Reinforcement Learning and Optimal Control

by

Dimitri P. Bertsekas

Massachusetts Institute of Technology

Chapter 4
Infinite Horizon Reinforcement Learning
DRAFT

This is Chapter 4 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

4

Infinite Horizon Reinforcement Learning

Contents

In this chapter, we first provide an introduction to the theory of infinite horizon problems, and then consider the use of approximate DP/RL methods for suboptimal solution. Infinite horizon problems differ from their finite horizon counterparts in two main respects:

(a) The number of stages is infinite.

(b) The system is stationary, i.e., the system equation, the cost per stage, and the random disturbance statistics do not change from one stage to the next.

The assumption of an infinite number of stages is never satisfied in practice, but is a reasonable approximation for problems involving a finite but very large number of stages. The assumption of stationarity is often satisfied in practice, and in other cases it approximates well a situation where the system parameters vary relatively slowly with time.

Infinite horizon problems give rise to elegant and insightful analysis, and the implementation of their optimal policies is often simple. For example, optimal policies are typically stationary, i.e., the optimal rule for choosing controls does not change from one stage to the next.

On the other hand, infinite horizon problems generally require a more sophisticated mathematical treatment. Our discussion will be limited to relatively simple finite-state problems. Still some theoretical results will be needed in this chapter. They will be explained intuitively to the extent possible, and their mathematical proofs will be provided in an end-of-chapter appendix.

4.1 AN OVERVIEW OF INFINITE HORIZON PROBLEMS

We will focus on two types of infinite horizon problems, where we aim to minimize the total cost over an infinite number of stages, given by

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

Here, $J_\pi(x_0)$ denotes the cost associated with an initial state x_0 and a policy $\pi = \{\mu_0, \mu_1, \ldots\}$, and α is a positive scalar. We will assume that $\alpha = 1$ in SSP problems, and $\alpha < 1$ in discounted problems. The meaning of $\alpha < 1$ is that future costs matter to us less than the same costs incurred at the present time.

Thus the infinite horizon costs of a policy is the limit of its finite horizon costs as the horizon tends to infinity. (We assume that the limit exists for the moment, and address the issue later.) The two types of problems, considered in Sections 4.2 and 4.3, respectively, are:

(a) *Stochastic shortest path problems* (SSP for short). Here, $\alpha = 1$ but there is a special cost-free termination state; once the system reaches

that state it remains there at no further cost. We will assume a problem structure such that termination is inevitable. Thus the horizon is in effect finite, but its length is random and may be affected by the policy being used.

(b) *Discounted problems*. Here, $\alpha < 1$ and there need not be a termination state. However, we will see that a discounted problem can be readily converted to an SSP problem. This can be done by introducing an artificial termination state to which the system moves with probability $1 - \alpha$ at every stage, thus making termination inevitable. As a result, our algorithms and analysis for SSP problems can be easily adapted to discounted problems.

A Preview of Infinite Horizon Theory

There are several analytical and computational issues regarding our infinite horizon problems. Many of them revolve around the relation between the optimal cost-to-go function J^* of the infinite horizon problem and the optimal cost-to-go functions of the corresponding N-stage problems. In particular, consider the SSP case and let $J_N(x)$ denote the optimal cost of the problem involving N stages, initial state x, cost per stage $g(x, u, w)$, and zero terminal cost. This cost is generated after N iterations of the DP algorithm

$$J_{k+1}(x) = \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x, u, w) + J_k\big(f(x, u, w)\big) \Big\}, \qquad k = 0, 1, \ldots, \quad (4.1)$$

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

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

$$J^*(x) = \lim_{N \to \infty} J_N(x) \quad (4.2)$$

for all states x.

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

$$J^*(x) = \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x, u, w) + J^*\big(f(x, u, w)\big) \Big\}. \quad (4.3)$$

† This is just the finite horizon DP algorithm of Chapter 1. However, we have reversed the time indexing to suit our purposes. Thus the index of the cost functions produced by the algorithm is incremented with each iteration, and not decremented as in the case of finite horizon.

This is obtained by taking the limit as $N \to \infty$ in the VI algorithm (4.1) using Eq. (4.2). Equation (4.3) is really a system of equations (one equation per state x), which has as solution the costs-to-go of all the states. It can also be viewed as a *functional equation* for the optimal cost function J^*, and it is called *Bellman's equation*.

(3) If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (4.3) for each x, then the policy $\{\mu, \mu, \ldots\}$ should be optimal. This type of policy is called *stationary*. Intuitively, optimal policies can be found within this class of policies, since the future optimization problem when starting at a given state looks the same regardless of the time when we start.

All three of the preceding results hold for SSP problems under our assumptions, as we will state later in Section 4.2 and prove in the appendix to this chapter. They also hold for discounted problems in suitably modified form that incorporates the discount factor. In fact the algorithms and analysis of this chapter are quite similar for SSP and discounted problems, to the point where we may discuss a particular method for one of the two problems with the understanding that its application to the other problem can be straightforwardly adapted.

Transition Probability Notation for Infinite Horizon Problems

Throughout this chapter we assume a finite-state discrete-time dynamic system, and we will use a special transition probability notation that is suitable for such a system. We generally denote states by the symbol i and successor states by the symbol j. We will assume that there are n states (in addition to the termination state t for SSP problems). These states are denoted $1, \ldots, n$. The control u is constrained to take values in a given finite constraint set $U(i)$, which may depend on the current state i. The use of a control u at state i specifies the transition probability $p_{ij}(u)$ to the next state j, at a cost $g(i, u, j)$.†

Given an admissible policy $\pi = \{\mu_0, \mu_1, \ldots\}$ [one with $\mu^k(i) \in U(i)$ for all i and k] and an initial state i_0, the system becomes a Markov chain whose generated trajectory under π, denoted $\{i_0, i_1, \ldots\}$, has a well-defined probability distribution. The total expected cost associated with an initial

† To convert from the transition probability format to the system equation format used in the preceding chapters, we can simply use the system equation

$$x_{k+1} = w_k,$$

where w_k is the disturbance that takes values according to the transition probabilities $p_{x_k w_k}(u_k)$.

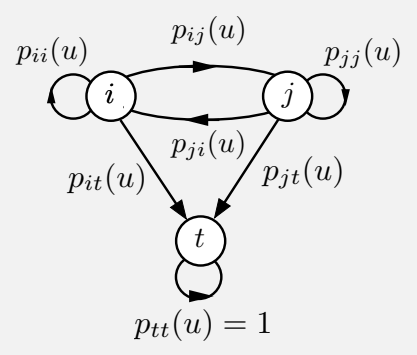

Figure 4.2.1 The transition graph of an SSP problem. There are n states, plus the termination state t, with transition probabilities $p_{ij}(u)$. The termination state is cost-free and absorbing.

state i is

$$J_\pi(i) = \lim_{N\to\infty} E\left\{ \sum_{k=0}^{N-1} \alpha^k g\big(i_k, \mu^k(i_k), i_{k+1}\big) \,\Big|\, i_0 = i,\, \pi \right\},$$

where α is either 1 (for SSP problems) or less than 1 for discounted problems. The expected value is taken with respect to the joint distribution of the states $i_1, i_2, \ldots$, conditioned on $i_0 = i$ and the use of π. The optimal cost from state i, i.e., the minimum of $J_\pi(i)$ over all policies π, is denoted by $J^*(i)$.

The cost function of a stationary policy $\pi = \{\mu, \mu, \ldots\}$ is denoted by $J_\mu(i)$. For brevity, we refer to π as the stationary policy μ. We say that μ is optimal if

$$J_\mu(i) = J^*(i) = \min_\pi J_\pi(i), \qquad \text{for all states } i.$$

As noted earlier, under our assumptions, we will show that there will always exist an optimal policy, which is stationary.

4.2 STOCHASTIC SHORTEST PATH PROBLEMS

In the SSP problem we assume that there is no discounting ($\alpha = 1$), and that *there is a special cost-free termination state* t. Once the system reaches that state, it remains there at no further cost, i.e.,

$$p_{tt}(u) = 1, \quad g(t, u, t) = 0, \qquad \text{for all } u \in U(t).$$

We denote by $1, \ldots, n$ the states other than the termination state t; see Fig. 4.2.1.

With this notation, the Bellman equation (4.3) and the VI algorithm (4.1) take the following form.

Bellman Equation and Value Iteration for SSP Problems:

For all $i = 1, \ldots, n$, we have

$$J^*(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J^*(j) \big) \right]. \tag{4.4}$$

For all $i = 1, \ldots, n$, and any initial conditions $J_0(1), \ldots, J_0(n)$, the VI algorithm generates the sequence $\{J_k\}$ according to

$$J_{k+1}(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J_k(j) \big) \right]. \tag{4.5}$$

The right-hand bracketed expression in the Bellman equation (4.4) represents an expected value, which is similar to the expectation we have seen in earlier DP expressions. It is the sum of:

(a) The contribution $p_{it}(u) g(i,u,t)$ to the expected cost of the current stage of the terminating i-to-t transition.

(b) The contribution $\sum_{j=1}^{n} p_{ij}(u) g(i,u,j)$ to the expected cost of the current stage of the nonterminating i-to-j transitions.

(c) The optimal expected cost-to-go

$$\sum_{j=1}^{n} p_{ij}(u) J^*(j)$$

starting from the next state j (if the next state is t, the corresponding optimal cost is $J^*(t) = 0$, so it does not appear in the sum).

Note that the deterministic shortest path problem of Section 1.3.1 is obtained as the special case of the SSP problem where for each state-control pair (i,u), the transition probability $p_{ij}(u)$ is equal to 1 for a unique state j that depends on (i,u). Moreover, any deterministic or stochastic finite-state, finite horizon problem with a termination state (cf. Section 1.3.3) can be converted to an SSP problem. In particular, the reader may verify that the finite-state N-step horizon problem of Chapter 1 can be obtained as a special case of an SSP problem by viewing as state the pair (x_k, k) and lumping all pairs (x_N, N) into a termination state t.

We are interested in problems where reaching the termination state is inevitable. Thus, the essence of the problem is to reach the termination state with minimum expected cost. Throughout this chapter for SSP problems, we will make the following assumption, which will be shown to guarantee eventual termination under *all* policies.†

Assumption 4.2.1: (Termination is Inevitable Under All Policies) There exists an integer m such that regardless of the policy used and the initial state, there is positive probability that the termination state will be reached after no more that m stages; i.e., for all admissible policies π we have

$$\rho_\pi = \max_{i=1,\ldots,n} P\{x_m \neq t \mid x_0 = i, \pi\} < 1.$$

Let ρ be the maximum probability of not reaching t, over all starting states and policies:

$$\rho = \max_\pi \rho_\pi.$$

Note that ρ_π depends only on the first m components of the policy π. Furthermore, since the number of controls available at each state is finite, the number of distinct m-stage policies is also finite. It follows that there can be only a finite number of distinct values of ρ_π, so that

$$\rho < 1.$$

This implies that *the probability of not reaching t over a finite horizon diminishes to 0 as the horizon becomes longer*, regardless of the starting state and policy used.

† The main analytical and algorithmic results for SSP problems are valid under more general conditions, which involve the notion of a proper policy (see the end-of-chapter references). In particular, a stationary policy is called *proper* if starting from every state, it is guaranteed to eventually reach the destination. The policy is called *improper* if it is not proper.

It can be shown that Assumption 4.2.1 is equivalent to the seemingly weaker assumption that all stationary policies are proper. However, the subsequent four propositions can also be shown under the genuinely weaker assumption that there exists at least one proper policy, and furthermore, every improper policy results in infinite expected cost from at least one initial state (see [BeT89], [BeT91], or [Ber12], Chapter 3). These assumptions, when specialized to deterministic shortest path problems, are similar to the assumptions of Section 1.3.1. They imply that there is at least one path to the destination from every starting state and that all cycles have positive cost.

To see this, note that for any π and any initial state i

$$\begin{aligned} P\{x_{2m} \neq t \mid x_0 = i, \pi\} &= P\{x_{2m} \neq t \mid x_m \neq t,\, x_0 = i, \pi\} \\ &\quad \cdot P\{x_m \neq t \mid x_0 = i, \pi\} \\ &\leq \rho^2. \end{aligned}$$

More generally, for each π, the probability of not reaching the termination state after km stages diminishes like ρ^k regardless of the initial state, i.e.,

$$P\{x_{km} \neq t \mid x_0 = i, \pi\} \leq \rho^k, \qquad i = 1, \ldots, n. \tag{4.6}$$

This fact implies that the limit defining the associated total cost vector J_π exists and is finite, and is central in the proof of the following results (given in the appendix to this chapter).

We now describe the main theoretical results for SSP problems; the proofs are given in the appendix to this chapter. Our first result is that the infinite horizon version of the DP algorithm, which is VI [cf. Eq. (4.1)], converges to the optimal cost function J^*. The optimal cost $J^*(t)$ starting from t is of course 0, so it is just neglected where appropriate in the subsequent analysis. Generally, J^* is obtained in the limit, after an infinite number of iterations. However, there are important cases where convergence is obtained in finitely many iterations (see [Ber12], Chapter 3).

Proposition 4.2.1: (Convergence of VI) Given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\{J_k(i)\}$ generated by the VI algorithm

$$J_{k+1}(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J_k(j) \big) \right],$$

converges to the optimal cost $J^*(i)$ for each $i = 1, \ldots, n$.

Our next result is that the limiting form of the DP equation, Bellman's equation, has J^* as its unique solution.

Proposition 4.2.2: (Bellman's Equation) The optimal cost function $J^* = \big(J^*(1), \ldots, J^*(n)\big)$ satisfies for all $i = 1, \ldots, n$, the equation

$$J^*(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J^*(j) \big) \right], \tag{4.7}$$

and in fact it is the unique solution of this equation.

Our next result expresses that by restricting attention to a single policy μ, we obtain a Bellman equation specific to μ, which has J_μ as its unique solution.

Proposition 4.2.3: (VI and Bellman's Equation for Policies) For any stationary policy μ, the corresponding cost function $J_\mu = \big(J_\mu(1), \ldots, J_\mu(n)\big)$ satisfies for all $i = 1, \ldots, n$ the equation

$$J_\mu(i) = p_{it}\big(\mu(i)\big)\big(g(i, \mu(i), t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + J_\mu(j)\Big),$$

and is the unique solution of this equation. Furthermore, given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\big\{J_k(i)\big\}$ generated by the VI algorithm that is specific to μ,

$$J_{k+1}(i) = p_{it}\big(\mu(i)\big)\big(g(i, \mu(i), t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + J_k(j)\Big),$$

converges to the cost $J_\mu(i)$ for each i.

Our final result provides a necessary and sufficient condition for optimality of a stationary policy.

Proposition 4.2.4: (Optimality Condition) A stationary policy μ is optimal if and only if for every state i, $\mu(i)$ attains the minimum in the Bellman equation (4.7).

We provide an example illustrating Bellman's equation.

Example 4.2.1 (Maximum Expected Time to Termination)

The case where

$$g(i, u, j) = -1, \qquad \text{for all } i,\ u \in U(i), \text{ and } j,$$

corresponds to a problem where the objective is to terminate as late as possible on the average, while the opposite of the optimal cost, $-J^*(i)$, is the maximum expected time to termination starting from state i. Under our assumptions, the optimal costs $J^*(i)$ uniquely solve Bellman's equation, which has the form

$$J^*(i) = \min_{u \in U(i)} \left[-1 + \sum_{j=1}^{n} p_{ij}(u) J^*(j)\right], \qquad i = 1, \ldots, n.$$

In the special case of a single policy μ, where there is only one control at each state, $-J_\mu(i)$ represents the expected time to reach t starting from i. This is known as the mean first passage time from i to t, and is given as the unique solution of the corresponding Bellman equation

$$J_\mu(i) = -1 + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) J_\mu(j)\big), \qquad i = 1, \ldots, n.$$

We will now provide an insightful mathematical result about SSP problems, which is proved in the appendix with the aid of the preceding example. Let us write VI compactly as the algorithm

$$J_{k+1}(i) = (TJ_k)(i),$$

where for any vector $J = \big(J(1), \ldots, J(n)\big)$, we use the notation

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J(j)\big), \qquad i = 1, \ldots, n.$$

Here T is the DP operator that maps the vector J into the vector

$$TJ = \big((TJ)(1), \ldots, (TJ)(n)\big).$$

Bellman's equation can be written in terms of this operator as the fixed point equation $J^* = TJ^*$.

The next proposition states that T is a contraction mapping, so the unique fixed point property of this mapping follows from general mathematical results about contraction mappings (see e.g., [Ber12], [Ber18a]). Moreover the contraction property provides a convergence rate estimate for VI, and is the basis for further analysis of exact and approximate methods for SSP problems (see the author's monograph [Ber18a] for a theoretical development of DP, which is based on fixed point theory and an abstract operator viewpoint).

Proposition 4.2.5: (Contraction Property of the DP Operator) The DP operator T defined by

$$(TJ)(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J(j)\big) \right], \tag{4.8}$$

for all $i = 1, \ldots, n$, and vectors $J = \big(J(1), \ldots, J(n)\big)$, is a contraction mapping with respect to the weighted norm

$$\|J\| = \max_{i=1,\ldots,n} \frac{\big|J(i)\big|}{v(i)},$$

defined by some vector $v = \big(v(1), \ldots, v(n)\big)$ with positive components. In other words, there exists a positive scalar $\rho < 1$ such that for any two n-dimensional vectors J and J', we have

$$\|TJ - TJ'\| \leq \rho \|J - J'\|.$$

The preceding contraction property provides a convergence rate estimate for VI, namely that the generated sequence $\{J_k\}$ satisfies

$$\|J_k - J^*\| \leq \rho^k \|J_0 - J^*\|.$$

This follows from the fact that J_k and J^* can be viewed as the results of the k-fold application of T to the vectors J_0 and J^*, respectively. The proof of the contraction property, given in the appendix, shows that the weights $v(i)$ and the modulus of contraction ρ are related to the maximum expected number of steps $-m^*(i)$ to reach t from i (cf. Example 4.2.1). In particular, we have

$$v(i) = -m^*(i), \qquad \rho = \max_{i=1,\ldots,n} \frac{v(i) - 1}{v(i)}.$$

Bellman Equation and Value Iteration for Q-Factors

The results just given have counterparts involving Q-factors. The optimal Q-factors are defined for all $i = 1, \ldots, n$, and $u \in U(i)$ by

$$Q^*(i,u) = p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J^*(j)\big).$$

As in the finite horizon case, $Q^*(i,u)$ can be interpreted as the cost of starting at i, using u for the first stage, and using an optimal policy afterwards. Once Q^* is computed by some method, an optimal policy μ^* can be obtained from the minimization

$$\mu^*(i) \in \arg\min_{u \in U(i)} Q^*(i,u), \qquad i = 1, \ldots, n.$$

Similarly, if approximately optimal Q-factors $\tilde{Q}(i,u)$ are obtained by some method (model-based or model-free), a suboptimal policy $\tilde{\mu}$ can be obtained from the minimization

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \tilde{Q}(i,u), \qquad i = 1, \ldots, n.$$

Our basic results relating Bellman's equation and the VI algorithm are stated as follows.

Bellman Equation and Value Iteration for Q-Factors and SSP Problems:

For all $i = 1, \ldots, n$, and $u \in U(i)$ we have

$$Q^*(i,u) = p_{it}(u)g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \min_{v \in U(j)} Q^*(j,v)\right),$$

For all $i = 1, \ldots, n$, and $u \in U(i)$, and any initial conditions $Q_0(i,u)$, the VI algorithm generates the sequence $\{Q_k\}$ according to

$$Q_{k+1}(i,u) = p_{it}(u)g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \min_{v \in U(j)} Q_k(j,v)\right).$$

Actually, the optimal Q-factors $Q^*(i,u)$ can be viewed as optimal state costs associated with a modified SSP problem, which involves a new state for each pair (i,u) with transition probabilities $p_{ij}(u)$ to the states $j = 1, \ldots, n, t$; see Fig. 4.2.2. Then the preceding Bellman equation for the optimal Q-factors, together with the Bellman equation (4.7) for the optimal costs $J^*(j)$, can be viewed as the Bellman equation for the modified SSP problem.

Temporal Differences and Cost Shaping

Bellman's equation can be written in an alternative form, which involves the differential

$$\hat{J} = J^* - V,$$

where $V = \big(V(1), \ldots, V(n)\big)$ is *any* n-dimensional vector and $V(t) = 0$. In particular, by subtracting $V(i)$ from both sides of the Bellman equation (4.7), and adding and subtracting $V(j)$ within the right-hand side summation, we obtain

$$\hat{J}(i) = p_{it}(u)\hat{g}(i,u,t) + \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(\hat{g}(i,u,j) + \hat{J}(j)\big), \tag{4.9}$$

for all $i = 1, \ldots, n$, where

$$\hat{g}(i,u,j) = \begin{cases} g(i,u,j) + V(j) - V(i) & \text{if } i,j = 1, \ldots, n, \\ g(i,u,t) - V(i) & \text{if } i = 1, \ldots, n,\ j = t. \end{cases} \tag{4.10}$$

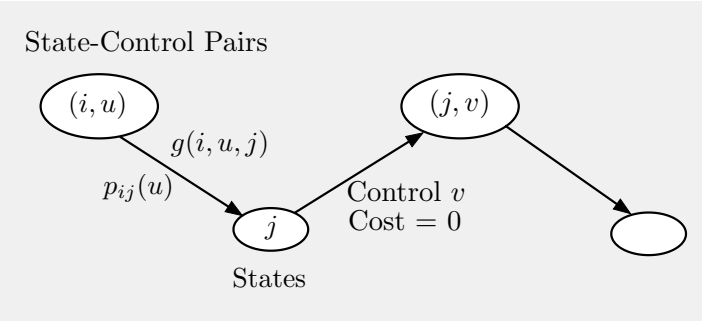

Figure 4.2.2 States, transition probabilities, and stage costs corresponding to a modified SSP problem, which yields the optimal Q-factors as well as the optimal costs. The states of this problem are the pairs (i,u), $u \in U(i)$, the original problem states $i = 1,\ldots,n$, and the termination state t. A control $v \in U(j)$ is available only at the original system states j, leading to the pair (j,v) at cost 0. The transition from a pair (i,u) leads to j with probability $p_{ij}(u)$ and cost 0. The Bellman equation for this modified problem is

$$Q^*(i,u) = p_{it}(u)g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \min_{v\in U(j)} Q^*(j,v)\right),$$

for the states (i,u), $u \in U(i)$, and

$$J^*(j) = \min_{v\in U(j)} Q^*(j,v),$$

for the states $j = 1,\ldots,n$. Note that a policy μ for this problem leads from a state j to the state $\big(j,\mu(j)\big)$, so in any system trajectory, only pairs of the form $\big(j,\mu(j)\big)$ are visited after the first transition.

We refer to Eq. (4.9) as the *variational form of Bellman's equation*, and to the modified cost per stage $\hat{g}$ as the *temporal difference* corresponding to V. Temporal differences play a significant role in several algorithmic RL contexts; see the approximate DP/RL books referenced earlier.

Note that Eq. (4.9) is the Bellman equation for a cost-modified problem, where the cost per stage g has been replaced by the temporal difference $\hat{g}$. Thus by applying Prop. 4.2.2 it follows that $\hat{J} = J^* - V$ is the unique solution of this equation. Thus J^* can be obtained by solving either the original or the cost-modified version of the problem. It follows that the original and the cost-modified SSP problems are essentially equivalent, and the choice of V does not matter when exact DP methods are used to solve them. However, when approximate methods are used, different results may be obtained, which can be more favorable with an appropriate choice of V.

In particular, we have the option to choose V and an approximation

architecture methodology that matches the differential $\hat{J} = J^* - V$ better than it matches J^*. For example, we may obtain V with some problem approximation scheme as a rough estimate of J^*, and then use a different approximation in value space scheme, based on different principles, for the corresponding cost-modified problem. We refer to this as *cost shaping* (the name "reward shaping" in used in the RL literature, where reward maximization is used). While cost shaping does not change the optimal policies of the original DP problem, it may change significantly the suboptimal policies produced by approximate DP methods, such as the ones that we will discuss in this chapter and the next.

4.3 DISCOUNTED PROBLEMS

We now consider the discounted problem, where there is a discount factor $\alpha < 1$. Using our transition probability notation notation, the Bellman equation and the VI algorithm take the following form.

Bellman Equation and Value Iteration for Discounted Problems:

For all $i = 1, \ldots, n$, we have

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J^*(j)\big).$$

For all $i = 1, \ldots, n$, and any initial conditions $J_0(1), \ldots, J_0(n)$, the VI algorithm generates the sequence $\{J_k\}$ according to

$$J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J_k(j)\big).$$

We will now show that the discounted problem can be converted to an SSP problem for which the analysis of the preceding section applies. To see this, let $i = 1, \ldots, n$ be the states, and consider an associated SSP problem involving the states $1, \ldots, n$ plus an artificial termination state t, with state transitions and costs obtained as follows: From a state $i \neq t$, when control u is applied, the next state is j with probability $\alpha p_{ij}(u)$ at a cost $g(i,u,j)$, and t with probability $1 - \alpha$ at zero cost; see Fig. 4.3.1. Note that Assumption 4.2.1 of the preceding section is satisfied for this SSP problem, since t is reached with probability $1 - \alpha > 0$ from every state in a single step.

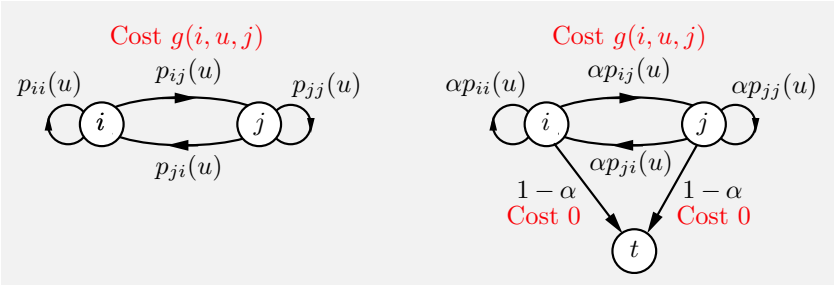

Figure 4.3.1 Transition probabilities for an α-discounted problem and its associated SSP problem. In the latter problem, the probability that the state is not t after k stages is α^k. The transition costs at the kth stage are $g(i,u,j)$ for both problems, but they must be multiplied by α^k because of discounting (in the discounted case) or because it is incurred with probability α^k when termination has not yet been reached (in the SSP case).

Suppose now that we use the same policy in the discounted problem and in the associated SSP problem. Then, as long as termination has not occurred, the state evolution in the two problems is governed by the same transition probabilities. Furthermore, the cost of the kth stage of the associated shortest path problem is $g\big(i_k, \mu^k(i_k), i_{k+1}\big)$ multiplied by the probability that state t has not yet been reached, which is α^k. This is also the expected cost of the kth stage for the discounted problem. Thus the cost of any policy starting from a given state, is the same for the original discounted problem and for the associated SSP problem.

It follows that we can apply Props. 4.2.1-4.2.5 of the preceding section to the associated SSP problem and obtain corresponding results for the discounted problem, which properly incorporate the discount factor in accordance with the SSP-to-discounted equivalence just established.

Proposition 4.3.1: (Convergence of VI) Given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\big\{J_k(i)\big\}$ generated by the VI algorithm

$$J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_k(j)\big), \qquad i = 1, \ldots, n,$$

converges to the optimal cost $J^*(i)$ for each i.

Proposition 4.3.2: (Bellman's Equation) The optimal cost function $J^* = \big(J^*(1), \ldots, J^*(n)\big)$ satisfies the equation

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J^*(j)\big), \qquad i = 1, \ldots, n, \quad (4.11)$$

and is the unique solution of this equation.

Proposition 4.3.3: (VI and Bellman's Equation for Policies) For any stationary policy μ, the corresponding cost function $J_\mu = \big(J_\mu(1), \ldots, J_\mu(n)\big)$ is the unique solution of the equation

$$J_\mu(i) = \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + \alpha J_\mu(j)\Big), \qquad i = 1, \ldots, n.$$

Furthermore, given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\big\{J_k(i)\big\}$ generated by the VI algorithm

$$J_{k+1}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + \alpha J_k(j)\Big), \qquad i = 1, \ldots, n,$$

converges to the cost $J_\mu(i)$ for each i.

Proposition 4.3.4: (Optimality Condition) A stationary policy μ is optimal if and only if for every state i, $\mu(i)$ attains the minimum in the Bellman equation (4.11).

Bellman's equation (4.11) has a familiar DP interpretation. At state i, the optimal cost $J^*(i)$ is the minimum over all controls of the sum of the expected current stage cost and the expected optimal cost of all future stages. The former cost is $g(i,u,j)$. The latter cost is $J^*(j)$, but since this cost starts accumulating after one stage, it is discounted by multiplication with α.

Similar to Prop. 4.2.5, there is a contraction mapping result and convergence rate estimate for value iteration. It is useful for the analysis of exact and approximate methods for discounted problems.

Proposition 4.3.5: (Contraction Property of the DP Operator) The DP operator T defined by

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J(j)\big), \qquad i = 1, \ldots, n, \tag{4.12}$$

for all vectors $J = \big(J(1), \ldots, J(n)\big)$, is a contraction mapping of modulus α with respect to the norm

$$\|J\| = \max_{i=1,\ldots,n} \big|J(i)\big|.$$

In particular, for any two n-dimensional vectors J and J', we have

$$\|TJ - TJ'\| \leq \alpha \|J - J'\|.$$

We finally mention the variational form of Bellman's equation, which for any given vector V takes the form

$$\hat{J}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(\hat{g}(i,u,j) + \alpha \hat{J}(j)\big), \qquad i = 1, \ldots, n,$$

where

$$\hat{g}(i,u,j) = g(i,u,j) + \alpha V(j) - V(i), \qquad i = 1, \ldots, n,$$

is the temporal difference corresponding to V; cf. Eqs. (4.9) and (4.10).

Example 4.3.1 (Asset Selling)

Consider of problem of selling an asset over an infinite number of periods. At each period an offer becomes available. We assume that offers at different periods are independent and that they can take n values $v_1, \ldots, v_n$ with corresponding probabilities according to given probability $p(1), \ldots, p(n)$. Here, if accepted, the amount i_k offered in period k, will be invested at a rate of interest r. By depreciating the sale amount to period 0 dollars, we view $(1+r)^{-k} i_k$ as the reward for selling the asset in period k at a price i_k, where $r > 0$ is the rate of interest. Then we have a discounted reward problem with discount factor $\alpha = 1/(1+r)$. The analysis of the present section is applicable, and the optimal value function J^* is the unique solution of Bellman's equation

$$J^*(i) = \max \left[i, \frac{1}{1+r} \sum_{j=1}^{n} p_j J^*(j) \right].$$

Thus the optimal reward function is characterized by the critical number

$$c = \frac{1}{1+r}\sum_{j=1}^{n} p_j J^*(j).$$

An optimal policy is obtained by minimizing over the two controls. It is to sell if and only if the current offer i is greater than c.

Bellman Equation and Value Iteration for Q-factors

As in the SSP case, the results just given have counterparts involving the optimal Q-factors, defined by

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J^*(j)\big), \qquad i = 1,\ldots,n,\ u \in U(i).$$

They can be obtained from the corresponding SSP results, by viewing the discounted problem as a special case of the SSP problem. Once Q^* or an approximation $\tilde{Q}$ is computed by some method (model-based or model-free), an optimal policy μ^* or approximately optimal policy $\tilde{\mu}$ can be obtained from the minimization

$$\mu^*(i) \in \arg\min_{u\in U(i)} Q^*(i,u), \qquad i = 1,\ldots,n,$$

or the approximate version

$$\tilde{\mu}(i) \in \arg\min_{u\in U(i)} \tilde{Q}(i,u), \qquad i = 1,\ldots,n.$$

Our basic results relating Bellman's equation and the VI algorithm are stated as follows.

Bellman Equation and Value Iteration for Q-Factors and Discounted Problems:

For all $i = 1,\ldots,n$, and $u \in U(i)$ we have

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{v\in U(j)} Q^*(j,v)\right), \tag{4.13}$$

For all $i = 1,\ldots,n$, and $u \in U(i)$, and any initial conditions $Q_0(i,u)$, the VI algorithm generates the sequence $\{Q_k\}$ according to

$$Q_{k+1}(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{v\in U(j)} Q_k(j,v)\right). \tag{4.14}$$

The VI algorithn (4.14) forms the basis for various Q-learning methods to be discussed later.

4.4 EXACT AND APPROXIMATE VALUE ITERATION

We have already encountered the VI algorithm

$$J_{k+1}(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J_k(j) \big) \right], \quad (4.15)$$

for SSP problems, and its discounted version

$$J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + \alpha J_k(j) \big). \quad (4.16)$$

It is one of the principal methods for calculating the optimal cost function J^*.

Unfortunately, when the number of states is large, the iterations (4.15) and (4.16) may be prohibitively time consuming. This motivates an approximate version of VI, which is patterned after the least squares regression/fitted VI scheme of Section 3.3. We start with some initial approximation to J^*, call it $\tilde{J}_0$. Then we generate a sequence $\{\tilde{J}_k\}$ where $\tilde{J}_{k+1}$ is equal to the exact value iterate $T\tilde{J}_k$ plus some error [we are using here the shorthand notation for the DP operator T given in Eqs. (4.8) and (4.12)]. Assuming that values $(T\tilde{J}_k)(i)$ may be generated for sample states i, we may obtain $\tilde{J}_{k+1}$ by some form of least squares regression.

Error Bounds for Approximate Value Iteration

We will focus on approximate VI for discounted problems. The analysis for SSP problems is qualitatively similar. We first consider estimates of the *cost function error*

$$\max_{i=1,\ldots,n} \big| \tilde{J}_k(i) - J^*(i) \big|, \quad (4.17)$$

and the *policy error*

$$\max_{i=1,\ldots,n} \big| J_{\tilde{\mu}^k}(i) - J^*(i) \big|, \quad (4.18)$$

where the policy $\tilde{\mu}^k$ is obtained from the minimization

$$\tilde{\mu}^k(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + \alpha \tilde{J}_k(j) \big).$$

It turns out that such estimates are possible, but under assumptions whose validity may be hard to guarantee. In particular, it is natural to assume that the error in generating the value iterates $(T\tilde{J}_k)(i)$ is within some $\delta > 0$ for every state i and iteration k, i.e., that

$$\max_{i=1,\ldots,n} \left| \tilde{J}_{k+1}(i) - \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_k(j)\big) \right| \leq \delta. \tag{4.19}$$

It is then possible to show that asymptotically, as $k \to \infty$, the cost error (4.17) becomes less or equal to $\delta/(1-\alpha)$, while the policy error (4.18) becomes less or equal to $2\delta/(1-\alpha)^2$.

These error bounds are given as Prop. 2.5.3 of [Ber12], but it is important to note that the condition (4.19) may not be satisfied by the natural least squares regression/fitted VI scheme of Section 3.3. This is illustrated by the following simple example from [BeT96], Section 6.5.3, which shows that the errors from successive approximate value iterations can accumulate to the point where the condition (4.19) cannot be maintained, and the approximate value iterates $\tilde{J}_k$ can grow unbounded.

Example 4.4.1 (Error Amplification in Approximate Value Iteration)

Consider a two-state discounted problem with states 1 and 2, and a single policy. The transitions are deterministic: from state 1 to state 2, and from state 2 to state 2. These transitions are also cost-free. Thus we have $J^*(1) = J^*(2) = 0$.

We consider a VI approach that approximates cost functions within the one-dimensional subspace of linear functions $S = \big\{(r, 2r) \mid r \in \Re\big\}$; this is a favorable choice since the optimal cost function $J^* = (0,0)$ belongs to S. We use a weighted least squares regression scheme. In particular, given $\tilde{J}_k = (r_k, 2r_k)$, we find $\tilde{J}_{k+1} = (r_{k+1}, 2r_{k+1})$, where for some weights $\xi_1, \xi_2 > 0$, the scalar r_{k+1} is obtained as

$$r_{k+1} \in \arg\min_r \Big[\xi_1\big(r - (T\tilde{J}_k)(1)\big)^2 + \xi_2\big(2r - (T\tilde{J}_k)(2)\big)^2\Big].$$

Since for a zero cost per stage and the given deterministic transitions, we have

$$T\tilde{J}_k = \big(\alpha J_k(2), \alpha J_k(2)\big) = (2\alpha r_k, 2\alpha r_k),$$

the preceding minimization is written as

$$r_{k+1} \in \arg\min_r \big[\xi_1(r - 2\alpha r_k)^2 + \xi_2(2r - 2\alpha r_k)^2\big],$$

which by writing the corresponding optimality condition yields

$$r_{k+1} = \alpha\zeta r_k \quad \text{where} \quad \zeta = \frac{2(\xi_1 + 2\xi_2)}{\xi_1 + 4\xi_2} > 1.$$

Thus if ξ_1 and ξ_2 are chosen so that $\alpha > 1/\zeta$, the sequence $\{r_k\}$ diverges and so does $\{\tilde{J}_k\}$. In particular, for the natural choice $\xi_1 = \xi_2 = 1$, we have $\zeta = 6/5$, so the approximate VI scheme diverges for α in the range $(5/6, 1)$.

The difficulty here is that the approximate VI mapping that generates $\tilde{J}_{k+1}$ by a weighted least squares-based approximation of $T\tilde{J}_k$ is not a contraction (even though T itself is a contraction). At the same time there is no δ such that the condition (4.19) is satisfied for all k, because of error amplification in each approximate VI.

The preceding example indicates that the choice of the least squares weights is important in determining the success of least squares-based approximate VI schemes. Generally, in regression-based parametric architecture training schemes of the type discussed is Section 3.1.2, the weights are related to the way samples are collected: the weight ξ_i for state i is the proportion of the number of samples in the least squares summation that correspond to state i. Thus $\xi_1 = \xi_2 = 1$ in the preceding example means that we use an equal number samples for each of the two states 1 and 2.

Now let the kth value iterate be

$$\tilde{J}_k(i) = \tilde{J}(i, r_k), \qquad i = 1, \ldots, n,$$

where r_k is the parameter vector of an approximation architecture $\tilde{J}(i, \cdot)$ that is used to represent the kth value iterate. Then the parameter r_{k+1} used to represent the next value iterate as

$$\tilde{J}_{k+1}(i) = \tilde{J}(i, r_{k+1}), \qquad i = 1, \ldots, n,$$

is obtained by the minimization

$$r_{k+1} \in \arg\min_r \sum_{s=1}^{q} \big(\tilde{J}(i^s, r) - \beta^s\big)^2, \tag{4.20}$$

where (i^s, β^s), $s = 1, \ldots, q$, is a training set with each β^s being the value iterate at the state i^s:

$$\beta^s = (T\tilde{J}_k)(i^s).$$

The critical question now is how to select the sample states i^s, $s = 1, \ldots, q$, to guarantee that the iterates r_k remain bounded, so that a condition of the form (4.19) is satisfied and the instability illustrated with Example 4.4.1 is avoided. It turns out that there is no known general method to guarantee this in infinite horizon problems. However, some practical methods have been developed. One such method is to weigh each state according to its "long-term importance," i.e., proportionally to the number of its occurrences over a long trajectory under a "good" heuristic policy. To implement this, we may run the system with the heuristic policy starting from a number of representative states, wait for some time

for the system to approach steady-state, and record the generated states i^s, $s = 1, \ldots, q$, to be used in the regression scheme (4.20). There is no theoretical guarantee for the stability of this scheme in the absence of additional conditions: it has been used with success in several reported case studies, although its rationale has only a tenuous basis in analysis; we refer to the end-of-chapter references.

4.5 POLICY ITERATION

The major alternative to value iteration is *policy iteration* (PI for short). This algorithm starts with a stationary policy μ^0, and generates iteratively a sequence of new policies $\mu^1, \mu^2, \ldots$. The algorithm has solid convergence guarantees when implemented in its exact form, as we will show shortly. When implemented in approximate form, as it is necessary when the number of states is large, its performance guarantees are more favorable than those of the approximate VI of the preceding section, and it does not exhibit any instability of the type highlighted by Example 4.4.1.

The closest analog of PI that we have encountered so far is the rollout algorithm of Chapter 2. There we have started with some policy and produced an improved policy though a process of cost function evaluation and one-step or multistep minimization. This idea is extended in the context of PI, which consists of multiple successive policy evaluations and policy improvements.

4.5.1 Exact Policy Iteration

Consider first the SSP problem. Here, each policy iteration consists of two phases: *policy evaluation* and *policy improvement*; see Fig. 4.5.1.

Exact Policy Iteration: SSP Problems

Given the typical policy μ^k:

Policy evaluation computes $J_{\mu^k}(i)$, $i = 1, \ldots, n$, as the solution of the (linear) system of Bellman equations

$$J_{\mu^k}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu^k(i)\big)\Big(g\big(i, \mu^k(i), j\big) + J_{\mu^k}(j)\Big), \quad i = 1, \ldots, n,$$

(cf. Prop. 4.2.3).

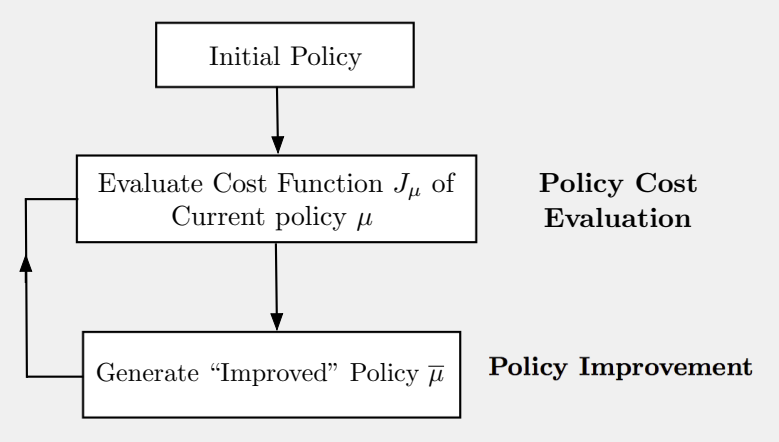

Figure 4.5.1 Illustration of exact PI. Each iteration consists of a policy evaluation using the current policy μ, followed by generation of an improved policy $\overline{\mu}$.

Policy improvement then computes a new policy μ^{k+1} as

$$\mu^{k+1}(i) \in \arg\min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+J_{\mu^k}(j)\big), \quad i=1,\ldots,n.$$

The process is repeated with μ^{k+1} used in place of μ^k, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all i, in which case the algorithm terminates with the policy μ^k.

The counterpart for discounted problems is as follows.

Exact Policy Iteration: Discounted Problems

Given the typical policy μ^k:

Policy evaluation computes $J_{\mu^k}(i)$, $i=1,\ldots,n$, as the solution of the (linear) system of Bellman equations

$$J_{\mu^k}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu^k(i)\big)\Big(g\big(i,\mu^k(i),j\big)+\alpha J_{\mu^k}(j)\Big), \quad i=1,\ldots,n, \tag{4.21}$$

(cf. Prop. 4.2.3).

Policy improvement then computes a new policy μ^{k+1} as

$$\mu^{k+1}(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+\alpha J_{\mu^k}(j)\big), \quad i = 1, \ldots, n. \tag{4.22}$$

The process is repeated with μ^{k+1} used in place of μ^k, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all i, in which case the algorithm terminates with the policy μ^k.

The following proposition, proved in the appendix, establishes the validity of PI, including finite termination with an optimal policy.

Proposition 4.5.1: (Convergence of Exact PI) For both the SSP and the discounted problems, the exact PI algorithm generates an improving sequence of policies, i.e.,

$$J_{\mu^{k+1}}(i) \leq J_{\mu^k}(i), \qquad \text{for all } i \text{ and } k,$$

and terminates with an optimal policy.

Example 4.5.1 (Treasure Hunting)

A treasure hunter has obtained a lease to search a site that contains n treasures, and wants to find a searching policy that maximizes his expected gain over an infinite number of days. At each day, knowing the current number of treasures not yet found, he may decide to continue searching for more treasures at a cost c per day, or to permanently stop searching. If he searches on a day when there are i treasures on the site, he finds $m \in [0, i]$ treasures with given probability $p(m \mid i)$, where we assume that $p(0 \mid i) < 1$ for all $i \geq 1$, and that the expected number of treasures found,

$$r(i) = \sum_{m=0}^{i} mp(m \mid i),$$

is monotonically nondecreasing with i. Each found treasure is worth 1 unit.

We formulate the problem as an SSP problem, with state equal to the number of treasures not yet found. The termination state is state 0, where the hunter stops searching. When the hunter decides to search at a state $i \geq 1$, the state moves to $i - m$ with probability $p(m \mid i)$. Here the inevitable termination Assumption 4.2.1 is satisfied, in view of our condition $p(0 \mid i) < 1$ for all i. Bellman's equation is

$$J^*(i) = \max\left[0, r(i) - c + \sum_{m=0}^{i} p(m \mid i)J^*(i-m)\right], \qquad i = 1, \ldots, n,$$

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

Let us apply PI starting with the policy μ^0 that never searches. This policy has value function

$$J_{\mu^0}(i) = 0, \qquad \text{for all } i.$$

The policy μ^1 subsequently produced by PI is the one that searches at a state i if and only if $r(i) > c$, and has value function satisfying the Bellman equation

$$J_{\mu^1}(i) = \begin{cases} 0 & \text{if } r(i) \leq c, \\ r(i) - c + \sum_{m=0}^{i} p(m \mid i) J_{\mu^1}(i-m) & \text{if } r(i) > c. \end{cases} \tag{4.23}$$

Note that the values $J_{\mu^1}(i)$ are nonnegative for all i, since by Prop. 4.5.1, we have

$$J_{\mu^1}(i) \geq J_{\mu^0}(i) = 0.$$

The next policy generated by PI is obtained from the minimization

$$\mu^2(i) = \arg\max \left[0, r(i) - c + \sum_{m=0}^{i} p(m \mid i) J_{\mu^1}(i-m) \right], \qquad i = 1, \ldots, n.$$

For i such that $r(i) \leq c$, we have $r(j) \leq c$ for all $j < i$ because $r(i)$ is monotonically nondecreasing in i. Moreover, using Eq. (4.23), we have $J_{\mu^1}(i-m) = 0$ for all $m \geq 0$. It follows that for i such that $r(i) \leq c$,

$$0 \geq r(i) - c + \sum_{m=0}^{i} p(m \mid i) J_{\mu^1}(i-m),$$

and $\mu^2(i) = \textit{stop searching}$.

For i such that $r(i) > c$, we have $\mu^2(i) = \textit{search}$, since $J_{\mu^1}(i) \geq 0$ for all i, so that

$$0 < r(i) - c + \sum_{m=0}^{i} p(m \mid i) J_{\mu^1}(i-m).$$

Thus, μ^2 is the same as μ^1 and the PI algorithm terminates. By Prop. 4.5.1, it follows that μ^2 is optimal.

Optimistic and Multistep Lookahead Policy Iteration

The PI algorithm that we have discussed so far uses exact policy evaluation of the current policy μ^k and one-step lookahead policy improvement, i.e., it computes exactly J_{μ^k}, and it obtains the next policy μ^{k+1} by a one-step lookahead minimization using J_{μ^k} as an approximation to J^*. It is possible to use a more flexible algorithm whereby J_{μ^k} is approximated by

any number of value iterations corresponding to μ^k (cf. Prop. 4.3.3) and the policy improvement is done using multistep lookahead.

A PI algorithm that uses a finite number of steps for VI for policy evaluation is referred to as *optimistic*. It can be viewed as a combination of VI and PI. Its convergence properties are solid (see e.g., [BeT96], Section 5.4, and [Ber12], Chapter 2) although it may require an infinite number of iterations to converge to J^*. To see why this is so, suppose that we evaluate each policy by a single value iteration. Then the method is essentially identical to the VI method, which requires an infinite number of iterations to converge. Generally, many approximate policy evaluation schemes use forms of optimistic PI, particularly in connection with Q-learning, which will be discussed later in Section 4.8.

The motivation for multistep policy improvement is that it may yield a better policy μ^{k+1} than with one-step lookeahead. In fact this makes even more sense when the evaluation of μ^k is approximate, since then the longer lookahead may compensate for errors in the policy evaluation. The method in its exact nonoptimistic form is given below (in a different version it may be combined with optimistic PI).

Multistep Lookahead Exact Policy Iteration: Discounted Problems

Given the typical policy μ^k:

Policy evaluation computes $J_{\mu^k}(i)$, $i = 1, \ldots, n$, as the solution of the (linear) system of Bellman equations

$$J_{\mu^k}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu^k(i)\big)\Big(g\big(i, \mu^k(i), j\big) + \alpha J_{\mu^k}(j)\Big), \quad i = 1, \ldots, n,$$

(cf. Prop. 4.2.3).

Policy improvement with ℓ-step lookahead then solves the ℓ-stage problem with terminal cost function J_{μ^k}. If $\{\hat{\mu}_0, \ldots, \hat{\mu}_{\ell-1}\}$ is the optimal policy of this ℓ-stage problem, then the new policy μ^{k+1} is $\hat{\mu}_0$.

The process is repeated with μ^{k+1} used in place of μ^k, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all i, in which case the algorithm terminates with the policy μ^k.

Exact PI with multistep lookahead has the same solid convergence properties as its one-step lookahead counterpart: it terminates with an optimal policy, and the generated sequence of policies is monotonically improving. The proof is somewhat complicated and will not be given. It

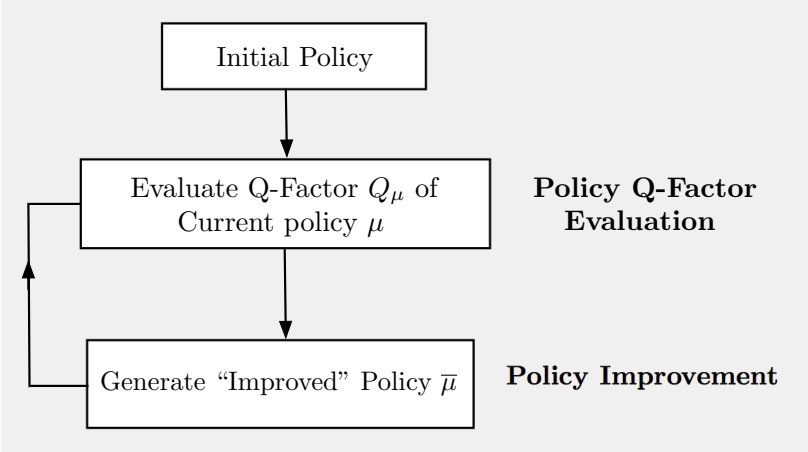

Figure 4.5.2 Block diagram of exact PI for Q-factors. Each iteration consists of a policy evaluation using the current policy μ, followed by generation of an improved policy $\overline{\mu}$.

combines arguments from the proof of Prop. 4.5.1 and proofs of convergence of optimistic PI (see e.g., [Ber12], Chapter 2).

4.5.2 Policy Iteration for Q-factors

Similar to VI, we may also equivalently implement PI through the use of Q-factors. To see this, first note that the policy improvement step may be implemented by minimizing over $u \in U(i)$ the expression

$$Q_\mu(i,u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_\mu(j)\big), \qquad i = 1,\ldots,n,\ u \in U(i),$$

which we view as the *Q-factor of the pair* (i,u) *corresponding to* μ. Note that we have

$$J_\mu(j) = Q_\mu\big(j,\mu(j)\big),$$

(cf. Prop. 4.2.3).

The following algorithm is thus obtained; see Fig. 4.5.2.

Exact Policy Iteration for Q-Factors: Discounted Problems

Given the typical policy μ^k:

Policy evaluation computes $Q_{\mu^k}(i,u)$, for all $i = 1,\ldots,n$, and $u \in U(i)$, as the solution of the (linear) system of equations

$$Q_{\mu^k}(i,u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha Q_{\mu^k}\big(j,\mu^k(j)\big)\Big). \qquad (4.24)$$

Policy improvement then computes a new policy μ^{k+1} as

$$\mu^{k+1}(i) \in \arg\min_{u\in U(i)} Q_{\mu^k}(i,u), \qquad i = 1,\ldots,n. \qquad (4.25)$$

The process is repeated with μ^{k+1} used in place of μ^k, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all i, in which case the algorithm terminates with the policy μ^k.

Note that the system (4.24) has a unique solution, since from the uniqueness of solution of Bellman's equation, any solution must satisfy

$$Q_{\mu^k}\big(j,\mu^k(j)\big) = J_{\mu^k}(j).$$

Hence the Q-factors $Q_{\mu^k}\big(j,\mu^k(j)\big)$ are uniquely determined, and then the remaining Q-factors $Q_{\mu^k}(i,u)$ are also uniquely determined from Eq. (4.24).

The PI algorithm for Q-factors is mathematically equivalent to PI for costs, as given in the preceding subsection. The only difference is that we calculate *all* the Q-factors $Q_{\mu^k}(i,u)$, rather than just the costs $J_{\mu^k}(j) = Q_{\mu^k}\big(j,\mu^k(j)\big)$, i.e., just the Q-factors corresponding to the controls chosen by the current policy. However, the remaining Q-factors $Q_{\mu^k}(i,u)$ are needed for the policy improvement step (4.25), so no extra computation is required. It can be verified also that the PI algorithm (4.24)-(4.25) can be viewed as the PI algorithm for the discounted version of the modified problem of Fig. 4.2.2.

4.5.3 Limited Lookahead Policies and Rollout

We will now start a more focused discussion of infinite horizon approximations, beginning with discounted problems. Consistently with the finite horizon approximation in value space schemes of Chapter 2, the general idea is to compute some approximation $\tilde{J}$ of the optimal cost function J^*, and then use one-step or multistep lookahead to implement an optimal policy $\tilde{\mu}$. Thus, a *one-step lookahead policy* applies at state i the control $\tilde{\mu}(i)$

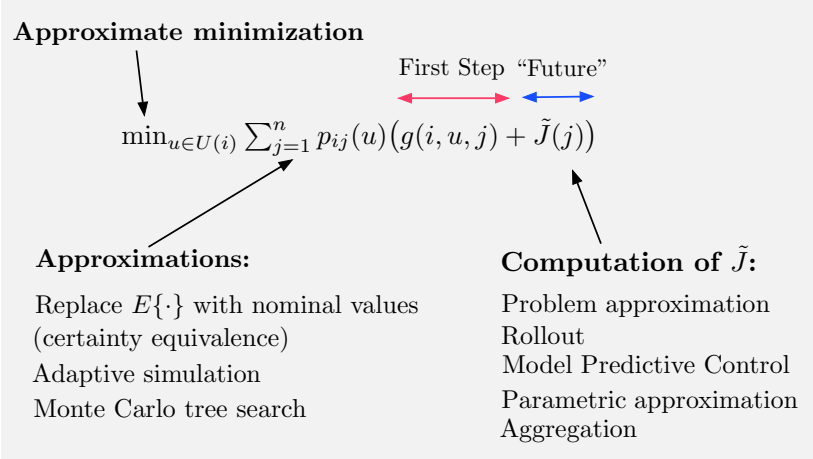

Figure 4.5.3 Schematic illustration of various options for approximation in value space with one-step lookahead in infinite horizon problems. The lookahead function values $\tilde{J}(j)$ approximate the optimal cost-to-go values $J^*(j)$, and can be computed by a variety of methods. There may be additional approximations in the minimization over u_k and the computation of the expected value.

that attains the minimum in the expression

$$\min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+\alpha\tilde{J}(j)\big),$$

see Fig. 4.5.3.

Similarly, at state i, a *two-step lookahead policy* applies the control $\tilde{\mu}(i)$ attaining the minimum in the preceding equation, where now $\tilde{J}$ is obtained itself on the basis of a one-step lookahead approximation. In other words, for all states j that can be reached from i, we have

$$\tilde{J}(j) = \min_{u\in U(j)} \sum_{\ell=1}^{n} p_{j\ell}(u)\big(g(j,u,\ell)+\alpha\hat{J}(\ell)\big),$$

where $\hat{J}$ is some approximation of J^*. Thus $\tilde{J}$ is the result of a single value iteration starting from $\hat{J}$. The policy $\tilde{\mu}$ is the one used at the first stage of a two-stage optimal control problem with terminal cost function $\hat{J}$. Policies with lookahead of more than two stages are similarly defined.

In Chapter 2 we gave several types of limited lookahead schemes, where $\tilde{J}$ is obtained in different ways, such as enforced decomposition, probabilistic approximation, and others. These schemes can be adapted to infinite horizon problems.

Let us briefly discuss rollout, where $\tilde{J}$ is the cost-to-go of some stationary policy μ (also called the *base policy* or *base heuristic*), i.e., $\tilde{J} = J_\mu$.

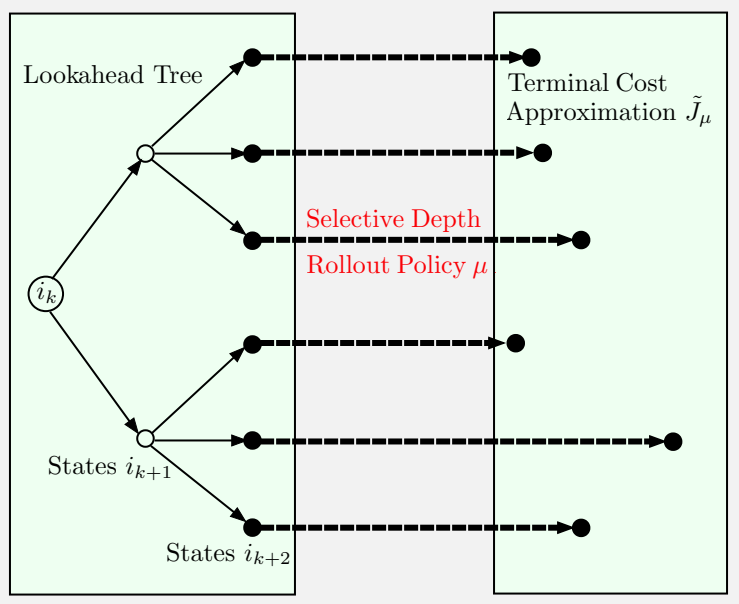

Figure 4.5.4 Illustration of two-step lookahead, rollout with a policy μ for a limited and state-dependent number of steps, and a terminal cost function approximation $\tilde{J}_\mu$. A Monte Carlo tree search scheme may also be used; cf. Section 2.4.2.

Thus, by definition, *the rollout policy is the result of a single policy improvement starting from* μ. The policy evaluation that yields the costs $J_\mu(j)$ needed for policy improvement may be done in any suitable way. Monte-Carlo simulation (averaging the costs of many trajectories starting from j) is one major possibility. Of course if the problem is deterministic, a single simulation trajectory starting from j is sufficient, in which case the implementation of the rollout policy is much less computationally demanding.

An important fact is that *the rollout policy improves over the base policy*. This is to be expected since rollout is one-step PI, so Prop. 4.5.1 applies, and it can also be verified with the type of argument we gave in Section 2.4. We also note the rollout algorithm that uses multiple base heuristics and simultaneously improves on all of them (cf. Section 2.4).

Combined Multistep Lookahead, Rollout, and Terminal Cost Approximation

In a variant of the rollout approach, we may use ℓ-step lookahead, and then use rollout with a policy μ at the end of the ℓ steps. The number of steps of the multistep lookahead may also be variable, based on problem-specific characteristics. Moreover, the rollout with policy μ may be applied for a

limited (and variable) number of steps, and the cost of the remaining steps may be approximated using some terminal cost approximation $\tilde{J}_\mu$; see Fig. 4.5.4.

The terminal cost approximation $\tilde{J}_\mu$ may be heuristic in nature, based on problem approximation, or based on a more systematic simulation methodology. For example, the values $J_\mu(i)$ may be computed by simulation for all i in a subset of representative states, and $\tilde{J}_\mu$ may be selected from a parametric class of vectors by a least squares regression of the computed values. This approximation may be performed off-line, outside the time-sensitive restrictions of a real-time implementation, and the result $\tilde{J}_\mu$ may be used on-line in place of J_μ as a terminal cost function approximation. Note, however, that once cost function approximation is introduced at the end of the rollout, the cost improvement property of the rollout policy over the base policy may be lost.

The scheme of Fig. 4.5.4 has been adopted in the rollout backgammon algorithm of Tesauro and Galperin [TeG96], with μ and the terminal cost function approximation $\tilde{J}_\mu$ provided by the TD-Gammon algorithm of Tesauro [Tes94], which was based on a neural network, trained using a form of optimistic policy iteration and TD(λ). A similar type of algorithm was used in the AlphaGo program (Silver et al. [SHM16]), with the policy and the terminal cost function obtained with a deep neural network, trained using a form of approximate policy iteration. Also the multistep lookahead in the AlphaGo algorithm was implemented during the training phase using Monte Carlo tree search (cf. Section 2.4.2). This was also true for the more recent AlphaZero programs for playing chess, Go, and other games (Silver at al. [SHS17]).

4.5.4 Approximate Policy Iteration - Error Bounds

When the number of states is very large, the policy evaluation step and/or the policy improvement step of the method may be implementable only through approximations. We are thus led to consider an approximate PI method that generates a sequence of stationary policies $\{\mu^k\}$ and a corresponding sequence of approximate cost functions $\{\tilde{J}_{\mu^k}\}$ according to the approximate policy evaluation error

$$\max_{i=1,\ldots,n} \left|\tilde{J}_{\mu^k}(i) - J_{\mu^k}(i)\right| \leq \delta, \tag{4.26}$$

and the approximate policy improvement error

$$\max_{i=1,\ldots,n} \left| \sum_{j=1}^{n} p_{ij}\big(\mu_{k+1}(i)\big)\Big(g\big(i,\mu_{k+1}(i),j\big) + \alpha \tilde{J}_{\mu^k}(j)\Big) - \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\Big(g(i,u,j) + \alpha \tilde{J}_{\mu^k}(j)\Big)\right| \leq \epsilon, \tag{4.27}$$

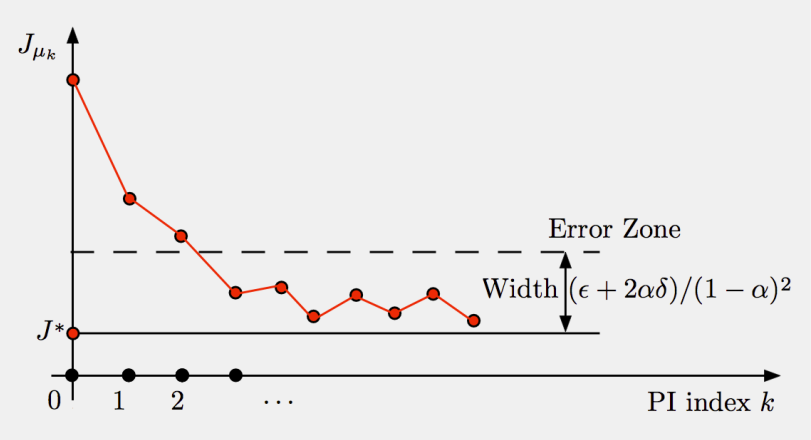

Figure 4.5.5 Illustration of typical behavior of approximate PI. In the early iterations, the method tends to make rapid and fairly monotonic progress, until J_{μ^k} gets within an error zone of size less than $(\delta + 2\epsilon)/(1-\alpha)^2$. After that J_{μ^k} oscillates randomly within that zone.

where δ and ϵ are some nonnegative scalars. The following proposition, proved in the appendix (and also in the original source [BeT96], Section 6.2.2), provides an error bound for discounted problems (a similar result is available for SSP problems; see [BeT96], Section 6.2.2).

Proposition 4.5.2: (Error Bound for Approximate PI) Consider the discounted problem, and let $\{\mu^k\}$ be the sequence generated by the approximate PI algorithm defined by the approximate policy evaluation (4.26) and the approximate policy improvement (4.27). Then the policy error

$$\max_{i=1,\ldots,n} \left|J_{\mu^k}(i) - J^*(i)\right|,$$

becomes less or equal to

$$\frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2},$$

asymptotically as $k \to \infty$.

The practical behavior of approximate PI is in qualitative agreement with the insights provided by the error bound of Prop. 4.5.2. In the beginning, the method tends to make rapid and fairly monotonic progress, but eventually it gets into an oscillatory pattern. This happens after J_{μ^k} gets within an error zone of size $(\delta + 2\epsilon)/(1-\alpha)^2$ or smaller, and then J_{μ^k}

oscillates fairly randomly within that zone; see Fig. 4.5.5. In practice, the error bound of Prop. 4.5.2 tends to be pessimistic, so the zone of oscillation is usually much narrower than what is suggested by the bound. However, the bound itself can be proved to be tight, in worst case. This is shown with an example in the book [BeT96], Section 6.2.3.

We finally note that since the set of policies is finite, the sequence $\{J_{\mu^k}\}$ is guaranteed to be bounded, so approximate PI is not hampered by the instability that was highlighted by Example 4.4.1 for approximate VI.

The Case Where Policies Converge

Generally, the policy sequence $\{\mu^k\}$ generated by approximate PI may oscillate between several policies, as noted earlier. However, under some circumstances the sequence converges to some policy $\overline{\mu}$, in the sense that

$$\mu^{\overline{k}+1} = \mu^{\overline{k}} = \overline{\mu} \qquad \text{for some } \overline{k}. \tag{4.28}$$

An important case where this happens is aggregation methods, which will be discussed in Chapter 5. In this case the behavior of the method is more regular, and we can show a more favorable bound than the one of Prop. 4.5.2, by a factor $1/(1-\alpha)$.

Proposition 4.5.3: (Error Bound for Approximate PI when Policies Converge) Let $\overline{\mu}$ be a policy generated by the approximate PI algorithm under conditions (4.26), (4.27), and (4.28). Then we have

$$\max_{i=1,\ldots,n} \left| J_{\overline{\mu}}(i) - J^*(i) \right| \leq \frac{\epsilon + 2\alpha\delta}{1-\alpha}.$$

We finally note that similar error bounds can be obtained for optimistic PI methods, where the policy evaluation is performed with just a few approximate value iterations, and policy improvement is approximate (cf. Section 4.5.1). These bounds are similar to the ones of the nonoptimistic PI case given in this section, but their derivation is quite complicated; see the end-of-chapter references.

4.6 SIMULATION-BASED POLICY ITERATION WITH PARAMETRIC APPROXIMATION

In this section we will discuss PI methods where the policy evaluation step is carried out with the use of a parametric approximation method and Monte-Carlo simulation. We will focus on the discounted problem, but similar methods can be used for SSP problems.

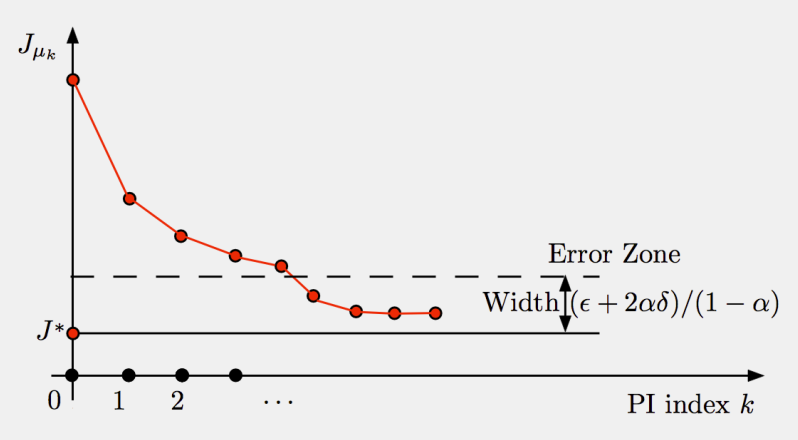

Figure 4.5.6 Illustration of typical behavior of approximate PI when policies converge. The method tends to make monotonic progress, and J_{μ^k} converges within an error zone of size less than $(\delta + 2\epsilon)/(1-\alpha)$.

4.6.1 Self-Learning and Actor-Critic Systems

The name *self-learning* in RL usually refers to some form of PI method that involves the use of simulation for approximate policy evaluation, and/or approximate Q-factor evaluation. A parametric architecture is used for this, and the algorithm that performs the policy evaluation is usually called a *critic*. If a neural network is used as the parametric architecture, it is called a *critic network*. The PI algorithm generates a sequence of stationary policies $\{\mu^k\}$ and a corresponding sequence of approximate cost function evaluations $\{\tilde{J}_{\mu^k}\}$ using a simulator of the system.

As in all PI methods, the policy evaluation $\tilde{J}_{\mu^k}$ is used for policy improvement, to generate the next policy μ^{k+1}. The algorithm that performs the policy improvement is usually called an *actor*, and if a neural network is involved, it is called an *actor network*.

The two operations needed at each policy iteration are as follows:

(a) *Evaluate the current policy μ^k (critic)*: Here algorithm, system, and simulator are merged in one, and the system "observes itself" by generating simulation cost samples under the policy μ^k. It then combines these samples to "learn" a policy evaluation $\tilde{J}_{\mu^k}$. Usually this is done through some kind of incremental method that involves a least squares minimization using cost samples, and either a linear architecture or a neural network.

(b) *Improve the current policy μ^k (actor)*: Given the approximate policy evaluation $\tilde{J}_{\mu^k}$, the system can generate or "learn" the new policy

μ^{k+1} through the minimization

$$\mu^{k+1}(i) \in \arg\min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_{\mu^k}(j)\big), \qquad i = 1,\ldots,n.$$

Alternatively the system can compute the minimizing control u^s at a set of sample states i^s, $s = 1,\ldots,q$, through

$$u^s \in \arg\min_{u\in U(i^s)} \sum_{j=1}^{n} p_{i^s j}(u)\big(g(i^s,u,j) + \alpha \tilde{J}_{\mu^k}(j)\big).$$

These are the sample values of the improved policy μ^{k+1} at the sample states i^s. They are generalized to "learn" a complete policy μ^{k+1} by using some approximation in policy space scheme (cf. Section 2.1.3).

We can thus describe simulation-based PI as a process where the system learns better and better policies by observing its behavior. This is true up to the point where either policy oscillations occur (cf. Fig. 4.5.5) or the algorithm terminates (cf. Fig. 4.5.6), at which time learning essentially stops.

It is worth noting that the system learns by itself, but it does not learn itself, in the sense that it does not construct a mathematical model for itself. It only learns to behave better, i.e., construct improved policies, through experience gained by simulating state and control trajectories generated with these policies. We may adopt instead an alternative two-phase approach: first use system identification and simulation to construct a mathematical model of the system, and then use a model-based PI method. We will not discuss this approach in this book.

4.6.2 A Model-Based Variant

We will first provide an example model-based method that is conceptually simple, and then discuss its model-free version. In particular, we assume that the transition probabilities $p_{ij}(u)$ are available, and that the cost function J_μ of any given policy μ is approximated using a parametric architecture $\tilde{J}_\mu(i,r)$.

We recall that given any policy μ, the exact PI algorithm for costs [cf. Eqs. (4.21)-(4.22)] generates the new policy $\overline{\mu}$ with a policy evaluation/policy improvement process. We approximate this process as follows; see Fig. 4.6.1.

(a) *Approximate policy evaluation*: To evaluate μ, we determine the value of the parameter vector r by generating a large number of training pairs (i^s, β^s), $s = 1,\ldots,q$, and by using least squares training:

$$\overline{r} \in \arg\min_{r} \sum_{s=1}^{q} \big(\tilde{J}_\mu(i^s,r) - \beta^s\big)^2. \tag{4.29}$$

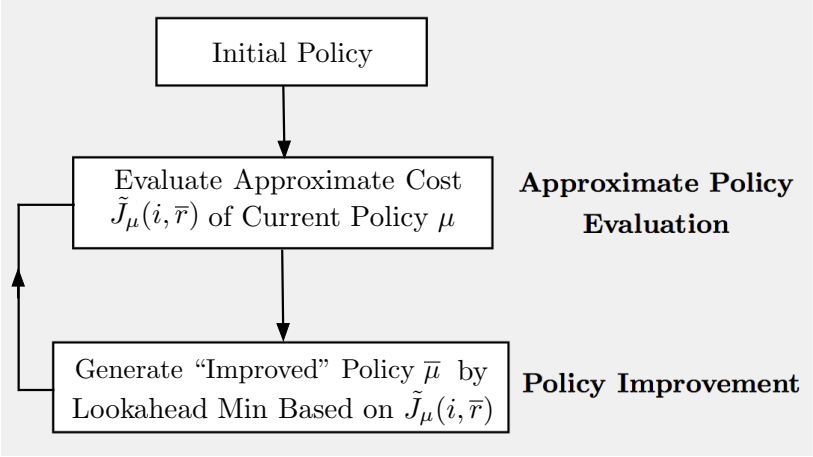

Figure 4.6.1 Block diagram of model-based approximate PI for cost functions.

For a given state i^s, the scalar β^s is a sample cost corresponding to i^s.

In particular β^s is generated by starting at i^s, simulating a trajectory of states and controls using μ and the known transition probabilities for some number N of stages, accumulating the corresponding discounted costs, and adding a terminal cost approximation

$$\alpha^N \hat{J}(i_N),$$

where i_N is the terminal state of the N-stage trajectory and $\hat{J}$ is some initial guess of J_μ. The guess $\hat{J}$ may be obtained with additional training or some other means, such as using the result of the policy evaluation of the preceding policy μ^{k-1}. It is also possible to use $\hat{J}(i_N) = 0$.

The approximate policy evaluation problem of Eq. (4.29) can be solved by the incremental methods discussed in Section 3.1.3. In particular the incremental gradient method is given by

$$r^{k+1} = r^k - \gamma^k \nabla \tilde{J}(i^{s_k}, r^k)\big(\tilde{J}(i^{s_k}, r^k) - \beta^{s_k}\big),$$

where (i^{s_k}, β^{s_k}) is the state-cost sample pair that is used at the kth iteration, and r^0 is an initial parameter guess. Here the approximation architecture $\tilde{J}(i,r)$ may be linear or may be nonlinear and differentiable. In the case of a linear architecture it is also possible to solve the problem (4.29) using the exact linear least squares formula.

(b) *Approximate policy improvement*: Having solved the approximate policy evaluation problem (4.29), the new "improved" policy $\overline{\mu}$ is

obtained by the approximate policy improvement operation

$$\overline{\mu}(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}(j,\overline{r})\big), \qquad i = 1, \ldots, n, \tag{4.30}$$

where $\overline{r}$ is the parameter vector obtained from the policy evaluation operation (4.29).

Trajectory Reuse and Bias-Variance Tradeoff

As we have noted, to each training pair (i^s, β^s) there corresponds an N-stage trajectory over which the sample cost β^s is accumulated, but the length of the trajectory may depend on s. This allows sampling effort economies based on *trajectory reuse*. In particular, suppose that starting at some state i_0 we generate a long trajectory $(i_0, i_1, \ldots, i_N)$ using the policy μ. Then we can obtain the state-cost sample that corresponds to i_0, as discussed above, but we can also obtain additional cost samples for the subsequent states i_1, i_2, etc, by using the tail portions of the trajectory $(i_0, i_1, \ldots, i_N)$ that start at these states.

Clearly, it is necessary to truncate the sample trajectories to some number of stages N, since we cannot simulate an infinite length trajectory in practice. If N is large, then because of the discount factor, the error for neglecting the stage costs beyond stage N will be small. However, there are other important concerns when choosing the trajectory lengths N.

In particular, a short length reduces the sampling effort, but is also a source of inaccuracy. The reason is that the cost of the tail portion of the trajectory (from stage N to infinity) is approximated by $\alpha^N \hat{J}(i_N)$, where i_N is the terminal state of the N-stage trajectory and $\hat{J}$ is the initial guess of J_μ. This terminal cost compensates for the costs of the neglected stages, but adds an error to the cost samples β^s, which becomes larger as the trajectory length N becomes smaller.

We note two additional benefits of using many training trajectories, each with a relatively short trajectory length:

(1) The cost samples β^s are less noisy, as they correspond to summation of fewer random stage costs. This leads to the so-called *bias-variance tradeoff*: short trajectories lead to larger bias but smaller variance of the cost samples.

(2) With more starting states i_0, there is better opportunity for *exploration of the state space*. By this we mean adequate representation of all possible initial trajectory states in the sample set. This a major issue in approximate PI, as we will discuss in Section 4.6.4.

Let us also note that the bias-variance tradeoff underlies the motivation for a number of alternative policy evaluation methods such as TD(λ),

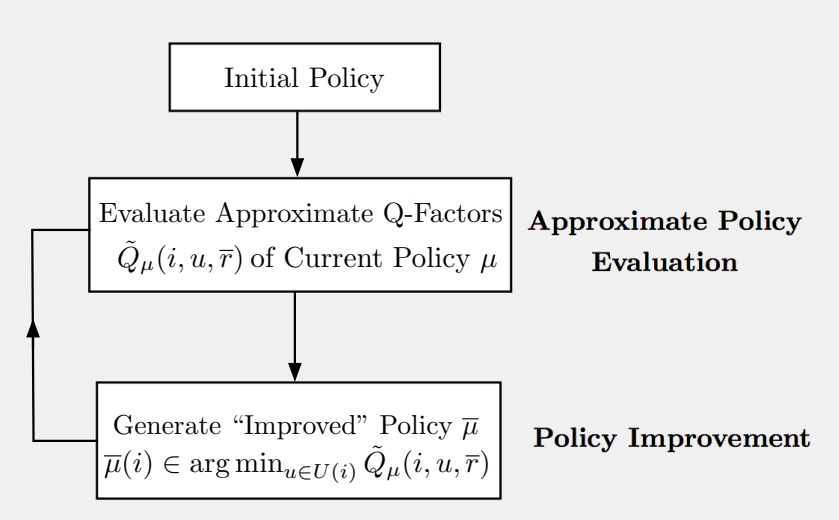

Figure 4.6.2 Block diagram of model-free approximate PI for Q-factors.

LSTD(λ), and LSPE(λ), which we will summarize in Section 4.9; see Section 6.3 of the book [Ber12] and other approximate DP/RL books referenced earlier. The papers [Ber11b], [YuB12], and the book [Ber12], Section 6.4, discuss a broad range of short trajectory sampling methods.

4.6.3 A Model-Free Variant

We next provide an example model-free method. Let us restate the PI method in terms of Q-factors, and in a form that involves approximations and simulation-based implementations. We recall that given any policy μ, the exact PI algorithm for Q-factors [cf. Eqs. (4.24)-(4.25)] generates the new policy $\overline{\mu}$ with a policy evaluation-policy improvement process. We approximate this process as follows; see Fig. 4.6.2.

(a) *Approximate policy evaluation*: Here we introduce a parametric architecture $\tilde{Q}_\mu(i,u,r)$ for the Q-factors of μ. We determine the value of the parameter vector r by generating (using a simulator of the system) a large number of training triplets (i^s,u^s,β^s), $s=1,\ldots,q$, and by using a least squares fit:

$$\overline{r} \in \arg\min_r \sum_{s=1}^{q} \big(\tilde{Q}_\mu(i^s,u^s,r)-\beta^s\big)^2. \tag{4.31}$$

In particular, for a given pair (i^s,u^s), the scalar β^s is a sample Q-factor corresponding to (i,u). It is generated by starting at i^s, using u^s at the first stage, and simulating a trajectory of states and controls

using μ for a total of N of stages, and accumulating the corresponding discounted costs. Thus, β^s is a sample of $Q_\mu^N(i^s, u^s)$, the N-stage Q-factor of μ, given by

$$Q_\mu^N(i,u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_\mu^{N-1}(j)\big).$$

The number of stages N in the sample trajectories may be different for different samples, and can be either large, or fairly small, and a terminal cost $\alpha^N \hat{J}(i_N)$ may be added as in the model-based case of Section 4.6.2. Again an incremental method may be used to solve the training problem (4.31).

(b) *Approximate policy improvement*: Here we compute the new policy $\overline{\mu}$ according to

$$\overline{\mu}(i) \in \arg\min_{u \in U(i)} \tilde{Q}_\mu(i,u,\overline{r}), \qquad i = 1, \ldots, n, \tag{4.32}$$

where $\overline{r}$ is the parameter vector obtained from the policy evaluation operation (4.31).

Unfortunately, trajectory reuse is more problematic in Q-factor evaluation than in cost evaluation, because each trajectory generates state-control pairs of the special form $\big(i, \mu(i)\big)$ at every stage after the first, so *pairs* (i,u) *with* $u \neq \mu(i)$ *are not adequately explored*; cf. the discussion in in Section 4.6.2. For this reason, it is necessary to make an effort to include in the samples a rich enough set of trajectories that start at such pairs (i,u).

An important alternative to the preceding procedure is a two-stage process for policy evaluation: first compute in model-free fashion a cost function approximation $\tilde{J}(j,\overline{r})$, using the regression (4.29), and then use a *second sampling process and regression* to approximate further the (already approximate) Q-factor

$$\sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}(j,\overline{r})\big),$$

with some $\tilde{Q}_\mu(i,u,\overline{r})$ possibly obtained with a policy approximation architecture (see the discussion of Section 2.1.3 on model-free approximation in policy space). Finally, once $\tilde{Q}_\mu(i,u,\overline{r})$ is obtained with this approximation in policy space, the "improved" policy $\overline{\mu}$ is obtained from the minimization (4.32). The overall scheme can be viewed as model-free approximate PI that is based on approximation in both value and policy space. In view of the two-fold approximation needed to obtain $\tilde{Q}_\mu(i,u,\overline{r})$, this scheme is more complex, but allows trajectory reuse and thus deals better with the exploration issue.

4.6.4 Issues Relating to Approximate Policy Iteration

Approximate PI in its various forms has been the subject of extensive research, both theoretical and applied. Let us provide a few comments, focusing on the preceding parametric PI schemes.

Architectural Issues

The choice of architectures for costs $\tilde{J}_\mu(i,r)$ and Q-factors $\tilde{Q}_\mu(i,u,r)$ is critical for the success of parametric approximation schemes. These architectures may involve the use of features, and they could be linear, or they could be nonlinear such as a neural network. A major advantage of a linear feature-based architecture is that the policy evaluations (4.29) and (4.31) involve linear least squares problems, which admit a closed-form solution. Moreover, when linear architectures are used, there is a broader variety of approximate policy evaluation methods with solid theoretical performance guarantees, such as TD(λ), LSTD(λ), and LSPE(λ), which will be summarized in Section 4.9, and are described in detail in several textbook sources.

Another interesting possibility for architecture choice has to do with *cost shaping*, which we discussed in Section 4.2. This possibility involves a modified cost per stage

$$\hat{g}(i,u,j) = g(i,u,j) + V(j) - V(i), \qquad i = 1,\ldots,n,$$

[cf. Eq. (4.10)] for SSP problems, where V can be any approximation to J^*. The corresponding formula for discounted problems is

$$\hat{g}(i,u,j) = g(i,u,j) + \alpha V(j) - V(i), \qquad i = 1,\ldots,n.$$

As noted in Section 4.2, cost shaping may change significantly the suboptimal policies produced by approximate DP methods and approximate PI in particular. Generally, V should be chosen close (at least in terms of "shape") to J^* or to the current policy cost function J_{μ^k}, so that the difference $J^* - V$ or $J_{\mu^k} - V$, respectively, can be approximated by an architecture that matches well the characteristics of the problem. It is possible to approximate either V or $\hat{J}$ with a parametric architecture or with a different approximation method, depending on the problem at hand. Moreover, in the context of approximate PI, the choice of V may change from one policy evaluation to the next.

The literature referenced at the end of the chapter provide some applications of cost shaping. An interesting possibility is to use complementary approximations for V and for J^* or J_{μ^k}. For example V may be approximated by a neural network-based approach that aims to discover the general form of J^* or J_{μ^k}, and then a different method may be applied to provide a local correction to V in order to refine the approximation. The next chapter will also illustrate this idea within the context of aggregation.

Exploration Issues

Generating an appropriate set of training pairs (i^s, β^s) or triplets (i^s, u^s, β^s) at the policy evaluation step of approximate PI poses considerable challenges, and the literature contains several related proposals. A generic difficulty has to do with *inadequate exploration*, which we noted in Section 4.6.2.

In particular, when evaluating a policy μ with trajectory reuse, we will be generating many cost or Q-factor samples that start from states frequently visited by μ, but this may bias the simulation by underrepresenting states that are unlikely to occur under μ. As a result, the cost or Q-factor estimates of these underrepresented states may be highly inaccurate, causing potentially serious errors in the calculation of the improved policy $\overline{\mu}$ via the policy improvement operation.

One possibility to improve the exploration of the state space is to use a large number of initial states to form a rich and representative subset, thereby limiting trajectory reuse. It may then be necessary to use relatively short trajectories to keep the cost of the simulation low. However, when using short trajectories it will be important to introduce a terminal cost function approximation in the policy evaluation step in order to make the cost sample β^s more accurate, as noted earlier.

There have been other related approaches to improve exploration, particularly in connection with the temporal difference methods to be discussed in Section 4.9. In some of these approaches, trajectories are generated through a mix of two policies: the policy being evaluated, sometimes called the *target policy*, to distinguish from the other policy, used with some probability at each stage, which is called *behavior policy* and is introduced to enhance exploration; see the end-of-chapter references. Also, methods that use a behavior policy are called *off-policy* methods, while methods that do not are called *on-policy* methods. Note, however, that it may still be difficult to ensure that the mixed on-and-off policy will induce sufficient exploration. This is an area of continuing research.

Oscillation Issues

Contrary to exact PI, which is guaranteed to yield an optimal policy, approximate PI produces a sequence of policies, which are only guaranteed to lie asymptotically within a certain error bound from the optimal; cf. Prop. 4.5.2. Moreover, the generated policies may oscillate. By this we mean that after a few iterations, policies tend to repeat in cycles.

This oscillation phenomenon occurs systematically in the absence of special conditions, for both optimistic and nonoptimistic PI methods. It can be observed even in very simple examples, and it is graphically explained in the books [BeT96] (Section 6.4.2) and [Ber12] (Section 6.4.3).

Oscillations can be particularly damaging, because there is no guarantee that the oscillating policies are "good" policies, and there is often no way to verify how well they perform relative to the optimal. We note, however, that oscillations can be avoided and approximate PI can be shown to converge under special conditions, which arise in particular when an aggregation approach is used; see Chapter 5 and the approximate policy iteration survey [Ber11a]. Also, when policies converge, there is a more favorable error bound, cf. Prop. 4.5.3.

4.7 EXACT AND APPROXIMATE LINEAR PROGRAMMING

The optimal cost function J^* has an interesting property that can be used to compute it by linear programming methods. In particular, for SSP problems, J^* is the "largest" J that satisfies the constraint

$$J(i) \leq g(i,u) + \sum_{j=1}^{n} p_{ij}(u)J(j), \qquad \text{for all } i = 1,\ldots,n \text{ and } u \in U(i). \tag{4.33}$$

It follows that $J^*(1),\ldots,J^*(n)$ solve the linear program

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{n} J(i) \\ &\text{subject to the constraint (4.33),} \end{aligned} \tag{4.34}$$

(see Fig. 4.7.1).

To verify this, let us use the VI algorithm to generate a sequence of vectors $J_k = \big(J_k(1),\ldots,J_k(n)\big)$ starting with an initial condition vector $J_0 = \big(J(1),\ldots,J(n)\big)$ such that

$$J_0(i) \leq \min_{u\in U(i)} \left[g(i,u) + \sum_{j=1}^{n} p_{ij}(u)J_0(j) \right] = J_1(i), \qquad \text{for all } i.$$

This inequality can be used to show that

$$J_0(i) \leq J_1(i) \leq \min_{u\in U(i)} \left[g(i,u) + \sum_{j=1}^{n} p_{ij}(u)J_1(j) \right] = J_2(i), \qquad \text{for all } i,$$

and similarly

$$J(i) = J_0(i) \leq J_k(i) \leq J_{k+1}(i), \qquad \text{for all } i.$$

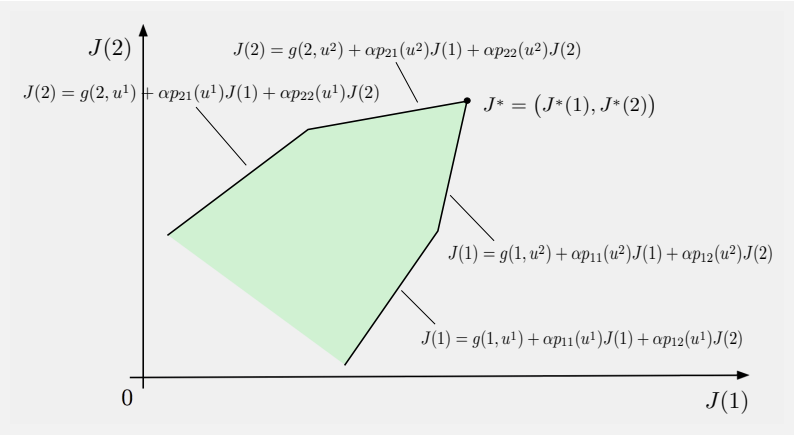

Figure 4.7.1 Linear program associated with a two-state SSP problem. The constraint set is shaded, and the objective to maximize is $J(1) + J(2)$. Note that because we have $J(i) \leq J^*(i)$ for all i and vectors J in the constraint set, the vector J^* maximizes any linear cost function of the form

$$\sum_{i=1}^{n} \beta_i J(i),$$

where $\beta_i \geq 0$ for all i. If $\beta_i > 0$ for all i, then J^* is the unique optimal solution of the corresponding linear program.

Since $J_k(i)$ converges to $J^*(i)$ as $k \to \infty$, it follows that we will also have

$$J(i) = J_0(i) \leq J^*(i), \qquad \text{for all } i.$$

Thus out of all J satisfying the constraint (4.33), J^* is the largest on a component-by-component basis.

Unfortunately, for large n the dimension of the linear program (4.34) can be very large and its solution can be impractical, particularly in the absence of special structure. In this case, we may consider finding an approximation to J^*, which can be used in turn to obtain a (suboptimal) policy through approximation in value space.

One possibility is to approximate $J^*(i)$ with a linear feature-based architecture

$$\tilde{J}(i,r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(i),$$

where $r = (r_1, \ldots, r_m)$ is a vector of parameters, and for each state i, $\phi_\ell(i)$ are some features. It is then possible to determine r by using $\tilde{J}(i,r)$ in place of J^* in the preceding linear programming approach. In particular,

we compute r as the solution of the program

$$\text{maximize} \quad \sum_{i \in \tilde{I}} \tilde{J}(i,r)$$

$$\text{subject to} \quad \tilde{J}(i,r) \leq \sum_{i=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}(j,r)\big), \quad i \in I,\ u \in \tilde{U}(i),$$

where $\tilde{I}$ is either the state space $I = \{1, \ldots, n\}$ or a suitably chosen subset of I, and $\tilde{U}(i)$ is either $U(i)$ or a suitably chosen subset of $U(i)$. This is a linear program because $\tilde{J}(i,r)$ is linear in the parameter vector r.

The major difficulty with this approximation approach is that while the dimension of r may be moderate, the number of constraints can be extremely large. It can be as large as nm, where n is the number of states and m is the maximum number of elements of the control constraint sets $U(i)$. Thus for a large problem it is essential to reduce drastically the number of constraints. Random sampling methods may be used to select a suitable subset of the constraints to enforce (perhaps using some known suboptimal policies), and progressively enrich the subset as necessary. With such constraint sampling schemes, the linear programming approach may be practical even for problems with a very large number of states. Its application, however, may require considerable sophistication, and a substantial amount of computation (see de Farias and VanRoy [DFV03], [DFV04], [DeF04]).

We finally mention the possibility of using linear programming to evaluate approximately the cost function J_μ of a stationary policy μ in the context of approximate PI. The motivation for this is that the linear program to evaluate a given policy involves fewer constraints.

4.8 Q-LEARNING

In this section we will discuss various Q-learning algorithms for discounted problems, which can be implemented in model-free fashion. The original method of this type is related to VI and can be used directly in the case of multiple policies. Instead of approximating the cost functions of successive policies as in the PI method, it updates the Q-factors associated with an *optimal* policy, thereby avoiding the multiple policy evaluation steps of PI. We will consider Q-learning as well as a variety of related methods with the shared characteristic that they involve exact or approximate Q-factors.

We first discuss the original form of Q-learning for discounted problems; the books [BeT96] and [Ber12] contain discussions of Q-learning for SSP problems. Then we discuss PI algorithms for Q-factors, including optimistic asynchronous versions, which lead to algorithms with reduced overhead per iteration. Finally we focus on Q-learning algorithms with Q-factor approximation.

Q-Learning: A Stochastic VI Algorithm

In the discounted problem, the optimal Q-factors are defined for all pairs (i,u) with $u \in U(i)$, by

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J^*(j)\big).$$

As discussed in Section 4.3, these Q-factors satisfy for all (i,u),

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{v\in U(j)} Q^*(j,v)\right),$$

and are the unique solution of this set of equations. Moreover the optimal Q-factors can be obtained by the VI algorithm $Q_{k+1} = FQ_k$, where F is the operator defined by

$$(FQ)(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{v\in U(j)} Q(j,v)\right), \quad \forall\ (i,u). \quad (4.35)$$

It is straightforward to show that F is a contraction with modulus α, similar to the DP operator T. Thus the algorithm $Q_{k+1} = FQ_k$ converges to Q^* from every starting point Q_0.

The original and most widely known Q-learning algorithm ([Wat89]) is a stochastic version of VI, whereby the expected value in Eq. (4.35) is suitably approximated by sampling and simulation. In particular, an infinitely long sequence of state-control pairs $\{(i_k,u_k)\}$ is generated according to some probabilistic mechanism. For each pair (i_k,u_k), a state j_k is generated according to the probabilities $p_{i_kj}(u_k)$. Then the Q-factor of (i_k,u_k) is updated using a stepsize $\gamma_k \in (0,1]$ while all other Q-factors are left unchanged:

$$Q_{k+1}(i,u) = (1-\gamma_k)Q_k(i,u) + \gamma_k(F_kQ_k)(i,u), \qquad \forall\ (i,u), \qquad (4.36)$$

where

$$(F_kQ_k)(i,u) = \begin{cases} g(i_k,u_k,j_k) + \alpha \min_{v\in U(j_k)} Q_k(j_k,v) & \text{if } (i,u) = (i_k,u_k), \\ Q_k(i,u) & \text{if } (i,u) \neq (i_k,u_k). \end{cases} \quad (4.37)$$

Note that $(F_kQ_k)(i_k,u_k)$ is a single sample approximation of the expected value defining $(FQ_k)(i_k,u_k)$ in Eq. (4.35).

To guarantee the convergence of the algorithm (4.36)-(4.37) to the optimal Q-factors, some conditions must be satisfied. Chief among these are that all state-control pairs (i,u) must be generated infinitely often within the infinitely long sequence $\{(i_k,u_k)\}$, and that the successor states

j must be independently sampled at each occurrence of a given state-control pair. Furthermore, the stepsize γ_k should satisfy

$$\gamma_k > 0, \quad \forall\, k, \qquad \sum_{k=0}^{\infty} \gamma_k = \infty, \qquad \sum_{k=0}^{\infty} \gamma_k^2 < \infty,$$

which are typical of stochastic approximation methods (see e.g, the books [BeT96], [Ber12], Section 6.1.4), as for example when $\gamma_k = c_1/(k + c_2)$, where c_1 and c_2 are some positive constants. In addition some other technical conditions should hold. A mathematically rigorous convergence proof was given in the paper [Tsi94], which embeds Q-learning within a broad class of asynchronous stochastic approximation algorithms. This proof (also reproduced in [BeT96]) combines the theory of stochastic approximation algorithms with the convergence theory of asynchronous iterative methods; see the books [BeT89] and [Ber16].

In practice, Q-learning has some drawbacks, the most important of which is that the number of Q-factors/state-control pairs (i, u) may be excessive. To alleviate this difficulty, we may introduce a Q-factor approximation architecture, which could be linear or nonlinear based for example on a neural network. One of these possibilities will be discussed next.

Optimistic Policy Iteration Methods with Q-Factor Approximation - SARSA

We have discussed so far Q-learning algorithms with an exact representation of Q-factors. We will now consider Q-learning with linear feature-based Q-factor approximation. As we noted earlier, we may view Q-factors as optimal costs of a certain discounted DP problem, whose states are the state-control pairs (i, u) in addition to the original states; cf. Fig. 4.2.2. We may thus apply the approximate PI methods discussed earlier. For this, we need to introduce a linear parametric architecture $\tilde{Q}(i, u, r)$,

$$\tilde{Q}(i, u, r) = \phi(i, u)'r,$$

where $\phi(i, u)$ is a feature vector that depends on both state and control.

We have already discussed in Section 4.6 a model-free approximate PI method that is based on Q-factors and least squares training/regression. There are also optimistic approximate PI methods, which use only a few samples in between policy updates. As an example, let us consider a Q-learning algorithm that uses a single sample between policy updates. At the start of iteration k, we have the current parameter vector r_k, we are at some state i_k, and we have chosen a control u_k. Then:

(1) We simulate the next transition (i_k, i_{k+1}) using the transition probabilities $p_{i_k j}(u_k)$.

(2) We generate the control u_{k+1} from the minimization

$$u_{k+1} \in \arg \min_{u \in U(i_{k+1})} \tilde{Q}(i_{k+1}, u, r_k).$$

[In some schemes, u_{k+1} is chosen with a small probability to be a different or random element of $U(i_{k+1})$ in order to enhance exploration.]

(3) We update the parameter vector via

$$r_{k+1} = r_k - \gamma_k \phi(i_k, u_k) q_{k,k},$$

where γ_k is a positive stepsize, and $q_{k,k}$ is given by

$$q_{k,k} = \phi(i_k, u_k)' r_k - \alpha \phi(i_{k+1}, u_{k+1})' r_k - g(i_k, u_k, i_{k+1}).$$

The vector $\phi(i_k, u_k) q_{k,k}$ can be interpreted as an approximate gradient direction based on an underlying regression procedure, and $q_{k,k}$ is referred to as a temporal difference (cf. Section 4.9).

The process is now repeated with r_{k+1}, i_{k+1}, and u_{k+1} replacing r_k, i_k, and u_k, respectively.

Extreme optimistic schemes of the type just described, including nonlinear architecture versions, have been used in practice, and are often referred to as SARSA (State-Action-Reward-State-Action); see e.g., the books [BeT96], [BBD10], [SuB18]. When Q-factor approximation is used, their behavior is very complex, their theoretical convergence properties are unclear, and there are no associated error bounds in the literature.

We finally note that in simulation-based PI methods for Q-factors, a major concern is the issue of exploration in the approximate evaluation step of the current policy μ, to ensure that state-control pairs $(i, u) \neq \big(i, \mu(i)\big)$ are generated sufficiently often in the simulation.

4.9 ADDITIONAL METHODS - TEMPORAL DIFFERENCES

In this section, we summarize a few additional methods for approximation in value space in infinite horizon problems. These include the important simulation-based temporal difference methods for policy evaluation with a linear parametric architecture. Our presentation is brief, somewhat abstract, and makes use of linear algebra mathematics. It may be skipped without loss of continuity. This is only a summary; it is meant to provide a connection to other material in this chapter, and orientation for further reading into both the optimization and artificial intelligence literature on the subject.

Approximate Policy Evaluation as a Projected Bellman Equation

Our main concern in policy evaluation is to solve approximately the Bellman equation corresponding to a given policy μ. Thus, for discounted problems, we are interested in solving the linear system of equations

$$J_\mu(i) = \sum_{i=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i),j\big) + \alpha J_\mu(j)\Big), \qquad i = 1,\ldots,n,$$

or in shorthand,

$$J_\mu = T_\mu J_\mu, \tag{4.38}$$

where T_μ is the DP operator for μ, given by

$$(T_\mu J)(i) = \sum_{i=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i),j\big) + \alpha J(j)\Big), \qquad i = 1,\ldots,n.$$

Solving this equation by parametric approximation (cf. Section 4.6), amounts to replacing J_μ with some vector that lies within the manifold represented by the approximation architecture

$$\mathcal{M} = \Big\{\big(J(1,r),\ldots,J(n,r)\big) \;\Big|\; \text{all parameter vectors } r\Big\}. \tag{4.39}$$

For a linear architecture this manifold is a subspace of the form

$$\mathcal{M} = \{\Phi r \mid r \in \Re^m\}, \tag{4.40}$$

where $\Re^m$ denote the space of m-dimensional vectors, and Φ is an $n \times m$ matrix. The subspace $\mathcal{M}$ is the space spanned by the n-dimensional columns of Φ, which may be viewed as basis functions.

The approximate solution of systems of equations within an approximation manifold of the form (4.39) or (4.40) has a long history in scientific computation. A central approach involves the use of projections with respect to a weighted quadratic norm

$$\|J\|^2 = \sum_{i=1}^{n} \xi_i\big(J(i)\big)^2, \tag{4.41}$$

where $J(i)$ are the components of the vector J and ξ_i are some positive weights. The projection of a vector J onto the manifold is denoted by $\Pi(J)$. Thus

$$\Pi(J) \in \arg\min_{V\in\mathcal{M}} \|J - V\|^2. \tag{4.42}$$

Note that for a nonlinear parametric architecture, such as a neural network, the projection may not exist and may not be unique. However, in the

case of a linear architecture, where the approximation manifold $\mathcal{M}$ is a subspace, the projection does exist and is unique; this is a consequence of the fundamental projection theorem of real analysis.

A key idea is to replace a general linear equation of the form $J = b + AJ$, where b and A are an n-dimensional vector and a square matrix, respectively, with a projected equation $J = \Pi(b+AJ)$ and use its solution as an approximation of the solution of the original. In particular, for the case of the approximation subspace of vectors Φr [cf. Eq. (4.40)], the projected equation has the form

$$\Phi r = \Pi(b + A\Phi r).$$

It turns out that the principal methods used in approximate policy evaluation involve the solution by simulation of a projected version of either the Bellman equation $J_\mu = T_\mu J_\mu$ of Eq. (4.38), or an "equivalent" equation whose solution is also J_μ.

Projection by Monte Carlo Simulation

Computing the projection $\Pi(J)$ of a given vector J involves the solution of a weighted least squares problem; cf. Eqs. (4.41)-(4.42). The least squares training problems of the type occurring in approximate VI and PI contexts, can be viewed as simulation-based approximations of projection operations. We formalize this in the following example for the case of a linear architecture.

Suppose that we wish to compute the projection $\Pi(J)$ of a vector $J \in \Re^n$ onto the subspace

$$\mathcal{M} = \{\Phi r \mid r \in \Re^m\},$$

where $\Re^n$ and $\Re^m$ denote the spaces of n-dimensional and m-dimensional vectors, and Φ is an $n \times m$ matrix with rows denoted by $\phi(i)'$, $i = 1, \ldots, n$. Here we use the notational convention that all vectors are column vectors, and prime denotes transposition, so $\phi(i)'$ is an m-dimensional row vector, and the subspace $\mathcal{M}$ may be viewed as the space spanned by the n-dimensional columns of Φ. The projection is with respect to a weighted Euclidean norm of Eq. (4.41), so $\Pi(J)$ is of the form Φr^*, where†

$$r^* \in \arg\min_{r\in\Re^m} \|\Phi r - J\|_\xi^2 = \arg\min_{r\in\Re^m} \sum_{i=1}^n \xi_i\big(\phi(i)'r - J(i)\big)^2. \tag{4.43}$$

† In the case of a nonlinear architecture $\tilde{J}(i,r)$, the projection $\Pi(J)$ is obtained as $\tilde{J}(i,r^*)$ where r^* is the solution of the nonlinear least squares problem

$$r^* \in \arg\min_{r\in\Re^m} \|\tilde{J}(i,r) - J\|_\xi^2 = \arg\min_{r\in\Re^m} \sum_{i=1}^n \xi_i\big(\tilde{J}(i,r) - J(i)\big)^2.$$

The above minimization can be approximated by a minimization involving sampled terms, similar to the linear architecture case.

By setting to 0 the gradient at r^* of the minimized expression above,

$$2\sum_{i=1}^{n}\xi_i\phi(i)\big(\phi(i)'r^* - J(i)\big) = 0,$$

we obtain the solution in closed form,

$$r^* = \left(\sum_{i=1}^{n}\xi_i\phi(i)\phi(i)'\right)^{-1}\sum_{i=1}^{n}\xi_i\phi(i)J(i), \tag{4.44}$$

assuming that the inverse above exists. The difficulty here is that when n is very large, the matrix-vector calculations in this formula can be very time-consuming.

On the other hand, assuming (by normalizing ξ if necessary) that $\xi = (\xi_1, \ldots, \xi_n)$ is a probability distribution, we may view the two terms in Eq. (4.44) as expected values with respect to ξ, and approximate them by Monte Carlo simulation. In particular, suppose that we generate a set of index samples i^s, $s = 1, \ldots, q$, according to the distribution ξ, and form the Monte Carlo estimates

$$\frac{1}{q}\sum_{s=1}^{q}\phi(i^s)\phi(i^s)' \approx \sum_{i=1}^{n}\xi_i\phi(i)\phi(i)', \qquad \frac{1}{q}\sum_{t=1}^{q}\phi(i^s)J(i^s) \approx \sum_{i=1}^{n}\xi_i\phi(i)J(i).$$

We can then estimate r^* using the corresponding approximation of Eq. (4.44):

$$\overline{r} = \left(\sum_{t=1}^{q}\phi(i^s)\phi(i^s)'\right)^{-1}\sum_{t=1}^{q}\phi(i^s)J(i^s), \tag{4.45}$$

(assuming sufficient samples are obtained to ensure the existence of the inverse above).† This is also equivalent to estimating r^* by approximating the least squares minimization (4.43) with the following least squares training problem

$$\overline{r} \in \arg\min_{r\in\Re^m}\sum_{s=1}^{q}\big(\phi(i^s)'r - J(i^s)\big)^2. \tag{4.46}$$

Thus simulation-based projection can be implemented in two equivalent ways:

† The preceding derivation and the formula (4.45) actually make sense even if $\xi = (\xi_1, \ldots, \xi_n)$ has some zero components, as long as the inverses in Eqs. (4.44) and (4.45) exist. This is related to the concept of *seminorm projection*; see [YuB12] for an approximate DP-related discussion.

(a) Replacing expected values in the exact projection formula (4.44) by simulation-based estimates [cf. Eq. (4.45)].

(b) Replacing the exact least squares problem (4.43) with a simulation-based least squares approximation [cf. Eq. (4.46)].

These dual possibilities of implementing projection by simulation can be used interchangeably. In particular, *the least squares training problems considered in this book may be viewed as simulation-based approximate projection calculations*.

Generally, we wish that the estimate $\overline{r}$ converges to r^* as the number of samples q increases. An important point is that it is not necessary that the simulation produces independent samples. Instead it is sufficient that the long term empirical frequencies by which the indices i appear in the simulation sequence are consistent with the probabilities ξ_i of the projection norm, i.e.,

$$\xi_i = \lim_{k\to\infty} \frac{1}{q} \sum_{s=1}^{q} \delta(i^s = i), \qquad i = 1, \ldots, n, \tag{4.47}$$

where $\delta(i^s = i) = 1$ if $i^s = i$ and $\delta(i^s = i) = 0$ if $i^s \neq i$.

Another important point is that the probabilities ξ_i need not be predetermined. In fact, often the exact values of ξ_i do not matter much, and one may wish to first specify a reasonable and convenient sampling scheme, and let ξ_i be implicitly specified via Eq. (4.47).

Projected Equation View of Approximate Policy Evaluation

Let us now discuss the approximate policy evaluation method for costs of Section 4.6.2 [cf. Eq. (4.29)]. It can be interpreted in terms of a projected equation, written abstractly as

$$\tilde{J}_\mu \approx \Pi(T_\mu^N \hat{J}), \tag{4.48}$$

where:†

(a) $\hat{J}$ is some initial guess of J_μ, and $\tilde{J}_\mu$ is the vector

$$\tilde{J}_\mu = \big(J(1,\overline{r}), \ldots, J(n,\overline{r})\big),$$

which is the approximate policy evaluation of μ, used in the policy improvement operation (4.30). Here $\overline{r}$ is the solution of the training problem (4.29).

† The equation (4.48) assumes that all trajectories have equal length N, and thus does not allow trajectory reuse. If trajectories of different lengths are allowed, the term T_μ^N in the equation should be replaced by a more complicated weighted sum of powers of T_μ; see the paper [YuB12] for related ideas.

(b) T_μ is the DP operator corresponding to μ, which maps a vector $J = \big(J(1), \ldots, J(n)\big)$ into the vector $T_\mu J$ with components

$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}(\mu(i))\Big(g\big(i, \mu(i), j\big) + \alpha J(j)\Big), \qquad i = 1, \ldots, n. \tag{4.49}$$

(c) T_μ^N denotes the N-fold application of the operator T_μ, where N is the length of the sample trajectories used in the least squares regression problem (4.29). In particular, $(T_\mu^N \hat{J})(i)$ is the cost associated with starting at i, using μ for N stages, and incurring a terminal cost specified by the terminal cost function $\hat{J}$. The sample state-cost pairs (i^s, β^s) are obtained from trajectories corresponding to this N-stage problem.

(d) $\Pi(T_\mu^N \hat{J})$ denotes projection of the vector $T_\mu^N \hat{J}$ on the manifold of possible approximating vectors $\mathcal{M}$ with respect to a weighted norm, where each weight ξ_i represents the relative frequency of the state i as initial state of a training trajectory. This projection is approximated by the least squares regression (4.29). In particular, the cost samples β^s of the training set are noisy samples of the values $(T_\mu^N \hat{J})(i^s)$, and the projection is approximated with a least squares minimization, to yield the function $\tilde{J}_\mu$ of Eq. (4.48).

Suppose now that $T_\mu^N \hat{J}$ is close to J_μ (which happens if either N is large or $\hat{J}$ is close to J_μ, or both) and the number of samples q is large (so that the simulation-based regression approximates well the projection operation Π). Then from Eq. (4.48), the approximate evaluation $\tilde{J}_\mu$ of μ approaches the projection of J_μ on the approximation manifold (4.39), which can be viewed as the best possible approximation of J_μ (at least relative to the distance metric defined by the weighted projection norm). This provides an abstract formal rationale for the parametric PI method of Section 4.6.2, which is based on Eq. (4.48).

TD(λ), LSTD(λ), and LSPE(λ)

Projected equations also fundamentally underlie *temporal difference methods* (TD for short), a prominent class of simulation-based methods for approximate evaluation of a policy. Examples of such methods are TD(λ), LSTD(λ), and LSPE(λ), where λ is a scalar with $0 \leq \lambda < 1$.†

These three methods require a linear parametric approximation architecture $\tilde{J}_\mu = \Phi r$, and all aim at the same problem. This is the problem of solving a projected equation of the form

$$\Phi r = \Pi\big(T_\mu^{(\lambda)} \Phi r\big), \tag{4.50}$$

† TD stands for "temporal difference," LSTD stands for "least squares temporal difference," and LSPE stands for "least squares policy evaluation."

where T_μ is the operator (4.49), $T_\mu^{(\lambda)}J$ is defined by

$$(T_\mu^{(\lambda)}J)(i) = (1-\lambda)\sum_{\ell=0}^{\infty}\lambda^\ell(T_\mu^{\ell+1}J)(i), \qquad i = 1,\ldots,n,$$

and Π is projection on the approximation subspace

$$\mathcal{M} = \{\Phi r \mid r \in \Re^m\},$$

with respect to some weighted projection norm. One interpretation of the equation $J = T_\mu^{(\lambda)}J$ is as a *multistep version of Bellman's equation*. It has the same solution, J_μ, as the "one-step" Bellman equation $J = T_\mu J$, which corresponds to $\lambda = 0$.

Of course the projected equation (4.50) cannot be solved exactly when the number of states n is large, since the projection is a high dimensional linear operation that requires operations of order n. Instead the key idea is to *replace the projection by a simulation-based approximate projection*, discussed earlier. This yields the equation,

$$\Phi r = \tilde{\Pi}\big(T_\mu^{(\lambda)}\Phi r\big), \tag{4.51}$$

where $\tilde{\Pi}$ is the approximate projection obtained by sampling.

For a more concrete description, let the ith row of the matrix Φ be the m-dimensional row vector $\phi(i)'$, so that the cost $J_\mu(i)$ is approximated as the inner product $\phi(i)'r$:

$$J_\mu(i) \approx \phi(i)'r.$$

Suppose that we collect q samples of initial states i^s, $s = 1,\ldots,q$, together with the corresponding transition costs $g(i^s, i^{s+1})$, $s = 1,\ldots,q$. Then the parameter vector $\overline{r}$ that solves Eq. (4.51) is

$$\overline{r} \in \arg\min_r \sum_{s=1}^{q}\big(\phi(i^s)'r - \text{sample of } (T_\mu^{(\lambda)}\Phi\overline{r})(i^s)\big)^2, \tag{4.52}$$

[cf. Eq. (4.46)], and defines the approximate evaluation $\Phi\overline{r}$ of J_μ. This relation can be expressed as a linear equation, which in principle can be solved in closed form, and indeed LSTD(λ) does exactly that. By contrast LSPE(λ) and TD(λ) solve this relation iteratively.

We will first give a high level description of the three methods, and then provide a more concrete description for the simpler case where $\lambda = 0$.

(a) The LSTD(λ) method, after the q samples have been collected, solves the relation (4.52) by matrix inversion, taking advantage of the fact

that this relation can be written as a linear equation. In particular, it can be written as

$$C\overline{r} = d, \tag{4.53}$$

where C is some $m \times m$ square matrix, and d is an m-dimensional vector. The components of C and d are explicitly computed, and LSTD(λ) produces the approximate cost function $\tilde{J}_\mu(i) = \Phi\overline{r}$ where $\overline{r} = C^{-1}d$ is the solution of Eq. (4.53).

(b) The LSPE(λ) method solves the projected equation (4.50) by using a simulation-based *projected value iteration*,

$$J_{k+1} = \tilde{\Pi}\big(T_\mu^{(\lambda)} J_k\big). \tag{4.54}$$

Here the projection is implemented iteratively, with sampling-based least squares regression, in a manner that resembles the incremental aggregated method of Section 3.1.3.

(c) The TD(λ) method is a simpler iterative stochastic approximation method for solving the linear equation (4.53). It can also be viewed as a stochastic gradient method, or as a stochastic version of the proximal algorithm for solving this linear equation (see the author's papers [Ber16c] and [Ber18d]).

An interesting question is how to select λ and what is its role. There is a bias-variance tradeoff here, similar to the one we discussed in Section 4.6. We will address this issue later in this section.

TD(0), LSTD(0), and LSPE(0)

Let us describe in more detail LSTD(0) for evaluation of a given policy μ. We assume that the simulation generates a sample sequence of q transitions using μ:

$$(i^1, j^1), (i^2, j^2), \ldots, (i^q, j^q),$$

with corresponding transition costs

$$g(i^1, j^1), g(i^2, j^2), \ldots, g(i^q, j^q).$$

Here, to simplify notation, we do not show the dependence of the transition costs on the control applied by μ. Let the ith row of the matrix Φ be the m-dimensional row vector $\phi(i)'$, so that the cost $J_\mu(i)$ is approximated as the inner product $\phi(i)'r$:

$$J_\mu(i) \approx \phi(i)'r.$$

Since $\lambda = 0$, we have $T^{(\lambda)} = T$, the samples of $T_\mu\Phi\overline{r}$ in Eq. (4.52) are

$$g(i^s, i^{s+1}) + \alpha\phi(i^{s+1})'\overline{r},$$

and the least squares problem in Eq. (4.52) has the form

$$\min_r \sum_{s=1}^{q} \big(\phi(i^s)'r - g(i^s, i^{s+1}) - \alpha\phi(i^{s+1})'\overline{r}\big)^2. \tag{4.55}$$

By setting the gradient of the minimized expression to zero, we obtain the condition for $\overline{r}$ to attain the above minimum:

$$\sum_{s=1}^{q} \phi(i^s)\big(\phi(i^s)'\overline{r} - g(i^s, i^{s+1}) - \alpha\phi(i^{s+1})'\overline{r}\big) = 0. \tag{4.56}$$

Solving this equation for $\overline{r}$ yields the LSTD(0) solution:

$$\overline{r} = \left(\sum_{s=1}^{q} \phi(i^s)\big(\phi(i^s) - \alpha\phi(i^{s+1})\big)'\right)^{-1} \sum_{s=1}^{q} \phi(i^s)g(i^s, i^{s+1}). \tag{4.57}$$

Note that the inverse in the preceding equation must exist for the method to be well-defined; otherwise the iteration has to be modified. A modification may also be needed when the matrix inverted is nearly singular; in this case the simulation noise may introduce serious numerical problems. Various methods have been developed to deal with the near singularity issue; see Wang and Bertsekas [WaB13a], [WaB13b], and the DP textbook [Ber12], Section 7.3.

The expression

$$d^s(\overline{r}) = \phi(i^s)'\overline{r} - g(i^s, i^{s+1}) - \alpha\phi(i^s)'\overline{r} \tag{4.58}$$

that appears in the least squares sum minimization (4.55) and Eq. (4.56) is referred to as the *temporal difference associated with the sth transition and parameter vector* $\overline{r}$. In the artificial intelligence literature, temporal differences are viewed as fundamental to learning and are accordingly interpreted, but we will not go further in this direction; see the RL textbooks that we have cited.

The LSPE(0) method is similarly derived. It consists of a simulation-based approximation of the projected value iteration method

$$J_{k+1} = \tilde{\Pi}\big(T_\mu J_k\big),$$

[cf. Eq. (4.54)]. At the kth iteration, it uses only the samples $s = 1, \ldots, k$, and updates the parameter vector according to

$$r^{k+1} = r^k - \left(\sum_{s=1}^{k} \phi(i^s)\phi(i^s)'\right)^{-1} \sum_{s=1}^{k} \phi(i^s)d^s(r^s), \qquad k = 1, 2, \ldots, \tag{4.59}$$

where $d^s(r^s)$ is the temporal difference of Eq. (4.58), evaluated at the iterate of iteration s; the form of this iteration is derived similar to the case of LSTD(0). After q iterations, when all the samples have been processed, the vector r^q obtained is the one used for the approximate evaluation of J_μ. Note that the inverse in Eq. (4.59) can be updated economically from one iteration to the next, using fast linear algebra operations (cf. the discussion of the incremental Newton method in Section 3.1.3).

Overall, it can be shown that LSTD(0) and LSPE(0) [with efficient matrix inversion in Eq. (4.59)] require essentially identical amount of work to process the q samples associated with the current policy μ [this is also true for the LSPD(λ) and LSTE(λ) methods; see [Ber12]]. An advantage offered by LSPE(0) is that because it is iterative, it allows carrying over the final parameter vector r^q, as a "hot start" when passing from one policy evaluation to the next, in the context of an approximate PI scheme.

The TD(0) method has the form

$$r^{k+1} = r^k - \gamma^k \phi(i^k) d^k(r^k), \qquad k = 1, 2, \ldots, \tag{4.60}$$

where γ^k is a diminishing stepsize parameter. It can be seen that TD(0) resembles an *incremental gradient* iteration for solving the least squares training problem (4.55), but with $\overline{r}$ replaced by the current iterate r^k. The reason is that the gradient of the typical kth term in the least squares sum of Eq. (4.55) is the vector $\phi(i^k)d^k(r^k)$ that appears in the TD(0) iteration (4.60) (cf. Section 3.1.3). Thus at each iteration, TD(0) uses only one sample, and changes r^k in the opposite direction to the corresponding incremental gradient using a stepsize γ^k that must be carefully controlled.

By contrast the LSPE(0) iteration (4.59) uses the full sum

$$\sum_{s=1}^{k} \phi(i^s) d^s(r^s),$$

which may be viewed as an *aggregated incremental* method, with scaling provided by the matrix $\left(\sum_{s=1}^{k} \phi(i^s)\phi(i^s)'\right)^{-1}$. This explains why TD(0) is generally much slower and more fragile than LSPE(0). On the other hand TD(0) is simpler than both LSTD(0) and LSPE(0), and does not require a matrix inversion, which may be inconvenient when the dimension m is large.

The properties, the analysis, and the implementation of TD methods in the context of approximate PI are quite complicated. In particular, the issue of exploration is important and must be addressed. Moreover there are convergence, oscillation, and reliability issues to contend with. LSTD(λ) relies on matrix inversion and not on iteration, so it does not have a serious convergence issue, but the system (4.53) may be singular or near singular, in which case very accurate simulation is needed to approximate C well enough for its inversion to be reliable; remedies for the

case of a singular or near singular system are discussed by Wang and Bertsekas [WaB13a], [WaB13b] (see also [Ber12], Section 7.3). LSPE(λ) has a convergence issue because the mapping $\Pi T_\mu^{(\lambda)}$ may not be a contraction mapping (even though T_μ is) and the projected value iteration (4.54) may not be convergent (it turns out that the mapping $\Pi T_\mu^{(\lambda)}$ is guaranteed to be a contraction for λ sufficiently close to 1).

Direct and Indirect Policy Evaluation Methods

In trying to compare the approximate policy evaluation methods discussed in this section, we may draw a distinction between *direct methods*, which aim to compute approximately the projection $\Pi(J_\mu)$, and *indirect methods*, which try to solve the projected equation (4.50).

The method of Section 4.6.2 is direct and is based on Eq. (4.48). In particular, as $N \to \infty$ and $q \to \infty$, it yields the approximate evaluation $\Pi(J_\mu)$. The TD methods are indirect, and aim at computing the solution of the projected equation (4.50). The solution of this equation is of the form Φr_λ^*, where the parameter vector r_λ^* depends on λ. In particular the projected equation solution Φr_λ^* is different from $\Pi(J_\mu)$. It can be shown that it satisfies the error bound

$$\|J_\mu - \Phi r_\lambda^*\|_\xi \leq \frac{1}{\sqrt{1-\alpha_\lambda^2}}\,\|J_\mu - \Pi J_\mu\|_\xi, \tag{4.61}$$

where

$$\alpha_\lambda = \frac{\alpha(1-\lambda)}{1-\alpha\lambda}$$

and $\|\cdot\|_\xi$ is a special projection norm of the form (4.41), where ξ is the steady-state probability distribution of the controlled system Markov chain under policy μ. Moreover as $\lambda \to 1$ the projected equation solution Φr_λ^* approaches $\Pi(J_\mu)$. Based on this fact, methods which aim to compute $\Pi(J_\mu)$, such as the direct method of Section 4.6.2 are sometimes called TD(1). We refer to [Ber12], Section 6.3, for an account of this analysis, which is beyond the scope of this book.

The difference $\Pi(J_\mu) - \Phi r_\lambda^*$ is commonly referred to as the *bias* and is illustrated in Figure 4.9.1. As indicated in this figure and as the estimate (4.61) suggests, there is a *bias-variance tradeoff*. As λ is decreased, the solution of the projected equation (4.50) changes and more bias is introduced relative to the "ideal" approximation ΠJ_μ (this bias can be embarassingly large as shown by examples in the paper [Ber95]). At the same time, however, the simulation samples of $T_\mu^{(\lambda)} J$ contain less noise as λ is decreased; see [Ber12], Section 6.3.

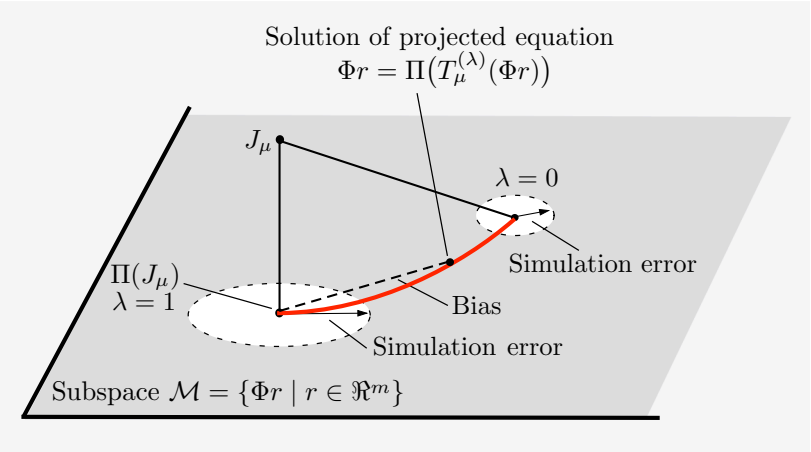

Figure 4.9.1 Illustration of the bias-variance tradeoff in estimating the solution of the projected equation for different values of λ. As λ increases from $\lambda = 0$ towards $\lambda = 1$, the solution Φr_λ^* of the projected equation $\Phi r = \Pi T^{(\lambda)}(\Phi r)$ approaches the projection ΠJ_μ. The difference $\Phi r_\lambda^* - \Pi J_\mu$ is the bias, and it decreases to 0 as λ approaches 1, while the simulation error variance increases.

4.10 APPROXIMATION IN POLICY SPACE

We will now consider briefly an alternative to approximation in value space: approximation within the space of policies. In particular, we parametrize the set of stationary policies with a parameter vector r and denote them by $\tilde{\mu}(r)$, with components $\tilde{\mu}(i, r)$, $i = 1, \ldots, n$. The parametrization may be feature-based and may involve a neural network. The idea is then to optimize some measure of performance with respect to the parameter vector r.

Note that it is possible for a suboptimal control scheme to employ both types of approximation: in policy space and in value space, with a distinct architecture for each case. When neural networks are used, this is known as the simultaneous use of a "policy network" (or "actor network") and a "value network" (or "critic network"), each with its own set of parameters (see the following discussion on expert training).

In what follows we will discuss briefly two training approaches for approximation in policy space.

Training by Cost Optimization

According to the first approach, we parametrize the policies by the parameter vector r, and we optimize the corresponding expected cost over r. In particular, we determine r through the minimization

$$\min_r E\big\{J_{\tilde{\mu}(r)}(i_0)\big\},$$

where $J_{\tilde{\mu}(r)}(i_0)$ is the cost of the policy $\tilde{\mu}(r)$ starting from the initial state i_0, and the expected value above is taken with respect to a suitable probability distribution of the initial state i_0.

We may use a gradient-based approach for minimizing $E\{J_{\tilde{\mu}(r)}(x_0)\}$ over r when the required gradients can be computed or approximated. Methods in this category are generally known as *policy gradient methods*. There is extensive literature on this subject to which we refer for details. The main drawback of policy gradient methods is potential unreliability due to slow convergence and the presence of local minima.

We may also use a method of random search within the space of the parameter vector r. The *cross-entropy method* (see Rubinstein and Kroese [RuK04], [RuK17], de Boer et al [BKM05]) has gained considerable attention in this regard. It bears some resemblance to policy gradient methods, in that it generates a parameter sequence $\{r_k\}$, by changing r_k to r_{k+1} in a direction of "improvement." This direction is obtained based on randomly collected cost samples using the policy $\tilde{\mu}(r_k)$, which are then screened and accepted or rejected based on a cost improvement criterion. This idea is similar to the ideas underlying evolutionary programming.

While random search methods are simple to implement, they may turn out to be very time-consuming and unreliable. However, the cross-entropy method has gained a favorable reputation through some impressive successes. In particular, it was used for learning a high-scoring strategy in the game of tetris; see Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09].

In an important special case of the cost optimization approach, the parametrization of the policies is indirect through a parametrization of an approximate cost function. In particular, for a given parameter vector r, we define

$$\tilde{\mu}(i,r) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \tilde{J}(j,r)\big),$$

where $\tilde{J}$ is a function of a given form that depends on r. For example, $\tilde{J}$ may be a linear feature-based architecture, with features possibly obtained through a separately trained neural network. The policies $\tilde{\mu}(r)$ thus defined form a class of one-step lookahead policies parametrized by r. By optimizing over r the expected cost

$$E\{J_{\tilde{\mu}(r)}(i_0)\},$$

we end up with an optimal one-step lookahead policy within this class.

Expert Training

According to the second approach, we again parametrize the policies, and we choose the parameter r by "training" on a large number of sample state-control pairs (i^s, u^s), $s = 1, \ldots, q$, such that for each s, u^s is a "good"

control at state i^s. This can be done for example by solving the least squares problem

$$\min_r \sum_{s=1}^{q} \big\| u^s - \tilde{\mu}(i^s, r) \big\|^2 \tag{4.62}$$

(possibly with added regularization). In particular, we may determine u^s by a human or a software "expert" that can choose "near-optimal" controls at given states, so $\tilde{\mu}$ is trained to match the behavior of the expert. We have also discussed this approach in Section 2.1.3, in the context of finite horizon problems.

Another possibility is to suitably select a large number of sample states i^s, $s = 1, \ldots, q$, and generate the controls u^s, $s = 1, \ldots, q$, through a one-step lookahead minimization of the form

$$u^s = \arg \min_{u \in U(i^s)} \sum_{j=1}^{n} p_{ij}(u) \big\{ g(i^s, u, j) + \tilde{J}_{k+1}(j) \big\}, \tag{4.63}$$

where $\tilde{J}$ is a suitable one-step lookahead function (multistep lookahead can also be used). Similarly, once a parametric Q-factor approximation architecture $\tilde{Q}(i, u, r)$ is chosen, we can select a large number of sample states i^s, $s = 1, \ldots, q$, and then compute the controls u^s, $s = 1, \ldots, q$, through the one-step lookahead minimization

$$u^s = \arg \min_{u \in U(i^s)} \tilde{Q}(i^s, u, r). \tag{4.64}$$

In this case, we will be collecting sample state-control pairs (i^s, u^s), $s = 1, \ldots, q$, using approximation in value space through Eq. (4.63) or Eq. (4.64), and then applying approximation in policy space through Eq. (4.62).

Of course in the expert training approach we cannot expect to obtain a policy that performs better than the expert with which it is trained, in the case of Eq. (4.62), or the one-step lookahead policy that is based on the approximation $\tilde{J}$ or $\tilde{Q}$, in the case of Eq. (4.63) or Eq. (4.64), respectively. However, a major advantage is that once the parametrized policy is obtained, the on-line implementation of the policy is fast and does not involve extensive calculations such as minimizations of the form (4.63). This advantage is generally shared by schemes that are based on approximation in policy space.

4.11 NOTES AND SOURCES

In this chapter we have provided a gentle introduction to infinite horizon DP with a view towards approximate solution methods that are suitable for large-scale problems. We have restricted ourselves to finite-state problems

with perfect state information. Infinite-state problems as well as partial state information and average cost problems exhibit more complex behaviors and present many challenges for approximate DP methods. The theory of these problems is developed in several books, including the author's [Ber12] and [Ber18a]. The latter book contains much recent advanced research on infinite-state deterministic and stochastic shortest path problems. The book by Puterman [Put94] contains a detailed account of discounted and average cost finite-state Markovian decision problems.

The variational form of Bellman's equation (Section 4.2) has been used in various contexts, involving error bounds for value iteration, since the early days of DP theory, see e.g., [Ber12], Section 2.1.1. The variational form of Bellman's equation is also implicit in the adaptive aggregation framework of Bertsekas and Castanon [BeC89]. In the context of RL, the variational equation has been used in various algorithmic contexts under the name *reward shaping* or *potential-based shaping* (we have used the term "cost shaping" here as we are focusing on cost minimization); see e.g., the papers by Ng, Harada, and Russell [NHR99], Wiewiora [Wie03], Asmuth, Littman, and Zinkov [ALZ08], Devlin and Kudenko [DeK11], Grzes [Grz17] for some representative works. While reward shaping does not change the optimal policies of the original DP problem, it may change significantly the suboptimal policies produced by approximate DP methods that use linear feature-based approximation. Basically, with reward shaping and a linear approximation architecture, *V is used as an extra feature*. This is closely related with the idea of using approximate cost functions of policies as basis functions, already suggested in the neuro-dynamic programming book [BeT96], Section 3.1.4.

Fitted VI algorithms have been used for finite horizon problems since the early days of DP. They are conceptually simple and easily implementable, and they are in wide use for approximation of either optimal costs or Q-factors (see e.g., Gordon [Gor95], Longstaff and Schwartz [LoS01], Ormoneit and Sen [OrS02], Ernst, Geurts, and Wehenkel [EGW06], Antos, Munos, and Szepesvari [AMS07], and Munos and Szepesvari [MuS08]).

The approximate PI method of Section 4.6 has been proposed by Fern, Yoon, and Givan [FYG06], and variants have also been discussed and analyzed by several other authors. The method (with some variations) has been used to train a tetris playing computer program that performs impressively better than programs that are based on other variants of approximate policy iteration, and various other methods; see Scherrer [Sch13], Scherrer et al. [SGG15], and Gabillon, Ghavamzadeh, and Scherrer [GGS13], who also provide an analysis of the method.

Q-learning (Section 4.8) was first proposed by Watkins [Wat89], and had a major impact in the development of the field. A rigorous convergence proof of Q-learning was given by Tsitsiklis [Tsi94], in a more general framework that combined several ideas from stochastic approximation theory and the theory of distributed asynchronous computation. This proof covered

discounted problems, and SSP problems where all policies are proper. It also covered SSP problems with improper policies, assuming that the Q-learning iterates are either nonnegative or bounded. Convergence without the nonnegativity or the boundedness assumption was shown by Yu and Bertsekas [YuB13a]. Optimistic asynchronous versions of PI based on Q-learning, which have solid convergence properties, are given by Bertsekas and Yu [BeY10], [BeY12], [YuB13a].

The advantage updating idea, which was noted in the context of finite horizon problems in Section 3.3, can be readily extended to infinite horizon problems. In this context, it was proposed by Baird [Bai93], [Bai94]; see [BeT96], Section 6.6. A related variant of approximate policy iteration and Q-learning, called *differential training*, has been proposed by the author in [Ber97b] (see also Weaver and Baxter [WeB99]).

Projected equations (Section 4.9) underlie Galerkin methods, which have a long history in scientific computation. They are widely used for many types of problems, including the approximate solution of large linear systems arising from discretization of partial differential and integral equations. The connection of approximate policy evaluation based on projected equations with Galerkin methods was first discussed by Yu and Bertsekas [YuB10], and Bertsekas [Ber11c], and is potentially important as it may lead to cross-fertilization of ideas. However, the Monte Carlo simulation ideas that are central in approximate DP differentiate the projected equation methods of the present chapter from the Galerkin methodology. On the other hand, Galerkin methods apply to a wide range of problems, far beyond DP, and the simulation-based ideas of approximate DP can consequently be extended to apply more broadly (see [Ber12], Section 7.3).

Temporal difference methods originated in RL, where they are viewed as a means to encode the error in predicting future costs, which is associated with an approximation architecture. They were introduced in the works of Samuel [Sam59], [Sam67] on a checkers-playing program. The work by Sutton [Sut88], following earlier work by Barto, Sutton, and Anderson [BSA84], formalized temporal differences and proposed the TD(λ) method. This was a major development and motivated a lot of research in simulation-based DP, particularly following an impressive early success with the backgammon playing program of Tesauro [Tes92], [Tes94].

The three methods TD(λ), LSTD(λ), and LSPE(λ) are discussed in detail in the journal and textbook RL literature. For a discussion that is closely related to our presentation of Section 4.9, see Chapters 6 and 7 of the book [Ber12].

The convergence of TD(λ) was proved by Tsitsiklis and VanRoy [TsV97], with extensions in [TsV99a] and [TsV99b]. The author's papers [Ber16b], [Ber18d] describe the connection of TD and proximal methods, a central methodology in convex optimization. In particular, TD(λ) is shown to be a stochastic version of the proximal algorithm for solving linear systems of equations, and extensions of TD(λ) for solving nonlinear systems

of equations are described.

The LSTD(λ) algorithm was first proposed by Bradtke and Barto [BrB96] for $\lambda = 0$, and was extended for $\lambda > 0$ later by Boyan [Boy02]. Convergence analyses of LSTD(λ) under assumptions of increasing generality were given by Nedić and Bertsekas [NeB03], Bertsekas and Yu [BeY09], and Yu [Yu12].

The LSPE(λ) algorithm was first proposed by Bertsekas and Ioffe [BeI96] under the name λ-*policy iteration*. LSPE(λ) was also given in the book [BeT96], Section 2.3.1, with subsequent contributions by Nedic, Borkar, Yu, Scherrer, and the author [NeB03], [BBN04], [YuB07], [BeY09], [YuB09], [Ber11b], [Yu12], [Sch13], [Ber18].

In our discussion here, we did not go much into the implementation details of TD(λ), LSTD(λ), and LSPE(λ); see the approximate DP/RL textbooks cited earlier, and the paper by Bertsekas and Yu [BeY09], which adapts the TD methodology to the solution of large systems of linear equations.

4.12 APPENDIX: MATHEMATICAL ANALYSIS

In this appendix we provide proofs of the mathematical results stated in this chapter. We also prove some supplementary results that are described in the chapter without formal statement.

4.12.1 Proofs for Stochastic Shortest Path Problems

We provide the proofs of Props. 4.2.1-4.2.5 from Section 4.2. A key insight for the analysis is that the expected cost incurred within an m-stage block vanishes exponentially as the start of the block moves forward. In particular, the cost in the m stages between km and $(k+1)m-1$ is bounded in absolute value by $\rho^k C$, where

$$C = m \max_{\substack{i=1,\ldots,n \\ j=1,\ldots,n,t \\ u\in U(i)}} \big|g(i,u,j)\big|. \tag{4.65}$$

Thus, we have

$$\big|J_\pi(i)\big| \leq \sum_{k=0}^{\infty} \rho^k C = \frac{1}{1-\rho} C. \tag{4.66}$$

This shows that the "tail" of the cost series,

$$\sum_{k=mK}^{\infty} E\Big\{g\big(x_k, \mu_k(x_k), w_k\big)\Big\},$$

vanishes as K increases to ∞, since the probability that $x_{mK} \neq t$ decreases like ρ^K [cf. Eq. (4.6)]. Intuitively, since the "tail" of the cost series can

be neglected as $K \to \infty$, it is valid to take the limit in the finite horizon DP algorithm, and obtain the infinite horizon Bellman equation and VI convergence. Mathematically, this is the essence of the following proofs.

Proposition 4.2.1: (Convergence of VI) Given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\{J_k(i)\}$ generated by the VI algorithm

$$J_{k+1}(i) = \min_{u \in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u) \big(g(i,u,j) + J_k(j) \big) \right], \tag{4.67}$$

converges to the optimal cost $J^*(i)$ for each $i = 1, \ldots, n$.

Proof: For every positive integer K, initial state x_0, and policy $\pi = \{\mu_0, \mu_1, \ldots\}$, we break down the cost $J_\pi(x_0)$ into the portions incurred over the first mK stages and over the remaining stages:

$$\begin{aligned} J_\pi(x_0) &= \lim_{N \to \infty} E \left\{ \sum_{k=0}^{N-1} g\big(x_k, \mu_k(x_k), w_k\big) \right\} \\ &= E \left\{ \sum_{k=0}^{mK-1} g\big(x_k, \mu_k(x_k), w_k\big) \right\} + \lim_{N \to \infty} E \left\{ \sum_{k=mK}^{N-1} g\big(x_k, \mu_k(x_k), w_k\big) \right\}. \end{aligned}$$

The expected cost during the Kth m-stage cycle [stages Km to $(K+1)m-1$] is upper bounded by $C\rho^K$ [cf. Eqs. (4.6) and (4.66)], so that

$$\left| \lim_{N \to \infty} E \left\{ \sum_{k=mK}^{N-1} g\big(x_k, \mu_k(x_k), w_k\big) \right\} \right| \leq C \sum_{k=K}^{\infty} \rho^k = \frac{\rho^K C}{1-\rho}.$$

Also, denoting $J_0(t) = 0$, let us view J_0 as a terminal cost function and bound its expected value under π after mK stages. We have

$$\begin{aligned} \Big| E\big\{ J_0(x_{mK}) \big\} \Big| &= \left| \sum_{i=1}^{n} P(x_{mK} = i \mid x_0, \pi) J_0(i) \right| \\ &\leq \left(\sum_{i=1}^{n} P(x_{mK} = i \mid x_0, \pi) \right) \max_{i=1,\ldots,n} \big| J_0(i) \big| \\ &\leq \rho^K \max_{i=1,\ldots,n} \big| J_0(i) \big|, \end{aligned}$$

since the probability that $x_{mK} \neq t$ is less or equal to ρ^K for any policy. Combining the preceding relations, we obtain

$$\begin{aligned} -\rho^K \max_{i=1,\ldots,n} \big|J_0(i)\big| + J_\pi(x_0) - \frac{\rho^K C}{1-\rho} & \\ \leq E\left\{ J_0(x_{mK}) + \sum_{k=0}^{mK-1} g\big(x_k, \mu_k(x_k), w_k\big) \right\} & \qquad (4.68) \\ \leq \rho^K \max_{i=1,\ldots,n} \big|J_0(i)\big| + J_\pi(x_0) + \frac{\rho^K C}{1-\rho}. & \end{aligned}$$

Note that the expected value in the middle term of the above inequalities is the mK-stage cost of policy π starting from state x_0, with a terminal cost $J_0(x_{mK})$; the minimum of this cost over all π is equal to the value $J_{mK}(x_0)$, which is generated by the DP recursion (4.67) after mK iterations. Thus, by taking the minimum over π in Eq. (4.68), we obtain for all x_0 and K,

$$\begin{aligned} -\rho^K \max_{i=1,\ldots,n} \big|J_0(i)\big| + J^*(x_0) - \frac{\rho^K C}{1-\rho} & \\ \leq J_{mK}(x_0) & \\ \leq \rho^K \max_{i=1,\ldots,n} \big|J_0(i)\big| + J^*(x_0) + \frac{\rho^K C}{1-\rho}, & \end{aligned}$$

and by taking the limit as $K \to \infty$, we obtain

$$\lim_{K\to\infty} J_{mK}(x_0) = J^*(x_0)$$

for all x_0. Since

$$\big|J_{mK+\ell}(x_0) - J_{mK}(x_0)\big| \leq \rho^K C, \qquad \ell = 1, \ldots, m,$$

we see that $\lim_{K\to\infty} J_{mK+\ell}(x_0)$ is the same for all $\ell = 1, \ldots, m$, so that

$$\lim_{k\to\infty} J_k(x_0) = J^*(x_0).$$

Q.E.D.

Proposition 4.2.2: (Bellman's Equation) The optimal cost function $J^* = \big(J^*(1), \ldots, J^*(n)\big)$ satisfies for all $i = 1, \ldots, n$, the equation

$$J^*(i) = \min_{u\in U(i)} \left[p_{it}(u) g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J^*(j)\big) \right],$$

and in fact it is the unique solution of this equation.

Proof: By taking the limit as $k \to \infty$ in the DP iteration (4.67) and using the result of Prop. 4.2.1, we see that $J^*(1), \ldots, J^*(n)$ satisfy Bellman's equation (we are using here the fact that the limit and minimization operations commute when the minimization is over a finite number of alternatives). To show uniqueness, observe that if $J(1), \ldots, J(n)$ satisfy Bellman's equation, then the DP iteration (4.67) starting from $J(1), \ldots, J(n)$ just replicates $J(1), \ldots, J(n)$. It follows from the convergence result of Prop. 4.2.1 that $J(i) = J^*(i)$ for all i. **Q.E.D.**

Proposition 4.2.3: (VI and Bellman's Equation for Policies) For any stationary policy μ, the corresponding cost function $J_\mu = \big(J_\mu(1), \ldots, J_\mu(n)\big)$ satisfies for all $i = 1, \ldots, n$ the equation

$$J_\mu(i) = p_{it}\big(\mu(i)\big)g\big(i, \mu(i), t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + J_\mu(j)\Big),$$

and is in fact the unique solution of this equation. Furthermore, given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\big\{J_k(i)\big\}$ generated by the VI algorithm that is specific to μ,

$$J_{k+1}(i) = p_{it}\big(\mu(i)\big)g\big(i, \mu(i), t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + J_k(j)\Big),$$

converges to the cost $J_\mu(i)$ for each i.

Proof: Given the stationary policy μ, we can consider a modified stochastic shortest path problem, which is the same as the original except that the control constraint set contains only one element for each state i, the control $\mu(i)$; i.e., the control constraint set is $\tilde{U}(i) = \big\{\mu(i)\big\}$ instead of $U(i)$. From Prop. 4.2.2 we then obtain that $J_\mu(1), \ldots, J_\mu(n)$ solve uniquely Bellman's equation for this modified problem, i.e.,

$$J_\mu(i) = p_{it}\big(\mu(i)\big)g\big(i, \mu(i), t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i, \mu(i), j\big) + J_\mu(j)\Big),$$

for all $i = 1, \ldots, n$. From Prop. 4.2.1, the corresponding value iteration converges to $J_\mu(i)$. **Q.E.D.**

Proposition 4.2.4: (Optimality Condition) A stationary policy μ is optimal if and only if for every state i, $\mu(i)$ attains the minimum in Bellman's equation (4.7).

Proof: We have that $\mu(i)$ attains the minimum in Eq. (4.7) if and only if for all $i = 1, \ldots, n$, we have

$$\begin{aligned} J^*(i) &= p_{it}(u)g(i,u,t) + \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J^*(j)\big) \\ &= p_{it}\big(\mu(i)\big)g\big(i,\mu(i),t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i)\big) + J^*(j)\Big). \end{aligned}$$

Proposition 4.2.3 and this equation imply that $J_\mu(i) = J^*(i)$ for all i. Conversely, if $J_\mu(i) = J^*(i)$ for all i, Props. 4.2.2 and 4.2.3 imply this equation. **Q.E.D.**

Proposition 4.2.5: (Contraction Property of the DP Operator) The DP operator T defined by

$$(TJ)(i) = \min_{u \in U(i)} \left[p_{it}(u)g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J(j)\big) \right],$$

for all $i = 1, \ldots, n$, and vectors $J = \big(J(1), \ldots, J(n)\big)$, is a contraction mapping with respect to the weighted norm

$$\|J\| = \max_{i=1,\ldots,n} \frac{\big|J(i)\big|}{v(i)},$$

defined by some vector $v = \big(v(1), \ldots, v(n)\big)$ with positive components. In other words, there exists a positive scalar $\rho < 1$ such that for any two n-dimensional vectors J and J', we have

$$\|TJ - TJ'\| \leq \rho \|J - J'\|.$$

Proof: We first define the vector v using the problem of Example 4.2.1. In particular, we let $v(i)$ be the maximal expected number of steps to termination starting from state i. From Bellman's equation in Example

4.2.1, we have for all $i = 1, \ldots, n$, and stationary policies μ,

$$v(i) = 1 + \max_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) v(j) \geq 1 + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) v(j), \qquad i = 1, \ldots, n.$$

Thus we obtain for all μ,

$$\sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) v(j) \leq v(i) - 1 \leq \rho\, v(i), \qquad i = 1, \ldots, n, \tag{4.69}$$

where ρ is defined by

$$\rho = \max_{i=1,\ldots,n} \frac{v(i) - 1}{v(i)}.$$

Since $v(i) \geq 1$ for all i, we have $\rho < 1$.

We will now show that Eq. (4.69) implies the desired contraction property. Indeed, consider the operator T_μ, which when applied to a vector $J = \big(J(1), \ldots, J(n)\big)$ produces the vector $T_\mu J = \big((T_\mu J)(1), \ldots, (T_\mu J)(n)\big)$ defined by

$$(T_\mu J)(i) = p_{it}\big(\mu(i)\big) g\big(i, \mu(i), t\big) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) \Big(g\big(i, \mu(i), j\big) + J(j) \Big),$$

for all $i = 1, \ldots, n$. We have for all J, J', and i

$$\begin{aligned}
(T_\mu J)(i) &= (T_\mu J')(i) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) \big(J(j) - J'(j) \big) \\
&= (T_\mu J')(i) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) v(j) \frac{\big(J(j) - J'(j) \big)}{v(j)} \\
&\leq (T_\mu J')(i) + \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big) v(j)\, \|J - J'\| \\
&\leq (T_\mu J')(i) + \rho\, v(i)\, \|J - J'\|,
\end{aligned}$$

where the last inequality follows from Eq. (4.69). By minimizing both sides over all $\mu(i) \in U(i)$, we obtain

$$(TJ)(i) \leq (TJ')(i) + \rho\, v(i)\, \|J - J'\|, \qquad i = 1, \ldots, n.$$

Thus we have

$$\frac{(TJ)(i) - (TJ')(i)}{v(i)} \leq \rho\, \|J - J'\|, \qquad i = 1, \ldots, n.$$

Similarly, by reversing the roles of J and J', we obtain

$$\frac{(TJ')(i) - (TJ)(i)}{v(i)} \leq \rho \, \|J - J'\|, \qquad i = 1, \ldots, n.$$

By combining the preceding two inequalities, we have

$$\frac{\big|(TJ)(i) - (TJ')(i)\big|}{v(i)} \leq \rho \, \|J - J'\|, \qquad i = 1, \ldots, n,$$

and by maximizing the left-hand side over i, the contraction property $\|TJ - TJ'\| \leq \rho \|J - J'\|$ follows. **Q.E.D.**

4.12.2 Proofs for Discounted Problems

Since we have shown that the discounted problem can be converted to the equivalent SSP problem of Fig. 4.3.1, we can apply Props. 4.2.1-4.2.4. Then Props. 4.3.1-4.3.4 are obtained from the construction of Fig. 4.3.1. The contraction property of Prop. 4.3.5 can also be proved in the same way, since in the SSP problem of Fig. 4.3.1, the expected number of steps to terminate starting from a state $i \neq t$ can be obtained as the mean of a geometrically distributed random variable with parameter $1 - \alpha$:

$$v(i) = 1 \cdot (1 - \alpha) + 2 \cdot \alpha(1 - \alpha) + 3 \cdot \alpha^2(1 - \alpha) + \cdots = \frac{1}{1 - \alpha}, \quad i = 1, \ldots, n.$$

so that the modulus of contraction is

$$\rho = \frac{v(i) - 1}{v(i)} = \alpha.$$

Thus by applying Prop. 4.2.5, we obtain Prop. 4.3.5.

4.12.3 Convergence of Exact Policy Iteration

We provide a proof of the convergence of exact PI for the case of a discounted problem. The proof for the SSP problem is similar.

Proposition 4.5.1: (Convergence of Exact PI) For both the SSP and the discounted problems, the exact PI algorithm generates an improving sequence of policies [i.e., $J_{\mu^{k+1}}(i) \leq J_{\mu^k}(i)$ for all i and k] and terminates with an optimal policy.

Proof: For any k, consider the sequence generated by the VI algorithm for policy μ^{k+1}:

$$J_{N+1}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu^{k+1}(i)\big)\Big(g\big(i,\mu^{k+1}(i),j\big) + \alpha J_N(j)\Big), \qquad i = 1,\ldots,n,$$

where $N = 0, 1, \ldots$, and

$$J_0(i) = J_{\mu^k}(i), \qquad i = 1,\ldots,n.$$

From Eqs. (4.21) and (4.22), we have

$$\begin{aligned} J_0(i) &= \sum_{j=1}^{n} p_{ij}\big(\mu^{k}(i)\big)\Big(g\big(i,\mu^{k}(i),j\big) + \alpha J_0(j)\Big) \\ &\geq \sum_{j=1}^{n} p_{ij}\big(\mu^{k+1}(i)\big)\Big(g\big(i,\mu^{k+1}(i),j\big) + \alpha J_0(j)\Big) \\ &= J_1(i), \end{aligned}$$

for all i. By using the above inequality we obtain

$$\begin{aligned} J_1(i) &= \sum_{j=1}^{n} p_{ij}\big(\mu^{k+1}(i)\big)\Big(g\big(i,\mu^{k+1}(i),j\big) + \alpha J_0(j)\Big) \\ &\geq \sum_{j=1}^{n} p_{ij}\big(\mu^{k+1}(i)\big)\Big(g\big(i,\mu^{k+1}(i),j\big) + \alpha J_1(j)\Big) \\ &= J_2(i), \end{aligned}$$

for all i, and by continuing similarly we have

$$J_0(i) \geq J_1(i) \geq \cdots \geq J_N(i) \geq J_{N+1}(i) \geq \cdots, \qquad i = 1,\ldots,n. \tag{4.70}$$

Since by Prop. 4.3.3, $J_N(i) \to J_{\mu^{k+1}}(i)$, we obtain $J_0(i) \geq J_{\mu^{k+1}}(i)$ or

$$J_{\mu^k}(i) \geq J_{\mu^{k+1}}(i), \qquad i = 1,\ldots,n, \quad k = 0, 1, \ldots.$$

Thus the sequence of generated policies is improving, and since the number of stationary policies is finite, we must after a finite number of iterations, say $k+1$, obtain $J_{\mu^k}(i) = J_{\mu^{k+1}}(i)$ for all i. Then we will have equality throughout in Eq. (4.70), which means that

$$J_{\mu^k}(i) = \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{\mu^k}(j)\big), \qquad i = 1,\ldots,n.$$

Thus the costs $J_{\mu^k}(1),\ldots,J_{\mu^k}(n)$ solve Bellman's equation, and by Prop. 4.3.2, it follows that $J_{\mu^k}(i) = J^*(i)$ and that μ^k is optimal. **Q.E.D.**

4.12.4 Error Bounds for Approximate Policy Iteration

We focus on the discounted problem, and we make use of the contraction property of Prop. 4.3.5:

$$\|T_\mu J - T_\mu J'\| \leq \alpha \|J - J'\|, \qquad \|TJ - TJ'\| \leq \alpha \|J - J'\|,$$

for all J, J', and μ, where $\|J\|$ is the norm

$$\|J\| = \max_{i=1,\ldots,n} \big|J(i)\big|.$$

We want to prove the following error bound.

Proposition 4.5.2: (Error Bound for Approximate PI) Consider the discounted problem, and let $\{\mu^k\}$ be the sequence generated by the approximate PI algorithm defined by the approximate policy evaluation (4.26) and the approximate policy improvement (4.27). Then the policy error satisfies

$$\limsup_{k\to\infty} \|J_{\mu^k} - J^*\| \leq \frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2}.$$

The essence of the proof is contained in the following proposition, which quantifies the amount of approximate policy improvement at each iteration.

Proposition 4.12.1: Consider the discounted problem, and let J, $\overline{\mu}$, and μ satisfy

$$\|J - J_\mu\| \leq \delta, \qquad \|T_{\overline{\mu}} J - TJ\| \leq \epsilon, \tag{4.71}$$

where δ and ϵ are some scalars. Then

$$\|J_{\overline{\mu}} - J^*\| \leq \alpha \|J_\mu - J^*\| + \frac{\epsilon + 2\alpha\delta}{1-\alpha}. \tag{4.72}$$

Proof: The contraction property of T and $T_{\overline{\mu}}$ implies that

$$\|T_{\overline{\mu}} J_\mu - T_{\overline{\mu}} J\| \leq \alpha\delta, \qquad \|TJ - TJ_\mu\| \leq \alpha\delta,$$

and hence

$$T_{\overline{\mu}} J_\mu \leq T_{\overline{\mu}} J + \alpha\delta v, \qquad TJ \leq TJ_\mu + \alpha\delta v,$$

where v is the unit vector, i.e., $v(i) = 1$ for all i. Using also Eq. (4.71), we have

$$T_{\overline{\mu}}J_\mu \leq T_{\overline{\mu}}J + \alpha\delta\, v \leq TJ + (\epsilon + \alpha\delta)v \leq TJ_\mu + (\epsilon + 2\alpha)v. \tag{4.73}$$

Combining this inequality with $TJ_\mu \leq T_\mu J_\mu = J_\mu$, we obtain

$$T_{\overline{\mu}}J_\mu \leq J_\mu + (\epsilon + 2\alpha\delta)v. \tag{4.74}$$

We will show that this relation implies that

$$J_{\overline{\mu}} \leq J_\mu + \frac{\epsilon + 2\alpha\delta}{1-\alpha}. \tag{4.75}$$

Indeed, by applying $T_{\overline{\mu}}$ to both sides of Eq. (4.74), we obtain

$$T^2_{\overline{\mu}}J_\mu \leq T_{\overline{\mu}}J_\mu + \alpha(\epsilon + 2\alpha\delta)v \leq J_\mu + (1+\alpha)(\epsilon + 2\alpha\delta)v.$$

Applying $T_{\overline{\mu}}$ again to both sides of this relation, and continuing similarly, we have for all k,

$$T^k_{\overline{\mu}}J_\mu \leq J_\mu + (1 + \alpha + \cdots + \alpha^{k-1})(\epsilon + 2\alpha\delta)v.$$

By taking the limit as $k \to \infty$, and by using the VI convergence property $T^k_{\overline{\mu}}J_\mu \to J_{\overline{\mu}}$, we obtain Eq. (4.75).

Using now the contraction property of $T_{\overline{\mu}}$ and Eq. (4.75), we have

$$J_{\overline{\mu}} = T_{\overline{\mu}}J_{\overline{\mu}} = T_{\overline{\mu}}J_\mu + (T_{\overline{\mu}}J_{\overline{\mu}} - T_{\overline{\mu}}J_\mu) \leq T_{\overline{\mu}}J_\mu + \frac{\alpha(\epsilon + 2\alpha\delta)}{1-\alpha}v.$$

Subtracting J^* from both sides, we obtain

$$J_{\overline{\mu}} - J^* \leq T_{\overline{\mu}}J_\mu - J^* + \frac{\alpha(\epsilon + 2\alpha\delta)}{1-\alpha}v. \tag{4.76}$$

Also from the contraction property of T,

$$TJ_\mu - J^* = TJ_\mu - TJ^* \leq \alpha\|J_\mu - J^*\|v$$

which, in conjunction with Eq. (4.73), yields

$$T_{\overline{\mu}}J_\mu - J^* \leq TJ_\mu - J^* + (\epsilon + 2\alpha\delta)v \leq \alpha\|J_\mu - J^*\| + (\epsilon + 2\alpha\delta)v.$$

Combining this relation with Eq. (4.76), we obtain

$$J_{\overline{\mu}} - J^* \leq \alpha\|J_\mu - J^*\|\, v + \frac{\alpha(\epsilon + 2\alpha\delta)}{1-\alpha}v + (\epsilon + 2\alpha\delta)v = \alpha\|J_\mu - J^*\|\, v + \frac{\epsilon + 2\alpha\delta}{1-\alpha}v,$$

which is equivalent to the desired relation (4.72). **Q.E.D.**

Proof of Prop. 4.5.2: Applying Prop. 4.12.1, we have

$$\|J_{\mu^{k+1}} - J^*\| \leq \alpha \|J_{\mu^k} - J^*\| + \frac{\epsilon + 2\alpha\delta}{1-\alpha},$$

which by taking the lim sup of both sides as $k \to \infty$ yields the desired result. **Q.E.D.**

We next prove the error bound for approximate PI, assuming that the generated policy sequence is convergent.

Proposition 4.5.3: (Error Bound for Approximate PI when Policies Converge) Let $\overline{\mu}$ be a policy generated by the approximate PI algorithm under conditions (4.26), (4.27), and (4.28). Then we have

$$\max_{i=1,\ldots,n} \left| J_{\overline{\mu}}(i) - J^*(i) \right| \leq \frac{\epsilon + 2\alpha\delta}{1-\alpha}.$$

Proof: Let $\bar{J}$ be the cost vector obtained by approximate policy evaluation of $\overline{\mu}$. Then in view of Eqs. (4.26), (4.27), we have

$$\|\bar{J} - J_{\overline{\mu}}\| \leq \delta, \qquad \|T_{\overline{\mu}}\bar{J} - T\bar{J}\| \leq \epsilon.$$

From this relation and the fact $J_{\overline{\mu}} = T_{\overline{\mu}} J_{\overline{\mu}}$, we have

$$\begin{aligned} \|TJ_{\overline{\mu}} - J_{\overline{\mu}}\| &\leq \|TJ_{\overline{\mu}} - T\bar{J}\| + \|T\bar{J} - T_{\overline{\mu}}\bar{J}\| + \|T_{\overline{\mu}}\bar{J} - J_{\overline{\mu}}\| \\ &= \|TJ_{\overline{\mu}} - T\bar{J}\| + \|T\bar{J} - T_{\overline{\mu}}\bar{J}\| + \|T_{\overline{\mu}}\bar{J} - T_{\overline{\mu}}J_{\overline{\mu}}\| \\ &\leq \alpha\|J_{\overline{\mu}} - \bar{J}\| + \epsilon + \alpha\|\bar{J} - J_{\overline{\mu}}\| \\ &\leq \epsilon + 2\alpha\delta. \end{aligned} \tag{4.77}$$

For every k, by using repeatedly the triangle inequality and the contraction property of T, we have

$$\|T^k J_{\overline{\mu}} - J_{\overline{\mu}}\| \leq \sum_{\ell=1}^{k} \|T^\ell J_{\overline{\mu}} - T^{\ell-1} J_{\overline{\mu}}\| \leq \sum_{\ell=1}^{k} \alpha^{\ell-1} \|TJ_{\overline{\mu}} - J_{\overline{\mu}}\|,$$

and by taking the limit as $k \to \infty$,

$$\|J^* - J_{\overline{\mu}}\| \leq \frac{1}{1-\alpha} \|TJ_{\overline{\mu}} - J_{\overline{\mu}}\|.$$

Combining this relation with Eq. (4.77), we obtain the desired error bound. **Q.E.D.**