Chapter 5

Monte Carlo Methods

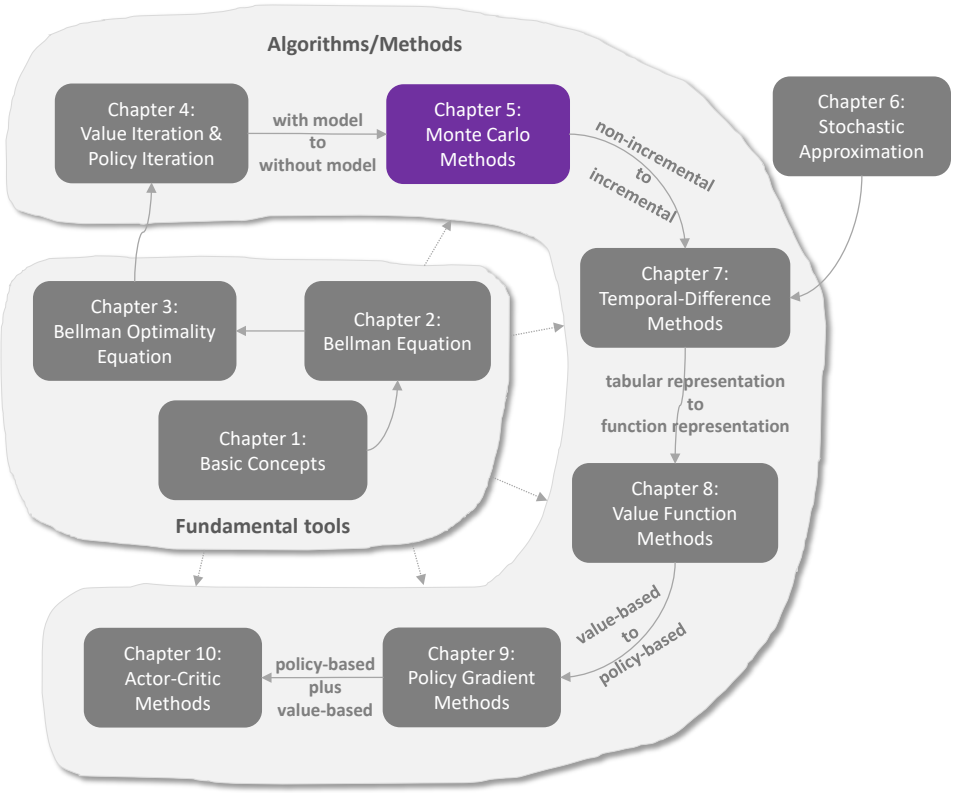


Figure 5.1: Where we are in this book.

In the previous chapter, we introduced algorithms that can find optimal policies based on the system model. In this chapter, we start introducing *model-free* reinforcement learning algorithms that do not presume system models.

While this is the first time we introduce model-free algorithms in this book, we must fill a knowledge gap: how can we find optimal policies without models? The philosophy is simple: If we do not have a model, we must have some data. If we do not have data, we must have a model. If we have neither, then we are not able to find optimal policies. The "data" in reinforcement learning usually refers to the agent's interaction experiences with the environment.

To demonstrate how to learn from data rather than a model, we start this chapter by introducing the *mean estimation* problem, where the expected value of a random variable is estimated from some samples. Understanding this problem is crucial for understanding the fundamental idea of *learning from data*.

Then, we introduce three algorithms based on Monte Carlo (MC) methods. These algorithms can learn optimal policies from experience samples. The first and simplest algorithm is called MC Basic, which can be readily obtained by modifying the policy iteration algorithm introduced in the last chapter. Understanding this algorithm is important for grasping the fundamental idea of MC-based reinforcement learning. By extending this algorithm, we further introduce another two algorithms that are more complicated but more efficient.

5.1 Motivating example: Mean estimation

We next introduce the *mean estimation* problem to demonstrate how to learn from data rather than a model. We will see that mean estimation can be achieved based on *Monte Carlo* methods, which refer to a broad class of techniques that use stochastic samples to solve estimation problems. The reader may wonder why we care about the mean estimation problem. It is simply because state and action values are both defined as the means of returns. Estimating a state or action value is actually a mean estimation problem.

Consider a random variable X that can take values from a finite set of real numbers denoted as $\mathcal{X}$. Suppose that our task is to calculate the mean or expected value of X: $\mathbb{E}[X]$. Two approaches can be used to calculate $\mathbb{E}[X]$.

- The first approach is *model-based*. Here, the model refers to the probability distribution of X. If the model is known, then the mean can be directly calculated based on the definition of the expected value:

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} p(x)x.$$

  In this book, we use the terms *expected value*, *mean*, and *average* interchangeably.

- The second approach is *model-free*. When the probability distribution (i.e., the model) of X is unknown, suppose that we have some samples $\{x_1, x_2, \ldots, x_n\}$ of X. Then, the mean can be approximated as

$$\mathbb{E}[X] \approx \bar{x} = \frac{1}{n} \sum_{j=1}^{n} x_j.$$

  When n is small, this approximation may not be accurate. However, as n increases, the approximation becomes increasingly accurate. When $n \to \infty$, we have $\bar{x} \to \mathbb{E}[X]$.

This is guaranteed by the *law of large numbers*: the average of a large number of samples is close to the expected value. The law of large numbers is introduced in Box 5.1.

The following example illustrates the two approaches described above. Consider a coin flipping game. Let random variable X denote which side is showing when the coin lands. X has two possible values: $X = 1$ when the head is showing, and $X = -1$ when the tail is showing. Suppose that the true probability distribution (i.e., the model) of X is

$$p(X = 1) = 0.5, \quad p(X = -1) = 0.5.$$

If the probability distribution is known in advance, we can directly calculate the mean as

$$\mathbb{E}[X] = 0.5 \cdot 1 + 0.5 \cdot (-1) = 0.$$

If the probability distribution is unknown, then we can flip the coin many times and record the sampling results $\{x_i\}_{i=1}^n$. By calculating the average of the samples, we can obtain an estimate of the mean. As shown in Figure 5.2, the estimated mean becomes increasingly accurate as the number of samples increases.

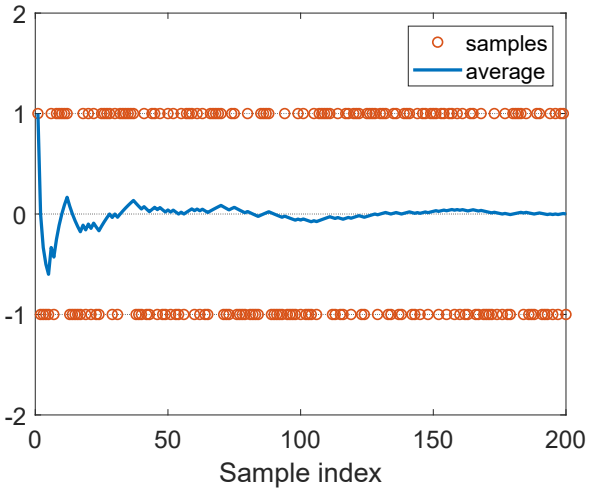


Figure 5.2: An example for demonstrating the law of large numbers. Here, the samples are drawn from $\{+1, -1\}$ following a uniform distribution. The average of the samples gradually converges to zero, which is the true expected value, as the number of samples increases.

It is worth mentioning that the samples used for mean estimation must be *independent and identically distributed* (i.i.d. or iid). Otherwise, if the sampling values correlate, it may be impossible to correctly estimate the expected value. An extreme case is that all the sampling values are the same as the first one, whatever the first one is. In this case, the average of the samples is always equal to the first sample, no matter how many samples we use.

Box 5.1: Law of large numbers

For a random variable X, suppose that $\{x_i\}_{i=1}^n$ are some i.i.d. samples. Let $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$ be the average of the samples. Then,

$$\mathbb{E}[\bar{x}] = \mathbb{E}[X],$$
$$\text{var}[\bar{x}] = \frac{1}{n}\text{var}[X].$$

The above two equations indicate that $\bar{x}$ is an unbiased estimate of $\mathbb{E}[X]$, and its variance decreases to zero as n increases to infinity.

The proof is given below.

First, $\mathbb{E}[\bar{x}] = \mathbb{E}\big[\sum_{i=1}^n x_i/n\big] = \sum_{i=1}^n \mathbb{E}[x_i]/n = \mathbb{E}[X]$, where the last equability is due to the fact that the samples are *identically distributed* (that is, $\mathbb{E}[x_i] = \mathbb{E}[X]$).

Second, $\text{var}(\bar{x}) = \text{var}\big[\sum_{i=1}^n x_i/n\big] = \sum_{i=1}^n \text{var}[x_i]/n^2 = (n \cdot \text{var}[X])/n^2 = \text{var}[X]/n$, where the second equality is due to the fact that the samples are *independent*, and the third equability is a result of the samples being *identically distributed* (that is, $\text{var}[x_i] = \text{var}[X]$).

5.2 MC Basic: The simplest MC-based algorithm

This section introduces the first and the simplest MC-based reinforcement learning algorithm. This algorithm is obtained by replacing the *model-based policy evaluation step* in the policy iteration algorithm introduced in Section 4.2 with a *model-free MC estimation step*.

5.2.1 Converting policy iteration to be model-free

There are two steps in every iteration of the policy iteration algorithm (see Section 4.2). The first step is *policy evaluation*, which aims to compute v_{π_k} by solving $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$. The second step is *policy improvement*, which aims to compute the greedy policy $\pi_{k+1} = \arg\max_\pi (r_\pi + \gamma P_\pi v_{\pi_k})$. The elementwise form of the policy improvement step is

$$\begin{aligned}\pi_{k+1}(s) &= \arg\max_\pi \sum_a \pi(a|s)\left[\sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi_k}(s')\right] \\ &= \arg\max_\pi \sum_a \pi(a|s) q_{\pi_k}(s,a), \quad s \in \mathcal{S}.\end{aligned}$$

It must be noted that the action values lie in the *core* of these two steps. Specifically. in the first step, the state values are calculated for the purpose of calculating the action

values. In the second step, the new policy is generated based on the calculated action values. Let us reconsider how we can calculate the action values. Two approaches are available.

- ◇ The first is a *model-based* approach. This is the approach adopted by the policy iteration algorithm. In particular, we can first calculate the state value v_{π_k} by solving the Bellman equation. Then, we can calculate the action values by using

$$q_{\pi_k}(s,a) = \sum_r p(r|s,a)r + \gamma \sum_{s'} p(s'|s,a)v_{\pi_k}(s'). \tag{5.1}$$

  This approach requires the system model $\{p(r|s,a), p(s'|s,a)\}$ to be known.

- ◇ The second is a *model-free* approach. Recall that the definition of an action value is

$$\begin{aligned} q_{\pi_k}(s,a) &= \mathbb{E}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots |S_t = s, A_t = a], \end{aligned}$$

  which is the expected return obtained when starting from (s,a). Since $q_{\pi_k}(s,a)$ is an expectation, it can be estimated by MC methods as demonstrated in Section 5.1. To do that, starting from (s,a), the agent can interact with the environment by following policy π_k and then obtain a certain number of episodes. Suppose that there are n episodes and that the return of the ith episode is $g^{(i)}_{\pi_k}(s,a)$. Then, $q_{\pi_k}(s,a)$ can be approximated as

$$q_{\pi_k}(s,a) = \mathbb{E}[G_t|S_t = s, A_t = a] \approx \frac{1}{n}\sum_{i=1}^{n} g^{(i)}_{\pi_k}(s,a). \tag{5.2}$$

  We already know that, if the number of episodes n is sufficiently large, the approximation will be sufficiently accurate according to the law of large numbers.

The fundamental idea of MC-based reinforcement learning is to use a model-free method for estimating action values, as shown in (5.2), to replace the model-based method in the policy iteration algorithm.

5.2.2 The MC Basic algorithm

We are now ready to present the first MC-based reinforcement learning algorithm. Starting from an initial policy π_0, the algorithm has two steps in the kth iteration ($k = 0, 1, 2, \dots$).

- ◇ *Step 1: Policy evaluation.* This step is used to estimate $q_{\pi_k}(s,a)$ for all (s,a). Specifically, for every (s,a), we collect sufficiently many episodes and use the average of the returns, denoted as $q_k(s,a)$, to approximate $q_{\pi_k}(s,a)$.

Algorithm 5.1: MC Basic (a model-free variant of policy iteration)

Initialization: Initial guess π_0.
Goal: Search for an optimal policy.

For the kth iteration ($k = 0, 1, 2, \dots$), do
  For every state $s \in \mathcal{S}$, do
    For every action $a \in \mathcal{A}(s)$, do
      Collect sufficiently many episodes starting from (s, a) by following π_k
      Policy evaluation:
      $q_{\pi_k}(s, a) \approx q_k(s, a) =$ the average return of all the episodes starting from (s, a)
    Policy improvement:
    $a_k^*(s) = \arg\max_a q_k(s, a)$
    $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

◇ *Step 2: Policy improvement.* This step solves $\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) q_k(s, a)$ for all $s \in \mathcal{S}$. The greedy optimal policy is $\pi_{k+1}(a_k^*|s) = 1$ where $a_k^* = \arg\max_a q_k(s, a)$.

This is the simplest MC-based reinforcement learning algorithm, which is called *MC Basic* in this book. The pseudocode of the MC Basic algorithm is given in Algorithm 5.1. As can be seen, it is very similar to the policy iteration algorithm. The only difference is that it calculates action values directly from experience samples, whereas policy iteration calculates state values first and then calculates the action values based on the system model. It should be noted that the model-free algorithm directly estimates action values. Otherwise, if it estimates state values instead, we still need to calculate action values from these state values using the system model, as shown in (5.1).

Since policy iteration is convergent, MC Basic is also convergent when given sufficient samples. That is, for every (s, a), suppose that there are sufficiently many episodes starting from (s, a). Then, the average of the returns of these episodes can accurately approximate the action value of (s, a). In practice, we usually do not have sufficient episodes for every (s, a). As a result, the approximation of the action values may not be accurate. Nevertheless, the algorithm usually can still work. This is similar to the truncated policy iteration algorithm, where the action values are neither accurately calculated.

Finally, MC Basic is too simple to be practical due to its low sample efficiency. The reason why we introduce this algorithm is to let readers grasp the core idea of MC-based reinforcement learning. It is important to understand this algorithm well before studying more complex algorithms introduced later in this chapter. We will see that more complex and sample-efficient algorithms can be readily obtained by extending the MC Basic algorithm.

5.2.3 Illustrative examples

A simple example: A step-by-step implementation

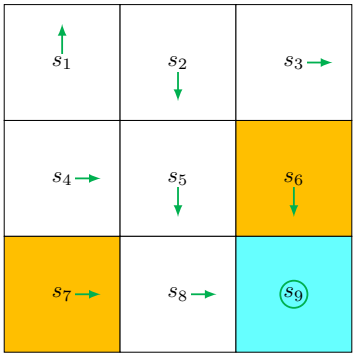


Figure 5.3: An example for illustrating the MC Basic algorithm.

We next use an example to demonstrate the implementation details of the MC Basic algorithm. The reward settings are $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$. The initial policy π_0 is shown in Figure 5.3. This initial policy is not optimal for s_1 or s_3.

While all the action values should be calculated, we merely present those of s_1 due to space limitations. At s_1, there are five possible actions. For each action, we need to collect many episodes that are sufficiently long to effectively approximate the action value. However, since this example is deterministic in terms of both the policy and model, running multiple times would generate the same trajectory. As a result, the estimation of each action value merely requires a single episode.

Following π_0, we can obtain the following episodes by respectively starting from (s_1, a_1), (s_1, a_2), $\ldots$, (s_1, a_5).

- ◇ Starting from (s_1, a_1), the episode is $s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$. The action value equals the discounted return of the episode:

$$q_{\pi_0}(s_1, a_1) = -1 + \gamma(-1) + \gamma^2(-1) + \cdots = \frac{-1}{1-\gamma}.$$

- ◇ Starting from (s_1, a_2), the episode is $s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} \ldots$. The action value equals the discounted return of the episode:

$$q_{\pi_0}(s_1, a_2) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3(1) + \gamma^4(1) + \cdots = \frac{\gamma^3}{1-\gamma}.$$

- ◇ Starting from (s_1, a_3), the episode is $s_1 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_5 \xrightarrow{a_3} \ldots$. The action value equals

the discounted return of the episode:

$$q_{\pi_0}(s_1, a_3) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3(1) + \gamma^4(1) + \cdots = \frac{\gamma^3}{1-\gamma}.$$

◇ Starting from (s_1, a_4), the episode is $s_1 \xrightarrow{a_4} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$. The action value equals the discounted return of the episode:

$$q_{\pi_0}(s_1, a_4) = -1 + \gamma(-1) + \gamma^2(-1) + \cdots = \frac{-1}{1-\gamma}.$$

◇ Starting from (s_1, a_5), the episode is $s_1 \xrightarrow{a_5} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$. The action value equals the discounted return of the episode:

$$q_{\pi_0}(s_1, a_5) = 0 + \gamma(-1) + \gamma^2(-1) + \cdots = \frac{-\gamma}{1-\gamma}.$$

By comparing the five action values, we see that

$$q_{\pi_0}(s_1, a_2) = q_{\pi_0}(s_1, a_3) = \frac{\gamma^3}{1-\gamma} > 0$$

are the maximum values. As a result, the new policy can be obtained as

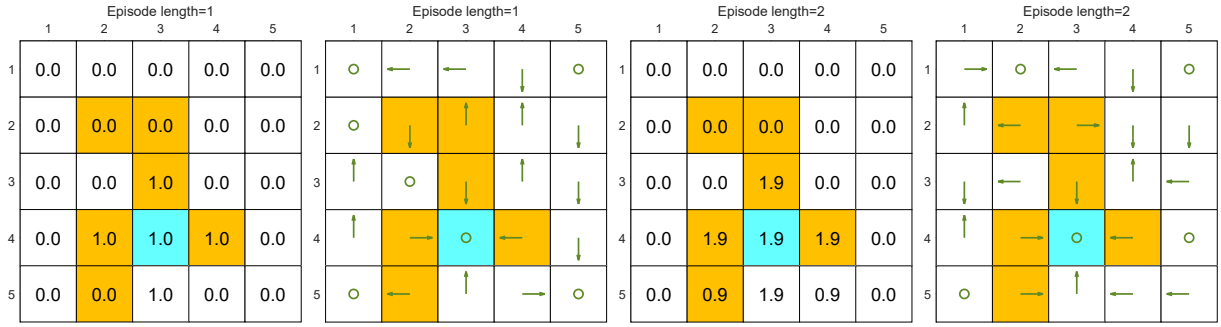
$$\pi_1(a_2|s_1) = 1 \quad \text{or} \quad \pi_1(a_3|s_1) = 1.$$

It is intuitive that the improved policy, which takes either a_2 or a_3 at s_1, is optimal. Therefore, we can successfully obtain an optimal policy by using merely one iteration for this simple example. In this simple example, the initial policy is already optimal for all the states except s_1 and s_3. Therefore, the policy can become optimal after merely a single iteration. When the policy is nonoptimal for other states, more iterations are needed.

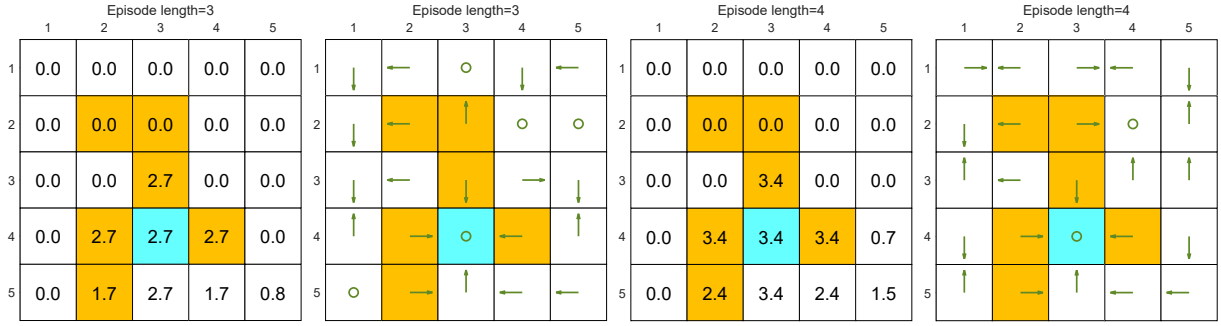
A comprehensive example: Episode length and sparse rewards

We next discuss some interesting properties of the MC Basic algorithm by examining a more comprehensive example. The example is a 5-by-5 grid world (Figure 5.4). The reward settings are $r_{\text{boundary}} = -1$, $r_{\text{forbidden}} = -10$, and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

First, we demonstrate that the *episode length* greatly impacts the final optimal policies. In particular, Figure 5.4 shows the final results generated by the MC Basic algorithm with different episode lengths. When the length of each episode is too short, neither the policy nor the value estimate is optimal (see Figures 5.4(a)-(d)). In the extreme case where the episode length is one, only the states that are adjacent to the target have

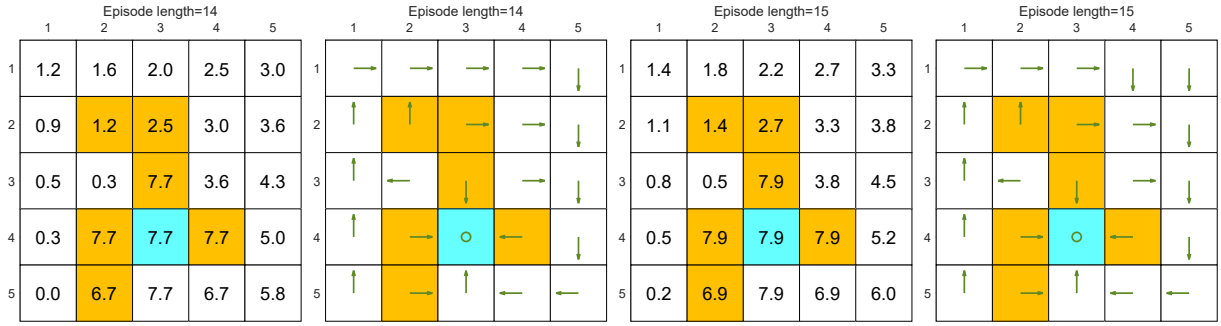


(a) Final value and policy with episode length=1 (b) Final value and policy with episode length=2

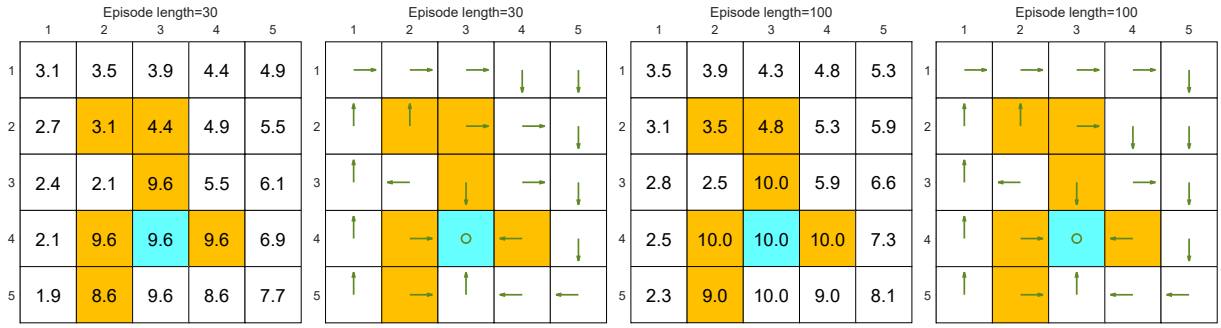


(c) Final value and policy with episode length=3 (d) Final value and policy with episode length=4

⋮ ⋮ ⋮



(e) Final value and policy with episode length=14 (f) Final value and policy with episode length=15



(g) Final value and policy with episode length=30 (h) Final value and policy with episode length=100

Figure 5.4: The policies and state values obtained by the MC Basic algorithm when given different episode lengths. Only if the length of each episode is sufficiently long, can the state values be accurately estimated.

nonzero values, and all the other states have zero values since each episode is too short to reach the target or get positive rewards (see Figure 5.4(a)). As the episode length increases, the policy and value estimates gradually approach the optimal ones (see Figure 5.4(h)).

As the episode length increases, an interesting spatial pattern emerges. That is, the states that are closer to the target possess nonzero values earlier than those that are farther away. The reason for this phenomenon is as follows. Starting from a state, the agent must travel at least a certain number of steps to reach the target state and then receive positive rewards. If the length of an episode is less than the minimum desired number of steps, it is certain that the return is zero, and so is the estimated state value. In this example, the episode length must be no less than 15, which is the minimum number of steps required to reach the target when starting from the bottom-left state.

While the above analysis suggests that each episode must be sufficiently long, the episodes are not necessarily infinitely long. As shown in Figure 5.4(g), when the length is 30, the algorithm can find an optimal policy, although the value estimate is not yet optimal.

The above analysis is related to an important reward design problem, *sparse reward*, which refers to the scenario in which no positive rewards can be obtained unless the target is reached. The sparse reward setting requires long episodes that can reach the target. This requirement is challenging to satisfy when the state space is large. As a result, the sparse reward problem downgrades the learning efficiency. One simple technique for solving this problem is to design *nonsparse rewards*. For instance, in the above grid world example, we can redesign the reward setting so that the agent can obtain a small positive reward when reaching the states near the target. In this way, an "attractive field" can be formed around the target so that the agent can find the target more easily. More information about sparse reward problems can be found in [17–19].

5.3 MC Exploring Starts

We next extend the MC Basic algorithm to obtain another MC-based reinforcement learning algorithm that is slightly more complicated but more sample-efficient.

5.3.1 Utilizing samples more efficiently

An important aspect of MC-based reinforcement learning is how to use samples more efficiently. Specifically, suppose that we have an episode of samples obtained by following a policy π:

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \tag{5.3}$$

where the subscripts refer to the state or action indexes rather than time steps. Every time a state-action pair appears in an episode, it is called a *visit* of that state-action pair. Different strategies can be employed to utilize the visits.

The first and simplest strategy is to use the *initial visit*. That is, an episode is only used to estimate the action value of the initial state-action pair that the episode starts from. For the example in (5.3), the initial-visit strategy merely estimates the action value of (s_1, a_2). The MC Basic algorithm utilizes the initial-visit strategy. However, this strategy is *not sample-efficient* because the episode also visits many other state-action pairs such as (s_2, a_4), (s_2, a_3), and (s_5, a_1). These visits can also be used to estimate the corresponding action values. In particular, we can decompose the episode in (5.3) into multiple subepisodes:

$$
\begin{aligned}
&s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots && \text{[original episode]} \\
&\quad s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots && \text{[subepisode starting from } (s_2, a_4)\text{]} \\
&\quad\quad s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots && \text{[subepisode starting from } (s_1, a_2)\text{]} \\
&\quad\quad\quad s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots && \text{[subepisode starting from } (s_2, a_3)\text{]} \\
&\quad\quad\quad\quad s_5 \xrightarrow{a_1} \dots && \text{[subepisode starting from } (s_5, a_1)\text{]}
\end{aligned}
$$

The trajectory generated after the visit of a state-action pair can be viewed as a new episode. These new episodes can be used to estimate more action values. In this way, the samples in the episode can be utilized more efficiently.

Moreover, a state-action pair may be visited multiple times in an episode. For example, (s_1, a_2) is visited twice in the episode in (5.3). If we only count the first-time visit, this is called a *first-visit* strategy. If we count every visit of a state-action pair, such a strategy is called *every-visit* [20].

In terms of sample usage efficiency, the every-visit strategy is the best. If an episode is sufficiently long such that it can visit all the state-action pairs many times, then this single episode may be sufficient for estimating all the action values using the every-visit strategy. However, the samples obtained by the every-visit strategy are correlated because the trajectory starting from the second visit is merely a subset of the trajectory starting from the first visit. Nevertheless, the correlation would not be strong if the two visits are far away from each other in the trajectory.

5.3.2 Updating policies more efficiently

Another aspect of MC-based reinforcement learning is when to update the policy. Two strategies are available.

- ◇ The first strategy is, in the policy evaluation step, to collect all the episodes starting from the same state-action pair and then approximate the action value using the *average return of these episodes*. This strategy is adopted in the MC Basic algorithm.

Algorithm 5.2: MC Exploring Starts (an efficient variant of MC Basic)

Initialization: Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all (s,a). $\text{Returns}(s,a) = 0$ and $\text{Num}(s,a) = 0$ for all (s,a).
Goal: Search for an optimal policy.

For each episode, do
Episode generation: Select a starting state-action pair (s_0, a_0) and ensure that all pairs can be possibly selected (this is the exploring-starts condition). Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
Initialization for each episode: $g \leftarrow 0$
For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
$g \leftarrow \gamma g + r_{t+1}$
$\text{Returns}(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) + g$
$\text{Num}(s_t, a_t) \leftarrow \text{Num}(s_t, a_t) + 1$
Policy evaluation:
$q(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t)/\text{Num}(s_t, a_t)$
Policy improvement:
$\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$ and $\pi(a|s_t) = 0$ otherwise

The drawback of this strategy is that the agent must wait until all the episodes have been collected before the estimate can be updated.

◇ The second strategy, which can overcome this drawback, is to use the *return of a single episode* to approximate the corresponding action value. In this way, we can immediately obtain a rough estimate when we receive an episode. Then, the policy can be improved in an episode-by-episode fashion.

Since the return of a single episode cannot accurately approximate the corresponding action value, one may wonder whether the second strategy is good. In fact, this strategy falls into the scope of *generalized policy iteration* introduced in the last chapter. That is, we can still update the policy even if the value estimate is not sufficiently accurate.

5.3.3 Algorithm description

We can use the techniques introduced in Sections 5.3.1 and 5.3.2 to enhance the efficiency of the MC Basic algorithm. Then, a new algorithm called *MC Exploring Starts* can be obtained.

The details of MC Exploring Starts are given in Algorithm 5.2. This algorithm uses the every-visit strategy. Interestingly, when calculating the discounted return obtained by starting from each state-action pair, the procedure starts from the ending states and travels back to the starting state. Such techniques can make the algorithm more efficient, but it also makes the algorithm more complex. This is why the MC Basic algorithm,

which is free of such techniques, is introduced first to reveal the core idea of MC-based reinforcement learning.

The *exploring starts* condition requires sufficiently many episodes starting from *every* state-action pair. Only if every state-action pair is well explored, can we accurately estimate their action values (according to the law of large numbers) and hence successfully find optimal policies. Otherwise, if an action is not well explored, its action value may be inaccurately estimated, and this action may not be selected by the policy even though it is indeed the best action. Both MC Basic and MC Exploring Starts require this condition. However, this condition is difficult to meet in many applications, especially those involving physical interactions with environments. Can we remove the exploring starts requirement? The answer is yes, as shown in the next section.

5.4 MC ϵ-Greedy: Learning without exploring starts

We next extend the MC Exploring Starts algorithm by removing the exploring starts condition. This condition actually requires that every state-action pair can be visited sufficiently many times, which can also be achieved based on soft policies.

5.4.1 ϵ-greedy policies

A policy is *soft* if it has a positive probability of taking any action at any state. Consider an extreme case in which we only have a single episode. With a soft policy, a single episode that is sufficiently long can visit *every* state-action pair many times (see the examples in Figure 5.8). Thus, we do not need to generate a large number of episodes starting from different state-action pairs, and then the exploring starts requirement can be removed.

One type of common soft policies is *ϵ-greedy* policies. An ϵ-greedy policy is a stochastic policy that has a higher chance of choosing the *greedy action* and the same nonzero probability of taking any other action. Here, the greedy action refers to the action with the greatest action value. In particular, suppose that $\epsilon \in [0, 1]$. The corresponding ϵ-greedy policy has the following form:

$$\pi(a|s) = \begin{cases} 1 - \dfrac{\epsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1), & \text{for the greedy action,} \\ \dfrac{\epsilon}{|\mathcal{A}(s)|}, & \text{for the other } |\mathcal{A}(s)| - 1 \text{ actions,} \end{cases}$$

where $|\mathcal{A}(s)|$ denotes the number of actions associated with s.

When $\epsilon = 0$, ϵ-greedy becomes greedy. When $\epsilon = 1$, the probability of taking any action equals $\frac{1}{|\mathcal{A}(s)|}$.

The probability of taking the greedy action is always greater than that of taking any

other action because

$$1 - \frac{\epsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} \geq \frac{\epsilon}{|\mathcal{A}(s)|}$$

for any $\epsilon \in [0, 1]$.

While an ϵ-greedy policy is stochastic, how can we select an action by following such a policy? We can first generate a random number x in $[0, 1]$ by following a uniform distribution. If $x \geq \epsilon$, then we select the greedy action. If $x < \epsilon$, then we randomly select an action in $\mathcal{A}(s)$ with the probability of $\frac{1}{|\mathcal{A}(s)|}$ (we may select the greedy action again). In this way, the total probability of selecting the greedy action is $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$, and the probability of selecting any other action is $\frac{\epsilon}{|\mathcal{A}(s)|}$.

5.4.2 Algorithm description

To integrate ϵ-greedy policies into MC learning, we only need to change the policy improvement step from greedy to ϵ-greedy.

In particular, the policy improvement step in MC Basic or MC Exploring Starts aims to solve

$$\pi_{k+1}(s) = \arg\max_{\pi \in \Pi} \sum_a \pi(a|s) q_{\pi_k}(s, a), \tag{5.4}$$

where Π denotes *the set of all possible policies*. We know that the solution of (5.4) is a greedy policy:

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

where $a_k^* = \arg\max_a q_{\pi_k}(s, a)$.

Now, the policy improvement step is changed to solve

$$\pi_{k+1}(s) = \arg\max_{\pi \in \Pi_\epsilon} \sum_a \pi(a|s) q_{\pi_k}(s, a), \tag{5.5}$$

where Π_ϵ denotes *the set of all ϵ-greedy policies* with a given value of ϵ. In this way, we force the policy to be ϵ-greedy. The solution of (5.5) is

$$\pi_{k+1}(a|s) = \begin{cases} 1 - \frac{|\mathcal{A}(s)|-1}{|\mathcal{A}(s)|}\epsilon, & a = a_k^*, \\ \frac{1}{|\mathcal{A}(s)|}\epsilon, & a \neq a_k^*, \end{cases}$$

where $a_k^* = \arg\max_a q_{\pi_k}(s, a)$. With the above change, we obtain another algorithm called *MC ϵ-Greedy*. The details of this algorithm are given in Algorithm 5.3. Here, the every-visit strategy is employed to better utilize the samples.

Algorithm 5.3: MC ϵ-Greedy (a variant of MC Exploring Starts)

Initialization: Initial policy $\pi_0(a|s)$ and initial value $q(s,a)$ for all (s,a). Returns$(s,a) = 0$ and Num$(s,a) = 0$ for all (s,a). $\epsilon \in (0,1]$
Goal: Search for an optimal policy.

For each episode, do

Episode generation: Select a starting state-action pair (s_0, a_0) (the exploring starts condition is not required). Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.

Initialization for each episode: $g \leftarrow 0$

For each step of the episode, $t = T-1, T-2, \ldots, 0$, do

$g \leftarrow \gamma g + r_{t+1}$

Returns$(s_t, a_t) \leftarrow$ Returns$(s_t, a_t) + g$

Num$(s_t, a_t) \leftarrow$ Num$(s_t, a_t) + 1$

Policy evaluation:

$q(s_t, a_t) \leftarrow$ Returns(s_t, a_t)/Num(s_t, a_t)

Policy improvement:

Let $a^* = \arg\max_a q(s_t, a)$ and

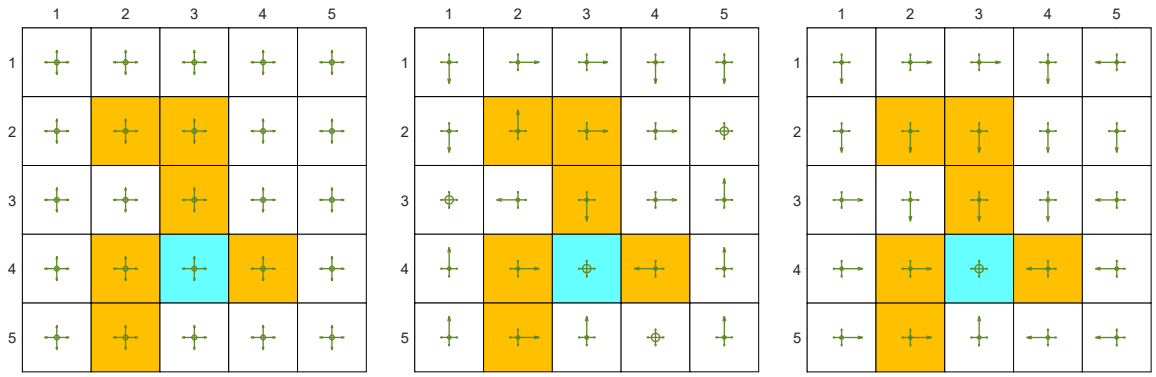
$$\pi(a|s_t) = \begin{cases} 1 - \frac{|\mathcal{A}(s_t)|-1}{|\mathcal{A}(s_t)|}\epsilon, & a = a^* \\ \frac{1}{|\mathcal{A}(s_t)|}\epsilon, & a \neq a^* \end{cases}$$

If greedy policies are replaced by ϵ-greedy policies in the policy improvement step, can we still guarantee to obtain optimal policies? The answer is both yes and no. By yes, we mean that, when given sufficient samples, the algorithm can converge to an ϵ-greedy policy that is optimal in the set Π_ϵ. By no, we mean that the policy is merely optimal in Π_ϵ but may not be optimal in Π. However, if ϵ is sufficiently small, the optimal policies in Π_ϵ are close to those in Π.

5.4.3 Illustrative examples

Consider the grid world example shown in Figure 5.5. The aim is to find the optimal policy for every state. A single episode with one million steps is generated in every iteration of the MC ϵ-Greedy algorithm. Here, we deliberately consider the extreme case with merely one single episode. We set $r_{\text{boundary}} = r_{\text{forbidden}} = -1$, $r_{\text{target}} = 1$, and $\gamma = 0.9$.

The initial policy is a uniform policy that has the same probability 0.2 of taking any action, as shown in Figure 5.5. The optimal ϵ-greedy policy with $\epsilon = 0.5$ can be obtained after two iterations. Although each iteration merely uses a single episode, the policy gradually improves because all the state-action pairs can be visited and hence their values can be accurately estimated.



(a) Initial policy (b) After the first iteration (c) After the second iteration

Figure 5.5: The evolution process of the MC ϵ-Greedy algorithm based on single episodes.

5.5 Exploration and exploitation of ϵ-greedy policies

Exploration and *exploitation* constitute a fundamental tradeoff in reinforcement learning. Here, exploration means that the policy can possibly take as many actions as possible. In this way, all the actions can be visited and evaluated well. Exploitation means that the improved policy should take the greedy action that has the greatest action value. However, since the action values obtained at the current moment may not be accurate due to insufficient exploration, we should keep exploring while conducting exploitation to avoid missing optimal actions.

ϵ-greedy policies provide one way to balance exploration and exploitation. On the one hand, an ϵ-greedy policy has a higher probability of taking the greedy action so that it can exploit the estimated values. On the other hand, the ϵ-greedy policy also has a chance to take other actions so that it can keep exploring. ϵ-greedy policies are used not only in MC-based reinforcement learning but also in other reinforcement learning algorithms such as temporal-difference learning as introduced in Chapter 7.

Exploitation is related to *optimality* because optimal policies should be greedy. The fundamental idea of ϵ-greedy policies is to enhance exploration by sacrificing optimality/exploitation. If we would like to enhance exploitation and optimality, we need to reduce the value of ϵ. However, if we would like to enhance exploration, we need to increase the value of ϵ.

We next discuss this tradeoff based on some interesting examples. The reinforcement learning task here is a 5-by-5 grid world. The reward settings are $r_{\text{boundary}} = -1$, $r_{\text{forbidden}} = -10$, and $r_{\text{target}} = 1$. The discount rate is $\gamma = 0.9$.

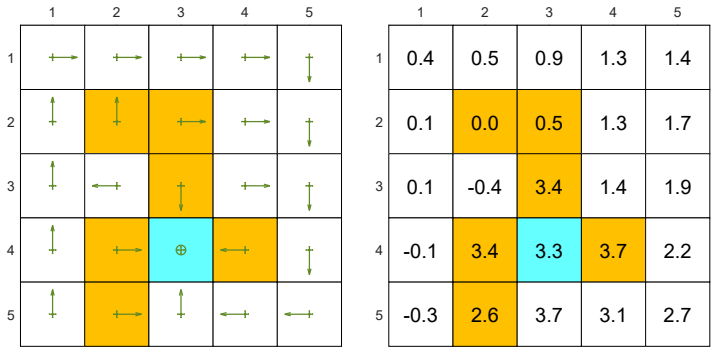
Optimality of ϵ-greedy policies

We next show that the optimality of ϵ-greedy policies becomes worse when ϵ increases.

- ◇ First, a greedy optimal policy and the corresponding optimal state values are shown in Figure 5.6(a). The state values of some *consistent* ϵ-greedy policies are shown in

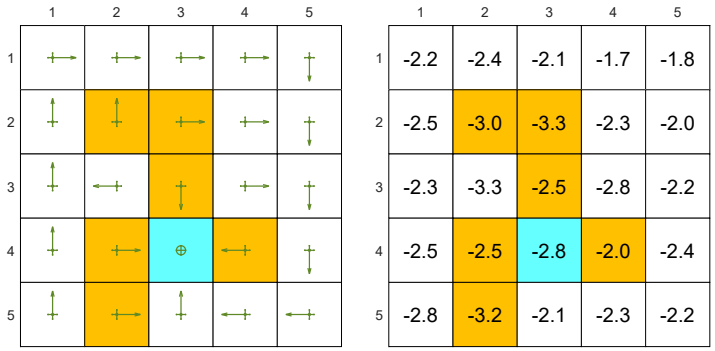
	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

(a) A given ϵ-greedy policy and its state values: $\epsilon = 0$



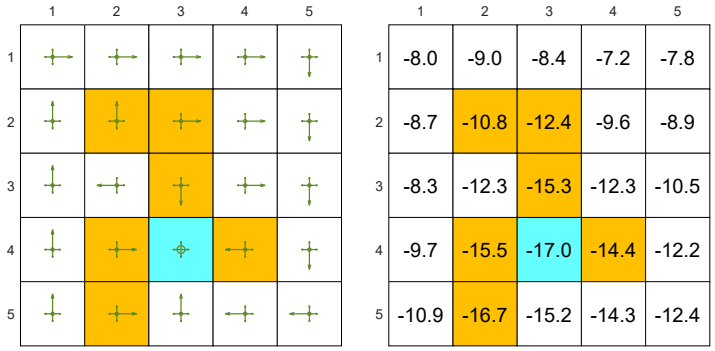
	1	2	3	4	5
1	0.4	0.5	0.9	1.3	1.4
2	0.1	0.0	0.5	1.3	1.7
3	0.1	-0.4	3.4	1.4	1.9
4	-0.1	3.4	3.3	3.7	2.2
5	-0.3	2.6	3.7	3.1	2.7

(b) A given ϵ-greedy policy and its state values: $\epsilon = 0.1$



	1	2	3	4	5
1	-2.2	-2.4	-2.1	-1.7	-1.8
2	-2.5	-3.0	-3.3	-2.3	-2.0
3	-2.3	-3.3	-2.5	-2.8	-2.2
4	-2.5	-2.5	-2.8	-2.0	-2.4
5	-2.8	-3.2	-2.1	-2.3	-2.2

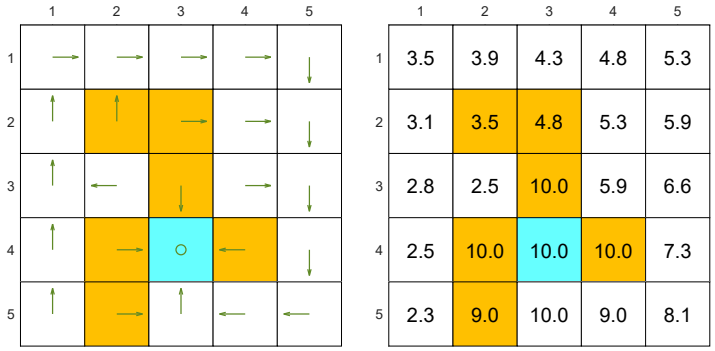
(c) A given ϵ-greedy policy and its state values: $\epsilon = 0.2$



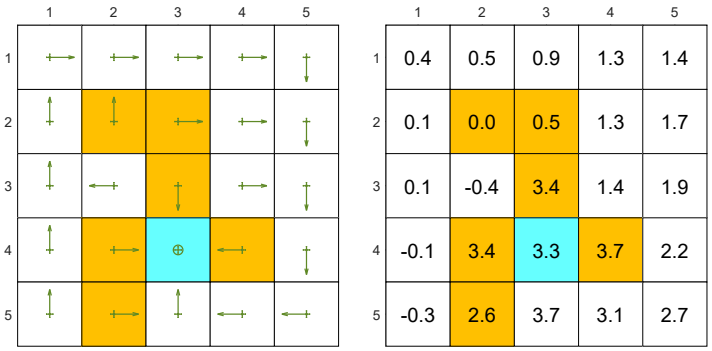
	1	2	3	4	5
1	-8.0	-9.0	-8.4	-7.2	-7.8
2	-8.7	-10.8	-12.4	-9.6	-8.9
3	-8.3	-12.3	-15.3	-12.3	-10.5
4	-9.7	-15.5	-17.0	-14.4	-12.2
5	-10.9	-16.7	-15.2	-14.3	-12.4

(d) A given ϵ-greedy policy and its state values: $\epsilon = 0.5$

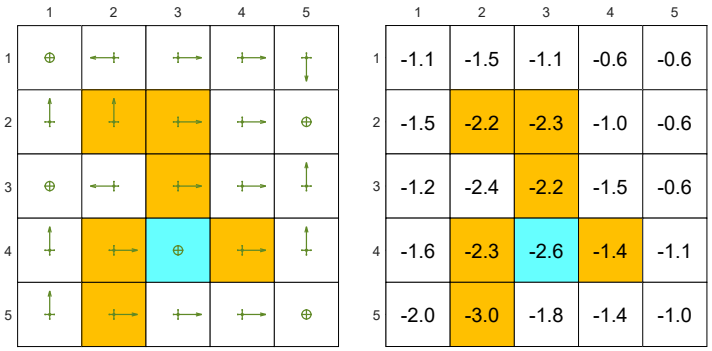
Figure 5.6: The state values of some ϵ-greedy policies. These ϵ-greedy policies are consistent with each other in the sense that the actions with the greatest probabilities are the same. It can be seen that, when the value of ϵ increases, the state values of the ϵ-greedy policies decrease and hence their optimality becomes worse.



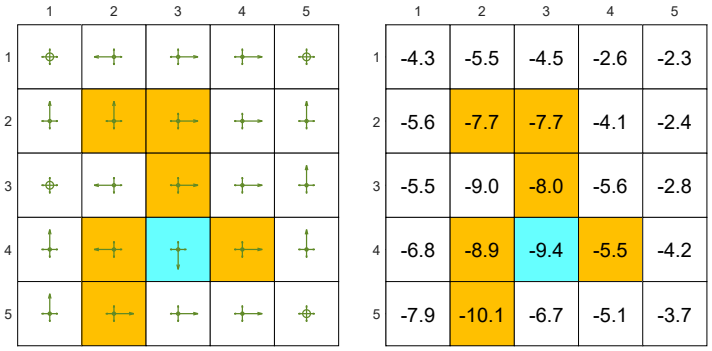
(a) The optimal ϵ-greedy policy and its state values: $\epsilon = 0$



(b) The optimal ϵ-greedy policy and its state values: $\epsilon = 0.1$



(c) The optimal ϵ-greedy policy and its state values: $\epsilon = 0.2$



(d) The optimal ϵ-greedy policy and its state values: $\epsilon = 0.5$

Figure 5.7: The optimal ϵ-greedy policies and their corresponding state values under different values of ϵ. Here, these ϵ-greedy policies are optimal among all ϵ-greedy ones (with the same value of ϵ). It can be seen that, when the value of ϵ increases, the optimal ϵ-greedy policies are no longer consistent with the optimal one as in (a).

Figures 5.6(b)-(d). Here, two ϵ-greedy policies are *consistent* if the actions with the greatest probabilities in the policies are the same.

As the value of ϵ increases, the state values of the ϵ-greedy policies decrease, indicating that the optimality of these ϵ-greedy policies becomes worse. Notably, the value of the target state becomes the smallest when ϵ is as large as 0.5. This is because, when ϵ is large, the agent starting from the target area may enter the surrounding forbidden areas and hence receive negative rewards with a higher probability.

◇ Second, Figure 5.7 shows the optimal ϵ-greedy policies (they are optimal in Π_ϵ). When $\epsilon = 0$, the policy is greedy and optimal among all policies. When ϵ is as small as 0.1, the optimal ϵ-greedy policy is consistent with the optimal greedy one. However, when ϵ increases to, for example, 0.2, the obtained ϵ-greedy policies are not consistent with the optimal greedy one. Therefore, if we want to obtain ϵ-greedy policies that are consistent with the optimal greedy ones, the value of ϵ should be sufficiently small.

Why are the ϵ-greedy policies inconsistent with the optimal greedy one when ϵ is large? We can answer this question by considering the target state. In the greedy case, the optimal policy at the target state is to stay still to gain positive rewards. However, when ϵ is large, there is a high chance of entering the forbidden areas and receiving negative rewards. Therefore, the optimal policy at the target state in this case is to escape instead of staying still.

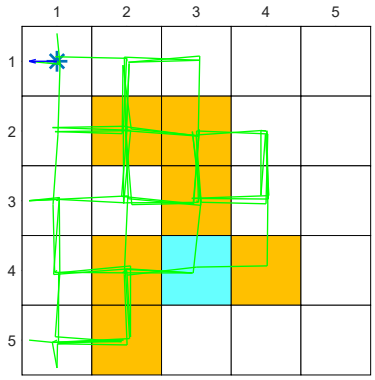
Exploration abilities of ϵ-greedy policies

We next illustrate that the exploration ability of an ϵ-greedy policy is strong when ϵ is large.

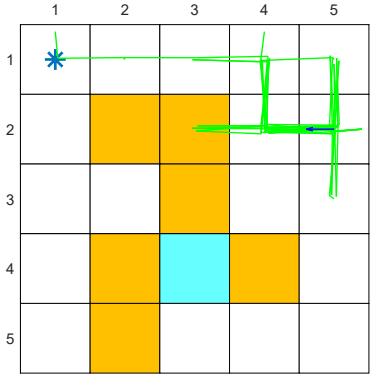
First, consider an ϵ-greedy policy with $\epsilon = 1$ (see Figure 5.5(a)). In this case, the exploration ability of the ϵ-greedy policy is strong since it has a 0.2 probability of taking any action at any state. Starting from (s_1, a_1), an episode generated by the ϵ-policy is given in Figures 5.8(a)-(c). It can be seen that this single episode can visit all the state-action pairs many times when the episode is sufficiently long due to the strong exploration ability of the policy. Moreover, the numbers of times that all the state-action pairs are visited are almost even, as shown in Figure 5.8(d).

Second, consider an ϵ-policy with $\epsilon = 0.5$ (see Figure 5.6(d)). In this case, the ϵ-greedy policy has a weaker exploration ability than the case of $\epsilon = 1$. Starting from (s_1, a_1), an episode generated by the ϵ-policy is given in Figures 5.8(e)-(g). Although every action can still be visited when the episode is sufficiently long, the distribution of the number of visits may be extremely uneven. For example, given an episode with one million steps, some actions are visited more than 250,000 times, while most actions are visited merely hundreds or even tens of times, as shown in Figure 5.8(h).

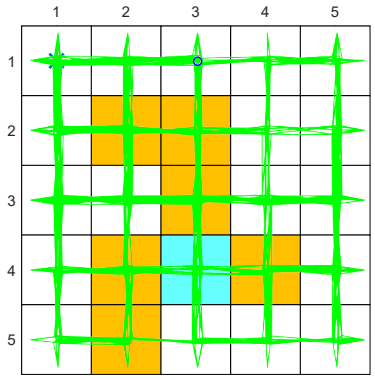
The above examples demonstrate that the exploration abilities of ϵ-greedy policies decrease when ϵ decreases. One useful technique is to initially set ϵ to be large to enhance



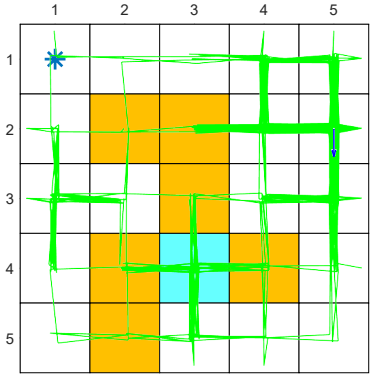
(a) $\epsilon = 1$, trajectory of 100 steps



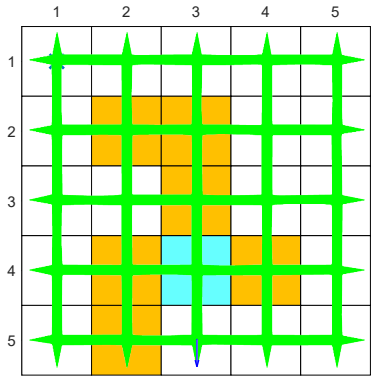
(e) $\epsilon = 0.5$, trajectory of 100 steps



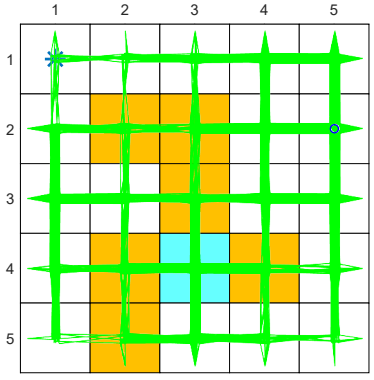
(b) $\epsilon = 1$, trajectory of 1,000 steps



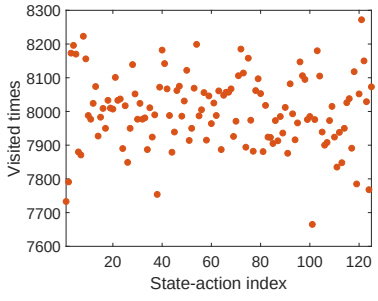
(f) $\epsilon = 0.5$, trajectory of 1,000 steps



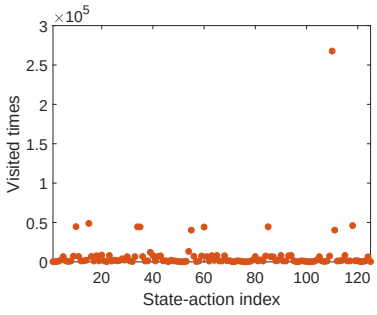
(c) $\epsilon = 1$, trajectory of 10,000 steps



(g) $\epsilon = 0.5$, trajectory of 10,000 steps



(d) $\epsilon = 1$, number of times each action is visited within 1 million steps



(h) $\epsilon = 0.5$, number of times each action is visited within 1 million steps

Figure 5.8: Exploration abilities of ϵ-greedy policies with different values of ϵ.

exploration and gradually reduce it to ensure the optimality of the final policy [21–23].

5.6 Summary

The algorithms in this chapter are the first model-free reinforcement learning algorithms ever introduced in this book. We first introduced the idea of MC estimation by examining an important mean estimation problem. Then, three MC-based algorithms were introduced.

- ◇ MC Basic: This is the simplest MC-based reinforcement learning algorithm. This algorithm is obtained by replacing the model-based policy evaluation step in the policy iteration algorithm with a model-free MC-based estimation component. Given sufficient samples, it is guaranteed that this algorithm can converge to optimal policies and optimal state values.
- ◇ MC Exploring Starts: This algorithm is a variant of MC Basic. It can be obtained from the MC Basic algorithm using the first-visit or every-visit strategy to use samples more efficiently.
- ◇ MC ϵ-Greedy: This algorithm is a variant of MC Exploring Starts. Specifically, in the policy improvement step, it searches for the best ϵ-greedy policies instead of greedy policies. In this way, the exploration ability of the policy is enhanced and hence the condition of exploring starts can be removed.

Finally, a tradeoff between exploration and exploitation was introduced by examining the properties of ϵ-greedy policies. As the value of ϵ increases, the exploration ability of ϵ-greedy policies increases, and the exploitation of greedy actions decreases. On the other hand, if the value of ϵ decreases, we can better exploit the greedy actions, but the exploration ability is compromised.

5.7 Q&A

- ◇ Q: What is Monte Carlo estimation?

  A: Monte Carlo estimation refers to a broad class of techniques that use stochastic samples to solve approximation problems.

- ◇ Q: What is the mean estimation problem?

  A: The mean estimation problem refers to calculating the expected value of a random variable based on stochastic samples.

- ◇ Q: How to solve the mean estimation problem?

  A: There are two approaches: model-based and model-free. In particular, if the probability distribution of a random variable is known, the expected value can be calculated

based on its definition. If the probability distribution is unknown, we can use Monte Carlo estimation to approximate the expected value. Such an approximation is accurate when the number of samples is large.

◇ Q: Why is the mean estimation problem important for reinforcement learning?

A: Both state and action values are defined as expected values of returns. Hence, estimating state or action values is essentially a mean estimation problem.

◇ Q: What is the core idea of model-free MC-based reinforcement learning?

A: The core idea is to convert the policy iteration algorithm to a model-free one. In particular, while the policy iteration algorithm aims to calculate values based on the system model, MC-based reinforcement learning replaces the model-based policy evaluation step in the policy iteration algorithm with a model-free MC-based policy evaluation step.

◇ Q: What are initial-visit, first-visit, and every-visit strategies?

A: They are different strategies for utilizing the samples in an episode. An episode may visit many state-action pairs. The initial-visit strategy uses the entire episode to estimate the action value of the initial state-action pair. The every-visit and first-visit strategies can better utilize the given samples. If the rest of the episode is used to estimate the action value of a state-action pair every time it is visited, such a strategy is called every-visit. If we only count the first time a state-action pair is visited in the episode, such a strategy is called first-visit.

◇ Q: What is exploring starts? Why is it important?

A: Exploring starts requires an infinite number of (or sufficiently many) episodes to be generated when starting from every state-action pair. In theory, the exploring starts condition is necessary to find optimal policies. That is, only if every action value is well explored, can we accurately evaluate all the actions and then correctly select the optimal ones.

◇ Q: What is the idea used to avoid exploring starts?

A: The fundamental idea is to make policies soft. Soft policies are stochastic, enabling an episode to visit many state-action pairs. In this way, we do not need a large number of episodes starting from every state-action pair.

◇ Q: Can an ϵ-greedy policy be optimal?

A: The answer is both yes and no. By yes, we mean that, if given sufficient samples, the MC ϵ-Greedy algorithm can converge to an optimal ϵ-greedy policy. By no, we mean that the converged policy is merely optimal among all ϵ-greedy policies (with the same value of ϵ).

◇ Q: Is it possible to use one episode to visit all state-action pairs?

A: Yes, it is possible. If the policy is soft (e.g., ϵ-greedy) and the episode is sufficiently long.

◇ Q: What is the relationship between MC Basic, MC Exploring Starts, and MC ϵ-Greedy?

A: MC Basic is the simplest MC-based reinforcement learning algorithm. It is important because it reveals the fundamental idea of model-free MC-based reinforcement learning. MC Exploring Starts is a variant of MC Basic that adjusts the sample usage strategy. Furthermore, MC ϵ-Greedy is a variant of MC Exploring Starts that removes the exploring starts requirement. Therefore, while the basic idea is simple, complication appears when we want to achieve better performance. It is important to split the core idea from the complications that may be distracting for beginners.