

# *Exact Dynamic Programming*

<b>Contents</b>	
1.1. Deterministic Dynamic Programming . . . . .	p. 2
1.1.1. Deterministic Problems . . . . .	p. 2
1.1.2. The Dynamic Programming Algorithm . . . . .	p. 7
1.1.3. Approximation in Value Space . . . . .	p. 12
1.2. Stochastic Dynamic Programming . . . . .	p. 14
1.3. Examples, Variations, and Simplifications . . . . .	p. 18
1.3.1. Deterministic Shortest Path Problems . . . . .	p. 19
1.3.2. Discrete Deterministic Optimization . . . . .	p. 21
1.3.3. Problems with a Termination State . . . . .	p. 25
1.3.4. Forecasts . . . . .	p. 26
1.3.5. Problems with Uncontrollable State Components . . . . .	p. 29
1.3.6. Partial State Information and Belief States . . . . .	p. 34
1.3.7. Linear Quadratic Optimal Control . . . . .	p. 38
1.3.8. Systems with Unknown Parameters - Adaptive Control . . . . .	p. 40
1.4. Reinforcement Learning and Optimal Control - Some Terminology . . . . .	p. 43
1.5. Notes and Sources . . . . .	p. 45

In this chapter, we provide some background on exact dynamic programming (DP for short), with a view towards the suboptimal solution methods that are the main subject of this book. These methods are known by several essentially equivalent names: *reinforcement learning*, *approximate dynamic programming*, and *neuro-dynamic programming*. In this book, we will use primarily the most popular name: reinforcement learning (RL for short).

We first consider finite horizon problems, which involve a finite sequence of successive decisions, and are thus conceptually and analytically simpler. We defer the discussion of the more intricate infinite horizon problems to Chapters 4-6. We also discuss separately deterministic and stochastic problems (Sections 1.1 and 1.2, respectively). The reason is that deterministic problems are simpler and lend themselves better as an entry point to the optimal control methodology. Moreover, they have some favorable characteristics, which allow the application of a broader variety of methods. For example, simulation-based methods are greatly simplified and sometimes better understood in the context of deterministic optimal control.

Finally, in Section 1.3 we provide various examples of DP formulations, illustrating some of the concepts of Sections 1.1 and 1.2. The reader with substantial background in DP may wish to just scan Section 1.3 and skip to the next chapter, where we start the development of the approximate DP methodology.

## 1.1 DETERMINISTIC DYNAMIC PROGRAMMING

All DP problems involve a discrete-time dynamic system that generates a sequence of states under the influence of control. In finite horizon problems the system evolves over a finite number  $N$  of time steps (also called stages). The state and control at time  $k$  are denoted by  $x_k$  and  $u_k$ , respectively. In deterministic systems,  $x_{k+1}$  is generated nonrandomly, i.e., it is determined solely by  $x_k$  and  $u_k$ .

### 1.1.1 Deterministic Problems

A deterministic DP problem involves a discrete-time dynamic system of the form

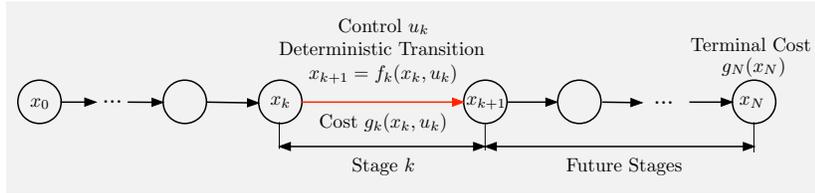
$$x_{k+1} = f_k(x_k, u_k), \quad k = 0, 1, \dots, N-1, \quad (1.1)$$

where

$k$  is the time index,

$x_k$  is the state of the system, an element of some space,

$u_k$  is the control or decision variable, to be selected at time  $k$  from some given set  $U_k(x_k)$  that depends on  $x_k$ ,



**Figure 1.1.1** Illustration of a deterministic  $N$ -stage optimal control problem. Starting from state  $x_k$ , the next state under control  $u_k$  is generated nonrandomly, according to

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

and a stage cost  $g_k(x_k, u_k)$  is incurred.

$f_k$  is a function of  $(x_k, u_k)$  that describes the mechanism by which the state is updated from time  $k$  to time  $k + 1$ .

$N$  is the horizon or number of times control is applied,

The set of all possible  $x_k$  is called the *state space* at time  $k$ . It can be any set and can depend on  $k$ ; *this generality is one of the great strengths of the DP methodology*. Similarly, the set of all possible  $u_k$  is called the *control space* at time  $k$ . Again it can be any set and can depend on  $k$ .

The problem also involves a cost function that is additive in the sense that the cost incurred at time  $k$ , denoted by  $g_k(x_k, u_k)$ , accumulates over time. Formally,  $g_k$  is a function of  $(x_k, u_k)$  that takes real number values, and may depend on  $k$ . For a given initial state  $x_0$ , the total cost of a control sequence  $\{u_0, \dots, u_{N-1}\}$  is

$$J(x_0; u_0, \dots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k), \quad (1.2)$$

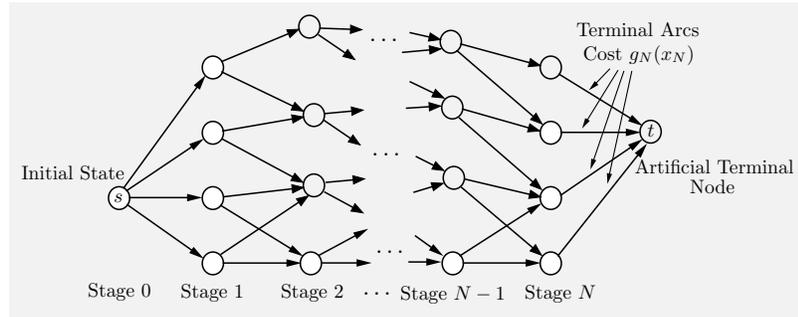
where  $g_N(x_N)$  is a terminal cost incurred at the end of the process. The total cost is a well-defined number, since the control sequence  $\{u_0, \dots, u_{N-1}\}$  together with  $x_0$  determines exactly the state sequence  $\{x_1, \dots, x_N\}$  via the system equation (1.1). We want to minimize the cost (1.2) over all sequences  $\{u_0, \dots, u_{N-1}\}$  that satisfy the control constraints, thereby obtaining the optimal value<sup>†</sup>

$$J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k=0, \dots, N-1}} J(x_0; u_0, \dots, u_{N-1}),$$

as a function of  $x_0$ . Figure 1.1.1 illustrates the main elements of the problem.

We will next illustrate deterministic problems with some examples.

<sup>†</sup> We use throughout “min” (in place of “inf”) to indicate minimal value over a feasible set of controls, even when we are not sure that the minimum is attained by some feasible control.



**Figure 1.1.2** Transition graph for a deterministic finite-state system. Nodes correspond to states  $x_k$ . Arcs correspond to state-control pairs  $(x_k, u_k)$ . An arc  $(x_k, u_k)$  has start and end nodes  $x_k$  and  $x_{k+1} = f_k(x_k, u_k)$ , respectively. We view the cost  $g_k(x_k, u_k)$  of the transition as the length of this arc. The problem is equivalent to finding a shortest path from initial node  $s$  to terminal node  $t$ .

## Discrete Optimal Control Problems

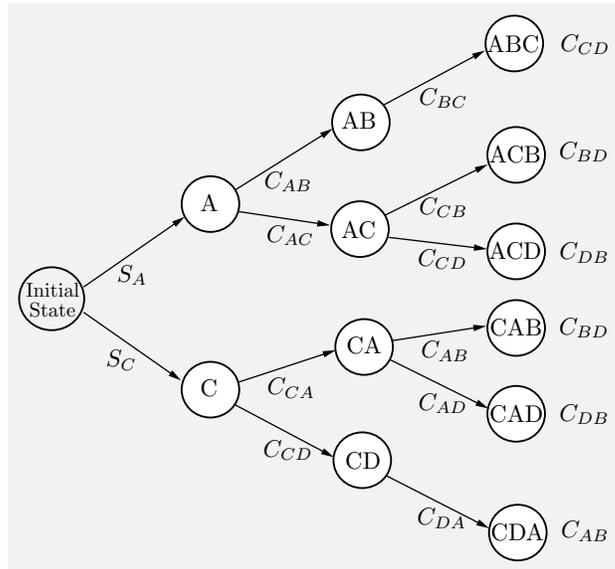
There are many situations where the state and control are naturally discrete and take a finite number of values. Such problems are often conveniently described with an acyclic graph specifying for each state  $x_k$  the possible transitions to next states  $x_{k+1}$ . The nodes of the graph correspond to states  $x_k$  and the arcs of the graph correspond to state-control pairs  $(x_k, u_k)$ . Each arc with start node  $x_k$  corresponds to a choice of a single control  $u_k \in U_k(x_k)$  and has as end node the next state  $f_k(x_k, u_k)$ . The cost of an arc  $(x_k, u_k)$  is defined as  $g_k(x_k, u_k)$ ; see Fig. 1.1.2. To handle the final stage, an artificial terminal node  $t$  is added. Each state  $x_N$  at stage  $N$  is connected to the terminal node  $t$  with an arc having cost  $g_N(x_N)$ .

Note that control sequences  $\{u_0, \dots, u_{N-1}\}$  correspond to paths originating at the initial state (node  $s$  at stage 0) and terminating at one of the nodes corresponding to the final stage  $N$ . If we view the cost of an arc as its length, we see that *a deterministic finite-state finite-horizon problem is equivalent to finding a minimum-length (or shortest) path from the initial node  $s$  of the graph to the terminal node  $t$* . Here, by the length of a path we mean the sum of the lengths of its arcs.<sup>†</sup>

Generally, combinatorial optimization problems can be formulated as deterministic finite-state finite-horizon optimal control problem. The following scheduling example illustrates the idea.

---

<sup>†</sup> It turns out also that any shortest path problem (with a possibly nonacyclic graph) can be reformulated as a finite-state deterministic optimal control problem, as we will show in Section 1.3.1. See [Ber17], Section 2.1, and [Ber91], [Ber98] for an extensive discussion of shortest path methods, which connects with our discussion here.



**Figure 1.1.3** The transition graph of the deterministic scheduling problem of Example 1.1.1. Each arc of the graph corresponds to a decision leading from some state (the start node of the arc) to some other state (the end node of the arc). The corresponding cost is shown next to the arc. The cost of the last operation is shown as a terminal cost next to the terminal nodes of the graph.

**Example 1.1.1 (A Deterministic Scheduling Problem)**

Suppose that to produce a certain product, four operations must be performed on a certain machine. The operations are denoted by A, B, C, and D. We assume that operation B can be performed only after operation A has been performed, and operation D can be performed only after operation C has been performed. (Thus the sequence CDAB is allowable but the sequence CDBA is not.) The setup cost  $C_{mn}$  for passing from any operation  $m$  to any other operation  $n$  is given. There is also an initial startup cost  $S_A$  or  $S_C$  for starting with operation A or C, respectively (cf. Fig. 1.1.3). The cost of a sequence is the sum of the setup costs associated with it; for example, the operation sequence ACDB has cost

$$S_A + C_{AC} + C_{CD} + C_{DB}.$$

We can view this problem as a sequence of three decisions, namely the choice of the first three operations to be performed (the last operation is determined from the preceding three). It is appropriate to consider as state the set of operations already performed, the initial state being an artificial state corresponding to the beginning of the decision process. The possible state transitions corresponding to the possible states and decisions for this

problem are shown in Fig. 1.1.3. Here the problem is deterministic, i.e., at a given state, each choice of control leads to a uniquely determined state. For example, at state AC the decision to perform operation D leads to state ACD with certainty, and has cost  $C_{CD}$ . Thus the problem can be conveniently represented with the transition graph of Fig. 1.1.3. The optimal solution corresponds to the path that starts at the initial state and ends at some state at the terminal time and has minimum sum of arc costs plus the terminal cost.

### Continuous-Spaces Optimal Control Problems

Many classical problems in control theory involve a state that belongs to a Euclidean space, i.e., the space of  $n$ -dimensional vectors of real variables, where  $n$  is some positive integer. The following is representative of the class of *linear-quadratic problems*, where the system equation is linear, the cost function is quadratic, and there are no control constraints. In our example, the states and controls are one-dimensional, but there are multidimensional extensions, which are very popular (see [Ber17], Section 3.1).

#### Example 1.1.2 (A Linear-Quadratic Problem)

A certain material is passed through a sequence of  $N$  ovens (see Fig. 1.1.4). Denote

$x_0$ : initial temperature of the material,

$x_k, k = 1, \dots, N$ : temperature of the material at the exit of oven  $k$ ,

$u_{k-1}, k = 1, \dots, N$ : heat energy applied to the material in oven  $k$ .

In practice there will be some constraints on  $u_k$ , such as nonnegativity. However, for analytical tractability one may also consider the case where  $u_k$  is unconstrained, and check later if the solution satisfies some natural restrictions in the problem at hand.

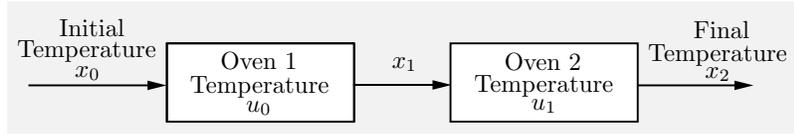
We assume a system equation of the form

$$x_{k+1} = (1 - a)x_k + au_k, \quad k = 0, 1, \dots, N - 1,$$

where  $a$  is a known scalar from the interval  $(0, 1)$ . The objective is to get the final temperature  $x_N$  close to a given target  $T$ , while expending relatively little energy. We express this with a cost function of the form

$$r(x_N - T)^2 + \sum_{k=0}^{N-1} u_k^2,$$

where  $r > 0$  is a given scalar that trades off approaching the final temperature with the expended energy.



**Figure 1.1.4** The linear-quadratic problem of Example 1.1.2 for  $N = 2$ . The temperature of the material evolves according to the system equation  $x_{k+1} = (1 - a)x_k + au_k$ , where  $a$  is some scalar with  $0 < a < 1$ .

Linear-quadratic problems with no constraints on the state or the control admit a nice analytical solution, as we will see later in Section 1.3.7. In another frequently arising optimal control problem there are linear constraints on the state and/or the control. In the preceding example it would have been natural to require that  $a_k \leq x_k \leq b_k$  and/or  $c_k \leq u_k \leq d_k$ , where  $a_k, b_k, c_k, d_k$  are given scalars. Then the problem would be solvable not only by DP but also by quadratic programming methods. Generally deterministic optimal control problems with continuous state and control spaces (in addition to DP) admit a solution by nonlinear programming methods, such as gradient, conjugate gradient, and Newton's method, which can be suitably adapted to their special structure.

### 1.1.2 The Dynamic Programming Algorithm

In this section we will state the DP algorithm and formally justify it. The algorithm rests on a simple idea, the *principle of optimality*, which roughly states the following; see Fig. 1.1.5.

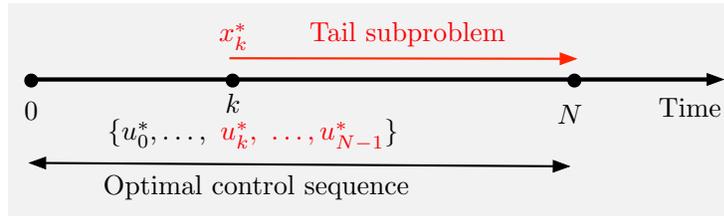
#### Principle of Optimality

Let  $\{u_0^*, \dots, u_{N-1}^*\}$  be an optimal control sequence, which together with  $x_0$  determines the corresponding state sequence  $\{x_1^*, \dots, x_N^*\}$  via the system equation (1.1). Consider the subproblem whereby we start at  $x_k^*$  at time  $k$  and wish to minimize the “cost-to-go” from time  $k$  to time  $N$ ,

$$g_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N),$$

over  $\{u_k, \dots, u_{N-1}\}$  with  $u_m \in U_m(x_m)$ ,  $m = k, \dots, N - 1$ . Then the truncated optimal control sequence  $\{u_k^*, \dots, u_{N-1}^*\}$  is optimal for this subproblem.

The subproblem referred to above is called the *tail subproblem* that starts at  $x_k^*$ . Stated succinctly, the principle of optimality says that *the tail of an optimal sequence is optimal for the tail subproblem*. Its intuitive



**Figure 1.1.5** Illustration of the principle of optimality. The tail  $\{u_k^*, \dots, u_{N-1}^*\}$  of an optimal sequence  $\{u_0^*, \dots, u_{N-1}^*\}$  is optimal for the tail subproblem that starts at the state  $x_k^*$  of the optimal trajectory  $\{x_1^*, \dots, x_N^*\}$ .

justification is simple. If the truncated control sequence  $\{u_k^*, \dots, u_{N-1}^*\}$  were not optimal as stated, we would be able to reduce the cost further by switching to an optimal sequence for the subproblem once we reach  $x_k^*$  (since the preceding choices of controls,  $u_0^*, \dots, u_{k-1}^*$ , do not restrict our future choices). For an auto travel analogy, suppose that the fastest route from Los Angeles to Boston passes through Chicago. The principle of optimality translates to the obvious fact that the Chicago to Boston portion of the route is also the fastest route for a trip that starts from Chicago and ends in Boston.

The principle of optimality suggests that the optimal cost function can be constructed in piecemeal fashion going backwards: first compute the optimal cost function for the “tail subproblem” involving the last stage, then solve the “tail subproblem” involving the last two stages, and continue in this manner until the optimal cost function for the entire problem is constructed.

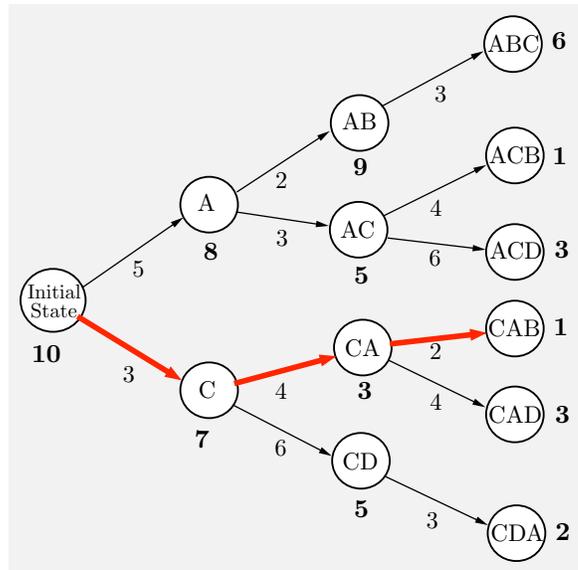
The DP algorithm is based on this idea: it proceeds sequentially, by *solving all the tail subproblems of a given time length, using the solution of the tail subproblems of shorter time length*. We illustrate the algorithm with the scheduling problem of Example 1.1.1. The calculations are simple but tedious, and may be skipped without loss of continuity. However, they may be worth going over by a reader that has no prior experience in the use of DP.

### Example 1.1.1 (Scheduling Problem - Continued)

Let us consider the scheduling Example 1.1.1, and let us apply the principle of optimality to calculate the optimal schedule. We have to schedule optimally the four operations A, B, C, and D. There a cost for a transition between two operations, and the numerical values of the transition costs are shown in Fig. 1.1.6 next to the corresponding arcs.

According to the principle of optimality, the “tail” portion of an optimal schedule must be optimal. For example, suppose that the optimal schedule is CABD. Then, having scheduled first C and then A, it must be optimal to complete the schedule with BD rather than with DB. With this in mind, we

solve all possible tail subproblems of length two, then all tail subproblems of length three, and finally the original problem that has length four (the subproblems of length one are of course trivial because there is only one operation that is as yet unscheduled). As we will see shortly, the tail subproblems of length  $k + 1$  are easily solved once we have solved the tail subproblems of length  $k$ , and this is the essence of the DP technique.



**Figure 1.1.6** Transition graph of the deterministic scheduling problem, with the cost of each decision shown next to the corresponding arc. Next to each node/state we show the cost to optimally complete the schedule starting from that state. This is the optimal cost of the corresponding tail subproblem (cf. the principle of optimality). The optimal cost for the original problem is equal to 10, as shown next to the initial state. The optimal schedule corresponds to the thick-line arcs.

*Tail Subproblems of Length 2:* These subproblems are the ones that involve two unscheduled operations and correspond to the states AB, AC, CA, and CD (see Fig. 1.1.6).

*State AB:* Here it is only possible to schedule operation C as the next operation, so the optimal cost of this subproblem is 9 (the cost of scheduling C after B, which is 3, plus the cost of scheduling D after C, which is 6).

*State AC:* Here the possibilities are to (a) schedule operation B and then D, which has cost 5, or (b) schedule operation D and then B, which has cost 9. The first possibility is optimal, and the corresponding cost of the tail subproblem is 5, as shown next to node AC in Fig. 1.1.6.

*State CA:* Here the possibilities are to (a) schedule operation B and then D, which has cost 3, or (b) schedule operation D and then B, which has cost 7. The first possibility is optimal, and the corresponding cost of the tail subproblem is 3, as shown next to node CA in Fig. 1.1.6.

*State CD:* Here it is only possible to schedule operation A as the next operation, so the optimal cost of this subproblem is 5.

*Tail Subproblems of Length 3:* These subproblems can now be solved using the optimal costs of the subproblems of length 2.

*State A:* Here the possibilities are to (a) schedule next operation B (cost 2) and then solve optimally the corresponding subproblem of length 2 (cost 9, as computed earlier), a total cost of 11, or (b) schedule next operation C (cost 3) and then solve optimally the corresponding subproblem of length 2 (cost 5, as computed earlier), a total cost of 8. The second possibility is optimal, and the corresponding cost of the tail subproblem is 8, as shown next to node A in Fig. 1.1.6.

*State C:* Here the possibilities are to (a) schedule next operation A (cost 4) and then solve optimally the corresponding subproblem of length 2 (cost 3, as computed earlier), a total cost of 7, or (b) schedule next operation D (cost 6) and then solve optimally the corresponding subproblem of length 2 (cost 5, as computed earlier), a total cost of 11. The first possibility is optimal, and the corresponding cost of the tail subproblem is 7, as shown next to node C in Fig. 1.1.6.

*Original Problem of Length 4:* The possibilities here are (a) start with operation A (cost 5) and then solve optimally the corresponding subproblem of length 3 (cost 8, as computed earlier), a total cost of 13, or (b) start with operation C (cost 3) and then solve optimally the corresponding subproblem of length 3 (cost 7, as computed earlier), a total cost of 10. The second possibility is optimal, and the corresponding optimal cost is 10, as shown next to the initial state node in Fig. 1.1.6.

Note that having computed the optimal cost of the original problem through the solution of all the tail subproblems, we can construct the optimal schedule: we begin at the initial node and proceed forward, each time choosing the optimal operation, i.e., the one that starts the optimal schedule for the corresponding tail subproblem. In this way, by inspection of the graph and the computational results of Fig. 1.1.6, we determine that CABD is the optimal schedule.

## Finding an Optimal Control Sequence by DP

We now state the DP algorithm for deterministic finite horizon problem by translating into mathematical terms the heuristic argument underlying the principle of optimality. The algorithm constructs functions

$$J_N^*(x_N), J_{N-1}^*(x_{N-1}), \dots, J_0^*(x_0),$$

sequentially, starting from  $J_N^*$ , and proceeding backwards to  $J_{N-1}^*$ ,  $J_{N-2}^*$ , etc.

**DP Algorithm for Deterministic Finite Horizon Problems**

Start with

$$J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N, \quad (1.3)$$

and for  $k = 0, \dots, N - 1$ , let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \left[ g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k)) \right], \quad \text{for all } x_k. \quad (1.4)$$

Note that at stage  $k$ , the calculation in (1.4) must be done for all states  $x_k$  before proceeding to stage  $k - 1$ . The key fact about the DP algorithm is that for every initial state  $x_0$ , the number  $J_0^*(x_0)$  obtained at the last step, is equal to the optimal cost  $J^*(x_0)$ . Indeed, a more general fact can be shown, namely that for all  $k = 0, 1, \dots, N - 1$ , and all states  $x_k$  at time  $k$ , we have

$$J_k^*(x_k) = \min_{\substack{u_m \in U_m(x_m) \\ m=k, \dots, N-1}} J(x_k; u_k, \dots, u_{N-1}), \quad (1.5)$$

where

$$J(x_k; u_k, \dots, u_{N-1}) = g_N(x_N) + \sum_{m=k}^{N-1} g_m(x_m, u_m), \quad (1.6)$$

i.e.,  $J_k^*(x_k)$  is the optimal cost for an  $(N - k)$ -stage tail subproblem that starts at state  $x_k$  and time  $k$ , and ends at time  $N$ .<sup>†</sup> Based on this fact, we

---

<sup>†</sup> We can prove this by induction. The assertion holds for  $k = N$  in view of the initial condition  $J_N^*(x_N) = g_N(x_N)$ . To show that it holds for all  $k$ , we use Eqs. (1.5) and (1.6) to write

$$\begin{aligned} J_k^*(x_k) &= \min_{\substack{u_m \in U_m(x_m) \\ m=k, \dots, N-1}} \left[ g_N(x_N) + \sum_{m=k}^{N-1} g_m(x_m, u_m) \right] \\ &= \min_{u_k \in U_k(x_k)} \left[ g_k(x_k, u_k) \right. \\ &\quad \left. + \min_{\substack{u_m \in U_m(x_m) \\ m=k+1, \dots, N-1}} \left[ g_N(x_N) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) \right] \right] \\ &= \min_{u_k \in U_k(x_k)} \left[ g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k)) \right], \end{aligned}$$

call  $J_k^*(x_k)$  the *optimal cost-to-go* at state  $x_k$  and time  $k$ , and refer to  $J_k^*$  as the *optimal cost-to-go function* or *optimal cost function* at time  $k$ . In maximization problems the DP algorithm (1.4) is written with maximization in place of minimization, and then  $J_k^*$  is referred to as the *optimal value function* at time  $k$ .

Note that the algorithm solves every tail subproblem, i.e., the minimization of the cost accumulated additively starting from an intermediate state up to the end of the horizon. Once the functions  $J_0^*, \dots, J_N^*$  have been obtained, we can use the following forward algorithm to construct an optimal control sequence  $\{u_0^*, \dots, u_{N-1}^*\}$  and corresponding state trajectory  $\{x_1^*, \dots, x_N^*\}$  for the given initial state  $x_0$ .

**Construction of Optimal Control Sequence  $\{u_0^*, \dots, u_{N-1}^*\}$**

Set

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \left[ g_0(x_0, u_0) + J_1^*(f_0(x_0, u_0)) \right],$$

and

$$x_1^* = f_0(x_0, u_0^*).$$

Sequentially, going forward, for  $k = 1, 2, \dots, N - 1$ , set

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \left[ g_k(x_k^*, u_k) + J_{k+1}^*(f_k(x_k^*, u_k)) \right], \quad (1.7)$$

and

$$x_{k+1}^* = f_k(x_k^*, u_k^*).$$

The same algorithm can be used to find an optimal control sequence for any tail subproblem. Figure 1.1.6 traces the calculations of the DP algorithm for the scheduling Example 1.1.1. The numbers next to the nodes, give the corresponding cost-to-go values, and the thick-line arcs give the construction of the optimal control sequence using the preceding algorithm.

### 1.1.3 Approximation in Value Space

The preceding forward optimal control sequence construction is possible only after we have computed  $J_k^*(x_k)$  by DP for all  $x_k$  and  $k$ . Unfortunately, in practice this is often prohibitively time-consuming, because of

---

where for the last equality we use the induction hypothesis. A subtle mathematical point here is that, through the minimization operation, the cost-to-go functions  $J_k^*$  may take the value  $-\infty$  for some  $x_k$ . Still the preceding induction argument is valid even if this is so.

the number of possible  $x_k$  and  $k$  can be very large. However, a similar forward algorithmic process can be used if the optimal cost-to-go functions  $J_k^*$  are replaced by some approximations  $\tilde{J}_k$ . This is the basis for *approximation in value space*, which will be central in our future discussions. It constructs a suboptimal solution  $\{\tilde{u}_0, \dots, \tilde{u}_{N-1}\}$  in place of the optimal  $\{u_0^*, \dots, u_{N-1}^*\}$ , based on using  $\tilde{J}_k$  in place of  $J_k^*$  in the DP procedure (1.7).

### Approximation in Value Space - Use of $\tilde{J}_k$ in Place of $J_k^*$

Start with

$$\tilde{u}_0 \in \arg \min_{u_0 \in U_0(x_0)} \left[ g_0(x_0, u_0) + \tilde{J}_1(f_0(x_0, u_0)) \right],$$

and set

$$\tilde{x}_1 = f_0(x_0, \tilde{u}_0).$$

Sequentially, going forward, for  $k = 1, 2, \dots, N - 1$ , set

$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{x}_k)} \left[ g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}(f_k(\tilde{x}_k, u_k)) \right], \quad (1.8)$$

and

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

The construction of suitable approximate cost-to-go functions  $\tilde{J}_k$  is a major focal point of the RL methodology. There are several possible methods, depending on the context, and they will be taken up starting with the next chapter.

## Q-Factors and Q-Learning

The expression

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

which appears in the right-hand side of Eq. (1.8) is known as the (approximate) *Q-factor of  $(x_k, u_k)$* .<sup>†</sup> In particular, the computation of the

<sup>†</sup> The term “Q-learning” and some of the associated algorithmic ideas were introduced in the thesis by Watkins [Wat89] (after the symbol “Q” that he used to represent Q-factors). The term “Q-factor” was used in the book [BeT96], and is adopted here as well. Watkins [Wat89] used the term “action value” (at a given state). The terms “state-action value” and “Q-value” are also common in the literature.

approximately optimal control (1.8) can be done through the Q-factor minimization

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

This suggests the possibility of using Q-factors in place of cost functions in approximation in value space schemes. Methods of this type use as starting point an alternative (and equivalent) form of the DP algorithm, which instead of the optimal cost-to-go functions  $J_k^*$ , generates the *optimal Q-factors*, defined for all pairs  $(x_k, u_k)$  and  $k$  by

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k)). \quad (1.9)$$

Thus the optimal Q-factors are simply the expressions that are minimized in the right-hand side of the DP equation (1.4). Note that this equation implies that the optimal cost function  $J_k^*$  can be recovered from the optimal Q-factor  $Q_k^*$  by means of

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k).$$

Moreover, using the above relation, the DP algorithm can be written in an essentially equivalent form that involves Q-factors only

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + \min_{u_{k+1} \in U_{k+1}(f_k(x_k, u_k))} Q_{k+1}^*(f_k(x_k, u_k), u_{k+1}).$$

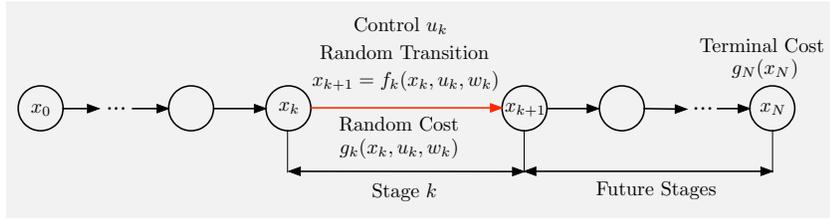
We will discuss later exact and approximate forms of related algorithms in the context of a class of RL methods known as *Q-learning*.

## 1.2 STOCHASTIC DYNAMIC PROGRAMMING

The stochastic finite horizon optimal control problem differs from the deterministic version primarily in the nature of the discrete-time dynamic system that governs the evolution of the state  $x_k$ . This system includes a random “disturbance”  $w_k$ , which is characterized by a probability distribution  $P_k(\cdot | x_k, u_k)$  that may depend explicitly on  $x_k$  and  $u_k$ , but not on values of prior disturbances  $w_{k-1}, \dots, w_0$ . The system has the form

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \dots, N-1,$$

where as before  $x_k$  is an element of some state space  $S_k$ , the control  $u_k$  is an element of some control space. The cost per stage is denoted  $g_k(x_k, u_k, w_k)$  and also depends on the random disturbance  $w_k$ ; see Fig. 1.2.1. The control



**Figure 1.2.1** Illustration of an  $N$ -stage stochastic optimal control problem. Starting from state  $x_k$ , the next state under control  $u_k$  is generated randomly, according to

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

where  $w_k$  is the random disturbance, and a random stage cost  $g_k(x_k, u_k, w_k)$  is incurred.

$u_k$  is constrained to take values in a given subset  $U_k(x_k)$ , which depends on the current state  $x_k$ .

An important difference is that we optimize not over control sequences  $\{u_0, \dots, u_{N-1}\}$ , but rather over *policies* (also called *closed-loop control laws*, or *feedback policies*) that consist of a sequence of functions

$$\pi = \{\mu_0, \dots, \mu_{N-1}\},$$

where  $\mu_k$  maps states  $x_k$  into controls  $u_k = \mu_k(x_k)$ , and satisfies the control constraints, i.e., is such that  $\mu_k(x_k) \in U_k(x_k)$  for all  $x_k \in S_k$ . Policies are more general objects than control sequences, and in the presence of stochastic uncertainty, they can result in improved cost, since they allow choices of controls  $u_k$  that incorporate knowledge of the state  $x_k$ . Without this knowledge, the controller cannot adapt appropriately to unexpected values of the state, and as a result the cost can be adversely affected. This is a fundamental distinction between deterministic and stochastic optimal control problems.

Another important distinction between deterministic and stochastic problems is that in the latter, the evaluation of various quantities such as cost function values involves forming expected values. Consequently, several of the methods that we will discuss for stochastic problems will involve the use of Monte Carlo simulation.

Given an initial state  $x_0$  and a policy  $\pi = \{\mu_0, \dots, \mu_{N-1}\}$ , the future states  $x_k$  and disturbances  $w_k$  are random variables with distributions defined through the system equation

$$x_{k+1} = f_k(x_k, \mu_k(x_k), w_k), \quad k = 0, 1, \dots, N-1.$$

Thus, for given functions  $g_k$ ,  $k = 0, 1, \dots, N$ , the expected cost of  $\pi$  starting at  $x_0$  is

$$J_\pi(x_0) = E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\},$$

where the expected value operation  $E\{\cdot\}$  is over all the random variables  $w_k$  and  $x_k$ . An optimal policy  $\pi^*$  is one that minimizes this cost; i.e.,

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_{\pi}(x_0),$$

where  $\Pi$  is the set of all policies.

The optimal cost depends on  $x_0$  and is denoted by  $J^*(x_0)$ ; i.e.,

$$J^*(x_0) = \min_{\pi \in \Pi} J_{\pi}(x_0).$$

It is useful to view  $J^*$  as a function that assigns to each initial state  $x_0$  the optimal cost  $J^*(x_0)$ , and call it the *optimal cost function* or *optimal value function*.

### Finite Horizon Stochastic Dynamic Programming

The DP algorithm for the stochastic finite horizon optimal control problem has a similar form to its deterministic version, and shares several of its major characteristics:

- (a) Using tail subproblems to break down the minimization over multiple stages to single stage minimizations.
- (b) Generating backwards for all  $k$  and  $x_k$  the values  $J_k^*(x_k)$ , which give the optimal cost-to-go starting at stage  $k$  at state  $x_k$ .
- (c) Obtaining an optimal policy by minimization in the DP equations.
- (d) A structure that is suitable for approximation in value space, whereby we replace  $J_k^*$  by approximations  $\tilde{J}_k$ , and obtain a suboptimal policy by the corresponding minimization.

#### DP Algorithm for Stochastic Finite Horizon Problems

Start with

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

and for  $k = 0, \dots, N - 1$ , let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E\left\{g_k(x_k, u_k, w_k) + J_{k+1}^*(f_k(x_k, u_k, w_k))\right\}. \quad (1.10)$$

If  $u_k^* = \mu_k^*(x_k)$  minimizes the right side of this equation for each  $x_k$  and  $k$ , the policy  $\pi^* = \{\mu_0^*, \dots, \mu_{N-1}^*\}$  is optimal.

The key fact is that for every initial state  $x_0$ , the optimal cost  $J^*(x_0)$  is equal to the function  $J_0^*(x_0)$ , obtained at the last step of the above DP

algorithm. This can be proved by induction similar to the deterministic case; we will omit the proof (see the discussion of Section 1.3 in the textbook [Ber17]).<sup>†</sup>

Simultaneously with the off-line computation of the optimal cost-to-go functions  $J_0^*, \dots, J_N^*$ , we can compute and store an optimal policy  $\pi^* = \{\mu_0^*, \dots, \mu_{N-1}^*\}$  by minimization in Eq. (1.10). We can then use this policy on-line to retrieve from memory and apply the control  $\mu_k^*(x_k)$  once we reach state  $x_k$ .

The alternative is to forego the storage of the policy  $\pi^*$  and to calculate the control  $\mu_k^*(x_k)$  by executing the minimization (1.10) on-line. This method of on-line control calculation is called *one-step lookahead minimization*. Its main use is not so much in the context of exact DP, but rather in the context of approximate DP methods that involve approximation in value space. There, approximations  $\tilde{J}_k$  are used in place of  $J_k^*$ , similar to the deterministic case; cf. Eqs. (1.7) and (1.8).

#### Approximation in Value Space - Use of $\tilde{J}_k$ in Place of $J_k^*$

At any state  $x_k$  encountered at stage  $k$ , compute and apply the control

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E \left\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right\}. \quad (1.11)$$

As in deterministic problems, the motivation for approximation in value space is that the DP algorithm can be very time-consuming. The one-step lookahead minimization (1.11) needs to be performed only for the  $N$  states  $x_0, \dots, x_{N-1}$  that are encountered during the on-line control of the system, and not for every state within the potentially enormous state space. Of course this simplification entails the loss of optimality, and requires the construction of suitable approximate cost-to-go functions  $\tilde{J}_k$ . This is a major focal point of the RL methodology, and will be discussed at length in the following chapters.

#### Q-Factors for Stochastic Problems

We can define optimal Q-factors for a stochastic problem, similar to the

---

<sup>†</sup> There are some technical/mathematical difficulties here, having to do with the expected value operation in Eq. (1.10) being well-defined and finite. These difficulties are of no concern in practice, and disappear completely when the disturbance spaces  $w_k$  can take only a finite number of values, in which case all expected values consist of sums of finitely many real number terms. For a mathematical treatment, see the relevant discussion in Chapter 1 of [Ber17] and the book [BeS78].

case of deterministic problems [cf. Eq. (1.9)], as the expressions that are minimized in the right-hand side of the stochastic DP equation (1.10). They are given by

$$Q_k^*(x_k, u_k) = E \left\{ g_k(x_k, u_k, w_k) + J_{k+1}^*(f_k(x_k, u_k, w_k)) \right\}.$$

The optimal cost-to-go functions  $J_k^*$  can be recovered from the optimal Q-factors  $Q_k^*$  by means of

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k),$$

and the DP algorithm can be written in terms of Q-factors as

$$Q_k^*(x_k, u_k) = E \left\{ g_k(x_k, u_k, w_k) + \min_{u_{k+1} \in U_{k+1}(f_k(x_k, u_k, w_k))} Q_{k+1}^*(f_k(x_k, u_k, w_k), u_{k+1}) \right\}.$$

### 1.3 EXAMPLES, VARIATIONS, AND SIMPLIFICATIONS

In this section we provide some examples to illustrate problem formulation techniques, solution methods, and adaptations of the basic DP algorithm to various contexts. As a guide for formulating optimal control problems in a manner that is suitable for DP solution, the following two-stage process is suggested:

- (a) Identify the controls/decisions  $u_k$  and the times  $k$  at which these controls are applied. Usually this step is fairly straightforward. However, in some cases there may be some choices to make. For example in deterministic problems, where the objective is to select an optimal sequence of controls  $\{u_0, \dots, u_{N-1}\}$ , one may lump multiple controls to be chosen together, e.g., view the pair  $(u_0, u_1)$  as a single choice. This is usually not possible in stochastic problems, where distinct decisions are differentiated by the information/feedback available when making them.
- (b) Select the states  $x_k$ . The basic guideline here is that  $x_k$  should encompass all the information that is known to the controller at time  $k$  and can be used with advantage in choosing  $u_k$ . In effect, at time  $k$  the state  $x_k$  should separate the past from the future, in the sense that anything that has happened in the past (states, controls, and disturbances from stages prior to stage  $k$ ) is irrelevant to the choices of future controls as long we know  $x_k$ . Sometimes this is described

by saying that the state should have a “Markov property” to express an analogy with states of Markov chains, where (by definition) the conditional probability distribution of future states depends on the past history of the chain only through the present state.

Note that there may be multiple possibilities for selecting the states, because information may be packaged in several different ways that are equally useful from the point of view of control. It is thus worth considering alternative ways to choose the states; for example try to use states that minimize the dimensionality of the state space. For a trivial example that illustrates the point, if a quantity  $x_k$  qualifies as state, then  $(x_{k-1}, x_k)$  also qualifies as state, since  $(x_{k-1}, x_k)$  contains all the information contained within  $x_k$  that can be useful to the controller when selecting  $u_k$ . However, using  $(x_{k-1}, x_k)$  in place of  $x_k$ , gains nothing in terms of optimal cost while complicating the DP algorithm that would have to be executed over a larger space. The concept of a *sufficient statistic*, which refers to a quantity that summarizes all the essential content of the information available to the controller, may be useful in reducing the size of the state space (see the discussion in Section 3.1.1, and in [Ber17], Section 4.3). Section 1.3.6 provides an example, and Section 3.1.1 contains further discussion.

Generally minimizing the dimension of the state makes sense but there are exceptions. A case in point is problems involving *partial* or *imperfect* state information, where we collect measurements to use for control of some quantity of interest  $y_k$  that evolves over time (for example,  $y_k$  may be the position/velocity vector of a moving vehicle). If  $I_k$  is the collection of all measurements and controls up to time  $k$ , it is correct to use  $I_k$  as state. However, a better alternative may be to use as state the conditional probability distribution  $P_k(y_k | I_k)$ , called *belief state*, which may subsume all the information that is useful for the purposes of choosing a control. On the other hand, the belief state  $P_k(y_k | I_k)$  is an infinite-dimensional object, whereas  $I_k$  may be finite dimensional, so the best choice may be problem-dependent; see the textbooks [Ber17] and [Kri16] for further discussion of partial state information problems.

We refer to DP textbooks for extensive additional discussions of modeling and problem formulation techniques. The subsequent chapters do not rely substantially on the material of the present section, so the reader may selectively skip forward to the next chapter and return to this material later as needed.

### 1.3.1 Deterministic Shortest Path Problems

Consider a directed graph with a special node, called the *destination*. Let  $\{1, 2, \dots, N, t\}$  be the set of nodes of the graph, where  $t$  is the destination, and let  $a_{ij}$  be the cost of moving from node  $i$  to node  $j$  [also referred to as the *length* of the directed arc  $(i, j)$  that joins  $i$  and  $j$ ]. By a path we mean a sequence of arcs such that the end node of each arc in the sequence is

the start node of the next arc. The length of a path from a given node to another node is the sum of the lengths of the arcs on the path. We want to find a shortest (i.e., minimum length) path from each node  $i$  to node  $t$ .

We make an assumption relating to cycles, i.e., paths of the form  $(i, j_1), (j_1, j_2), \dots, (j_k, i)$  that start and end at the same node. In particular, we exclude the possibility that a cycle has negative total length. Otherwise, it would be possible to decrease the length of some paths to arbitrarily small values simply by adding more and more negative-length cycles. We thus assume that *all cycles have nonnegative length*. With this assumption, it is clear that an optimal path need not take more than  $N$  moves, so we may limit the number of moves to  $N$ . To conform to the  $N$ -stage DP framework, we formulate the problem as one where *we require exactly  $N$  moves but allow degenerate moves from a node  $i$  to itself with cost  $a_{ii} = 0$* . We also assume that *for every node  $i$  there exists at least one path from  $i$  to  $t$* , so that the problem has at least one solution.

We can formulate this problem as a deterministic DP problem with  $N$  stages, where the states at any stage  $0, \dots, N-1$  are the nodes  $\{1, \dots, N\}$ , the destination  $t$  is the unique state at stage  $N$ , and the controls correspond to the arcs  $(i, j)$ , including the self arcs  $(i, i)$ . Thus at each state  $i$  we select a control  $(i, j)$  and move to state  $j$  at cost  $a_{ij}$ .

We can write the DP algorithm for our problem, with the optimal cost-to-go functions  $J_k^*$  having the meaning

$$J_k^*(i) = \text{optimal cost of getting from } i \text{ to } t \text{ in } N - k \text{ moves,}$$

for  $i = 1, \dots, N$  and  $k = 0, \dots, N-1$ . The cost of the optimal path from  $i$  to  $t$  is  $J_0^*(i)$ . The DP algorithm takes the intuitively clear form

$$\begin{aligned} & \text{optimal cost from } i \text{ to } t \text{ in } N - k \text{ moves} \\ &= \min_{\text{All arcs } (i,j)} [a_{ij} + (\text{optimal cost from } j \text{ to } t \text{ in } N - k - 1 \text{ moves})], \end{aligned}$$

or

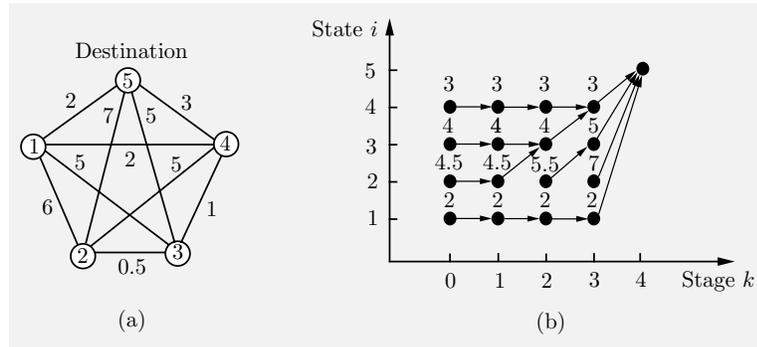
$$J_k^*(i) = \min_{\text{All arcs } (i,j)} [a_{ij} + J_{k+1}^*(j)], \quad k = 0, 1, \dots, N-2,$$

with

$$J_{N-1}^*(i) = a_{it}, \quad i = 1, 2, \dots, N.$$

This algorithm is also known as the *Bellman-Ford algorithm* for shortest paths.

The optimal policy when at node  $i$  after  $k$  moves is to move to a node  $j^*$  that minimizes  $a_{ij} + J_{k+1}^*(j)$  over all  $j$  such that  $(i, j)$  is an arc. If the optimal path obtained from the algorithm contains degenerate moves from a node to itself, this simply means that the path involves in reality less than  $N$  moves.



**Figure 1.3.1** (a) Shortest path problem data. The destination is node 5. Arc lengths are equal in both directions and are shown along the line segments connecting nodes. (b) Costs-to-go generated by the DP algorithm. The number along stage  $k$  and state  $i$  is  $J_k^*(i)$ . Arrows indicate the optimal moves at each stage and node. The optimal paths that start from nodes 1,2,3,4 are  $1 \rightarrow 5$ ,  $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ ,  $3 \rightarrow 4 \rightarrow 5$ ,  $4 \rightarrow 5$ , respectively.

Note that if for some  $k > 0$ , we have

$$J_k^*(i) = J_{k+1}^*(i), \quad \text{for all } i,$$

then subsequent DP iterations will not change the values of the cost-to-go [ $J_{k-m}^*(i) = J_k^*(i)$  for all  $m > 0$  and  $i$ ], so the algorithm can be terminated with  $J_k^*(i)$  being the shortest distance from  $i$  to  $t$ , for all  $i$ .

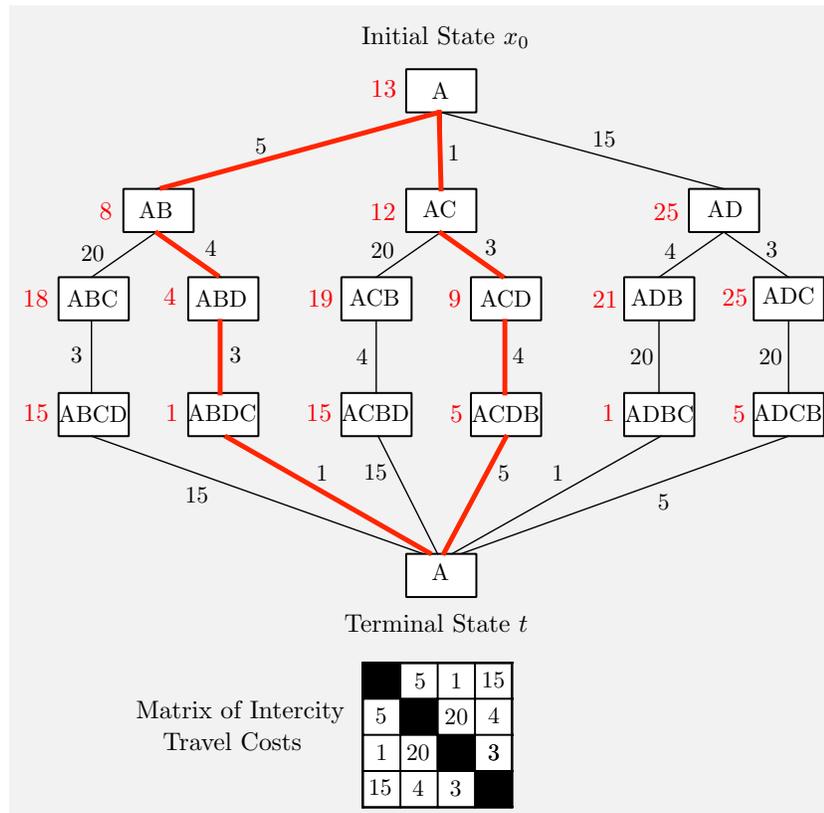
To demonstrate the algorithm, consider the problem shown in Fig. 1.3.1(a) where the costs  $a_{ij}$  with  $i \neq j$  are shown along the connecting line segments (we assume that  $a_{ij} = a_{ji}$ ). Figure 1.3.1(b) shows the optimal cost-to-go  $J_k^*(i)$  at each  $i$  and  $k$  together with the optimal paths.

### 1.3.2 Discrete Deterministic Optimization

Discrete optimization problems can be typically formulated as DP problems by breaking down each feasible solution into a sequence of decisions/controls; see e.g., the scheduling Example 1.1.1. This formulation will often lead to an intractable DP computation because of an exponential explosion of the number of states. However, it brings to bear approximate DP methods, such as rollout and others that we will discuss in future chapters. We illustrate the reformulation by an example and then generalize.

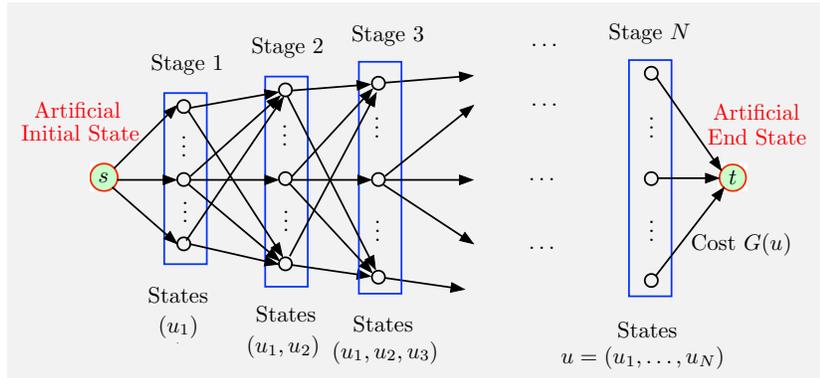
#### Example 1.3.1 (The Traveling Salesman Problem)

An important model for scheduling a sequence of operations is the classical traveling salesman problem. Here we are given  $N$  cities and the travel time between each pair of cities. We wish to find a minimum time travel that visits



**Figure 1.3.2** Example of a DP formulation of the traveling salesman problem. The travel times between the four cities A, B, C, and D are shown in the matrix at the bottom. We form a graph whose nodes are the  $k$ -city sequences and correspond to the states of the  $k$ th stage. The transition costs/travel times are shown next to the arcs. The optimal costs-to-go are generated by DP starting from the terminal state and going backwards towards the initial state, and are shown next to the nodes. There are two optimal sequences here (ABDCA and ACDBA), and they are marked with thick lines. Both optimal sequences can be obtained by forward minimization [cf. Eq. (1.7)], starting from the initial state  $x_0$ .

each of the cities exactly once and returns to the start city. To convert this problem to a DP problem, we form a graph whose nodes are the sequences of  $k$  distinct cities, where  $k = 1, \dots, N$ . The  $k$ -city sequences correspond to the states of the  $k$ th stage. The initial state  $x_0$  consists of some city, taken as the start (city A in the example of Fig. 1.3.2). A  $k$ -city node/state leads to a  $(k + 1)$ -city node/state by adding a new city at a cost equal to the travel time between the last two of the  $k + 1$  cities; see Fig. 1.3.2. Each sequence of  $N$  cities is connected to an artificial terminal node  $t$  with an arc of cost equal to the travel time from the last city of the sequence to the starting city, thus



**Figure 1.3.3.** Formulation of a discrete optimization problem as a DP problem with  $N + 1$  stages. There is a cost  $G(u)$  only at the terminal stage on the arc connecting an  $N$ -solution  $u = (u_1, \dots, u_N)$  to the artificial terminal state. Alternative formulations may use fewer states by taking advantage of the problem's structure.

completing the transformation to a DP problem.

The optimal costs-to-go from each node to the terminal state can be obtained by the DP algorithm and are shown next to the nodes. Note, however, that the number of nodes grows exponentially with the number of cities  $N$ . This makes the DP solution intractable for large  $N$ . As a result, large traveling salesman and related scheduling problems are typically addressed with approximation methods, some of which are based on DP, and will be discussed as part of our subsequent development.

Let us now extend the ideas of the preceding example to the general discrete optimization problem:

$$\begin{aligned} & \text{minimize } G(u) \\ & \text{subject to } u \in U, \end{aligned}$$

where  $U$  is a finite set of feasible solutions and  $G(u)$  is a cost function. We assume that each solution  $u$  has  $N$  components; i.e., it has the form  $u = (u_1, \dots, u_N)$ , where  $N$  is a positive integer. We can then view the problem as a sequential decision problem, where the components  $u_1, \dots, u_N$  are selected one-at-a-time. A  $k$ -tuple  $(u_1, \dots, u_k)$  consisting of the first  $k$  components of a solution is called a  $k$ -solution. We associate  $k$ -solutions with the  $k$ th stage of the finite horizon DP problem shown in Fig. 1.3.3. In particular, for  $k = 1, \dots, N$ , we view as the states of the  $k$ th stage all the  $k$ -tuples  $(u_1, \dots, u_k)$ . The initial state is an artificial state denoted  $s$ . From this state we may move to any state  $(u_1)$ , with  $u_1$  belonging to the set

$$U_1 = \{\tilde{u}_1 \mid \text{there exists a solution of the form } (\tilde{u}_1, \tilde{u}_2, \dots, \tilde{u}_N) \in U\}.$$

Thus  $U_1$  is the set of choices of  $u_1$  that are consistent with feasibility.

More generally, from a state  $(u_1, \dots, u_k)$ , we may move to any state of the form  $(u_1, \dots, u_k, u_{k+1})$ , with  $u_{k+1}$  belonging to the set

$$U_{k+1}(u_1, \dots, u_k) = \{\tilde{u}_{k+1} \mid \text{there exists a solution of the form} \\ (u_1, \dots, u_k, \tilde{u}_{k+1}, \dots, \tilde{u}_N) \in U\}.$$

At state  $(u_1, \dots, u_k)$  we must choose  $u_{k+1}$  from the set  $U_{k+1}(u_1, \dots, u_k)$ . These are the choices of  $u_{k+1}$  that are consistent with the preceding choices  $u_1, \dots, u_k$ , and are also consistent with feasibility. The last stage corresponds to the  $N$ -solutions  $u = (u_1, \dots, u_N)$ , which in turn lead to an artificial end state  $t$ . The cost-to-go from each  $u$  in the last stage to  $t$  is  $G(u)$ , the cost of the solution  $u$ ; see Fig. 1.3.3. All other transitions in this DP problem formulation have cost 0.

Let  $J_k^*(u_1, \dots, u_k)$  denote the optimal cost starting from the  $k$ -solution  $(u_1, \dots, u_k)$ , i.e., the optimal cost of the problem over solutions whose first  $k$  components are constrained to be equal to  $u_i$ ,  $i = 1, \dots, k$ , respectively. The DP algorithm is described by the equation

$$J_k^*(u_1, \dots, u_k) = \min_{u_{k+1} \in U_{k+1}(u_1, \dots, u_k)} J_{k+1}^*(u_1, \dots, u_k, u_{k+1}), \quad (1.12)$$

with the terminal condition

$$J_N^*(u_1, \dots, u_N) = G(u_1, \dots, u_N).$$

The algorithm (1.12) executes backwards in time: starting with the known function  $J_N^* = G$ , we compute  $J_{N-1}^*$ , then  $J_{N-2}^*$ , and so on up to computing  $J_1^*$ . An optimal solution  $(u_1^*, \dots, u_N^*)$  is then constructed by going forward through the algorithm

$$u_{k+1}^* \in \arg \min_{u_{k+1} \in U_{k+1}(u_1^*, \dots, u_k^*)} J_{k+1}^*(u_1^*, \dots, u_k^*, u_{k+1}), \quad k = 0, \dots, N-1, \quad (1.13)$$

first compute  $u_1^*$ , then  $u_2^*$ , and so on up to  $u_N^*$ ; cf. Eq. (1.7).

Of course here the number of states typically grows exponentially with  $N$ , but we can use the DP minimization (1.13) as a starting point for the use of approximation methods. For example we may try to use approximation in value space, whereby we replace  $J_{k+1}^*$  with some suboptimal  $\tilde{J}_{k+1}$  in Eq. (1.13). One possibility is to use as

$$\tilde{J}_{k+1}(u_1^*, \dots, u_k^*, u_{k+1}),$$

the cost generated by a heuristic method that solves the problem suboptimally with the values of the first  $k+1$  decision components fixed at  $u_1^*, \dots, u_k^*, u_{k+1}$ . This is called a *rollout algorithm*, and it is a very simple and effective approach for approximate combinatorial optimization. It will

be discussed later in this book, in Chapter 2 for finite horizon stochastic problems, and in Chapter 5 for infinite horizon problems, where it will be related to the method of policy iteration.

Finally, let us mention that shortest path and discrete optimization problems with a sequential character can be addressed by a variety of approximate shortest path methods. These include the so called *label correcting*, *A\**, and *branch and bound methods* for which extensive discussions can be found in the literature. The author's DP textbook [Ber17] (Chapter 2) contains a substantial account, which connects with the material of this section, as well as with a much more detailed discussion of shortest path methods in the author's network optimization textbook [Ber98].

### 1.3.3 Problems with a Termination State

Many DP problems of interest involve a *termination state*, i.e., a state  $t$  that is cost-free and absorbing in the sense that for all  $k$ ,

$$g_k(t, u_k, w_k) = 0, \quad f_k(t, u_k, w_k) = t, \quad \text{for all } w_k \text{ and } u_k \in U_k(t).$$

Thus the control process essentially terminates upon reaching  $t$ , even if this happens before the end of the horizon. One may reach  $t$  by choice if a special stopping decision is available, or by means of a random transition from another state.

Generally, when it is known that an optimal policy will reach the termination state within at most some given number of stages  $N$ , the DP problem can be formulated as an  $N$ -stage horizon problem.<sup>†</sup> The reason is that even if the termination state  $t$  is reached at a time  $k < N$ , we can extend our stay at  $t$  for an additional  $N - k$  stages at no additional cost. An illustration of this was given in the deterministic shortest path problem that we discussed in Section 1.3.1.

Discrete deterministic optimization problems generally have a close connection to shortest path problems as we have seen in Section 1.3.2. In the problem discussed in that section, the termination state is reached after exactly  $N$  stages (cf. Fig. 1.3.3), but in other problems it is possible that termination can happen earlier. The following well known puzzle is an example.

#### Example 1.3.2 (The Four Queens Problem)

Four queens must be placed on a  $4 \times 4$  portion of a chessboard so that no queen can attack another. In other words, the placement must be such that every row, column, or diagonal of the  $4 \times 4$  board contains at most one queen.

---

<sup>†</sup> When an upper bound on the number of stages to termination is not known, the problem must be formulated as an infinite horizon problem, as will be discussed in a subsequent chapter.

Equivalently, we can view the problem as a sequence of problems; first, placing a queen in one of the first two squares in the top row, then placing another queen in the second row so that it is not attacked by the first, and similarly placing the third and fourth queens. (It is sufficient to consider only the first two squares of the top row, since the other two squares lead to symmetric positions; this is an example of a situation where we have a choice between several possible state spaces, but we select the one that is smallest.)

We can associate positions with nodes of an acyclic graph where the root node  $s$  corresponds to the position with no queens and the terminal nodes correspond to the positions where no additional queens can be placed without some queen attacking another. Let us connect each terminal position with an artificial terminal node  $t$  by means of an arc. Let us also assign to all arcs cost zero except for the artificial arcs connecting terminal positions with less than four queens with the artificial node  $t$ . These latter arcs are assigned a cost of 1 (see Fig. 1.3.4) to express the fact that they correspond to dead-end positions that cannot lead to a solution. Then, the four queens problem reduces to finding a minimal cost path from node  $s$  to node  $t$ , with an optimal sequence of queen placements corresponding to cost 0.

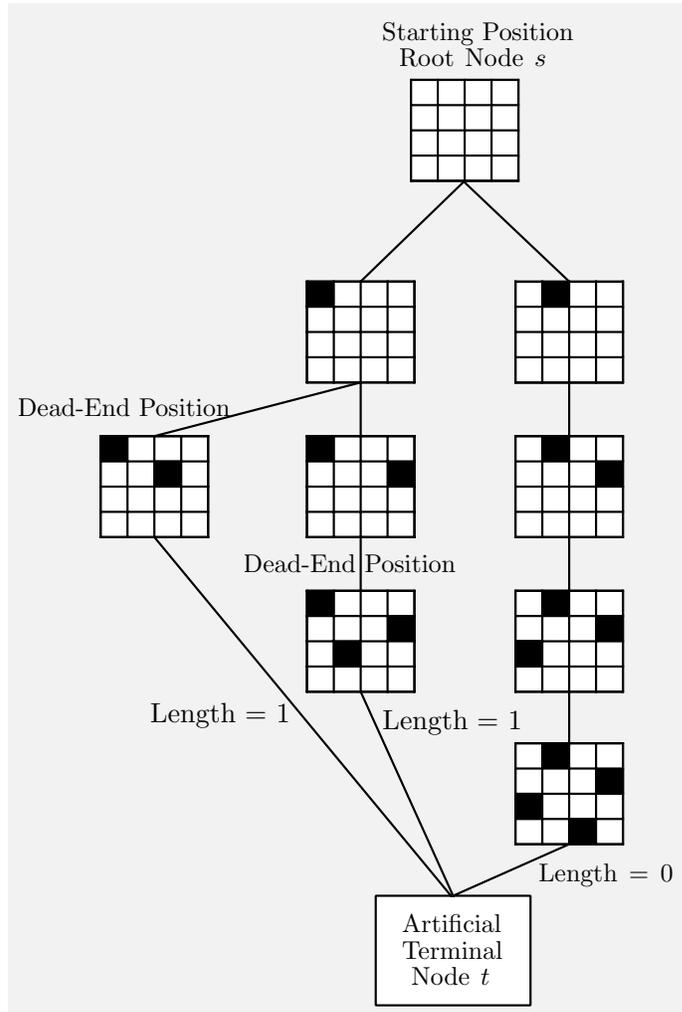
Note that once the states/nodes of the graph are enumerated, the problem is essentially solved. In this  $4 \times 4$  problem the states are few and can be easily enumerated. However, we can think of similar problems with much larger state spaces. For example consider the problem of placing  $N$  queens on an  $N \times N$  board without any queen attacking another. Even for moderate values of  $N$ , the state space for this problem can be extremely large (for  $N = 8$  the number of possible placements with exactly one queen in each row is  $8^8 = 16,777,216$ ). It can be shown that there exist solutions to the  $N$  queens problem for all  $N \geq 4$  (for  $N = 2$  and  $N = 3$ , clearly there is no solution).

There are also several variants of the  $N$  queens problem. For example finding the minimal number of queens that can be placed on an  $N \times N$  board so that they either occupy or attack every square; this is known as the *queen domination problem*. The minimal number can be found in principle by DP, and it is known for some  $N$  (for example the minimal number is 5 for  $N = 8$ ), but not for all  $N$  (see e.g., the paper by Fernau [Fer10]).

### 1.3.4 Forecasts

Consider a situation where at time  $k$  the controller has access to a forecast  $y_k$  that results in a reassessment of the probability distribution of the subsequent disturbance  $w_k$  and, possibly, future disturbances. For example,  $y_k$  may be an exact prediction of  $w_k$  or an exact prediction that the probability distribution of  $w_k$  is a specific one out of a finite collection of distributions. Forecasts of interest in practice are, for example, probabilistic predictions on the state of the weather, the interest rate for money, and the demand for inventory.

Generally, forecasts can be handled by introducing additional states



**Figure 1.3.4** Discrete optimization formulation of the four queens problem. Symmetric positions resulting from placing a queen in one of the rightmost squares in the top row have been ignored. Squares containing a queen have been darkened. All arcs have length zero except for those connecting dead-end positions to the artificial terminal node.

corresponding to the information that the forecasts provide.† We will il-

---

† The device of introducing additional states to modify a given problem so that it fits the DP formalism is known as *state augmentation*. It finds extensive use in the DP formulation of practical problems (see e.g. [Ber17]). Examples of problem reformulations that are based on state augmentation arise when

illustrate the process with a simple example.

Assume that at the beginning of each stage  $k$ , the controller receives an accurate prediction that the next disturbance  $w_k$  will be selected according to a particular probability distribution out of a given collection of distributions  $\{P_1, \dots, P_m\}$ ; i.e., if the forecast is  $i$ , then  $w_k$  is selected according to  $P_i$ . The a priori probability that the forecast will be  $i$  is denoted by  $p_i$  and is given.

The forecasting process can be represented by means of the equation

$$y_{k+1} = \xi_k,$$

where  $y_{k+1}$  can take the values  $1, \dots, m$ , corresponding to the  $m$  possible forecasts, and  $\xi_k$  is a random variable taking the value  $i$  with probability  $p_i$ . The interpretation here is that when  $\xi_k$  takes the value  $i$ , then  $w_{k+1}$  will occur according to the distribution  $P_i$ .

By combining the system equation with the forecast equation  $y_{k+1} = \xi_k$ , we obtain an augmented system given by

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, u_k, w_k) \\ \xi_k \end{pmatrix}.$$

The new state is

$$\tilde{x}_k = (x_k, y_k).$$

The new disturbance is

$$\tilde{w}_k = (w_k, \xi_k),$$

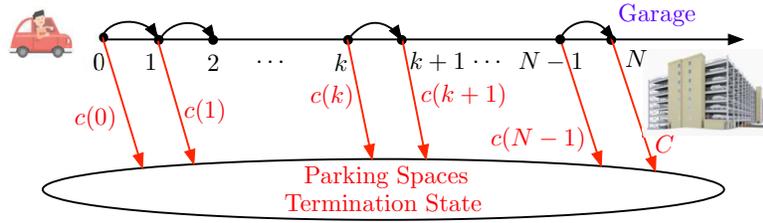
and its probability distribution is determined by the distributions  $P_i$  and the probabilities  $p_i$ , and depends explicitly on  $\tilde{x}_k$  (via  $y_k$ ) but not on the prior disturbances.

Thus, by suitable reformulation of the cost, the problem can be cast as a stochastic DP problem. Note that the control applied depends on both the current state and the current forecast. The DP algorithm takes the form

$$\begin{aligned} J_N^*(x_N, y_N) &= g_N(x_N), \\ J_k^*(x_k, y_k) &= \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ g_k(x_k, u_k, w_k) \right. \\ &\quad \left. + \sum_{i=1}^m p_i J_{k+1}^*(f_k(x_k, u_k, w_k), i) \mid y_k \right\}, \end{aligned} \quad (1.14)$$

---

the disturbances exhibit correlations over time. They also arise in the presence of *post-decision* states, where the system function  $f_k(x_k, u_k, w_k)$  has the form  $\tilde{f}_k(h_k(x_k, u_k), w_k)$ , where  $\tilde{f}_k$  is some function, and  $h_k(x_k, u_k)$  represents an intermediate “state” that occurs after the control is applied. With post-decision states, the DP algorithm may be reformulated to a simpler form (see [Ber12], Section 6.1.5).



**Figure 1.3.5** Cost structure of the parking problem. The driver may park at space  $k = 0, 1, \dots, N - 1$  at cost  $c(k)$ , if the space is free, or continue to the next space  $k + 1$  at no cost. At space  $N$  (the garage) the driver must park at cost  $C$ .

where  $y_k$  may take the values  $1, \dots, m$ , and the expectation over  $w_k$  is taken with respect to the distribution  $P_{y_k}$ .

It should be clear that the preceding formulation admits several extensions. One example is the case where forecasts can be influenced by the control action (e.g., pay extra for a more accurate forecast) and involve several future disturbances. However, the price for these extensions is increased complexity of the corresponding DP algorithm.

### 1.3.5 Problems with Uncontrollable State Components

In many problems of interest the natural state of the problem consists of several components, some of which cannot be affected by the choice of control. In such cases the DP algorithm can be simplified considerably, and be executed over the controllable components of the state. Before describing how this can be done in generality, let us consider an example.

#### Example 1.3.3 (Parking)

A driver is looking for inexpensive parking on the way to his destination. The parking area contains  $N$  spaces, numbered  $0, \dots, N - 1$ , and a garage following space  $N - 1$ . The driver starts at space 0 and traverses the parking spaces sequentially, i.e., from space  $k$  he goes next to space  $k + 1$ , etc. Each parking space  $k$  costs  $c(k)$  and is free with probability  $p(k)$  independently of whether other parking spaces are free or not. If the driver reaches the last parking space  $N - 1$  and does not park there, he must park at the garage, which costs  $C$ . The driver can observe whether a parking space is free only when he reaches it, and then, if it is free, he makes a decision to park in that space or not to park and check the next space. The problem is to find the minimum expected cost parking policy.

We formulate the problem as a DP problem with  $N$  stages, corresponding to the parking spaces, and an artificial termination state  $t$  that corresponds to having parked; see Fig. 1.3.5. At each stage  $k = 1, \dots, N - 1$ , we have three states: the artificial termination state  $t$ , and the two states  $(k, F)$  and  $(k, \bar{F})$ , corresponding to space  $k$  being free or taken, respectively.

At stage 0, we have only two states,  $(0, F)$  and  $(0, \overline{F})$ , and at the final stage there is only one state, the termination state  $t$ . The decision/control is to park or continue at state  $(k, F)$  [there is no choice at states  $(k, \overline{F})$  and states  $(k, t)$ ,  $k = 1, \dots, N - 1$ ]. The termination state  $t$  is reached at cost  $c(k)$  when a parking decision is made at the states  $(k, F)$ ,  $k = 0, \dots, N - 1$ , at cost  $C$ , when the driver continues at states  $(N - 1, F)$  or  $(N - 1, \overline{F})$ , and at no cost at  $(k, t)$ ,  $k = 0, \dots, N - 1$ .

Let us now derive the form of the DP algorithm, denoting:

$J_k^*(F)$ : The optimal cost-to-go upon arrival at a space  $k$  that is free.

$J_k^*(\overline{F})$ : The optimal cost-to-go upon arrival at a space  $k$  that is taken.

$J_k^*(t)$ : The cost-to-go of the “parked”/termination state  $t$ .

The DP algorithm for  $k = 0, \dots, N - 1$  takes the form

$$J_k^*(F) = \begin{cases} \min [c(k), p(k+1)J_{k+1}^*(F) + (1-p(k+1))J_{k+1}^*(\overline{F})] & \text{if } k < N - 1, \\ \min [c(N-1), C] & \text{if } k = N - 1, \end{cases}$$

$$J_k^*(\overline{F}) = \begin{cases} p(k+1)J_{k+1}^*(F) + (1-p(k+1))J_{k+1}^*(\overline{F}) & \text{if } k < N - 1, \\ C & \text{if } k = N - 1, \end{cases}$$

for the states other than the termination state  $t$ , while for  $t$  we have

$$J_k^*(t) = 0, \quad k = 1, \dots, N.$$

While this algorithm is easily executed, it can be written in a simpler and equivalent form, which takes advantage of the fact that the second component ( $F$  or  $\overline{F}$ ) of the state is uncontrollable. This can be done by introducing the scalars

$$\hat{J}_k = p(k)J_k^*(F) + (1-p(k))J_k^*(\overline{F}), \quad k = 0, \dots, N - 1,$$

which can be viewed as the optimal expected cost-to-go upon arriving at space  $k$  but *before verifying its free or taken status*.

Indeed, from the preceding DP algorithm, we have

$$\hat{J}_{N-1} = p(N-1) \min [c(N-1), C] + (1-p(N-1))C,$$

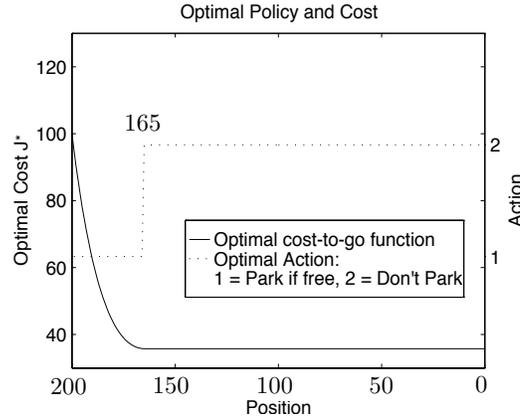
$$\hat{J}_k = p(k) \min [c(k), \hat{J}_{k+1}] + (1-p(k))\hat{J}_{k+1}, \quad k = 0, \dots, N - 2.$$

From this algorithm we can also obtain the optimal parking policy, which is to park at space  $k = 0, \dots, N - 1$  if it is free and we have  $c(k) \leq \hat{J}_{k+1}$ .

Figure 1.3.6 provides a plot for  $\hat{J}_k$  for the case where

$$p(k) \equiv 0.05, \quad c(k) = N - k, \quad C = 100, \quad N = 200. \quad (1.15)$$

The optimal policy is to travel to space 165 and then to park at the first available space. The reader may verify that this type of policy, characterized by a single threshold distance, is optimal not just for the form of  $c(k)$  given



**Figure 1.3.6** Optimal cost-to-go and optimal policy for the parking problem with the data in Eq. (1.15). The optimal policy is to travel from space 0 to space 165 and then to park at the first available space.

above, but also for any form of  $c(k)$  that is monotonically decreasing as  $k$  increases.

We will now formalize the procedure illustrated in the preceding example. Let the state of the system be a composite  $(x_k, y_k)$  of two components  $x_k$  and  $y_k$ . The evolution of the main component,  $x_k$ , is affected by the control  $u_k$  according to the equation

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

where the distribution  $P_k(w_k | x_k, y_k, u_k)$  is given. The evolution of the other component,  $y_k$ , is governed by a given conditional distribution  $P_k(y_k | x_k)$  and cannot be affected by the control, except indirectly through  $x_k$ . One is tempted to view  $y_k$  as a disturbance, but there is a difference:  $y_k$  is observed by the controller before applying  $u_k$ , while  $w_k$  occurs after  $u_k$  is applied, and indeed  $w_k$  may probabilistically depend on  $u_k$ .

It turns out that we can formulate a DP algorithm that is executed over the controllable component of the state, with the dependence on the uncontrollable component being “averaged out” as in the preceding example. In particular, let  $J_k^*(x_k, y_k)$  denote the optimal cost-to-go at stage  $k$  and state  $(x_k, y_k)$ , and define

$$\hat{J}_k(x_k) = E_{y_k} \{ J_k^*(x_k, y_k) | x_k \}.$$

Then, similar to the preceding parking example, a DP algorithm that gen-

erates  $\hat{J}_k(x_k)$  can be obtained, and has the following form:†

$$\hat{J}_k(x_k) = E_{y_k} \left\{ \min_{u_k \in U_k(x_k, y_k)} E_{w_k} \left\{ g_k(x_k, y_k, u_k, w_k) + \hat{J}_{k+1}(f_k(x_k, y_k, u_k, w_k)) \mid x_k, y_k, u_k \right\} \mid x_k \right\}. \quad (1.16)$$

Note that the minimization in the right-hand side of the preceding equation must still be performed for all values of the full state  $(x_k, y_k)$  in order to yield an optimal control law as a function of  $(x_k, y_k)$ . Nonetheless, the equivalent DP algorithm (1.16) has the advantage that it is executed over a significantly reduced state space. Later, when we consider approximation in value space, we will find that it is often more convenient to approximate  $\hat{J}_k(x_k)$  than to approximate  $J_k^*(x_k, y_k)$ ; see the following discussions of forecasts and of the game of tetris.

As an example, consider the augmented state resulting from the incorporation of forecasts, as described earlier in Section 1.3.4. Then, the forecast  $y_k$  represents an uncontrolled state component, so that the DP algorithm can be simplified as in Eq. (1.16). In particular, using the notation of Section 1.3.4, by defining

$$\hat{J}_k(x_k) = \sum_{i=1}^m p_i J_k^*(x_k, i), \quad k = 0, 1, \dots, N-1,$$

and

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

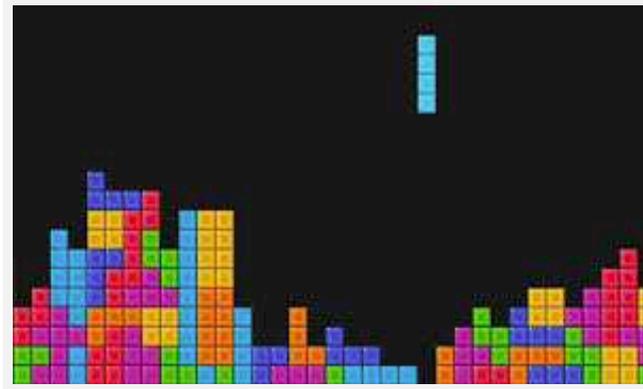
we have, using Eq. (1.14),

$$\hat{J}_k(x_k) = \sum_{i=1}^m p_i \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ g_k(x_k, u_k, w_k) + \hat{J}_{k+1}(f_k(x_k, u_k, w_k)) \mid y_k = i \right\},$$

---

† This is a consequence of the calculation

$$\begin{aligned} \hat{J}_k(x_k) &= E_{y_k} \left\{ J_k^*(x_k, y_k) \mid x_k \right\} \\ &= E_{y_k} \left\{ \min_{u_k \in U_k(x_k, y_k)} E_{w_k, x_{k+1}, y_{k+1}} \left\{ g_k(x_k, y_k, u_k, w_k) \right. \right. \\ &\quad \left. \left. + J_{k+1}^*(x_{k+1}, y_{k+1}) \mid x_k, y_k, u_k \right\} \mid x_k \right\} \\ &= E_{y_k} \left\{ \min_{u_k \in U_k(x_k, y_k)} E_{w_k, x_{k+1}} \left\{ g_k(x_k, y_k, u_k, w_k) \right. \right. \\ &\quad \left. \left. + E_{y_{k+1}} \left\{ J_{k+1}^*(x_{k+1}, y_{k+1}) \mid x_{k+1} \right\} \mid x_k, y_k, u_k \right\} \mid x_k \right\}. \end{aligned}$$



**Figure 1.3.7** Illustration of a tetris board.

which is executed over the space of  $x_k$  rather than  $x_k$  and  $y_k$ . Note that this is a simpler algorithm to approximate than the one of Eq. (1.14).

Uncontrollable state components often occur in arrival systems, such as queueing, where action must be taken in response to a random event (such as a customer arrival) that cannot be influenced by the choice of control. Then the state of the arrival system must be augmented to include the random event, but the DP algorithm can be executed over a smaller space, as per Eq. (1.16). Here is an example of this type.

#### **Example 1.3.4 (Tetris)**

Tetris is a popular video game played on a two-dimensional grid. Each square in the grid can be full or empty, making up a “wall of bricks” with “holes” and a “jagged top” (see Fig. 1.3.7). 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. 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. The falling blocks are generated independently according to some probability distribution, defined over a finite set of standard shapes. The game starts with an empty grid and ends when a square in the top row becomes full and the top of the wall reaches the top of the grid. 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.

We can model the problem of finding an optimal tetris playing strategy as a stochastic DP problem. The control, denoted by  $u$ , is the horizontal positioning and rotation applied to the falling block. The state consists of two components:

- (1) The board position, i.e., a binary description of the full/empty status of each square, denoted by  $x$ .

(2) The shape of the current falling block, denoted by  $y$ .

There is also an additional termination state which is cost-free. Once the state reaches the termination state, it stays there with no change in cost.

The shape  $y$  is generated according to a probability distribution  $p(y)$ , independently of the control, so it can be viewed as an uncontrollable state component. The DP algorithm (1.16) is executed over the space of board positions  $x$  and has the intuitive form

$$\hat{J}_k(x) = \sum_y p(y) \max_u \left[ g(x, y, u) + \hat{J}_{k+1}(f(x, y, u)) \right], \quad \text{for all } x,$$

where

$g(x, y, u)$  is the number of points scored (rows removed),

$f(x, y, u)$  is the next board position (or termination state),

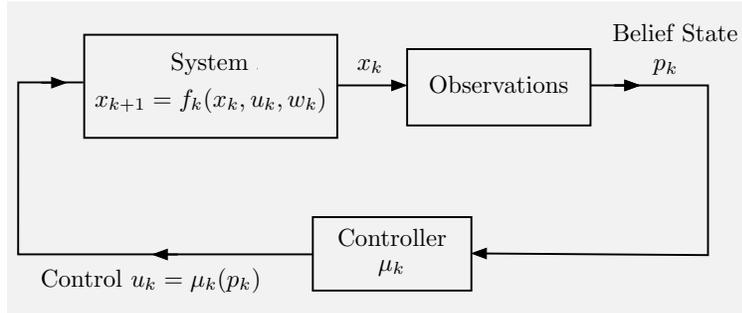
when the state is  $(x, y)$  and control  $u$  is applied, respectively. Note, however, that despite the simplification in the DP algorithm achieved by eliminating the uncontrollable portion of the state, the number of states  $x$  is still enormous, and the problem can only be addressed by suboptimal methods, which will be discussed later in this book.

### 1.3.6 Partial State Information and Belief States

We have assumed so far that the controller has access to the exact value of the current state  $x_k$ , so a policy consists of a sequence of functions  $\mu_k(x_k)$ ,  $k = 0, \dots, N - 1$ . However, in many practical settings this assumption is unrealistic, because some components of the state may be inaccessible for measurement, the sensors used for measuring them may be inaccurate, or the cost of obtaining accurate measurements may be prohibitive.

Often in such situations the controller has access to only some of the components of the current state, and the corresponding measurements may also be corrupted by stochastic uncertainty. For example in three-dimensional motion problems, the state may consist of the six-tuple of position and velocity components, but the measurements may consist of noise-corrupted radar measurements of the three position components. This gives rise to problems of *partial* or *imperfect* state information, which have received a lot of attention in the optimization and artificial intelligence literature (see e.g., [Ber17], [RuN16]). Even though there are DP algorithms for partial information problems, these algorithms are far more computationally intensive than their perfect information counterparts. For this reason, in the absence of an analytical solution, partial information problems are typically solved suboptimally in practice.

On the other hand it turns out that conceptually, partial state information problems are no different than the perfect state information problems we have been addressing so far. In fact by various reformulations, we can reduce a partial state information problem to one with perfect state



**Figure 1.3.8** Schematic illustration of a control system with imperfect state observations. The belief state  $p_k$  is the conditional probability distribution of  $x_k$  given all the observations up to time  $k$ .

information (see [Ber17], Ch. 4). The most common approach is to replace the state  $x_k$  with a *belief state*, which is the probability distribution of  $x_k$  given all the observations that have been obtained by the controller up to time  $k$  (see Fig. 1.3.8). This probability distribution can in principle be computed, and it can serve as “state” in an appropriate DP algorithm. We illustrate this process with a simple example.

#### Example 1.3.5 (Treasure Hunting)

In a classical problem of search, one has to decide at each of  $N$  periods whether to search a site that may contain a treasure. If a treasure is present, the search reveals it with probability  $\beta$ , in which case the treasure is removed from the site. Here the state  $x_k$  has two values: either a treasure is present in the site or it is not. The control  $u_k$  takes two values: search and not search. If the site is searched, we obtain an observation, which takes one of two values: treasure found or not found. If the site is not searched, no information is obtained.

Denote

$p_k$  : probability a treasure is present at the beginning of period  $k$ .

This is the belief state at time  $k$  and it evolves according to the equation

$$p_{k+1} = \begin{cases} p_k & \text{if the site is not searched at time } k, \\ 0 & \text{if the site is searched and a treasure is found,} \\ \frac{p_k(1-\beta)}{p_k(1-\beta)+1-p_k} & \text{if the site is searched but no treasure is found.} \end{cases} \quad (1.17)$$

The third relation above follows by application of Bayes’ rule ( $p_{k+1}$  is equal to the  $k$ th period probability of a treasure being present *and* the search being unsuccessful, divided by the probability of an unsuccessful search). The second relation holds because the treasure is removed after a successful search.

Let us view  $p_k$  as the state of a “belief system” given by Eq. (1.17), and write a DP algorithm, assuming that the treasure’s worth is  $V$ , that each

search costs  $C$ . Denoting by  $J_k^*(p)$  the optimal cost-to-go from belief state  $p$  at time  $k$ , the algorithm takes the form

$$J_k^*(p_k) = \max \left[ J_{k+1}^*(p_k), \right. \\ \left. -C + p_k \beta V + (1 - p_k \beta) J_{k+1}^* \left( \frac{p_k(1 - \beta)}{p_k(1 - \beta) + 1 - p_k} \right) \right], \quad (1.18)$$

with  $J_N^*(p_N) = 0$ . The two options in the maximization of Eq. (1.18) correspond to not searching (in which case  $p_k$  remains unchanged), and searching [in which case  $p_k$  evolves according to Eq. (1.17)].

This DP algorithm can be used to obtain an analytical solution. In particular, it is straightforward to show by induction that the functions  $J_k^*$  satisfy  $J_k^*(p_k) \geq 0$  if  $p_k \in [0, 1]$  and

$$J_k^*(p_k) = 0 \quad \text{if} \quad p_k \leq \frac{C}{\beta V}.$$

From this it follows that it is optimal to search at period  $k$  if and only if

$$\frac{C}{\beta V} \leq p_k.$$

Thus, it is optimal to search if and only if the expected reward from the next search,  $p_k \beta V$ , is greater or equal to the cost  $C$  of the search - a myopic policy that focuses on just the next stage.

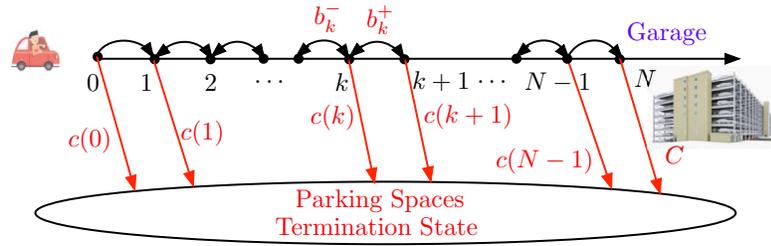
Of course the preceding example is extremely simple, involving a state  $x_k$  that takes just two values. As a result, the belief state  $p_k$  takes values within the interval  $[0, 1]$ . Still there are infinitely many values in this interval, and if a computational solution were necessary, the belief state would have to be discretized and the DP algorithm (1.18) would have to be adapted to the discretization.

In problems where the state  $x_k$  can take a finite but large number of values, say  $n$ , the belief states comprise an  $n$ -dimensional simplex, so discretization becomes problematic. As a result, alternative suboptimal solution methods are often used in partial state information problems. Some of these methods will be described in future chapters.

The following is a simple example of a partial state information problem whose belief state has large enough size to make an exact DP solution impossible.

### Example 1.3.6 (Bidirectional Parking)

Let us consider a more complex version of the parking problem of Example 1.3.3. As in that example, a driver is looking for inexpensive parking on the



**Figure 1.3.9** Cost structure and transitions of the bidirectional parking problem. The driver may park at space  $k = 0, 1, \dots, N - 1$  at cost  $c(k)$ , if the space is free, can move to  $k - 1$  at cost  $b_k^-$  or can move to  $k + 1$  at cost  $b_k^+$ . At space  $N$  (the garage) the driver must park at cost  $C$ .

way to his destination, along a line of  $N$  parking spaces with a garage at the end. The difference is that the driver can move in either direction, rather than just forward towards the garage. In particular, at space  $i$ , the driver can park at cost  $c(i)$  if  $i$  is free, can move to  $i - 1$  at a cost  $b_i^-$  or can move to  $i + 1$  at a cost  $b_i^+$ . Moreover, the driver records the free/taken status of the spaces previously visited and may return to any of these spaces; see Fig. 1.3.9.

Let us assume that the probability  $p(i)$  of a space  $i$  being free changes over time, i.e., a space found free (or taken) at a given visit may get taken (or become free, respectively) by the time of the next visit. The initial probabilities  $p(i)$ , before visiting any spaces, are known, and the mechanism by which these probabilities change over time is also known to the driver. As an example, we may assume that at each time period,  $p(i)$  increases by a certain known factor with some probability  $\xi$  and decreases by another known factor with the complementary probability  $1 - \xi$ .

Here the belief state is the vector of current probabilities

$$(p(0), \dots, p(N)),$$

and it is updated at each time based on the new observation: the free/taken status of the space visited at that time. Thus the belief state can be computed exactly by the driver, given the parking status observations of the spaces visited thus far. While it is possible to state an exact DP algorithm that is defined over the set of belief states, and we will do so later, the algorithm is impossible to execute in practice.† Thus the problem can only be solved with approximations.

---

† The problem as stated is an infinite horizon problem because there is nothing to prevent the driver from moving forever in the parking lot without ever parking. We can convert the problem to a similarly difficult finite horizon problem by restricting the number of moves to a given upper limit  $\bar{N} > N$ , and requiring that if the driver is at distance of  $k$  spaces from the garage at time  $\bar{N} - k$ , then driving in the direction away from the garage is not an option.

### 1.3.7 Linear Quadratic Optimal Control

In a few exceptional special cases the DP algorithm yields an analytical solution, which can be used among other purposes, as a starting point for approximate DP schemes to solve related problems. Prominent among such cases are various linear quadratic optimal control problems, which involve a linear (possibly multidimensional) system, a quadratic cost function, and no constraints on the control. Let us illustrate this with the deterministic scalar linear quadratic Example 1.1.2. We will apply the DP algorithm for the case of just two stages ( $N = 2$ ), and illustrate the method for obtaining a nice analytical solution.

As defined in Example 1.1.2, the terminal cost is

$$g_2(x_2) = r(x_2 - T)^2.$$

Thus the DP algorithm starts with

$$J_2^*(x_2) = g_2(x_2) = r(x_2 - T)^2,$$

[cf. Eq. (1.3)].

For the next-to-last stage, we have [cf. Eq. (1.4)]

$$J_1^*(x_1) = \min_{u_1} [u_1^2 + J_2^*(x_2)] = \min_{u_1} \left[ u_1^2 + J_2^*((1-a)x_1 + au_1) \right].$$

Substituting the previous form of  $J_2^*$ , we obtain

$$J_1^*(x_1) = \min_{u_1} \left[ u_1^2 + r((1-a)x_1 + au_1 - T)^2 \right]. \quad (1.19)$$

This minimization will be done by setting to zero the derivative with respect to  $u_1$ . This yields

$$0 = 2u_1 + 2ra((1-a)x_1 + au_1 - T),$$

and by collecting terms and solving for  $u_1$ , we obtain the optimal temperature for the last oven as a function of  $x_1$ :

$$\mu_1^*(x_1) = \frac{ra(T - (1-a)x_1)}{1 + ra^2}. \quad (1.20)$$

By substituting the optimal  $u_1$  in the expression (1.19) for  $J_1^*$ , we obtain

$$\begin{aligned} J_1^*(x_1) &= \frac{r^2a^2((1-a)x_1 - T)^2}{(1+ra^2)^2} + r \left( (1-a)x_1 + \frac{ra^2(T - (1-a)x_1)}{1+ra^2} - T \right)^2 \\ &= \frac{r^2a^2((1-a)x_1 - T)^2}{(1+ra^2)^2} + r \left( \frac{ra^2}{1+ra^2} - 1 \right)^2 ((1-a)x_1 - T)^2 \\ &= \frac{r((1-a)x_1 - T)^2}{1+ra^2}. \end{aligned}$$

We now go back one stage. We have [cf. Eq. (1.4)]

$$J_0^*(x_0) = \min_{u_0} [u_0^2 + J_1^*(x_1)] = \min_{u_0} \left[ u_0^2 + J_1^*((1-a)x_0 + au_0) \right],$$

and by substituting the expression already obtained for  $J_1^*$ , we have

$$J_0^*(x_0) = \min_{u_0} \left[ u_0^2 + \frac{r((1-a)^2x_0 + (1-a)au_0 - T)^2}{1 + ra^2} \right].$$

We minimize with respect to  $u_0$  by setting the corresponding derivative to zero. We obtain

$$0 = 2u_0 + \frac{2r(1-a)a((1-a)^2x_0 + (1-a)au_0 - T)}{1 + ra^2}.$$

This yields, after some calculation, the optimal temperature of the first oven:

$$\mu_0^*(x_0) = \frac{r(1-a)a(T - (1-a)^2x_0)}{1 + ra^2(1 + (1-a)^2)}. \quad (1.21)$$

The optimal cost is obtained by substituting this expression in the formula for  $J_0^*$ . This leads to a straightforward but lengthy calculation, which in the end yields the rather simple formula

$$J_0^*(x_0) = \frac{r((1-a)^2x_0 - T)^2}{1 + ra^2(1 + (1-a)^2)}.$$

This completes the solution of the problem.

Note that the algorithm has simultaneously yielded an optimal policy  $\{\mu_0^*, \mu_1^*\}$  via Eqs. (1.21) and (1.20): a rule that tells us the optimal oven temperatures  $u_0 = \mu_0^*(x_0)$  and  $u_1 = \mu_1^*(x_1)$  for every possible value of the states  $x_0$  and  $x_1$ , respectively. Thus the DP algorithm solves all the tail subproblems and provides a feedback policy.

A noteworthy feature in this example is the facility with which we obtained an analytical solution. A little thought while tracing the steps of the algorithm will convince the reader that what simplifies the solution is the quadratic nature of the cost and the linearity of the system equation. Indeed, it can be shown in generality that when the system is linear and the cost is quadratic, the optimal policy and cost-to-go function are given by closed-form expressions, regardless of the number of stages  $N$  (see [Ber17], Section 3.1).

### Stochastic Linear Quadratic Problems - Certainty Equivalence

Let us now introduce a zero-mean stochastic additive disturbance in the linear system equation. Remarkably, it turns out that the optimal policy remains unaffected. To see this, assume that the material's temperature evolves according to

$$x_{k+1} = (1 - a)x_k + au_k + w_k, \quad k = 0, 1,$$

where  $w_0$  and  $w_1$  are independent random variables with given distribution, zero mean

$$E\{w_0\} = E\{w_1\} = 0,$$

and finite variance. Then the equation for  $J_1^*$  [cf. Eq. (1.4)] becomes

$$\begin{aligned} J_1^*(x_1) &= \min_{u_1} E_{w_1} \left\{ u_1^2 + r((1 - a)x_1 + au_1 + w_1 - T)^2 \right\} \\ &= \min_{u_1} \left[ u_1^2 + r((1 - a)x_1 + au_1 - T)^2 \right. \\ &\quad \left. + 2rE\{w_1\}((1 - a)x_1 + au_1 - T) + rE\{w_1^2\} \right]. \end{aligned}$$

Since  $E\{w_1\} = 0$ , we obtain

$$J_1^*(x_1) = \min_{u_1} \left[ u_1^2 + r((1 - a)x_1 + au_1 - T)^2 \right] + rE\{w_1^2\}.$$

Comparing this equation with Eq. (1.19), we see that the presence of  $w_1$  has resulted in an additional inconsequential constant term,  $rE\{w_1^2\}$ . Therefore, the optimal policy for the last stage remains unaffected by the presence of  $w_1$ , while  $J_1^*(x_1)$  is increased by  $rE\{w_1^2\}$ . It can be seen that a similar situation also holds for the first stage. In particular, the optimal cost is given by the same expression as before except for an additive constant that depends on  $E\{w_0^2\}$  and  $E\{w_1^2\}$ .

Generally, if the optimal policy is unaffected when the disturbances are replaced by their means, we say that *certainty equivalence* holds. This occurs in several types of problems involving a linear system and a quadratic cost; see [Ber17], Sections 3.1 and 4.2. For other problems, certainty equivalence can be used as a basis for problem approximation, e.g., assume that certainty equivalence holds (i.e., replace stochastic quantities by some typical values, such as their expected values) and apply exact DP to the resulting deterministic optimal control problem (see Section 2.3.2).

#### 1.3.8 Systems with Unknown Parameters - Adaptive Control

We have been dealing so far with systems having a known system equation. In practice, however, there are many cases where the system parameters are either not known exactly or change over time.

As an example consider a car cruise control system. The car's velocity at time  $k$ , denoted by  $x_k$ , evolves according to

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

where  $u_k$  is the force that propels the car forward ( $u_k$  can be related to the pressure applied on the car's gas pedal). However, the coefficient  $b$  changes frequently and cannot be modeled with any precision because it depends on unpredictable time-varying conditions, such as the slope and condition of the road, and the weight of the car (which is affected by the number of passengers). This points to the need of controllers that yield satisfactory performance over a potentially broad range of system parameters.

To construct a formal optimization framework for dealing with such a situation, we may embed the problem within an imperfect state information framework by modeling the unknown parameters as unobservable states. Indeed, let the system equation be of the form

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

where  $\theta$  is a vector of unknown parameters, which for simplicity we assume to be fixed over time. We introduce an additional state variable  $y_k = \theta$  and obtain a system equation of the form

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, w_k) \\ y_k \end{pmatrix}.$$

This equation can be written compactly as

$$\tilde{x}_{k+1} = \tilde{f}_k(\tilde{x}_k, u_k, w_k),$$

where  $\tilde{x}_k = (x_k, y_k)$  is the new state, and  $\tilde{f}_k$  is an appropriate function. The initial state is

$$\tilde{x}_0 = (x_0, \theta).$$

Unfortunately, however, since  $y_k$  (i.e.,  $\theta$ ) is unobservable, the problem is one of partial state information even if the controller knows the state  $x_k$  exactly. This makes the exact solution by DP intractable. To address this situation, several suboptimal solution approaches have been suggested.

An apparently reasonable form of suboptimal control is to separate the control process into two phases, a *parameter estimation (or identification) phase* and a *control phase*. In the first phase the unknown parameters are identified, while the control takes no account of the interim results of estimation. The final parameter estimates from the first phase are then used to implement an optimal control law in the second phase. This alternation of estimation and control phases may be repeated several times during any system run in order to take into account subsequent changes of the parameters.

One drawback of this approach is that it is not always easy to determine when to terminate one phase and start the other. A second difficulty, of a more fundamental nature, is that the unknown parameters must often be estimated as the system is being controlled. Then, unfortunately, the control process may make some of the unknown parameters invisible to the estimation process. This is known as the problem of parameter *identifiability*, which is discussed in the context of optimal control in several sources, including [BoV79] and [Kum83]; see also [Ber17], Section 6.7. For a simple example, consider the scalar system

$$x_{k+1} = ax_k + bu_k, \quad k = 0, \dots, N-1,$$

and the quadratic cost

$$\sum_{k=1}^N (x_k)^2.$$

Assuming perfect state information, if the parameters  $a$  and  $b$  are known, it can be seen that the optimal control law is

$$\mu_k^*(x_k) = -\frac{a}{b}x_k,$$

which sets all future states to 0. Assume now that the parameters  $a$  and  $b$  are unknown, and consider the two-phase method. During the first phase the control law

$$\tilde{\mu}_k(x_k) = \gamma x_k \tag{1.22}$$

is used ( $\gamma$  is some scalar; for example,  $\gamma = -\bar{a}/\bar{b}$ , where  $\bar{a}$  and  $\bar{b}$  are some a priori estimates of  $a$  and  $b$ , respectively). At the end of the first phase, the control law is changed to

$$\bar{\mu}_k(x_k) = -\frac{\hat{a}}{\hat{b}}x_k,$$

where  $\hat{a}$  and  $\hat{b}$  are the estimates obtained from the estimation process. However, with the control law (1.22), the closed-loop system is

$$x_{k+1} = (a + b\gamma)x_k,$$

so the estimation process can at best yield the value of  $(a + b\gamma)$  but not the values of both  $a$  and  $b$ . In other words, the estimation process cannot discriminate between pairs of values  $(a_1, b_1)$  and  $(a_2, b_2)$  such that  $a_1 + b_1\gamma = a_2 + b_2\gamma$ . Therefore,  $a$  and  $b$  are not identifiable when feedback control of the form (1.22) is applied.

The issues discussed above and the methods to address them are part of a broad field known as *adaptive control*, which deals with the design of controllers for systems with unknown parameters. This is a rich subject

with many and diverse applications. We will not discuss adaptive control in this book, and we refer to textbook treatments, such as Aström and Wittenmark [AsW94], Goodwin and Sin [GoS84], Ioannou and Sun [IoS96], Krstic, Kanellakopoulos, and Kokotovic [KKK95], Kumar and Varaiya [KuV86], Sastry and Bodson [SaB11], and Slotine and Li [SIL91].

We note, however, a simple and popular methodology, *PID (Proportional-Integral-Derivative) control*, which can be used for problems involving an unknown or changing mathematical model; see e.g., the books by Aström and Hagglund [AsH95], [AsH06]. In particular, PID control aims to maintain the output of a single-input single-output dynamic system around a set point or to follow a given trajectory, as the system parameters change within a relatively broad range. In its simplest form, the PID controller is parametrized by three scalar parameters, which may be determined by a variety of methods, some of them manual/heuristic. We will later discuss briefly PID control in Section 5.7, and point out that the automatic choice of its parameters can be considered within the context of the broader methodology of approximation in policy space.

#### 1.4 REINFORCEMENT LEARNING AND OPTIMAL CONTROL - SOME TERMINOLOGY

There has been intense interest in DP-related approximations in view of their promise to deal with the *curse of dimensionality* (the explosion of the computation as the number of states increases is dealt with the use of approximate cost functions) and the *curse of modeling* (a simulator/computer model may be used in place of a mathematical model of the problem). The current state of the subject owes much to an enormously beneficial cross-fertilization of ideas from optimal control (with its traditional emphasis on sequential decision making and formal optimization methodologies), and from artificial intelligence (and its traditional emphasis on learning through observation and experience, heuristic evaluation functions in game-playing programs, and the use of feature-based and other representations).

The boundaries between these two fields are now diminished thanks to a deeper understanding of the foundational issues, and the associated methods and core applications. Unfortunately, however, there have been substantial differences in language and emphasis in RL-based discussions (where artificial intelligence-related terminology is used) and DP-based discussions (where optimal control-related terminology is used). This includes the typical use of maximization/value function/reward in the former field and the use of minimization/cost function/cost per stage in the latter field, and goes much further.

The terminology used in this book is standard in DP and optimal control, and in an effort to forestall confusion of readers that are accustomed to either the RL or the optimal control terminology, we provide a list of terms commonly used in RL, and their optimal control counterparts.

- (a) **Environment** = System.
- (b) **Agent** = Decision maker or controller.
- (c) **Action** = Decision or control.
- (d) **Reward of a stage** = (Opposite of) Cost of a stage.
- (e) **State value** = (Opposite of) Cost starting from a state.
- (f) **Value (or reward) function** = (Opposite of) Cost function.
- (g) **Maximizing the value function** = Minimizing the cost function.
- (h) **Action (or state-action) value** = Q-factor (or Q-value) of a state-control pair. (Q-value is also used often in RL.)
- (i) **Planning** = Solving a DP problem with a known mathematical model.
- (j) **Learning** = Solving a DP problem without using an explicit mathematical model. (This is the principal meaning of the term “learning” in RL. Other meanings are also common.)
- (k) **Self-learning** (or self-play in the context of games) = Solving a DP problem using some form of policy iteration.
- (l) **Deep reinforcement learning** = Approximate DP using value and/or policy approximation with deep neural networks.
- (m) **Prediction** = Policy evaluation.
- (n) **Generalized policy iteration** = Optimistic policy iteration.
- (o) **State abstraction** = State aggregation.
- (p) **Temporal abstraction** = Time aggregation.
- (q) **Learning a model** = System identification.
- (r) **Episodic task or episode** = Finite-step system trajectory.
- (s) **Continuing task** = Infinite-step system trajectory.
- (t) **Experience replay** = Reuse of samples in a simulation process.
- (u) **Bellman operator** = DP mapping or operator.
- (v) **Backup** = Applying the DP operator at some state.
- (w) **Sweep** = Applying the DP operator at all states.
- (x) **Greedy policy with respect to a cost function  $J$**  = Minimizing policy in the DP expression defined by  $J$ .
- (y) **Afterstate** = Post-decision state.
- (z) **Ground truth** = Empirical evidence or information provided by direct observation.

Some of the preceding terms will be introduced in future chapters. The reader may then wish to return to this section as an aid in connecting with the relevant RL literature.

### Notation

Unfortunately the confusion arising from different terminology has been exacerbated by the use of different notations. The present textbook roughly follows the “standard” notation of the Bellman/Pontryagin optimal control era; see e.g., the classical books by Athans and Falb [AtF66], Bellman [Bel67], and Bryson and Ho [BrH75]. This notation is consistent with the author’s other DP books.

A summary of our most prominently used symbols is as follows:

- (a)  $x$ : state.
- (b)  $u$ : control.
- (c)  $J$ : cost function.
- (d)  $g$ : cost per stage.
- (e)  $f$ : system function.
- (f)  $i$ : discrete state.
- (g)  $p_{ij}(u)$ : transition probability from state  $i$  to state  $j$  under control  $u$ .
- (h)  $\alpha$ : discount factor in discounted problems.

The  $x$ - $u$ - $J$  notation is standard in optimal control textbooks (e.g., the books by Athans and Falb [AtF66], and Bryson and Ho [BrH75], as well as the more recent book by Liberzon [Lib11]). The notations  $f$  and  $g$  are also used most commonly in the literature of the early optimal control period as well as later (unfortunately the more natural symbol “ $c$ ” has not been used much in place of “ $g$ ” for the cost stage). The discrete system notations  $i$  and  $p_{ij}(u)$  are very common in the discrete-state Markov decision problem and operations research literature, where discrete-state problems have been treated extensively [sometimes the alternative notation  $p(j | i, u)$  is used for the transition probabilities].

The RL literature addresses for the most part finite-state Markov decision problems, most frequently the discounted and stochastic shortest path infinite horizon problems that are discussed in Chapter 4. The most commonly used notation is  $s$  for state,  $a$  for action,  $r(s, a, s')$  for reward per stage,  $p(s' | s, a)$  or  $P_{s,a}(s')$  for transition probability from  $s$  to  $s'$  under action  $a$ , and  $\gamma$  for discount factor (see e.g., Sutton and Barto [SuB18]).

## 1.5 NOTES AND SOURCES

Our discussion of exact DP in this chapter has been brief since our focus in this book will be on approximate DP and RL. The author’s DP text-

book [Ber17] provides an extensive discussion of finite horizon exact DP, and its applications to discrete and continuous spaces problems, using a notation and style that is consistent with the present book. The books by Puterman [Put94] and by the author [Ber12] provide detailed treatments of infinite horizon finite-state Markovian decision problems. Continuous spaces infinite horizon problems are covered in the author's book [Ber12], while some of the more complex mathematical aspects of exact DP are discussed in the monograph by Bertsekas and Shreve [BeS78] (particularly the probabilistic/measure-theoretic issues associated with stochastic optimal control).

The author's abstract DP monograph [Ber18a] aims at a unified development of the core theory and algorithms of total cost sequential decision problems, and addresses simultaneously stochastic, minimax, game, risk-sensitive, and other DP problems, through the use of the abstract DP operator (or Bellman operator as it is often called in RL). The idea here is to gain insight through abstraction. In particular, the structure of a DP model is encoded in its abstract Bellman operator, which serves as the "mathematical signature" of the model. Thus, characteristics of this operator (such as monotonicity and contraction) largely determine the analytical results and computational algorithms that can be applied to that model. It is likely that some of the approximation algorithms of the present book are transferable to a broad variety of DP models, well beyond the ones studied here. The design and analysis of these algorithms can then benefit from the broad algorithmic principles that have been developed through an abstract viewpoint.

The approximate DP and RL literature has expanded tremendously since the connections between DP and RL became apparent in the late 80s and early 90s. We restrict ourselves to mentioning textbooks, research monographs, and broad surveys, which supplement our discussions, express related viewpoints, and collectively provide a guide to the literature. Moreover, inevitably our referencing reflects a cultural bias, and an overemphasis on sources that are familiar to the author and are written in a similar style to the present book (including the author's own works). Thus we wish to apologize in advance for the many omissions of important research references that are somewhat outside our own understanding and view of the field.

Two books were written on our subject in the 1990s, setting the tone for subsequent developments in the field. One in 1996 by Bertsekas and Tsitsiklis [BeT96], which reflects a decision, control, and optimization viewpoint, and another in 1998 by Sutton and Barto, which reflects an artificial intelligence viewpoint (a 2nd edition, [SuB18], was published in 2018). We refer to the former book and also to the author's DP textbooks [Ber12], [Ber17] for a broader discussion of some of the topics of the present book, including algorithmic convergence issues and additional DP models, such as those based on average cost optimization. For historical accounts of

the early development of the subject, see [BeT96], Section 6.7, and [SuB18], Section 1.7.

More recent books are by Gosavi [Gos15] (a much expanded 2nd edition of his 2003 monograph), which emphasizes simulation-based optimization and RL algorithms, Cao [Cao07], which focuses on a sensitivity approach to simulation-based methods, Chang, Fu, Hu, and Marcus [CFH13], which emphasizes finite-horizon/limited lookahead schemes and adaptive sampling, Busoniu et al. [BBD10], which focuses on function approximation methods for continuous space systems and includes a discussion of random search methods, Powell [Pow11], which emphasizes resource allocation and operations research applications, Vrable, Vamvoudakis, and Lewis [VVL13], which discusses neural network-based methods, on-line adaptive control, and continuous-time optimal control applications, Kochenderfer et al. [KAC15], which selectively discusses applications and approximations in DP and the treatment of uncertainty, Jiang and Jiang [JiJ17], which develops adaptive control within an approximate DP framework, and Liu et al. [LWW17], which deals with forms of adaptive dynamic programming, and topics in both RL and optimal control. The book by Krishnamurthy [Kri16] focuses on partial state information problems, with discussion of both exact DP, and approximate DP/RL methods. The book by Haykin [Hay08] discusses approximate DP in the broader context of neural network-related subjects. The book by Borkar [Bor08] is an advanced monograph that addresses rigorously many of the convergence issues of iterative stochastic algorithms in approximate DP, mainly using the so called ODE approach. The book by Meyn [Mey07] is broader in its coverage, but touches upon some of the approximate DP algorithms that we discuss.

Influential early surveys were written, from an artificial intelligence viewpoint, by Barto, Bradtke, and Singh [BBS95] (which dealt with the methodologies of real-time DP and its antecedent, real-time heuristic search [Kor90], and the use of asynchronous DP ideas [Ber82], [Ber83], [BeT89] within their context), and by Kaelbling, Littman, and Moore [KLM96] (which focused on general principles of reinforcement learning).

The volume by White and Sofge [WhS92] contains several surveys describing early work in the field. Several survey papers in the volume by Si, Barto, Powell, and Wunsch [SBP04] describe some approximation methods that we will not be covering in much detail in this book: linear programming approaches (De Farias [DeF04]), large-scale resource allocation methods (Powell and Van Roy [PoV04]), and deterministic optimal control approaches (Ferrari and Stengel [FeS04], and Si, Yang, and Liu [SYL04]). Updated accounts of these and other related topics are given in the survey collections by Lewis, Liu, and Lendaris [LLL08], and Lewis and Liu [LeL13].

Recent extended surveys and short monographs are Borkar [Bor09] (a methodological point of view that explores connections with other Monte Carlo schemes), Lewis and Vrable [LeV09] (a control theory point of view),

Szepesvari [Sze10] (which discusses approximation in value space from a RL point of view), Deisenroth, Neumann, and Peters [DNP11], and Grondman et al. [GBL12] (which focus on policy gradient methods), Browne et al. [BPW12] (which focuses on Monte Carlo Tree Search), Mausam and Kolobov [MaK12] (which deals with Markovian decision problems from an artificial intelligence viewpoint), Schmidhuber [Sch15], Arulkumaran et al. [ADB17], Li [Li17], Busoniu et al. [BDT18], and Caterini and Chang [CaC18] (which deal with reinforcement learning schemes that are based on the use of deep neural networks), the author's [Ber05a] (which focuses on rollout algorithms and model predictive control), [Ber11a] (which focuses on approximate policy iteration), and [Ber18b] (which focuses on aggregation methods), and Recht [Rec18a] (which focuses on continuous spaces optimal control).