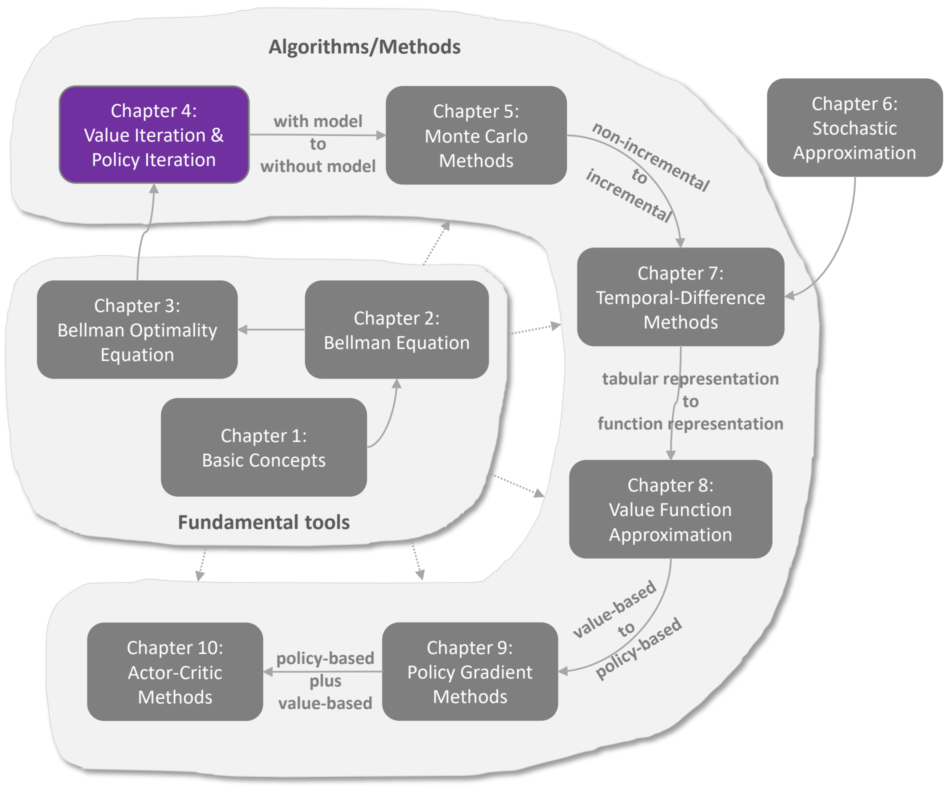
Chapter 4

Value Iteration and Policy Iteration



Algorithms/Methods
Chapter 4: Value Iteration & Policy Iteration
with model to without model
Chapter 5: Monte Carlo Methods
non-incremental to incremental
Chapter 6: Stochastic Approximation
Chapter 7: Temporal-Difference Methods
Chapter 3: Bellman Optimality Equation
Chapter 2: Bellman Equation
Chapter 1: Basic Concepts
Fundamental tools
tabular representation to function representation
Chapter 8: Value Function Approximation
value-based to policy-based
Chapter 10: Actor-Critic Methods
policy-based plus value-based
Chapter 9: Policy Gradient Methods

Figure 4.1: Where we are in this book.

With the preparation in the previous chapters, we are now ready to present the first algorithms that can find optimal policies. This chapter introduces three algorithms that are closely related to each other. The first is the value iteration algorithm, which is exactly the algorithm suggested by the contraction mapping theorem for solving the Bellman optimality equation as discussed in the last chapter. We focus more on the implementation details of this algorithm in the present chapter. The second is the policy iteration algorithm, whose idea is widely used in reinforcement learning algorithms. The third is the truncated policy iteration algorithm, which is a unified algorithm that includes the value iteration and policy iteration algorithms as special cases.

The algorithms introduced in this chapter are called *dynamic programming* algorithms [10, 11], which require the system model. These algorithms are important foundations of the model-free reinforcement learning algorithms introduced in the subsequent chapters. For example, the Monte Carlo algorithms introduced in Chapter 5 can be immediately obtained by extending the policy iteration algorithm introduced in this chapter.

4.1 Value iteration

This section introduces the *value iteration* algorithm. It is exactly the algorithm suggested by the contraction mapping theorem for solving the Bellman optimality equation, as introduced in the last chapter (Theorem 3.3). In particular, the algorithm is

$$v_{k+1} = \max_{\pi \in \Pi}(r_\pi + \gamma P_\pi v_k), \quad k = 0, 1, 2, \ldots$$

It is guaranteed by Theorem 3.3 that v_k and π_k converge to the optimal state value and an optimal policy as $k \to \infty$, respectively.

This algorithm is iterative and has two steps in every iteration.

◇ The first step in every iteration is a *policy update* step. Mathematically, it aims to find a policy that can solve the following optimization problem:

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k),$$

where v_k is obtained in the previous iteration.

◇ The second step is called a *value update* step. Mathematically, it calculates a new value v_{k+1} by

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k, \tag{4.1}$$

where v_{k+1} will be used in the next iteration.

The value iteration algorithm introduced above is in a matrix-vector form. To implement this algorithm, we need to further examine its elementwise form. While the matrix-vector form is useful for understanding the core idea of the algorithm, the elementwise form is necessary for explaining the implementation details.

4.1.1 Elementwise form and implementation

Consider the time step k and a state s.

◇ First, the elementwise form of the *policy update step* $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_k)$ is

$$\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) \underbrace{\left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_k(s') \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}.$$

We showed in Section 3.3.1 that the optimal policy that can solve the above optimization problem is

$$\pi_{k+1}(a|s) = \begin{cases} 1 & a = a_k^*(s), \\ 0 & a \neq a_k^*(s), \end{cases} \tag{4.2}$$

where $a_k^*(s) = \arg\max_a q_k(s,a)$. If $a_k^*(s) = \arg\max_a q_k(s,a)$ has multiple solutions, we can select any of them without affecting the convergence of the algorithm. Since the new policy π_{k+1} selects the action with the greatest $q_k(s,a)$, such a policy is called greedy.

◇ Second, the elementwise form of the *value update step* $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$ is

$$v_{k+1}(s) = \sum_a \pi_{k+1}(a|s) \underbrace{\left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_k(s') \right)}_{q_k(s,a)}, \quad s \in \mathcal{S}.$$

Substituting (4.2) into the above equation gives

$$v_{k+1}(s) = \max_a q_k(s,a).$$

In summary, the above steps can be illustrated as

$$v_k(s) \to q_k(s,a) \to \text{new greedy policy } \pi_{k+1}(s) \to \text{new value } v_{k+1}(s) = \max_a q_k(s,a)$$

The implementation details are summarized in Algorithm 4.1.

One problem that may be confusing is whether v_k in (4.1) is a state value. The answer is no. Although v_k eventually converges to the optimal state value, it is not ensured to satisfy the Bellman equation of any policy. For example, it does not satisfy $v_k = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$ or $v_k = r_{\pi_k} + \gamma P_{\pi_k} v_k$ in general. It is merely an intermediate value generated by the algorithm. In addition, since v_k is not a state value, q_k is not an action value.

4.1.2 Illustrative examples

We next present an example to illustrate the step-by-step implementation of the value iteration algorithm. This example is a two-by-two grid with one forbidden area (Fig-

Algorithm 4.1: Value iteration algorithm

```
Initialization: The probability models $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $v_0$.
Goal: Search for the optimal state value and an optimal policy for solving the Bellman optimality equation.

While $v_k$ has not converged in the sense that $\|v_k - v_{k-1}\|$ is greater than a predefined small threshold, for the $k$th iteration, do
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}(s)$, do
            q-value: $q_k(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_k(s')$
        Maximum action value: $a_k^*(s) = \arg\max_a q_k(s,a)$
        Policy update: $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
        Value update: $v_{k+1}(s) = \max_a q_k(s,a)$
```

q-table	a_1	a_2	a_3	a_4	a_5
s_1	$-1+\gamma v(s_1)$	$-1+\gamma v(s_2)$	$0+\gamma v(s_3)$	$-1+\gamma v(s_1)$	$0+\gamma v(s_1)$
s_2	$-1+\gamma v(s_2)$	$-1+\gamma v(s_2)$	$1+\gamma v(s_4)$	$0+\gamma v(s_1)$	$-1+\gamma v(s_2)$
s_3	$0+\gamma v(s_1)$	$1+\gamma v(s_4)$	$-1+\gamma v(s_3)$	$-1+\gamma v(s_3)$	$0+\gamma v(s_3)$
s_4	$-1+\gamma v(s_2)$	$-1+\gamma v(s_4)$	$-1+\gamma v(s_4)$	$0+\gamma v(s_3)$	$1+\gamma v(s_4)$

Table 4.1: The expression of $q(s,a)$ for the example as shown in Figure 4.2.

ure 4.2). The target area is s_4. The reward settings are $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

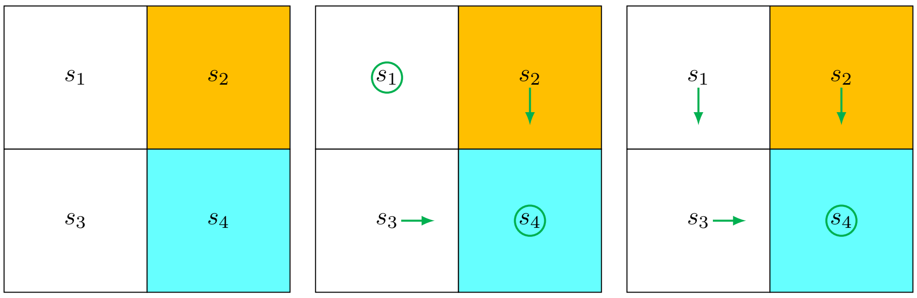


Figure 4.2: An example for demonstrating the implementation of the value iteration algorithm.

The expression of the q-value for each state-action pair is shown in Table 4.1.

◇ $k = 0$:

Without loss of generality, select the initial values as $v_0(s_1) = v_0(s_2) = v_0(s_3) = v_0(s_4) = 0$.

q-value calculation: Substituting $v_0(s_i)$ into Table 4.1 gives the q-values shown in Table 4.2.

q-table	a_1	a_2	a_3	a_4	a_5
s_1	-1	-1	0	-1	0
s_2	-1	-1	1	0	-1
s_3	0	1	-1	-1	0
s_4	-1	-1	-1	0	1

Table 4.2: The value of $q(s,a)$ at $k=0$.

q-table	a_1	a_2	a_3	a_4	a_5
s_1	$-1+\gamma 0$	$-1+\gamma 1$	$0+\gamma 1$	$-1+\gamma 0$	$0+\gamma 0$
s_2	$-1+\gamma 1$	$-1+\gamma 1$	$1+\gamma 1$	$0+\gamma 0$	$-1+\gamma 1$
s_3	$0+\gamma 0$	$1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$
s_4	$-1+\gamma 1$	$-1+\gamma 1$	$-1+\gamma 1$	$0+\gamma 1$	$1+\gamma 1$

Table 4.3: The value of $q(s,a)$ at $k=1$.

Policy update: π_1 is obtained by selecting the actions with the greatest q-values for every state:

$$\pi_1(a_5|s_1)=1, \quad \pi_1(a_3|s_2)=1, \quad \pi_1(a_2|s_3)=1, \quad \pi_1(a_5|s_4)=1.$$

This policy is visualized in Figure 4.2 (the middle subfigure). It is clear that this policy is not optimal because it selects to stay unchanged at s_1. Notably, the q-values for (s_1,a_5) and (s_1,a_3) are actually the same, and we can randomly select either action.

Value update: v_1 is obtained by updating the v-value to the greatest q-value for each state:

$$v_1(s_1)=0, \quad v_1(s_2)=1, \quad v_1(s_3)=1, \quad v_1(s_4)=1.$$

◇ $k=1$:

q-value calculation: Substituting $v_1(s_i)$ into Table 4.1 yields the q-values shown in Table 4.3.

Policy update: π_2 is obtained by selecting the greatest q-values:

$$\pi_2(a_3|s_1)=1, \quad \pi_2(a_3|s_2)=1, \quad \pi_2(a_2|s_3)=1, \quad \pi_2(a_5|s_4)=1.$$

This policy is visualized in Figure 4.2 (the right subfigure).

Value update: v_2 is obtained by updating the v-value to the greatest q-value for each state:

$$v_2(s_1)=\gamma 1, \quad v_2(s_2)=1+\gamma 1, \quad v_2(s_3)=1+\gamma 1, \quad v_2(s_4)=1+\gamma 1.$$

◇ $k=2,3,4,\dots$

It is notable that policy π_2, as illustrated in Figure 4.2(c), is already optimal. Therefore, we only need to run two iterations to obtain an optimal policy in this simple example. For more complex examples, we need to run more iterations until the value of v_k converges (e.g., until $\|v_{k+1} - v_k\|$ is smaller than a pre-specified threshold).

4.2 Policy iteration

This section presents another important algorithm: *policy iteration*. Unlike value iteration, policy iteration is not for directly solving the Bellman optimality equation. However, it has an intimate relationship with value iteration, as shown later. Moreover, the idea of policy iteration is very important since it is widely utilized in reinforcement learning algorithms.

4.2.1 Algorithm analysis

Policy iteration is an iterative algorithm. Each iteration has two steps.

- ◇ The first is a *policy evaluation* step. As its name suggests, this step evaluates a given policy by calculating the corresponding state value. That is to solve the following Bellman equation:

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}, \tag{4.3}$$

  where π_k is the policy obtained in the last iteration and v_{π_k} is the state value to be calculated. The values of r_{π_k} and P_{π_k} can be obtained from the system model.

- ◇ The second is a *policy improvement* step. As its name suggests, this step is used to improve the policy. In particular, once v_{π_k} has been calculated in the first step, a new policy π_{k+1} can be obtained as

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k}).$$

Three questions naturally follow the above description of the algorithm.

- ◇ In the policy evaluation step, how to solve the state value v_{π_k}?
- ◇ In the policy improvement step, why is the new policy π_{k+1} better than π_k?
- ◇ Why can this algorithm finally converge to an optimal policy?

We next answer these questions one by one.

In the policy evaluation step, how to calculate v_{π_k}?

We introduced two methods in Chapter 2 for solving the Bellman equation in (4.3). We next revisit the two methods briefly. The first method is a closed-form solution:

$v_{\pi_k} = (I - \gamma P_{\pi_k})^{-1} r_{\pi_k}$. This closed-form solution is useful for theoretical analysis purposes, but it is inefficient to implement since it requires other numerical algorithms to compute the matrix inverse. The second method is an iterative algorithm that can be easily implemented:

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, ... \tag{4.4}$$

where $v_{\pi_k}^{(j)}$ denotes the jth estimate of v_{π_k}. Starting from any initial guess $v_{\pi_k}^{(0)}$, it is ensured that $v_{\pi_k}^{(j)} \to v_{\pi_k}$ as $j \to \infty$. Details can be found in Section 2.7.

Interestingly, policy iteration is an iterative algorithm with another iterative algorithm (4.4) embedded in the policy evaluation step. In theory, this embedded iterative algorithm requires an *infinite* number of steps (that is, $j \to \infty$) to converge to the true state value v_{π_k}. This is, however, impossible to realize. In practice, the iterative process terminates when a certain criterion is satisfied. For example, the termination criterion can be that $\|v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)}\|$ is less than a prespecified threshold or that j exceeds a prespecified value. If we do not run an infinite number of iterations, we can only obtain an imprecise value of v_{π_k}, which will be used in the subsequent policy improvement step. Would this cause problems? The answer is no. The reason will become clear when we introduce the truncated policy iteration algorithm later in Section 4.3.

In the policy improvement step, why is π_{k+1} better than π_k?

The policy improvement step can improve the given policy, as shown below.

Lemma 4.1 (Policy improvement). *If $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, then $v_{\pi_{k+1}} \geq v_{\pi_k}$.*

Here, $v_{\pi_{k+1}} \geq v_{\pi_k}$ means that $v_{\pi_{k+1}}(s) \geq v_{\pi_k}(s)$ for all s. The proof of this lemma is given in Box 4.1.

Box 4.1: Proof of Lemma 4.1

Since $v_{\pi_{k+1}}$ and v_{π_k} are state values, they satisfy the Bellman equations:

$$\begin{aligned} v_{\pi_{k+1}} &= r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}, \\ v_{\pi_k} &= r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}. \end{aligned}$$

Since $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$, we know that

$$r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k} \geq r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

It then follows that

$$\begin{aligned} v_{\pi_k} - v_{\pi_{k+1}} &= (r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}) - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) \\ &\leq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) \\ &\leq \gamma P_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}). \end{aligned}$$

Therefore,

$$\begin{aligned} v_{\pi_k} - v_{\pi_{k+1}} \leq \gamma^2 P^2_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) \leq \ldots &\leq \gamma^n P^n_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) \\ &\leq \lim_{n\to\infty} \gamma^n P^n_{\pi_{k+1}} (v_{\pi_k} - v_{\pi_{k+1}}) = 0. \end{aligned}$$

The limit is due to the facts that $\gamma^n \to 0$ as $n \to \infty$ and $P^n_{\pi_{k+1}}$ is a nonnegative stochastic matrix for any n. Here, a stochastic matrix refers to a nonnegative matrix whose row sums are equal to one for all rows.

Why can the policy iteration algorithm eventually find an optimal policy?

The policy iteration algorithm generates two sequences. The first is a sequence of policies: $\{\pi_0, \pi_1, \ldots, \pi_k, \ldots\}$. The second is a sequence of state values: $\{v_{\pi_0}, v_{\pi_1}, \ldots, v_{\pi_k}, \ldots\}$. Suppose that v^* is the optimal state value. Then, $v_{\pi_k} \leq v^*$ for all k. Since the policies are continuously improved according to Lemma 4.1, we know that

$$v_{\pi_0} \leq v_{\pi_1} \leq v_{\pi_2} \leq \cdots \leq v_{\pi_k} \leq \cdots \leq v^*.$$

Since v_{π_k} is nondecreasing and always bounded from above by v^*, it follows from the monotone convergence theorem [12] (Appendix C) that v_{π_k} converges to a constant value, denoted as v_∞, when $k \to \infty$. The following analysis shows that $v_\infty = v^*$.

Theorem 4.1 (Convergence of policy iteration). *The state value sequence $\{v_{\pi_k}\}_{k=0}^\infty$ generated by the policy iteration algorithm converges to the optimal state value v^*. As a result, the policy sequence $\{\pi_k\}_{k=0}^\infty$ converges to an optimal policy.*

The proof of this theorem is given in Box 4.2. The proof not only shows the convergence of the policy iteration algorithm but also reveals the relationship between the policy iteration and value iteration algorithms. Loosely speaking, if both algorithms start from the same initial guess, policy iteration will converge faster than value iteration due to the additional iterations embedded in the policy evaluation step. This point will become clearer when we introduce the truncated policy iteration algorithm in Section 4.3.

Box 4.2: Proof of Theorem 4.1

The idea of the proof is to show that the policy iteration algorithm converges faster than the value iteration algorithm.

In particular, to prove the convergence of $\{v_{\pi_k}\}_{k=0}^{\infty}$, we introduce another sequence $\{v_k\}_{k=0}^{\infty}$ generated by

$$v_{k+1} = f(v_k) = \max_{\pi}(r_\pi + \gamma P_\pi v_k).$$

This iterative algorithm is exactly the value iteration algorithm. We already know that v_k converges to v^* when given any initial value v_0.

For $k = 0$, we can always find a v_0 such that $v_{\pi_0} \geq v_0$ for any π_0.

We next show that $v_k \leq v_{\pi_k} \leq v^*$ for all k by induction.

For $k \geq 0$, suppose that $v_{\pi_k} \geq v_k$.

For $k+1$, we have

$$\begin{aligned}
v_{\pi_{k+1}} - v_{k+1} &= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_{k+1}}) - \max_{\pi}(r_\pi + \gamma P_\pi v_k) \\
&\geq (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - \max_{\pi}(r_\pi + \gamma P_\pi v_k) \\
&\qquad \big(\text{because } v_{\pi_{k+1}} \geq v_{\pi_k} \text{ by Lemma 4.1 and } P_{\pi_{k+1}} \geq 0\big) \\
&= (r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\
&\qquad \big(\text{suppose } \pi'_k = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k)\big) \\
&\geq (r_{\pi'_k} + \gamma P_{\pi'_k} v_{\pi_k}) - (r_{\pi'_k} + \gamma P_{\pi'_k} v_k) \\
&\qquad \big(\text{because } \pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k})\big) \\
&= \gamma P_{\pi'_k}(v_{\pi_k} - v_k).
\end{aligned}$$

Since $v_{\pi_k} - v_k \geq 0$ and $P_{\pi'_k}$ is nonnegative, we have $P_{\pi'_k}(v_{\pi_k} - v_k) \geq 0$ and hence $v_{\pi_{k+1}} - v_{k+1} \geq 0$.

Therefore, we can show by induction that $v_k \leq v_{\pi_k} \leq v^*$ for any $k \geq 0$. Since v_k converges to v^*, v_{π_k} also converges to v^*.

4.2.2 Elementwise form and implementation

To implement the policy iteration algorithm, we need to study its elementwise form.

◇ First, the policy evaluation step solves v_{π_k} from $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$ by using the

Algorithm 4.2: Policy iteration algorithm

```
Initialization: The system model, $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$, is known. Initial guess $\pi_0$.
Goal: Search for the optimal state value and an optimal policy.

While $v_{\pi_k}$ has not converged, for the $k$th iteration, do
    Policy evaluation:
    Initialization: an arbitrary initial guess $v_{\pi_k}^{(0)}$
    While $v_{\pi_k}^{(j)}$ has not converged, for the $j$th iteration, do
        For every state $s \in \mathcal{S}$, do
            $v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s')\right]$
    Policy improvement:
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}$, do
            $q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s')$
        $a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$
        $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
```

iterative algorithm in (4.4). The elementwise form of this algorithm is

$$v_{\pi_k}^{(j+1)}(s) = \sum_a \pi_k(a|s) \left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}^{(j)}(s') \right), \quad s \in \mathcal{S},$$

where $j = 0, 1, 2, \ldots$.

◇ Second, the policy improvement step solves $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$. The elementwise form of this equation is

$$\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) \underbrace{\left(\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_{\pi_k}(s') \right)}_{q_{\pi_k}(s,a)}, \quad s \in \mathcal{S},$$

where $q_{\pi_k}(s,a)$ is the action value under policy π_k. Let $a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$. Then, the greedy optimal policy is

$$\pi_{k+1}(a|s) = \begin{cases} 1, & a = a_k^*(s), \\ 0, & a \neq a_k^*(s). \end{cases}$$

The implementation details are summarized in Algorithm 4.2.

4.2.3 Illustrative examples

A simple example

Consider a simple example shown in Figure 4.3. There are two states with three possible actions: $\mathcal{A} = \{a_\ell, a_0, a_r\}$. The three actions represent moving leftward, staying unchanged, and moving rightward. The reward settings are $r_{\text{boundary}} = -1$ and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

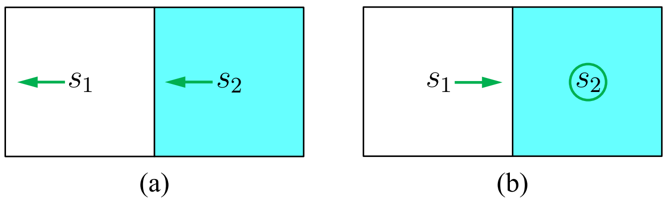


Figure 4.3: An example for illustrating the implementation of the policy iteration algorithm.

We next present the implementation of the policy iteration algorithm in a step-by-step manner. When $k = 0$, we start with the initial policy shown in Figure 4.3(a). This policy is not good because it does not move toward the target area. We next show how to apply the policy iteration algorithm to obtain an optimal policy.

◇ First, in the policy evaluation step, we need to solve the Bellman equation:

$$\begin{aligned} v_{\pi_0}(s_1) &= -1 + \gamma v_{\pi_0}(s_1), \\ v_{\pi_0}(s_2) &= 0 + \gamma v_{\pi_0}(s_1). \end{aligned}$$

Since the equation is simple, it can be manually solved that

$$v_{\pi_0}(s_1) = -10, \quad v_{\pi_0}(s_2) = -9.$$

In practice, the equation can be solved by the iterative algorithm in (4.4). For example, select the initial state values as $v_{\pi_0}^{(0)}(s_1) = v_{\pi_0}^{(0)}(s_2) = 0$. It follows from (4.3) that

$$\begin{cases} v_{\pi_0}^{(1)}(s_1) = -1 + \gamma v_{\pi_0}^{(0)}(s_1) = -1, \\ v_{\pi_0}^{(1)}(s_2) = 0 + \gamma v_{\pi_0}^{(0)}(s_1) = 0, \end{cases}$$
$$\begin{cases} v_{\pi_0}^{(2)}(s_1) = -1 + \gamma v_{\pi_0}^{(1)}(s_1) = -1.9, \\ v_{\pi_0}^{(2)}(s_2) = 0 + \gamma v_{\pi_0}^{(1)}(s_1) = -0.9, \end{cases}$$
$$\begin{cases} v_{\pi_0}^{(3)}(s_1) = -1 + \gamma v_{\pi_0}^{(2)}(s_1) = -2.71, \\ v_{\pi_0}^{(3)}(s_2) = 0 + \gamma v_{\pi_0}^{(2)}(s_1) = -1.71, \end{cases}$$
$$\vdots$$

With more iterations, we can see the trend: $v_{\pi_0}^{(j)}(s_1) \to v_{\pi_0}(s_1) = -10$ and $v_{\pi_0}^{(j)}(s_2) \to v_{\pi_0}(s_2) = -9$ as j increases.

- ◇ Second, in the policy improvement step, the key is to calculate $q_{\pi_0}(s,a)$ for each state-action pair. The following q-table can be used to demonstrate such a process:

$q_{\pi_k}(s,a)$	a_ℓ	a_0	a_r
s_1	$-1+\gamma v_{\pi_k}(s_1)$	$0+\gamma v_{\pi_k}(s_1)$	$1+\gamma v_{\pi_k}(s_2)$
s_2	$0+\gamma v_{\pi_k}(s_1)$	$1+\gamma v_{\pi_k}(s_2)$	$-1+\gamma v_{\pi_k}(s_2)$

Table 4.4: The expression of $q_{\pi_k}(s,a)$ for the example in Figure 4.3.

Substituting $v_{\pi_0}(s_1)=-10, v_{\pi_0}(s_2)=-9$ obtained in the previous policy evaluation step into Table 4.4 yields Table 4.5.

$q_{\pi_0}(s,a)$	a_ℓ	a_0	a_r
s_1	-10	-9	-7.1
s_2	-9	-7.1	-9.1

Table 4.5: The value of $q_{\pi_k}(s,a)$ when $k=0$.

By seeking the greatest value of q_{π_0}, the improved policy π_1 can be obtained as

$$\pi_1(a_r|s_1)=1, \quad \pi_1(a_0|s_2)=1.$$

This policy is illustrated in Figure 4.3(b). It is clear that this policy is optimal.

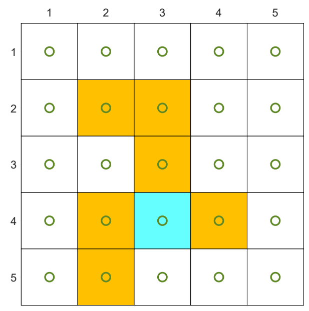
The above process shows that a single iteration is sufficient for finding the optimal policy in this simple example. More iterations are required for more complex examples.

A more complicated example

We next demonstrate the policy iteration algorithm using a more complicated example shown in Figure 4.4. The reward settings are $r_{\text{boundary}}=-1$, $r_{\text{forbidden}}=-10$, and $r_{\text{target}}=1$. The discount rate is $\gamma=0.9$. The policy iteration algorithm can converge to the optimal policy (Figure 4.4(h)) when starting from a random initial policy (Figure 4.4(a)).

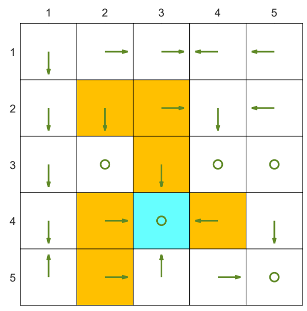
Two interesting phenomena are observed during the iteration process.

- ◇ First, if we observe how the policy evolves, an interesting pattern is that the states that are close to the target area find the optimal policies earlier than those far away. Only if the close states can find trajectories to the target first, can the farther states find trajectories passing through the close states to reach the target.
- ◇ Second, the spatial distribution of the state values exhibits an interesting pattern: the states that are located closer to the target have greater state values. The reason for this pattern is that an agent starting from a farther state must travel for many steps to obtain a positive reward. Such rewards would be severely discounted and hence relatively small.



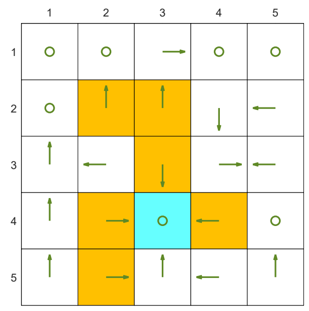
	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	-100.0	-100.0	0.0	0.0
3	0.0	0.0	-100.0	0.0	0.0
4	0.0	-100.0	10.0	-100.0	0.0
5	0.0	-100.0	0.0	0.0	0.0

(a) π_0 and v_{π_0}



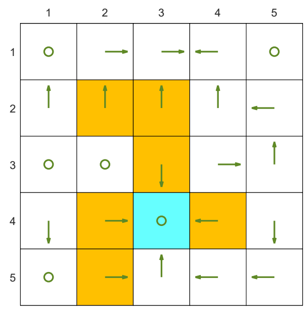
	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	0.0
5	0.0	9.0	10.0	0.0	0.0

(b) π_1 and v_{π_1}



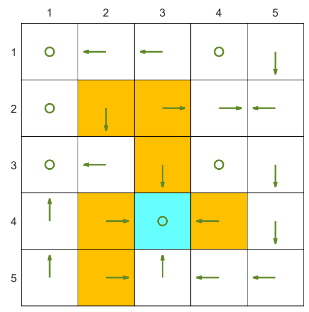
	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	0.0
5	0.0	9.0	10.0	9.0	0.0

(c) π_2 and v_{π_2}



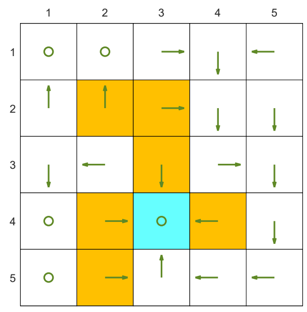
	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	0.0
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

(d) π_3 and v_{π_3}



	1	2	3	4	5
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	10.0	0.0	6.6
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

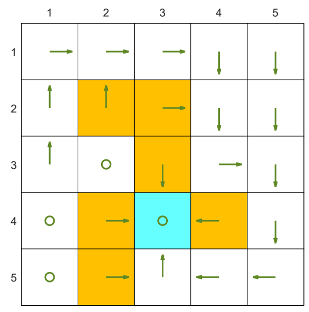
(e) π_4 and v_{π_4}



	1	2	3	4	5
1	0.0	0.0	4.3	4.8	4.3
2	0.0	0.0	4.8	5.3	5.9
3	0.0	0.0	10.0	5.9	6.6
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

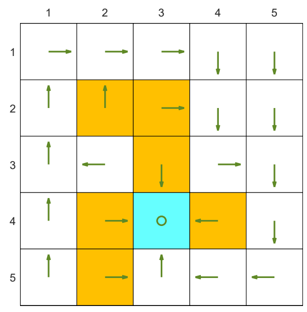
(f) π_5 and v_{π_5}

$\vdots$ $\vdots$



	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	0.0	10.0	5.9	6.6
4	0.0	10.0	10.0	10.0	7.3
5	0.0	9.0	10.0	9.0	8.1

(g) π_9 and v_{π_9}



	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

(h) π_{10} and $v_{\pi_{10}}$

Figure 4.4: The evolution processes of the policies generated by the policy iteration algorithm.

4.3 Truncated policy iteration

We next introduce a more general algorithm called *truncated policy iteration*. We will see that the value iteration and policy iteration algorithms are two special cases of the truncated policy iteration algorithm.

4.3.1 Comparing value iteration and policy iteration

First of all, we compare the value iteration and policy iteration algorithms by listing their steps as follows.

◇ Policy iteration: Select an arbitrary initial policy π_0. In the kth iteration, do the following two steps.

  - Step 1: Policy evaluation (PE). Given π_k, solve v_{π_k} from

$$v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}.$$

  - Step 2: Policy improvement (PI). Given v_{π_k}, solve π_{k+1} from

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k}).$$

◇ Value iteration: Select an arbitrary initial value v_0. In the kth iteration, do the following two steps.

  - Step 1: Policy update (PU). Given v_k, solve π_{k+1} from

$$\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_k).$$

  - Step 2: Value update (VU). Given π_{k+1}, solve v_{k+1} from

$$v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k.$$

The above steps of the two algorithms can be illustrated as

$$\begin{aligned}
&\text{Policy iteration: } \pi_0 \xrightarrow{PE} v_{\pi_0} \xrightarrow{PI} \pi_1 \xrightarrow{PE} v_{\pi_1} \xrightarrow{PI} \pi_2 \xrightarrow{PE} v_{\pi_2} \xrightarrow{PI} \dots \\
&\text{Value iteration: } \qquad v_0 \xrightarrow{PU} \pi_1' \xrightarrow{VU} v_1 \xrightarrow{PU} \pi_2' \xrightarrow{VU} v_2 \xrightarrow{PU} \dots
\end{aligned}$$

It can be seen that the procedures of the two algorithms are very similar.

We examine their value steps more closely to see the difference between the two algorithms. In particular, let both algorithms start from the *same initial condition*: $v_0 = v_{\pi_0}$. The procedures of the two algorithms are listed in Table 4.6. In the first three steps, the two algorithms generate the same results since $v_0 = v_{\pi_0}$. They become

	Policy iteration algorithm	Value iteration algorithm	Comments
1) Policy:	π_0	N/A	
2) Value:	$v_{\pi_0} = r_{\pi_0} + \gamma P_{\pi_0} v_{\pi_0}$	$v_0 \doteq v_{\pi_0}$	
3) Policy:	$\pi_1 = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_0})$	$\pi_1 = \arg\max_\pi (r_\pi + \gamma P_\pi v_0)$	The two policies are the same
4) Value:	$v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$	$v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$	$v_{\pi_1} \geq v_1$ since $v_{\pi_1} \geq v_{\pi_0}$
5) Policy:	$\pi_2 = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_1})$	$\pi_2' = \arg\max_\pi (r_\pi + \gamma P_\pi v_1)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 4.6: A comparison between the implementation steps of policy iteration and value iteration.

different in the fourth step. During the fourth step, the value iteration algorithm executes $v_1 = r_{\pi_1} + \gamma P_{\pi_1} v_0$, which is a one-step calculation, whereas the policy iteration algorithm solves $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$, which requires an infinite number of iterations. If we explicitly write out the iterative process for solving $v_{\pi_1} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}$ in the fourth step, everything becomes clear. By letting $v_{\pi_1}^{(0)} = v_0$, we have

$$\begin{aligned}
& v_{\pi_1}^{(0)} = v_0 \\
\text{value iteration} \leftarrow v_1 \longleftarrow \quad & v_{\pi_1}^{(1)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(0)} \\
& v_{\pi_1}^{(2)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(1)} \\
& \vdots \\
\text{truncated policy iteration} \leftarrow \bar{v}_1 \longleftarrow \quad & v_{\pi_1}^{(j)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(j-1)} \\
& \vdots \\
\text{policy iteration} \leftarrow v_{\pi_1} \longleftarrow \quad & v_{\pi_1}^{(\infty)} = r_{\pi_1} + \gamma P_{\pi_1} v_{\pi_1}^{(\infty)}
\end{aligned}$$

The following observations can be obtained from the above process.

◇ If the iteration is run *only once*, then $v_{\pi_1}^{(1)}$ is actually v_1, as calculated in the value iteration algorithm.

◇ If the iteration is run *an infinite number of times*, then $v_{\pi_1}^{(\infty)}$ is actually v_{π_1}, as calculated in the policy iteration algorithm.

◇ If the iteration is run *a finite number of times* (denoted as j_{truncate}), then such an algorithm is called *truncated policy iteration.* It is called *truncated* because the remaining iterations from j_{truncate} to ∞ are truncated.

As a result, the value iteration and policy iteration algorithms can be viewed as two extreme cases of the truncated policy iteration algorithm: value iteration terminates

Algorithm 4.3: Truncated policy iteration algorithm

```
Initialization: The probability models $p(r|s,a)$ and $p(s'|s,a)$ for all $(s,a)$ are known. Initial guess $\pi_0$.
Goal: Search for the optimal state value and an optimal policy.
While $v_k$ has not converged, for the $k$th iteration, do
    Policy evaluation:
    Initialization: select the initial guess as $v_k^{(0)} = v_{k-1}$. The maximum number of iterations is set as $j_{\text{truncate}}$.
    While $j < j_{\text{truncate}}$, do
        For every state $s \in \mathcal{S}$, do
            $v_k^{(j+1)}(s) = \sum_a \pi_k(a|s) \left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_k^{(j)}(s')\right]$
    Set $v_k = v_k^{(j_{\text{truncate}})}$
    Policy improvement:
    For every state $s \in \mathcal{S}$, do
        For every action $a \in \mathcal{A}(s)$, do
            $q_k(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a) v_k(s')$
        $a_k^*(s) = \arg\max_a q_k(s,a)$
        $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise
```

at $j_{\text{truncate}} = 1$, and policy iteration terminates at $j_{\text{truncate}} = \infty$. It should be noted that, although the above comparison is illustrative, it is based on the condition that $v_{\pi_1}^{(0)} = v_0 = v_{\pi_0}$. The two algorithms cannot be directly compared without this condition.

4.3.2 Truncated policy iteration algorithm

In a nutshell, the truncated policy iteration algorithm is the same as the policy iteration algorithm except that it merely runs a finite number of iterations in the policy evaluation step. Its implementation details are summarized in Algorithm 4.3. It is notable that v_k and $v_k^{(j)}$ in the algorithm are not state values. Instead, they are approximations of the true state values because only a finite number of iterations are executed in the policy evaluation step.

If v_k does not equal v_{π_k}, will the algorithm still be able to find optimal policies? The answer is yes. Intuitively, truncated policy iteration is in between value iteration and policy iteration. On the one hand, it converges faster than the value iteration algorithm because it computes more than one iteration during the policy evaluation step. On the other hand, it converges slower than the policy iteration algorithm because it only computes a finite number of iterations. This intuition is illustrated in Figure 4.5. Such intuition is also supported by the following analysis.

Proposition 4.1 (Value improvement). *Consider the iterative algorithm in the policy*

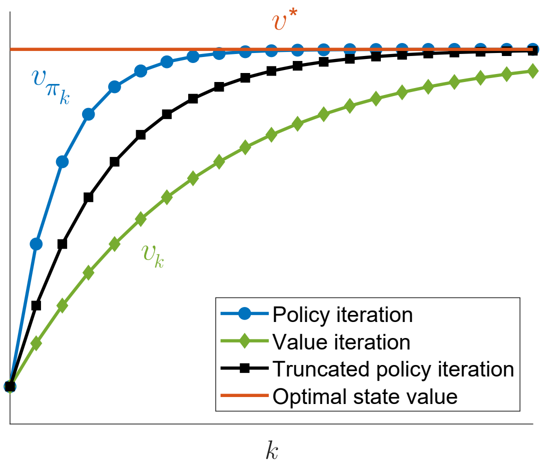


Figure 4.5: An illustration of the relationships between the value iteration, policy iteration, and truncated policy iteration algorithms.

evaluation step:

$$v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \ldots$$

If the initial guess is selected as $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$*, it holds that*

$$v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$$

for $j = 0, 1, 2, \ldots$.

Box 4.3: Proof of Proposition 4.1

First, since $v_{\pi_k}^{(j)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j-1)}$ and $v_{\pi_k}^{(j+1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(j)}$, we have

$$v_{\pi_k}^{(j+1)} - v_{\pi_k}^{(j)} = \gamma P_{\pi_k} (v_{\pi_k}^{(j)} - v_{\pi_k}^{(j-1)}) = \cdots = \gamma^j P_{\pi_k}^j (v_{\pi_k}^{(1)} - v_{\pi_k}^{(0)}). \tag{4.5}$$

Second, since $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$, we have

$$v_{\pi_k}^{(1)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}^{(0)} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_{k-1}} \geq r_{\pi_{k-1}} + \gamma P_{\pi_{k-1}} v_{\pi_{k-1}} = v_{\pi_{k-1}} = v_{\pi_k}^{(0)},$$

where the inequality is due to $\pi_k = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_{k-1}})$. Substituting $v_{\pi_k}^{(1)} \geq v_{\pi_k}^{(0)}$ into (4.5) yields $v_{\pi_k}^{(j+1)} \geq v_{\pi_k}^{(j)}$.

Notably, Proposition 4.1 requires the assumption that $v_{\pi_k}^{(0)} = v_{\pi_{k-1}}$. However, $v_{\pi_{k-1}}$ is unavailable in practice, and only v_{k-1} is available. Nevertheless, Proposition 4.1 still sheds light on the convergence of the truncated policy iteration algorithm. A more in-depth discussion of this topic can be found in [2, Section 6.5].

Up to now, the advantages of truncated policy iteration are clear. Compared to the policy iteration algorithm, the truncated one merely requires a finite number of iterations in the policy evaluation step and hence is more computationally efficient. Compared to value iteration, the truncated policy iteration algorithm can speed up its convergence rate by running for a few more iterations in the policy evaluation step.

4.4 Summary

This chapter introduced three algorithms that can be used to find optimal policies.

◇ Value iteration: The value iteration algorithm is the same as the algorithm suggested by the contraction mapping theorem for solving the Bellman optimality equation. It can be decomposed into two steps: value update and policy update.

◇ Policy iteration: The policy iteration algorithm is slightly more complicated than the value iteration algorithm. It also contains two steps: policy evaluation and policy improvement.

◇ Truncated policy iteration: The value iteration and policy iteration algorithms can be viewed as two extreme cases of the truncated policy iteration algorithm.

A common property of the three algorithms is that every iteration has two steps. One step is to update the value, and the other step is to update the policy. The idea of interaction between value and policy updates widely exists in reinforcement learning algorithms. This idea is also called *generalized policy iteration* [3].

Finally, the algorithms introduced in this chapter require the system model. Starting in Chapter 5, we will study model-free reinforcement learning algorithms. We will see that the model-free can be obtained by extending the algorithms introduced in this chapter.

4.5 Q&A

◇ Q: Is the value iteration algorithm guaranteed to find optimal policies?

A: Yes. This is because value iteration is exactly the algorithm suggested by the contraction mapping theorem for solving the Bellman optimality equation in the last chapter. The convergence of this algorithm is guaranteed by the contraction mapping theorem.

◇ Q: Are the intermediate values generated by the value iteration algorithm state values?

A: No. These values are not guaranteed to satisfy the Bellman equation of any policy.

◇ Q: What steps are included in the policy iteration algorithm?

A: Each iteration of the policy iteration algorithm contains two steps: policy evaluation and policy improvement. In the policy evaluation step, the algorithm aims to solve the Bellman equation to obtain the state value of the current policy. In the policy improvement step, the algorithm aims to update the policy so that the newly generated policy has greater state values.

◇ Q: Is another iterative algorithm embedded in the policy iteration algorithm?

A: Yes. In the policy evaluation step of the policy iteration algorithm, an iterative algorithm is required to solve the Bellman equation of the current policy.

◇ Q: Are the intermediate values generated by the policy iteration algorithm state values?

A: Yes. This is because these values are the solutions of the Bellman equation of the current policy.

◇ Q: Is the policy iteration algorithm guaranteed to find optimal policies?

A: Yes. We have presented a rigorous proof of its convergence in this chapter.

◇ Q: What is the relationship between the truncated policy iteration and policy iteration algorithms?

A: As its name suggests, the truncated policy iteration algorithm can be obtained from the policy iteration algorithm by simply executing a finite number of iterations during the policy evaluation step.

◇ Q: What is the relationship between truncated policy iteration and value iteration?

A: Value iteration can be viewed as an extreme case of truncated policy iteration, where a single iteration is run during the policy evaluation step.

◇ Q: Are the intermediate values generated by the truncated policy iteration algorithm state values?

A: No. Only if we run an infinite number of iterations in the policy evaluation step, can we obtain true state values. If we run a finite number of iterations, we can only obtain approximates of the true state values.

◇ Q: How many iterations should we run in the policy evaluation step of the truncated policy iteration algorithm?

A: The general guideline is to run a few iterations but not too many. The use of a few iterations in the policy evaluation step can speed up the overall convergence rate, but running too many iterations would not significantly speed up the convergence rate.

◇ Q: What is generalized policy iteration?

A: Generalized policy iteration is not a specific algorithm. Instead, it refers to the general idea of the interaction between value and policy updates. This idea is rooted in the policy iteration algorithm. Most of the reinforcement learning algorithms introduced in this book fall into the scope of generalized policy iteration.

◇ Q: What are model-based and model-free reinforcement learning?

A: Although the algorithms introduced in this chapter can find optimal policies, they are usually called dynamic programming algorithms rather than reinforcement learning algorithms because they require the system model. Reinforcement learning algorithms can be classified into two categories: model-based and model-free. Here, “model-based” does not refer to the requirement of the system model. Instead, model-based reinforcement learning uses data to estimate the system model and uses this model during the learning process. By contrast, model-free reinforcement learning does not involve model estimation during the learning process. All the reinforcement learning algorithms introduced in this book are model-free algorithms. More information about model-based reinforcement learning can be found in [13–16].