第5章 蒙特卡洛方法 (Monte Carlo Methods)

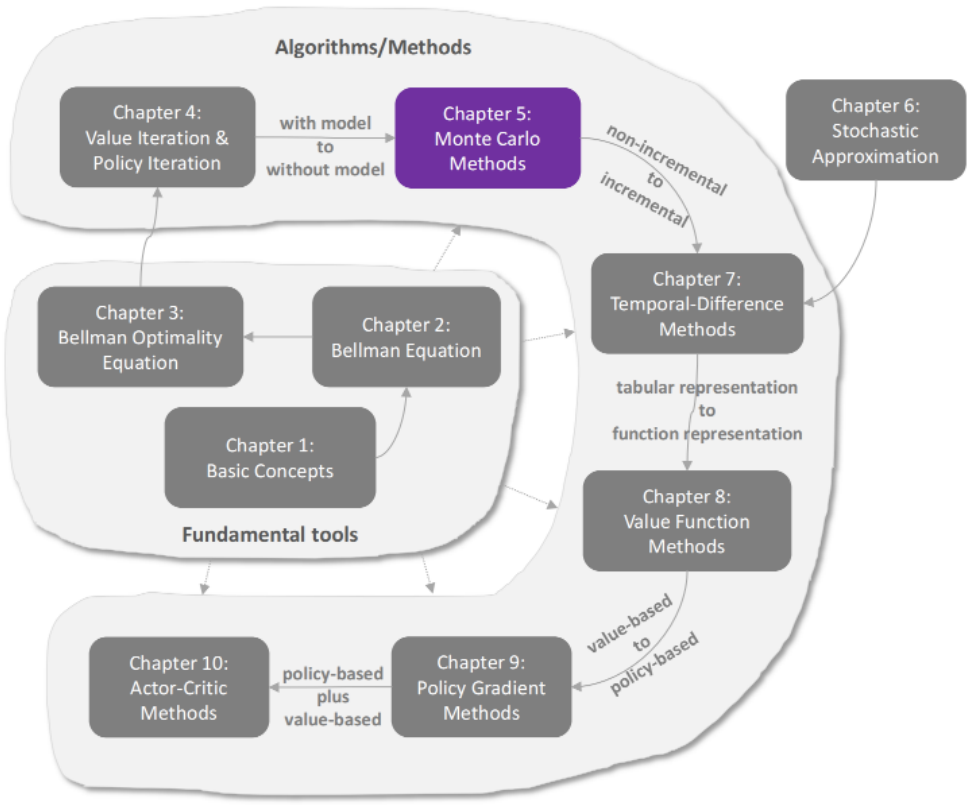


Figure 5.1: Where we are in this book.

图 5.1：本书的结构脉络。

在上一章中，我们介绍了基于系统模型寻找最优策略的算法。在本章中，我们将开始介绍不预设系统模型的*无模型 (model-free)* 强化学习算法。

虽然这是本书首次介绍无模型算法，但我们必须填补一个知识空白：如何在没有模型的情况下找到最优策略？其基本思想很简单：如果我们没有模型，我们就必须有一些数据。如果我们没有数据，我们就必须有一个模型。如果二者皆无，那么我们就无法找到最优策略。强化学习中的“数据”通常指的是智能体与环境交互的经验。

为了演示如何从数据而不是模型中学习，我们在本章开头先介绍*均值估计 (mean estimation)* 问题，即通过一些样本来估计随机变量的期望值。理解这个问题对于掌握*从数据中学习 (learning from data)* 的基本思想至关重要。

然后，我们将介绍三种基于蒙特卡洛 (Monte Carlo, MC) 方法的算法。这些算法可以从经验样本中学习最优策略。第一个也是最简单的算法称为 MC Basic，它可以通过修改上一章介绍的策略迭代算法轻松获得。理解该算法对于掌握基于 MC 的强化学习的基本思想非常重要。通过扩展该算法，我们进一步介绍了另外两种更复杂但更高效的算法。

5.1 激励性示例：均值估计 (Motivating example: Mean estimation)

接下来我们要介绍 **均值估计 (mean estimation)** 问题，以演示如何从数据而不是模型中学习。我们将看到，均值估计可以通过*蒙特卡洛 (Monte Carlo)* 方法来实现，这是一类使用随机样本来解决估计问题的技术的统称。读者可能会想，我们为什么要关心均值估计问题。原因很简单，因为**状态价值和动作价值都被定义为回报 (return) 的均值**。估计状态或动作价值实际上就是一个均值估计问题。

考虑一个随机变量 X，它可以从一个有限实数集合 $\mathcal{X}$ 中取值。假设我们的任务是计算 X 的均值或期望值：$\mathbb{E}[X]$。有两种方法可以用来计算 $\mathbb{E}[X]$。

- 第一种方法是*基于模型的 (model-based)*。在这里，**模型指的是 X 的概率分布**。如果模型已知，那么均值可以直接根据期望值的定义计算得出：

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

在本书中，我们交替使用术语*期望值 (expected value)*、*均值 (mean)* 和*平均值 (average)*。

- 第二种方法是*无模型的 (model-free)*。当 X 的概率分布（即模型）未知时，假设我们有一些 X 的样本 $\{x_1, x_2, \dots, x_n\}$。那么，均值可以近似为

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

当 n 很小时，这种近似可能不准确。然而，随着 n 的增加，近似会变得越来越准确。当 $n \to \infty$ 时，我们有 $\bar{x} \to \mathbb{E}[X]$。

这由*大数定律 (law of large numbers)* 保证：大量样本的平均值接近于期望值。大数定律将在方框 5.1 中介绍。

下面的例子说明了上述两种方法。考虑一个抛硬币游戏。令随机变量 X 表示硬币落地时显示的一面。X 有两个可能的值：当正面朝上时 $X = 1$，当反面朝上时 $X = -1$。假设 X 的真实概率分布（即模型）为

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

如果概率分布是预先已知的，我们可以直接计算均值为

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

如果概率分布未知，那么我们可以抛很多次硬币并记录采样结果 $\{x_i\}_{i=1}^n$。通过计算样本的平均值，我们可以获得均值的估计。如图 5.2 所示，随着样本数量的增加，估计的均值变得越来越准确。

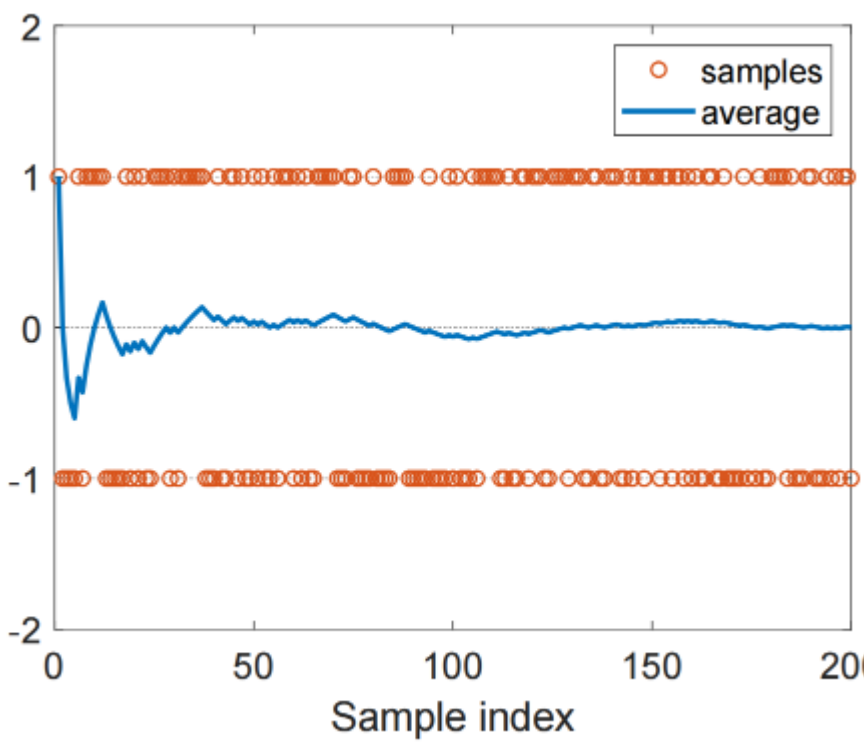


图 5.2：演示大数定律的例子。这里的样本是从遵从均匀分布的 $\{+1, -1\}$ 中抽取的。随着样本数量的增加，样本的平均值逐渐收敛到零（即真实的期望值）。

值得一提的是，用于均值估计的样本必须是*独立同分布的 (independent and identically distributed, i.i.d. 或 iid)*。否则，如果采样值相关，可能就无法正确估计期望值。一个极端的例子是，无论第一个采样值是什么，所有的采样值都与第一个相同。在这种情况下，无论我们使用多少个样本，样本的平均值总是等于第一个样本。

> **方框 5.1：大数定律 (Law of large numbers)**
>
> 对于随机变量 X，假设 $\{x_i\}_{i=1}^n$ 是一些独立同分布 (i.i.d.) 的样本。
>
> 令 $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ 为样本的平均值。那么，
>
> $$\mathbb{E}[\bar{x}] = \mathbb{E}[X],$$
> $$\mathrm{var}[\bar{x}] = \frac{1}{n}\mathrm{var}[X].$$
>
> 上述两个方程表明 $\bar{x}$ 是 $\mathbb{E}[X]$ 的无偏估计，并且随着 n 增加到无穷大，其方差减小到零。
>
> 证明如下。
>
> 首先，$\mathbb{E}[\bar{x}] = \mathbb{E}[\sum_{i=1}^{n} x_i/n] = \sum_{i=1}^{n} \mathbb{E}[x_i]/n = \mathbb{E}[X]$，其中最后一个等式是由于样本是同分布的（即 $\mathbb{E}[x_i] = \mathbb{E}[X]$）。
>
> 其次，$\mathrm{var}(\bar{x}) = \mathrm{var}[\sum_{i=1}^{n} x_i/n] = \sum_{i=1}^{n} \mathrm{var}[x_i]/n^2 = (n \cdot \mathrm{var}[X])/n^2 = \mathrm{var}[X]/n$，其中第二个等式是由于样本是独立的，第三个等式是由于样本是同分布的（即 $\mathrm{var}[x_i] = \mathrm{var}[X]$）。

5.2 MC Basic: 最简单的基于 MC 的算法 (MC Basic: The simplest MC-based algorithm)

本节介绍第一个也是最简单的基于 MC 的强化学习算法。

该算法是通过将 4.2 节介绍的**策略迭代算法**中的***基于模型的策略评估步骤*替换为*无模型的 MC 估计步骤*而获得的**。

5.2.1 将策略迭代转换为无模型形式 (Converting policy iteration to be model-free)

策略迭代算法的每次迭代有两个步骤（见 4.2 节）。

- 第一步是*策略评估*，旨在通过求解 $v_{\pi_k} = r_{\pi_k} + \gamma P_{\pi_k} v_{\pi_k}$ 来计算 v_{π_k}。
- 第二步是*策略改进*，旨在计算贪婪策略 $\pi_{k+1} = \arg\max_{\pi}(r_\pi + \gamma P_\pi v_{\pi_k})$。策略改进步骤的逐元素形式为

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

必须注意的是，动作价值位于这两个步骤的*核心*。

具体来说，在第一步中，计算状态价值是为了计算动作价值。在第二步中，新的策略是基于计算出的动作价值生成的。让我们重新考虑如何计算动作价值。有两种方法可用。

- 第一种是*基于模型的方法 (model-based approach)*。这是策略迭代算法采用的方法。具体来说，我们可以首先通过求解贝尔曼方程来计算状态价值 v_{π_k}。然后，我们可以通过下式计算动作价值：

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

这种方法需要已知系统模型 $\{p(r|s,a), p(s'|s,a)\}$ **(也就是说，你必须要知道转移概率或者奖励概率分布？)**。

- 第二种是*无模型的方法 (model-free approach)*。回顾一下动作价值的定义是

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

即从 (s,a) 开始时获得的期望回报。由于 $q_{\pi_k}(s,a)$ 是一个期望值，它可以像 5.1 节演示的那样通过 MC 方法进行估计。**为此，智能体可以从 (s,a) 开始，按照策略 π_k 与环境交互，从而获得一定数量的 episodes。假设有 n 个episodes，且第 i 个episode的return为 $g_{\pi_k}^{(i)}(s,a)$。那么，$q_{\pi_k}(s,a)$ 可以近似为**

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

我们已经知道，根据大数定律，如果episodes的数量 n 足够大，这种近似将足够准确。

基于 MC 的强化学习的基本思想是使用一种无模型的方法来估计动作价值（如式 (5.2) 所示），以替代策略迭代算法中的基于模型的方法。

5.2.2 MC Basic 算法 (The MC Basic algorithm)

我们现在准备介绍第一个基于 MC 的强化学习算法。从初始策略 π_0 开始，该算法在第 k 次迭代 ($k = 0, 1, 2, \ldots$) 中包含两个步骤。

- *步骤 1：**策略评估 (Policy evaluation)**。*此步骤用于估计所有 (s,a) 的 $q_{\pi_k}(s,a)$。具体来说，对于每一个 (s,a)，我们收集足够多的episodes，并利用回报的平均值（记为 $q_k(s,a)$）来近似 $q_{\pi_k}(s,a)$。
- *步骤 2：**策略改进 (Policy improvement)**。*该步骤求解 $\pi_{k+1}(s) = \arg\max_{\pi} \sum_a \pi(a|s) q_k(s,a)$，对于所有 $s \in \mathcal{S}$。贪婪最优策略为 $\pi_{k+1}(a_k^*|s) = 1$，其中 $a_k^* = \arg\max_a q_k(s,a)$。

> **算法 5.1：MC Basic（策略迭代的无模型变体）**
>
> 初始化：初始猜测 π_0。

> 目标：搜索最优策略。
>
> 对于第 k 次迭代 ($k = 0, 1, 2, \ldots$)，做
>
> > 对于每一个状态 $s \in \mathcal{S}$ ，做
> >
> > > 对于每一个动作 $a \in \mathcal{A}(s)$ ，做
> > >
> > > > 遵循 π_k 从 (s, a) 开始收集足够多的episodes
> > >
> > > 策略评估：
> > >
> > > $q_{\pi_k}(s, a) \approx q_k(s, a) =$ 从 (s, a) 开始的所有episodes的平均回报
> >
> > 策略改进：
> >
> > $a_k^*(s) = \arg\max_a q_k(s, a)$
> >
> > 如果 $a = a_k^*$ ，则 $\pi_{k+1}(a|s) = 1$ ，否则 $\pi_{k+1}(a|s) = 0$

这是最简单的基于 MC 的强化学习算法，在本书中称为 *MC Basic*。**MC Basic 算法的伪代码在算法 5.1 中给出。可以看出，它与策略迭代算法非常相似。唯一的区别在于它直接从经验样本中计算动作价值，而策略迭代先计算状态价值，然后基于系统模型计算动作价值。**值得注意的是，**无模型算法直接估计动作价值。**否则，如果它估计的是状态价值，我们仍然需要使用系统模型根据这些状态价值来计算动作价值，如式 (5.1) 所示。

由于策略迭代是收敛的，因此在给定足够样本的情况下，**MC Basic 也是收敛的。**也就是说，对于每一个 (s, a) ，假设有足够多从 (s, a) 开始的episodes。那么，这些episodes的回报平均值可以准确地近似 (s, a) 的动作价值。在实践中，针对每一个 (s, a) 我们通常没有足够多的episodes。结果是，动作价值的近似可能并不准确。尽管如此，该算法通常仍然有效。这类似于截断策略迭代算法，其中动作价值也没有被精确计算。

最后，MC Basic 由于其低样本效率，过于简单而无法在实际中应用。我们介绍这个算法的原因是让读者掌握基于 MC 的强化学习的核心思想。在学习本章稍后介绍的更复杂的算法之前，充分理解这个算法非常重要。我们将看到，通过扩展 MC Basic 算法，可以很容易地获得更复杂且样本效率更高的算法。

5.2.3 说明性示例 (Illustrative examples)

一个简单的例子：逐步实现

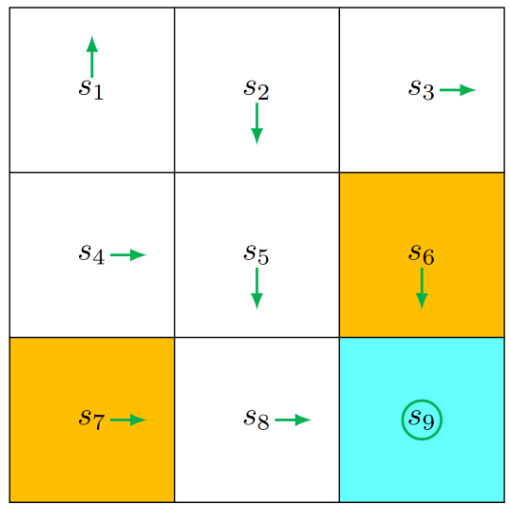


图 5.3：演示 MC Basic 算法的例子。

接下来，我们使用一个例子来演示 MC Basic 算法的实现细节。奖励设置如下： $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ 且 $r_{\text{target}} = 1$ 。折扣率为 $\gamma = 0.9$ 。初始策略 π_0 如图 5.3 所示。该初始策略对于 s_1 或 s_3 并不是最优的。

虽然应该计算所有的动作价值，但由于篇幅限制，我们仅展示 s_1 的动作价值。在 s_1 处，有五个可能的动作。对于每个动作，我们需要收集足够多的episodes以有效地近似动作价值。然而，由于本例在策略和模型方面都是确定性的，多次运行会生成相同的轨迹。因此，估计每个动作价值仅需要一个episode。

遵循 π_0 ，我们可以获得分别从 $(s_1, a_1), (s_1, a_2), \ldots, (s_1, a_5)$ 开始的以下episodes。

- 从 (s_1, a_1) 开始，episode为 $s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$ 。动作价值等于该episode的折现回报：

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

- 从 (s_1, a_2) 开始，episode为 $s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} \ldots$ 。动作价值等于该episode的折现回报：

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

- 从 (s_1,a_3) 开始，episode为 $s_1 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_5 \xrightarrow{a_3} \ldots$。动作价值等于该episode的折现回报：

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

- 从 (s_1,a_4) 开始，episode为 $s_1 \xrightarrow{a_4} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$。动作价值等于该episode的折现回报：

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

- 从 (s_1,a_5) 开始，episode为 $s_1 \xrightarrow{a_5} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \ldots$。动作价值等于该episode的折现回报：

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

通过比较这五个动作价值，我们看到

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

是最大值。因此，可以获得新策略为

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

直观上，改进后的策略（在 s_1 处采取 a_2 或 a_3 ）是最优的。因此，在这个简单的例子中，我们仅通过一次迭代就成功获得了最优策略。在这个简单的例子中，初始策略对于除 s_1 和 s_3 以外的所有状态都已经是最优的。因此，策略仅需一次迭代即可变为最优。当其他状态的策略不是最优时，则需要更多次迭代。

一个综合示例：episode长度与稀疏奖励 (A comprehensive example: Episode length and sparse rewards)

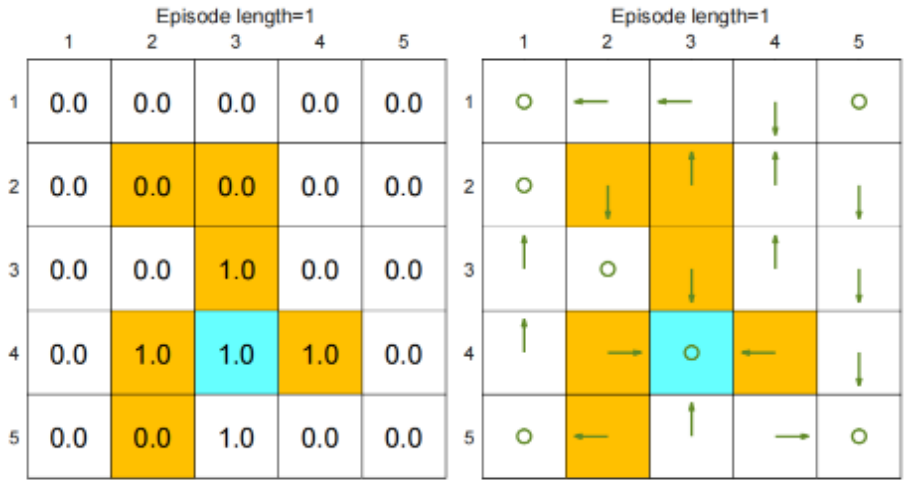
接下来，我们通过考察一个更全面的例子来讨论 MC Basic 算法的一些有趣性质。该例子是一个 5×5 的网格世界（图 5.4）。奖励设置如下：$r_{\text{boundary}} = -1$ ，$r_{\text{forbidden}} = -10$ ，以及 $r_{\text{target}} = 1$ 。折扣率为 $\gamma = 0.9$ 。

首先，我们要证明episode*长度 (episode length)* 对最终的最优策略有很大影响。特别地，图 5.4 展示了 MC Basic 算法在不同episode长度下生成的最终结果。当**每一episode的长度太短时，策略和价值估计都不是最优的**（见图 5.4(a)-(d)）。在episode长度为 1 的极端情况下，只有与目标相邻的状态才具有非零值，而所有其他状态的值都为零，因为每个episode 太短，无法到达目标或获得正奖励（见图 5.4(a)）。随着episode长度的增加，策略和价值估计逐渐接近最优值（见图 5.4(h)）。

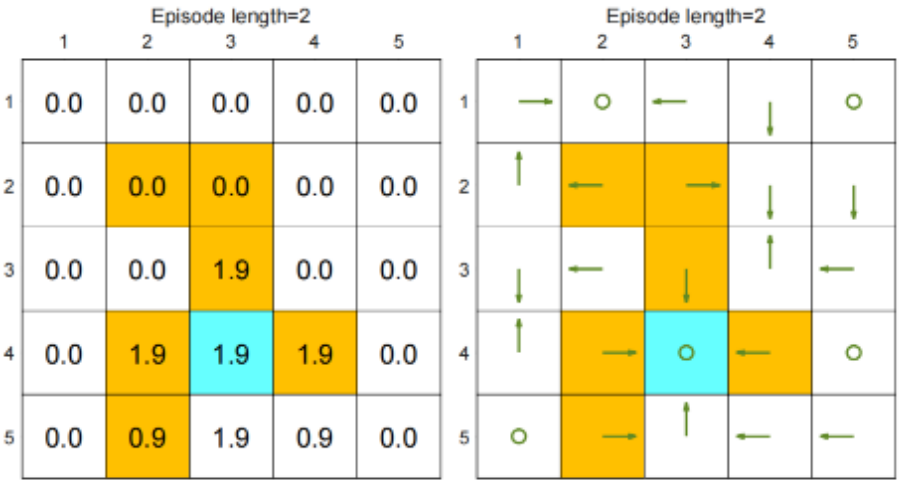
随着episode长度的增加，会出现一种有趣的空间模式。也就是说，距离目标较近的状态比距离较远的状态更早拥有非零值。这种现象的原因如下：从一个状态出发，智能体必须移动至少一定数量的步数才能到达目标状态并获得正奖励。如果幕的长度小于所需的最小步数，那么回报肯定为零，估计的状态价值也是如此。在这个例子中，从左下角状态出发到达目标所需的最小步数为 15，因此episode长度必须不小于 15。

虽然上述分析表明每个episode必须足够长，但episode并不一定需要无限长。如图 5.4(g) 所示，当长度为 30 时，算法可以找到最优策略，尽管价值估计尚未达到最优。

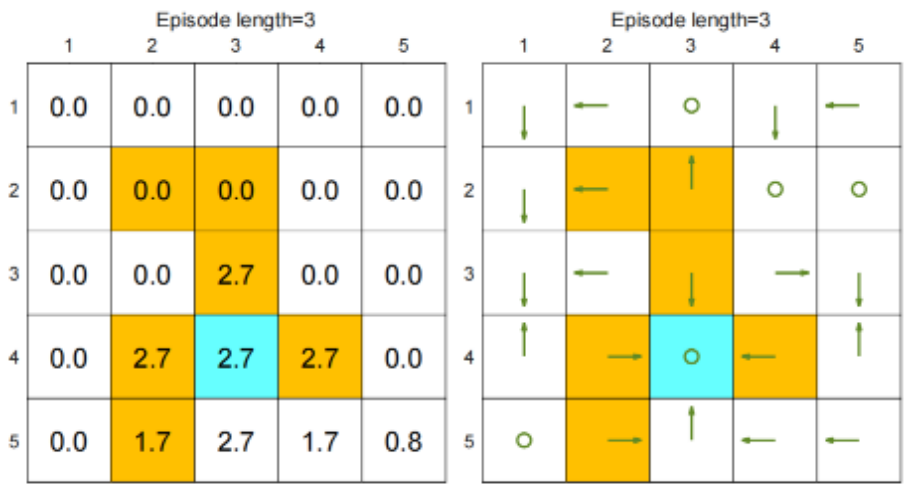
上述分析与一个重要的奖励设计问题有关，即***稀疏奖励 (sparse reward)***，它指的是除非到达目标，否则无法获得正奖励的场景。稀疏奖励设置需要能够到达目标的长episode。当状态空间很大时，这一要求很难满足。结果是，稀疏奖励问题降低了学习效率。解决这个问题的一种简单技术是设计***非稀疏奖励 (nonsparse rewards)***。例如，在上面的网格世界例子中，我们可以重新设计奖励设置，**使得智能体在到达目标附近的状态时可以获得较小的正奖励。通过这种方式，可以在目标周围形成一个“吸引场 (attractive field)”**，以便智能体更容易找到目标。关于稀疏奖励问题的更多信息可以在 [17–19] 中找到。



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

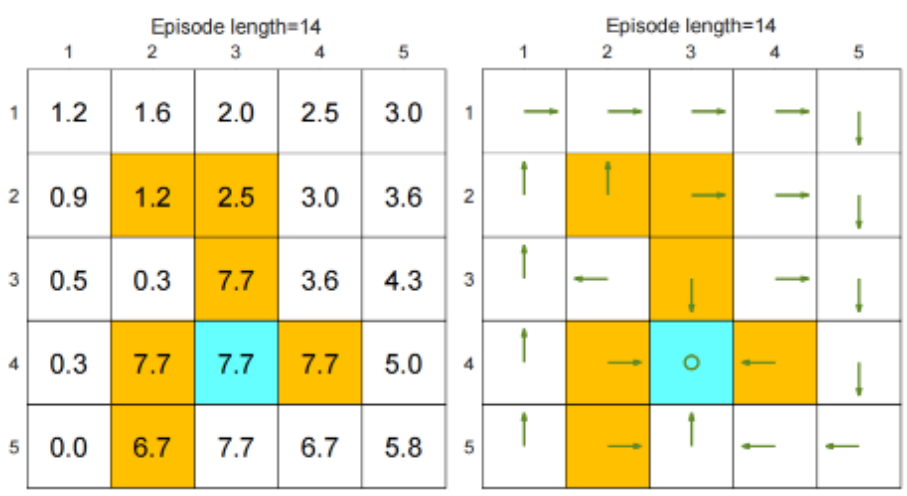


(b) Final value and policy with episode length=2

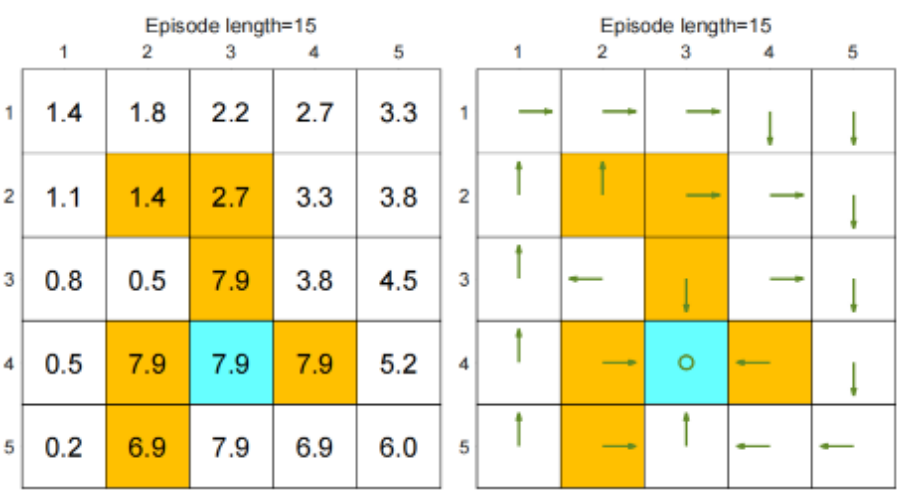


(c) Final value and policy with episode length=3

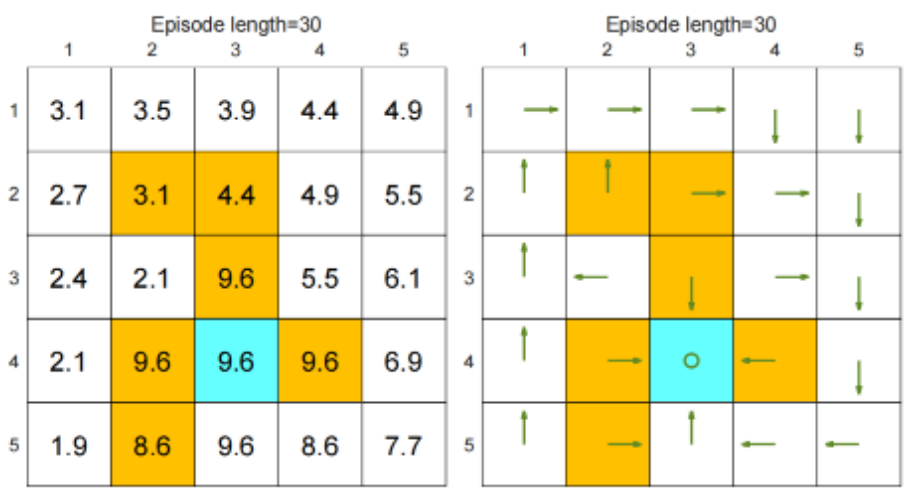
(d) Final value and policy with episode length=4



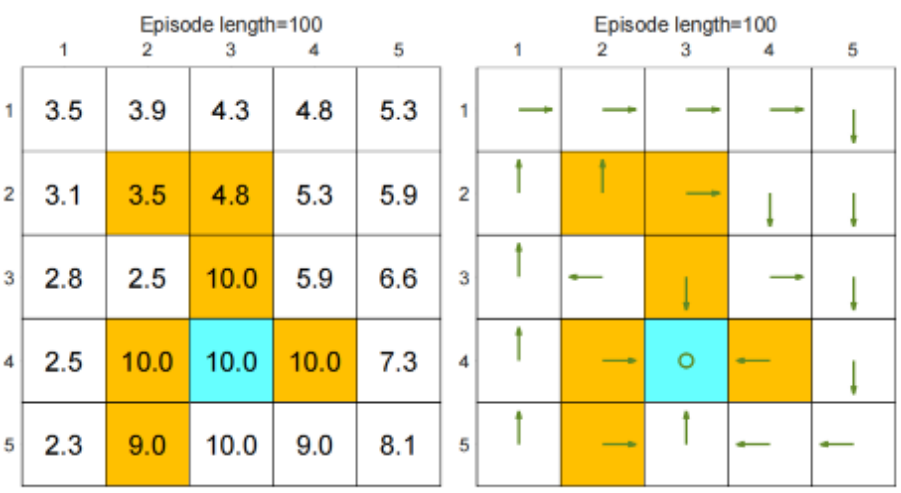
(e) Final value and policy with episode length=14



(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.

5.3 MC 探索起始 (MC Exploring Starts)

接下来，我们将扩展 MC Basic 算法，以获得另一种稍微复杂一些但样本效率更高的基于 MC 的强化学习算法。

5.3.1 更高效地利用样本 (Utilizing samples more efficiently)

基于 MC 的强化学习的一个重要方面是如何更高效地利用样本。具体来说，假设我们有一个通过遵循策略 π 获得的样本episode：

$$s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad (5.3)$$

其中的下标指的是状态或动作的索引，而不是时间步。每当一个状态-动作对出现在一个episode 中时，它被称为该状态-动作对的一次*访问 (visit)*。可以采用不同的策略来利用这些访问。

第一种也是最简单的策略是利用*初始访问 (initial visit)*。也就是说，一个episode仅用于估计该幕开始时的初始状态-动作对的动作价值。对于式 (5.3) 中的例子，初始访问策略仅估计 (s_1, a_2) 的动作价值。MC Basic 算法利用了初始访问策略。然而，这种策略*样本效率不高 (not sample-efficient)*，因为该episode还访问了许多其他状态-动作对，如 (s_2, a_4)，(s_2, a_3) 和 (s_5, a_1)。这些访问也可以用来估计相应的动作价值。具体来说，我们可以将式 (5.3) 中的episode分解为多个子episode：

$$\begin{aligned}
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & [\text{原始episode}] \\
s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & [\text{从 } (s_2, a_4) \text{ 开始的子episode}] \\
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & [\text{从 } (s_1, a_2) \text{ 开始的子episode}] \\
s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \dots \quad & [\text{从 } (s_2, a_3) \text{ 开始的子episode}] \\
s_5 \xrightarrow{a_1} \dots \quad & [\text{从 } (s_5, a_1) \text{ 开始的子episode}]
\end{aligned}$$

在访问某个状态-动作对之后生成的轨迹可以看作是一个新的幕。这些新episode可以用来估计更多的动作价值。通过这种方式，episode中的样本可以被更有效地利用。

此外，一个状态-动作对在一个幕中可能会被访问多次。例如，(s_1, a_2) 在式 (5.3) 的episode中被访问了两次。

如果我们只计算第一次访问，这被称为*首次访问 (first-visit)* 策略。如果我们计算状态-动作对的每一次访问，这种策略被称为*每次访问 (every-visit)* [20]。

就样本使用效率而言，每次访问策略是最好的。如果一个episode足够长，以至于它可以多次访问所有的状态-动作对，那么这单个episode可能足以使用每次访问策略来估计所有的动作价值。**然而，通过每次访问策略获得的样本是相关的，因为从第二次访问开始的轨迹仅仅是从第一次访问开始的轨迹的一个子集。尽管如此，如果两次访问在轨迹中相距很远，相关性就不会很强。**

5.3.2 更高效地更新策略 (Updating policies more efficiently)

基于 MC 的强化学习的另一个方面是何时更新策略。有两种策略可用。

- 第一种策略是，在策略评估步骤中，**收集所有从同一状态-动作对开始的episode，然后使用这些episode的*平均回报 (average return)* 来近似动作价值。**MC Basic 算法采用了这种策略。这种策略的缺点是智能体必须等到收集完所有episode之后才能更新估计值。
- 第二种策略，可以克服这个缺点，是使用*单个episode的回报 (return of a single episode)* 来近似相应的动作价值。通过这种方式，当我们接收到一个*episode*时，就可以立即获得一个粗略的估计。然后，策略可以以逐幕 (episode-by-episode) 的方式进行改进。

由于单个*episode*的回报无法准确地近似相应的动作价值，人们可能会怀疑第二种策略是否有效。事实上，这种策略属于上一章介绍的*广义策略迭代 (generalized policy iteration)* 的范畴。也就是说，即使价值估计不够准确，我们仍然可以更新策略。

5.3.3 算法描述

我们可以使用 5.3.1 和 5.3.2 节介绍的技术来提高 MC Basic 算法的效率。然后，可以获得一种称为 *MC Exploring Starts* 的新算法。

MC Exploring Starts 的详细信息在算法 5.2 中给出。该算法使用了每次访问 (every-visit) 策略。有趣的是，在计算从每个状态-动作对开始获得的折现回报时，该过程从终止状态开始并回溯到起始状态。这种技术可以使算法更高效，但也使算法更复杂。这就是为什么 首先介绍不包含这些技术的（MC Basic 算法），是为了揭示基于 MC 的强化学习的核心思想。

探索起始 (exploring starts) 条件要求有足够多的episodes)从 *每一个* 状态-动作对开始。**只有当每一个状态-动作对都被充分探索时，我们才能准确地估计它们的动作价值（根据大数定律），并因此成功找到最优策略。**否则，如果一个动作没有被充分探索，其动作价值的估计可能不准确，即使它确实是最好的动作，策略也可能不会选择它。MC Basic 和 MC Exploring Starts 都需要这个条

件。然而，在许多应用中，特别是那些涉及与环境进行物理交互的应用中，这个条件很难满足。我们能移除探索起始的要求吗？答案是肯定的，如下一节所示。

5.4 MC ϵ -贪婪：无探索起始的学习 (MC ϵ -Greedy: Learning without exploring starts)

接下来，我们通过移除探索起始条件来扩展 MC Exploring Starts 算法。这个条件实际上要求每一个状态-动作对都能被访问足够多次，这也可以通过软策略 (soft policies) 来实现。

> **算法 5.2：MC Exploring Starts（MC Basic 的一种高效变体）**
>
> 初始化：初始策略 $\pi_0(a|s)$ 和初始价值 $q(s,a)$ 对所有 (s,a) 已知。对于所有 (s,a)，设 $\text{Returns}(s,a)=0$ 且 $\text{Num}(s,a)=0$。
>
> 目标：搜索最优策略。
>
> 对于每一个episode，做
>
> episode生成：选择一个起始状态-动作对 (s_0,a_0) 并确保所有对都有可能被选中（这是探索起始条件）。遵循当前策略，生成一个长度为 T 的episode： $s_0,a_0,r_1,\dots,s_{T-1},a_{T-1},r_T$ 。
>
> 初始化每个episode： $g\leftarrow 0$
>
> 对于episode中的每一步， $t=T-1,T-2,\dots,0$ ，做
>
> $g\leftarrow \gamma g+r_{t+1}$
>
> $\text{Returns}(s_t,a_t)\leftarrow \text{Returns}(s_t,a_t)+g$
>
> $\text{Num}(s_t,a_t)\leftarrow \text{Num}(s_t,a_t)+1$
>
> 策略评估：
>
> $q(s_t,a_t)\leftarrow \text{Returns}(s_t,a_t)/\text{Num}(s_t,a_t)$
>
> 策略改进：
>
> 如果 $a=\arg\max_a q(s_t,a)$ ，则 $\pi(a|s_t)=1$ ，否则 $\pi(a|s_t)=0$

5.4.1 ϵ -贪婪策略 (ϵ -greedy policies)

如果一个策略对任意状态下的任意动作都有正的被选概率，那么该策略就是 *软 (soft)* 的。考虑一个极端情况，即我们只有一个episode。对于一个软策略，只要这个episode足够长，它就可以多次访问 *每一个* 状态-动作对（见图 5.8 的例子）。**因此，我们不需要生成大量从不同状态-动作对开始的episode，从而可以移除探索起始的要求。???**

一种常见的软策略是 ϵ *-贪婪 (greedy)* 策略。 ϵ -贪婪策略是一种随机策略，它以较高的概率选择 *贪婪动作 (greedy action)*，并以相同的非零概率选择任何其他动作。这里，贪婪动作指的是具有最大动作价值的动作。特别地，假设 $\epsilon\in[0,1]$。相应的 ϵ -贪婪策略具有以下形式：

$$\pi(a|s)=\begin{cases}1-\frac{\epsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)|-1), & \text{针对贪婪动作,}\\ \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{针对其他 } |\mathcal{A}(s)|-1 \text{ 个动作,}\end{cases}$$

其中 $|\mathcal{A}(s)|$ 表示与 s 关联的动作数量。

当 $\epsilon=0$ 时， ϵ -贪婪变为贪婪策略。当 $\epsilon=1$ 时，采取任意动作的概率等于 $\frac{1}{|\mathcal{A}(s)|}$。

采取贪婪动作的概率总是大于采取任何其他动作，因为

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

对于任意 $\epsilon\in[0,1]$。

虽然 ϵ -贪婪策略是随机的，但我们如何根据这种策略选择动作呢？

我们可以首先根据均匀分布生成一个 $[0,1]$ 范围内的随机数 x 。如果 $x\geq\epsilon$，那么我们选择贪婪动作。如果 $x<\epsilon$，那么我们要以 $\frac{1}{|\mathcal{A}(s)|}$ 的概率在 $\mathcal{A}(s)$ 中随机选择一个动作（我们可能会再次选中贪婪动作）。

通过这种方式，选择贪婪动作的总概率为 $1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|}$，而选择任何其他动作的概率为 $\frac{\epsilon}{|\mathcal{A}(s)|}$。

5.4.2 算法描述

为了将 ϵ -贪婪策略整合到 MC 学习中，我们只需要将策略改进步骤从贪婪改进更改为 ϵ -贪婪改进。

具体来说，MC Basic 或 MC Exploring Starts 中的策略改进步骤旨在求解

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

其中 Π 表示*所有可能策略的集合*。我们知道式 (5.4) 的解是一个贪婪策略：

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

其中 $a_k^*=\arg\max_a q_{\pi_k}(s,a)$。

现在，策略改进步骤更改为求解

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

其中 Π_ϵ 表示给定 ϵ 值的*所有 ϵ -贪婪策略的集合*。通过这种方式，我们强制策略为 ϵ -贪婪策略。式 (5.5) 的解为

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

其中 $a_k^*=\arg\max_a q_{\pi_k}(s,a)$。经过上述更改，我们获得了另一种称为 *MC ϵ -Greedy*的算法。该算法的详细信息在算法 5.3 中给出。在这里，采用每次访问 (every-visit) 策略以更好地利用样本。

> **算法 5.3：MC ϵ -贪婪（MC Exploring Starts 的一种变体）**
>
> 初始化：初始策略 $\pi_0(a|s)$ 和初始价值 $q(s,a)$ 对所有 (s,a) 已知。对于所有 (s,a)，设 $\mathrm{Returns}(s,a)=0$ 且 $\mathrm{Num}(s,a)=0$。$\epsilon\in(0,1]$。
>
> 目标：搜索最优策略。
>
> 对于每一个episode，做
>
> episode生成：选择一个起始状态-动作对 (s_0,a_0)（不需要探索起始条件）。遵循当前策略，生成一个长度为 T 的episode：$s_0,a_0,r_1,\ldots,s_{T-1},a_{T-1},r_T$。
>
> 初始化每个episode：$g\leftarrow 0$
>
> 对于episode中的每一步，$t=T-1,T-2,\ldots,0$，做
>
> $g\leftarrow\gamma g+r_{t+1}$
>
> $\mathrm{Returns}(s_t,a_t)\leftarrow\mathrm{Returns}(s_t,a_t)+g$
>
> $\mathrm{Num}(s_t,a_t)\leftarrow\mathrm{Num}(s_t,a_t)+1$
>
> 策略评估：
>
> $q(s_t,a_t)\leftarrow\mathrm{Returns}(s_t,a_t)/\mathrm{Num}(s_t,a_t)$
>
> 策略改进：
>
> 令 $a^*=\arg\max_a q(s_t,a)$ 且
>
> $$\pi(a|s_t)=\begin{cases}1-\frac{|\mathcal{A}(s_t)|-1}{|\mathcal{A}(s_t)|}\epsilon, & a=a^*\\ \frac{1}{|\mathcal{A}(s_t)|}\epsilon, & a\neq a^*\end{cases}$$

如果在策略改进步骤中将贪婪策略替换为 ϵ -贪婪策略，我们还能保证获得最优策略吗？

答案既是肯定的也是否定的。说肯定是意味着，当给定足够多的样本时，算法可以收敛到在集合 Π_ϵ 中最优的 ϵ -贪婪策略。说否定是意味着，该策略仅仅在 Π_ϵ 中是最优的，但在 Π 中可能不是最优的。然而，如果 ϵ 足够小，Π_ϵ 中的最优策略与 Π 中的最优策略非常接近。

5.4.3 说明性示例 (Illustrative examples)

考虑图 5.5 所示的网格世界示例。目标是为每个状态找到最优策略。在 MC ϵ -贪婪算法的每次迭代中，生成一个包含一百万步的单一episode (single episode)。在这里，我们特意考虑仅仅只有一个单独episode的极端情况。我们设定 $r_{\text{boundary}} = r_{\text{forbidden}} = -1$，$r_{\text{target}} = 1$，以及 $\gamma = 0.9$。

初始策略是一个均匀策略，即采取任何动作的概率相同，均为 0.2，如图 5.5 所示。$\epsilon = 0.5$ 的最优 ϵ -贪婪策略可以在两次迭代后获得。虽然每次迭代仅使用一个episode，但策略会逐渐改进，因为所有的状态-动作对都能被访问到，因此它们的值可以被准确地估计。

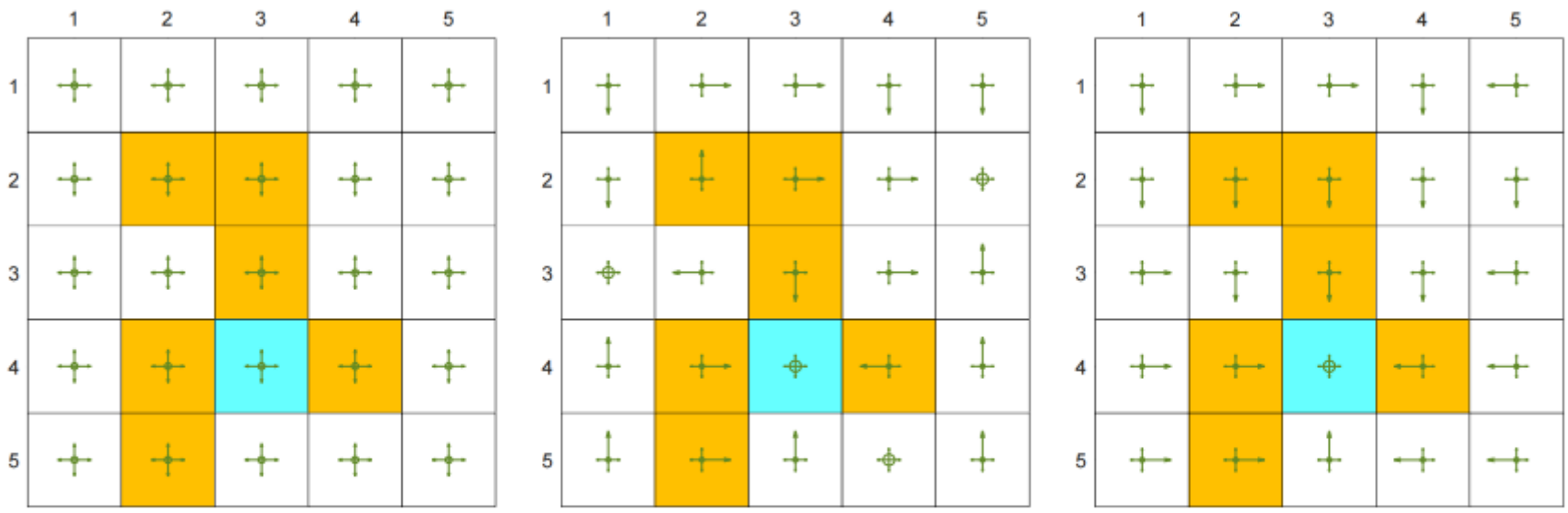


Figure 5.5: The evolution process of the MC ϵ-Greedy algorithm based on single episodes.

图 5.5：基于single episodes 的 MC ϵ -贪婪算法的演化过程。

5.5 ϵ -贪婪策略的探索与利用 (Exploration and exploitation of ϵ -greedy policies)

探索 (Exploration) 和 *利用 (exploitation)* 构成了强化学习中的一个基本权衡。在这里，探索意味着策略可能采取尽可能多的动作。通过这种方式，所有的动作都可以被访问并得到良好的评估。利用意味着改进后的策略应该采取具有最大动作价值的贪婪动作。然而，由于探索不足，当前时刻获得的动作价值可能并不准确，因此我们在进行利用的同时应该保持探索，以避免错过最优动作。

ϵ **-贪婪策略提供了一种平衡探索与利用的方法。**一方面，ϵ -贪婪策略采取贪婪动作的概率较高，因此它可以利用估计出的价值。另一方面，ϵ -贪婪策略也有机会采取其他动作，从而保持探索。ϵ -贪婪策略不仅用于基于 MC 的强化学习，还用于其他强化学习算法，**例如第 7 章介绍的时序差分学习 (temporal-difference learning)**。

利用 与 *最优性 (optimality)* 有关，因为最优策略应该是贪婪的。ϵ **-贪婪策略的基本思想是通过牺牲最优性/利用来增强探索。**如果我们想要增强利用和最优性，**我们需要减小**ϵ **的值。**然而，如果我们想要增强探索，我们需要增大 ϵ 的值。

接下来我们将基于一些有趣的例子来讨论这种权衡。这里的强化学习任务是一个 5×5 的网格世界。奖励设置如下：$r_{\text{boundary}} = -1$，$r_{\text{forbidden}} = -10$，以及 $r_{\text{target}} = 1$。折扣率为 $\gamma = 0.9$。

ϵ -贪婪策略的最优性 (Optimality of ϵ -greedy policies)

接下来我们要展示，**当 ϵ 增加时，ϵ -贪婪策略的最优性会变差。**

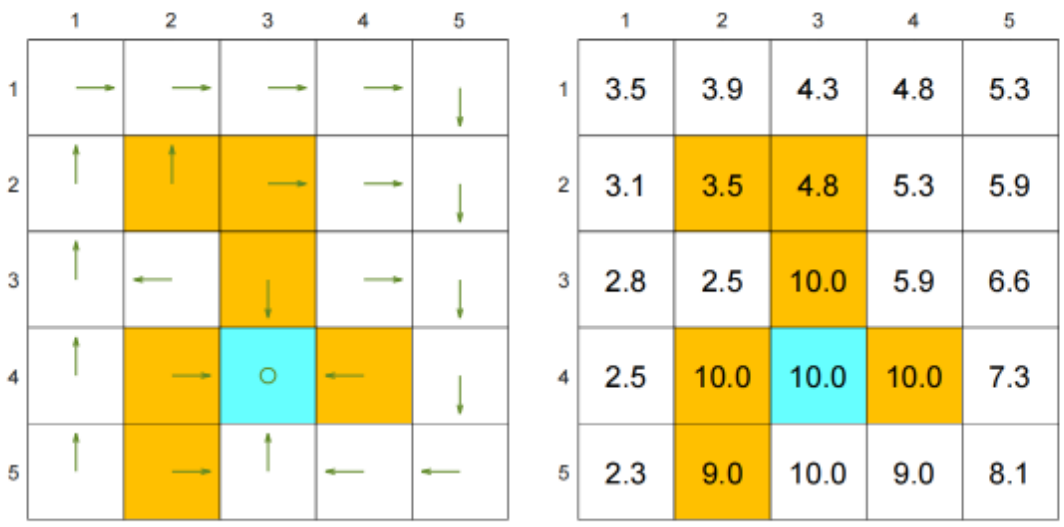
- 首先，**图 5.6(a) 展示了一个贪婪最优策略及其对应的最优状态价值。**一些一致的 ϵ -贪婪策略的状态价值展示在

图 5.6(b)-(d) 中给出。在这里，如果两个 ϵ -贪婪策略中具有最大概率的动作是相同的，则称这两个ϵ -贪婪策略是 *一致的 (consistent)*。

随着 ϵ 值的增加，ϵ -贪婪策略的状态价值减小，表明这些 ϵ -贪婪策略的最优性变差。值得注意的是，当 ϵ 大到 0.5 时，目标状态的价值变得最小。**这是因为，当 ϵ 很大时，从目标区域出发的智能体可能会进入周围的禁区，从而以较高的概率获得负奖励。**

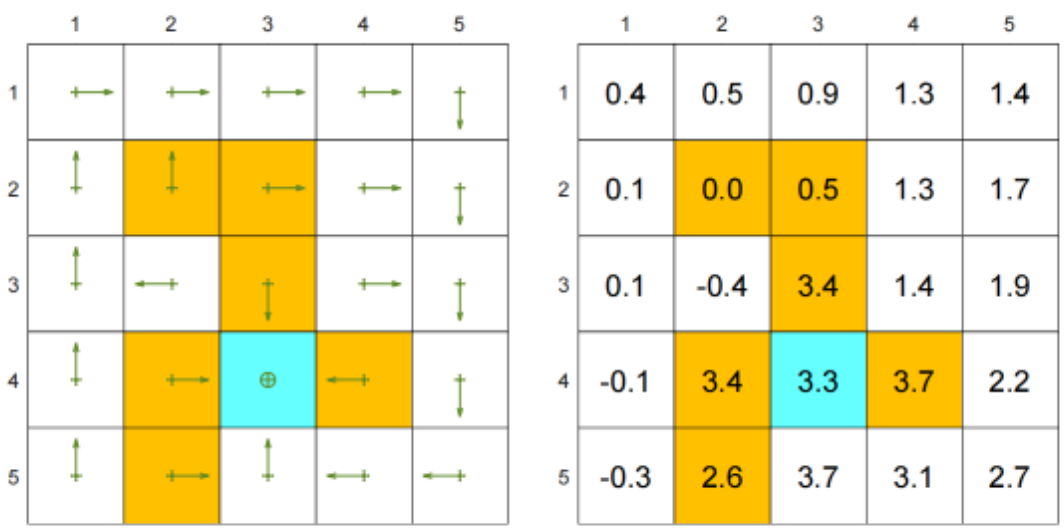
- 第二，图 5.7 展示了最优ϵ -贪婪策略（它们在 Π_ϵ 中是最优的）。当 $\epsilon = 0$ 时，策略是贪婪的，并且在所有策略中是最优的。当ϵ 小至 0.1 时，最优 ϵ -贪婪策略与最优贪婪策略是一致的。然而，当 ϵ 增加到例如 0.2 时，获得的ϵ -贪婪策略与最优贪婪策略不一致。因此，如果我们想要获得与最优贪婪策略一致的 ϵ -贪婪策略，ϵ 的值应该足够小。

为什么当 ϵ 较大时，ϵ -贪婪策略与最优贪婪策略不一致？我们可以通过考虑目标状态来回答这个问题。在贪婪的情况下，目标状态处的最优策略是保持静止以获得正奖励。然而，当 ϵ 较大时，有很高的概率进入禁区并收到负奖励。因此，在这种情况下，目标状态处的最优策略是逃离而不是保持静止。



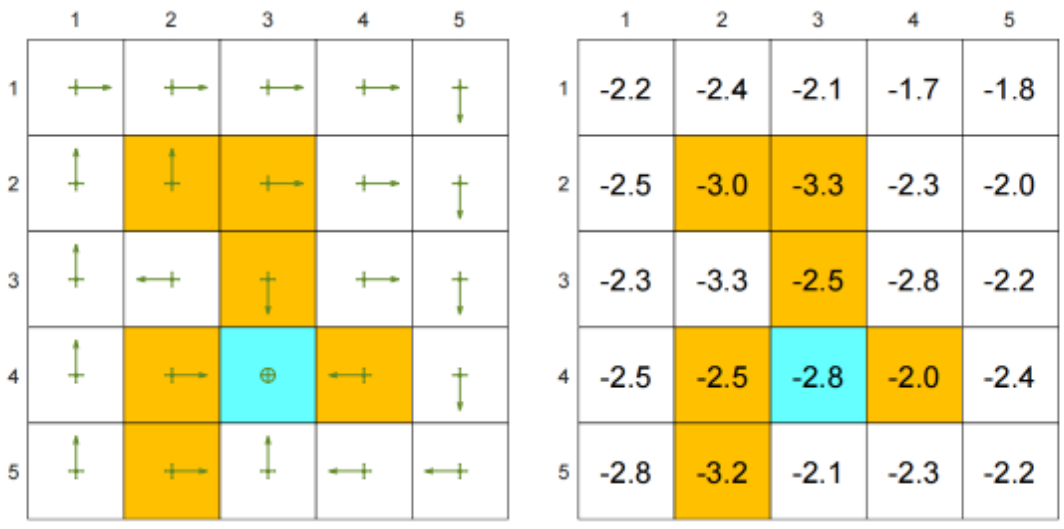
	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

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



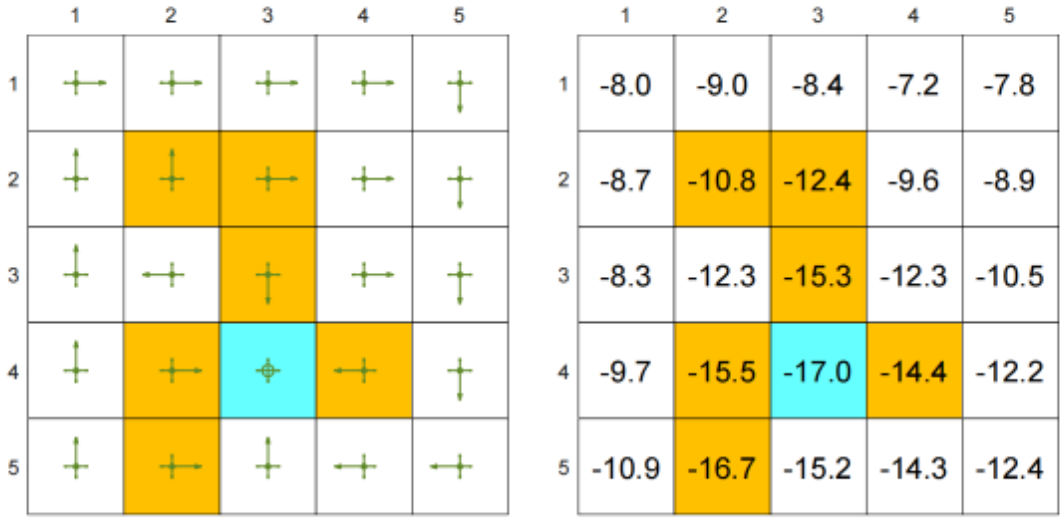
	1	2	3	4	5
1	0.4	0.5	0.9	1.3	1.4
2	0.1	0.0	0.5	1.3	1.7
3	0.1	-0.4	3.4	1.4	1.9
4	-0.1	3.4	3.3	3.7	2.2
5	-0.3	2.6	3.7	3.1	2.7

(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

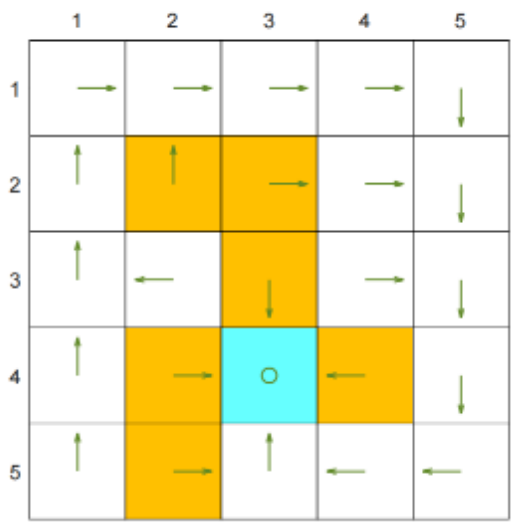
(c) A given ϵ-greedy policy and its state values: $\epsilon = 0.2$



	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

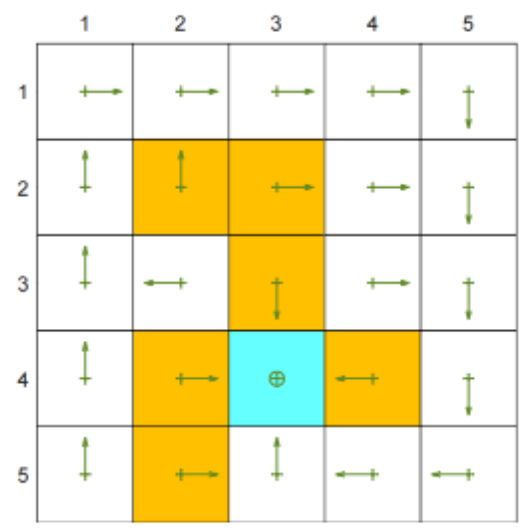
(d) A given ϵ-greedy policy and its state values: $\epsilon = 0.5$

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.



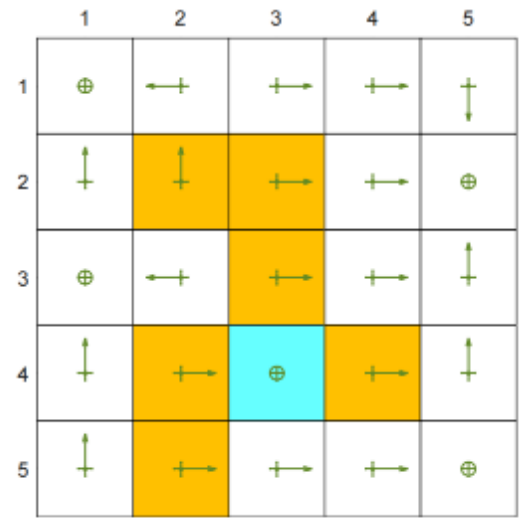
	1	2	3	4	5
1	3.5	3.9	4.3	4.8	5.3
2	3.1	3.5	4.8	5.3	5.9
3	2.8	2.5	10.0	5.9	6.6
4	2.5	10.0	10.0	10.0	7.3
5	2.3	9.0	10.0	9.0	8.1

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



	1	2	3	4	5
1	-1.1	-1.5	-1.1	-0.6	-0.6
2	-1.5	-2.2	-2.3	-1.0	-0.6
3	-1.2	-2.4	-2.2	-1.5	-0.6
4	-1.6	-2.3	-2.6	-1.4	-1.1
5	-2.0	-3.0	-1.8	-1.4	-1.0

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

	1	2	3	4	5
1	-4.3	-5.5	-4.5	-2.6	-2.3
2	-5.6	-7.7	-7.7	-4.1	-2.4
3	-5.5	-9.0	-8.0	-5.6	-2.8
4	-6.8	-8.9	-9.4	-5.5	-4.2

(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).

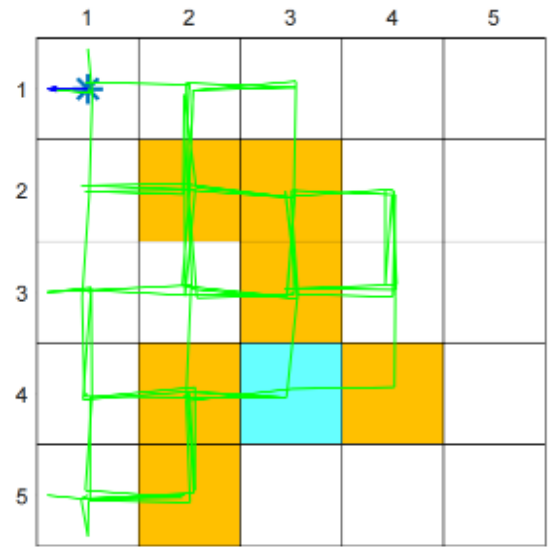
ϵ -贪婪策略的探索能力 (Exploration abilities of ϵ -greedy policies)

接下来我们将说明，当 ϵ 较大时， ϵ -贪婪策略的探索能力很强。

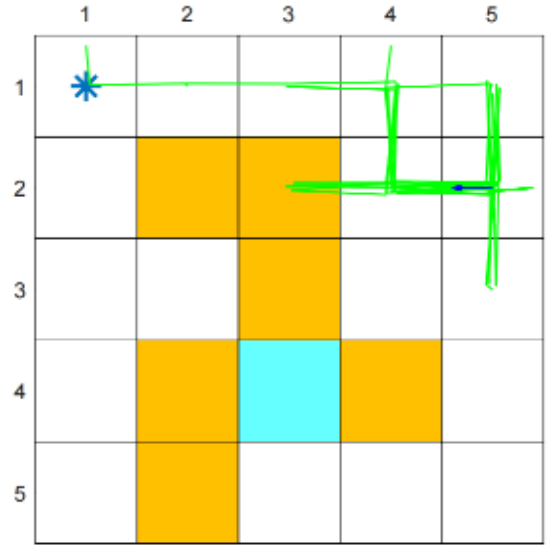
首先，考虑一个 $\epsilon = 1$ 的 ϵ -贪婪策略（见图 5.5(a)）。在这种情况下， ϵ -贪婪策略的探索能力很强，因为它在任何状态下采取任何动作的概率都是 0.2 。从 (s_1, a_1) 开始，由 ϵ -策略生成的episode在图 5.8(a)-(c) 中给出。可以看出，由于策略具有很强的探索能力，当episode足够长时，这个单一的episode可以多次访问所有的状态-动作对。此外，所有状态-动作对被访问的次数几乎是均匀的，如图 5.8(d) 所示。

其次，考虑一个 $\epsilon = 0.5$ 的 ϵ -策略（见图 5.6(d)）。在这种情况下， ϵ -贪婪策略的探索能力比 $\epsilon = 1$ 的情况要弱。从 (s_1, a_1) 开始，由 ϵ -策略生成的episode在图 5.8(e)-(g) 中给出。**虽然当episode足够长时，每个动作仍然可以被访问到，但访问次数的分布可能会非常不均匀。例如，给定一个包含一百万步的episode，有些动作被访问了超过 250,000 次，而大多数动作仅被访问了数百次甚至数十次，如图 5.8(h) 所示。**

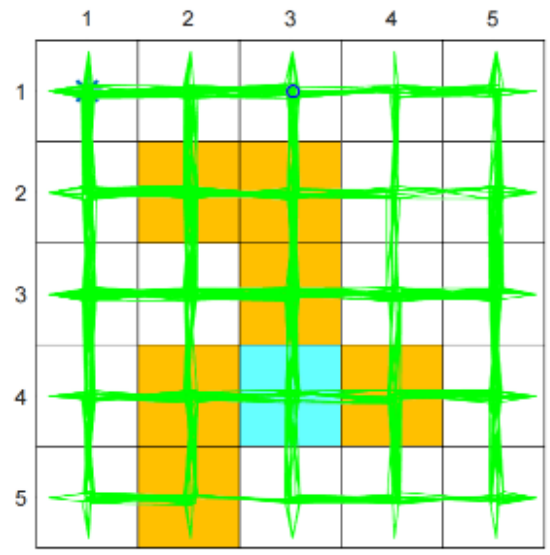
上述例子表明，当 ϵ 减小时， ϵ -贪婪策略的探索能力会下降。**一种有用的技术是初始时将 ϵ 设得较大以增强探索，并逐渐减小它以确保最终策略的最优性 [21–23]。**



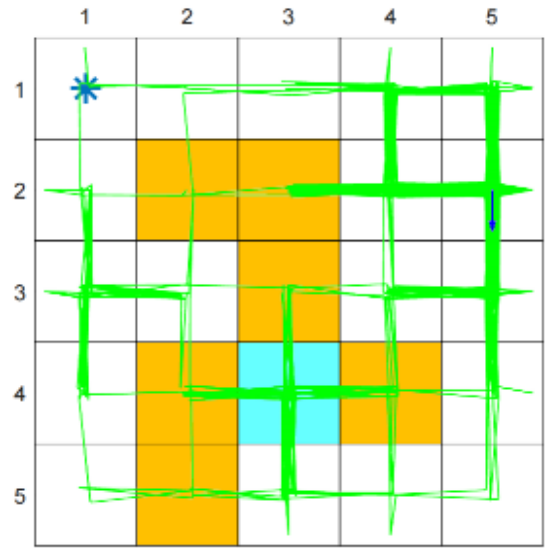
(a) $\epsilon = 1$, trajectory of 100 steps



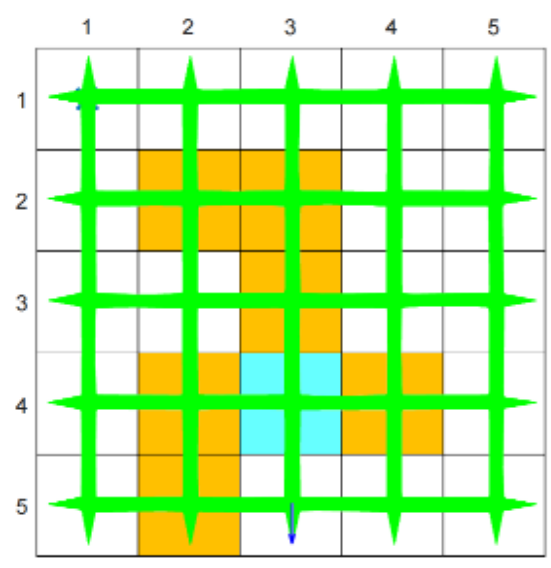
(e) $\epsilon = 0.5$, trajectory of 100 steps



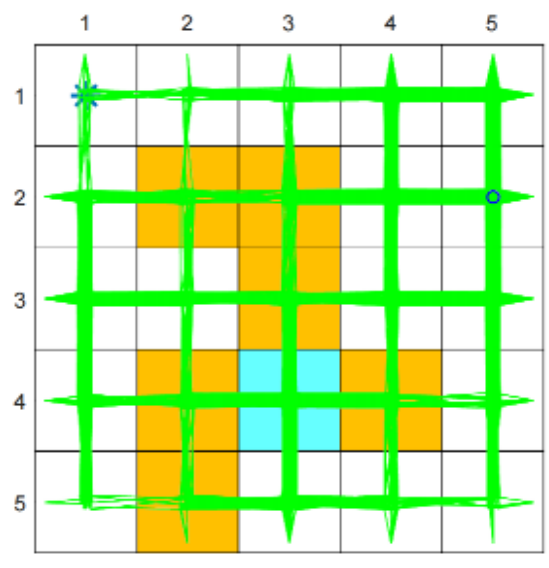
(b) $\epsilon = 1$, trajectory of 1,000 steps



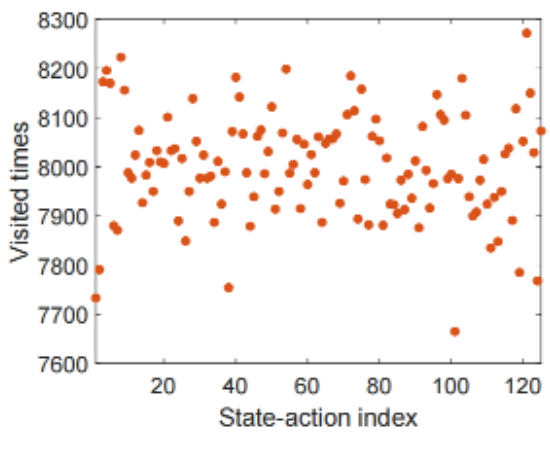
(f) $\epsilon = 0.5$, trajectory of 1,000 steps



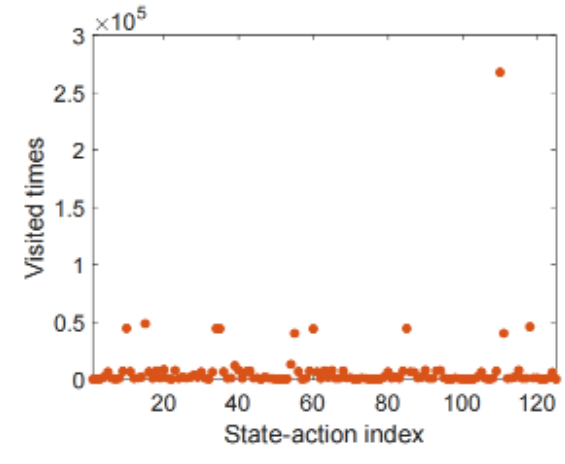
(c) $\epsilon = 1$, trajectory of 10,000 steps



(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.

5.6 总结 (Summary)

本章介绍的算法是本书中首次介绍的无模型强化学习算法。我们首先通过考察一个重要的均值估计问题介绍了 MC 估计的思想。然后，介绍了三种基于 MC 的算法。

- MC Basic：这是最简单的基于 MC 的强化学习算法。该算法是通过将策略迭代算法中的基于模型的策略评估步骤替换为无模型的基于 MC 的估计组件而获得的。给定足够的样本，可以保证该算法收敛到最优策略和最优状态价值。
- MC Exploring Starts（MC 探索起始）：该算法是 MC Basic 的一种变体。它可以从 MC Basic 算法中获得，通过使用首次访问 (first-visit) 或每次访问 (every-visit) 策略来更有效地利用样本。
- MC ϵ -Greedy（MC ϵ -贪婪）：该算法是 MC Exploring Starts 的一种变体。具体来说，在策略改进步骤中，它搜索最佳的 ϵ -贪婪策略而不是贪婪策略。通过这种方式，策略的探索能力得到增强，因此可以移除探索起始的条件。

最后，通过考察 ϵ -贪婪策略的性质，介绍了探索与利用之间的权衡。随着 ϵ 值的增加，ϵ -贪婪策略的探索能力增加，而对贪婪动作的利用减少。另一方面，如果 ϵ 值减小，我们可以更好地利用贪婪动作，但探索能力会受到损害。

5.7 问与答 (Q&A)

- 问：什么是蒙特卡洛估计 (Monte Carlo estimation)?

答： 蒙特卡洛估计指的是一类广泛的技术，它们使用随机样本来求解近似问题。

- 问：什么是均值估计 (mean estimation) 问题?

答： 均值估计问题指的是基于随机样本计算随机变量的期望值。

- 问：如何求解均值估计问题?

答： 有两种方法：基于模型的和无模型的。具体来说，如果随机变量的概率分布已知，则可以计算期望值...

根据其定义计算。如果概率分布未知，我们可以使用蒙特卡洛估计来近似期望值。当样本数量很大时，这种近似是准确的。

- 问：为什么均值估计问题对强化学习很重要?

答： 状态价值和动作价值都被定义为回报的期望值。因此，估计状态或动作价值本质上是一个均值估计问题。

- 问：无模型基于蒙特卡洛 (MC) 的强化学习的核心思想是什么?

答： 核心思想是将策略迭代算法转换为无模型的形式。具体来说，虽然策略迭代算法旨在基于系统模型计算价值，但基于 MC 的强化学习用无模型的基于 MC 的策略评估步骤替换了策略迭代算法中的基于模型的策略评估步骤。

- 问：什么是初始访问 (initial-visit)、首次访问 (first-visit) 和每次访问 (every-visit) 策略?

答： 它们是利用episode中样本的不同策略。一个幕可能会访问许多状态-动作对。初始访问策略使用整个幕来估计初始状态-动作对的动作价值。每次访问和首次访问策略可以更好地利用给定的样本。如果episode的剩余部分用于在每次访问某个状态-动作对时估计其动作价值，则这种策略称为每次访问。如果我们只计算episode中第一次访问某个状态-动作对的情况，则这种策略称为首次访问。

- 问：什么是探索起始 (exploring starts)？为什么它很重要?

答： 探索起始要求从每一个状态-动作对开始生成无限数量（或足够多）的episode。从理论上讲，探索起始条件对于找到最优策略是必要的。也就是说，只有当每一个动作价值都得到充分探索时，我们才能准确地评估所有动作，进而正确地选择最优动作。

- 问：避免探索起始的思路是什么?

答： 基本思想是使策略变软 (soft)。软策略是随机的，使得一个episode能够访问许多状态-动作对。通过这种方式，我们不需要从每一个状态-动作对开始生成大量的episode。

- 问：ϵ -贪婪策略能是最优的吗?
- 答： 答案既是肯定的也是否定的。说肯定是指，如果给定足够的样本，MC ϵ -贪婪算法可以收敛到一个最优的 ϵ -贪婪策略。说否定是指，收敛后的策略仅仅在所有 ϵ -贪婪策略（具有相同的 ϵ 值）中是最优的。
- **问：是否可能使用一个episode来访问所有的状态-动作对?**

答： 是的，这是可能的。只要策略是软的（例如 ϵ -贪婪）且episode足够长。

- 问：MC Basic、MC Exploring Starts 和 MC ϵ -Greedy 之间有什么关系?

答： MC Basic 是最简单的基于 MC 的强化学习算法。它之所以重要，是因为它揭示了无模型基于 MC 的强化学习的基本思想。MC Exploring Starts 是 MC Basic 的一种变体，它调整了样本使用策略。此外，MC ϵ -Greedy 是 MC Exploring Starts 的一种变体，它移除了探索起始的要求。因此，虽然基本思想很简单，但当我们想要获得更好的性能时，复杂性就出现了。将核心思想与可能干扰初学者的复杂细节区分开来是非常重要的。