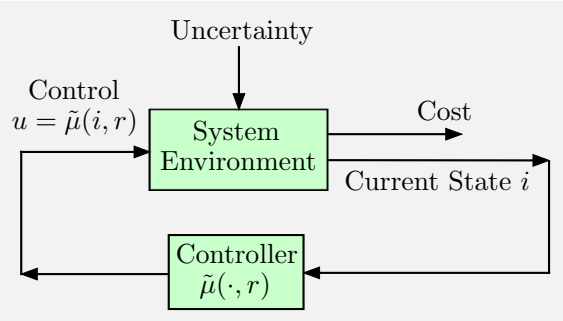

Figure 3.5.1 Illustration of the policy optimization framework for an infinite horizon problem. Policies are parametrized with a parameter vector r and denoted by $\tilde{\mu}(r)$, with components $\tilde{\mu}(i, r)$, $i = 1, \ldots, n$. Each parameter value r determines a policy $\tilde{\mu}(r)$, and a cost $J_{\tilde{\mu}(r)}(i_0)$ for each initial state i_0, as indicated in the figure. The optimization approach determines r through the minimization

$$\min_r E\Big\{ J_{\tilde{\mu}(r)}(i_0) \Big\},$$

where the expected value above is taken with respect to a suitable probability distribution of i_0.

3.5.1 Gradient Methods for Cost Optimization

Let us first consider methods that perform the minimization (3.39) by using a gradient method, and for simplicity let us assume that the initial condition i_0 is known. Thus the aim is to minimize $J_{\tilde{\mu}(r)}(i_0)$ over r by using the gradient method

$$r^{k+1} = r^k - \gamma^k \nabla J_{\tilde{\mu}(r^k)}(i_0), \qquad k = 0, 1, \ldots, \tag{3.40}$$

assuming that $J_{\tilde{\mu}(r)}(i_0)$ is differentiable with respect to r. Here γ^k is a positive stepsize parameter, and $\nabla(\cdot)$ denotes gradient with respect to r evaluated at the current iterate r^k.

The difficulty with this method is that the gradients $\nabla J_{\tilde{\mu}(r^k)}(i_0)$ may not be explicitly available. In this case, the gradients can be approximated by finite differences of cost function values $J_{\tilde{\mu}(r^k)}(i_0)$. Unfortunately, when the problem is stochastic, the cost function values may be computable only through Monte Carlo simulation. This may introduce a large amount of noise, so it is likely that many samples will need to be averaged in order to obtain sufficiently accurate gradients, thereby making the method inefficient. On the other hand, when the problem is deterministic, this difficulty does not appear, and the use of the gradient method (3.40) or other methods that do not rely on the use of gradients (such as coordinate descent) is facilitated.

In this section we will focus on alternative and typically more efficient gradient-like methods for stochastic problems, which are based on

sampling. Some popular methods of this type are based on incremental gradient ideas (cf. Section 3.1.3) and the use of randomized policies [i.e., policies that map a state i to a probability distribution over the set of controls $U(i)$, rather than mapping onto a single control].† We discuss these gradient-like methods next.

Policy Gradient-Like Methods

To get a sense of the general principle underlying the incremental gradient approach that uses randomization and sampling, let us digress from the DP context of this chapter, and consider the generic optimization problem

$$\min_{z \in Z} F(z), \tag{3.41}$$

where Z is a subset of the m-dimensional space $\Re^m$, and F is some real-valued function over $\Re^m$.

We will take the unusual step of converting this problem to the *stochastic* optimization problem

$$\min_{p \in \mathcal{P}_Z} E_p\big\{F(z)\big\}, \tag{3.42}$$

where z is viewed as a random variable, $\mathcal{P}_Z$ is the set of probability distributions over Z, p denotes the generic distribution in $\mathcal{P}_Z$, and $E_p\{\cdot\}$ denotes expected value with respect to p. Of course this enlarges the search space from Z to $\mathcal{P}_Z$, but it enhances the use of randomization schemes and simulation-based methods, even if the original problem is deterministic. Moreover, the stochastic optimization problem (3.42) may have some nice differentiability properties that are lacking in the original deterministic version (3.41); see the paper [Ber73] for an analysis of this differentiability issue under convexity assumptions on F.

At this point it is not clear how the stochastic optimization problem (3.42) relates to our stochastic DP context of this chapter. We will return to this question later, but for the purpose of orientation, we note that to obtain a problem of the form (3.42), we must enlarge the set of policies to include *randomized policies*, mapping a state i into a probability distribution over the set of controls $U(i)$.

† The AlphaGo and AlphaZero programs (Silver et al. [SHM16], [SHS17]) also use randomized policies, and a policy adjustment scheme that involves incremental changes along "directions of improvement." However, these changes are implemented through the MCTS algorithm used by these programs, without the explicit use of a gradient (see the discussion in Section 2.4.2). Thus it may be said that the AlphaGo and AlphaZero programs involve a form of approximation in policy space (as well as approximation in value space), which bears resemblance but cannot be classified as a policy gradient method.

Suppose now that we restrict attention to a subset $\tilde{\mathcal{P}}_Z \subset \mathcal{P}_Z$ of probability distributions $p(z;r)$ that are parametrized by some continuous parameter r, e.g., a vector in some Euclidean space.† In other words, we approximate the stochastic optimization problem (3.42) with the restricted problem

$$\min_r E_{p(z;r)}\big\{F(z)\big\}.$$

Then we may use a gradient method for solving this problem, such as

$$r^{k+1} = r^k - \gamma^k \nabla\Big(E_{p(z;r^k)}\big\{F(z)\big\}\Big), \qquad k = 0, 1, \ldots, \tag{3.43}$$

where $\nabla(\cdot)$ denotes gradient with respect to r of the function in parentheses, evaluated at the current iterate r^k.

Likelihood-Ratio Policy Gradient Methods

We will first consider an incremental version of the gradient method (3.43). This method requires that $p(z;r)$ is differentiable with respect to r. It relies on a convenient gradient formula, sometimes referred to as the *log-likelihood trick*, which involves the natural logarithm of the sampling distribution.

This formula is obtained by the following calculation, which is based on interchanging gradient and expected value, and using the gradient formula $\nabla(\log p) = \nabla p/p$. We have

$$\begin{aligned}
\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big) &= \nabla\left(\sum_{z\in Z} p(z;r)F(z)\right) \\
&= \sum_{z\in Z} \nabla p(z;r)F(z) \\
&= \sum_{z\in Z} p(z;r)\frac{\nabla p(z;r)}{p(z;r)}F(z) \\
&= \sum_{z\in Z} p(z;r)\nabla\Big(\log\big(p(z;r)\big)\Big)F(z),
\end{aligned}$$

and finally

$$\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big) = E_{p(z;r)}\Big\{\nabla\Big(\log\big(p(z;r)\big)\Big)F(z)\Big\}, \tag{3.44}$$

where for any given z, $\nabla\Big(\log\big(p(z;r)\big)\Big)$ is the gradient with respect to r of the function $\log\big(p(z;\cdot)\big)$, evaluated at r (the gradient is assumed to exist).

† To be on safe mathematical ground, we assume that $p(z;r)$ is a discrete distribution in what follows in this section.

The preceding formula suggests an incremental implementation of the gradient iteration (3.43) that approximates the expected value in the right side in Eq. (3.44) with a single sample (cf. Section 3.1.3). The typical iteration of this method is as follows.

Sample-Based Gradient Method for Parametric Approximation of $\min_{z\in Z} F(z)$

Let r^k be the current parameter vector.

(a) Obtain a sample z^k according to the distribution $p(z; r^k)$.

(b) Compute the gradient $\nabla\Big(\log\big(p(z^k; r^k)\big)\Big)$.

(c) Iterate according to

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k; r^k)\big)\Big) F(z^k). \tag{3.45}$$

The advantage of the preceding sample-based method is its simplicity and generality. It allows the use of parametric approximation for any minimization problem (well beyond DP), as long as the logarithm of the sampling distribution $p(z; r)$ can be conveniently differentiated with respect to r, and samples of z can be obtained using the distribution $p(z; r)$.

Note that in iteration (3.45) r is adjusted along a random direction. This direction does not involve at all the gradient of F, only the gradient of the logarithm of the sampling distribution! As a result the iteration has a *model-free character*: we don't need to know the form of the function F as long as we have a simulator that produces the cost function value $F(z)$ for any given z. This is also a major advantage offered by many random search methods.

An important issue is the efficient computation of the sampled gradient $\nabla\big(\log\big(p(z^k; r^k)\big)\big)$. In the context of DP, including the SSP and discounted problems that we have been dealing with, there are some specialized procedures and corresponding parametrizations to approximate this gradient conveniently. The following is an example.

Example 3.5.1 (Policy Gradient Method for Discounted DP)

Consider the α-discounted problem and denote by z the infinite horizon state-control trajectory:

$$z = \{i_0, u_0, i_1, u_1, \ldots\}.$$

We consider a parametrization of randomized policies with parameter r, so the control at state i is generated according to a distribution $p(u \mid i; r)$ over $U(i)$. Then for a given r, the state-control trajectory z is a random vector

with probability distribution denoted $p(z;r)$. The cost corresponding to the trajectory z is

$$F(z) = \sum_{m=0}^{\infty} \alpha^m g(i_m, u_m, i_{m+1}), \tag{3.46}$$

and the problem is to minimize over r

$$E_{p(z;r)}\big\{F(z)\big\}.$$

To apply the sample-based gradient method (3.45), given the current iterate r^k, we must generate the sample state-control trajectory z^k, according to the distribution $p(z;r^k)$, compute the corresponding cost $F(z^k)$, and also calculate the gradient

$$\nabla\Big(\log\big(p(z^k;r^k)\big)\Big). \tag{3.47}$$

Let us assume that the logarithm of the randomized policy distribution $p(u \mid i;r)$ is differentiable with respect to r (a soft-min policy parametrization is often recommended for this purpose). Then the logarithm that is differentiated in Eq. (3.47) can be written as

$$\begin{aligned}\log\big(p(z^k;r^k)\big) &= \log \prod_{m=0}^{\infty} p_{i_m i_{m+1}}(u_m)p(u_m \mid i_m;r^k) \\ &= \sum_{m=0}^{\infty} \log\big(p_{i_m i_{m+1}}(u_m)\big) + \sum_{m=0}^{\infty} \log\big(p(u_m \mid i_m;r^k)\big),\end{aligned}$$

and its gradient (3.47), which is needed in the iteration (3.45), is given by

$$\nabla\Big(\log\big(p(z^k;r^k)\big)\Big) = \sum_{m=0}^{\infty} \nabla\Big(\log\big(p(u_m \mid i_m;r^k)\big)\Big). \tag{3.48}$$

This gradient involves the current randomized policy, but does not involve the transition probabilities and the costs per stage.

The policy gradient method (3.45) can now be implemented with a finite horizon approximation whereby r^k is changed after a finite number N of time steps [so the infinite cost and gradient sums (3.46) and (3.48) are replaced by finite sums]. The method takes the form

$$r^{k+1} = r^k - \gamma^k \sum_{m=0}^{N-1} \nabla \log\big(p(u_m \mid i_m;r^k)\big)F_N(z_N^k),$$

where $z_N^k = (i_0, u_0, \ldots, i_{N-1}, u_{N-1})$ is the generated N-step trajectory, and $F_N(z_N^k)$ is the corresponding cost. The initial state i_0 of the trajectory is chosen randomly, with due regard to exploration issues.

Policy gradient methods for other types of DP problems can be similarly developed, as well as variations involving a combination of policy

and cost function approximations [e.g., replacing $F(z)$ of Eq. (3.46) by a parametrized estimate that has smaller variance, and possibly subtracting a suitable baseline, cf. the following discussion]. This leads to a class of actor-critic methods that differ from the PI-type methods that we discussed earlier in this chapter; we refer to the end-of-chapter literature for a variety of specific schemes.

Implementation Issues

There are several issues to consider in the implementation of the sample-based gradient method (3.45). The first of these is that the problem solved is a randomized version of the original. If the method produces a parameter $\overline{r}$ in the limit and the distribution $p(z;\overline{r})$ is not atomic (i.e., it is not concentrated at a single point), then a solution $\overline{z} \in Z$ must be extracted from $p(z;\overline{r})$. In the SSP and discounted problems of this chapter, the subset $\tilde{\mathcal{P}}_Z$ of parametric distributions typically contains the atomic distributions, while it can be shown that minimization over the set of all distributions $\mathcal{P}_Z$ produces the same optimal value as minimization over Z (the use of randomized policies does not improve the optimal cost of the problem), so this difficulty does not arise.

Another issue is how to collect the samples z^k. Different methods must strike a balance between convenient implementation and a reasonable guarantee that the search space Z is sufficiently well explored.

Finally, there is the issue of improving sampling efficiency. To this end, let us note a simple generalization of the gradient method (3.45), which can often improve its performance. It is based on the gradient formula

$$\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big) = E_{p(z;r)}\Big\{\nabla\Big(\log\big(p(z;r)\big)\Big)\big(F(z)-b\big)\Big\}, \qquad (3.49)$$

where b is any scalar. This formula generalizes Eq. (3.44), where $b = 0$, and holds in view of the following calculation, which shows that the term multiplying b in Eq. (3.49) is equal to 0:

$$\begin{aligned} E_{p(z;r)}\Big\{\nabla\Big(\log\big(p(z;r)\big)\Big)\Big\} &= E_{p(z;r)}\left\{\frac{\nabla p(z;r)}{p(z;r)}\right\} \\ &= \sum_{z\in Z} p(z;r)\frac{\nabla p(z;r)}{p(z;r)} \\ &= \sum_{z\in Z}\nabla p(z;r) = \nabla\left(\sum_{z\in Z} p(z;r)\right) = 0, \end{aligned}$$

where the last equality holds because $\sum_{z\in Z} p(z;r)$ is identically equal to 1 and hence does not depend on r.

Based on the gradient formula (3.49), we can modify the iteration (3.45) to read as follows:

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k; r^k)\big)\Big)\big(F(z^k) - b\big), \tag{3.50}$$

where b is some fixed scalar, called the *baseline*. Whereas the choice of b does not affect the gradient $\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big)$ [cf. Eq. (3.49)], it affects the incremental gradient

$$\nabla\Big(\log\big(p(z^k; r^k)\big)\Big)\big(F(z^k) - b\big),$$

which is used in the iteration (3.50). Thus, by optimizing the baseline b, empirically or through a calculation (see e.g., [DNP11]), we can improve the performance of the algorithm. Moreover, in the context of discounted and SSP problems, state-dependent baseline functions have been used. Ideas of cost shaping are useful within this context; we refer to the RL textbook [Ber19a] and specialized literature for further discussion.

Random Direction Methods

We will now consider an alternative class of incremental versions of the policy gradient method (3.43), repeated here for convenience:

$$r^{k+1} = r^k - \gamma^k \nabla\Big(E_{p(z;r^k)}\big\{F(z)\big\}\Big), \qquad k = 0, 1, \ldots. \tag{3.51}$$

These methods are based on the use of a random search direction and only *two sample function values per iteration*. They are generally faster than methods that use a finite difference approximation of the entire cost function gradient; see the book by Spall [Spa03] for a detailed discussion, and the paper by Nesterov and Spokoiny [NeS17] for a more theoretical view. Moreover, these methods do not require the derivative of the sampling distribution or its logarithm.

For simplicity we first consider the case where z and r are scalars, and later discuss the multidimensional case. In particular, we assume that $p(z; r)$ is symmetric and is concentrated with probabilities p_i at the points $r + \epsilon_i$ and $r - \epsilon_i$, where $\epsilon_1, \ldots, \epsilon_m$ are some small positive scalars. Thus we have

$$E_{p(z;r)}\big\{F(z)\big\} = \sum_{i=1}^{m} p_i\big(F(r + \epsilon_i) + F(r - \epsilon_i)\big),$$

and

$$\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big) = \sum_{i=1}^{m} p_i\big(\nabla F(r + \epsilon_i) + \nabla F(r - \epsilon_i)\big).$$

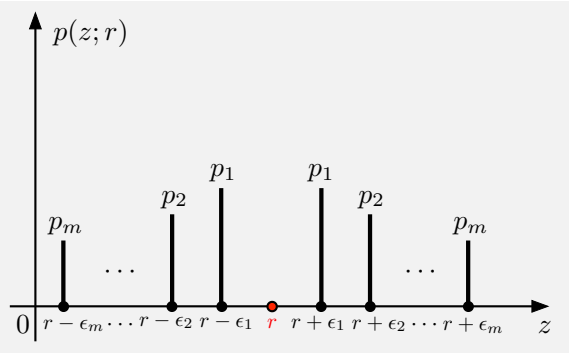

Figure 3.5.2 The distribution $p(z;r)$ used in the gradient iteration (3.52).

The gradient iteration (3.51) becomes

$$r^{k+1} = r^k - \gamma^k \sum_{i=1}^{m} p_i \big(\nabla F(r^k + \epsilon_i) + \nabla F(r^k - \epsilon_i)\big), \qquad k = 0, 1, \ldots, \quad (3.52)$$

Let us now consider approximation of the gradient by finite differences:

$$\nabla F(r + \epsilon_i) \approx \frac{F(r + \epsilon_i) - F(r)}{\epsilon_i}, \quad \nabla F(r - \epsilon_i) \approx \frac{F(r) - F(r - \epsilon_i)}{\epsilon_i}.$$

We approximate the gradient iteration (3.52) by

$$r^{k+1} = r^k - \gamma^k \sum_{i=1}^{m} p_i \frac{F(r^k + \epsilon_i) - F(r^k - \epsilon_i)}{\epsilon_i}, \qquad k = 0, 1, \ldots. \quad (3.53)$$

One possible sample-based/incremental version of this iteration is

$$r^{k+1} = r^k - \gamma^k \frac{F(r^k + \epsilon_{i^k}) - F(r^k - \epsilon_{i^k})}{\epsilon_{i^k}}, \quad (3.54)$$

where i^k is an index generated with probabilities that are proportional to p_{i^k}. This algorithm uses one out of the m terms of the gradient in Eq. (3.53).

The extension to the case, where z and r are multidimensional, is straightforward. Here $p(z;r)$ is a probability distribution, whereby z takes values of the form $r + \epsilon d$, where d is a random vector that lies on the surface of the unit sphere, and ϵ (independently of d) takes scalar values according to a distribution that is symmetric around 0. The idea is that at r^k, we first choose randomly a direction d^k on the surface of the unit sphere, and then change r^k along d^k or along $-d^k$, depending on the sign of the

corresponding directional derivative. For a finite difference approximation of this iteration, we sample z^k along the line $\{r^k + \epsilon d^k \mid \epsilon \in \Re\}$, and similar to the iteration (3.54), we set

$$r^{k+1} = r^k - \gamma^k \frac{F(r^k + \epsilon^k d^k) - F(r^k - \epsilon^k d^k)}{\epsilon^k} d^k, \tag{3.55}$$

where ϵ^k is the sampled value of ϵ.

Let us also discuss the case where $p(z; r)$ is a discrete but *nonsymmetric* distribution, i.e., z takes values of the form $r + \epsilon d$, where d is a random vector that lies on the surface of the unit sphere, and ϵ is a zero mean scalar. Then the analog of iteration (3.55) is

$$r^{k+1} = r^k - \gamma^k \frac{F(r^k + \epsilon^k d^k) - F(r^k)}{\epsilon^k} d^k, \tag{3.56}$$

where ϵ^k is the sampled value of ϵ. Thus in this case, we still require two function values per iteration. Generally, for a symmetric sampling distribution, iteration (3.55) tends to be more accurate than iteration (3.56), and is often preferred.

Algorithms of the form (3.55) and (3.56) are known as *random direction methods*. They use only two cost function values per iteration, and a direction d^k that need not be related to the gradient of F in any way. There is some freedom in selecting d^k, which could potentially be exploited in specific schemes. However, selecting the stepsize γ^k and the sampling distribution for ϵ can be tricky, particularly when the values of F are noisy.

3.5.2 Random Search and Cross-Entropy Methods

The main drawback of the policy gradient methods that we have considered in this section is the risk of unreliability due to the stochastic uncertainty corrupting the calculation of the gradients, the slow convergence that is typical of gradient methods in many settings, and the presence of local minima. For this reason, methods based on random search have been considered as potentially more reliable alternatives. Viewed from a high level, random search methods are similar to policy gradient methods in that they aim at iterative cost improvement through sampling. However, they need not involve randomized policies, they are not subject to cost differentiability restrictions, and they offer some global convergence guarantees, so in principle they are not affected much by local minima.

Let us consider a parametric policy optimization approach based on solving the problem

$$\min_r E\big\{J_{\tilde{\mu}(r)}(i_0)\big\},$$

cf. Eq. (3.39). Random search methods for this problem explore the space of the parameter vector r in some randomized but intelligent fashion. There

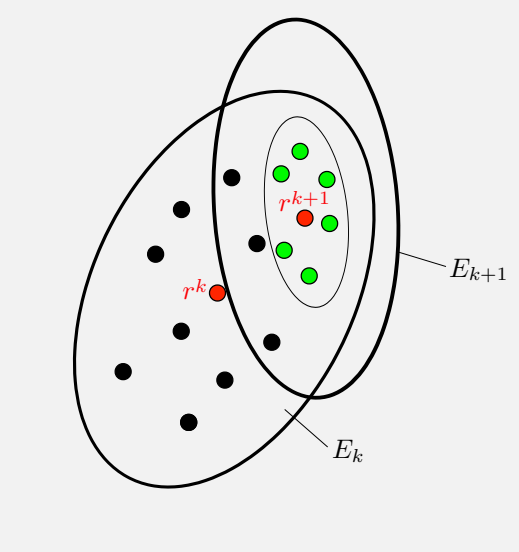

Figure 3.5.3 Schematic illustration of the cross-entropy method. At the current iterate r^k, we construct an ellipsoid E_k centered at r^k. We generate a number of random samples within E_k, and we "accept" a subset of the samples that have "low" cost. We then choose r^{k+1} to be the sample mean of the accepted samples, and construct a sample "covariance" matrix of the accepted samples. We then form the new ellipsoid E_{k+1} using this matrix and a suitably enlarged radius, and continue. Notice the resemblance with a policy gradient method: we move from r^k to r^{k+1} in a direction of cost improvement.

are several types of such methods for general optimization, and some of them have been suggested for approximate DP. We will briefly describe the *cross-entropy method*, which has gained considerable attention.

The method, when adapted to the approximate DP context, bears resemblance to policy gradient methods, in that it generates a parameter sequence $\{r^k\}$ by changing r^k to r^{k+1} along a direction of "improvement." This direction is obtained by using the policy $\tilde{\mu}(r^k)$ to generate randomly cost samples corresponding to a set of sample parameter values that are concentrated around r^k. The current set of sample parameters are then screened: some are accepted and the rest are rejected, based on a cost improvement criterion. Then r^{k+1} is determined as a "central point" or as the "sample mean" in the set of accepted sample parameters, some more samples are generated randomly around r^{k+1}, and the process is repeated; see Fig. 3.5.3. Thus successive iterates r^k are "central points" of successively better groups of samples, so in some broad sense, the random sample generation process is guided by cost improvement. This idea is shared with other popular classes of random search methods.

The cross-entropy method is very simple to implement, does not suffer from the fragility of gradient-based optimization, does not involve ran-

domized policies, and relies on some supportive theory. Importantly, the method does not require the calculation of gradients, and it does not require differentiability of the cost function. Moreover, it does not need a model to compute the required costs of different policies; a simulator is sufficient.

Like all random search methods, the convergence rate guarantees of the cross-entropy method are limited, and its success depends on domain-specific insights and the skilled use of heuristics. However, the method relies on solid ideas and has gained a favorable reputation. In particular, it was used with impressive success in the context of the game of tetris; see Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09]. There have also been reports of domain-specific successes with related random search methods; see Salimans et al. [SHC17]. We refer to the end-of-chapter literature for details and examples of implementation.

3.6 AGGREGATION

In this section we consider approximation in value space using a problem approximation approach that is based on aggregation. In particular, we construct a simpler and more tractable "aggregate" problem by creating special subsets of states, which we view as "aggregate states." We then solve the aggregate problem exactly by DP. This is the off-line training part of the aggregation approach, and it may be carried out with a variety of DP methods, including simulation-based value and policy iteration; we refer to the RL book [Ber19a] for a detailed account. Finally, we use the optimal cost-to-go function of the aggregate problem to construct a terminal cost approximation in a one-step or multistep lookahead approximation scheme for the original problem. Additionally, we may also use the optimal policy of the aggregate problem to construct a base policy for a truncated rollout scheme.

In addition to problem approximation, aggregation is related to feature-based parametric approximation. In particular, it often produces a piecewise constant cost function approximation, which may be viewed as a linear feature-based parametrization, where the features are 0-1 membership functions; see Example 3.1.1. Aggregation can also be combined with other approximation schemes, to add a local correction to a cost function approximation $\tilde{J}$, which is already available, possibly through the use of a neural network; see the discussion of biased aggregation later in Section 3.6.7.

Aggregation can be applied to both finite horizon and infinite horizon problems. In this section, we will focus primarily on the discounted infinite horizon problem. We will introduce aggregation in a simple intuitive form in Section 3.6.1, and generalize later to a more sophisticated form of feature-based aggregation, which we also discussed briefly in Example 3.1.7.

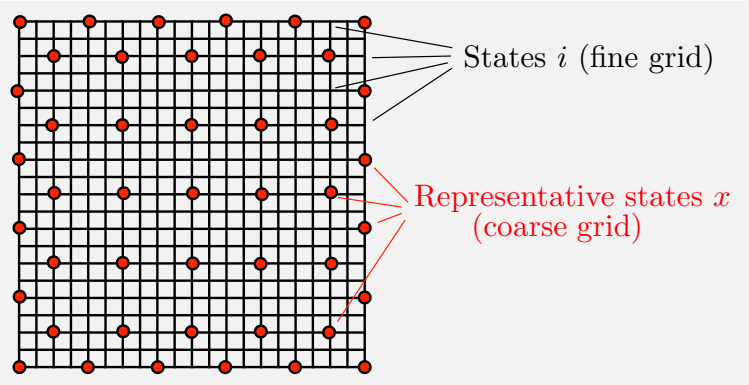

Figure 3.6.1 Illustration of aggregation with representative states; cf. Example 3.6.1. A relatively small number of states are viewed as representative. We define transition probabilities between pairs of aggregate states and we also define the associated expected transition costs. These specify a smaller DP problem, called the aggregate problem, which is solved exactly. The optimal cost function J^* of the original problem is approximated by interpolation from the optimal costs of the representative states r_y^* in the aggregate problem:

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n,$$

and is used in a one-step or multistep lookahead scheme.

3.6.1 Aggregation with Representative States

In this section we focus on a relatively simple form of aggregation, which involves a special subset of states, called *representative*. Our approach is to view these states as the states of a smaller optimal control problem, the aggregate problem, which we will formulate and solve exactly in place of the original. We will then use the optimal aggregate costs of the representative states to approximate the optimal costs of the original problem states by interpolation. In this chapter, whenever we consider a finite-state problem, we use notation that is more convenient for such a problem. In particular, states and successor states will be denoted by i and j, respectively, and the system equation is represented by control-dependent transition probabilities $p_{ij}(u)$; cf. Section 1.4.1. Let us describe a classical example.

Example 3.6.1 (Coarse Grid Approximation)

Consider a discounted problem where the state space is a grid of points $i = 1, \ldots, n$ on the plane. We introduce a coarser grid that consists of a subset $\mathcal{A}$ of the states/points, which we call representative and denote by x; see Fig. 3.6.1. We now wish to formulate a lower-dimensional DP problem just on the coarse grid of states. The difficulty here is that there may be positive

transition probabilities $p_{xj}(u)$ from some representative states x to some non-representative states j. To deal with this difficulty, we introduce artificial transition probabilities ϕ_{jy} from non-representative states j to representative states y, which we call *aggregation probabilities*. In particular, a transition from representative state x to a nonrepresentative state j, is followed by a transition from j to some other representative state y with probability ϕ_{jy}; see Fig. 3.6.2.

This process involves approximation but constructs a transition mechanism for an *aggregate problem* whose states are just the representative ones. The transition probabilities between representative states x, y under control $u \in U(x)$ and the corresponding expected transition costs are

$$\hat{p}_{xy}(u) = \sum_{j=1}^{n} p_{xj}(u)\phi_{jy}, \qquad \hat{g}(x,u) = \sum_{j=1}^{n} p_{xj}(u)g(x,u,j). \tag{3.57}$$

We can solve the aggregate problem by any suitable exact DP method. Let $\mathcal{A}$ denote the set of representative states and let r_x^* denote the corresponding optimal cost of representative state x. We can then approximate the optimal cost function of the original problem with the interpolation formula

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n. \tag{3.58}$$

This function may in turn be used in a one-step or multistep lookahead scheme for approximation in value space of the original problem.

Note that there is a lot of freedom in selecting the aggregation probabilities ϕ_{jy}. Intuitively, ϕ_{jy} should express a measure of proximity between j and y, e.g., ϕ_{jy} should be relatively large when y is geometrically close to j. For example, we could set $\phi_{jy_j} = 1$ for the representative state y_j that is "closest" to j, and $\phi_{jy_j} = 0$ for all other representative states $y \neq y_j$. In this case, Eq. (3.58) yields a piecewise constant cost function approximation $\tilde{J}$ (the constant values are the scalars r_y^* of the representative states y).

We will now formalize our framework for aggregation with representative states by generalizing the preceding example; see Fig. 3.6.3. We first consider the n-state version of the α-discounted problem of Section 1.4.1. We refer to this problem as the "original problem," to distinguish from the "aggregate problem," which we define next.

Aggregation Framework with Representative States

We introduce a finite subset $\mathcal{A}$ of the original system states, which we call *representative states*, and we denote them by symbols such as x and y. We construct an *aggregate problem*, with state space $\mathcal{A}$, and transition probabilities and transition costs defined as follows:

(a) We relate the original system states j to representative states $y \in \mathcal{A}$ with aggregation probabilities ϕ_{jy}; these are scalar "weights" satisfying $\phi_{jy} \geq 0$ for all $y \in \mathcal{A}$, and $\sum_{y \in \mathcal{A}} \phi_{jy} = 1$.

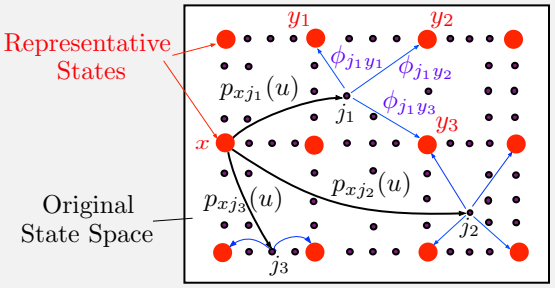

Figure 3.6.2 Illustration of the use of aggregation probabilities ϕ_{jy} from nonrepresentative states j to representative states y in Example 3.6.1. A transition from a state x to a nonrepresentative state j is followed by a transition to aggregate state y with probability ϕ_{jy}. In this figure, from representative state x, there are three possible transitions, to states j_1, j_2, and j_3, according to $p_{xj_1}(u), p_{xj_2}(u), p_{xj_3}(u)$, and each of these states is associated with a convex combination of representative states using the aggregation probabilities. For example, the state j_1 is associated with

$$\phi_{j_1 y_1} y_1 + \phi_{j_1 y_2} y_2 + \phi_{j_1 y_3} y_3.$$

(b) We define the transition probabilities between representative states x and y under control $u \in U(x)$ by

$$\hat{p}_{xy}(u) = \sum_{j=1}^{n} p_{xj}(u)\phi_{jy}. \tag{3.59}$$

(c) We define the expected transition costs at representative states x under control $u \in U(x)$ by

$$\hat{g}(x,u) = \sum_{j=1}^{n} p_{xj}(u)g(x,u,j). \tag{3.60}$$

The optimal costs of the representative states $y \in \mathcal{A}$ in the aggregate problem are denoted by r_y^*, and they define approximate costs for the original problem through the interpolation formula

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n. \tag{3.61}$$

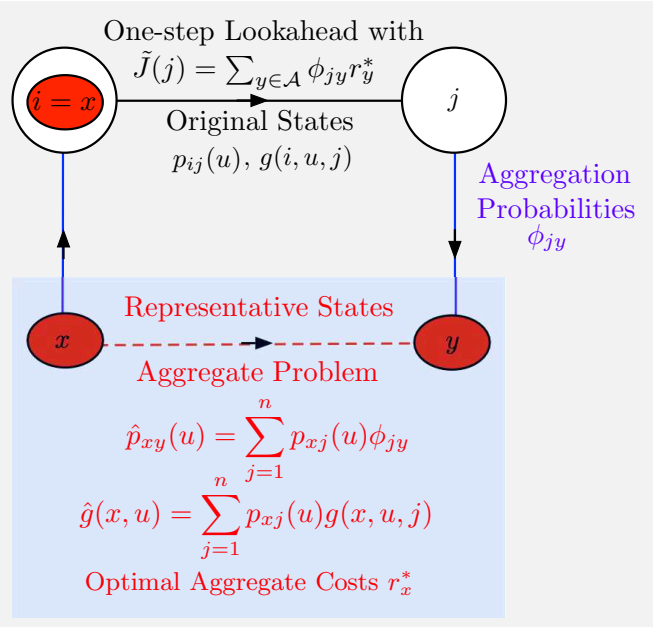

Figure 3.6.3 Illustration of the aggregate problem in the representative states framework. The transition probabilities $\hat{p}_{xy}(u)$ and transition costs $\hat{g}(x,u)$ are shown in the bottom part of the figure. Once the aggregate problem is solved (exactly) for its optimal costs r_y^*, we define approximate costs

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n,$$

which are used for one-step lookahead approximation of the original problem.

Aside from the selection of representative states, an important consideration is the choice of the aggregation probabilities. These probabilities express "similarity" or "proximity" of original to representative states (as in the case of the coarse grid Example 3.6.1), but in principle they can be arbitrary (as long as they are nonnegative and sum to 1 over y). Intuitively, ϕ_{jy} may be interpreted as some measure of "strength of relation" of j to y. The vectors $\{\phi_{jy} \mid j = 1, \ldots, n\}$ may also be viewed as basis functions for a linear cost function approximation via Eq. (3.61).

Hard Aggregation and Error Bound

A special case of interest, called *hard aggregation*, is when for every state j, we have $\phi_{jy} = 0$ for all representative states y, except a single one, denoted y_j, for which we have $\phi_{jy_j} = 1$. In this case, the one-step lookahead

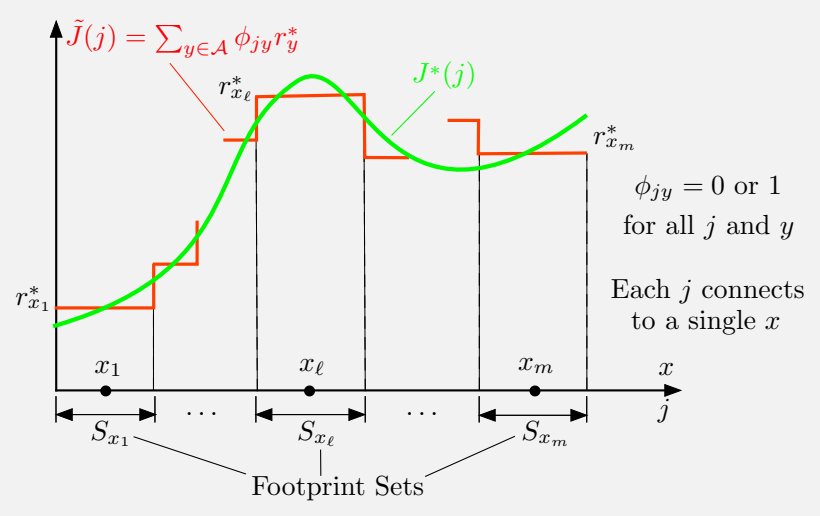

Figure 3.6.4 Illustration of the piecewise constant cost approximation

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n,$$

in the hard aggregation case where we have $\phi_{jy} = 0$ for all representative states y, except a single one. Here $\tilde{J}$ is constant and equal to r_y^* for all j in the footprint set

$$S_y = \{j \mid \phi_{jy} = 1\}, \qquad y \in \mathcal{A}.$$

approximation

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n,$$

is *piecewise constant*; it is constant and equal to r_y^* for all j in the set

$$S_y = \{j \mid \phi_{jy} = 1\}, \qquad y \in \mathcal{A},$$

called the *footprint* of representative state y; see Fig. 3.6.4. Moreover the footprints of all the representative states are disjoint and form a partition of the state space, i.e.,

$$\cup_{x \in \mathcal{A}} S_x = \{1, \ldots, n\}.$$

The footprint sets can be used to define a bound for the error $(J^* - \tilde{J})$. In particular, it can be shown that

$$\left|J^*(j) - \tilde{J}(j)\right| \leq \frac{\epsilon}{1 - \alpha}, \qquad j = 1, \ldots, n,$$

where

$$\epsilon = \max_{y \in \mathcal{A}} \max_{i,j \in S_y} \left| J^*(i) - J^*(j) \right|$$

is the *maximum variation of J^* within the footprint sets S_y*. This error bound result can be extended to the more general aggregation framework that will be given in the next section. Note the primary intuition derived from this bound: *the error due to hard aggregation is small if J^* varies little within each S_y*.

For a special hard aggregation case of interest, consider the geometrical context of Example 3.6.1. There, aggregation probabilities are often based on a nearest neighbor approximation scheme, whereby each non-representative state j takes the cost value of the "closest" representative state y, i.e.,

$$\phi_{jy_j} = 1 \qquad \text{if } y_j \text{ is the closest representative state to } j.$$

Then all states j for which a given representative state y is the closest to j (the footprint of y) are assigned equal approximate cost $\tilde{J}(j) = r^*_y$.

Methods for Solving the Aggregate Problem

The most straightforward way to solve the aggregate problem is to compute the aggregate problem transition probabilities $\hat{p}_{xy}(u)$ [cf. Eq. (3.59)] and transition costs $\hat{g}(x,u)$ [cf. Eq. (3.60)] by either an algebraic calculation or by simulation. The aggregate problem may then be solved by any one of the standard methods, such as VI or PI. This exact calculation is plausible if the number of representative states is relatively small. An alternative possibility is to use a simulation-based VI or PI method. We refer to a discussion of these methods in the author's books [Ber12], Section 6.5, and [Ber19a], Section 6.3. The idea is that a simulator for the original problem can be used to construct a simulator for the aggregate problem; cf. Fig. 3.6.3.

An important observation is that if the original problem is deterministic and hard aggregation is used, the aggregate problem is also deterministic, and can be solved by shortest-path like methods. This is true for both discounted problems and for undiscounted shortest path-type problems. In the latter case, the termination state of the original problem must be included as a representative state in the aggregate problem. However, if hard aggregation is not used, the aggregate problem will be stochastic, because of the introduction of the aggregation probabilities. Of course, once the aggregate problem is solved and the lookahead approximation $\tilde{J}$ is obtained, a deterministic structure in the original problem can be exploited to facilitate the lookahead minimizations.

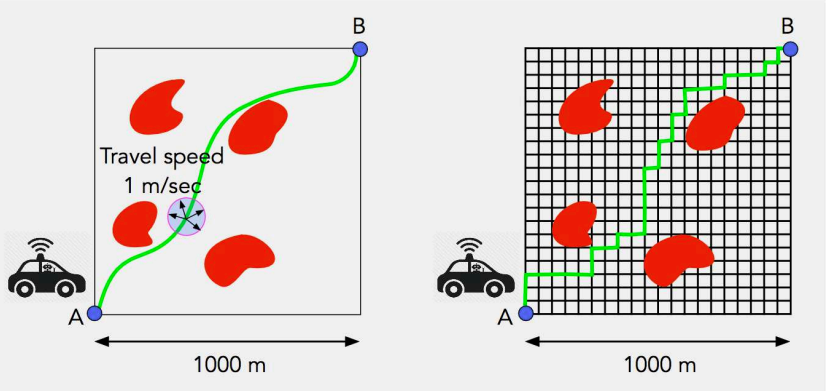

Figure 3.6.5 Illustration of discretization issues for problems with infinite state and control spaces.

3.6.2 Continuous Control Space Discretization

Aggregation with representative states extends without difficulty to problems with a continuous state space, as long as the control space is finite. Then once the representative states and the aggregation probabilities have been defined, the corresponding aggregate problem is a discounted problem with finite state and control spaces, which can be solved with the standard methods. The only potential difficulty arises when the disturbance space is also infinite, in which case the calculation of the transition probabilities and expected stage costs of the aggregate problem must be obtained by some form of integration process.

The case where both the state and the control spaces are continuous is somewhat more complicated, because both of these spaces must be discretized using representative state-control pairs, instead of just representative states. The following example illustrates what may happen if we use representative state discretization only.

Example 3.6.2 (Continuous Shortest Path Discretization)

Suppose that we want to find the fastest route for a car to travel between two points A and B located at the opposite ends of a square with side 1000 meters, while avoiding some known obstacles. We assume a constant car speed of 1 meter per second and that the car can drive in any direction; cf. Fig. 3.6.5.

Let us consider discretizing the space with a square grid (a set of representative states), and restrict the directions of motion to horizontal and vertical, so that at each stage the car moves from a grid point to one of the four closest grid points. Thus in the discretized version of the problem the car travels with a sequence of horizontal and vertical moves as indicated in the right side of Fig. 3.6.5. Is it possible to approximate the fastest route arbi-

trarily closely with the optimal solution of the discretized problem, assuming a sufficiently fine grid?

The answer is no! To see this note that in the discretized problem the optimal travel time is 2000 secs, regardless of how fine the discretization is. On the other hand, in the continuous space/nondiscretized problem the optimal travel time can be as little as $\sqrt{2} \cdot 1000$ secs (this corresponds to the favorable case where the straight line from A to B does not meet an obstacle).

The difficulty in the preceding example is that *the state space is discretized finely but the control space is not*. What is needed is to introduce a fine discretization of the control space as well, through some set of "representative controls." We can deal with this situation with a suitable form of discretized aggregate problem, which when solved provides an appropriate form of cost function approximation for use with one-step lookahead. The discretized problem is a stochastic infinite horizon problem, even if the original problem is deterministic. Further discussion of this approach is outside our scope, and we refer to the sources cited at the end of the chapter. Under reasonable assumptions it is possible to show consistency, i.e., that the optimal cost function of the discretized problem converges to the optimal cost function of the original continuous spaces problem as the discretization of both the state and the control spaces becomes increasingly fine.

The type of difficulty illustrated in Example 3.6.2 does not arise if the state space is continuous but the control space is finite. In particular, this is true in partially observed finite spaces Markov decision problems (POMDP), which are defined over their belief space (the space of probability distributions over their states). We briefly discuss this case next.

3.6.3 Continuous State Space - POMDP Discretization

Let us consider any α-discounted DP problem, where the state space is a bounded convex subset B of a Euclidean space, such as the unit simplex, but the control space U is finite. We use b to denote the states, to emphasize the connection with belief states in POMDP and to distinguish them from x, which we will use to denote representative states. Bellman's equation is $J = TJ$ with the Bellman operator T defined by

$$(TJ)(b) = \min_{u \in U} \mathop{E}_{w} \big\{ g(b,u,w) + \alpha J\big(f(b,u,w)\big) \big\}, \qquad b \in B.$$

We introduce a set of representative states $\{x_1, \ldots, x_m\} \subset B$. We assume that the convex hull of $\{x_1, \ldots, x_m\}$ is equal to B, so each state $b \in B$ can be expressed as

$$b = \sum_{i=1}^{m} \phi_{bx_i} x_i,$$

where $\{\phi_{bx_i} \mid i = 1, \ldots, m\}$ is a probability distribution:

$$\phi_{bx_i} \geq 0,\ i = 1, \ldots, m, \quad \sum_{i=1}^{m} \phi_{bx_i} = 1, \qquad \text{for all } b \in B.$$

We view ϕ_{bx_i} as aggregation probabilities.

Consider the operator $\hat{T}$ that transforms a vector $r = (r_{x_1}, \ldots, r_{x_m})$ into the vector $\hat{T}r$ with components $(\hat{T}r)(x_1), \ldots, (\hat{T}r)(x_m)$ defined by

$$(\hat{T}r)(x_i) = \min_{u \in U} \mathop{E}_{w} \left\{ g(x_i, u, w) + \alpha \sum_{j=1}^{m} \phi_{f(x_i,u,w)\, x_j} r_{x_j} \right\}, \qquad i = 1, \ldots, m,$$

where $\phi_{f(x_i,u,w)\, x_j}$ are the aggregation probabilities of the state $f(x_i, u, w)$. It can then be shown that $\hat{T}$ is a contraction mapping with respect to the maximum norm (we give the proof for a similar result in the next section). Bellman's equation for an aggregate finite-state discounted DP problem whose states are $x_1, \ldots, x_m$ has the form

$$r_{x_i} = (\hat{T}r)(x_i), \qquad i = 1, \ldots, m,$$

and has a unique solution.

The transitions in this problem occur as follows: from state x_i under control u, we first move to $f(x_i, u, w)$ at cost $g(x_i, u, w)$, and then we move to a state x_j, $j = 1, \ldots, m$, according to the probabilities $\phi_{f(x_i,u,w)\, x_j}$. The optimal costs $r^*_{x_i}$, $i = 1, \ldots, m$, of this problem can often be obtained by standard VI and PI methods that may or may not use simulation. We may then approximate the optimal cost function of the original problem by

$$\tilde{J}(b) = \sum_{i=1}^{m} \phi_{bx_i} r^*_{x_i}, \qquad \text{for all } b \in B,$$

and reasonably expect that the optimal discretized solution converges to the optimal as the number of representative states increases.

In the case where B is the belief space of an α-discounted POMDP, the representative states/beliefs and the aggregation probabilities define an aggregate problem, which is a finite-state α-discounted problem with a perfect state information structure. This problem can be solved with exact DP methods if either the aggregate transition probabilities and transition costs can be obtained analytically (in favorable cases) or if the number of representative states is small enough to allow their calculation by simulation. The aggregate problem can also be addressed with approximate DP method that we have discussed earlier, such as problem approximation/certainty equivalence approaches. It can also be addressed with a rollout method, which is suitable for an on-line implementation.

3.6.4 General Aggregation

We will now discuss a more general aggregation framework for the infinite horizon n-state α-discounted problem. We essentially replace the representative states x with *subsets* $I_x \subset \{1, \ldots, n\}$ of the original state space.

General Aggregation Framework

We introduce a finite subset $\mathcal{A}$ of aggregate states, which we denote by symbols such as x and y. We define:

(a) A collection of disjoint subsets $I_x \subset \{1, \ldots, n\}$, $x \in \mathcal{A}$.

(b) A probability distribution over $\{1, \ldots, n\}$ for each $x \in \mathcal{A}$, denoted by $\{d_{xi} \mid i = 1, \ldots, n\}$, and referred to the *disaggregation probabilities of* x. We require that the distribution corresponding to x is concentrated on the subset I_x:

$$d_{xi} = 0, \qquad \text{for all } i \notin I_x,\ x \in \mathcal{A}. \tag{3.62}$$

(c) For each original system state $j \in \{1, \ldots, n\}$, a probability distribution over $\mathcal{A}$, denoted by $\{\phi_{jy} \mid y \in \mathcal{A}\}$, and referred to as the *aggregation probabilities of* j. We require that

$$\phi_{jy} = 1, \qquad \text{for all } j \in I_y,\ y \in \mathcal{A}. \tag{3.63}$$

The aggregation and disaggregation probabilities specify a dynamic system involving both aggregate and original system states; cf. Fig. 3.6.6. In this system:

(i) From aggregate state x, we generate an original system state $i \in I_x$ according to d_{xi}.

(ii) We generate transitions between original system states i and j according to $p_{ij}(u)$, with cost $g(i, u, j)$.

(iii) From original system state j, we generate aggregate state y according to ϕ_{jy}. [Note that in view of Eq. (3.63), all states j within a set I_y, are aggregated onto y with $\phi_{jy} = 1$.]

The optimal costs of the aggregate states $y \in \mathcal{A}$ in the aggregate problem are denoted by r^*_y, and they define approximate costs for the original problem through the interpolation formula

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r^*_y, \qquad j = 1, \ldots, n. \tag{3.64}$$

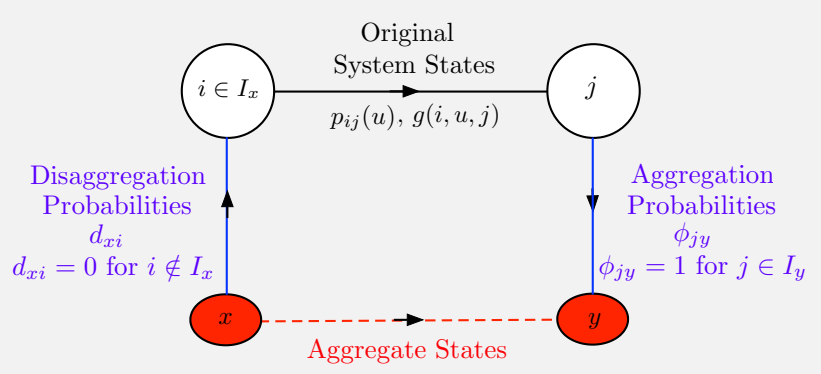

Figure 3.6.6 Illustration of the aggregate system, and the transition mechanism and the costs per stage of the aggregate problem.

Our general aggregation framework is illustrated in Fig. 3.6.6. The sets I_x are often constructed by using features, however, it is helpful to formulate our aggregation framework in a general form, and introduce features later. Note that if each set I_x consists of a single state, we obtain the representative states framework of the preceding section. In this case the disaggregation distribution $\{d_{xi} \mid i \in I_x\}$ is just the atomic distribution that assigns probability 1 to the unique state in I_x. Consistent with the special case of representative states, the disaggregation probability d_{xi} may be interpreted as a "measure of the relation of x and i."

The aggregate problem is a DP problem with an enlarged state space that consists of two copies of the original state space $\{1, \ldots, n\}$ plus the set of aggregate states $\mathcal{A}$. We introduce the corresponding optimal vectors $\tilde{J}_0$, $\tilde{J}_1$, and $r^* = \{r_x^* \mid x \in \mathcal{A}\}$ where:

r_x^* is the optimal cost-to-go from aggregate state x.

$\tilde{J}_0(i)$ is the optimal cost-to-go from original system state i that has just been generated from an aggregate state (left side of Fig. 3.6.6).

$\tilde{J}_1(j)$ is the optimal cost-to-go from original system state j that has just been generated from an original system state (right side of Fig. 3.6.6).

Note that because of the intermediate transitions to aggregate states, $\tilde{J}_0$ and $\tilde{J}_1$ are different.

These three vectors satisfy the following three Bellman equations:

$$r_x^* = \sum_{i \in I_x} d_{xi} \tilde{J}_0(i), \qquad x \in \mathcal{A}, \tag{3.65}$$

$$\tilde{J}_0(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_1(j)\big), \qquad i = 1, \ldots, n, \tag{3.66}$$

$$\tilde{J}_1(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n. \tag{3.67}$$

The objective is to solve for the optimal costs r_x^* of the aggregate states in order to obtain approximate costs for the original problem through the interpolation formula

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \qquad j = 1, \ldots, n;$$

cf. Eq. (3.64).

By combining the three Bellman equations (3.65)-(3.67), we see that r^* satisfies

$$r_x^* = \sum_{i \in I_x} d_{xi} \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \sum_{y \in \mathcal{A}} \phi_{jy}\, r_y^* \right), \qquad x \in \mathcal{A}, \tag{3.68}$$

or equivalently $r^* = Hr^*$, where H is the operator that maps the vector r to the vector Hr with components

$$(Hr)(x) = \sum_{i \in I_x} d_{xi} \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \sum_{y \in \mathcal{A}} \phi_{jy}\, r_y \right), \quad x \in \mathcal{A}. \tag{3.69}$$

It can be shown that *H is a contraction mapping with respect to the maximum norm*, and thus the composite Bellman equation (3.68) has r^* as its unique solution. To see this, we note for any vectors r and r', we have

$$\begin{aligned}
(Hr)(x) &= \sum_{i \in I_x} d_{xi} \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \sum_{y \in \mathcal{A}} \phi_{jy}\, r_y \right) \\
&\leq \sum_{i \in I_x} d_{xi} \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \Big(g(i,u,j) + \alpha \sum_{y \in \mathcal{A}} \phi_{jy} \left(r'_y + \|r - r'\| \right) \Big) \\
&= (Hr')(x) + \alpha \|r - r'\|,
\end{aligned}$$

where $\|\cdot\|$ is the maximum norm, and the equality follows from the definition of $(Hr')(x)$, and the fact that d_{xi}, $p_{ij}(u)$, and ϕ_{jy} are probabilities. Thus we have

$$(Hr)(x) - (Hr')(x) \leq \alpha \|r - r'\|, \qquad x \in \mathcal{A}.$$

By reversing the roles of r and r', we also have

$$(Hr')(x) - (Hr)(x) \leq \alpha \|r - r'\|, \qquad x \in \mathcal{A},$$

so that

$$\big|(Hr')(x) - (Hr)(x)\big| \leq \alpha \|r - r'\|, \qquad x \in \mathcal{A}.$$

By taking the maximum over $x \in \mathcal{A}$, it follows that

$$\|Hr - Hr'\| \leq \alpha \|r - r'\|,$$

and that H is a maximum norm contraction.

Note that the composite Bellman equation (3.68) has dimension equal to the number of aggregate states, which is potentially much smaller than n. To apply the aggregation framework of this section, we may solve exactly this equation for the optimal aggregate costs r^*_x, $x \in \mathcal{A}$, by simulation-based analogs of the VI and PI methods, and obtain a cost function approximation for the original problem through the interpolation formula (3.64). We will develop these methods later, but before doing so, we discuss various ways to formulate the aggregation framework, and in particular, how features can be used for this purpose.

3.6.5 Hard Aggregation and Error Bounds

Let us consider the special case of *hard aggregation*, where for every state j, we have $\phi_{jy} = 0$ for all aggregate states y, except a single one, denoted y_j, for which we have $\phi_{jy_j} = 1$. In this case, the one-step lookahead approximation

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r^*_y, \qquad j = 1, \ldots, n,$$

is piecewise constant; it is constant and equal to r^*_y for all j in the set

$$S_y = \{j \mid \phi_{jy} = 1\}, \qquad y \in \mathcal{A}, \tag{3.70}$$

called the *footprint* of aggregate state y; see Fig. 3.6.4. Note that the footprints of all the aggregate states are disjoint and form a partition of the state space, i.e.,

$$\cup_{x \in \mathcal{A}} S_x = \{1, \ldots, n\}.$$

We can show the following error bound, due to Tsitsiklis and Van Roy [TsV96]; a generalization of this error bound will be given later in this section.

Proposition 3.6.1: (Error Bound for Hard Aggregation) In the case of hard aggregation, we have

$$\big|J^*(j) - \tilde{J}(j)\big| \leq \frac{\epsilon}{1-\alpha}, \qquad \text{for all } j \text{ such that } j \in S_y,\ y \in \mathcal{A},$$

where ϵ is the maximum variation of the optimal cost function J^* over the footprint sets S_y, $y \in \mathcal{A}$:

$$\epsilon = \max_{y \in \mathcal{A}} \max_{i,j \in S_y} \big|J^*(i) - J^*(j)\big|.$$

The meaning of the preceding proposition is that if the optimal cost function J^* varies by at most ϵ within each set S_y, the hard aggregation scheme yields a piecewise constant approximation to the optimal cost function that is within $\epsilon/(1-\alpha)$ of the optimal.

Aside from its intuitive nature and error bound properties, hard aggregation provides a connection with another major approach for approximation in value space, the so called the *projected equation approach*, which we have not discussed here; see the books [Ber12] and [Ber19a]. In particular, it can be shown that for a given policy, the corresponding composite Bellman equation (3.68) for approximate evaluation of μ can be viewed as a projected equation, where a projection seminorm is used that is defined by the disaggregation probabilities; see the paper by Yu and Bertsekas [YuB12] (Section 5.5), or the book [Ber12] (Exercise 6.10).

Selecting the Aggregate States

Generally, the method to select the aggregate states is an important issue, for which there is no mathematical theory at present. However, in practical problems, based on intuition and problem-specific knowledge, there are usually evident choices, which may be fine-tuned by experimentation. For example, suppose that the optimal cost function J^* is piecewise constant over a partition $\{S_y \mid y \in \mathcal{A}\}$ of the state space $\{1, \ldots, n\}$. By this we mean that for some vector

$$r^* = \big\{r_y^* \mid y \in \mathcal{A}\big\},$$

we have

$$J^*(j) = r_y^* \qquad \text{for all } j \in S_y,\ y \in \mathcal{A}.$$

Then from Prop. 3.6.1 it follows that the hard aggregation scheme with $I_x = S_x$ for all $x \in \mathcal{A}$ is exact, so r_x^* are the optimal costs of the aggregate states x in the aggregate problem. This suggests that *in hard aggregation,*

the states in the footprint set S_y corresponding to an aggregate state y should have roughly equal optimal cost, consistently with the error bound of Prop. 3.6.1.

As an extension of the preceding argument, suppose that through some special insight into the problem's structure or some preliminary calculation, we know some features of the system's state that can "predict well" its optimal cost when combined through some approximation architecture, e.g., one that is linear. Then it seems reasonable to form the set aggregate states $\mathcal{A}$ of a hard aggregation scheme so that the sets I_y and S_y consist of states with "similar features" for every $y \in \mathcal{A}$. This is called *feature-based aggregation*, and was suggested in the neuro-dynamic book [BeT96], Section 3.1.2. The next section considers this possibility, and provides a way to introduce features and nonlinearities into the aggregation architecture, without compromising its other favorable aspects.

3.6.6 Aggregation Using Features

Let us consider the guideline for hard aggregation that we just discussed: *states i that belong to the same footprint set S_y should have nearly equal optimal costs*, i.e.,

$$\max_{i,j \in S_y} \left| J^*(i) - J^*(j) \right| \approx 0, \qquad \text{for all } y \in \mathcal{A}.$$

The question now is how to select the sets S_y according to this guideline.

An idea that comes to mind is to use a *feature mapping*, i.e., a function F that maps a state i into an m-dimensional feature vector $F(i)$; cf. Example 3.1.7. In particular, suppose that F has the property that states i with nearly equal feature vector have nearly equal optimal cost $J^*(i)$. Then we can form the sets S_y by grouping together states with nearly equal feature vector. In particular, given F, we introduce a more or less regular partition of the feature space [the subset of $\Re^m$ that consists of all possible feature vectors $F(i)$]. The partition of the feature space induces a possibly irregular collection of subsets of the original state space. Each of these subsets is then used as the footprint of a distinct aggregate state; see Fig. 3.6.7.

Note that in the resulting aggregation scheme the number of aggregate states may become very large. On the other hand, there is a significant advantage over the linear feature-based architectures of Section 3.1, which assign a single weight to each feature: in feature-based hard aggregation we are assigning *a weight to each subset of the feature space partition* (possibly a weight to every possible feature value, in the extreme case where each feature value is viewed by itself as a distinct set of the partition). In effect we use aggregation to construct a *nonlinear* (piecewise constant) feature-based architecture, which may be much more powerful than the corresponding linear architecture.

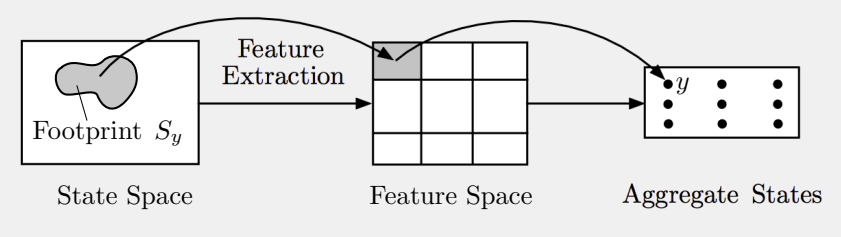

Figure 3.6.7 Feature-based hard aggregation using a partition of the space of features. Each aggregate state y has a footprint S_y that consists of states with "similar" features, i.e., states that map into the same subset of a partition in the space of features.

The question now arises how to obtain a suitable feature vector when there is no obvious choice, based on problem-specific considerations. One possibility, discussed in the book [Ber19a] (Section 6.4), is to obtain "good" features by using a neural network. In fact any method that automatically generates features from data may be used. Here we will discuss a simple possibility.

Using Scoring Functions

Suppose that we have obtained in some way a real-valued *scoring function* $V(i)$ of the state i, which serves as an index of undesirability of state i as a starting state (smaller values of V are assigned to more desirable states, consistent with the view of V as some form of "cost" function). One possibility is to use as V an approximation of the cost function of some "good" (e.g., near-optimal) policy. Another possibility is to obtain V by problem approximation, i.e., as the cost function of some reasonable policy applied to an approximation of the original problem. Still another possibility is to obtain V by training a neural network or other architecture using samples of state-cost pairs obtained by using a software or human expert, and some supervised learning technique.

Given the scoring function V, we will construct a feature mapping that groups together states i with roughly equal scores $V(i)$. In particular, we let R_x, $x = 1, \ldots, q$, be q disjoint intervals that form a partition of the range of possible values of V [i.e., are such that for any state i, there is a unique interval R_x such that $V(i) \in R_x$]. We define a feature vector $F(i)$ of the state i according to

$$F(i) = x, \qquad \text{for all } i \text{ such that } V(i) \in R_x, \quad x = 1, \ldots, q. \tag{3.71}$$

This feature in turn defines a partition of the state space into the sets

$$I_x = \big\{ i \mid F(i) = x \big\} = \big\{ i \mid V(i) \in R_x \big\}, \qquad x = 1, \ldots, q. \tag{3.72}$$

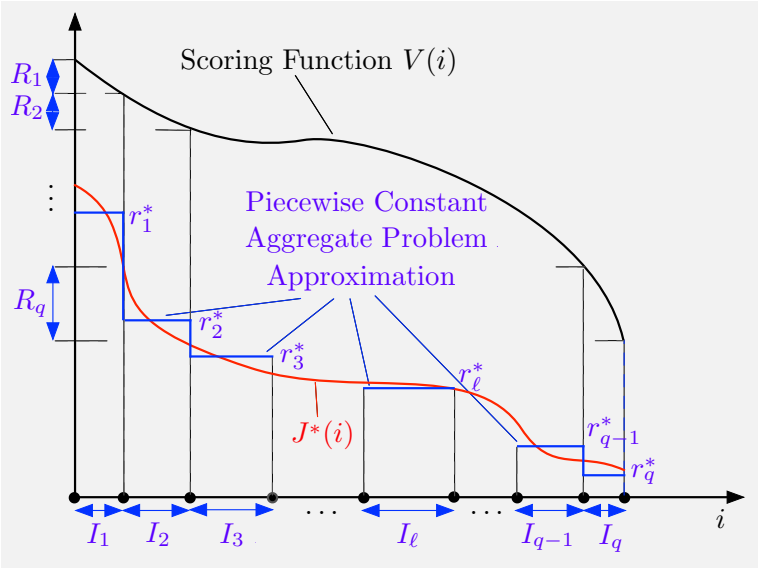

Figure 3.6.8. Hard aggregation scheme based on a single scoring function. We introduce q disjoint intervals $R_1, \ldots, R_q$ that form a partition of the set of possible values of V, and we define a feature vector $F(i)$ of the state i according to

$$F(i) = x, \qquad \text{for all } i \text{ such that } V(i) \in R_x,\ x = 1, \ldots, q.$$

This feature vector in turn defines a partition of the state space into the sets

$$I_x = \big\{ i \mid F(i) = x \big\} = \big\{ i \mid V(i) \in R_x \big\}, \qquad x = 1, \ldots, q.$$

The sets I_x coincide with the footprint sets S_x, and the solution of the aggregate problem yields a piecewise constant approximation of the optimal cost function of the original problem.

Assuming that all the sets I_x are nonempty, we thus obtain a hard aggregation scheme, where the aggregate states are $x = 1, \ldots, q$, and the aggregation probabilities are defined by

$$\phi_{jx} = \begin{cases} 1 & \text{if } j \in I_x, \\ 0 & \text{otherwise,} \end{cases} \qquad j = 1, \ldots, n,\ x = 1, \ldots, q, \tag{3.73}$$

see Fig. 3.6.8. Note that the sets I_x coincide with the footprint sets S_x.

The following proposition (due to Tsitsiklis and Van Roy [TsV96]) illustrates the important role of the *quantization error*, defined as

$$\delta = \max_{x=1,\ldots,q} \max_{i,j \in I_x} \big| V(i) - V(j) \big|. \tag{3.74}$$

It represents the maximum error that can be incurred by approximating V within each set I_x with a single value from its range within the subset. Its

proof with additional discussion can be found in Chapter 6 of the author's RL book [Ber19a].

Proposition 3.6.2: Consider the hard aggregation scheme defined by a scoring function V as described above. Assume that the variations of J^* and V over the sets $I_1, \ldots, I_q$ are within a factor $\beta \geq 0$ of each other, i.e., that

$$\big|J^*(i) - J^*(j)\big| \leq \beta \big|V(i) - V(j)\big|, \qquad \text{for all } i, j \in I_x,\ x = 1, \ldots, q.$$

(a) We have

$$\big|J^*(i) - r_x^*\big| \leq \frac{\beta\delta}{1-\alpha}, \qquad \text{for all } i \in I_x,\ x = 1, \ldots, q,$$

where δ is the quantization error of Eq. (3.74).

(b) Assume that there is no quantization error, i.e., V and J^* are constant within each set I_x. Then the aggregation scheme yields the optimal cost function J^* exactly, i.e.,

$$J^*(i) = r_x^*, \qquad \text{for all } i \in I_x,\ x = 1, \ldots, q.$$

3.6.7 Biased Aggregation

In this section we will introduce an extension of the preceding aggregation framework.† It involves a vector

$$V = \big(V(1), \ldots, V(n)\big)$$

called the *bias vector* or *bias function*, which affects the cost structure of the aggregate problem, and biases the values of its optimal cost function towards their correct levels. When $V = 0$, we will obtain the aggregation scheme of Section 3.6.4. When $V \neq 0$, we will obtain a different aggregation scheme, which yields an approximation to J^* that is equal to V plus a local correction; see Fig. 3.6.10. In this case the aggregate DP problem aims to provide a correction/improvement to V, which may itself be a reasonably good estimate of J^*.

An obvious context where biased aggregation can be used is to improve on an approximation to J^* obtained using a different method, such

† The aggregation framework of this section was proposed in the author's paper [Ber18c], which contains much additional material.

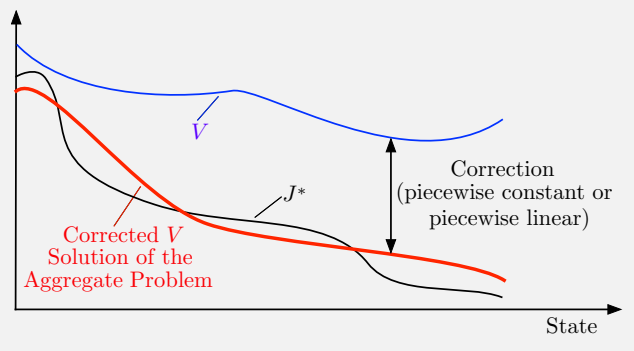

Figure 3.6.10 Schematic illustration of biased aggregation. It provides an approximation to J^* that is equal to the bias function V plus a local correction. When $V = 0$, we obtain the classical aggregation framework.

as for example by neural network-based approximate PI, by rollout, or by problem approximation. Generally, we may speculate that if V captures a fair amount of the nonlinearity of J^*, we may reduce the number of aggregate states needed for adequate performance.

Let us now formulate the aggregate problem in biased aggregation. It is a discounted infinite horizon problem that is similar to the (unbiased) aggregate problem of Section 3.6.4. It involves three sets of states: two copies of the original state space, denoted I_0 and I_1, as well as a finite set $\mathcal{A}$ of aggregate states, as depicted in Fig. 3.6.11. The state transitions in the aggregate problem go from a state in $\mathcal{A}$ to a state in I_0, according to disaggregation probabilities, then to a state in I_1, and then back to a state in $\mathcal{A}$, according to aggregation probabilities, and the process is repeated. At state $i \in I_0$ we must choose a control $u \in U(i)$, and then transition to a state $j \in I_1$ at a cost $g(i, u, j)$ according to the original system transition probabilities $p_{ij}(u)$.

The salient new characteristic of the biased aggregation scheme is a (possibly nonzero) cost $-V(i)$ for transition from any aggregate state to a state $i \in I_0$, and of a cost $V(j)$ from a state $j \in I_1$ to any aggregate state; cf. Fig. 3.6.11. The function V is the bias function, and we will argue that V *should be chosen as close as possible to* J^*. Moreover, for practical purposes its values at various states should be easily computable.

A key insight is that *biased aggregation can be viewed as unbiased aggregation applied to a modified DP problem*, which is equivalent to the original DP problem in the sense that it has the same optimal policies. The modified DP problem is obtained from the original by changing its cost per stage from $g(i, u, j)$ to

$$g(i, u, j) - V(i) + \alpha V(j), \qquad i, j = 1, \ldots, n, \ u \in U(i). \tag{3.75}$$

In particular, by comparing Figs. 3.6.6 and 3.6.11 it can be seen that unbiased aggregation applied to the modified DP problem gives the same

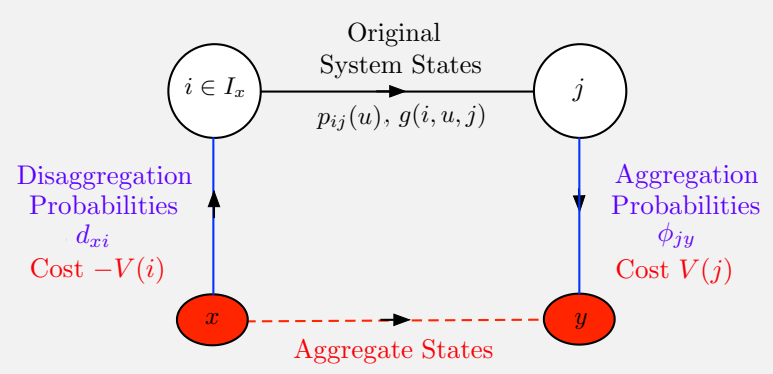

Figure 3.6.11 Illustration of the transition mechanism and the costs per stage of the aggregate problem in the biased aggregation framework. When the bias function V is identically zero, we obtain the aggregation framework of Section 3.6.4.

state-control trajectories as biased aggregation applied to the original DP problem, while the incurred transition costs (from aggregate state to aggregate state) are equal.

Moreover, there is a close connection between the optimal cost functions of the modified DP problem with cost per stage given by Eq. (3.75), and the original DP problem. In particular, the optimal cost function of the modified problem, call it $\tilde{J}$, satisfies the corresponding Bellman equation:

$$\tilde{J}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) - V(i) + \alpha V(j) + \alpha \tilde{J}(j)\big), \quad i = 1, \ldots, n,$$

or equivalently

$$\tilde{J}(i) + V(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha\big(\tilde{J}(j) + V(j)\big)\Big), \quad i = 1, \ldots, n.$$

By comparing this equation with the Bellman equation for the original problem, we see that the optimal cost functions of the modified and the original problems are related by

$$J^*(i) = \tilde{J}(i) + V(i), \qquad i = 1, \ldots, n,$$

and that the two problems have the same optimal policies. This of course assumes that the original and modified problems are solved exactly. If instead they are solved approximately using aggregation or another approximation architecture, such as a neural network, the policies obtained may be substantially different. In particular, the choice of V and the approximation architecture may affect substantially the quality of suboptimal policies obtained.

To summarize, any unbiased aggregation scheme and algorithm, when applied to the modified DP problem with cost per stage given by Eq. (3.75), yields a biased aggregation scheme and algorithm for the original DP problem. Thus, we can straightforwardly transfer results, algorithms, and intuition from our earlier unbiased aggregation analysis to the biased aggregation framework, by applying them to the unbiased aggregation framework that corresponds to the modified stage cost (3.75). Moreover, we may use simulation-based algorithms for policy evaluation, policy improvement, and Q-learning for the aggregate problem, with the only requirement that the value $V(i)$ for any state i is available when needed.

3.6.8 Asynchronous Distributed Multiagent Aggregation

Let us now discuss the distributed solution of large-scale discounted DP problems using cost function approximation, multiple agents/processors, and hard aggregation. Here we partition the original system states into aggregate states/subsets $x \in \mathcal{A} = \{x_1, \ldots, x_m\}$, and we envision a network of processors/agents, each updating asynchronously a detailed/exact local cost function, defined on a single aggregate state/subset. Each processor also maintains an aggregate cost for its aggregate state, which is a weighted average of the detailed cost of the (original system) states in the processor's subset, weighted by the corresponding disaggregation probabilities. These aggregate costs are communicated between processors and are used to perform the local updates.

In a synchronous VI method of this type, each processor $\ell = 1, \ldots, m$, maintains/updates a (local) cost $J(i)$ for every original system state $i \in x_\ell$, and an aggregate cost

$$R(\ell) = \sum_{i \in x_\ell} d_{x_\ell i} J(i),$$

where $d_{x_\ell i}$ are the corresponding disaggregation probabilities. We generically denote by J and R the vectors with components $J(i)$, $i = 1, \ldots, n$, and $R(\ell)$, $\ell = 1, \ldots, m$, respectively. These components are updated according to

$$J_{k+1}(i) = \min_{u \in U(i)} H_\ell(i, u, J_k, R_k), \qquad \forall\ i \in x_\ell, \tag{3.76}$$

with

$$R_k(\ell) = \sum_{i \in x_\ell} d_{x_\ell i} J_k(i), \qquad \ell = 1, \ldots, m, \tag{3.77}$$

where the mapping H_ℓ is defined for all $\ell = 1, \ldots, m$, $i \in x_\ell$, $u \in U(i)$, and $J \in \Re^n$, $R \in \Re^m$, by

$$H_\ell(i, u, J, R) = \sum_{j=1}^{n} p_{ij}(u) g(i, u, j) + \alpha \sum_{j \in x_\ell} p_{ij}(u) J(j) + \alpha \sum_{j \notin x_\ell} p_{ij}(u) R\big(x(j)\big), \tag{3.78}$$

and where for each original system state j, we denote by $x(j)$ the subset to which j belongs [i.e., $j \in x(j)$]. Thus the iteration (3.76) is the same as ordinary VI, except that instead of $J(j)$, we use the aggregate costs $R\big(x(j)\big)$ for the states j whose costs are updated by other processors.

It is possible to show that the iteration (3.76)-(3.77) involves a sup-norm contraction mapping of modulus α, so it converges to the unique solution of the system of equations in (J, R)

$$J(i) = \min_{u \in U(i)} H_\ell(i, u, J, R), \qquad R(\ell) = \sum_{i \in x_\ell} d_{x_\ell i} J(i), \qquad \forall\ i \in x_\ell,\ \ell = 1, \ldots, m. \tag{3.79}$$

This follows from the fact that $\{d_{x_\ell i} \mid i = 1, \ldots, n\}$ is a probability distribution. We may view the equations (3.79) as a set of Bellman equations for an "aggregate" DP problem, which similar to our earlier discussion, involves both the original and the aggregate system states. The difference from the Bellman equations (3.65)-(3.67) is that the mapping (3.78) involves $J(j)$ rather than $R\big(x(j)\big)$ for $j \in x_\ell$.

In the algorithm (3.76)-(3.77), all processors ℓ must be updating their local costs $J(i)$ and aggregate costs $R(\ell)$ synchronously, and communicate the aggregate costs to the other processors before a new iteration may begin. This is often impractical and time-wasting. In a more practical asynchronous version of the method, the aggregate costs $R(\ell)$ may be outdated to account for communication "delays" between processors. Moreover, the costs $J(i)$ need not be updated for all i; it is sufficient that they are updated by each processor ℓ only for a (possibly empty) subset of $I_{\ell,k}$ of the aggregate state/set x_ℓ. In this case, the iteration (3.76)-(3.77) is modified to take the form

$$J_{k+1}(i) = \min_{u \in U(i)} H_\ell\big(i, u, J_k, R_{\tau_{1,k}}(1), \ldots, R_{\tau_{m,k}}(m)\big), \qquad \forall\ i \in I_{\ell,k}, \tag{3.80}$$

with $0 \leq \tau_{\ell,k} \leq k$ for $\ell = 1, \ldots, m$, and

$$R_\tau(\ell) = \sum_{i \in x_\ell} d_{x_\ell i} J_\tau(i), \qquad \forall\ \ell = 1, \ldots, m.$$

The differences $k - \tau_{\ell,k}$, $\ell = 1, \ldots, m$, in Eq. (3.80) may be viewed as "delays" between the current time k and the times $\tau_{\ell,k}$ when the corresponding aggregate costs were computed at other processors. For convergence, it is of course essential that every $i \in x_\ell$ belongs to $I_{\ell,k}$ for infinitely many k (so each cost component is updated infinitely often), and $\lim_{k\to\infty} \tau_{\ell,k} = \infty$ for all $\ell = 1, \ldots, m$ (so that processors eventually communicate more recently computed aggregate costs to other processors).

The convergence of this type of method based on the sup-norm contraction property of the mapping underlying Eq. (3.79), can be established

using an asynchronous convergence theory for DP developed by the author in the paper [Ber82] (see also the books [BeT89], [Ber12]). The monotonicity property is also sufficient to establish convergence, and this is useful in the convergence analysis of related aggregation algorithms for nondiscounted DP models (see the paper by Bertsekas and Yu [BeY10]).

3.7 NOTES AND SOURCES

Section 3.1: Our discussion of approximation architectures, neural networks, and training has been limited, and aimed just to provide the connection with approximate DP. The literature on the subject is vast, and some of the textbooks mentioned in the references to Chapter 1 provide detailed accounts and many sources, in addition to the ones given in Sections 3.1 and 3.2.

There are two broad directions of inquiry in parametric architectures:

(1) The design of architectures, either in a general or a problem-specific context.

(2) The training of neural networks, as well as other linear and nonlinear architectures.

Research along both of these directions has been extensive and is continuing.

Methods for selection of basis functions have received much attention, particularly in the context of neural network research and deep reinforcement learning (see e.g., the book by Goodfellow, Bengio, and Courville [GBC16]). For discussions that are focused outside the neural network area, see Bertsekas and Tsitsiklis [BeT96], Keller, Mannor, and Precup [KMP06], Jung and Polani [JuP07], Bertsekas and Yu [BeY09], and Bhatnagar, Borkar, and Prashanth [BBP13]. Moreover, there has been considerable research on optimal feature selection within given parametric classes (see Menache, Mannor, and Shimkin [MMS05], Yu and Bertsekas [YuB09], Busoniu et al. [BBD10a], and Di Castro and Mannor [DiM10]).

Incremental algorithms are the principal methods for training approximation architectures. They are supported by substantial theoretical analysis, which addresses issues of convergence, rate of convergence, stepsize selection, and component order selection. Moreover, incremental algorithms have been extended to constrained optimization settings, where the constraints are also treated incrementally, first by Nedić [Ned11], and then by several other authors: Bertsekas [Ber11a], Wang and Bertsekas [WaB15], [WaB16], Bianchi [Bia16], Iusem, Jofre, and Thompson [IJT18]. It is beyond our scope to cover this analysis. The author's surveys [Ber10a] and [Ber15b], and convex optimization and nonlinear programming textbooks [Ber15a], [Ber16], collectively contain an extensive account of incremental methods, including the Kaczmarz, incremental gradient, subgradient, ag-

gregated gradient, Newton, Gauss-Newton, and extended Kalman filtering methods, and give many references. The book [BeT96] and paper [BeT00] by Bertsekas and Tsitsiklis, and the survey by Bottou, Curtis, and Nocedal [BCN18] provide theoretically oriented treatments.

Section 3.2: The publicly and commercially available neural network training programs incorporate heuristics for scaling and preprocessing data, stepsize selection, initialization, etc, which can be very effective in specialized problem domains. We refer to books on neural networks such as Bishop [Bis95], Goodfellow, Bengio, and Courville [GBC16], Haykin [Hay08]. The recent book by Bishop and Bishop [BiB24] includes discussions of deep neural networks and transformers.

Deep neural networks have created a lot of excitement in the machine learning field, in view of some high profile successes in image and speech recognition, and in RL with the AlphaGo and AlphaZero programs. One question is whether and for what classes of target functions we can enhance approximation power by increasing the number of layers while keeping the number of weights constant. For discussion, analysis, and speculation around this question, see Bengio [Ben09], Liang and Srikant [LiS16], Yarotsky [Yar17], and Daubechies et al. [DDF19].

Another important research question relates to the role of overparametrization in the success of deep neural networks. With more weights than training data, the training problem has infinitely many solutions, each providing an architecture that fits the training data perfectly. The question then is how to select a solution that works well on test data (i.e., data outside the training set); see Zhang et al. [ZBH16], [ZBH21], Belkin, Ma, and Mandal [BMM18], Belkin, Rakhlin, and Tsybakov [BRT18], Soltanolkotabi, Javanmard, and Lee [SJL18], Bartlett et al. [BLL19], Hastie et al. [HMR19], Muthukumar, Vodrahalli, and Sahai [MVS19], Su and Yang [SuY19], Sun [Sun19], Vaswani et al. [VLK21], Zhang et al. [ZBH21], and the discussions in the machine learning books by Hardt and Recht [HaR21], and Bishop and Bishop [BiB24]. Finally, transformers are coming into the mainstream of RL-based training contexts, and have generated a renewed interest in training methods for approximate policy iteration.

Section 3.3: Fitted value iteration has a long history; it was mentioned by Bellman among others. It has interesting properties, and at times exhibits pathological/unstable behavior due to accumulation of errors over a long horizon (see [Ber19a], Section 5.2).

The approximate policy iteration method of Section 3.3.3 has been proposed by Fern, Yoon, and Givan [FYG06], and variants have also been discussed and analyzed by several other authors. The method (with some variations) has been used to train a tetris playing computer program that performs impressively better than programs that are based on other variants of approximate policy iteration, and various other methods; see Scherrer [Sch13], Scherrer et al. [SGG15], and Gabillon, Ghavamzadeh, and

Scherrer [GGS13], who also provide an analysis of the method. The RL and approximate DP books collectively describe several alternative simulation-based methods for policy evaluation; see e.g., [BeT96], [SuB18], [Ber12], Chapters 6 and 7. These include the popular temporal difference methods, which are also closely related to Galerkin approximation, a major computational approach for solving large scale equations, as first observed by Yu and Bertsekas [YuB10], and Bertsekas [Ber11c]; see also Szepesvari [Sze11]. The book [Ber20a] describes distributed versions of approximate policy iteration, which are based on partitioning of the state space.

The original proposal of SARSA (Section 3.3.4) is attributed to Rummery and Niranjan [RuN94], with related work presented in the papers by Peng and Williams [PeW96], and Wiering and Schmidhuber [WiS98]. The ideas of the DQN algorithm attracted much attention following the paper by Mnih et al. [MKS15], which reported impressive test results on a suite of 49 classic Atari 2600 games.

The rollout and approximate PI methodology for POMDP of Section 3.3.5 was described in the author's RL book [Ber19a]. It was extended and tested in the paper by Bhattacharya et al. [BBW20] in the context of a challenging pipeline repair problem.

Advantage updating (Section 3.3.6) was proposed by Baird [Bai93], [Bai94], and is discussed further in Section 6.6 of the neuro-dynamic programming book [BeT96]. The differential training methodology (Section 3.3.7) was proposed by the author in the paper [Ber97b], and followup work was presented by Weaver and Baxter [WeB99].

Generally, the challenges of implementing successfully approximate value and policy iteration schemes are quite formidable, and tend to be underestimated, because the literature naturally tends to place emphasis on success stories, and tends to underreport failures. In practice, the training difficulties, particularly exploration, must often be addressed on a case-by-case basis, and may require long and tricky parameter tuning issues, with little guarantee of ultimate success or even a diagnosis of the causes of failure. By contrast, approximation in value space with long multistep lookahead and simple terminal cost function approximation, and rollout (a single policy iteration starting from a base policy), while less ambitious, are typically much easier to implement in practice, and often attain at least some modest success rather quickly. An intermediate approach that often works well is to use truncated rollout with a terminal cost function approximation that is trained with data.

Section 3.4: Classification (sometimes called "pattern classification" or "pattern recognition") is a major subject in machine learning, for which there are many approaches, an extensive literature, and an abundance of public domain and commercial software; see e.g. the textbooks by Bishop [Bis95], [Bis06], Duda, Hart, and Stork [DHS12], and Hardt and Recht [HaR21]. Approximation in policy space was formulated as a classifi-

cation problem in the context of DP by Lagoudakis and Parr [LaP03], and was followed up by several other authors (see e.g., Dimitrakakis and Lagoudakis [DiL08], Lazaric, Ghavamzadeh, and Munos [LGM10], Gabillon et al. [GLG11], Liu and Wei [LiW14], Farahmand et al. [FPB15], and the references quoted there). While we have focused on a classification approach that makes use of least squares regression and a parametric architecture, other classification methods may also be used. For example the paper [LaP03] discusses the use of nearest neighbor schemes, support vector machines, as well as neural networks.

Section 3.5: Our coverage of policy gradient, and random search methods has been limited, and aimed to provide an entry point into the field. For a detailed discussion and references on policy gradient methods, we refer to the book by Sutton and Barto [SuB18], the monographs by Deisenroth, Neumann, and Peters [DNP11], and the surveys by Peters and Schaal [PeS08], and Grondman et al. [GBL12]. An influential paper in this context by Williams [Wil92] proposed among others the likelihood-ratio policy gradient method given here. The methods of [Wil92] are commonly referred to as REINFORCE in the literature (see e.g., [SuB18], Ch. 13).

There are several related early works on search along randomly chosen directions (Rastrigin [Ras63], Matyas [Mat65], Aleksandrov, Sysoyev, and Shemeneva [ASS68], Rubinstein [Rub69]); see also Spall [Spa92], [Spa03], Duchi, Jordan, Wainwright, and Wibisono [DJW12], [DJW15], and Nesterov and Spokoiny [NeS17], for more modern related works. For early works on simulation-based policy gradient schemes for various DP problems, see Glynn [Gly87], [Gly90], L'Ecuyer [L'Ec91], Fu and Hu [FuH94], Jaakkola, Singh, and Jordan [JSJ95], Cao and Chen [CaC97], Cao and Wan [CaW98]. There are also variants of policy gradient methods that include stabilization heuristics to guard against unusual behavior. An example is the popular proximal policy optimization (PPO) method suggested in the paper by Shulman et al. [SWD17], which also reviews earlier approaches.

The challenge in the successful implementation of policy gradient methods is twofold: the difficulties with slow convergence and local minima that are inherent in gradient optimization, and the detrimental effects of simulation noise. Much work has been directed towards variations that address these difficulties, including the use of a baseline and variance reduction methods (Greensmith, Bartlett, and Baxter [GBB04], Greensmith [Gre05]), and scaling based on the so-called *natural gradient* (Kakade [Kak02]) or second order information (see Wang and Paschalidis [WaP17], and the references quoted there).

We have not covered actor-critic methods within the policy gradient context. Such methods were introduced in the paper by Barto, Sutton, and Anderson [BSA83]. The more recent works of Baxter and Bartlett [BaB01], Konda and Tsitsiklis [KoT99], [KoT03], Marbach and Tsitsiklis [MaT01], [MaT03], and Sutton et al. [SMS99] have been influential. Actor-critic

algorithms that are suitable for POMDP and involve gradient estimation have been given by Yu [Yu05], and Estanjini, Li, and Paschalidis [ELP12].

The cross-entropy method was initially developed in the context of rare event simulation and was later adapted for use in optimization. For textbook accounts, see Rubinstein and Kroese [RuK04], [RuK13], [RuK16], and Busoniu et al. [BBD10a], and for surveys see de Boer et al. [BKM05], and Kroese et al. [KRC13]. The method was proposed for policy search in an approximate DP context by Mannor, Rubinstein, and Gat [MRG03]. It was applied with success to the game of tetris by Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09]. For recent analysis, see Joseph and Bhatnagar [JoB16], [JoB18].

Section 3.6: The aggregation approach has a long history in scientific computation and operations research (see for example Bean, Birge, and Smith [BBS87], Chatelin and Miranker [ChM82], Douglas and Douglas [DoD93], and Rogers et al. [RPW91]). It was introduced in the simulation-based approximate DP context, mostly in the form of VI; see Singh, Jaakkola, and Jordan [SJJ95], Gordon [Gor95], and Tsitsiklis and Van Roy [TsV96]. It was further discussed in the neuro-dynamic programming book [BeT96], Sections 3.1.2 and 6.7.

The aggregation framework with representative features was introduced in the author's DP book [Ber12], was discussed in detail in the RL textbook [Ber19a] (Chapter 6), and was further developed in the author's survey paper [Ber18b], which provides an expanded view of the methodology. Biased aggregation (Section 3.6.7) was introduced in the author's paper [Ber18c], which contains further discussion, connections with rollout algorithms, and additional methods.

Distributed asynchronous aggregation (Section 3.6.8) was first proposed in the paper by Bertsekas and Yu [BeY10] (Example 2.5); see also the discussions in author's DP books [Ber12] (Section 6.5.4) and [Ber22b] (Example 1.2.11). A recent computational study, related to distributed traffic routing, was given by Vertovec and Margellos [VeM23].

Aggregation may also be used as a policy evaluation method in the context of policy iteration with linear feature-based cost function approximations. Within this context, aggregation provides an alternative to temporal difference methods, such as TD(λ), LSTD(λ), and LSPE(λ), which form another major class of policy evaluation methods (not discussed in this book; see [Ber12], [Ber19a]). The aggregation and the temporal difference approaches for policy evaluation are described and compared in the author's approximate policy iteration survey paper [Ber11b]. Generally speaking, aggregation methods are characterized by stronger theoretical properties, such as Bellman operator monotonicity, resilience to policy oscillations, and better error bounds. On the other hand, they are more restrictive in their use of linear approximation architectures, compared with temporal difference methods (see [Ber11b], [Ber18a]).

E X E R C I S E S

3.1 (Proof of Prop. 3.4.1)

Complete the details of the following proof of Prop. 3.4.1. Fix c, and for any scalar y, consider for a given x the conditional expected value $E\{(z(c,c') - y)^2 \mid x\}$. Here the random variable $z(c,c')$ takes the value 1 with probability $p(c \mid x)$ and the value 0 with probability $1 - p(c \mid x)$, so we have

$$E\Big\{\big(z(c,c') - y\big)^2 \;\Big|\; x\Big\} = p(c \mid x)(y-1)^2 + \big(1 - p(c \mid x)\big)y^2.$$

We minimize this expression with respect to y, by setting to 0 its derivative, i.e.,

$$0 = 2p(c \mid x)(y-1) + 2\big(1 - p(c \mid x)\big)y = 2\big(-p(c \mid x) + y\big).$$

We thus obtain the minimizing value of y, namely $y^* = p(c \mid x)$, so that

$$E\Big\{\big(z(c,c') - p(c \mid x)\big)^2 \mid x\Big\} \leq E\Big\{\big(z(c,c') - y\big)^2 \mid x\Big\}, \qquad \text{for all scalars } y.$$

We set $y = h(c,x)$ in the above expression and obtain

$$E\Big\{\big(z(c,c') - p(c \mid x)\big)^2 \mid x\Big\} \leq E\Big\{\big(z(c,c') - h(c,x)\big)^2 \mid x\Big\}.$$

Since this is true for all x, we also have

$$\sum_x \xi(x) E\Big\{\big(z(c,c') - p(c \mid x)\big)^2 \;\Big|\; x\Big\} \leq \sum_x \xi(x) E\Big\{\big(z(c,c') - h(c,x)\big)^2 \;\Big|\; x\Big\},$$

showing that Eq. (3.35) holds for all functions h and all classes c.

References

[ABB19] Agrawal, A., Barratt, S., Boyd, S., and Stellato, B., 2019. "Learning Convex Optimization Control Policies," arXiv:1912.09529.

[ACF02] Auer, P., Cesa-Bianchi, N., and Fischer, P., 2002. "Finite Time Analysis of the Multiarmed Bandit Problem," Machine Learning, Vol. 47, pp. 235-256.

[ADH19] Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R., 2019. "Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks," arXiv:1901.08584.

[AHZ19] Arcari, E., Hewing, L., and Zeilinger, M. N., 2019. "An Approximate Dynamic Programming Approach for Dual Stochastic Model Predictive Control," arXiv:1911.03728.

[ALZ08] Asmuth, J., Littman, M. L., and Zinkov, R., 2008. "Potential-Based Shaping in Model-Based Reinforcement Learning," Proc. of 23rd AAAI Conference, pp. 604-609.

[AMS09] Audibert, J.Y., Munos, R., and Szepesvari, C., 2009. "Exploration-Exploitation Tradeoff Using Variance Estimates in Multi-Armed Bandits," Theoretical Computer Science, Vol. 410, pp. 1876-1902.

[ASR20] Andersen, A. R., Stidsen, T. J. R., and Reinhardt, L. B., 2020. "Simulation-Based Rolling Horizon Scheduling for Operating Theatres," in SN Operations Research Forum, Vol. 1, pp. 1-26.

[ASS68] Aleksandrov, V. M., Sysoyev, V. I, and Shemeneva, V. V., 1968. "Stochastic Optimization of Systems," Engineering Cybernetics, Vol. 5, pp. 11-16.

[AXG16] Ames, A. D., Xu, X., Grizzle, J. W., and Tabuada, P., 2016. "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," IEEE Transactions on Automatic Control, Vol. 62, pp. 3861-3876.

[Abr90] Abramson, B., 1990. "Expected-Outcome: A General Model of Static Evaluation," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 12, pp. 182-193.

[Agr95] Agrawal, R., 1995. "Sample Mean Based Index Policies with $O(\log n)$ Regret for the Multiarmed Bandit Problem," Advances in Applied Probability, Vol. 27, pp. 1054-1078.

[Ala22] Alamir, M., 2022. "Learning Against Uncertainty in Control Engineering," Annual Reviews in Control.

[AnH14] Antunes, D., and Heemels, W.P.M.H., 2014. "Rollout Event-Triggered Control: Beyond Periodic Control Performance," IEEE Transactions on Automatic Control, Vol. 59, pp. 3296-3311.

[AnM79] Anderson, B. D. O., and Moore, J. B., 1979. Optimal Filtering, Prentice-Hall, Englewood Cliffs, N. J.

[AsH06] Aström, K. J., and Hagglund, T., 2006. Advanced PID Control, Instrument Society of America, Research Triangle Park, N. C.

[AsW94] Aström, K. J., and Wittenmark, B., 1994. Adaptive Control, 2nd Edition, Prentice-Hall, Englewood Cliffs, N. J.

[Ast83] Aström, K. J., 1983. "Theory and Applications of Adaptive Control - A Survey," Automatica, Vol. 19, pp. 471-486.

[AtF66] Athans, M., and Falb, P., 1966. Optimal Control, McGraw-Hill, N. Y.

[AvB20] Avrachenkov, K., and Borkar, V. S., 2020. "Whittle Index Based Q-Learning for Restless Bandits with Average Reward," arXiv:2004.14427.

[BBB22] Bhambri, S., Bhattacharjee, A., and Bertsekas, D. P., 2022. "Reinforcement Learning Methods for Wordle: A POMDP/Adaptive Control Approach," arXiv:2211.10298.

[BBB23] Bhambri, S., Bhattacharjee, A., and Bertsekas, D. P., 2023. "Playing Wordle Using an Online Rollout Algorithm for Deterministic POMDPs," 2023 IEEE Conference on Games, Boston, MA.

[BBD08] Busoniu, L., Babuska, R., and De Schutter, B., 2008. "A Comprehensive Survey of Multiagent Reinforcement Learning," IEEE Transactions on Systems, Man, and Cybernetics, Part C, Vol. 38, pp. 156-172.

[BBD10a] Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D., 2010. Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, N. Y.

[BBD10b] Busoniu, L., Babuska, R., and De Schutter, B., 2010. "Multi-Agent Reinforcement Learning: An Overview," in Innovations in Multi-Agent Systems and Applications, Springer, pp. 183-221.

[BBG13] Bertazzi, L., Bosco, A., Guerriero, F., and Lagana, D., 2013. "A Stochastic Inventory Routing Problem with Stock-Out," Transportation Research, Part C, Vol. 27, pp. 89-107.

[BBM17] Borrelli, F., Bemporad, A., and Morari, M., 2017. Predictive Control for Linear and Hybrid Systems, Cambridge Univ. Press, Cambridge, UK.

[BBP13] Bhatnagar, S., Borkar, V. S., and Prashanth, L. A., 2013. "Adaptive Feature Pursuit: Online Adaptation of Features in Reinforcement Learning," in *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 517-534.

[BBS87] Bean, J. C., Birge, J. R., and Smith, R. L., 1987. "Aggregation in Dynamic Programming," Operations Research, Vol. 35, pp. 215-220.

[BBW20] Bhattacharya, S., Badyal, S., Wheeler, T., Gil, S., Bertsekas, D. P., 2020. "Reinforcement Learning for POMDP: Partitioned Rollout and Policy Iteration with Application to Autonomous Sequential Repair Problems," to appear in IEEE Robotics and Automation Letters, 2020; arXiv:2002.04175.

[BCD10] Brochu, E., Cora, V. M., and De Freitas, N., 2010. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," arXiv:1012.2599.

[BCN18] Bottou, L., Curtis, F. E., and Nocedal, J., 2018. "Optimization Methods for Large-Scale Machine Learning," SIAM Review, Vol. 60, pp. 223-311.

[BFA22] Bouguila, N., Fan, W. and Amayri, M., eds., 2022. Hidden Markov Models and Applications. Springer, NY.

[BFH86] Breton, M., Filar, J. A., Haurie, A., and Schultz, T. A., 1986. "On the Computation of Equilibria in Discounted Stochastic Dynamic Games," in Dynamic Games and Applications in Economics, Springer, pp. 64-87.

[BLJ23] Bai, T., Li, Y., Johansson, K. H., and Martensson, J., 2023. "Rollout-Based Charging Strategy for Electric Trucks with Hours-of-Service Regulations," arXiv Preprint, arXiv:2303.08895.

[BKB20] Bhattacharya, S., Kailas, S., Badyal, S., Gil, S., and Bertsekas, D. P., 2020. "Multiagent Rollout and Policy Iteration for POMDP with Application to Multi-Robot Repair Problems," in Proc. of Conference on Robot Learning (CoRL); also arXiv preprint, arXiv:2011.04222.

[BKM05] de Boer, P. T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. 2005. "A Tutorial on the Cross-Entropy Method," Annals of Operations Research, Vol. 134, pp. 19-67.

[BLL19] Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A., 2019. "Benign Overfitting in Linear Regression," arXiv:1906.11300.

[BMM18] Belkin, M., Ma, S., and Mandal, S., 2018. "To Understand Deep Learning we Need to Understand Kernel Learning," arXiv:1802.01396.

[BPW12] Browne, C., Powley, E., Whitehouse, D., Lucas, L., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S., 2012. "A Survey of Monte Carlo Tree Search Methods," IEEE Trans. on Computational Intelligence and AI in Games, Vol. 4, pp. 1-43.

[BRT18] Belkin, M., Rakhlin, A., and Tsybakov, A. B., 2018. "Does Data Interpolation Contradict Statistical Optimality?" arXiv:1806.09471.

[BSA83] Barto, A. G., Sutton, R. S., and Anderson, C. W., 1983. "Neuronlike Elements that Can Solve Difficult Learning Control Problems," IEEE Trans. on Systems, Man, and Cybernetics, Vol. 13, pp. 835-846.

[BTW97] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., 1997. "Rollout Algorithms for Combinatorial Optimization," Heuristics, Vol. 3, pp. 245-262.

[BWL19] Beuchat, P. N., Warrington, J., and Lygeros, J., 2019. "Accelerated Point-Wise Maximum Approach to Approximate Dynamic Programming," arXiv:1901.03619.

[BYB94] Bradtke, S. J., Ydstie, B. E., and Barto, A. G., 1994. "Adaptive Linear Quadratic Control Using Policy Iteration," Proc. IEEE American Control Conference, Vol. 3, pp. 3475-3479.

[BaB01] Baxter, J., and Bartlett, P. L., 2001. "Infinite-Horizon Policy-Gradient Estimation," Journal of Artificial Intelligence Research, Vol. 15, pp. 319-350.

[BaF88] Bar-Shalom, Y., and Fortman, T. E., 1988. Tracking and Data Association, Academic Press, N. Y.

[BaL19] Banjac, G., and Lygeros, J., 2019. "A Data-Driven Policy Iteration Scheme Based on Linear Programming," Proc. 2019 IEEE CDC, pp. 816-821.

[BaP12] Bauso, D., and Pesenti, R., 2012. "Team Theory and Person-by-Person Optimization with Binary Decisions," SIAM Journal on Control and Optimization, Vol. 50, pp. 3011-3028.

[Bai93] Baird, L. C., 1993. "Advantage Updating," Report WL-TR-93-1146, Wright Patterson AFB, OH.

[Bai94] Baird, L. C., 1994. "Reinforcement Learning in Continuous Time: Advantage Updating," International Conf. on Neural Networks, Orlando, Fla.

[Bar90] Bar-Shalom, Y., 1990. Multitarget-Multisensor Tracking: Advanced Applications, Artech House, Norwood, MA.

[BeC89] Bertsekas, D. P., and Castañon, D. A., 1989. "The Auction Algorithm for Transportation Problems," Annals of Operations Research, Vol. 20, pp. 67-96.

[BeC99] Bertsekas, D. P., and Castañon, D. A., 1999. "Rollout Algorithms for Stochastic Scheduling Problems," Heuristics, Vol. 5, pp. 89-108.

[BeC02] Ben-Gal, I., and Caramanis, M., 2002. "Sequential DOE via Dynamic Programming," IIE Transactions, Vol. 34, pp. 1087-1100.

[BeC08] Besse, C., and Chaib-draa, B., 2008. "Parallel Rollout for Online Solution of DEC-POMDPs," Proc. of 21st International FLAIRS Conference, pp. 619-624.

[BeL14] Beyme, S., and Leung, C., 2014. "Rollout Algorithm for Target Search in a Wireless Sensor Network," 80th Vehicular Technology Conference (VTC2014), IEEE, pp. 1-5.

[BeI96] Bertsekas, D. P., and Ioffe, S., 1996. "Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming," Lab. for Info. and Decision Systems Report LIDS-P-2349, Massachusetts Institute of Technology.

[BeP03] Bertsimas, D., and Popescu, I., 2003. "Revenue Management in a Dynamic Network Environment," Transportation Science, Vol. 37, pp. 257-277.

[BeR71a] Bertsekas, D. P., and Rhodes, I. B., 1971. "On the Minimax Reachability of Target Sets and Target Tubes," Automatica, Vol. 7, pp. 233-247.

[BeR71b] Bertsekas, D. P., and Rhodes, I. B., 1971. "Recursive State Estimation for a Set-Membership Description of the Uncertainty," IEEE Trans. Automatic Control, Vol. AC-16, pp. 117-128.

[BeR73] Bertsekas, D. P., and Rhodes, I. B., 1973. "Sufficiently Informative Functions and the Minimax Feedback Control of Uncertain Dynamic Systems," IEEE Trans. Automatic Control, Vol. AC-18, pp. 117-124.

[BeS78] Bertsekas, D. P., and Shreve, S. E., 1978. Stochastic Optimal Control: The Discrete Time Case, Academic Press, N. Y.; republished by Athena Scientific, Belmont, MA, 1996 (can be downloaded in from the author's website).

[BeS18] Bertazzi, L., and Secomandi, N., 2018. "Faster Rollout Search for the Vehicle Routing Problem with Stochastic Demands and Restocking," European J. of Operational Research, Vol. 270, pp.487-497.

[BeT89] Bertsekas, D. P., and Tsitsiklis, J. N., 1989. Parallel and Distributed Computation: Numerical Methods, Prentice-Hall, Englewood Cliffs, N. J.; republished by Athena Scientific, Belmont, MA, 1997 (can be downloaded from the author's website).

[BeT91] Bertsekas, D. P., and Tsitsiklis, J. N., 1991. "An Analysis of Stochastic Shortest Path Problems," Math. Operations Res., Vol. 16, pp. 580-595.

[BeT96] Bertsekas, D. P., and Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming, Athena Scientific, Belmont, MA.

[BeT97] Bertsimas, D., and Tsitsiklis, J. N., 1997. Introduction to Linear Optimization, Athena Scientific, Belmont, MA.

[BeT00] Bertsekas, D. P., and Tsitsiklis, J. N., 2000. "Gradient Convergence of Gradient Methods with Errors," SIAM J. on Optimization, Vol. 36, pp. 627-642.

[BeT08] Bertsekas, D. P., and Tsitsiklis, J. N., 2008. Introduction to Probability, 2nd Edition, Athena Scientific, Belmont, MA.

[BeY09] Bertsekas, D. P., and Yu, H., 2009. "Projected Equation Methods for Approximate Solution of Large Linear Systems," J. of Computational and Applied Math., Vol. 227, pp. 27-50.

[BeY10] Bertsekas, D. P., and Yu, H., 2010. "Asynchronous Distributed Policy Iteration in Dynamic Programming," Proc. of Allerton Conf. on Communication, Control and Computing, Allerton Park, Ill, pp. 1368-1374.

[BeY12] Bertsekas, D. P., and Yu, H., 2012. "Q-Learning and Enhanced Policy Iteration in Discounted Dynamic Programming," Math. of Operations Research, Vol. 37, pp. 66-94.

[BeY16] Bertsekas, D. P., and Yu, H., 2016. "Stochastic Shortest Path Problems Under Weak Conditions," Lab. for Information and Decision Systems Report LIDS-2909, MIT.

[Bel56] Bellman, R., 1956. "A Problem in the Sequential Design of Experiments," Sankhya: The Indian Journal of Statistics, Vol. 16, pp. 221-229.

[Bel57] Bellman, R., 1957. Dynamic Programming, Princeton University Press, Princeton, N. J.

[Bel84] Bellman, R., 1984. Eye of the Hurricane, World Scientific Publishing, Singapore.

[Bel87] Bellman, R., 1987. Introduction to the Mathematical Theory of Control Processes, Academic Press, Vols. I and II, New York, N. Y.

[Ben09] Bengio, Y., 2009. "Learning Deep Architectures for AI," Foundations and Trends in Machine Learning, Vol. 2, pp. 1-127.

[Ber71] Bertsekas, D. P., 1971. "Control of Uncertain Systems With a Set-Membership Description of the Uncertainty," Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA (can be downloaded from the author's website).

[Ber72a] Bertsekas, D. P., 1972. "Infinite Time Reachability of State Space Regions by Using Feedback Control," IEEE Trans. Automatic Control, Vol. AC-17, pp. 604-613.

[Ber72b] Bertsekas, D. P., 1972. "On the Solution of Some Minimax Control Problems," Proc. 1972 IEEE Decision and Control Conf., New Orleans, LA.

[Ber73] Bertsekas, D. P., 1973. "Linear Convex Stochastic Control Problems over an Infinite Horizon," IEEE Trans. Automatic Control, Vol. AC-18, pp. 314-315.

[Ber77] Bertsekas, D. P., 1977. "Monotone Mappings with Application in Dynamic Programming," SIAM J. on Control and Opt., Vol. 15, pp. 438-464.

[Ber79] Bertsekas, D. P., 1979. "A Distributed Algorithm for the Assignment Problem," Lab. for Information and Decision Systems Report, MIT, May 1979.

[Ber82] Bertsekas, D. P., 1982. "Distributed Dynamic Programming," IEEE Trans. Automatic Control, Vol. AC-27, pp. 610-616.

[Ber83] Bertsekas, D. P., 1983. "Asynchronous Distributed Computation of Fixed Points," Math. Programming, Vol. 27, pp. 107-120.

[Ber91] Bertsekas, D. P., 1991. Linear Network Optimization: Algorithms and Codes, MIT Press, Cambridge, MA (can be downloaded from the author's website).

[Ber96] Bertsekas, D. P., 1996. "Incremental Least Squares Methods and the Extended Kalman Filter," SIAM J. on Optimization, Vol. 6, pp. 807-822.

[Ber97a] Bertsekas, D. P., 1997. "A New Class of Incremental Gradient Methods for Least Squares Problems," SIAM J. on Optimization, Vol. 7, pp. 913-926.

[Ber97b] Bertsekas, D. P., 1997. "Differential Training of Rollout Policies," Proc. of the 35th Allerton Conference on Communication, Control, and Computing, Allerton Park, Ill.

[Ber98] Bertsekas, D. P., 1998. Network Optimization: Continuous and Discrete Models, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber05a] Bertsekas, D. P., 2005. "Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC," European J. of Control, Vol. 11, pp. 310-334.

[Ber05b] Bertsekas, D. P., 2005. "Rollout Algorithms for Constrained Dynamic Programming," Lab. for Information and Decision Systems Report LIDS-P-2646, MIT.

[Ber07] Bertsekas, D. P., 2007. "Separable Dynamic Programming and Approximate Decomposition Methods," IEEE Trans. on Aut. Control, Vol. 52, pp. 911-916.

[Ber10a] Bertsekas, D. P., 2010. "Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey," Lab. for Information and Decision Systems Report LIDS-P-2848, MIT; a condensed version with the same title appears in Optimization for Machine Learning, by S. Sra, S. Nowozin, and S. J. Wright, (eds.), MIT Press, Cambridge, MA, 2012, pp. 85-119.

[Ber10b] Bertsekas, D. P., 2010. "Williams-Baird Counterexample for Q-Factor Asynchronous Policy Iteration,"
http://web.mit.edu/dimitrib/www/Williams-Baird Counterexample.pdf.

[Ber10c] Bertsekas, D. P., 2010. "Pathologies of Temporal Difference Methods in Approximate Dynamic Programming," Proc. 2010 IEEE Conference on Decision and Control, Atlanta, GA, Dec. 2010.

[Ber11a] Bertsekas, D. P., 2011. "Incremental Proximal Methods for Large Scale Convex Optimization," Math. Programming, Vol. 129, pp. 163-195.

[Ber11b] Bertsekas, D. P., 2011. "Approximate Policy Iteration: A Survey and Some New Methods," J. of Control Theory and Applications, Vol. 9, pp. 310-335.

[Ber11c] Bertsekas, D. P., 2011. "Temporal Difference Methods for General Projected Equations," IEEE Trans. on Automatic Control, Vol. 56, pp. 2128-2139.

[Ber12] Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, Vol. II, 4th Edition, Athena Scientific, Belmont, MA.

[Ber13a] Bertsekas, D. P., 2013. "Rollout Algorithms for Discrete Optimization: A Survey," Handbook of Combinatorial Optimization, Springer.

[Ber13b] Bertsekas, D. P., 2013. "λ-Policy Iteration: A Review and a New Implementation," in Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 381-409.

[Ber15a] Bertsekas, D. P., 2015. Convex Optimization Algorithms, Athena Scientific, Belmont, MA.

[Ber15b] Bertsekas, D. P., 2015. "Incremental Aggregated Proximal and Augmented Lagrangian Algorithms," Lab. for Information and Decision Systems Report LIDS-P-3176, MIT; arXiv:1507.1365936.

[Ber16] Bertsekas, D. P., 2016. Nonlinear Programming, 3rd Edition, Athena Scientific, Belmont, MA.

[Ber17a] Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, Vol. I, 4th Edition, Athena Scientific, Belmont, MA.

[Ber17b] Bertsekas, D. P., 2017. "Value and Policy Iteration in Deterministic Optimal Control and Adaptive Dynamic Programming," IEEE Transactions on Neural Networks and Learning Systems, Vol. 28, pp. 500-509.

[Ber18a] Bertsekas, D. P., 2018. "Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations," Lab. for Information and Decision Systems Report, MIT; arXiv:1804.04577; IEEE/CAA Journal of Automatica Sinica, Vol. 6, 2019, pp. 1-31.

[Ber18b] Bertsekas, D. P., 2018. "Biased Aggregation, Rollout, and Enhanced Policy Improvement for Reinforcement Learning," Lab. for Information and Decision Systems Report, MIT; arXiv:1910.02426.

[Ber19a] Bertsekas, D. P., 2019. Reinforcement Learning and Optimal Control, Athena Scientific, Belmont, MA.

[Ber19b] Bertsekas, D. P., 2019. "Robust Shortest Path Planning and Semicontractive Dynamic Programming," Naval Research Logistics, Vol. 66, pp. 15-37.

[Ber19c] Bertsekas, D. P., 2019. "Multiagent Rollout Algorithms and Reinforcement Learning," arXiv:1910.00120.

[Ber19d] Bertsekas, D. P., 2019. "Constrained Multiagent Rollout and Multidimensional Assignment with the Auction Algorithm," arXiv preprint, arxiv:2002.07407.

[Ber20a] Bertsekas, D. P., 2020. Rollout, Policy Iteration, and Distributed Reinforcement Learning, Athena Scientific, Belmont, MA.

[Ber20b] Bertsekas, D. P., 2020. "Multiagent Value Iteration Algorithms in Dynamic Programming and Reinforcement Learning," arXiv preprint, arxiv.org/abs/2005.01627; appears in Results in Control and Optimization Journal, Vol. 1, 2020.

[Ber21a] Bertsekas, D. P., 2021. "Multiagent Reinforcement Learning: Rollout and Policy Iteration," IEEE/CAA Journal of Automatica Sinica, Vol. 8, pp. 249-271.

[Ber21b] Bertsekas, D. P., 2021. "Distributed Asynchronous Policy Iteration for Sequential Zero-Sum Games and Minimax Control," arXiv:2107.10406

[Ber22a] Bertsekas, D. P., 2022. Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control, Athena Scientific, Belmont, MA.

[Ber22b] Bertsekas, D. P., 2022. Abstract Dynamic Programming, 3rd Edition, Athena Scientific, Belmont, MA (can be downloaded from the author's website).

[Ber22c] Bertsekas, D. P., 2022. "Newton's Method for Reinforcement Learning and Model Predictive Control," Results in Control and Optimization, Vol. 7, pp. 100-121.

[Ber22d] Bertsekas, D. P., 2022. "Rollout Algorithms and Approximate Dynamic Programming for Bayesian Optimization and Sequential Estimation," arXiv:2212.07998.

[Ber24] Bertsekas, D. P., 2024. "Model Predictive Control, and Reinforcement Learning: A Unified Framework Based on Dynamic Programming," to be published in Proc. IFAC NMPC.

[Bet10] Bethke, B. M., 2010. Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination, Ph.D. Thesis, MIT.

[BiB24] Bishop, C. M, and Bishop, H., 2024. Deep Learning: Foundations and Concepts, Springer, New York, N. Y.

[BiL97] Birge, J. R., and Louveaux, 1997. Introduction to Stochastic Programming, Springer, New York, N. Y.

[Bia16] Bianchi, P., 2016. "Ergodic Convergence of a Stochastic Proximal Point Algorithm," SIAM J. on Optimization, Vol. 26, pp. 2235-2260.

[Bis95] Bishop, C. M, 1995. Neural Networks for Pattern Recognition, Oxford University Press, N. Y.

[Bis06] Bishop, C. M, 2006. Pattern Recognition and Machine Learning, Springer, N. Y.

[BlG54] Blackwell, D., and Girshick, M. A., 1954. Theory of Games and Statistical Decisions, Wiley, N. Y.

[BlM08] Blanchini, F., and Miani, S., 2008. Set-Theoretic Methods in Control, Birkhauser, Boston.

[Bla86] Blackman, S. S., 1986. Multi-Target Tracking with Radar Applications, Artech House, Dehdam, MA.

[Bla99] Blanchini, F., 1999. "Set Invariance in Control – A Survey," Automatica, Vol. 35, pp. 1747-1768.

[BoV79] Borkar, V. and Varaiya, P., 1979. "Adaptive Control of Markov Chains, I: Finite Parameter Set," IEEE Trans. on Automatic Control, Vol. 24, pp. 953-957.

[Bod20] Bodson, M., 2020. Adaptive Estimation and Control, Independently Published.

[Bor08] Borkar, V. S., 2008. Stochastic Approximation: A Dynamical Systems Viewpoint, Cambridge Univ. Press.

[BrH75] Bryson, A., and Ho, Y. C., 1975. Applied Optimal Control: Optimization, Estimation, and Control, (revised edition), Taylor and Francis, Levittown, Penn.

[Bra21] Brandimarte, P., 2021. From Shortest Paths to Reinforcement Learning: A MATLAB-Based Tutorial on Dynamic Programming, Springer.

[BuK97] Burnetas, A. N., and Katehakis, M. N., 1997. "Optimal Adaptive Policies for Markov Decision Processes," Math. of Operations Research, Vol. 22, pp. 222-255.

[CBH09] Choi, H. L., Brunet, L., and How, J. P., 2009. "Consensus-Based Decentralized Auctions for Robust Task Allocation," IEEE Transactions on Robotics, Vol. 25, pp. 912-926.

[CFH05] Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I., 2005. "An Adaptive Sampling Algorithm for Solving Markov Decision Processes," Operations Research, Vol. 53, pp. 126-139.

[CFH13] Chang, H. S., Hu, J., Fu, M. C., and Marcus, S. I., 2013. Simulation-Based Algorithms for Markov Decision Processes, 2nd Edition, Springer, N. Y.

[CLT19] Chapman, M. P., Lacotte, J., Tamar, A., Lee, D., Smith, K. M., Cheng, V., Fisac, J. F., Jha, S., Pavone, M., and Tomlin, C. J., 2019. "A Risk-Sensitive Finite-Time Reachability Approach for Safety of Stochastic Dynamic Systems," arXiv:1902.11277.

[CMT87a] Clarke, D. W., Mohtadi, C., and Tuffs, P. S., 1987. "Generalized Predictive Control - Part I. The Basic Algorithm," Automatica, Vol. 23, pp. 137-148.

[CMT87b] Clarke, D. W., Mohtadi, C., and Tuffs, P. S., 1987. "Generalized Predictive Control - Part II," Automatica, Vol. 23, pp. 149-160.

[CRV06] Cogill, R., Rotkowitz, M., Van Roy, B., and Lall, S., 2006. "An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems," in Control of Uncertain Systems: Modelling, Approximation, and Design, Springer, Berlin, pp. 243-256.

[CXL19] Chu, Z., Xu, Z., and Li, H., 2019. "New Heuristics for the RCPSP with Multiple Overlapping Modes," Computers and Industrial Engineering, Vol. 131, pp. 146-156.

[CaB07] Camacho, E. F., and Bordons, C., 2007. Model Predictive Control, 2nd Edition, Springer, New York, N. Y.

[CaC97] Cao, X. R., and Chen, H. F., 1997. "Perturbation Realization Potentials and Sensitivity Analysis of Markov Processes," IEEE Trans. on Aut. Control, Vol. 32, pp. 1382-1393.

[CaW98] Cao, X. R., and Wan, Y. W., 1998. "Algorithms for Sensitivity Analysis of Markov Systems Through Potentials and Perturbation Realization," IEEE Trans. Control Systems Technology, Vol. 6, pp. 482-494.

[Can16] Candy, J. V., 2016. Bayesian Signal Processing: Classical, Modern, and Particle Filtering Methods, Wiley-IEEE Press.

[Cao07] Cao, X. R., 2007. Stochastic Learning and Optimization: A Sensitivity-Based Approach, Springer, N. Y.

[ChC17] Chui, C. K., and Chen, G., 2017. Kalman Filtering, Springer International Publishing.

[Cha24] Chang, H. S., 2024. "On the Convergence Rate of MCTS for the Optimal Value Estimation in Markov Decision Processes," arXiv preprint arXiv:2402.07063.

[Che72] Chernoff, H., 1972. "Sequential Analysis and Optimal Design," Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, PA.

[ChM82] Chatelin, F., and Miranker, W. L., 1982. "Acceleration by Aggregation of Successive Approximation Methods," Linear Algebra and its Applications, Vol. 43, pp. 17-47.

[Chr97] Christodouleas, J. D., 1997. "Solution Methods for Multiprocessor Network Scheduling Problems with Application to Railroad Operations," Ph.D. Thesis, Operations Research Center, Massachusetts Institute of Technology.

[Cou06] Coulom, R., 2006. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," International Conference on Computers and Games, Springer, pp. 72-83.

[CrS00] Cristianini, N., and Shawe-Taylor, J., 2000. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge Univ. Press.

[Cyb89] Cybenko, 1989. "Approximation by Superpositions of a Sigmoidal Function," Math. of Control, Signals, and Systems, Vol. 2, pp. 303-314.

[DDF19] Daubechies, I., DeVore, R., Foucart, S., Hanin, B., and Petrova, G., 2019. "Nonlinear Approximation and (Deep) ReLU Networks," arXiv:1905.02199.

[DEK98] Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G., 1998. Biological Sequence Analysis, Cambridge Univ. Press, Cambridge.

[DFM12] Desai, V. V., Farias, V. F., and Moallemi, C. C., 2012. "Aproximate Dynamic Programming via a Smoothed Approximate Linear Program," Operations Research, Vol. 60, pp. 655-674.

[DFM13] Desai, V. V., Farias, V. F., and Moallemi, C. C., 2013. "Bounds for Markov Decision Processes," in Reinforcement Learning and Approximate Dynamic Program-

ming for Feedback Control, by F. Lewis and D. Liu (eds.), IEEE Press, Piscataway, N. J., pp. 452-473.

[DFV03] de Farias, D. P., and Van Roy, B., 2003. "The Linear Programming Approach to Approximate Dynamic Programming," Operations Research, Vol. 51, pp. 850-865.

[DFV04] de Farias, D. P., and Van Roy, B., 2004. "On Constraint Sampling in the Linear Programming Approach to Approximate Dynamic Programming," Mathematics of Operations Research, Vol. 29, pp. 462-478.

[DHS11] Duchi, J., Hazan, E., and Singer, Y., 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," J. of Machine Learning Research, Vol. 12, pp. 2121-2159.

[DHS12] Duda, R. O., Hart, P. E., and Stork, D. G., 2012. Pattern Classification, J. Wiley, N. Y.

[DJW12] Duchi, J., Jordan, M. I., Wainwright, M. J., and Wibisono, A., 2012. "Finite Sample Convergence Rate of Zero-Order Stochastic Optimization Methods," NIPS, pp. 1448-1456.

[DJW15] Duchi, J., Jordan, M. I., Wainwright, M. J., and Wibisono, A., 2015. "Optimal Rates for Zero-Order Convex Optimization: The Power of Two Function Evaluations," IEEE Trans. on Information Theory, Vol. 61, pp. 2788-2806.

[DNP11] Deisenroth, M. P., Neumann, G., and Peters, J., 2011. "A Survey on Policy Search for Robotics," Foundations and Trends in Robotics, Vol. 2, pp. 1-142.

[DNW16] David, O. E., Netanyahu, N. S., and Wolf, L., 2016. "Deepchess: End-to-End Deep Neural Network for Automatic Learning in Chess," in International Conference on Artificial Neural Networks, pp. 88-96.

[DeF04] De Farias, D. P., 2004. "The Linear Programming Approach to Approximate Dynamic Programming," in Learning and Approximate Dynamic Programming, by J. Si, A. Barto, W. Powell, and D. Wunsch, (Eds.), IEEE Press, N. Y.

[DeG70] DeGroot, M. H., 1970. Optimal Statistical Decisions, McGraw-Hill, N. Y.

[DeK11] Devlin, S., and Kudenko, D., 2011. "Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems," in Proceedings of AAMAS.

[Den67] Denardo, E. V., 1967. "Contraction Mappings in the Theory Underlying Dynamic Programming," SIAM Review, Vol. 9, pp. 165-177.

[DiL08] Dimitrakakis, C., and Lagoudakis, M. G., 2008. "Rollout Sampling Approximate Policy Iteration," Machine Learning, Vol. 72, pp. 157-171.

[DiM10] Di Castro, D., and Mannor, S., 2010. "Adaptive Bases for Reinforcement Learning," Machine Learning and Knowledge Discovery in Databases, Vol. 6321, pp. 312-327.

[DiW02] Dietterich, T. G., and Wang, X., 2002. "Batch Value Function Approximation via Support Vectors," in Advances in Neural Information Processing Systems, pp. 1491-1498.

[DoD93] Douglas, C. C., and Douglas, J., 1993. "A Unified Convergence Theory for Abstract Multigrid or Multilevel Algorithms, Serial and Parallel," SIAM J. Num. Anal., Vol. 30, pp. 136-158.

[DoJ09] Doucet, A., and Johansen, A. M., 2009. "A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later," Handbook of Nonlinear Filtering, Oxford University Press, Vol. 12, p. 3.

[DrH01] Drezner, Z., and Hamacher, H. W. eds., 2001. Facility Location: Applications and Theory, Springer Science and Business Media.

[Dre65] Dreyfus, S. D., 1965. Dynamic Programming and the Calculus of Variations, Academic Press, N. Y.

[DuV99] Duin, C., and Voss, S., 1999. "The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs," Networks: An International Journal, Vol. 34, pp. 181-191.

[EDS18] Efroni, Y., Dalal, G., Scherrer, B., and Mannor, S., 2018. "Beyond the One-Step Greedy Approach in Reinforcement Learning," in Proc. International Conf. on Machine Learning, pp. 1387-1396.

[ELP12] Estanjini, R. M., Li, K., and Paschalidis, I. C., 2012. "A Least Squares Temporal Difference Actor-Critic Algorithm with Applications to Warehouse Management," Naval Research Logistics, Vol. 59, pp. 197-211.

[EMM05] Engel, Y., Mannor, S., and Meir, R., 2005. "Reinforcement Learning with Gaussian Processes," in Proc. of the 22nd ICML, pp. 201-208.

[EPE20] Emami, P., Pardalos, P. M., Elefteriadou, L., and Ranka, S., 2020. "Machine Learning Methods for Data Association in Multi-Object Tracking," ACM Computing Surveys (CSUR), Vol. 53, pp. 1-34.

[Edd96] Eddy, S. R., 1996. "Hidden Markov Models," Current Opinion in Structural Biology, Vol. 6, pp. 361-365.

[EpM02] Ephraim, Y., and Merhav, N., 2002. "Hidden Markov Processes," IEEE Trans. on Information Theory, Vol. 48, pp. 1518-1569.

[FHS09] Feitzinger, F., Hylla, T., and Sachs, E. W., 2009. "Inexact Kleinman-Newton Method for Riccati Equations," SIAM Journal on Matrix Analysis and Applications, Vol. 3, pp. 272-288.

[FIA03] Findeisen, R., Imsland, L., Allgower, F., and Foss, B.A., 2003. "State and Output Feedback Nonlinear Model Predictive Control: An Overview," European Journal of Control, Vol. 9, pp. 190-206.

[FPB15] Farahmand, A. M., Precup, D., Barreto, A. M., and Ghavamzadeh, M., 2015. "Classification-Based Approximate Policy Iteration," IEEE Trans. on Automatic Control, Vol. 60, pp. 2989-2993.

[FeV02] Ferris, M. C., and Voelker, M. M., 2002. "Neuro-Dynamic Programming for Radiation Treatment Planning," Numerical Analysis Group Research Report NA-02/06, Oxford University Computing Laboratory, Oxford University.

[FeV04] Ferris, M. C., and Voelker, M. M., 2004. "Fractionation in Radiation Treatment Planning," Mathematical Programming B, Vol. 102, pp. 387-413.

[Fel60] Feldbaum, A. A., 1960. "Dual Control Theory," Automation and Remote Control, Vol. 21, pp. 874-1039.

[Fel63] Feldbaum, A. A., 1963. "Dual Control Theory Problems," IFAC Proceedings, pp. 541-550.

[FiT91] Filar, J. A., and Tolwinski, B., 1991. "On the Algorithm of Pollatschek and Avi-ltzhak," in Stochastic Games and Related Topics, Theory and Decision Library, Springer, Vol. 7, pp. 59-70.

[FiV96] Filar, J., and Vrieze, K., 1996. Competitive Markov Decision Processes, Springer.

[FoK09] Forrester, A. I., and Keane, A. J., 2009. "Recent Advances in Surrogate-Based Optimization. Progress in Aerospace Sciences," Vol. 45, pp. 50-79.

[For73] Forney, G. D., 1973. "The Viterbi Algorithm," Proc. IEEE, Vol. 61, pp. 268-278.

[Fra18] Frazier, P. I., 2018. "A Tutorial on Bayesian Optimization," arXiv:1807.02811.

[Fu17] Fu, M. C., 2017. "Markov Decision Processes, AlphaGo, and Monte Carlo Tree Search: Back to the Future," Leading Developments from INFORMS Communities, INFORMS, pp. 68-88.

[FuH94] Fu, M. C., and Hu, J.-Q., 1994. "Smoothed Perturbation Analysis Derivative Estimation for Markov Chains," Oper. Res. Letters, Vol. 41, pp. 241-251.

[Fun89] Funahashi, K., 1989. "On the Approximate Realization of Continuous Mappings by Neural Networks," Neural Networks, Vol. 2, pp. 183-192.

[GBB04] Greensmith, E., Bartlett, P. L., and Baxter, J., 2004. "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning," Journal of Machine Learning Research, Vol. 5, pp. 1471-1530.

[GBC16] Goodfellow, I., Bengio, J., and Courville, A., Deep Learning, MIT Press, Cambridge, MA.

[GBL12] Grondman, I., Busoniu, L., Lopes, G. A. D., and Babuska, R., 2012. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," IEEE Trans. on Systems, Man, and Cybernetics, Part C, Vol. 42, pp. 1291-1307.

[GPG22] Garces, D., Bhattacharya, S., Gil, G., and Bertsekas, D., "Multiagent Reinforcement Learning for Autonomous Routing and Pickup Problem with Adaptation to Variable Demand," arXiv preprint arXiv:2211.14983.

[GBL19] Goodson, J. C., Bertazzi, L., and Levary, R. R., 2019. "Robust Dynamic Media Selection with Yield Uncertainty: Max-Min Policies and Dual Bounds," Report.

[GDM19] Guerriero, F., Di Puglia Pugliese, L., and Macrina, G., 2019. "A Rollout Algorithm for the Resource Constrained Elementary Shortest Path Problem," Optimization Methods and Software, Vol. 34, pp. 1056-1074.

[GGS13] Gabillon, V., Ghavamzadeh, M., and Scherrer, B., 2013. "Approximate Dynamic Programming Finally Performs Well in the Game of Tetris," in NIPS, pp. 1754-1762.

[GGW11] Gittins, J., Glazebrook, K., and Weber, R., 2011. Multi-Armed Bandit Allocation Indices, J. Wiley, N. Y.

[GHC21] Gerlach, T., Hoffmann, F., and Charlish, A., 2021. "Policy Rollout Action Selection with Knowledge Gradient for Sensor Path Planning," 2021 IEEE 24th International Conference on Information Fusion, pp. 1-8.

[GLG11] Gabillon, V., Lazaric, A., Ghavamzadeh, M., and Scherrer, B., 2011. "Classification-Based Policy Iteration with a Critic," in Proc. of ICML.

[GMP15] Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A., 2015. "Bayesian Reinforcement Learning: A Survey," Foundations and Trends in Machine Learning, Vol. 8, pp. 359-483.

[GSD06] Goodwin, G., Seron, M. M., and De Dona, J. A., 2006. Constrained Control and Estimation: An Optimisation Approach, Springer, N. Y.

[GSS93] Gordon, N. J., Salmond, D. J., and Smith, A. F., 1993. "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation," in IEE Proceedings, Vol. 140, pp. 107-113.

[GTA17] Gommans, T. M. P., Theunisse, T. A. F., Antunes, D. J., and Heemels, W. P. M. H., 2017. "Resource-Aware MPC for Constrained Linear Systems: Two Rollout Approaches," Journal of Process Control, Vol. 51, pp. 68-83.

[GTO15] Goodson, J. C., Thomas, B. W., and Ohlmann, J. W., 2015. "Restocking-Based Rollout Policies for the Vehicle Routing Problem with Stochastic Demand and Duration Limits," Transportation Science, Vol. 50, pp. 591-607.

[GTO17] Goodson, J. C., Thomas, B. W., and Ohlmann, J. W., 2017. "A Rollout Algorithm Framework for Heuristic Solutions to Finite-Horizon Stochastic Dynamic Programs," European Journal of Operational Research, Vol. 258, pp. 216-229.

[GeB13] Geffner, H., and Bonet, B., 2013. A Concise Introduction to Models and Methods for Automated Planning, Morgan and Claypool Publishers.

[Gly87] Glynn, P. W., 1987. "Likelihood Ratio Gradient Estimation: An Overview," Proc. of the 1987 Winter Simulation Conference, pp. 366-375.

[Gly90] Glynn, P. W., 1990. "Likelihood Ratio Gradient Estimation for Stochastic Systems," Communications of the ACM, Vol. 33, pp. 75-84.

[GoS84] Goodwin, G. C., and Sin, K. S. S., 1984. Adaptive Filtering, Prediction, and Control, Prentice-Hall, Englewood Cliffs, N. J.

[Gor95] Gordon, G. J., 1995. "Stable Function Approximation in Dynamic Programming," in Machine Learning: Proceedings of the Twelfth International Conference, Morgan Kaufmann, San Francisco, CA.

[Gos15] Gosavi, A., 2015. Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, 2nd Edition, Springer, N. Y.

[Grz17] Grzes, M., 2017. "Reward Shaping in Episodic Reinforcement Learning," in Proc. of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 565-573.

[GuM01] Guerriero, F., and Musmanno, R., 2001. "Label Correcting Methods to Solve Multicriteria Shortest Path Problems," J. Optimization Theory Appl., Vol. 111, pp. 589-613.

[GuM03] Guerriero, F., and Mancini, M., 2003. "A Cooperative Parallel Rollout Algorithm for the Sequential Ordering Problem," Parallel Computing, Vol. 29, pp. 663-677.

[Gup20] Gupta, A., 2020. "Existence of Team-Optimal Solutions in Static Teams with Common Information: A Topology of Information Approach," SIAM J. on Control and Optimization, Vol. 58, pp.998-1021.

[HCR21] Hoffmann, F., Charlish, A., Ritchie, M., and Griffiths, H., 2021. "Policy Rollout Action Selection in Continuous Domains for Sensor Path Planning," IEEE Trans. on Aerospace and Electronic Systems.

[HJG16] Huang, Q., Jia, Q. S., and Guan, X., 2016. "Robust Scheduling of EV Charging Load with Uncertain Wind Power Integration," IEEE Trans. on Smart Grid, Vol. 9, pp. 1043-1054.

[HLS06] Han, J., Lai, T. L. and Spivakovsky, V., 2006. "Approximate Policy Optimization and Adaptive Control in Regression Models," Computational Economics, Vol. 27, pp. 433-452.

[HMR19] Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J., 2019. "Surprises in High-Dimensional Ridgeless Least Squares Interpolation," arXiv:1903.08560.

[HLZ19] Ho, T. Y., Liu, S., and Zabinsky, Z. B., 2019. "A Multi-Fidelity Rollout Algorithm for Dynamic Resource Allocation in Population Disease Management," Health

Care Management Science, Vol. 22, pp. 727-755.

[HSS08] Hofmann, T., Scholkopf, B., and Smola, A. J., 2008. "Kernel Methods in Machine Learning," The Annals of Statistics, Vol. 36, pp. 1171-1220.

[HSW89] Hornick, K., Stinchcombe, M., and White, H., 1989. "Multilayer Feedforward Networks are Universal Approximators," Neural Networks, Vol. 2, pp. 359-159.

[HWM19] Hewing, L., Wabersich, K. P., Menner, M., and Zeilinger, M. N., 2019. "Learning-Based Model Predictive Control: Toward Safe Learning in Control," Annual Review of Control, Robotics, and Autonomous Systems.

[HWP22] Hu, J., Wang, Y., Pang, Y., and Liu, Y., 2022. "Optimal Maintenance Scheduling under Uncertainties using Linear Programming-Enhanced Reinforcement Learning," Engineering Applications of Artificial Intelligence, Vol. 109.

[HaR21] Hardt, M., and Recht, B., 2021. Patterns, Predictions, and Actions: A Story About Machine Learning, arXiv:2102.05242; published by Priceton Univ. Press, 2022.

[Han98] Hansen, E. A., 1998. "Solving POMDPs by Searching in Policy Space," in Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence, pp. 211-219.

[Hay08] Haykin, S., 2008. Neural Networks and Learning Machines, 3rd Edition, Prentice-Hall, Englewood-Cliffs, N. J.

[HeZ19] Hewing, L., and Zeilinger, M. N., 2019. "Scenario-Based Probabilistic Reachable Sets for Recursively Feasible Stochastic Model Predictive Control," IEEE Control Systems Letters, Vol. 4, pp. 450-455.

[Hew71] Hewer, G., 1971. "An Iterative Technique for the Computation of the Steady State Gains for the Discrete Optimal Regulator," IEEE Trans. on Automatic Control, Vol. 16, pp. 382-384.

[Ho80] Ho, Y. C., 1980. "Team Decision Theory and Information Structures," Proceedings of the IEEE, Vol. 68, pp. 644-654.

[HuM16] Huan, X., and Marzouk, Y. M., 2016. "Sequential Bayesian Optimal Experimental Design via Approximate Dynamic Programming," arXiv:1604.08320.

[Hua15] Huan, X., 2015. Numerical Approaches for Sequential Bayesian Optimal Experimental Design, Ph.D. Thesis, MIT.

[Hyl11] Hylla, T., 2011. Extension of Inexact Kleinman-Newton Methods to a General Monotonicity Preserving Convergence Theory, PhD Thesis, Univ. of Trier.

[IFT19] Issakkimuthu, M., Fern, A., and Tadepalli, P., 2019. "The Choice Function Framework for Online Policy Improvement," arXiv:1910.00614.

[IJT18] Iusem, A., Jofre, A., and Thompson, P., 2018. "Incremental Constraint Projection Methods for Monotone Stochastic Variational Inequalities," Math. of Operations Research, Vol. 44, pp. 236-263.

[IoS96] Ioannou, P. A., and Sun, J., 1996. Robust Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[JCG20] Jiang, S., Chai, H., Gonzalez, J., and Garnett, R., 2020. "BINOCULARS for Efficient, Nonmyopic Sequential Experimental Design," in Proc. Intern. Conference on Machine Learning, pp. 4794-4803.

[JGJ18] Jones, M., Goldstein, M., Jonathan, P., and Randell, D., 2018. "Bayes Linear Analysis of Risks in Sequential Optimal Design Problems," Electronic Journal of Statistics, Vol. 12, pp. 4002-4031.

[JJB20] Jiang, S., Jiang, D. R., Balandat, M., Karrer, B., Gardner, J. R., and Garnett, R., 2020. "Efficient Nonmyopic Bayesian Optimization via One-Shot Multi-Step Trees," arXiv preprint arXiv:2006.15779.

[JSJ95] Jaakkola, T., Singh, S. P., and Jordan, M. I., 1995. "Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems," NIPS, Vol. 7, pp. 345-352.

[JSW98] Jones, D. R., Schonlau, M., and Welch, W. J., 1998. "Efficient Global Optimization of Expensive Black-Box Functions," J. of Global Optimization, Vol. 13, pp. 455?492.

[JiJ17] Jiang, Y., and Jiang, Z. P., 2017. Robust Adaptive Dynamic Programming, J. Wiley, N. Y.

[JoB16] Joseph, A. G., and Bhatnagar, S., 2016. "Revisiting the Cross Entropy Method with Applications in Stochastic Global Optimization and Reinforcement Learning," in Proc. of the 22nd European Conference on Artificial Intelligence, pp. 1026-1034.

[JoB18] Joseph, A. G., and Bhatnagar, S., 2018. "A Cross Entropy Based Optimization Algorithm with Global Convergence Guarantees," arXiv preprint arXiv:1801.10291.

[Jon90] Jones, L. K., 1990. "Constructive Approximations for Neural Networks by Sigmoidal Functions," Proceedings of the IEEE, Vol. 78, pp. 1586-1589.

[JuM23] Jurafsky, D., and Martin, J. H., 2023. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, draft 3rd edition (on-line).

[JuP07] Jung, T., and Polani, D., 2007. "Kernelizing LSPE(λ)," Proc. 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, Honolulu, Ha., pp. 338-345.

[KAC15] Kochenderfer, M. J., with Amato, C., Chowdhary, G., How, J. P., Davison Reynolds, H. J., Thornton, J. R., Torres-Carrasquillo, P. A., Ore, N. K., Vian, J., 2015. Decision Making under Uncertainty: Theory and Application, MIT Press, Cambridge, MA.

[KAH15] Khashooei, B. A., Antunes, D. J. and Heemels, W.P.M.H., 2015. "Rollout Strategies for Output-Based Event-Triggered Control," in Proc. 2015 European Control Conference, pp. 2168-2173.

[KGB82] Kimemia, J., Gershwin, S. B., and Bertsekas, D. P., 1982. "Computation of Production Control Policies by a Dynamic Programming Technique," in Analysis and Optimization of Systems, A. Bensoussan and J. L. Lions (eds.), Springer, N. Y., pp. 243-269.

[KKK95] Krstic, M., Kanellakopoulos, I., Kokotovic, P., 1995. Nonlinear and Adaptive Control Design, J. Wiley, N. Y.

[KLC98] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R., 1998. "Planning and Acting in Partially Observable Stochastic Domains," Artificial Intelligence, Vol. 101, pp. 99-134.

[KLM82a] Krainak, J. L. S. J. C., Speyer, J., and Marcus, S., 1982. "Static Team Problems - Part I: Sufficient Conditions and the Exponential Cost Criterion," IEEE Transactions on Automatic Control, Vol. 27, pp. 839-848.

[KLM82b] Krainak, J. L. S. J. C., Speyer, J., and Marcus, S., 1982. "Static Team Problems - Part II: Affine Control Laws, Projections, Algorithms, and the LEGT Problem," IEEE Transactions on Automatic Control, Vol. 27, pp. 848-859.

[KLM96] Kaelbling, L. P., Littman, M. L., and Moore, A. W., 1996. "Reinforcement Learning: A Survey," J. of Artificial Intelligence Res., Vol. 4, pp. 237-285.

[KMP06] Keller, P. W., Mannor, S., and Precup, D., 2006. "Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning," Proc. of the 23rd ICML, Pittsburgh, Penn.

[KRC13] Kroese, D. P., Rubinstein, R. Y., Cohen, I., Porotsky, S., and Taimre, T., 2013. "Cross-Entropy Method," in Encyclopedia of Operations Research and Management Science, Springer, Boston, MA, pp. 326-333.

[KaW94] Kall, P., and Wallace, S. W., 1994. Stochastic Programming, Wiley, Chichester, UK.

[Kak02] Kakade, S. A., 2002. "Natural Policy Gradient," NIPS, Vol. 14, pp. 1531-1538.

[KeG88] Keerthi, S. S., and Gilbert, E. G., 1988. "Optimal, Infinite Horizon Feedback Laws for a General Class of Constrained Discrete Time Systems: Stability and Moving-Horizon Approximations," J. Optimization Theory Appl., Vo. 57, pp. 265-293.

[KiB14] Kingma, D. P., and Ba, J., 2014. "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980.

[Kir04] Kirk, D. E., 2004. Optimal Control Theory: An Introduction, Courier Corporation.

[Kim82] Kimemia, J., 1982. "Hierarchical Control of Production in Flexible Manufacturing Systems," Ph.D. Thesis, Dep. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[Kle09] Kleijnen, J. P., 2009. "Kriging Metamodeling in Simulation: A Review," European Journal of Operational Research, Vol. 192, pp. 707-716.

[Kle68] Kleinman, D. L., 1968. "On an Iterative Technique for Riccati Equation Computations," IEEE Trans. Aut. Control, Vol. AC-13, pp. 114-115.

[KoC16] Kouvaritakis, B., and Cannon, M., 2016. Model Predictive Control: Classical, Robust and Stochastic, Springer, N. Y.

[KoG98] Kolmanovsky, I.. and Gilbert, E. G., 1998. "Theory and Computation of Disturbance Invariant Sets for Discrete-Time Linear Systems," Math. Problems in Engineering, Vol. 4, pp. 317-367.

[KoS06] Kocsis, L., and Szepesvari, C., 2006. "Bandit Based Monte-Carlo Planning," Proc. of 17th European Conference on Machine Learning, Berlin, pp. 282-293.

[KoT99] Konda, V. R., and Tsitsiklis, J. N., 1999. "Actor-Critic Algorithms," NIPS, Denver, Colorado, pp. 1008-1014.

[KoT03] Konda, V. R., and Tsitsiklis, J. N., 2003. "Actor-Critic Algorithms," SIAM J. on Control and Optimization, Vol. 42, pp. 1143-1166.

[Kre19] Krener, A. J., 2019. "Adaptive Horizon Model Predictive Control and Al'brekht's Method," arXiv:1904.00053.

[Kri16] Krishnamurthy, V., 2016. Partially Observed Markov Decision Processes, Cambridge Univ. Press.

[KuV86] Kumar, P. R., and Varaiya, P. P., 1986. Stochastic Systems: Estimation, Identification, and Adaptive Control, Prentice-Hall, Englewood Cliffs, N. J.

[Kun14] Kung, S. Y., 2014. Kernel Methods and Machine Learning, Cambridge Univ. Press.

[L'Ec91] L'Ecuyer, P., 1991. "An Overview of Derivative Estimation," Proceedings of the 1991 Winter Simulation Conference, pp. 207-217.

[LEC20] Lee, E. H., Eriksson, D., Cheng, B., McCourt, M., and Bindel, D., 2020. "Efficient Rollout Strategies for Bayesian Optimization," arXiv:2002.10539.

[LEP21] Lee, E. H., Eriksson, D., Perrone, V., and Seeger, M., 2021. "A Nonmyopic Approach to Cost-Constrained Bayesian Optimization," In Uncertainty in Artificial Intelligence Proceedings, pp. 568-577.

[LGM10] Lazaric, A., Ghavamzadeh, M., and Munos, R., 2010. "Analysis of a Classification-Based Policy Iteration Algorithm," INRIA Report.

[LGW16] Lan, Y., Guan, X., and Wu, J., 2016. "Rollout Strategies for Real-Time Multi-Energy Scheduling in Microgrid with Storage System," IET Generation, Transmission and Distribution, Vol. 10, pp. 688-696.

[LJM19] Li, Y., Johansson, K. H., and Martensson, J., 2019. "Lambda-Policy Iteration with Randomization for Contractive Models with Infinite Policies: Well Posedness and Convergence," arXiv:1912.08504.

[LJM21] Li, Y., Johansson, K. H., Martensson, J., and Bertsekas, D. P., 2021. "Data-Driven Rollout for Deterministic Optimal Control," arXiv preprint arXiv:2105.03116.

[LKG21] Li, T., Krakow, L. W., and Gopalswamy, S., 2021. "Optimizing Consensus-Based Multi-Target Tracking with Multiagent Rollout Control Policies," In 2021 IEEE Conference on Control Technology and Applications, pp. 131-137.

[LLL19] Liu, Z., Lu, J., Liu, Z., Liao, G., Zhang, H. H., and Dong, J., 2019. "Patient Scheduling in Hemodialysis Service," J. of Combinatorial Optimization, Vol. 37, pp. 337-362.

[LLP93] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S., 1993. "Multilayer Feedforward Networks with a Nonpolynomial Activation Function can Approximate any Function," Neural Networks, Vol. 6, pp. 861-867.

[LPS22] Liu, M., Pedrielli, G., Sulc, P., Poppleton, E., Bertsekas, D. P., 2022. "ExpertRNA: A New Framework for RNA Structure Prediction," INFORMS Journal on Computing.

[LRD23] Laidlaw, C., Russell, S., and Dragan, A., 2023. "Bridging RL Theory and Practice with the Effective Horizon," arXiv preprint arXiv:2304.09853.

[LTZ19] Li, Y., Tang, Y., Zhang, R., and Li, N., 2019. "Distributed Reinforcement Learning for Decentralized Linear Quadratic Control: A Derivative-Free Policy Optimization Approach," arXiv:1912.09135.

[LWT17] Lowe, L., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I., 2017. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in Advances in Neural Information Processing Systems, pp. 6379-6390.

[LWW16] Lam, R., Willcox, K., and Wolpert, D. H., 2016. "Bayesian Optimization with a Finite Budget: An Approximate Dynamic Programming Approach," In Advances in Neural Information Processing Systems, pp. 883-891.

[LWW17] Liu, D., Wei, Q., Wang, D., Yang, X., and Li, H., 2017. Adaptive Dynamic Programming with Applications in Optimal Control, Springer, Berlin.

[LZS20] Li, H., Zhang, X., Sun, J., and Dong, X., 2020. "Dynamic Resource Levelling in Projects under Uncertainty," International J. of Production Research.

[LaP03] Lagoudakis, M. G., and Parr, R., 2003. "Reinforcement Learning as Classification: Leveraging Modern Classifiers," in Proc. of ICML, pp. 424-431.

[LaR85] Lai, T., and Robbins, H., 1985. "Asymptotically Efficient Adaptive Allocation Rules," Advances in Applied Math., Vol. 6, pp. 4-22.

[LaS20] Lattimore, T., and Szepesvari, C., 2020. Bandit Algorithms, Cambridge University Press.

[LaW13] Lavretsky, E., and Wise, K., 2013. Robust and Adaptive Control with Aerospace Applications, Springer.

[LaW17] Lam, R., and Willcox, K., 2017. "Lookahead Bayesian Optimization with Inequality Constraints," in Advances in Neural Information Processing Systems, pp. 1890-1900.

[Lee20] Lee, E. H., 2020. "Budget-Constrained Bayesian Optimization, Doctoral dissertation, Cornell University.

[LiB24], Li, Y., and Bertsekas, D. P., 2024. "Most Likely Sequence Generation for n-Grams, Transformers, HMMs, and Markov Chains, by Using Rollout Algorithms," arXiv:2403.15465.

[LiS16] Liang, S., and Srikant, R., 2016. "Why Deep Neural Networks for Function Approximation?" arXiv:1610.04161.

[LiW14] Liu, D., and Wei, Q., 2014. "Policy Iteration Adaptive Dynamic Programming Algorithm for Discrete-Time Nonlinear Systems," IEEE Trans. on Neural Networks and Learning Systems, Vol. 25, pp. 621-634.

[LiW15] Li, H., and Womer, N. K., 2015. "Solving Stochastic Resource-Constrained Project Scheduling Problems by Closed-Loop Approximate Dynamic Programming," European J. of Operational Research, Vol. 246, pp. 20-33.

[Lib11] Liberzon, D., 2011. Calculus of Variations and Optimal Control Theory: A Concise Introduction, Princeton Univ. Press.

[LoC23] Loxley, P. N., and Cheung, K. W., 2023. "A Dynamic Programming Algorithm for Finding an Optimal Sequence of Informative Measurements," Entropy, Vol. 25, p. 251.

[MCT10] Mishra, N., Choudhary, A. K., Tiwari, M. K., and Shankar, R., 2010. "Rollout Strategy-Based Probabilistic Causal Model Approach for the Multiple Fault Diagnosis," Robotics and Computer-Integrated Manufacturing, Vol. 26, pp. 325-332.

[MDM01] Magni, L., De Nicolao, G., Magnani, L., and Scattolini, R., 2001. "A Stabilizing Model-Based Predictive Control Algorithm for Nonlinear Systems," Automatica, Vol. 37, pp. 1351-1362.

[MKS15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and Petersen, S., 2015. "Human-Level Control Through Deep Reinforcement Learning," Nature, Vol. 518, p. 529.

[MLM20] Montenegro, M., Lopez, R., Menchaca-Mendez, R., Becerra, E., and Menchaca-Mendez, R., 2020. "A Parallel Rollout Algorithm for Wildfire Suppression," in Proc. Intern. Congress of Telematics and Computing, pp. 244-255.

[MMB02] McGovern, A., Moss, E., and Barto, A., 2002. "Building a Basic Building Block Scheduler Using Reinforcement Learning and Rollouts," Machine Learning, Vol. 49, pp. 141-160.

[MMS05] Menache, I., Mannor, S., and Shimkin, N., 2005. "Basis Function Adaptation in Temporal Difference Reinforcement Learning," Ann. Oper. Res., Vol. 134, pp. 215-238.

[MPK99] Meuleau, N., Peshkin, L., Kim, K. E., and Kaelbling, L. P., 1999. "Learning Finite-State Controllers for Partially Observable Environments," in Proc. of the 15th Conference on Uncertainty in Artificial Intelligence, pp. 427-436.

[MPP04] Meloni, C., Pacciarelli, D., and Pranzo, M., 2004. "A Rollout Metaheuristic for Job Shop Scheduling Problems," Annals of Operations Research, Vol. 131, pp. 215-235.

[MRG03] Mannor, S., Rubinstein, R. Y., and Gat, Y., 2003. "The Cross Entropy Method for Fast Policy Search," in Proc. of the 20th International Conference on Machine Learning (ICML-03), pp. 512-519.

[MRR00] Mayne, D., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M., 2000. "Constrained Model Predictive Control: Stability and Optimality," Automatica, Vol. 36, pp. 789-814.

[MVS19] Muthukumar, V., Vodrahalli, K., and Sahai, A., 2019. "Harmless Interpolation of Noisy Data in Regression," arXiv:1903.09139.

[MYF03] Moriyama, H., Yamashita, N., and Fukushima, M., 2003. "The Incremental Gauss-Newton Algorithm with Adaptive Stepsize Rule," Computational Optimization and Applications, Vol. 26, pp. 107-141.

[MaE07] Mamon, R. S., and Elliott, R. J. eds., 2007. Hidden Markov Models in Finance, Springer, NY.

[MaE14] Mamon, R. S., and Elliott, R. J. eds., 2007. Hidden Markov Models in Finance: Further Developments and Applications, Springer, NY.

[MaJ15] Mastin, A., and Jaillet, P., 2015. "Average-Case Performance of Rollout Algorithms for Knapsack Problems," J. of Optimization Theory and Applications, Vol. 165, pp. 964-984.

[MaS99] Manning, C., and Schutze, H., 1999. Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA.

[MaS02] Martinez, L., and Soares, S., 2002. "Comparison Between Closed-Loop and Partial Open-Loop Feedback Control Policies in Long Term Hydrothermal Scheduling," IEEE Transactions on Power Systems, Vol. 17, pp. 330-336.

[MaT01] Marbach, P., and Tsitsiklis, J. N., 2001. "Simulation-Based Optimization of Markov Reward Processes," IEEE Trans. on Aut. Control, Vol. 46, pp. 191-209.

[MaT03] Marbach, P., and Tsitsiklis, J. N., 2003. "Approximate Gradient Methods in Policy-Space Optimization of Markov Reward Processes," J. Discrete Event Dynamic Systems, Vol. 13, pp. 111-148.

[Mac02] Maciejowski, J. M., 2002. Predictive Control with Constraints, Addison-Wesley, Reading, MA.

[Mar55] Marschak, J., 1955. "Elements for a Theory of Teams," Management Science, Vol. 1, pp. 127-137.

[Mar84] Martins, E. Q. V., 1984. "On a Multicriteria Shortest Path Problem," European J. of Operational Research, Vol. 16, pp. 236-245.

[Mat65] J. Matyas, J., 1965. "Random Optimization," Automation and Remote Control, Vol. 26, pp. 246-253.

[May14] Mayne, D. Q., 2014. "Model Predictive Control: Recent Developments and Future Promise," Automatica, Vol. 50, pp. 2967-2986.

[MeB99] Meuleau, N., and Bourgine, P., 1999. "Exploration of Multi-State Environments: Local Measures and Back-Propagation of Uncertainty," Machine Learning, Vol. 35, pp. 117-154.

[MeK20] Meshram, R.. and Kaza, K., 2020. "Simulation Based Algorithms for Markov Decision Processes and Multi-Action Restless Bandits," arXiv:2007.12933.

[Mey07] Meyn, S., 2007. Control Techniques for Complex Networks, Cambridge Univ. Press, N. Y.

[Mey22] Meyn, S., 2022. Control Systems and Reinforcement Learning, Cambridge Univ. Press, N. Y.

[Min22] Minorsky, N., 1922. "Directional Stability of Automatically Steered Bodies," J. Amer. Soc. Naval Eng.,Vol. 34, pp. 280-309.

[MoL99] Morari, M., and Lee, J. H., 1999. "Model Predictive Control: Past, Present, and Future," Computers and Chemical Engineering, Vol. 23, pp. 667-682.

[Mon17] Montgomery, D. C., 2017. Design and Analysis of Experiments, J. Wiley.

[Mun14] Munos, R., 2014. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning," Foundations and Trends in Machine Learning, Vol. 7, pp. 1-129.

[NHR99] Ng, A. Y., Harada, D., and Russell, S. J., 1999. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping," in Proc. of the 16th International Conference on Machine Learning, pp. 278-287.

[NMT13] Nayyar, A., Mahajan, A. and Teneketzis, D., 2013. "Decentralized Stochastic Control with Partial History Sharing: A Common Information Approach," IEEE Transactions on Automatic Control, Vol. 58, pp. 1644-1658.

[NSE19] Nozhati, S., Sarkale, Y., Ellingwood, B., Chong, E. K., and Mahmoud, H., 2019. "Near-Optimal Planning Using Approximate Dynamic Programming to Enhance Post-Hazard Community Resilience Management," Reliability Engineering and System Safety, Vol. 181, pp. 116-126.

[NaA12] Narendra, K. S., and Annaswamy, A. M., 2012. Stable Adaptive Systems, Courier Corporation.

[NaT19] Nayyar, A., and Teneketzis, D., 2019. "Common Knowledge and Sequential Team Problems," IEEE Trans. on Automatic Control, Vol. 64, pp. 5108-5115.

[NeS17] Nesterov, Y., and Spokoiny, V., 2017. "Random Gradient-Free Minimization of Convex Functions," Foundations of Computational Mathematics, Vol. 17, pp. 527-566.

[Ned11] Nedić, A., 2011. "Random Algorithms for Convex Minimization Problems," Math. Programming, Ser. B, Vol. 129, pp. 225-253.

[NoS09] Novoa, C. and Storer, R., 2009. "An Approximate Dynamic Programming Approach for the Vehicle Routing Problem with Stochastic Demands," European J. of Operational Research, Vol. 196, pp. 509-515.

[OrS02] Ormoneit, D., and Sen, S., 2002. "Kernel-Based Reinforcement Learning," Machine Learning, Vol. 49, pp. 161-178.

[PDB92] Pattipati, K. R., Deb, S., Bar-Shalom, Y., and Washburn, R. B., 1992. "A New Relaxation Algorithm and Passive Sensor Data Association," IEEE Trans. Automatic Control, Vol. 37, pp. 198-213.

[PDC14] Pillonetto, G., Dinuzzo, F., Chen, T., De Nicolao, G., and Ljung, L., 2014. "Kernel Methods in System Identification, Machine Learning and Function Estimation: A Survey," Automatica, Vol. 50, pp. 657-682.

[PPB01] Popp, R. L., Pattipati, K. R., and Bar-Shalom, Y., 2001. "m-Best SD Assignment Algorithm with Application to Multitarget Tracking," IEEE Transactions on Aerospace and Electronic Systems, Vol. 37, pp. 22-39.

[PSC22] Paulson, J. A., Sonouifar, F., and Chakrabarty, A., 2022. "Efficient Multi-Step Lookahead Bayesian Optimization with Local Search Constraints," IEEE Conf. on Decision and Control, pp. 123-129.

[PPG16] Perolat, J., Piot, B., Geist, M., Scherrer, B., and Pietquin, O., 2016. "Softened Approximate Policy Iteration for Markov Games," in Proc. International Conference on Machine Learning, pp. 1860-1868.

[PSP15] Perolat, J., Scherrer, B., Piot, B., and Pietquin, O., 2015. "Approximate Dynamic Programming for Two-Player Zero-Sum Markov Games," in Proc. International Conference on Machine Learning, pp. 1321-1329.

[PaB99] Patek, S. D., and Bertsekas, D. P., 1999. "Stochastic Shortest Path Games," SIAM J. on Control and Optimization, Vol. 37, pp. 804-824.

[PaR12] Papahristou, N., and Refanidis, I., 2012. "On the Design and Training of Bots to Play Backgammon Variants," in IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 78-87.

[PaT00] Paschalidis, I. C., and Tsitsiklis, J. N., 2000. "Congestion-Dependent Pricing of Network Services," IEEE/ACM Trans. on Networking, Vol. 8, pp. 171-184.

[PeG04] Peret, L., and Garcia, F., 2004. "On-Line Search for Solving Markov Decision Processes via Heuristic Sampling," in Proc. of the 16th European Conference on Artificial Intelligence, pp. 530-534.

[PeS08] Peters, J., and Schaal, S., 2008. "Reinforcement Learning of Motor Skills with Policy Gradients," Neural Networks, Vol. 4, pp. 682-697.

[PeW96] Peng, J., and Williams, R., 1996. "Incremental Multi-Step Q-Learning," Machine Learning, Vol. 22, pp. 283-290.

[PoA69] Pollatschek, M. A. and Avi-Itzhak, B., 1969. "Algorithms for Stochastic Games with Geometrical Interpretation," Management Science, Vol. 15, pp. 399-415.

[PoB04] Poupart, P., and Boutilier, C., 2004. "Bounded Finite State Controllers," in Advances in Neural Information Processing Systems, pp. 823-830.

[PoF08] Powell, W. B. and Frazier, P., 2008. "Optimal Learning," in State-of-the-Art Decision-Making Tools in the Information-Intensive Age, INFORMS, pp. 213-246.

[PoR97] Poore, A. B., and Robertson, A. J. A., 1997. "New Lagrangian Relaxation Based Algorithm for a Class of Multidimensional Assignment Problems," Computational Optimization and Applications, Vol. 8, pp. 129-150.

[PoR12] Powell, W. B., and Ryzhov, I. O., 2012. Optimal Learning, J. Wiley, N. Y.

[Poo94] Poore, A. B., 1994. "Multidimensional Assignment Formulation of Data Association Problems Arising from Multitarget Tracking and Multisensor Data Fusion," Computational Optimization and Applications, Vol. 3, pp. 27-57.

[Pow11] Powell, W. B., 2011. Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition, J. Wiley and Sons, Hoboken, N. J.

[PrS01] Proakis, J. G., and Salehi, M., 2001. Communication Systems Engineering, Prentice-Hall, Englewood Cliffs, N. J.

[PrS08] Proakis, J. G., and Salehi, M., 2008. Digital Communications, McGraw-Hill, N. Y.

[Pre95] Prekopa, A., 1995. Stochastic Programming, Kluwer, Boston.

[PuB78] Puterman, M. L., and Brumelle, S. L., 1978. "The Analytic Theory of Policy Iteration," in Dynamic Programming and Its Applications, M. L. Puterman (ed.), Academic Press, N. Y.

[PuB79] Puterman, M. L., and Brumelle, S. L., 1979. "On the Convergence of Policy Iteration in Stationary Dynamic Programming," Mathematics of Operations Research, Vol. 4, pp. 60-69.

[PuS78] Puterman, M. L., and Shin, M. C., 1978. "Modified Policy Iteration Algorithms for Discounted Markov Decision Problems," Management Sci., Vol. 24, pp. 1127-1137.

[PuS82] Puterman, M. L., and Shin, M. C., 1982. "Action Elimination Procedures for Modified Policy Iteration Algorithms," Operations Research, Vol. 30, pp. 301-318.

[Put94] Puterman, M. L., 1994. Markovian Decision Problems, J. Wiley, N. Y.

[QHS05] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., and Tucker, P. K., 2005. "Surrogate-Based Analysis and Optimization," Progress in Aerospace Sciences, Vol. 41, pp. 1-28.

[QuL19] Qu, G., and Li, N., "Exploiting Fast Decaying and Locality in Multi-Agent MDP with Tree Dependence Structure," Proc. of 2019 CDC, Nice, France.

[RCR17] Rudi, A., Carratino, L., and Rosasco, L., 2017. "Falkon: An Optimal Large Scale Kernel Method," in Advances in Neural Information Processing Systems, pp. 3888-3898.

[RDM24] Ruoss, A., Delétang, G., Medapati, S., Grau-Moya, J., Wenliang, L. K., Catt, E., Reid, J., and Genewein, T., 2024. "Grandmaster-Level Chess Without Search," arXiv:2402.04494.

[RGG21] Rimélé, A., Grangier, P., Gamache, M., Gendreau, M., and Rousseau, L. M., 2021. "E-Commerce Warehousing: Learning a Storage Policy, arXiv:2101.08828.

[RMD17] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M., 2017. Model Predictive Control: Theory, Computation, and Design, 2nd Ed., Nob Hill Publishing.

[RPF12] Ryzhov, I. O., Powell, W. B., and Frazier, P. I., 2012. "The Knowledge Gradient Algorithm for a General Class of Online Learning Problems," Operations Research, Vol. 60, pp. 180-195.

[RPW91] Rogers, D. F., Plante, R. D., Wong, R. T., and Evans, J. R., 1991. "Aggregation and Disaggregation Techniques and Methodology in Optimization," Operations Research, Vol. 39, pp. 553-582.

[RSM08] Reisinger, J., Stone, P., and Miikkulainen, R., 2008. "Online Kernel Selection for Bayesian Reinforcement Learning," in Proc. of the 25th International Conference on Machine Learning, pp. 816-823.

[RST23] Rusmevichientong, P., Sumida, M., Topaloglu, H., and Bai, Y., 2023. "Revenue Management with Heterogeneous Resources: Unit Resource Capacities, Advance Bookings, and Itineraries over Time Intervals," Operations Research, Articles in Advance.

[RaF91] Raghavan, T. E. S., and Filar, J. A., 1991. "Algorithms for Stochastic Games - A Survey," Zeitschrift fur Operations Research, Vol. 35, pp. 437-472.

[RaR17] Rawlings, J. B., and Risbeck, M. J., 2017. "Model Predictive Control with Discrete Actuators: Theory and Application," Automatica, Vol. 78, pp. 258-265.

[RaW06] Rasmussen, C. E., and Williams, C. K., 2006. Gaussian Processes for Machine Learning, MIT Press, Cambridge, MA.

[Rab89] Rabiner, L. R., 1989. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," Proc. of the IEEE, Vol. 77, pp. 257-286.

[Rad62] Radner, R., 1962. "Team Decision Problems," Ann. Math. Statist., Vol. 33, pp. 857-881.

[Ras63] Rastrigin, R. A., 1963. "About Convergence of Random Search Method in Extremal Control of Multi-Parameter Systems," Avtomat. i Telemekh., Vol. 24, pp. 1467-1473.

[RoB17] Rosolia, U., and Borrelli, F., 2017. "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework," IEEE Trans. on Automatic Control, Vol. 63, pp. 1883-1896.

[RoB19] Rosolia, U., and Borrelli, F., 2019. "Sample-Based Learning Model Predictive Control for Linear Uncertain Systems," 58th Conference on Decision and Control (CDC), pp. 2702-2707.

[Rob52] Robbins, H., 1952. "Some Aspects of the Sequential Design of Experiments," Bulletin of the American Mathematical Society, Vol. 58, pp. 527-535.

[Ros70] Ross, S. M., 1970. Applied Probability Models with Optimization Applications, Holden-Day, San Francisco, CA.

[Ros12] Ross, S. M., 2012. Simulation, 5th Edition, Academic Press, Orlando, Fla.

[Rot79] Rothblum, U. G., 1979. "Iterated Successive Approximation for Sequential Decision Processes," in Stochastic Control and Optimization, by J. W. B. van Overhagen and H. C. Tijms (eds), Vrije University, Amsterdam.

[RuK04] Rubinstein, R. Y., and Kroese, D. P., 2004. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Springer, N. Y.

[RuK13] Rubinstein, R. Y., and Kroese, D. P., 2013. The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning, Springer Science and Business Media.

[RuK16] Rubinstein, R. Y., and Kroese, D. P., 2016. Simulation and the Monte Carlo Method, 3rd Edition, J. Wiley, N. Y.

[RuN94] Rummery, G. A., and Niranjan, M., 1994. "On-Line Q-Learning Using Connectionist Systems," University of Cambridge, England, Department of Engineering, TR-166.

[RuN16] Russell, S. J., and Norvig, P., 2016. Artificial Intelligence: A Modern Approach, Pearson Education Limited, Malaysia.

[RuS03] Ruszczynski, A., and Shapiro, A., 2003. "Stochastic Programming Models," in Handbooks in Operations Research and Management Science, Vol. 10, pp. 1-64.

[Rub69] Rubinstein, R. Y., 1969. Some Problems in Monte Carlo Optimization, Ph.D. Thesis.

[SGC02] Savagaonkar, U., Givan, R., and Chong, E. K. P., 2002. "Sampling Techniques for Zero-Sum, Discounted Markov Games," in Proc. 40th Allerton Conference on Communication, Control and Computing, Monticello, Ill.

[SGG15] Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., and Geist, M., 2015. "Approximate Modified Policy Iteration and its Application to the Game of Tetris," J. of Machine Learning Research, Vol. 16, pp. 1629-1676.

[SHB15] Simroth, A., Holfeld, D., and Brunsch, R., 2015. "Job Shop Production Planning under Uncertainty: A Monte Carlo Rollout Approach," Proc. of the International Scientific and Practical Conference, Vol. 3, pp. 175-179.

[SHM16] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., and Dieleman, S., 2016. "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, Vol. 529, pp. 484-489.

[SHS17] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., and Lillicrap, T., 2017. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," arXiv:1712.01815.

[SJJ95] Singh, S. P., Jaakkola, T., and Jordan, M. I., 1995. "Reinforcement Learning with Soft State Aggregation," in Advances in Neural Information Processing Systems 7, MIT Press, Cambridge, MA.

[SJL18] Soltanolkotabi, M., Javanmard, A., and Lee, J. D., 2018. "Theoretical Insights into the Optimization Landscape of Over-Parameterized Shallow Neural Networks," IEEE Trans. on Information Theory, Vol. 65, pp. 742-769.

[SLA12] Snoek, J., Larochelle, H., and Adams, R. P., 2012. "Practical Bayesian Optimization of Machine Learning Algorithms," in Advances in Neural Information Processing Systems, pp. 2951-2959.

[SLJ13] Sun, B., Luh, P. B., Jia, Q. S., Jiang, Z., Wang, F., and Song, C., 2013. "Building Energy Management: Integrated Control of Active and Passive Heating, Cooling, Lighting, Shading, and Ventilation Systems," IEEE Trans. on Automation Science and Engineering, Vol. 10, pp. 588-602.

[SMS99] Sutton, R. S., McAllester, D., Singh, S. P., and Mansour, Y., 1999. "Policy Gradient Methods for Reinforcement Learning with Function Approximation," NIPS, Denver, Colorado.

[SNC18] Sarkale, Y., Nozhati, S., Chong, E. K., Ellingwood, B. R., and Mahmoud, H., 2018. "Solving Markov Decision Processes for Network-Level Post-Hazard Recovery via Simulation Optimization and Rollout," in 2018 IEEE 14th International Conference on Automation Science and Engineering, pp. 906-912.

[SSS17] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. "Mastering the Game of Go Without Human Knowledge," Nature, Vol. 550, pp. 354-359.

[SSW16] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N., 2015. "Taking the Human Out of the Loop: A Review of Bayesian Optimization," Proc. of IEEE, Vol. 104, pp. 148-175.

[SWD17] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., 2017. "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347.

[SWM89] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., 1989. "Design and Analysis of Computer Experiments," Statistical Science, pp. 409-423.

[SXX22] Shah, D., Xie, Q., and Xu, Z., 2022. "Nonasymptotic Analysis of Monte Carlo Tree Search," Operations Research, vol. 70, pp. 3234-3260.

[SYL17] Saldi, N., Yuksel, S., and Linder, T., 2017. "Finite Model Approximations for Partially Observed Markov Decision Processes with Discounted Cost," arXiv:1710.07009.

[SZL08] Sun, T., Zhao, Q., Lun, P., and Tomastik, R., 2008. "Optimization of Joint Replacement Policies for Multipart Systems by a Rollout Framework," IEEE Trans. on Automation Science and Engineering, Vol. 5, pp. 609-619.

[SaB11] Sastry, S., and Bodson, M., 2011. Adaptive Control: Stability, Convergence and Robustness, Courier Corporation.

[Sal21] Saldi, N., 2021. "Regularized Stochastic Team Problems," Systems and Control Letters, Vol. 149.

[Sas02] Sasena, M. J., 2002. Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations, PhD Thesis, Univ. of Michigan.

[ScS02] Scholkopf, B., and Smola, A. J., 2002. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, Cambridge, MA.

[Sch13] Scherrer, B., 2013. "Performance Bounds for Lambda Policy Iteration and Application to the Game of Tetris," J. of Machine Learning Research, Vol. 14, pp. 1181-1227.

[Sco10] Scott, S. L., 2010. "A Modern Bayesian Look at the Multi-Armed Bandit," Applied Stochastic Models in Business and Industry, Vol. 26, pp. 639-658.

[Sec00] Secomandi, N., 2000. "Comparing Neuro-Dynamic Programming Algorithms for the Vehicle Routing Problem with Stochastic Demands," Computers and Operations Research, Vol. 27, pp. 1201-1225.

[Sec01] Secomandi, N., 2001. "A Rollout Policy for the Vehicle Routing Problem with Stochastic Demands," Operations Research, Vol. 49, pp. 796-802.

[Sec03] Secomandi, N., 2003. "Analysis of a Rollout Approach to Sequencing Problems with Stochastic Routing Applications," J. of Heuristics, Vol. 9, pp. 321-352.

[ShC04] Shawe-Taylor, J., and Cristianini, N., 2004. Kernel Methods for Pattern Analysis, Cambridge Univ. Press.

[Sha50] Shannon, C., 1950. "Programming a Digital Computer for Playing Chess," Phil. Mag., Vol. 41, pp. 356-375.

[Sha53] Shapley, L. S., 1953. "Stochastic Games," Proc. of the National Academy of Sciences, Vol. 39, pp. 1095-1100.

[SiB22] Silver, D., and Barreto, A., 2022. "Simulation-Based Search," in Proc. Int. Cong. Math, Vol. 6, pp. 4800-4819.

[SiK19] Singh, R., and Kumar, P. R., 2019. "Optimal Decentralized Dynamic Policies for Video Streaming over Wireless Channels," arXiv:1902.07418.

[SlL91] Slotine, J.-J. E., and Li, W., Applied Nonlinear Control, Prentice-Hall, Englewood Cliffs, N. J.

[Spa92] Spall, J. C., "Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation," IEEE Trans. on Automatic Control, Vol. pp. 332-341.

[Spa03] Spall, J. C., 2003. Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control, J. Wiley, Hoboken, N. J.

[StW91] Stewart, B. S., and White, C. C., 1991. "Multiobjective A^*," J. ACM, Vol. 38, pp. 775-814.

[Ste94] Stengel, R. F., 1994. Optimal Control and Estimation, Courier Corporation.

[SuB18] Sutton, R., and Barto, A. G., 2018. Reinforcement Learning, 2nd Edition, MIT Press, Cambridge, MA.

[SuY19] Su, L., and Yang, P., 2019. 'On Learning Over-Parameterized Neural Networks: A Functional Approximation Perspective," in Advances in Neural Information Processing Systems, pp. 2637-2646.

[Sun19] Sun, R., 2019. "Optimization for Deep Learning: Theory and Algorithms," arXiv:1912.08957.

[SzL06] Szita, I., and Lorinz, A., 2006. "Learning Tetris Using the Noisy Cross-Entropy Method," Neural Computation, Vol. 18, pp. 2936-2941.

[Sze10] Szepesvari, C., 2010. Algorithms for Reinforcement Learning, Morgan and Claypool Publishers, San Franscisco, CA.

[Sze11] Szepesvari, C., 2011. "Least Squares Temporal Difference Learning and Galerkin?s Method," Presentation at the Mini-Workshop: Mathematics of Machine Learning, Mathematisches Forschungsinstitut Oberwolfach.

[TBP21] Tuncel, Y., Bhat, G., Park, J., and Ogras, U., 2021. "ECO: Enabling Energy-Neutral IoT Devices through Runtime Allocation of Harvested Energy," arXiv:2102.13605.

[TCW19] Tseng, W. J., Chen, J. C., Wu, I. C., and Wei, T. H., 2019. "Comparison Training for Computer Chinese Chess," IEEE Trans. on Games, Vol. 12, pp. 169-176.

[TGL13] Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., and Prager, J. M., 2013. "Analysis of Watson's Strategies for Playing Jeopardy!," J. of Artificial Intelligence Research, Vol. 47, pp. 205-251.

[TRV16] Tu, S., Roelofs, R., Venkataraman, S., and Recht, B., 2016. "Large Scale Kernel Learning Using Block Coordinate Descent," arXiv:1602.05310.

[TaL20] Tanzanakis, A., and Lygeros, J., 2020. "Data-Driven Control of Unknown Systems: A Linear Programming Approach," arXiv:2003.00779.

[TeG96] Tesauro, G., and Galperin, G. R., 1996. "On-Line Policy Improvement Using Monte Carlo Search," NIPS, Denver, CO.

[Tes89a] Tesauro, G. J., 1989. "Neurogammon Wins Computer Olympiad," Neural Computation, Vol. 1, pp. 321-323.

[Tes89b] Tesauro, G. J., 1989. "Connectionist Learning of Expert Preferences by Comparison Training," in Advances in Neural Information Processing Systems, pp. 99-106.

[Tes92] Tesauro, G. J., 1992. "Practical Issues in Temporal Difference Learning," Machine Learning, Vol. 8, pp. 257-277.

[Tes94] Tesauro, G. J., 1994. "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," Neural Computation, Vol. 6, pp. 215-219.

[Tes95] Tesauro, G. J., 1995. "Temporal Difference Learning and TD-Gammon," Communications of the ACM, Vol. 38, pp. 58-68.

[Tes01] Tesauro, G. J., 2001. "Comparison Training of Chess Evaluation Functions," in Machines that Learn to Play Games, Nova Science Publishers, pp. 117-130.

[Tes02] Tesauro, G. J., 2002. "Programming Backgammon Using Self-Teaching Neural Nets," Artificial Intelligence, Vol. 134, pp. 181-199.

[ThS09] Thiery, C., and Scherrer, B., 2009. "Improvements on Learning Tetris with Cross-Entropy," International Computer Games Association J., Vol. 32, pp. 23-33.

[Tol89] Tolwinski, B., 1989. "Newton-Type Methods for Stochastic Games," in Basar T. S., and Bernhard P. (eds), Differential Games and Applications, Lecture Notes in Control and Information Sciences, vol. 119, Springer, pp. 128-144.

[TsV96] Tsitsiklis, J. N., and Van Roy, B., 1996. "Feature-Based Methods for Large-Scale Dynamic Programming," Machine Learning, Vol. 22, pp. 59-94.

[Tse98] Tseng, P., 1998. "Incremental Gradient(-Projection) Method with Momentum Term and Adaptive Stepsize Rule," SIAM J. on Optimization, Vol. 8, pp. 506-531.

[TuP03] Tu, F., and Pattipati, K. R., 2003. "Rollout Strategies for Sequential Fault Diagnosis," IEEE Trans. on Systems, Man and Cybernetics, Part A, pp. 86-99.

[UGM18] Ulmer, M. W., Goodson, J. C., Mattfeld, D. C., and Hennig, M., 2018. "Offline-Online Approximate Dynamic Programming for Dynamic Vehicle Routing with Stochastic Requests," Transportation Science, Vol. 53, pp. 185-202.

[Ulm17] Ulmer, M. W., 2017. Approximate Dynamic Programming for Dynamic Vehicle Routing, Springer, Berlin.

[VBC19] Vinyals, O., Babuschkin, I., Czarnecki, W. M., and thirty nine more authors, 2019. "Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning," Nature, Vol. 575, p. 350.

[VLK21] Vaswani, S., Laradji, I. H., Kunstner, F., Meng, S. Y., Schmidt, M., and Lacoste-Julien, S., 2021. "Adaptive Gradient Methods Converge Faster with Over-Parameterization (but you should do a line search)," arXiv:2006.06835.

[VPA09] Vrabie, D., Pastravanu, O., Abu-Khalaf, M., and Lewis, F. L., 2009. "Adaptive Optimal Control for Continuous-Time Linear Systems Based on Policy Iteration," Automatica, Vol. 45, pp. 477-484.

[VVL13] Vrabie, D., Vamvoudakis, K. G., and Lewis, F. L., 2013. Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles, The Institution of Engineering and Technology, London.

[Van76] Van Nunen, J. A., 1976. Contracting Markov Decision Processes, Mathematical Centre Report, Amsterdam.

[Van78] van der Wal, J., 1978. "Discounted Markov Games: Generalized Policy Iteration Method," J. of Optimization Theory and Applications, Vol. 25, pp. 125-138.

[VeM23] Vertovec, N., and Margellos, K., 2023. "State Aggregation for Distributed Value Iteration in Dynamic Programming," arXiv preprint arXiv:2303.10675.

[Vit67] Viterbi, A. J., 1967. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. on Info. Theory, Vol. IT-13, pp. 260-269.

[WCG02] Wu, G., Chong, E. K. P., and Givan, R. L., 2002. "Burst-Level Congestion Control Using Hindsight Optimization," IEEE Transactions on Aut. Control, Vol. 47, pp. 979-991.

[WCG03] Wu, G., Chong, E. K. P., and Givan, R. L., 2003. "Congestion Control Using Policy Rollout," Proc. 2nd IEEE CDC, Maui, Hawaii, pp. 4825-4830.

[WGB22] Wang, T. T., Gleave, A., Belrose, N., Tseng, T., Miller, J., Dennis, M. D., Duan, Y., Pogrebniak, V., Levine, S., and Russell, S., 2022. "Adversarial Policies Beat Professional-Level Go AIs," arXiv preprint arXiv:2211.00241.

[WGP23] Weber, J., Giriyan, D., Parkar, D., Richa, A., Bertsekas, D., "Distributed Online Rollout for Multivehicle Routing in Unmapped Environments," arXiv preprint

arXiv:2305.11596v1.

[WOB15] Wang, Y., O'Donoghue, B., and Boyd, S., 2015. "Approximate Dynamic Programming via Iterated Bellman Inequalities," International J. of Robust and Nonlinear Control, Vol. 25, pp. 1472-1496.

[WaB14] Wang, M., and Bertsekas, D. P., 2014. "Incremental Constraint Projection Methods for Variational Inequalities," Mathematical Programming, pp. 1-43.

[WaB16] Wang, M., and Bertsekas, D. P., 2016. "Stochastic First-Order Methods with Random Constraint Projection," SIAM Journal on Optimization, Vol. 26, pp. 681-717.

[WaP17] Wang, J., and Paschalidis, I. C., 2017. "An Actor-Critic Algorithm with Second-Order Actor and Critic," IEEE Trans. on Automatic Control, Vol. 62, pp. 2689-2703.

[WaS00] de Waal, P. R., and van Schuppen, J. H., 2000. "A Class of Team Problems with Discrete Action Spaces: Optimality Conditions Based on Multimodularity," SIAM J. on Control and Optimization, Vol. 38, pp. 875-892.

[Wat89] Watkins, C. J. C. H., Learning from Delayed Rewards, Ph.D. Thesis, Cambridge Univ., England.

[WeB99] Weaver, L., and Baxter, J., 1999. "Learning from State Differences: STD(λ)," Tech. Report, Dept. of Computer Science, Australian National University.

[WeV17] Westhead, D. R., and Vijayabaskar, M. S., 2017. Hidden Markov Models, Springer, Berlin.

[WhS94] White, C. C., and Scherer, W. T., 1994. "Finite-Memory Suboptimal Design for Partially Observed Markov Decision Processes," Operations Research, Vol. 42, pp. 439-455.

[Whi82] Whittle, P., 1982. Optimization Over Time, Wiley, N. Y., Vol. 1, 1982, Vol. 2, 1983.

[Whi88] Whittle, P., 1988. "Restless Bandits: Activity Allocation in a Changing World," J. of Applied Probability, pp. 287-298.

[Whi91] White, C. C., 1991. "A Survey of Solution Techniques for the Partially Observed Markov Decision Process," Annals of Operations Research, Vol. 32, pp. 215-230.

[WiB93] Williams, R. J., and Baird, L. C., 1993. "Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor-Critic Learning Systems," Report NU-CCS-93-11, College of Computer Science, Northeastern University, Boston, MA.

[WiS98] Wiering, M., and Schmidhuber, J., 1998. "Fast Online Q(λ)," Machine Learning, Vol. 33, pp. 105-115.

[Wie03] Wiewiora, E., 2003. "Potential-Based Shaping and Q-Value Initialization are Equivalent," J. of Artificial Intelligence Research, Vol. 19, pp. 205-208.

[Wil92] Williams, R. J., 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," Machine Learning, Vol. 8, pp. 229-256.

[Wit66] Witsenhausen, H. S., 1966. Minimax Control of Uncertain Systems, Ph.D. thesis, MIT.

[Wit68] Witsenhausen, H., 1968. "A Counterexample in Stochastic Optimum Control," SIAM Journal on Control, Vol. 6, pp. 131-147.

[Wit71a] Witsenhausen, H. S., 1971. "On Information Structures, Feedback and Causality," SIAM J. Control, Vol. 9, pp. 149-160.

[Wit71b] Witsenhausen, H., 1971. "Separation of Estimation and Control for Discrete Time Systems," Proceedings of the IEEE, Vol. 59, pp. 1557-1566.

[YDR04] Yan, X., Diaconis, P., Rusmevichientong, P., and Van Roy, B., 2004. "Solitaire: Man Versus Machine," Advances in Neural Information Processing Systems, Vol. 17, pp. 1553-1560.

[YXK24] Yilmaz, M. B., Xiang, L. and Klein, A., 2024. "Joint Beamforming and Trajectory Optimization for UAV-Aided ISAC with Dipole Antenna Array," Report.

[YYM20] Yu, L., Yang, H., Miao, L., and Zhang, C., 2019. "Rollout Algorithms for Resource Allocation in Humanitarian Logistics," IISE Transactions, Vol. 51, pp. 887-909.

[Yar17] Yarotsky, D., 2017. "Error Bounds for Approximations with Deep ReLU Networks," Neural Networks, Vol. 94, pp. 103-114.

[YuB08] Yu, H., and Bertsekas, D. P., 2008. "On Near-Optimality of the Set of Finite-State Controllers for Average Cost POMDP," Math. of OR, Vol. 33, pp. 1-11.

[YuB09] Yu, H., and Bertsekas, D. P., 2009. "Basis Function Adaptation Methods for Cost Approximation in MDP," Proceedings of 2009 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2009), Nashville, Tenn.

[YuB10] Yu, H., and Bertsekas, D. P., 2010. "Error Bounds for Approximations from Projected Linear Equations," Math. of Operations Research, Vol. 35, pp. 306-329.

[YuB13] Yu, H., and Bertsekas, D. P., 2013. "Q-Learning and Policy Iteration Algorithms for Stochastic Shortest Path Problems," Annals of Operations Research, Vol. 208, pp. 95-132.

[YuB15] Yu, H., and Bertsekas, D. P., 2015. "A Mixed Value and Policy Iteration Method for Stochastic Control with Universally Measurable Policies," Math. of OR, Vol. 40, pp. 926-968.

[YuK20] Yue, X., and Kontar, R. A., 2020. "Lookahead Bayesian Optimization via Rollout: Guarantees and Sequential Rolling Horizons," arXiv:1911.01004.

[Yu05] Yu, H., 2005. "A Function Approximation Approach to Estimation of Policy Gradient for POMDP with Structured Policies," Proc. of the 21st Conference on Uncertainty in Artificial Intelligence, Edinburgh, Scotland.

[Yu14] Yu, H., 2014. "Stochastic Shortest Path Games and Q-Learning," arXiv:1412.8570.

[Yua19] Yuanhong, L. I. U., 2019. "Optimal Selection of Tests for Fault Detection and Isolation in Multi-Operating Mode System," Journal of Systems Engineering and Electronics, Vol. 30, pp. 425-434.

[ZBH16] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2016. "Understanding Deep Learning Requires Rethinking Generalization," arXiv:1611.03530.

[ZBH21] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O., 2021. "Understanding Deep Learning (Still) Requires Rethinking Generalization," Communications of the ACM, Vol. 64, pp. 107-115.

[ZOT18] Zhang, S., Ohlmann, J. W., and Thomas, B. W., 2018. "Dynamic Orienteering on a Network of Queues," Transportation Science, Vol. 52, pp. 691-706.

[ZSG20] Zoppoli, R., Sanguineti, M., Gnecco, G., and Parisini, T., 2020. Neural Approximations for Optimal Control and Decision, Springer.

[ZYB21] Zhang, K., Yang, Z. and Basar, T., 2021. "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," Handbook of Reinforcement Learning

and Control, pp. 321-384.

[ZuS81] Zuker, M., and Stiegler, P., 1981. "Optimal Computer Folding of Larger RNA Sequences Using Thermodynamics and Auxiliary Information," Nucleic Acids Res., Vol. 9, pp. 133-148.

Neuro-Dynamic Programming
Dimitri P. Bertsekas and John N. Tsitsiklis

Athena Scientific, 1996
512 pp., hardcover, ISBN 1-886529-10-8

This is the first textbook that fully explains the neuro-dynamic programming/reinforcement learning methodology, a breakthrough in the practical application of neural networks and dynamic programming to complex problems of planning, optimal decision making, and intelligent control.

From the review by George Cybenko for IEEE Computational Science and Engineering, May 1998:

"Neuro-Dynamic Programming is a remarkable monograph that integrates a sweeping mathematical and computational landscape into a coherent body of rigorous knowledge. The topics are current, the writing is clear and to the point, the examples are comprehensive and the historical notes and comments are scholarly."

"In this monograph, Bertsekas and Tsitsiklis have performed a Herculean task that will be studied and appreciated by generations to come. I strongly recommend it to scientists and engineers eager to seriously understand the mathematics and computations behind modern behavioral machine learning."

Among its special features, the book:

- Describes and unifies a large number of NDP methods, including several that are new
- Describes new approaches to formulation and solution of important problems in stochastic optimal control, sequential decision making, and discrete optimization
- Rigorously explains the mathematical principles behind NDP
- Illustrates through examples and case studies the practical application of NDP to complex problems from optimal resource allocation, optimal feedback control, data communications, game playing, and combinatorial optimization
- Presents extensive background and new research material on dynamic programming and neural network training

Neuro-Dynamic Programming is the winner of the 1997 INFORMS CSTS prize for research excellence in the interface between Operations Research and Computer Science

Reinforcement Learning and Optimal Control
Dimitri P. Bertsekas

Athena Scientific, 2019
388 pp., hardcover, ISBN 978-1-886529-39-7

This book explores the common boundary between optimal control and artificial intelligence, as it relates to reinforcement learning and simulation-based neural network methods. These are popular fields with many applications, which can provide approximate solutions to challenging sequential decision problems and large-scale dynamic programming (DP). The aim of the book is to organize coherently the broad mosaic of methods in these fields, which have a solid analytical and logical foundation, and have also proved successful in practice.

The book discusses both approximation in value space and approximation in policy space. It adopts a gradual expository approach, which proceeds along four directions:

- From exact DP to approximate DP: We first discuss exact DP algorithms, explain why they may be difficult to implement, and then use them as the basis for approximations.
- From finite horizon to infinite horizon problems: We first discuss finite horizon exact and approximate DP methodologies, which are intuitive and mathematically simple, and then progress to infinite horizon problems.
- From model-based to model-free implementations: We first discuss model-based implementations, and then we identify schemes that can be appropriately modified to work with a simulator.

The mathematical style of this book is somewhat different from the one of the author's DP books, and the 1996 neuro-dynamic programming (NDP) research monograph, written jointly with John Tsitsiklis. While we provide a rigorous, albeit short, mathematical account of the theory of finite and infinite horizon DP, and some fundamental approximation methods, we rely more on intuitive explanations and less on proof-based insights. Moreover, our mathematical requirements are quite modest: calculus, a minimal use of matrix-vector algebra, and elementary probability (mathematically complicated arguments involving laws of large numbers and stochastic convergence are bypassed in favor of intuitive explanations).

The book is supported by on-line video lectures and slides, as well as new research material, some of which has been covered in the present monograph.

Rollout, Policy Iteration, and Distributed Reinforcement Learning

Dimitri P. Bertsekas

Athena Scientific, 2020
480 pp., hardcover, ISBN 978-1-886529-07-6

This book develops in greater depth some of the methods from the author's Reinforcement Learning and Optimal Control textbook (Athena Scientific, 2019). It presents new research, relating to rollout algorithms, policy iteration, multiagent systems, partitioned architectures, and distributed asynchronous computation.

The application of the methodology to challenging discrete optimization problems, such as routing, scheduling, assignment, and mixed integer programming, including the use of neural network approximations within these contexts, is also discussed.

Much of the new research is inspired by the remarkable AlphaZero chess program, where policy iteration, value and policy networks, approximate lookahead minimization, and parallel computation all play an important role.

Among its special features, the book:

- Presents new research relating to distributed asynchronous computation, partitioned architectures, and multiagent systems, with application to challenging large scale optimization problems, such as combinatorial/discrete optimization, as well as partially observed Markov decision problems.
- Describes variants of rollout and policy iteration for problems with a multiagent structure, which allow the dramatic reduction of the computational requirements for lookahead minimization.
- Establishes connections of rollout algorithms and model predictive control, one of the most prominent control system design methodology.
- Expands the coverage of some research areas discussed in the author's 2019 textbook Reinforcement Learning and Optimal Control.
- Provides the mathematical analysis that supports the Newton step interpretations and the conclusions of the present book.

The book is supported by on-line video lectures and slides, as well as new research material, some of which has been covered in the present monograph.