Reinforcement Learning and Optimal Control

by

Dimitri P. Bertsekas

Massachusetts Institute of Technology

Chapter 1
Exact Dynamic Programming
DRAFT

This is Chapter 1 of the draft textbook "Reinforcement Learning and Optimal Control." The chapter represents "work in progress," and it will be periodically updated. It more than likely contains errors (hopefully not serious ones). Furthermore, its references to the literature are incomplete. Your comments and suggestions to the author at dimitrib@mit.edu are welcome. The date of last revision is given below.

December 14, 2018

1

Exact Dynamic Programming

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. 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 Chapter 4 and later chapters. 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

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

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,

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 space 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 space 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 cost 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†

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

as a function of x_0.

We will next illustrate deterministic problems with some examples.

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 specified in terms of 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. 1.1.1. 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 $g_N(x_N)$.

† 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.

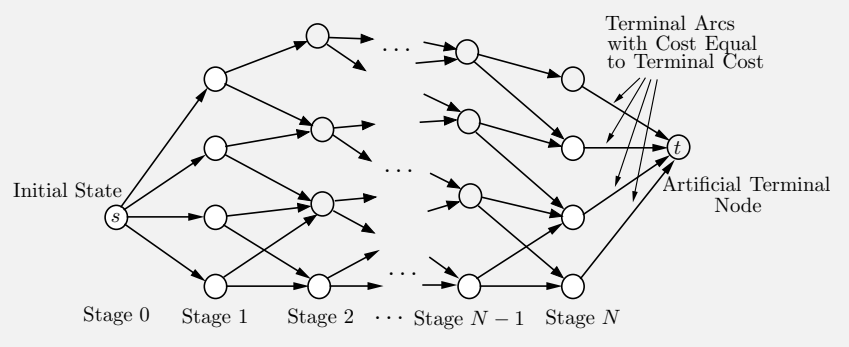

Figure 1.1.1 Transition graph for a deterministic finite-state system. Nodes correspond to states, and arcs correspond to state-control pairs (x_k, u_k). An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$, respectively. We view the cost $g_k(x_k, u_k)$ of the transition as the length of this arc. The problem is equivalent to finding a shortest path from the initial node s to the terminal node t.

Note that control sequences 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 a path we mean a sequence of arcs of the form $(j_1, j_2), (j_2, j_3), \ldots, (j_{k-1}, j_k)$, and 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. The following scheduling example illustrates the idea.

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. 1.1.2). The cost of a

† 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 see in Section 1.3.1. See also [Ber17], Section 2.1, and also [Ber98] for an extensive discussion of shortest path methods.

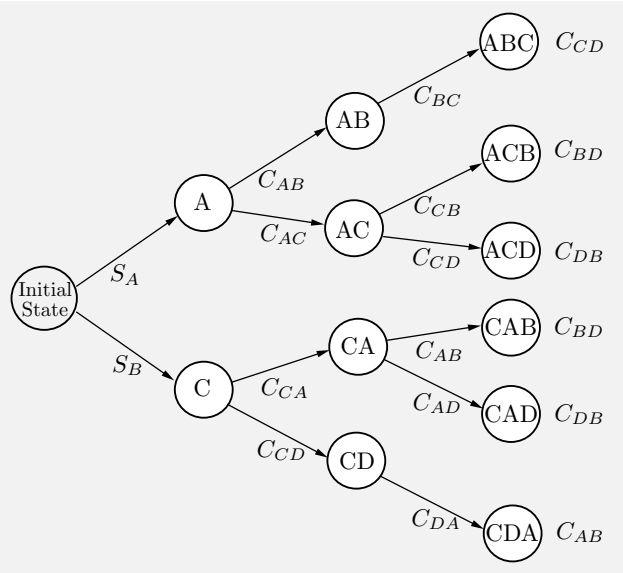

Figure 1.1.2 The transition graph of the deterministic scheduling problem of Example 1.1.1. Each arc of the graph corresponds to a decision leading from some state (the start node of the arc) to some other state (the end node of the arc). The corresponding cost is shown next to the arc. The cost of the last operation is shown as a terminal cost next to the terminal nodes of the graph.

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. 1.1.2. 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 in terms of the transition graph of Fig. 1.1.2. 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.

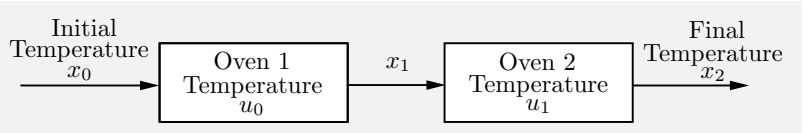

Figure 1.1.3 The linear-quadratic problem of Example 1.1.2 for $N = 2$. The temperature of the material evolves according to the system equation $x_{k+1} = (1-a)x_k + au_k$, where a is some scalar with $0 < a < 1$.

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 an n-dimensional vector of real variables, where n is a 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 $N-1$ ovens (see Fig. 1.1.3). Denote

x_0: initial temperature of the material,

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

u_{k-1}, $k = 1, \ldots, N-1$: 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

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

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.

Linear-quadratic problems with no constraints on the state or the control admit a nice analytical solution, as we will see later in Section 1.3.6.

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

The DP algorithm rests on a very simple idea, the *principle of optimality*, which roughly states the following rather obvious fact.

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_m^* at time m and wish to minimize the "cost-to-go" from time m to time N,

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

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

The intuitive justification of the principle of optimality is very simple. If the truncated control sequence $\{u_m^*, u_{m+1}^*, \ldots, u_{N-1}^*\}$ 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_i (since the preceding choices $u_0^*, \ldots, u_{m-1}^*$ of controls 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 an optimal control sequence can be constructed in piecemeal fashion, first constructing an optimal sequence for the "tail subproblem" involving the last stage, then extending the optimal policy to the "tail subproblem" involving the last two stages,

and continuing in this manner until an optimal policy 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. The numerical values of the transition and setup costs are shown in Fig. 1.1.4 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.

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. 1.1.4)

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. 1.1.4.

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. 1.1.4.

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.

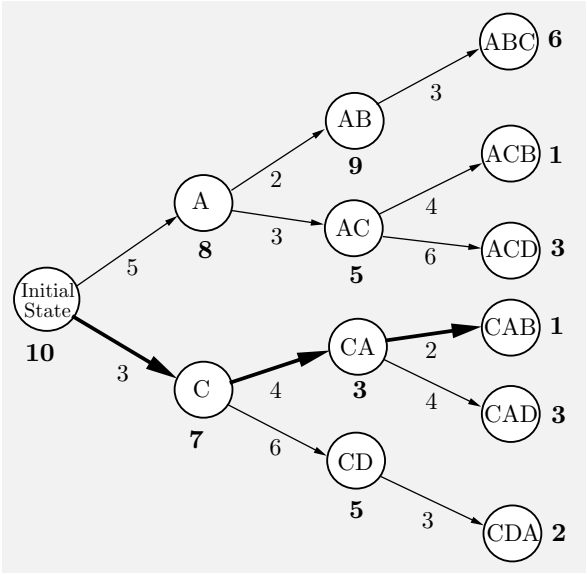

Figure 1.1.4 Transition graph of the deterministic scheduling problem, with the cost of each decision shown next to the corresponding arc. Next to each node/state we show the cost to optimally complete the schedule starting from that state. This is the optimal cost of the corresponding tail subproblem (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.

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. 1.1.4.

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 A in Fig. 1.1.4.

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 pos-

sibility is optimal, and the corresponding optimal cost is 10, as shown next to the initial state node in Fig. 1.1.4.

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 operation that starts the optimal schedule for the corresponding tail subproblem. In this way, by inspection of the graph and the computational results of Fig. 1.1.4, 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

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

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

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

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

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

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 algorithm is that for every initial state x_0, the optimal cost $J^*(x_0)$ is equal to the number $J_0^*(x_0)$, which is obtained at the last step of the DP algorithm. Indeed, a more general fact can be shown, namely that for all $m = 0, 1, \ldots, N-1$, and all states x_m at time m, we have

$$J_m^*(x_m) = \min_{\substack{u_k \in U_k(x_k) \\ k=m,\ldots,N-1}} J(x_m; u_m, \ldots, u_{N-1}), \tag{1.5}$$

where

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

i.e., $J_m^*(x_m)$ is the optimal cost for an $(N-m)$-stage tail subproblem that starts at state x_m and time m, and ends at time N.†

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

$$\begin{aligned} J_m^*(x_m) &= \min_{\substack{u_k \in U_k(x_k) \\ k=m,\ldots,N-1}} \left[g_N(x_N) + \sum_{k=m}^{N-1} g_k(x_k, u_k) \right] \\ &= \min_{u_m \in U_m(x_m)} \Bigg[g_m(x_m, u_m) \\ &\qquad + \min_{\substack{u_k \in U_k(x_k) \\ k=m+1,\ldots,N-1}} \left[g_N(x_N) + \sum_{k=m}^{N-1} g_k(x_k, u_k) \right] \Bigg] \\ &= \min_{u_m \in U_m(x_m)} \Big[g_m(x_m, u_m) + J_{m+1}^*\big(f_m(x_m, u_m)\big) \Big], \end{aligned}$$

where for the last equality we use the induction hypothesis.‡

Note that the algorithm solves every tail subproblem, i.e., the problem of 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 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

† Based on this fact, we call $J_m^*(x_m)$ the *optimal cost-to-go* at state x_m and time m, and refer to J_m^* as the *optimal cost-to-go function* or *optimal cost function* at time m. In maximization problems the DP algorithm (1.4) is written with maximization in place of minimization, and then J_m^* is referred to as the *optimal value function* at time m.

‡ A subtle mathematical point here is that, through the minimization operation, the cost-to-go functions J_m^* may take the value $-\infty$ for some x_m. Still the preceding induction argument is valid even if this is so.

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

and

$$x_{k+1}^* = f_k(x_k^*, u_k^*). \qquad (1.8)$$

The same algorithm can be used to find an optimal control sequence for any tail subproblem. Figure 1.1.4 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 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 $\{\tilde{u}_0, \ldots, \tilde{u}_{N-1}\}$ 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 (1.7).

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

Start with

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

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

$$\tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k). \qquad (1.10)$$

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.

1.1.4 Model-Free Approximate Solution - Q-Learning

Reducing the computational burden of DP through the use of approximate cost-to-go functions $\tilde{J}_k$ is a major objective of the RL methodology. However, there is another potential benefit: given $\tilde{J}_k$, we may be able to carry out the forward minimization (1.9) by using a simulator/computer model rather than a mathematical model of the functions f_k and g_k.

In particular, suppose that we have $\tilde{J}_k$ and a computer program, which for any pair (x_k, u_k) can generate the next state $f_k(x_k, u_k)$ and the stage cost $g_k(x_k, u_k)$. Then we can compute the expression

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

[cf. the right-hand side of Eq. (1.9)]; this is also known as the (approximate) *Q-factor of* (x_k, u_k). We can then implement the computation of the approximately optimal control (1.9) through the minimization

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

This may be viewed as a *model-free* approximate solution method, a simple example from a variety of methods that we will consider in subsequent chapters. Some of these methods are based on Monte-Carlo simulation, particularly for problems that involve stochastic uncertainty. Of course, for a method to fully qualify as model-free, the functions $\tilde{J}_k$ should also be obtained without the use of a model. There are several approaches for doing this, and they will be discussed starting with the next chapter.

There are also methods that 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

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big). \tag{1.11}$$

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

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

We will see later that exact and approximate forms of related algorithms can be implemented by using model-free simulation, 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 $P_k(\cdot \mid x_k, u_k)$ 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 before x_k is an element of some state space S_k, the control u_k is an element of some control space. The control u_k is constrained to take values in a given subset $U(x_k)$, which depends on the current state x_k.

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

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

where μ_k maps states x_k into controls $u_k = \mu_k(x_k)$, and is such that $\mu_k(x_k) \in U_k(x_k)$ for all $x_k \in S_k$. Such policies will be called *admissible*. 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, and this often necessitates the use of Monte Carlo simulation. As a result many of the methods that we will discuss for stochastic problems will involve the use of 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

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

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 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 admissible 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*, particularly in problems of maximizing reward.

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

$$J_N^*(x_N) = g_N(x_N), \tag{1.12}$$

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

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

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 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]).†

As in deterministic problems, the DP algorithm can be very time-consuming, in fact more so since it involves the expected value operation in Eq. (1.13). This motivates suboptimal control techniques, such as approximation in value space whereby we replace J_k^* with easier obtainable approximations $\tilde{J}_k$. We will discuss this approach at length in subsequent chapters.

Q-factors for Stochastic Problems

We can define optimal Q-factors for stochastic problem, similar to the case of deterministic problems [cf. Eq. (1.11)], as the expressions that are minimized in the right-hand side of the stochastic DP equation (1.13). They are given by

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

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

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

† There are some technical/mathematical difficulties here, having to do with the expected value operation in Eq. (1.13) 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 terms.

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

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

We will later discuss approximation in value space techniques based on approximately optimal Q-factors $\tilde{Q}_k(x_k,u_k)$. The corresponding suboptimal policy, denoted by $\{\tilde{\mu}_0,\ldots,\tilde{\mu}_{N-1}\}$, is obtained from the minimization

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

for all x_k and k.

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 $\{u_0,\ldots,u_{N-1}\}$, 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 known to the controller at time k and can be used with advantage in choosing u_k.

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 which would be defined 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 4.3 of [Ber17]).

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 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 $P_k(y_k \mid I_k)$, 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 $P_k(y_k \mid I_k)$ is an infinite-dimensional quantity, whereas I_k may be finite dimensional, so the best choice may be problem-dependent; see [Ber17] 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 this 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

Let $\{1, 2, \ldots, N, t\}$ be the set of nodes of a graph, and let a_{ij} be the cost of moving from node i to node j [also referred to as the *length* of the arc (i, j) that joins i and j]. Node t is a special node, which we call the *destination*. 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 $(i, j_1), (j_1, j_2), \ldots, (j_k, i)$ 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. 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.

We can formulate this problem as a deterministic DP problem with N stages, where the states at any stage $0, \ldots, N-1$ are the nodes $\{1, \ldots, N\}$, 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 $i = 1, \ldots, N$, $k = 0, \ldots, N-1$. The cost of the optimal path from i to t is $J_0(i)$. The DP algorithm takes the intuitively clear form

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

or

$$J_k(i) = \min_{\text{All arcs } (i,j)} \big[a_{ij} + J_{k+1}(j)\big], \qquad k = 0, 1, \ldots, N-2,$$

with

$$J_{N-1}(i) = a_{it}, \qquad i = 1, 2, \ldots, N.$$

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.

Note that if for some $k > 0$, we have $J_k(i) = J_{k+1}(i)$ 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 formulated as DP problems by breaking down each feasible solution into a sequence of decisions/controls. This formulation will often lead to an intractable DP computation because

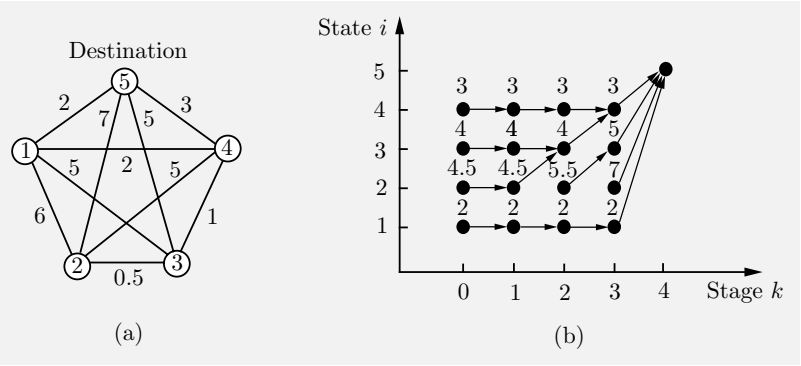

Figure 1.3.1 (a) Shortest path problem data. The destination is node 5. Arc lengths are equal in both directions and are shown along the line segments connecting nodes. (b) Costs-to-go generated by the DP algorithm. The number along stage k and state i is $J_k(i)$. Arrows indicate the optimal moves at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

$$1 \to 5, \quad 2 \to 3 \to 4 \to 5, \quad 3 \to 4 \to 5, \quad 4 \to 5.$$

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 means of an example and then we 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. 1.3.2). 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 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.

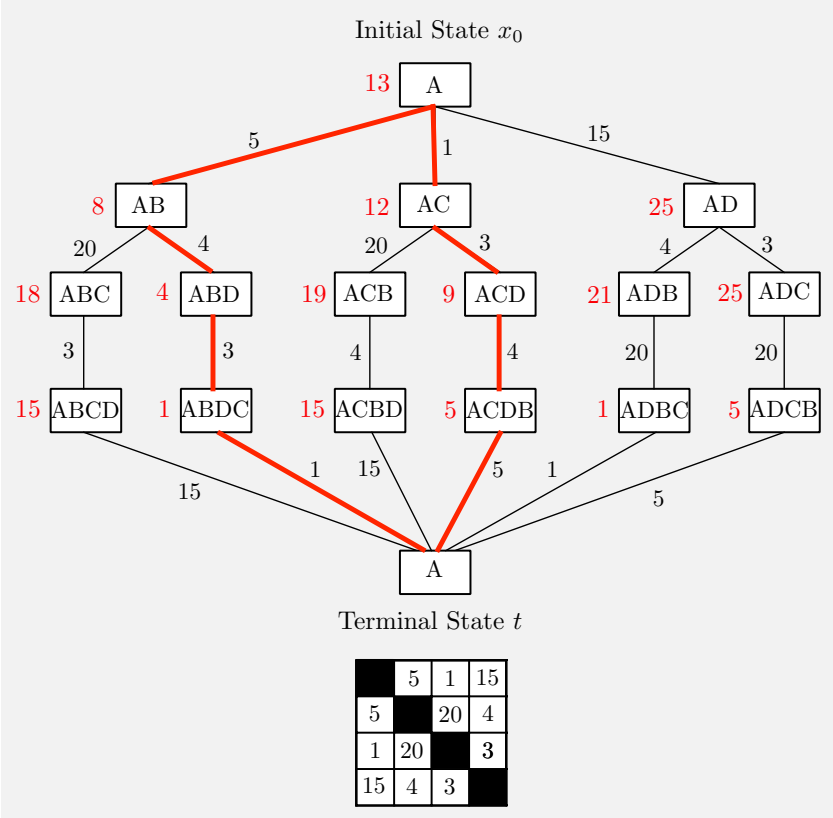

Figure 1.3.2 Example of a DP formulation of the traveling salesman problem. The travel times between the four cities A, B, C, and D are shown in the table. We form a graph whose nodes are the k-city sequences and correspond to the states of the kth stage. The transition costs/travel times are shown next 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.

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 $u = (u_1, \ldots, u_N)$, where N is a positive integer. We can then view the

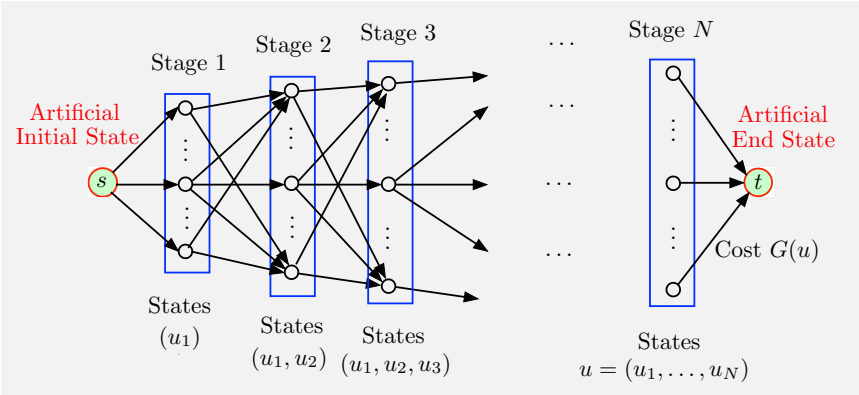

Figure 1.3.3. Formulation of a discrete optimization problem as a DP problem with $N+1$ stages. There is a cost $G(u)$ only at the terminal stage on the arc connecting an N-solution $u = (u_1, \ldots, u_N)$ to the artificial terminal state. Alternative formulations may use fewer states by taking advantage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

problem as a sequential decision problem, where the components $u_1, \ldots, u_N$ are selected one-at-a-time. A k-tuple $(u_1, \ldots, u_k)$ consisting of the first k components of a solution is called an *k-solution*. We associate k-solutions with the kth stage of the finite horizon DP problem shown in Fig. 1.3.3. In particular, for $k = 1, \ldots, N$, we view as the states of the kth stage all the k-tuples $(u_1, \ldots, u_k)$. 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

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

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

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

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

At state $(u_1, \ldots, u_k)$ we must choose u_{k+1} from the set $U_{k+1}(u_1, \ldots, u_k)$. These are the choices of u_{k+1} that are consistent with the preceding choices $u_1, \ldots, u_k$, and are also consistent with feasibility. The terminal states correspond to the N-solutions $u = (u_1, \ldots, u_N)$, and the only nonzero cost is the terminal cost $G(u)$. This terminal cost is incurred upon transition from u to an artificial end state; see Fig. 1.3.3.

Let $J_k^*(u_1, \ldots, u_k)$ denote the optimal cost starting from the k-solution $(u_1, \ldots, u_k)$, i.e., the optimal cost of the problem over solutions whose first

k components are constrained to be equal to u_i, $i = 1, \ldots, k$, respectively. The DP algorithm is described by the equation

$$J_k^*(u_1, \ldots, u_k) = \min_{u_{k+1} \in U_{k+1}(u_1, \ldots, u_k)} J_{k+1}^*(u_1, \ldots, u_k, u_{k+1}), \qquad (1.14)$$

with the terminal condition

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

The algorithm (1.14) 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 $(u_1^*, \ldots, u_N^*)$ is then constructed by going forward through the algorithm

$$u_{k+1}^* \in \arg \min_{u_{k+1} \in U_{k+1}(u_1^*, \ldots, u_k^*)} J_{k+1}^*(u_1^*, \ldots, u_k^*, u_{k+1}), \quad k = 0, \ldots, N-1, \qquad (1.15)$$

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 (1.15) 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. (1.15). One possibility is to use as

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

the cost generated by a heuristic method that solves the problem suboptimally with the values of the first $k+1$ decision components fixed at $u_1^*, \ldots, u_k^*, u_{k+1}$. 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 4 for infinite horizon problems, where it will be related to the method of policy iteration.

1.3.3 Problems with a Terminal State

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

$$g_k(t, u_k, w_k) = 0, \quad f_k(t, u_k, w_k) = t, \qquad \text{for all } u_k \in U_k(t),\ k = 0, 1, \ldots.$$

Thus the control process 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 transition from another state.

Generally, when it is known that an optimal policy will reach the terminal 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 terminal 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 example is 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 terminal 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 4×4 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. 1.3.4) 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 4×4 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

† 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.

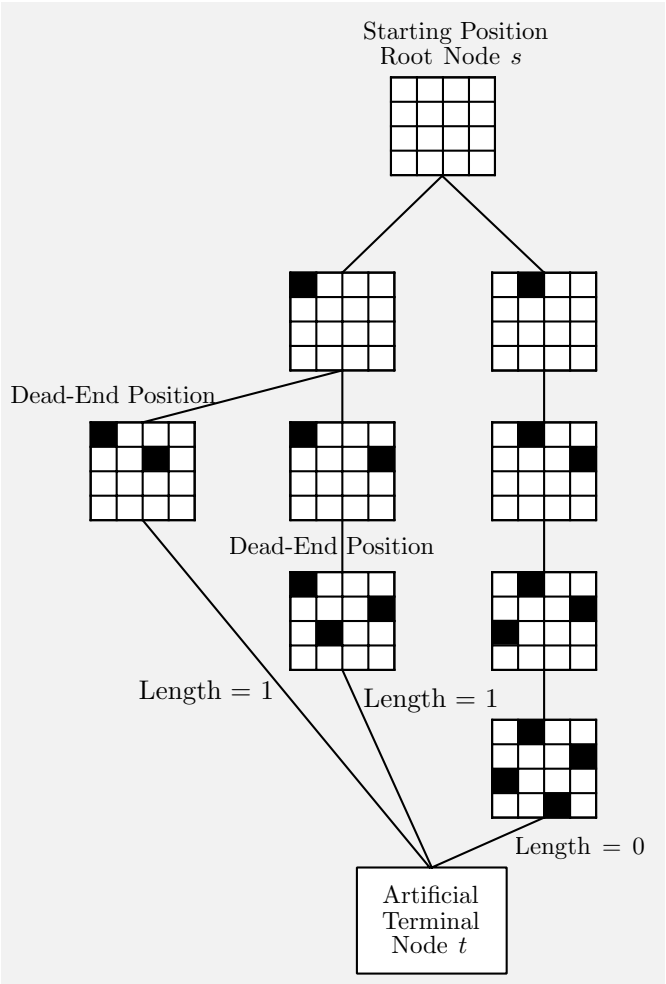

Figure 1.3.4 Discrete optimization formulation of the four queens problem. Symmetric positions resulting from placing a queen in one of the rightmost squares in the top row have been ignored. Squares containing a queen have been darkened. All arcs have length zero except for those connecting dead-end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

row is $8^8 = 16,777,216$). It can be shown that there exist solutions to this problem for all $N \geq 4$.

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 [Fe10]).

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 w_k and possibly of 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 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 into the basic problem format. Note that the control applied depends on both the current state and the current forecast. The DP algorithm takes the form

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

$$J_k(x_k, y_k) = \min_{u_k \in U_k(x_k)} \underset{w_k}{E} \Big\{ g_k(x_k, u_k, w_k) + \sum_{i=1}^{m} p_i J_{k+1}\big(f_k(x_k, u_k, w_k), i\big) \mid y_k \Big\}, \tag{1.16}$$

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 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, and a garage at the end. 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 garage without having parked, 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 $N = n + 1$ stages, corresponding to the parking spaces (including the garage), and an artificial terminal state t that corresponds to having parked; see Fig. 1.3.5. At each stage $k = 0, \ldots, N-1$, in addition to t, we have two states (k, F) and $(k, \overline{F})$, corresponding to space k being free or taken, respectively. The decision/control is to park or continue at state (k, F) [there is no choice at states $(k, \overline{F})$ and the garage].

Let us now derive the form of 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.

$J_N^*(t) = C$: The cost-to-go upon arrival at the garage.

$J_k^*(t) = 0$: The terminal cost-to-go.

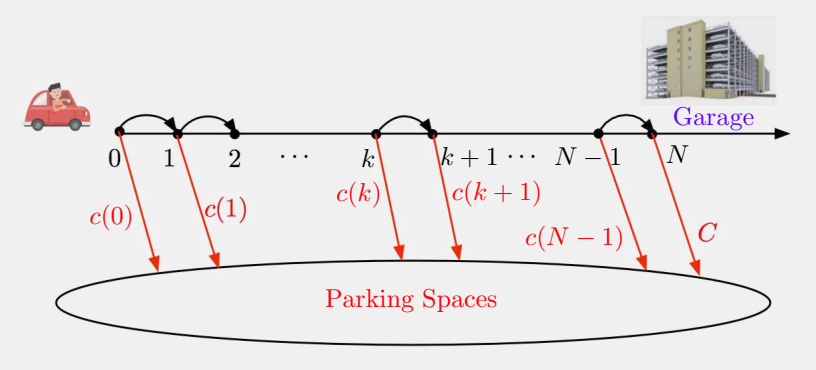

Figure 1.3.5 Cost structure of the parking problem. The driver may park at space $k = 0, 1, \ldots, N-1$ at cost $c(k)$, if the space is free, or continue to the next space $k+1$ at no cost. At space N (the garage) the driver must park at cost C.

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

$$J_k^*(F) = \begin{cases} \min\big[c(k),\, p(k)J_{k+1}^*(F) + \big(1-p(k)\big)J_{k+1}^*(\overline{F})\big] & \text{if } k = 0, \ldots, N-2, \\ \min\big[c(N-1),\, C\big] & \text{if } k = N-1, \end{cases}$$

$$J_k^*(\overline{F}) = \begin{cases} p(k)J_{k+1}^*(F) + \big(1-p(k)\big)J_{k+1}^*(\overline{F}) & \text{if } k = 0, \ldots, N-2, \\ C & \text{if } k = N-1, \end{cases}$$

(we omit here the obvious equations for the terminal state t and the garage state 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

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

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

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

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

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

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

$$p(k) \equiv 0.05, \quad c(k) = N-k, \quad C = 100, \quad N = 200. \tag{1.17}$$

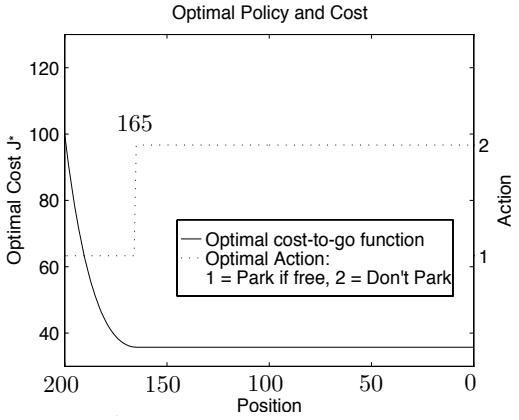

Figure 1.3.6 Optimal cost-to-go and optimal policy for the parking problem with the data in Eq. (1.17). The optimal policy is to travel from space 0 to space 165 and then to park at the first available space.

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 assuming that $c(k)$ is monotonically decreasing with k.

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 probability distribution $P_k(w_k \mid x_k, y_k, u_k)$ is given. The evolution of the other component, y_k, is governed by a given conditional distribution $P_k(y_k \mid x_k)$ 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.

We will formulate a DP algorithm that is executed over the controllable component of the state, with the dependence on the uncontrollable component being "averaged out" similar to 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

$$\hat{J}_k(x_k) = \mathop{E}_{y_k}\big\{ J_k(x_k, y_k) \mid x_k \big\}.$$

We will derive a DP algorithm that generates $\hat{J}_k(x_k)$.

Indeed, we have

$$\begin{aligned}\hat{J}_k(x_k) &= E_{y_k}\big\{J_k(x_k,y_k)\mid x_k\big\}\\ &= E_{y_k}\Big\{\min_{u_k\in U_k(x_k,y_k)} E_{w_k,x_{k+1},y_{k+1}}\big\{g_k(x_k,y_k,u_k,w_k)\\ &\qquad + J_{k+1}(x_{k+1},y_{k+1})\ \big|\ x_k,y_k,u_k\big\}\ \big|\ x_k\Big\}\\ &= E_{y_k}\Big\{\min_{u_k\in U_k(x_k,y_k)} E_{w_k,x_{k+1}}\Big\{g_k(x_k,y_k,u_k,w_k)\\ &\qquad + E_{y_{k+1}}\big\{J_{k+1}(x_{k+1},y_{k+1})\ \big|\ x_{k+1}\big\}\ \big|\ x_k,y_k,u_k\Big\}\ \big|\ x_k\Big\},\end{aligned}$$

and finally

$$\begin{aligned}\hat{J}_k(x_k) = \mathop{E}_{y_k}\Big\{\min_{u_k\in U_k(x_k,y_k)} \mathop{E}_{w_k}\Big\{g_k(x_k,y_k,u_k,w_k)\\ + \hat{J}_{k+1}\big(f_k(x_k,y_k,u_k,w_k)\big)\Big\}\Big|\ x_k\Big\}.\end{aligned}\tag{1.18}$$

The advantage of this equivalent DP algorithm is that it is executed over a significantly reduced state space. For example, if x_k takes n possible values and y_k takes m possible values, then DP is executed over n states instead of nm states. Note, however, that the minimization in the right-hand side of the preceding equation yields an optimal control law as a function of the full state (x_k, y_k).

As an example, consider the augmented state resulting from the incorporation of forecasts, as described earlier in Section 1.3.5. Then, the forecast y_k represents an uncontrolled state component, so that the DP algorithm can be simplified as in Eq. (1.18). In particular, using the notation of Section 1.3.5, by defining

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

and

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

we have, using Eq. (1.16),

$$\begin{aligned}\hat{J}_k(x_k) = \sum_{i=1}^{m} p_i \min_{u_k\in U_k(x_k)} \mathop{E}_{w_k}\Big\{g_k(x_k,u_k,w_k)\\ + \hat{J}_{k+1}\big(f_k(x_k,u_k,w_k)\big)\ \big|\ y_k=i\Big\},\end{aligned}$$

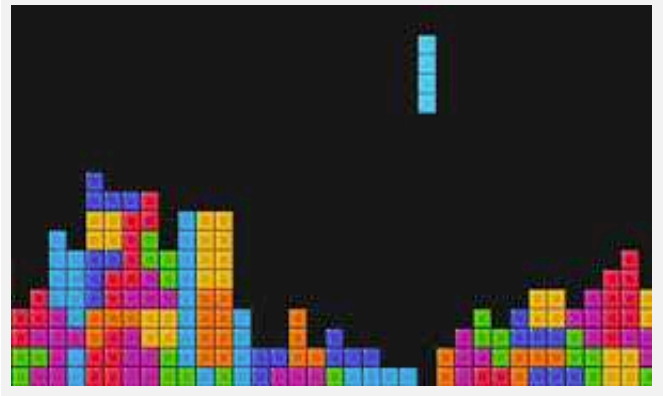

Figure 1.3.7 Illustration of a tetris board.

which is executed over the space of x_k rather than x_k *and* y_k. This is a simpler algorithm than the one of Eq. (1.16).

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. (1.18). Here is another example of similar 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 (1.18) is executed over the space of x and has the intuitive form

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

where

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

$f(x,y,u)$ is the 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 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 $\mu_k(x_k)$, $k = 0, \ldots, N-1$. 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 in the optimization and artificial intelligence literature (see e.g., [Ber17], Ch. 4). Even though there are DP algorithms for partial information problems, these algorithms are far more computationally intensive than in the perfect information case. 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

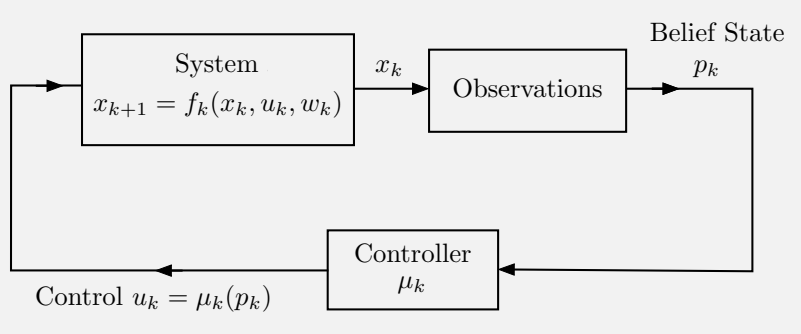

Figure 1.3.8 Schematic illustration of a control system with imperfect state observations. The belief state p_k is the conditional probability distribution of x_k given all the observations up to time k.

can reduce a partial state information problem to one with perfect state 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 β, 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

$$p_{k+1} = \begin{cases} p_k & \text{if the site is not searched at time } k, \\ 0 & \text{if the site is searched and a treasure is found,} \\ \frac{p_k(1-\beta)}{p_k(1-\beta)+1-p_k} & \text{if the site is searched but no treasure is found.} \end{cases} \tag{1.19}$$

The third relation above follows by application of Bayes' rule (p_{k+1} is equal to the kth 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. (1.19), and write a DP algorithm, assuming that the treasure's worth is V, that each search costs C, and that once we decide not to search at a particular time, then we cannot search at future times. The algorithm takes the form

$$J_k(p_k) = \max \left[J_{k+1}(p_k), \right. \\ \left. -C + p_k \beta V + (1 - p_k \beta) \hat{J}_{k+1} \left(\frac{p_k(1-\beta)}{p_k(1-\beta) + 1 - p_k} \right) \right], \tag{1.20}$$

with $\hat{J}_N(p_N) = 0$.

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

$$\hat{J}_k(p_k) = 0 \qquad \text{if} \qquad p_k \leq \frac{C}{\beta V}.$$

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 (1.20) would have to be adapted to the discretization.

In problems where the state x_k can take many values, say n, the belief state takes values in an n-dimensional simplex, so discretization becomes problematic for large n. 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 is complicated, and a solution by exact DP is 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

way to his destination, along a line of L 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 cost β_i^- or can move to $i+1$ at cost β_i^+. Moreover, the driver records the free/taken status of the spaces previously visited and may return to any of these spaces.

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 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

$$\big(p(1),\ldots,p(L)\big),$$

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 belongs to the unit simplex of L-dimensional vectors and can be perfectly computed 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 simplex 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.

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. 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

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

† 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 finite horizon problem by restricting the number of moves to a given upper limit, say $N > L$, and requiring that if the driver is at distance of k spaces from the garage at time $N-k$, then driving in the direction away from the garage is not an option.

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)]

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

Substituting the previous form of J_2, we obtain

$$J_1(x_1) = \min_{u_1}\Big[u_1^2 + r\big((1-a)x_1 + au_1 - T\big)^2\Big]. \tag{1.21}$$

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

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

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

$$\mu_1^*(x_1) = \frac{ra\big(T - (1-a)x_1\big)}{1 + ra^2}. \tag{1.22}$$

By substituting the optimal u_1 in the expression (1.21) for J_1, we obtain

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

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

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

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

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

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

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

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

$$\mu_0^*(x_0) = \frac{r(1-a)a\big(T - (1-a)^2x_0\big)}{1+ra^2\big(1+(1-a)^2\big)}. \tag{1.23}$$

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

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

This completes the solution of the problem.

Note that the algorithm has simultaneously yielded an optimal policy $\{\mu_0^*, \mu_1^*\}$ via Eqs. (1.23) and (1.22): 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

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

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

$$\begin{aligned} J_1(x_1) &= \min_{u_1} \mathop{E}_{w_1} \Big\{ u_1^2 + r\big((1-a)x_1 + au_1 + w_1 - T\big)^2 \Big\} \\ &= \min_{u_1} \Big[u_1^2 + r\big((1-a)x_1 + au_1 - T\big)^2 \\ &\qquad + 2rE\{w_1\}\big((1-a)x_1 + au_1 - T\big) + rE\{w_1^2\} \Big]. \end{aligned}$$

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

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

Comparing this equation with Eq. (1.21), 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).

1.4 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 decision making over time 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 the 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 notation and 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 selected terms commonly used in RL, and their optimal control counterparts.

(a) **Agent** = Controller or decision maker.

(b) **Action** = Control.

(c) **Environment** = System.

(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 (or state-value) function** = (Opposite of) Cost function.

(g) **Maximizing the value function** = Minimizing the cost function.

(h) **Action (or state-action) value** = Q-factor of a state-control pair.

(i) **Planning** = Solving a DP problem with a known mathematical model.

(j) **Learning** = Solving a DP problem in model-free fashion.

(k) **Self-learning** (or self-play in the context of games) = Solving a DP problem using 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** = Aggregation.

(p) **Episodic task or episode** = Finite-step system trajectory.

(q) **Continuing task** = Infinite-step system trajectory.

(r) **Backup** = Applying the DP operator at some state.

(s) **Sweep** = Applying the DP operator at all states.

(t) **Greedy policy with respect to a cost function** J = Minimizing policy in the DP expression defined by J.

(u) **Afterstate** = Post-decision state.

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.

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 textbooks [Ber12], [Ber17] provide an extensive discussion of exact DP and its applications. The mathematical aspects of exact DP are discussed in the monograph by Bertsekas and Shreve [BeS78], particularly the fine 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, based on the strong connections of the subject with fixed point theory.

The approximate DP literature has expanded tremendously since the connections between DP and RL became apparent in the late 80s and early 90s. We will restrict ourselves to mentioning textbooks, research monographs, and broad surveys, which supplement our discussions and collectively provide a guide to the literature. Thus the author wishes to apologize in advance for the many omissions of references from the research literature.

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.

More recent books are the 2003 book by Gosavi (a much expanded 2nd edition [Gos15] appeared in 2015), which emphasizes simulation-based optimization and RL algorithms, Cao [Cao07], which emphasizes a sensitivity approach to simulation-based methods, Chang, Fu, Hu, and Marcus [CFH07], 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, and Vrabie, Vamvoudakis, and Lewis [VVL13], which discusses neural network-based methods and

continuous-time optimal control applications. 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.

Several survey papers in the volumes by Si, Barto, Powell, and Wunsch [SBP04], and Lewis and Liu [LeL12], and the special issue by Lewis, Liu, and Lendaris [LLL08] describe approximation methodology that we we will not be covering in this book: linear programming-based 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]). The volume by White and Sofge [WhS92] contains several surveys that describe early work in the field. Influential surveys were written, from an artificial intelligence viewpoint, by Barto, Bradtke, and Singh [BBS95], and by Kaelbling, Littman, and Moore [KLM96]. More recent surveys are Borkar [Bor09] (a methodological point of view that explores connections with other Monte Carlo schemes), Lewis and Vrabie [LeV09] (a control theory point of view), Werbos [Web09] (which reviews potential connections between brain intelligence, neural networks, and DP), Szepesvari [Sze10] (which provides a detailed description of approximation in value space from a RL point of view), Browne et al. [BPW12] (which focuses on Monte Carlo Tree Search), Grondman et al. [GBL12] (which focuses on policy gradient methods), and the author's [Ber05a] (which focuses on rollout algorithms and model predictive control), [Ber10a] (which focuses on approximate policy iteration), and [Ber18b] (which focuses on aggregation methods).