Reinforcement Learning and Optimal Control

by

Dimitri P. Bertsekas

2019

Chapter 1
Exact Dynamic Programming

SELECTED SECTIONS

Athena Scientific, Belmont, Massachusetts

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 Chapters 4-6. We also discuss separately deterministic and stochastic problems (Sections 1.1 and 1.2, respectively). The reason is that deterministic problems are simpler and lend themselves better as an entry point to the optimal control methodology. Moreover, they have some favorable characteristics, which allow the application of a broader variety of methods. For example, simulation-based methods are greatly simplified and sometimes better understood in the context of deterministic optimal control.

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

1.1 DETERMINISTIC DYNAMIC PROGRAMMING

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

1.1.1 Deterministic Problems

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

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

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,

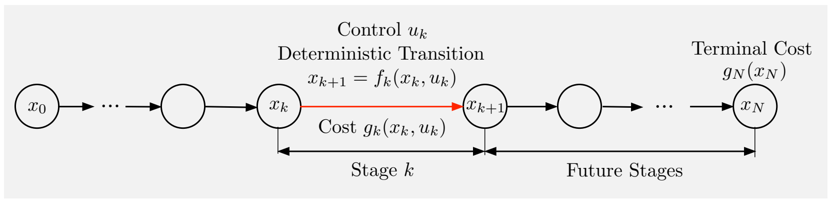

Figure 1.1.1 Illustration of a deterministic N-stage optimal control problem. Starting from state x_k, the next state under control u_k is generated nonrandomly, according to

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

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 or number of times control is applied,

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

The problem also involves a cost function that is additive in the sense that the cost incurred at time k, denoted by $g_k(x_k, u_k)$, accumulates over time. Formally, g_k is a function of (x_k, u_k) that takes real number values, and may depend on k. For a given initial state x_0, the total cost of a control sequence $\{u_0, \ldots, u_{N-1}\}$ 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. The total 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. Figure 1.1.1 illustrates the main elements of the problem.

We will next illustrate deterministic problems with some examples.

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

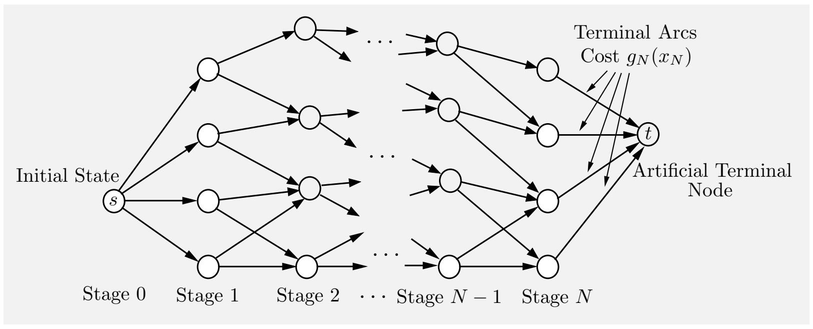

Figure 1.1.2 Transition graph for a deterministic finite-state system. Nodes correspond to states x_k. Arcs correspond to state-control pairs (x_k, u_k). An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$, respectively. We view the cost $g_k(x_k, u_k)$ of the transition as the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

Discrete Optimal Control Problems

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

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

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

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

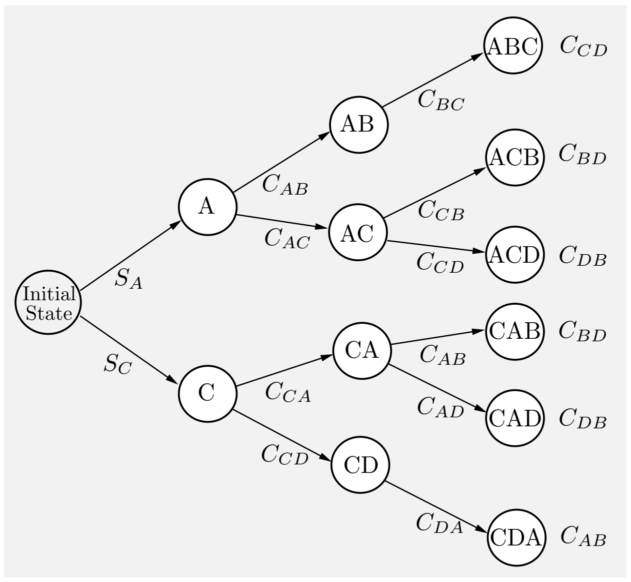

Figure 1.1.3 The transition graph of the deterministic scheduling problem of Example 1.1.1. Each arc of the graph corresponds to a decision leading from some state (the start node of the arc) to some other state (the end node of the arc). The corresponding cost is shown next to the arc. The cost of the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

Example 1.1.1 (A Deterministic Scheduling Problem)

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

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

We can view this problem as a sequence of three decisions, namely the choice of the first three operations to be performed (the last operation is determined from the preceding three). It is appropriate to consider as state the set of operations already performed, the initial state being an artificial state corresponding to the beginning of the decision process. The possible state transitions corresponding to the possible states and decisions for this problem are shown in Fig. 1.1.3. Here the problem is deterministic, i.e., at a given state, each choice of control leads to a uniquely determined state. For example, at state AC the decision to perform operation D leads to state ACD with certainty, and has cost C_{CD}. Thus the problem can be conveniently represented with the transition graph of Fig. 1.1.3. The optimal solution corresponds to the path that starts at the initial state and ends at some state at the terminal time and has minimum sum of arc costs plus the terminal cost.

Continuous-Spaces Optimal Control Problems

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

Example 1.1.2 (A Linear-Quadratic Problem)

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

x_0: initial temperature of the material,

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

u_{k-1}, $k = 1, \ldots, N$: heat energy applied to the material in oven k. In practice there will be some constraints on u_k, such as nonnegativity. However, for analytical tractability one may also consider the case where u_k is unconstrained, and check later if the solution satisfies some natural restrictions in the problem at hand.

We assume a system equation of the form

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

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 that trades off approaching the final temperature with the expended energy.

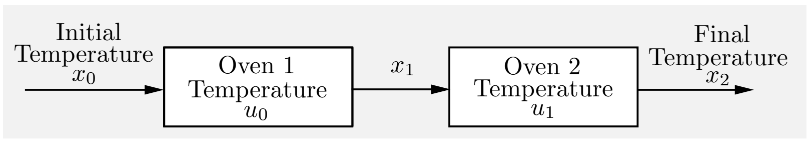

Figure 1.1.4 The linear-quadratic problem of Example 1.1.2 for $N = 2$. The temperature of the material evolves according to the system equation $x_{k+1} = (1-a)x_k + au_k$, where a is some scalar with $0 < a < 1$.

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

1.1.2 The Dynamic Programming Algorithm

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

Principle of Optimality

Let $\{u_0^*, \ldots, u_{N-1}^*\}$ 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

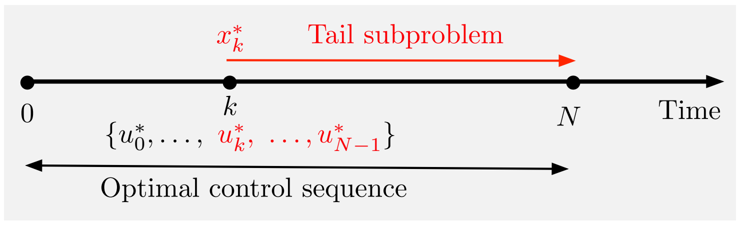

Figure 1.1.5 Illustration of the principle of optimality. The tail $\{u_k^*, \ldots, u_{N-1}^*\}$ of an optimal sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ is optimal for the tail subproblem that starts at the state x_k^* of the optimal trajectory $\{x_1^*, \ldots, x_N^*\}$.

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 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 Los Angeles to Boston passes through Chicago. The principle of optimality translates to the obvious fact that the Chicago to Boston portion of the route is also the fastest route for a trip that starts from Chicago and ends in Boston.

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

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

Example 1.1.1 (Scheduling Problem - Continued)

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

According to the principle of optimality, the "tail" portion of an optimal schedule must be optimal. For example, suppose that the optimal schedule is CABD. Then, having scheduled first C and then A, it must be optimal to complete the schedule with BD rather than with DB. With this in mind, we solve all possible tail subproblems of length two, then all tail subproblems of length three, and finally the original problem that has length four (the subproblems of length one are of course trivial because there is only one operation that is as yet unscheduled). As we will see shortly, the tail subproblems of length $k+1$ are easily solved once we have solved the tail subproblems of length k, and this is the essence of the DP technique.

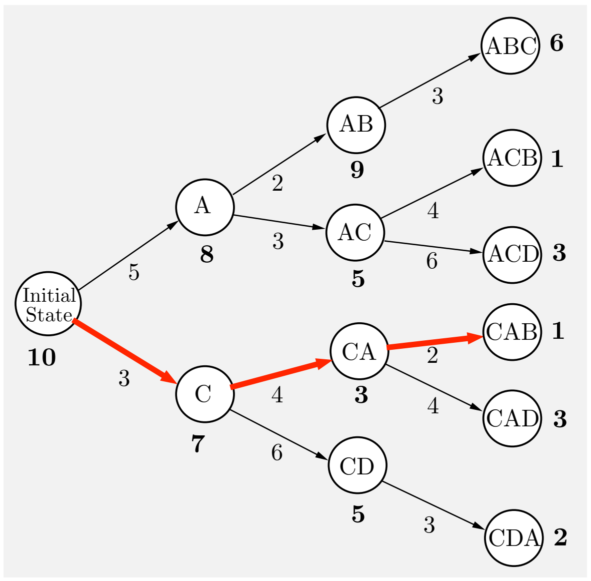

Figure 1.1.6 Transition graph of the deterministic scheduling problem, with the cost of each decision shown next to the corresponding arc. Next to each node/state we show the cost to optimally complete the schedule starting from that state. This is the optimal cost of the corresponding tail subproblem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

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

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

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

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

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

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

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

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

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

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

Finding an Optimal Control Sequence by DP

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

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

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 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

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

where

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

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

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

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

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

Note that the algorithm solves every tail subproblem, i.e., the minimization of the cost accumulated additively starting from an intermediate state up to the end of the horizon. Once the functions $J_0^*, \ldots, J_N^*$ 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

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

and

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

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

1.1.3 Approximation in Value Space

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

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

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

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

and

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

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

Q-Factors and Q-Learning

The expression

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

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

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

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

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

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

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

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

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

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

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

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

1.4 REINFORCEMENT LEARNING AND OPTIMAL CONTROL - SOME TERMINOLOGY

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

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

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

(a) **Environment** = System.

(b) **Agent** = Decision maker or controller.

(c) **Action** = Decision or control.

(d) **Reward of a stage** = (Opposite of) Cost of a stage.

(e) **State value** = (Opposite of) Cost starting from a state.

(f) **Value (or reward) function** = (Opposite of) Cost function.

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

(h) **Action (or state-action) value** = Q-factor (or Q-value) of a state-control pair. (Q-value is also used often in RL.)

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

(j) **Learning** = Solving a DP problem without using an explicit mathematical model. (This is the principal meaning of the term "learning" in RL. Other meanings are also common.)

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

(l) **Deep reinforcement learning** = Approximate DP using value and/or policy approximation with deep neural networks.

(m) **Prediction** = Policy evaluation.

(n) **Generalized policy iteration** = Optimistic policy iteration.

(o) **State abstraction** = State aggregation.

(p) **Temporal abstraction** = Time aggregation.

(q) **Learning a model** = System identification.

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

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

(t) **Experience replay** = Reuse of samples in a simulation process.

(u) **Bellman operator** = DP mapping or operator.

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

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

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

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

(z) **Ground truth** = Empirical evidence or information provided by direct observation.

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

Notation

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

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

(a) x: state.

(b) u: control.

(c) J: cost function.

(d) g: cost per stage.

(e) f: system function.

(f) i: discrete state.

(g) $p_{ij}(u)$: transition probability from state i to state j under control u.

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

1.5 NOTES AND SOURCES

Our discussion of exact DP in this chapter has been brief since our focus in this book will be on approximate DP and RL. The author's DP 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 the present book. The books by Puterman [Put94] and by the author [Ber12] provide detailed treatments of infinite horizon finite-state Markovian decision problems. Continuous spaces infinite horizon problems are covered in the author's book [Ber12], while some of the more complex mathematical aspects of exact DP are discussed in the monograph by Bertsekas and Shreve [BeS78] (particularly the probabilistic/measure-theoretic issues associated with stochastic optimal control).

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

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

Two books were written on our subject in the 1990s, setting the tone for subsequent developments in the field. One in 1996 by Bertsekas and Tsitsiklis [BeT96], which reflects a decision, control, and optimization viewpoint, and another in 1998 by Sutton and Barto, which reflects an artificial intelligence viewpoint (a 2nd edition, [SuB18], was published in 2018). We refer to the former book and also to the author's DP textbooks [Ber12], [Ber17] for a broader discussion of some of the topics of the present book, including algorithmic convergence issues and additional DP models, such as those based on average cost optimization. For historical accounts of the early development of the subject, see [BeT96], Section 6.7, and [SuB18], Section 1.7.

More recent books are by Gosavi [Gos15] (a much expanded 2nd edition of his 2003 monograph), which emphasizes simulation-based optimization and RL algorithms, Cao [Cao07], which focuses on a sensitivity approach to simulation-based methods, Chang, Fu, Hu, and Marcus [CFH13] (a 2nd edition of their 2007 book), 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, Vrabie, Vamvoudakis, and Lewis [VVL13], which discusses neural network-based methods, on-line adaptive control, and continuous-time optimal control applications, Kochenderfer et al. [KAC15], which selectively discusses applications and approximations in DP and the treatment of uncertainty, Jiang and Jiang [JiJ17], which develops adaptive control within an approximate DP framework, and Liu et al. [LWW17], which deals with forms of adaptive dynamic programming, and topics in both RL and optimal control. The book by Krishnamurthy [Kri16] focuses on partial state information problems, with discussion of both exact DP, and approximate DP/RL methods. The book by Haykin [Hay08] discusses approximate DP in the broader context of neural network-related subjects. The book by Borkar [Bor08] is an advanced monograph that addresses rigorously many of the convergence issues of iterative stochastic algorithms in approximate DP, mainly using the so called ODE approach. The book by Meyn [Mey07] is broader in its coverage, but touches upon some of the approximate DP algorithms that we discuss.

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

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

Recent extended surveys and short monographs are Borkar [Bor09] (a methodological point of view that explores connections with other Monte Carlo schemes), Lewis and Vrabie [LeV09] (a control theory point of view), Szepesvari [Sze10] (which discusses approximation in value space from a RL point of view), Deisenroth, Neumann, and Peters [DNP11], and Grondman et al. [GBL12] (which focus on policy gradient methods), Browne et al. [BPW12] (which focuses on Monte Carlo Tree Search), Mausam and Kolobov [MaK12] (which deals with Markovian decision problems from an artificial intelligence viewpoint), Schmidhuber [Sch15], Arulkumaran et al. [ADB17], Li [Li17], Busoniu et al. [BDT18], and Caterini and Chang [CaC18] (which deal with reinforcement learning schemes that are based on the use of deep neural networks), the author's [Ber05a] (which focuses on rollout algorithms and model predictive control), [Ber11a] (which focuses on approximate policy iteration), and [Ber18b] (which focuses on aggregation methods), and Recht [Rec18a] (which focuses on continuous spaces optimal control).