Distributed Reinforcement Learning, Rollout, and Approximate Policy Iteration

by

Dimitri P. Bertsekas

Chapter 1

Dynamic Programming Principles

WWW site for book information and orders

http://www.athenasc.com

Athena Scientific, Belmont, Massachusetts

1

Dynamic Programming Principles

Contents

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. We primarily consider finite horizon problems, which involve a finite sequence of successive decisions, and are thus conceptually and analytically simpler. We also consider briefly the more intricate infinite horizon problems, but defer a more detailed discussion for Chapter 5.

We will discuss separately deterministic and stochastic problems (Sections 1.1 and 1.2, respectively). The reason is that deterministic problems are simpler and have some favorable characteristics, which allow the application of a broader variety of methods. Significantly they include challenging discrete and combinatorial optimization problems, which can be fruitfully addressed with some of the rollout and approximate policy iteration methods that are the main focus of this book.

In subsequent chapters, we will discuss selectively some major algorithmic topics in RL. A broader discussion of the subject may be found in the author's RL textbook [Ber19a], and the DP textbooks [Ber12], [Ber17], [Ber18a], the neuro-dynamic programming monograph [BeT96], as well as the textbook literature described in the last section of this chapter.

1.1 DETERMINISTIC DYNAMIC PROGRAMMING

In all DP problems, the central object is a discrete-time dynamic system that generates a sequence of states under the influence of control. The system may evolve deterministically or randomly (under the additional influence of a random disturbance).

1.1.1 Basic Finite Horizon Problem Formulation

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 of the system will be generally 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. Thus, a deterministic DP problem involves a system of the form

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, 1, \ldots, N-1, \tag{1.1}$$

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,

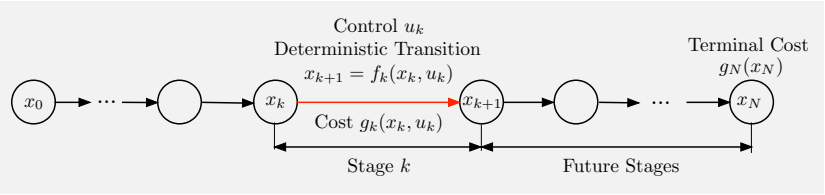

Figure 1.1.1 Illustration of a deterministic N-stage optimal control problem. Starting from state x_k, the next state under control u_k is generated nonrandomly, according to

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

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, i.e., the 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 $\{u_0, \ldots, u_{N-1}\}$ is

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

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

$$J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k=0,\ldots,N-1}} J(x_0; u_0, \ldots, u_{N-1}), \tag{1.3}$$

as a function of x_0 [here and later we write "min" (rather than "inf") even if we are not sure that the minimum is attained; similarly we write "max" (rather than "sup") even if we are not sure that the maximum is attained]. Figure 1.1.1 illustrates the main elements of the problem.

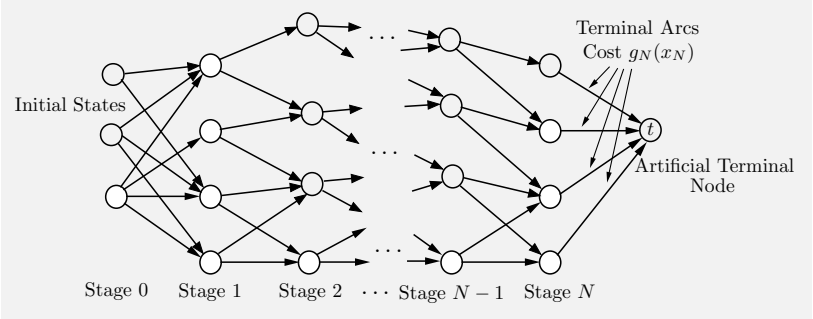

Figure 1.1.2 Transition graph for a deterministic finite-state system. Nodes correspond to states x_k. Arcs correspond to state-control pairs (x_k, u_k). An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$, respectively. We view the cost $g_k(x_k, u_k)$ of the transition as the length of this arc. The problem is equivalent to finding a shortest path from initial nodes of stage 0 to terminal node t.

Discrete Optimal Control Problems

There are many situations where the state and control spaces are naturally discrete and consist of a finite number of elements. 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 $\{u_0, \ldots, u_{N-1}\}$ correspond to paths originating at the initial state (a node 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 nodes of the graph (stage 0) to the terminal node t.* Here, by the length of a path we mean the sum of the lengths of its arcs.†

Generally, combinatorial optimization problems can be formulated as deterministic finite-state finite-horizon optimal control problem, as we will illustrate in Section 1.3.

† 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. See [Ber17], Section 2.1, and [Ber91], [Ber98] for extensive accounts of shortest path methods, which connect with our discussion here.

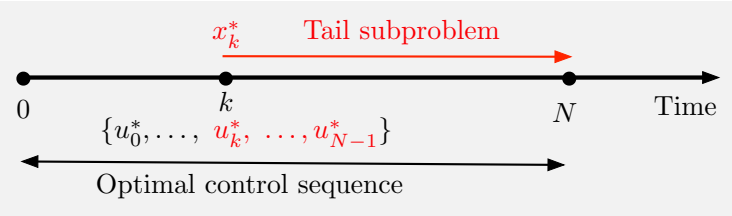

Figure 1.1.3 Schematic illustration of the principle of optimality. The tail

$$\{u_k^*, \ldots, u_{N-1}^*\}$$

of an optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ is optimal for the tail subproblem that starts at the state x_k^* of the optimal state trajectory.

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. 1.1.3.†

Principle of Optimality

Let $\{u_0^*, \ldots, u_{N-1}^*\}$ be an optimal control sequence, which together with x_0 determines the corresponding state sequence $\{x_1^*, \ldots, x_N^*\}$ 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 $\{u_k, \ldots, u_{N-1}\}$ with $u_m \in U_m(x_m)$, $m = k, \ldots, N-1$. Then the truncated optimal control sequence $\{u_k^*, \ldots, u_{N-1}^*\}$ 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 justification is simple. If the truncated control sequence $\{u_k^*, \ldots, u_{N-1}^*\}$ were not optimal as stated, we would be able to reduce the cost further

† In the words of Bellman [Bel57]: "An optimal trajectory has the property that at an intermediate point, no matter how it was reached, the rest of the trajectory must coincide with an optimal trajectory as computed from this intermediate point as the starting point."

by switching to an optimal sequence for the subproblem once we reach x_k^* (since the preceding choices of controls, $u_0^*, \ldots, u_{k-1}^*$, do not restrict our future choices).

For an auto travel analogy, suppose that the fastest route from Phoenix to Boston passes through St Louis. The principle of optimality translates to the obvious fact that the St Louis to Boston portion of the route is also the fastest route for a trip that starts from St Louis 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.

Finding an Optimal Control Sequence by DP

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

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

sequentially, starting from J_N^*, and proceeding backwards to J_{N-1}^*, J_{N-2}^*, etc. *The value $J_k^*(x_k)$ represents the optimal cost of the tail subproblem that starts at state x_k at time k.*

DP Algorithm for Deterministic Finite Horizon Problems

Start with

$$J_N^*(x_N) = g_N(x_N), \qquad \text{for all } x_N, \tag{1.4}$$

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

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \qquad \text{for all } x_k. \tag{1.5}$$

Note that at stage k, the calculation in (1.5) 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 $k = 0, 1, \ldots, N-1$, and all states x_k at time k, we have

$$J_k^*(x_k) = \min_{\substack{u_m \in U_m(x_m) \\ m=k,\ldots,N-1}} J(x_k; u_k, \ldots, u_{N-1}), \tag{1.6}$$

where

$$J(x_k; u_k, \ldots, u_{N-1}) = g_N(x_N) + \sum_{m=k}^{N-1} g_m(x_m, u_m), \tag{1.7}$$

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. This is intuitively clear from the relation (1.5), and a mathematical induction proof, starting from the initial condition (1.4), is very simple (see [Ber17], [Ber19a]).

Based on the interpretation (1.6) of $J_k^*(x_k)$, we call it 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 (1.5) 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 $J_0^*, \ldots, J_N^*$ have been obtained, we can use the following forward algorithm to construct an optimal control sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ and corresponding state trajectory $\{x_1^*, \ldots, x_N^*\}$ for the given initial state x_0.

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

Set

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

and

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

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

$$u_k^* \in \arg\min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*\big(f_k(x_k^*, u_k)\big) \Big], \tag{1.8}$$

and

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

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. Unfortu-

nately, in practice this is often prohibitively time-consuming, because 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 an idea that is central in RL: *approximation in value space*.† It constructs a suboptimal solution $\{\tilde{u}_0, \ldots, \tilde{u}_{N-1}\}$ in place of the optimal $\{u_0^*, \ldots, u_{N-1}^*\}$, based on using $\tilde{J}_k$ in place of J_k^* in the DP procedure (1.8).

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

Start with

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

and set

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

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

$$\tilde{u}_k \in \arg\min_{u_k \in U_k(\tilde{x}_k)} \Big[g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}\big(f_k(\tilde{x}_k, u_k)\big) \Big], \qquad (1.9)$$

and

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

In approximation in value space the calculation of the suboptimal sequence $\{\tilde{u}_0, \ldots, \tilde{u}_{N-1}\}$ is done by going forward (no backward calculation is needed once the approximate cost-to-go functions $\tilde{J}_k$ are available). This is similar to the calculation of the optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$, and is independent of how the functions $\tilde{J}_k$ are computed.

The construction of suitable $\tilde{J}_k$ can be done in many different ways, giving rise to some of the principal RL methods. One of these methods is rollout, which will be discussed in detail starting with the next chapter. In rollout, the approximate values $\tilde{J}_k(x_k)$ are obtained when needed by running a heuristic from the state x_k to the end of the horizon. The major theoretical property of rollout is *policy improvement*: the cost obtained by the preceding approximation in value space method is less or equal to the cost obtained with the heuristic starting from x_0.

† Approximation in value space is a simple idea that has been used quite extensively for deterministic problems, well before the development of the modern RL methodology. For example it underlies the widely used A^* method for computing approximate solutions to large scale shortest path problems.

Q-Factors and Q-Learning

An alternative (and equivalent) form of the DP algorithm (1.5), 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

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big). \tag{1.10}$$

Thus the optimal Q-factors are simply the expressions that are minimized in the right-hand side of the DP equation (1.5).

Note that the optimal cost function J_k^* can be recovered from the optimal Q-factor Q_k^* by means of the minimization

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

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

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

Exact and approximate forms of this and other related algorithms comprise an important class of RL methods known as *Q-learning*.†

The expression

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

which is minimized in approximation in value space [cf. Eq. (1.9)] is known as the (approximate) *Q-factor of* (x_k, u_k). Note that the computation of the suboptimal control (1.9) 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.

---

† The term "Q-factor" has been used in the books [BeT96], [Ber19a], and is adopted here as well. Another term used is "action value" (at a given state). The terms "state-action value" and "Q-value" are also common in the literature. Also, the term "Q-learning" is used with several different meanings in the literature. In this book we use the term to refer to a broad class of methods where Q-factors play an important role in the computations, as opposed to a specific method.

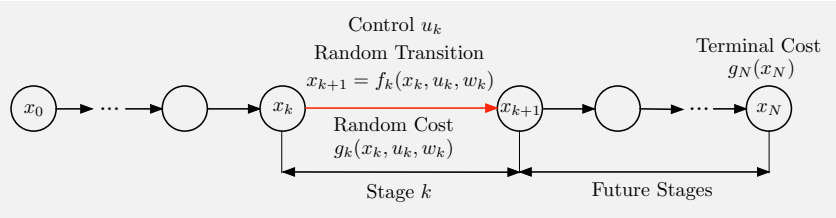

Figure 1.2.1 Illustration of an N-stage stochastic optimal control problem. Starting from state x_k, the next state under control u_k is generated randomly, according to

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

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

1.2 STOCHASTIC DYNAMIC PROGRAMMING

We will now discuss the DP algorithm for stochastic problems, which involve stochastic uncertainty in their system equation. We will start with the finite horizon case, and the extension of the ideas underlying the principle of optimality. Subsequently, we will consider the infinite horizon version of the problem, and provide an overview of the underlying theory and algorithmic methodology, in sufficient detail to allow us to speculate about infinite horizon extensions of finite horizon RL algorithms to be developed in Chapters 2-4. A more detailed discussion of the infinite horizon RL methodology will be given in Chapter 5.

1.2.1 Finite Horizon Problems

The stochastic 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 $P_k(\cdot \mid x_k, u_k)$ that may depend explicitly on x_k and u_k, but not on values of prior disturbances $w_{k-1}, \ldots, w_0$. The system has the form

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

where as earlier 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 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 from the deterministic case is that we optimize not over control sequences $\{u_0, \ldots, u_{N-1}\}$ [cf. Eq. (1.3)], but rather

over *policies* (also called *closed-loop control laws*, or *feedback policies*) that consist of a sequence of functions

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

where μ_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 or stage costs) 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 $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, the future states x_k and disturbances w_k are random variables with distributions defined through the system equation

$$x_{k+1} = f_k\big(x_k, \mu_k(x_k), w_k\big), \qquad k = 0, 1, \ldots, N-1,$$

and the given distributions $P_k(\cdot \mid x_k, u_k)$. Thus, for given functions g_k, $k = 0, 1, \ldots, N$, the expected cost of π starting at x_0 is

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

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

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

where Π is the set of all policies.

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

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

We 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 $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big) \Big\}. \quad (1.11)$$

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

The key fact is that starting from any initial state x_0, the optimal cost is equal to the number $J_0^*(x_0)$, 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]).

Simultaneously with the off-line computation of the optimal cost-to-go functions $J_0^*, \ldots, J_N^*$, we can compute and store an optimal policy $\pi^* = \{\mu_0^*, \ldots, \mu_{N-1}^*\}$ by minimization in Eq. (1.11). 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 π^* and to calculate the control $\mu_k^*(x_k)$ by executing the minimization (1.11) 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. (1.8) and (1.9).

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

$$\tilde{\mu}_k(x_k) \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}. \tag{1.12}$$

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 (1.12) needs to be performed only for the N states $x_0, \ldots, x_{N-1}$ 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 a 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 case of deterministic problems [cf. Eq. (1.10)], as the expressions that are minimized in the right-hand side of the stochastic DP equation (1.11). They are given by

$$Q_k^*(x_k, u_k) = E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big) \Big\}. \tag{1.13}$$

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

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

Similar to the deterministic case, Q-learning involves the calculation of either the optimal Q-factors (1.13) or approximations $\tilde{Q}_k(x_k, u_k)$. The approximate Q-factors may be obtained using approximation in value space

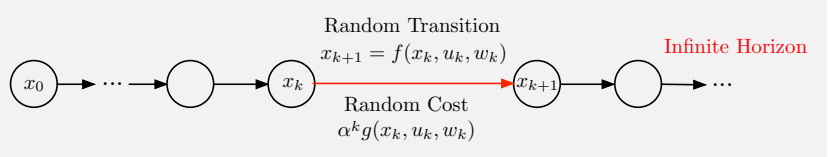

Figure 1.2.2 Illustration of an infinite horizon problem. The system and cost per stage are stationary, except for the use of a discount factor α. If $\alpha = 1$, there is typically a special cost-free termination state that we aim to reach.

schemes, and can be used to obtain approximately optimal policies through the Q-factor minimization

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

1.2.2 Infinite Horizon Problems - An Overview

In Chapter 5 we will deal with two types of infinite horizon problems, where we aim to minimize the total cost over an infinite number of stages, given by

$$J_\pi(x_0) = \lim_{N \to \infty} \underset{\substack{w_k \\ k=0,1,\ldots}}{E} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big) \right\}; \tag{1.14}$$

see Fig. 1.2.2. Here, $J_\pi(x_0)$ denotes the cost associated with an initial state x_0 and a policy $\pi = \{\mu_0, \mu_1, \ldots\}$, and α is a scalar in the interval $(0, 1]$.

When α is strictly less that 1, it has the meaning of a *discount factor*, and its effect is that future costs matter to us less than the same costs incurred at the present time. Among others, a discount factor guarantees that the limit defining $J_\pi(x_0)$ exists and is finite (assuming that the range of values of the stage cost g is bounded). This is a nice mathematical property that makes discounted problems analytically and algorithmically tractable.

Thus, by definition, the infinite horizon cost of a policy is the limit of its finite horizon costs as the horizon tends to infinity. The two types of problems that we will focus on are:

(a) *Stochastic shortest path problems* (SSP for short). Here, $\alpha = 1$ but there is a special cost-free termination state; once the system reaches that state it remains there at no further cost. In some types of problems, the termination state may represent a goal state that we are trying to reach at minimum cost, while in others it may be a state that we are trying to avoid for as long as possible. We will mostly assume a problem structure such that termination is inevitable under all policies. Thus the horizon is in effect finite, but its length is random and may be affected by the policy being used. A significantly more

complicated type of SSP problems, which we will discuss selectively, arises where termination can be guaranteed only under an optimal policy. Some common types of SSP belong to this category, including deterministic shortest path problems that involve graphs with cycles.

(b) *Discounted problems*. Here, $\alpha < 1$ and there need not be a termination state. However, we will see that a discounted problem can be readily converted to an SSP problem. It turns out that this can be done by introducing an artificial termination state to which the system moves with probability $1-\alpha$ at every state and stage, thus making termination inevitable. As a result, algorithms and analysis for SSP problems can be easily adapted to discounted problems. Moreover, a common line of analysis and algorithmic development may often be used for both SSP and discounted problems with a finite number of states. This also makes possible a unified line of analysis, which we will develop in Chapter 5.

Infinite Horizon Theory and Algorithms

There are several analytical and computational issues regarding our infinite horizon problems. Many of them revolve around the relation between the optimal cost function J^* of the infinite horizon problem and the optimal cost functions of the corresponding N-stage problems.

In particular, consider the undiscounted case ($\alpha = 1$) and let $J_N(x)$ denote the optimal cost of the problem involving N stages, initial state x, cost per stage $g(x, u, w)$, and zero terminal cost. This cost is generated after N iterations of the DP algorithm

$$J_{k+1}(x) = \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x,u,w) + J_k\big(f(x,u,w)\big) \Big\}, \qquad k = 0, 1, \ldots, \quad (1.15)$$

starting from the initial condition $J_0(x) = 0$ for all x.† The algorithm (1.15) is known as the *value iteration* algorithm (VI for short). Since the infinite horizon cost of a given policy is, by definition, the limit of the corresponding N-stage costs as $N \to \infty$, it is natural to speculate that:

(1) The optimal infinite horizon cost is the limit of the corresponding N-stage optimal costs as $N \to \infty$; i.e.,

$$J^*(x) = \lim_{N \to \infty} J_N(x) \quad (1.16)$$

for all states x.

† This is just the finite horizon DP algorithm, except that we have reversed the time indexing to suit our infinite horizon context. In particular, the index of the N-step optimal cost functions produced by the algorithm is incremented with each iteration, and not decremented as in the finite horizon case.

(2) The following equation should hold for all states x,

$$J^*(x) = \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x,u,w) + J^*\big(f(x,u,w)\big) \Big\}. \tag{1.17}$$

This is obtained by taking the limit as $N \to \infty$ in the VI algorithm (1.15) using Eq. (1.16). The preceding equation, called *Bellman's equation*, is really a system of equations (one equation per state x), which has as solution the optimal costs-to-go of all the states.

(3) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (1.17) for each x, then the policy $\{\mu, \mu, \ldots\}$ should be optimal. This type of policy is called *stationary*. Intuitively, optimal policies can be found within this class of policies, since optimization of the future costs (the tail subproblem) when starting at a given state looks the same regardless of the time when we start.

All three of the preceding results hold for finite-state SSP problems under suitable assumptions. They also hold for discounted problems in suitably modified form that incorporates the discount factor, provided the cost per stage function g is bounded over the set of possible values of (x,u,w) (see [Ber12], Chapter 1). In particular, the discounted version of the VI algorithm of Eq. (1.15) is

$$J_{k+1}(x) = \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x,u,w) + \alpha J_k\big(f(x,u,w)\big) \Big\}, \qquad k = 0, 1, \ldots, \tag{1.18}$$

while the discounted version of Bellman's equation [cf. Eq. (1.17)] is

$$J^*(x) = \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x,u,w) + \alpha J^*\big(f(x,u,w)\big) \Big\}. \tag{1.19}$$

The VI algorithm of Eq. (1.18) simply expresses the fact that the optimal cost for $N+1$ stages is obtained by minimizing the sum of the first stage cost and the optimal cost for the next N stages starting from the next state $f(x,u,w)$. The latter cost, however, should be discounted by α in view of the cost definition (1.14). The intuitive interpretation of the Bellman equation (1.19) is that it is the limit as $N \to \infty$ of the VI algorithm (1.18) assuming that $J_k \to J^*$.

Example 1.2.1

This example is representative of "search-and-rescue" problems, but we will describe it in more graphic terms. A spider and a fly move along a straight line at times $k = 0, 1, \ldots$ The initial positions of the fly and the spider are integer. At each time period, the fly moves one unit to the left with probability p, one unit to the right with probability p, and stays where it is with probability $1 - 2p$. The spider, knows the position of the fly at the beginning of each

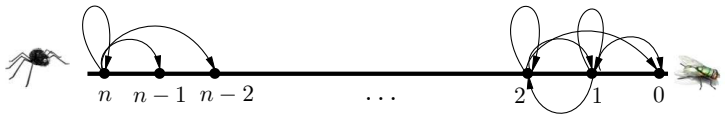

Figure 1.2.3 Illustration of transition probabilities of the spider and fly problem. The state is the distance between spider and fly, and cannot increase, except when the spider and fly are one unit apart. There are $n + 1$ states, with state 0 being the termination state.

period, and will always move one unit towards the fly if its distance from the fly is more that one unit. If the spider is one unit away from the fly, it will either move one unit towards the fly or stay where it is. If the spider and the fly land in the same position at the end of a period, then the spider captures the fly and the process terminates. The spider's objective is to capture the fly in minimum expected time; see Fig. 1.2.3.

We view as state the distance between spider and fly. Then the problem can be formulated as an SSP problem with states $0, 1, \ldots, n$, where n is the initial distance. State 0 is a termination state where the spider captures the fly. Let us denote $p_{1y}(m)$ and $p_{1y}(\overline{m})$ the transition probabilities from state 1 to state y under the two spider choices/controls of moving and not moving, respectively, and let us denote by p_{xy} the transition probabilities from a state $x \geq 2$.† We have

$$p_{xx} = p, \quad p_{x(x-1)} = 1 - 2p, \quad p_{x(x-2)} = p, \qquad x \geq 2,$$

$$p_{11}(m) = 2p, \quad p_{10}(m) = 1 - 2p,$$

$$p_{12}(\overline{m}) = p, \quad p_{11}(\overline{m}) = 1 - 2p, \quad p_{10}(\overline{m}) = p,$$

with all other transition probabilities being 0.

For states $x \geq 2$, Bellman's equation is written as

$$J^*(x) = 1 + pJ^*(x) + (1 - 2p)J^*(x - 1) + pJ^*(x - 2), \quad x \geq 2, \tag{1.20}$$

where for the termination state 0, we have

$$J^*(0) = 0.$$

The only state where the spider has a choice is when it is one unit away from the fly, and for that state Bellman's equation is given by

$$J^*(1) = 1 + \min\big[2pJ^*(1),\, pJ^*(2) + (1 - 2p)J^*(1)\big], \tag{1.21}$$

† Transition probabilities are a common way to describe stochastic finite-state systems. In particular, given the transition probabilities $p_{xy}(u)$, i.e., the conditional probabilities of moving from state x to state y under control u, an equivalent system equation description is $x_{k+1} = w_k$, where the disturbance w_k is generated according to the conditional distribution

$$p(w_k = y \mid x_k = x,\, u_k = u) = p_{xy}(u),$$

for all possible values of (x, y, u).

where the first and the second expression within the bracket above are associated with the spider moving and not moving, respectively. By writing Eq. (1.20) for $x = 2$, we obtain

$$J^*(2) = 1 + pJ^*(2) + (1 - 2p)J^*(1),$$

from which

$$J^*(2) = \frac{1}{1-p} + \frac{(1-2p)J^*(1)}{1-p}. \tag{1.22}$$

Substituting this expression in Eq. (1.21), we obtain

$$J^*(1) = 1 + \min\left[2pJ^*(1),\ \frac{p}{1-p} + \frac{p(1-2p)J^*(1)}{1-p} + (1-2p)J^*(1)\right],$$

or equivalently,

$$J^*(1) = 1 + \min\left[2pJ^*(1),\ \frac{p}{1-p} + \frac{(1-2p)J^*(1)}{1-p}\right]. \tag{1.23}$$

Thus Bellman's equation at state 1 can be reduced to the one-dimensional Eq. (1.23), and once this equation is solved it can be use to solve the Bellman Eq. (1.20) for all x. To solve Eq. (1.23), we consider the two cases where the first expression within the bracket is larger and is smaller than the second expression. Thus we solve for $J^*(1)$ in the two cases where

$$J^*(1) = 1 + 2pJ^*(1), \tag{1.24}$$

$$2pJ^*(1) \leq \frac{p}{1-p} + \frac{(1-2p)J^*(1)}{1-p}, \tag{1.25}$$

and

$$J^*(1) = 1 + \frac{p}{1-p} + \frac{(1-2p)J^*(1)}{1-p}, \tag{1.26}$$

$$2pJ^*(1) \geq \frac{p}{1-p} + \frac{(1-2p)J^*(1)}{1-p}. \tag{1.27}$$

The solution of Eq. (1.24) is seen to be

$$J^*(1) = \frac{1}{1-2p},$$

and by substitution in Eq. (1.25), we find that this solution is valid when

$$\frac{2p}{1-2p} \leq \frac{p}{1-p} + \frac{1}{1-p},$$

or equivalently (after some calculation), $p \leq 1/3$. Thus for $p \leq 1/3$, it is optimal for the spider to move when it is one unit away from the fly.

Similarly, the solution of Eq. (1.26) is seen to be

$$J^*(1) = \frac{1}{p},$$

and by substitution in Eq. (1.27), we find that this solution is valid when

$$2 \geq \frac{p}{1-p} + \frac{1-2p}{p(1-p)},$$

or equivalently (after some calculation), $p \geq 1/3$. Thus, for $p \geq 1/3$ it is optimal for the spider not to move when it is one unit away from the fly.

The minimal expected number of steps for capture when the spider is one unit away from the fly was calculated earlier to be

$$J^*(1) = \begin{cases} 1/(1-2p) & \text{if } p \leq 1/3, \\ 1/p & \text{if } p \geq 1/3. \end{cases}$$

Given the value of $J^*(1)$, we can calculate from Eq. (1.22) the minimal expected number of steps for capture when two units away, $J^*(2)$, and we can then obtain the remaining values $J^*(i)$, $i = 3, \ldots, n$, from Eq. (1.20).

The problem can also be solved by using the VI algorithm, which takes the form

$$J_{k+1}(0) = 0, \qquad J_{k+1}(1) = 1 + \min\big[2pJ_k(1),\, pJ_k(2) + (1-2p)J_k(1)\big],$$

$$J_{k+1}(x) = 1 + pJ_k(x) + (1-2p)J_k(x-1) + pJ_k(x-2), \quad x \geq 2,$$

where the initial conditions are $J_0(0) = 0$ and $J_0(x)$ are arbitrary scalars for $x \geq 1$. The algorithm is iterative and can be shown to converge to the optimal costs $J^*(x)$ as $k \to \infty$.

Despite the simplicity of its context, the spider-and-fly problem can become quite difficult even with seemingly minor modifications. For example if the fly is "faster" than the spider, and can move up to two units to the left or right with positive probabilities, the distance between spider and fly may become arbitrarily large with positive probability, and the number of states becomes infinite. Then, not only the Bellman equation cannot be solved analytically, but also the standard theory of SSP problems does not apply (a more complex extension of the theory that applies to infinite state space problems is needed; see [Ber18a], Chapter 4). As another example, suppose that the spider has "blurry" vision and cannot see the exact location of the fly. Then we may formulate the problem as one of partial state observation, where the choice of the fly depends on all its past observations. We will consider problems of this type later, and we will see that they are far more difficult, both analytically and computationally, than the problems we have discussed so far, where controls are chosen with exact knowledge of the state.

Other extensions of the problem are also possible. For example, instead of moving along a line, the spider and fly may move within a two-dimensional space. Moreover, there may be multiple spiders that cooperate as they aim to capture the fly. (A less violent formulation of this problem is typical of

search-and-rescue problems, where multiple rescuers try to rescue a stranded hiker.) The control here is the set of collective actions of the spiders (or rescuers).†

Let us also note that in addition to the VI algorithm, another major class of infinite horizon methods is based on *policy iteration*, which involves the repeated use of policy improvement and rollout (starting from some policy and generating a "better" policy). These are concepts that we will explain in Chapters 2 and 3 for the finite horizon context. We will discuss several different forms of policy iteration and its approximate versions in Chapter 5. We will argue there that policy iteration is the DP concept that forms the foundation for self-learning in RL, i.e., learning from data that is self-generated (from the system itself as it operates) rather than from data supplied from an external source.

1.3 EXAMPLES, VARIATIONS, AND SIMPLIFICATIONS

In this section we provide a few examples that illustrate problem formulation techniques, solution methods, and adaptations of the basic DP algorithm to various contexts. We refer to DP textbooks for extensive additional discussions of modeling and problem formulation techniques. In particular, a broad set of examples can be found in the author's DP and RL books [Ber12], [Ber17], [Ber19a], as well as in the neuro-dynamic programming monograph [BeT96].

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 $\{u_0, \ldots, u_{N-1}\}$, 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 relevant for future optimization*, i.e., 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

† A structure that involves a control with multiple components applied by autonomous decision makers, possibly with some coordination, is typical of "multiagent" problems. We will discuss such problems later.

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 may thus be 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 providing alternative descriptions of the state space. An important paradigm is problems involving *partial* or *imperfect* state information, where x_k evolves over time but is not fully accessible for measurement (for example, x_k may be the position/velocity vector of a moving vehicle, but we may obtain measurements of just the position). If I_k is the collection of all measurements and controls up to time k (the *information vector*), it is correct to use I_k as state in a reformulated DP problem that involves perfect state observation. However, a better alternative may be to use as state the conditional probability distribution $P_k(x_k \mid I_k)$, called *belief state*, which (as it turns out) subsumes all the information that is useful for the purposes of choosing a control. On the other hand, the belief state $P_k(x_k \mid I_k)$ is an infinite-dimensional object, whereas I_k may be finite dimensional, so the best choice may be problem-dependent. Still, in either case, the stochastic DP algorithm applies, with the sufficient statistic [whether I_k or $P_k(x_k \mid I_k)$] playing the role of the state; see Section 1.3.4.

1.3.1 Discrete Deterministic Optimization

Discrete deterministic optimization problems, including challenging combinatorial problems, can be typically formulated as DP problems by breaking down each feasible solution into a sequence of decisions/controls. This formulation often leads to an intractable DP computation because of an exponential explosion of the number of states as time progresses. However, a DP formulation brings to bear approximate DP methods, such as rollout

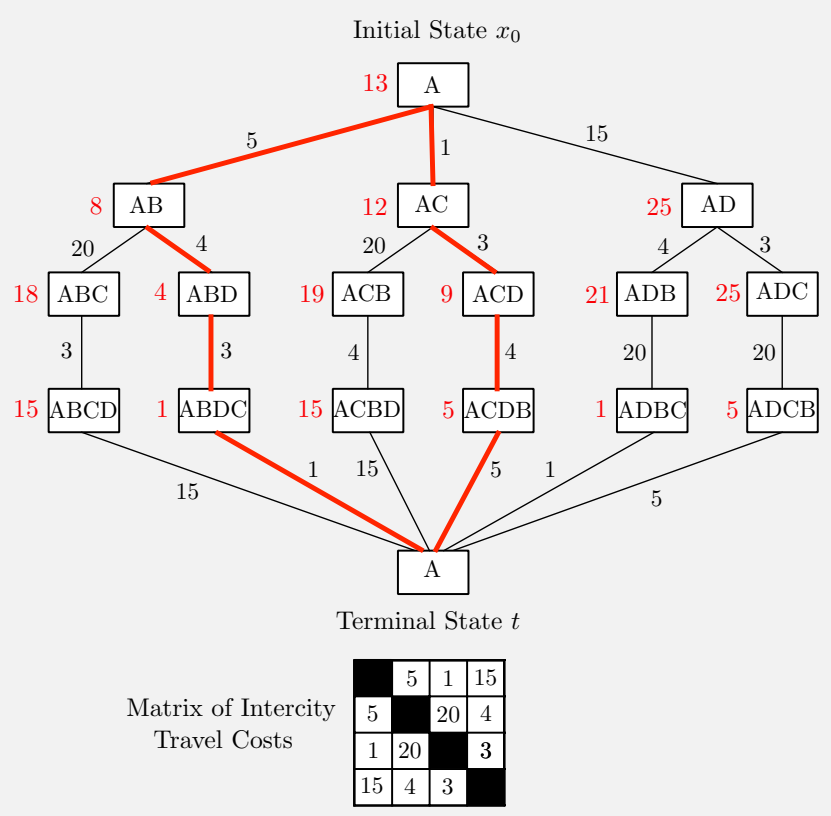

Figure 1.3.1 Example of a DP formulation of the traveling salesman problem. The travel times between the four cities A, B, C, and D are shown in the matrix at the bottom. We form a graph whose nodes are the k-city sequences and correspond to the states of the kth stage, assuming that A is the starting city. 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. (1.8)], starting from the initial state x_0.

and others, to be discussed in future chapters. These methods may involve computations that are tractable because they are not affected by a large space as much as exact DP. 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 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 $k = 1, \ldots, N$. The k-city sequences correspond to the states of the kth stage. The initial state x_0 consists of some city, taken as the start (city A in the example of Fig. 1.3.1). 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. 1.3.1. 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 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 in future chapters.

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

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

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 $u = (u_0, \ldots, u_{N-1})$, where N is a positive integer. We can then view the problem as a sequential decision problem, where the components $u_0, \ldots, u_{N-1}$ are selected one-at-a-time. A k-tuple $(u_0, \ldots, u_{k-1})$ consisting of the first k components of a solution is called a *k-solution*. We associate k-solutions with the kth stage of the finite horizon DP problem shown in Fig. 1.3.2. In particular, for $k = 1, \ldots, N$, we view as the states of the kth stage all the k-tuples $(u_0, \ldots, u_{k-1})$. For stage $k = 0, \ldots, N-1$, we view u_k as the control. The initial state is an artificial state denoted s. From this state, by applying u_0, we may move to any "state" (u_0), with u_0 belonging to the set

$$U_0 = \big\{\tilde{u}_0 \mid \text{there exists a solution of the form } (\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_{N-1}) \in U\big\}.$$

Thus U_0 is the set of choices of u_0 that are consistent with feasibility.

More generally, from a state $(u_0, \ldots, u_{k-1})$, we may move to any state of the form $(u_0, \ldots, u_{k-1}, u_k)$, upon choosing a control u_k that belongs to the set

$$U_k(u_0, \ldots, u_{k-1}) = \big\{u_k \mid \text{for some } \overline{u}_{k+1}, \ldots, \overline{u}_{N-1} \text{ we have } (u_0, \ldots, u_{k-1}, u_k, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}) \in U\big\}.$$

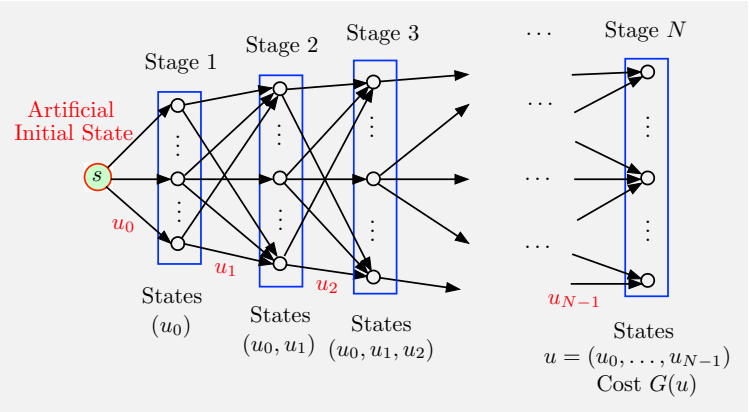

Figure 1.3.2. Formulation of a discrete optimization problem as a DP problem with N stages. There is a cost $G(u)$ only at the terminal stage on the arc connecting an N-solution $u = (u_0, \ldots, u_{N-1})$ upon reaching the terminal state. Alternative formulations may use fewer states by taking advantage of the problem's structure.

These are the choices of u_k that are consistent with the preceding choices $u_0, \ldots, u_{k-1}$, and are also consistent with feasibility. The last stage corresponds to the N-solutions $u = (u_0, \ldots, u_{N-1})$, and the terminal cost is $G(u)$; see Fig. 1.3.2. All other transitions in this DP problem formulation have cost 0.

Let $J_k^*(u_0, \ldots, u_{k-1})$ denote the optimal cost starting from the k-solution $(u_0, \ldots, u_{k-1})$, i.e., the optimal cost of the problem over solutions whose first k components are constrained to be equal to $u_0, \ldots, u_{k-1}$. The DP algorithm is described by the equation

$$J_k^*(u_0, \ldots, u_{k-1}) = \min_{u_k \in U_k(u_0, \ldots, u_{k-1})} J_{k+1}^*(u_0, \ldots, u_{k-1}, u_k), \qquad (1.28)$$

with the terminal condition

$$J_N^*(u_0, \ldots, u_{N-1}) = G(u_0, \ldots, u_{N-1}).$$

This algorithm 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_0^*. An optimal solution $(u_0^*, \ldots, u_{N-1}^*)$ is then constructed by going forward through the algorithm

$$u_k^* \in \arg \min_{u_k \in U_k(u_0^*, \ldots, u_{k-1}^*)} J_{k+1}^*(u_0^*, \ldots, u_{k-1}^*, u_k), \quad k = 0, \ldots, N-1, \qquad (1.29)$$

first compute u_0^*, then u_1^*, and so on up to u_{N-1}^*; cf. Eq. (1.8).

Of course here the number of states typically grows exponentially with N, but we can use the DP minimization (1.29) 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. (1.29). One possibility is to use as

$$\tilde{J}_{k+1}(u^*_0, \ldots, u^*_{k-1}, u_k),$$

the cost generated by a heuristic method that solves the problem suboptimally with the values of the first $k+1$ decision components fixed at $u^*_0, \ldots, u^*_{k-1}, u_k$. This is called the *rollout algorithm*, and it is a very simple and effective approach for approximate combinatorial optimization. It will be discussed in Chapters 2 and 3, and it will be related to the method of policy iteration and self-learning ideas.

Let us finally note that while we have used a general cost function G and constraint set C in our discrete optimization model of this section, in many problems G and/or C may have a special structure, which is consistent with a sequential decision making process. The traveling salesman Example 1.3.1 is a case in point, where G consists of N components (the intercity travel costs), one per stage.

1.3.2 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,

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

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. Problems involving games, such as chess, Go, backgammon, and others involve a termination state (the end of the game) and have played an important role in the development of the RL methodology.†

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.‡ 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.

† Games often involve two players/decision makers, in which case they can be addressed by suitably modified exact or approximate DP algorithms. The DP algorithm that we have discussed in this chapter involves a single decision maker, but can be used to find an optimal policy for one player against a fixed and known policy of the other player.

‡ When an upper bound on the number of stages to termination is not known, the problem must be formulated as an infinite horizon problem of the stochastic shortest path type.

Example 1.3.2 (The Four Queens Problem)

Four queens must be placed on a 4×4 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 4×4 board contains at most one queen. 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. 1.3.3) 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 4×4 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).

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 $0, \ldots, N-1$, 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 corre-

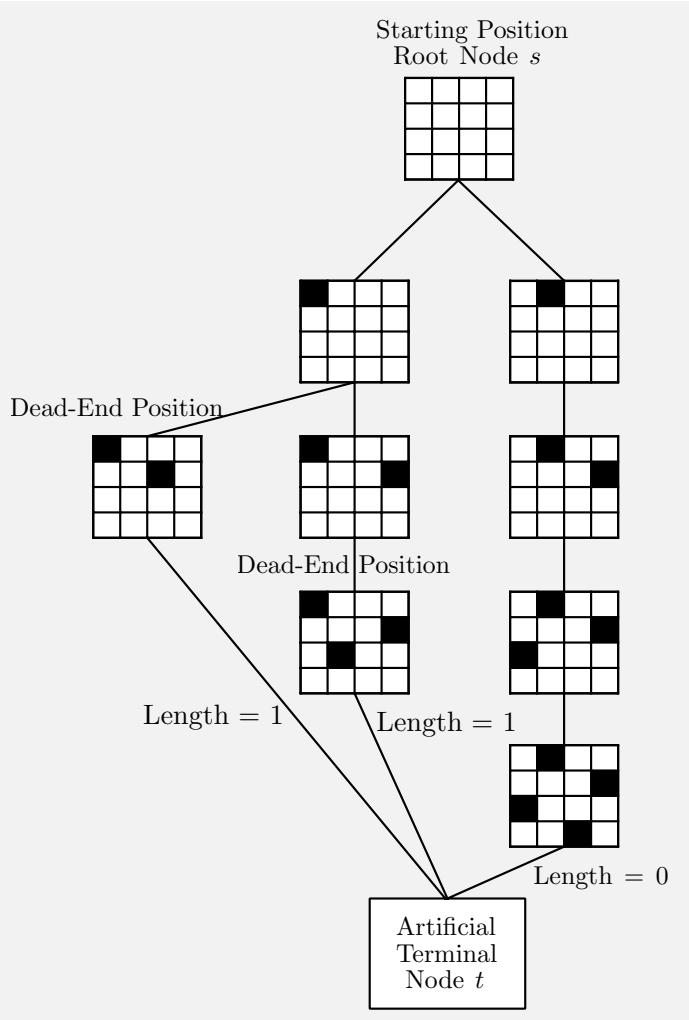

Figure 1.3.3 Discrete optimization formulation of the four queens problem. Symmetric positions resulting from placing a queen in one of the rightmost squares in the top row have been ignored. Squares containing a queen have been darkened. All arcs have length zero except for those connecting dead-end positions to the artificial terminal node.

sponds to having parked; see Fig. 1.3.4. At each stage $k = 1, \ldots, N-1$, we have three states: the artificial termination state t, and the two states F and $\overline{F}$, corresponding to space k being free or taken, respectively. At stage 0, we have only two states, F and $\overline{F}$, and at the final stage there is only one state, the termination state t. The decision/control is to park or continue at state F [there is no choice at states $\overline{F}$ and state t]. At stage k, the termination state

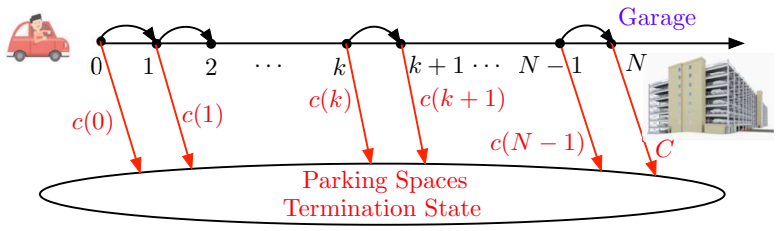

Figure 1.3.4 Cost structure of the parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, or continue to the next space $k+1$ at no cost. At space N (the garage) the driver must park at cost C.

t is reached at cost $c(k)$ when a parking decision is made (assuming location k is free). Otherwise, the driver continues to the next state at no cost. At stage N, the driver must park at cost C.

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 $k = 0, \ldots, N-1$ takes the form

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

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

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

$$J_k^*(t) = 0, \qquad k = 1, \ldots, N.$$

The minimization above corresponds to the two choices (park or not park) at the states F that correspond to a free parking space.

While this algorithm is easily executed, it can be written in a simpler and equivalent form. This can be done by introducing the scalars

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

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

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

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

From this algorithm we can also obtain the optimal parking policy:

Park at space $k = 0, \ldots, N-1$ if it is free and we have $c(k) \leq \hat{J}_{k+1}$.

1.3.3 State Augmentation, Time Delays, and Forecasts

In practice, we are often faced with situations where some of the assumptions of our stochastic optimal control problem are violated. For example, the disturbances may involve a complex probabilistic description that may create correlations that extend across stages, or the system equation may include dependences on controls applied in earlier stages, which affect the state with some delay. Generally, in such cases the problem can be reformulated into our basic problem format through a technique, which is called *state augmentation* because it typically involves the enlargement of the state space. The general guideline in state augmentation is to *include in the enlarged state at time k all the information that is known to the controller at time k and can be used with advantage in selecting u_k*. This includes time delays in the effects of control on future states, forecasts of probability distributions of future disturbances, and others control spaces. Generally, state augmentation often comes at a price: the reformulated problem may have a very complex state space. We provide some examples.

Time Delays

In some applications the system state x_{k+1} depends not only on the preceding state x_k and control u_k but also on earlier states and controls. Such situations can be handled by expanding the state to include an appropriate number of earlier states and controls.

As an example, assume that there is at most a single period time delay in the state and control; i.e., the system equation has the form

$$x_{k+1} = f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), \qquad k = 1, 2, \ldots, N-1, \tag{1.30}$$

$$x_1 = f_0(x_0, u_0, w_0).$$

If we introduce additional state variables y_k and s_k, and we make the identifications $y_k = x_{k-1}$, $s_k = u_{k-1}$, the system equation (1.30) yields

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ s_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, s_k, w_k) \\ x_k \\ u_k \end{pmatrix}. \tag{1.31}$$

By defining $\tilde{x}_k = (x_k, y_k, s_k)$ as the new state, we have

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

where the system function $\tilde{f}_k$ is defined from Eq. (1.31).

By using the preceding equation as the system equation and by expressing the cost function in terms of the new state, the problem is reduced to a problem without time delays. Naturally, the control u_k should now

depend on the new state $\tilde{x}_k$, or equivalently a policy should consist of functions μ_k of the current state x_k, as well as the preceding state x_{k-1} and the preceding control u_{k-1}.

When the DP algorithm for the reformulated problem is translated in terms of the variables of the original problem, it takes the form

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

$$\begin{aligned} &J_{N-1}(x_{N-1}, x_{N-2}, u_{N-2}) \\ &\quad = \min_{u_{N-1} \in U_{N-1}(x_{N-1})} \mathop{E}_{w_{N-1}} \Big\{ g_{N-1}(x_{N-1}, u_{N-1}, w_{N-1}) \\ &\qquad\qquad + J_N\big(f_{N-1}(x_{N-1}, x_{N-2}, u_{N-1}, u_{N-2}, w_{N-1})\big) \Big\}, \end{aligned}$$

$$\begin{aligned} J_k(x_k, x_{k-1}, u_{k-1}) = \min_{u_k \in U_k(x_k)} \mathop{E}_{w_k} \Big\{ & g_k(x_k, u_k, w_k) \\ & + J_{k+1}\big(f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), x_k, u_k\big) \Big\}, \quad k = 1, \ldots, N-2, \end{aligned}$$

$$J_0(x_0) = \min_{u_0 \in U_0(x_0)} \mathop{E}_{w_0} \Big\{ g_0(x_0, u_0, w_0) + J_1\big(f_0(x_0, u_0, w_0), x_0, u_0\big) \Big\}.$$

Similar reformulations are possible when time delays appear in the cost or the control constraints; for example, in the case where the cost has the form

$$E\left\{ g_N(x_N, x_{N-1}) + g_0(x_0, u_0, w_0) + \sum_{k=1}^{N-1} g_k(x_k, x_{k-1}, u_k, w_k) \right\}.$$

The extreme case of time delays in the cost arises in the nonadditive form

$$E\big\{ g_N(x_N, x_{N-1}, \ldots, x_0, u_{N-1}, \ldots, u_0, w_{N-1}, \ldots, w_0) \big\}.$$

Then, the problem can be reduced to the standard problem format, by taking as augmented state

$$\tilde{x}_k = \big(x_k, x_{k-1}, \ldots, x_0, u_{k-1}, \ldots, u_0, w_{k-1}, \ldots, w_0\big)$$

and $E\big\{g_N(\tilde{x}_N)\big\}$ as reformulated cost. Policies consist of functions μ_k of the present and past states $x_k, \ldots, x_0$, the past controls $u_{k-1}, \ldots, u_0$, and the past disturbances $w_{k-1}, \ldots, w_0$. Naturally, we must assume that the past disturbances are known to the controller. Otherwise, we are faced with a problem where the state is imprecisely known to the controller, which will be discussed in the next section.

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 state variables corresponding to the information that the forecasts provide. We will 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 $\{P_1, \ldots, P_m\}$; 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 $1, \ldots, m$, corresponding to the m possible forecasts, and ξ_k is a random variable taking the value i with probability p_i. The interpretation here is that when ξ_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

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

$$J_k^*(x_k, y_k) = \min_{u_k \in U_k(x_k)} \underset{w_k}{E} \Big\{ g_k(x_k, u_k, w_k) + \sum_{i=1}^{m} p_i J_{k+1}^*\big(f_k(x_k, u_k, w_k), i\big) \mid y_k \Big\}, \tag{1.32}$$

where y_k may take the values $1, \ldots, m$, 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.

1.3.4 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 $\mu_k(x_k)$, $k = 0, \ldots, N-1$. 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]; these problems are also popularly referred to with the acronym POMDP for *partially observed Markovian Decision problem*). 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.

The most common approach to deal with imperfect state information problems is to replace the state x_k with a *belief state*, which we will denote by b_k. It is the probability distribution of x_k given all the observations that have been obtained by the controller up to time k, and it can serve as "state" in an appropriate DP algorithm. The belief state can in principle be computed and updated by a variety of methods that are based on Bayes' rule, such as *Kalman filtering* (see e.g., [AnM79], [KuV86], [Kri16], [ChC17]) and *particle filtering* (see e.g., [GSS93], [DoJ09], [Can16],

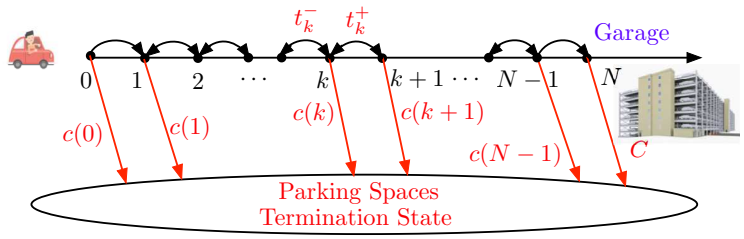

Figure 1.3.5 Cost structure and transitions of the bidirectional parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, can move to $k-1$ at cost t_k^- or can move to $k+1$ at cost t_k^+. At space N (the garage) the driver must park at cost C.

[Kri16]). 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 where the belief state is large enough to make an exact DP solution impossible.

Example 1.3.4 (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 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 t_i^- or can move to $i+1$ at a cost t_i^+. Moreover, the driver records and remembers the free/taken status of the spaces previously visited and may return to any of these spaces; see Fig. 1.3.5.

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 ξ and decreases by another known factor with the complementary probability $1-\xi$.

Here the belief state is the vector of current probabilities

$$\big(p(0), \ldots, p(N-1)\big),$$

and it is updated at each time based on the new observation: the free/taken status of the space visited at that time. This belief state can be computed exactly by the driver, given the parking status observations of the spaces visited thus far.

Despite their inherent computational difficulty, 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 information. Once this is done, it is possible to state an exact DP algorithm that is defined over the set of belief states. This algorithm has the form

$$J_k^*(b_k) = \min_{u_k \in U_k} \left[\hat{g}_k(b_k, u_k) + E_{z_{k+1}} \Big\{ J_{k+1}^* \big(F_k(b_k, u_k, z_{k+1}) \big) \Big\} \right], \quad (1.33)$$

where:

$J_k^*(b_k)$ denotes the optimal cost-to-go starting from belief state b_k at stage k.

U_k is the control constraint set at time k (since the state x_k is unknown at stage k, U_k must be independent of x_k).

$\hat{g}_k(b_k, u_k)$ denotes the expected stage cost of stage k. It is calculated as the expected value of the stage cost $g_k(x_k, u_k, w_k)$, with the joint distribution of (x_k, w_k) determined by the belief state b_k and the distribution of w_k.

$F_k(b_k, u_k, z_{k+1})$ denotes the belief state at the next stage $k+1$, given that the current belief state is b_k, control u_k is applied, and observation z_{k+1} is received following the application of u_k:

$$b_{k+1} = F_k(b_k, u_k, z_{k+1}). \quad (1.34)$$

This is the system equation for a perfect state information problem with state b_k, control u_k, "disturbance" z_{k+1}, and cost per stage $\hat{g}_k(b_k, u_k)$. The function F_k is viewed as a sequential belief estimator, which updates the current belief state b_k based on the new observation z_{k+1}. It is given by either an explicit formula or an algorithm (such as Kalman filtering or particle filtering) that is based on the probability distribution of z_k and the use of Bayes' rule.

The expected value $E_{z_{k+1}}\{\cdot\}$ is taken with respect to the distribution of z_{k+1}, given b_k and u_k. Note that z_{k+1} is random, and its distribution depends on x_k and u_k, so the expected value

$$E_{z_{k+1}} \Big\{ J_{k+1}^* \big(F_k(b_k, u_k, z_{k+1}) \big) \Big\}$$

in Eq. (1.33) is a function of b_k and u_k.

This algorithm is just the ordinary DP algorithm for the perfect state information problem shown in Fig. 1.3.6. It involves the system (1.34) and the cost per stage $\hat{g}_k(b_k, u_k)$. Note that since b_k takes values in a continuous

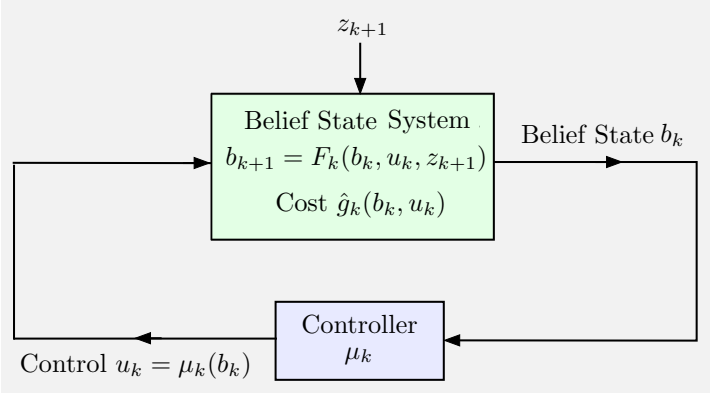

Figure 1.3.6 Schematic illustration of the view of an imperfect state information problem as one of perfect state information, whose state is the belief state b_k, i.e., the conditional probability distribution of x_k given all the observations up to time k. The observation z_{k+1} plays the role of the stochastic disturbance. The function F_k is a sequential estimator that updates the current belief state b_k.

space, the algorithm (1.33) can only be executed approximately, using the approximation in value space methods that we will discuss in subsequent chapters.

We refer to the textbook [Ber17], Chapter 4, for a detailed derivation of the DP algorithm (1.33), and to the monograph [BeS78] for a mathematical treatment that applies to infinite-dimensional state and disturbance spaces as well.

1.4 REINFORCEMENT LEARNING AND OPTIMAL CONTROL - SOME TERMINOLOGY

The current state of RL has greatly benefited from the cross-fertilization of ideas from optimal control and from artificial intelligence. The strong connections between these two fields are now widely recognized. Still, however, substantial differences in language and emphasis remain between RL-based discussions (where artificial intelligence-related terminology is used) and DP-based discussions (where optimal control-related terminology is used).

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 artificial intelligence 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; see also the RL textbook [Ber19a]. 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 book 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) α: 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 per 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 $p(j \mid i, u)$ is used for the transition probabilities].

The artificial intelligence 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 5. The most commonly used notation is s for state, a for action, $r(s, a, s')$ for reward per stage, $p(s' \mid s, a)$ or $P_{s,a}(s')$ for transition probability from s to s' under action a, and γ for discount factor.

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 textbook [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 this monograph. The books by Puterman

[Put94] and by the author [Ber12] provide detailed treatments of infinite horizon finite-state stochastic DP problems. The book [Ber12] also covers continuous/infinite state and control spaces problems, which present special analytical and computational challenges.

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, including partial state information problems. The book [Ber12] provides in an appendix an accessible summary introduction of the measure-theoretic framework of the book [BeS78], while the long paper by Yu and Bertsekas [YuB15] contains recent supplementary research with a particular focus on the analysis of policy iteration methods, which were not treated in [BeS78].

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. This abstract viewpoint has been adopted in Sections 5.3-5.6 of the present book.

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 and research monographs, which supplement our discussion, express related viewpoints, and collectively provide a guide to the literature. Two books were written 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 this book, including algorithmic convergence issues and additional DP models, such as those based on average cost and semi-Markov problem optimization.

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] (a 2nd edition of their 2007 monograph), which emphasizes finite-horizon/multistep lookahead schemes and adaptive sampling, Buso-

niu, Babuska, De Schutter, and Ernst [BBD10a], 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, Powell and Ryzhov [PoR12], which focuses on specialized topics in learning and Bayesian optimization, Vrabie, Vamvoudakis, and Lewis [VVL13], which discusses neural network-based methods and on-line adaptive control, 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, Liu, Wei, Wang, Yang, and Li [LWW17], which deals with forms of adaptive dynamic programming, and topics in both RL and optimal control, and Zoppoli, Sanguineti, Gnecco, and Parisini [ZSG20], which addresses neural network approximations in 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 popular approximate DP/RL algorithms.

The present monograph is similar in style, terminology, and notation to the author's recent RL textbook [Ber19a], which provides a more comprehensive account of the subject. It particular, the 2019 RL textbook includes a broader coverage of approximation in value space methods, including certainty equivalent control and aggregation methods. It also covers substantially policy gradient methods for approximation in policy space, which we have not addressed here.

In addition to textbooks, there are many surveys and short research monographs relating to our subject, which are rapidly multiplying in number. We refer to the author's RL textbook [Ber19a] for a fairly comprehensive list up to 2019.