

第五章 蒙特卡罗方法

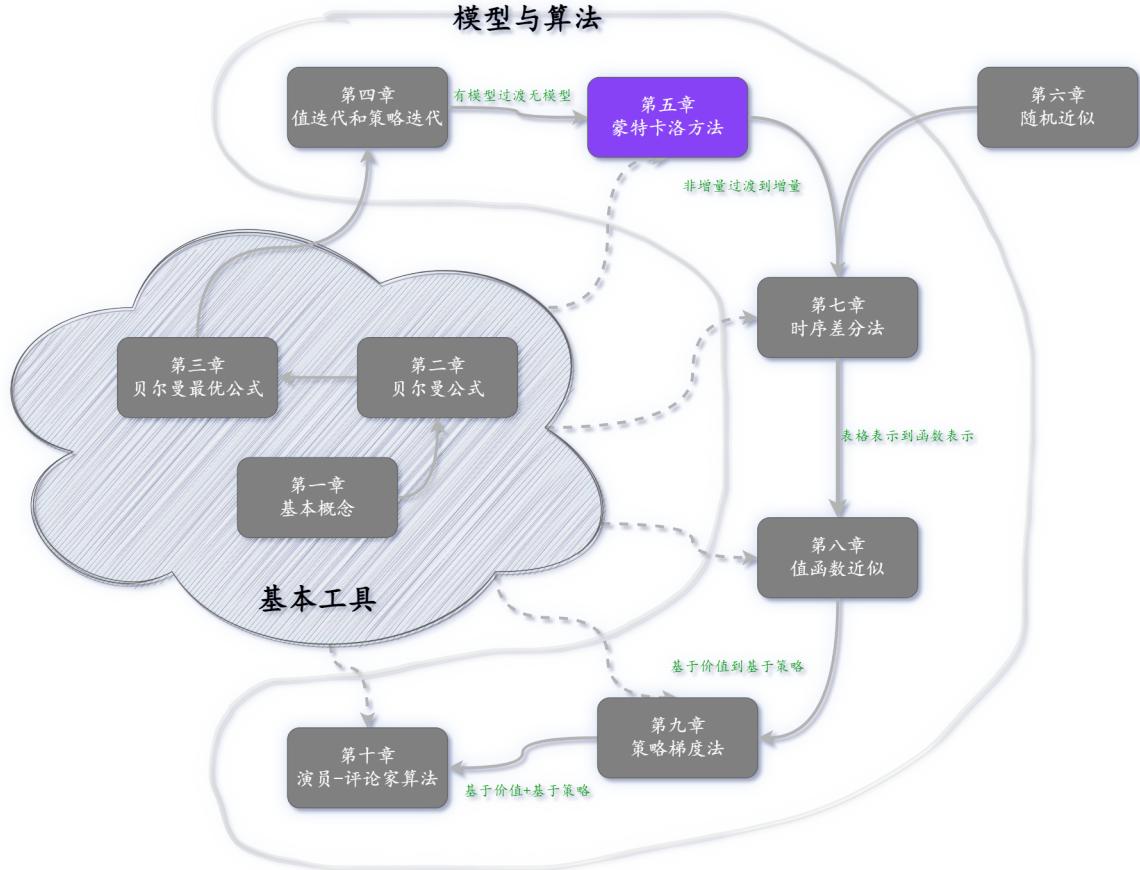


图 5.1

在上一章中，我们介绍了可以根据系统模型找到最优策略的算法。在本章中，我们开始介绍不假定系统模型的无模型（Model-free）强化学习算法。

虽然这是我们第一次在本书中引入无模型算法，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？这个理念很简单：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，我们就必须有模型。如果我们两者都没有，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的交互经验。

为了演示如何从数据而不是模型中学习，我们在本章开始时介绍均值估计（Mean Estimation）问题，其中随机变量的期望值是根据一些样本估计的。理解这个问题对于理解从数据中学习（Learning from Data）的基本思想至关重要。

然后，我们介绍了三种基于蒙特卡罗（Monte Carlo，简称 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，可以通过修改上一章中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，我们进一步引入另外两种更复杂但更高效的算法。

5.1 示例：均值估计

接下来我们引入均值估计问题来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基于蒙特卡罗方法来实现，蒙特卡罗方法是指使用随机样本来解决估计问题的一大类技术。读者可能想知道为什么我们关心均值估计问题。这只是因为状态值和动作值都被定义为回报的均值。估计状态值或动作值实际上是一个均值估计问题。

考虑一个随机变量 X ，它可以从表示为 \mathcal{X} 的有限实数集中获取值。假设我们的任务是计算 X 的平均值或期望值： $\mathbb{E}[X]$ 。可以使用两种方法来计算 $\mathbb{E}[X]$ 。

- 方法一是基于模型。这里的模型是指 X 的概率分布。如果模型已知，那么可以根据期望值的定义直接计算均值：

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

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）**。

- 方法二是无模型的。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，平均值可以近似为：

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

当 n 很小时，这种近似可能不准确。然而，随着 n 的增加，近似值变得越来越准确。当 $n \rightarrow \infty$ 时，我们有 $\bar{x} \rightarrow \mathbb{E}[X]$ 。这是由**大数定律（The Law of Large Numbers）**保证的：大量样本的平均值接近期望值。框 5.1 介绍了大数定律。

【5.1】大数定律

对于随机变量 X ，假设 $\{x_i\}_{i=1}^n$ 是某个独立同分布。样品。令 $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ 为样本的平均值。然后，

$$\begin{aligned}\mathbb{E}[\bar{x}] &= \mathbb{E}[X], \\ \text{Var}[\bar{x}] &= \frac{1}{n} \text{Var}[X].\end{aligned}$$

上述两个方程表明 \bar{x} 是 $\mathbb{E}[X]$ 的无偏估计，并且随着 n 增大到无穷大，其方差减小到零。

下面给出证明：

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（即是， $\mathbb{E}[x_i] = \mathbb{E}[X]$ ）。

其次， $\text{Var}[\bar{x}] = \text{Var}[\sum_{i=1}^n x_i/n] = \sum_{i=1}^n \text{Var}[x_i]/n^2 = (n \cdot \text{Var}[X])/n^2 = \text{Var}[X]/n$ ，其中第二个等式因为样本是独立的，而第三个等式是样本同分布的结果（即 $\text{Var}[x_i] = \text{Var}[X]$ ）。

以下示例说明了上述两种方法。考虑一个抛硬币游戏。让随机变量 X 表示硬币落地时显示的是哪一面。 X 有两个可能的值：当显示头部时 $X = 1$ ，当显示尾部时 $X = -1$ 。假设 X 的真实概率分布（即模型）为：

$$p(x = 1) = \frac{1}{2}, \quad p(x = -1) = \frac{1}{2}.$$

如果预先知道概率分布，我们可以直接计算平均值：

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

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

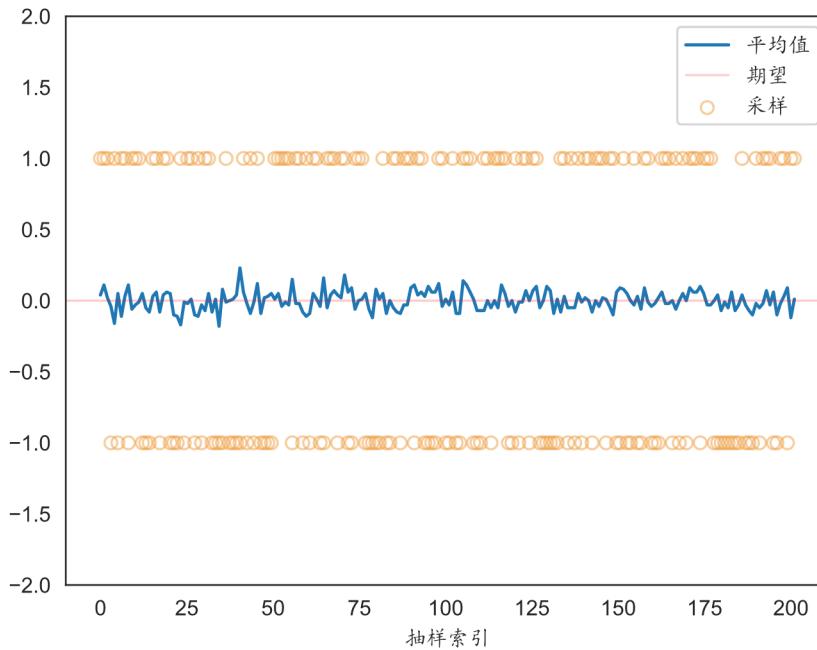


图 5.2

值得一提的是，用于均值估计的样本必须是独立同分布（Independent and Identically Distributed，简写 i.i.d 或 iid）的。否则，如果采样值相关，则可能无法正确估计期望值。一个极端的情况是，无论第一个采样值是什么，所有采样值都与第一个采样值相同。在这种情况下，无论我们使用多少个样本，样本的平均值始终等于第一个样本。

5.2 MC Basic：最简单的基于 MC 的算法

本节介绍第一个也是最简单的基于 MC 的强化学习算法。该算法是通过用无模型的 MC 估计步骤替换 4.2 节中介绍的策略迭代算法中基于模型的策略评估步骤而获得的。

5.2.1 将策略迭代转换为无模型

策略迭代算法的每次迭代都有两个步骤（参见第 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} 。第二步是策略提升，旨在计算贪婪策略 $\pi_{k+1} = \text{argmax}_{\pi} (r_{\pi} + \gamma P_{\pi} v_{\pi_k})$ 。策略提升步骤的元素形式为：

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

必须指出的是，动作价值是这两个步骤的核心。具体来说。第一步，计算状态值以计算动作值。第二步，根据计算出的动作值生成新策略。让我们重新考虑如何计算动作值。有两种方法可供选择。

- 第一种是基于模型的方法。这就是策略迭代算法所采用的方法。具体来说，我们可以首先通过求解贝尔曼方程来计算状态值 v_{π_k} 。然后，我们可以使用以下方法计算动作值：

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

这种方法需要系统模型 $\{p(r|s, a), p(s'|s, a)\}$ 已知。

- 第二种是无模型方法。回想一下，动作值的定义是：

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

这是从 (s, a) 开始时获得的期望回报。由于 $q_{\pi_k}(s, a)$ 是一个期望，因此可以通过 MC 方法进行估计，如第 5.1 节所示。为此，从 (s, a) 开始，智能体可以通过遵循策略 π_k 与环境交互，然后获得一定数量的回合 (Episode)。假设有 n 个回合，第 i 个回合的回报为 $g_{\pi_k}^{(i)}(s, a)$ 。那么， $q_{\pi_k}(s, a)$ 可以近似为：

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

我们已经知道，如果回合数 n 足够大，根据大数定律，近似值将足够准确。

基于 MC 的强化学习的基本思想是使用无模型方法估计动作值，如 (5.2) 所示，以取代策略迭代算法中基于模型的方法。

5.2.2 MC Basic 算法

我们现在准备提出第一个基于 MC 的强化学习算法。从初始策略 π_0 开始，该算法在第 k 次 ($k = 1, 2, \dots$) 迭代中有两个步骤：

- 第一步：策略评估。该步骤用于估计所有 (s, a) 的 $q_{\pi_k}(s, a)$ 。具体来说，对于每个 (s, a) ，我们收集足够多的回合，并使用回报的平均值，表示为 $q_k(s, a)$ ，来近似 $q_{\pi_k}(s, a)$ 。
- 第二步：政策提升。这一步对所有 $s \in \mathcal{S}$ 求解 $\pi_{k+1}(s) = \operatorname{argmax}_{\pi} \sum_a \pi(a|s)q_k(s, a)$ 。贪婪最优策略是 $\pi_{k+1}(a_k^*|s) = 1$ ，其中 $a_k^* = \operatorname{argmax}_a q_k(s, a)$ 。

这是最简单的基于 MC 的强化学习算法，本书中将其称为 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。唯一的区别是，它直接根据经验样本计算动作值，而策略迭代则先计算状态值，然后根据系统模型计算动作值。需要注意的是，无模型算法是直接估计动作值。否则，如果它估计状态值，我们仍然需要使用系统模型从这些状态值计算动作值，如 (5.1) 所示。

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：寻找最优策略

For 第 k , ($k = 0, 1, 2 \dots$) 次迭代, **do**

For 每个状态 $s \in \mathcal{S}$, **do**

For 每个动作 $a \in \mathcal{A}(s)$, **do**

通过以下 π_k 收集从 (s, a) 开始的足够多的回合

策略评估:

$q_{\pi_k}(s, a) \approx q_k(s, a) =$ 从 (s, a) 开始的所有回合的平均回报

策略改进:

$$a_k^*(s) = \operatorname{argmax}_a q_k(s, a)$$

If $a = a_k^*$, $\pi_{k+1}(a|s) = 1$, **Else** $\pi_{k+1}(a|s) = 0$

由于策略迭代是收敛的, 因此当给定足够的样本时, MC Basic 也是收敛的。也就是说, 对于每个 (s, a) , 假设有足够多的从 (s, a) 开始的回合。那么, 这些回合的回报的平均值就可以准确地逼近 (s, a) 的动作值。在实践中, 我们通常不会为每个 (s, a) 提供足够的回合。因此, 动作值的近似值可能不准确。尽管如此, 该算法通常仍然可以工作。这类似于截断策略迭代算法, 其中动作值都没有被准确计算。

最后, MC Basic 由于样本效率低而过于简单而不实用。我们之所以介绍这个算法, 是为了让读者掌握基于 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, 充分理解该算法非常重要。我们将看到通过扩展 MC Basic 算法可以很容易地获得更复杂和样本效率更高的算法。

5.2.3 说明性例子

- 一个简单的例子: 逐步实施 □

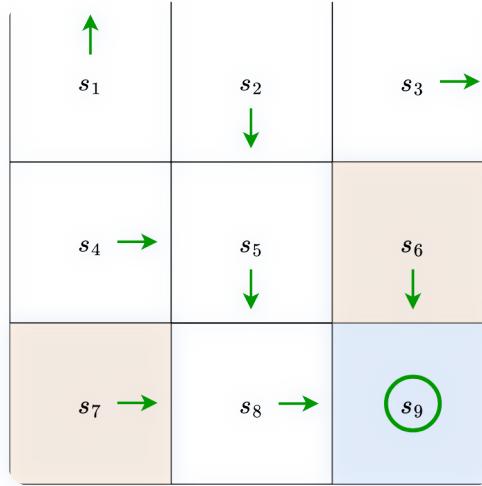


图 5.3

接下来我们通过一个例子来演示 MC Basic 算法的实现细节。奖励设置为 $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ 和 $r_{\text{target}} = 1$ 。折扣因子为 $\gamma = 0.9$ 。初始策略 π_0 如图 5.3 所示。此初始策略对于 s_1 或 s_3 来说并不是最佳的。

虽然所有的动作值都应该被计算，但由于空间限制，我们仅呈现 s_1 的动作值。在 s_1 处，有五种可能的操作。对于每个动作，我们需要收集许多足够长的回合来有效地近似动作值。然而，由于该示例在策略和模型方面都是确定性的，因此多次运行将生成相同的轨迹。因此，每个动作值的估计仅需要单个回合。

继 π_0 之后，我们可以分别从 $(s_1, a_1), (s_1, a_2), \dots, (s_1, a_5)$ 开始得到以下回合：

- 从 (s_1, a_1) 开始，回合为 $s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$ 动作值等于回合的折扣回报：

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

- 从 (s_1, a_2) 开始，回合为 $s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_3} \dots$ 动作值等于回合的折扣回报：

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

- 从 (s_1, a_3) 开始，回合为 $s_1 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_5 \xrightarrow{a_3} \dots$ 动作值等于回合的折扣回报：

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

- 从 (s_1, a_4) 开始，回合为 $s_1 \xrightarrow{a_4} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$ 动作值等于回合的折扣回报：

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

- 从 (s_1, a_5) 开始，回合为 $s_1 \xrightarrow{a_5} s_1 \xrightarrow{a_1} s_1 \xrightarrow{a_1} \dots$ 动作值等于剧集的折扣回报：

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

通过比较五个动作值，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，新策略可得为：

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

直观地说，改进后的策略在 s_1 处采用 a_2 或 a_3 是最优的。因此，对于这个简单的例子，我们只需使用一次迭代就可以成功地获得最优策略。在这个简单的示例中，初始策略对于除 s_1 和 s_3 之外的所有状态都已经是最优的。因此，策略仅经过一次迭代就可以变得最优。当策略对于其他状态不是最优时，需要更多迭代。

- □一个综合示例：回合长度和稀疏奖励 □

接下来，我们通过一个更全面的示例来讨论 MC Basic 算法的一些有趣性质。图 5.4 的示例是一个 5×5 网格世界。奖励设置为 $r_{\text{boundary}} = -1$ 、 $r_{\text{forbidden}} = -10$ 、 $r_{\text{target}} = 1$ 。折扣因子为 $\gamma = 0.9$ 。



图 5.4

首先，我们证明回合长度（Episode Length）极大地影响最终的最优策略。特别是，图 5.4 显示了 MC Basic 算法在不同回合长度下生成的最终结果。当每回合的长度太短时，策略和价值估计都不是最优的（见图 5.4(a) - (d)）。在回合长度为 1 的极端情况下，只有与目标相邻的状态具有非零值，所有其他状态都具有零值，因为每个回合太短而无法达到目标或获得正奖励（见图 5.4(a)）。随着回合长度的增加，策略和价值估计逐渐接近最优（见图 5.4(h)）。

随着回合长度的增加，出现了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。造成这种现象的原因如下。从一个状态开始，智能体必须至少移动一定数量的步数才能到达目标状态，然后获得正奖励。如果一个回合的长度小于最小期望步数，则返回肯定为零，估计的状态值也为零。在此示例中，情节长度必须不小于 15，这是从左下状态开始时到达目标所需的最小步数。

虽然上述分析表明每个回合必须足够长，但回合不一定是无限长的。如图 5.4(g) 所示，当长度为 30 时，算法可以找到最优策略，尽管值估计还不是最优的。

上述分析涉及一个重要的奖励设计问题 — [稀疏奖励 1 2 3](#) (Sparse Reward)，稀疏奖励是指除非达到目标否则无法获得正奖励的场景。稀疏的奖励设置需要较长的回合才能达到目标。当状态空间很大时，这个要求很难满足。结果，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，在上面的网格世界示例中，我们可以重新设计奖励设置，以便智能体在达到目标附近的状态时可以获得较小的正奖励。这样，就可以在目标周围形成一个“吸引力场”，使智能体更容易找到目标。

5.3 MC 探索开始

接下来我们扩展 MC Basic 算法以获得另一种基于 MC 的强化学习算法，该算法稍微复杂但样本效率更高。

5.3.1 更有效地利用样本

基于 MC 的强化学习一个重要方面是如何更有效地使用样本。具体来说，假设我们有一个通过遵循策略 π 获得的样本集：

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

其中下标指的是状态或动作索引而不是时间步长。每次状态 - 动作对出现在一个回合中，就称为该状态-动作对的访问（Visit）。可以采用不同的策略来利用访问。

第一个也是最简单的策略是使用[初始访问 \(Initial Visit\)](#)。也就是说，一个回合仅用于估计该回合开始的初始状态 - 动作对的动作值。对于(5.3)中的示例，初始访问策略仅估计 (s_1, a_2) 的动作值。MC Basic 算法采用初始访问策略。然而，这种策略的样本效率不高，因为该回合还访问了许多其他状态 - 动作对，例如 (s_2, a_4) , (s_2, a_3) 和 (s_5, a_1) 。这些访问还可用于估计相应的动作值。特别是，我们可以将(5.3)中的回合分解为多个子回合：

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

状态 - 动作对访问后生成的轨迹可以被视为一个新的回合。这些新的回合可用于估计更多的动作值。这样，可以更有效地利用回合中的样本。

此外，状态 - 动作对可能在一个回合中被多次访问。例如， (s_1, a_2) 在 (5.3) 的回合中被访问两次。如果我们只计算首次访问，这称为首次访问策略（First - Visit）。如果我们计算状态 - 动作对的每次访问，这样的策略称为每次访问⁴（Every - Visit）。

从样本使用效率来看，每次访问策略是最好的。如果一个回合足够长，以至于它可以多次访问所有状态 - 动作对，那么这个单一回合可能足以使用每次访问策略来估计所有动作值。然而，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子集。然而，如果两次访问在轨迹上彼此远离，则相关性不会很强。

5.3.2 更有效地更新策略

基于 MC 的强化学习的另一个方面是何时更新策略。有两种策略可供选择：

- 第一个策略是，在策略评估步骤中，收集从同一状态 - 动作对开始的所有回合，然后使用这些回合的平均回报来近似动作值。MC Basic 算法就采用了这种策略。该策略的缺点是智能体必须等到收集完所有回合后才能更新估计。
- 第二种策略可以克服这个缺点，是使用单个回合的返回值来近似相应的动作值。这样，当我们收到一个回合时，我们就可以立即得到一个粗略的估计。然后，可以逐回合（Episode-by-Episode）地改进策略。

由于单个回合的回报无法准确地近似相应的动作值，人们可能会怀疑第二种策略是否良好。事实上，该策略属于上一章介绍的广义策略迭代（Generalized Policy Iteration）的范围。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 算法说明

我们可以使用 5.3.1 和 5.3.2 节中介绍的技术来提高 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。该算法使用每次访问策略。有趣的是，当计算从每个状态 - 动作对开始获得的折扣回报时，该过程从结束状态开始并返回到起始状态。这些技术可以使算法更加高效，但也使算法更加复杂。这就是为什么首先引入没有这些技术的 MC Basic 算法来揭示基于MC的强化学习的核心思想。

算法 5.2：MC Exploring Starts（MC Basic 的有效变体）

初始化：初始策略 $\pi_0(a|s)$ 和所有 (s, a) 的初始值 $q(s, a)$ 。对于所有 (s, a) ，
 $Returns(s, a) = 0$ 且 $Num(s, a) = 0$

目标：寻找最优策略

对于每个回合，进行

回合生成：

选择一个起始状态 - 动作对 (s_0, a_0) 并确保所有对都可能被选择（这是探索开始条件）

按照当前策略，生成长度为 T 的回合： $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$

每回合初始化： $g \leftarrow 0$

For 回合的每个步骤， $t = T-1, T-2, \dots, 0$ ，**do**

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

$$\text{Returns}(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) + g$$

$$\text{Num}(s_t, a_t) \leftarrow \text{Num}(s_t, a_t) + 1$$

策略评估：

$$q(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) \div \text{Num}(s_t, a_t)$$

策略提升：

$$\text{If } a = \text{argmax}_a q(s_t, a), \quad \pi(a|s_t) = 1, \quad \text{Else } \pi(a|s_t) = 0$$

探索开始条件需要从每个状态 - 动作对开始有足够的回合。只有充分探索每个状态 - 动作对，我们才能准确估计它们的动作值（根据大数定律），从而成功找到最优策略。否则，如果一个动作没有被很好地探索，它的动作值可能会被不准确地估计，并且这个动作可能不会被策略选择，即使它确实是最好的动作。MC Basic 和 MC Exploring Starts 都需要此条件。然而，这个条件在许多应用中很难满足，特别是那些涉及与环境物理交互的应用。我们可以取消探索开始的要求吗？答案是肯定的，如下一节所示。

5.4 MC ϵ - 贪婪：不探索就开始学习

接下来，我们通过删除探索开始条件来扩展 MC Exploring Starts 算法。这个条件实际上要求每个状态 - 动作对都可以被访问足够多次，这也可以基于软策略来实现。

5.4.1 ϵ - 贪婪策略

如果一个策略在任何状态下采取任何动作的可能性均为正，那么该策略就是软策略。考虑一个极端的情况，其中我们只有一个回合。使用软策略，足够长的单个回合可以多次访问每个状态 - 动作对（参见图 5.8 中的示例）。因此，我们不需要从不同的状态 - 动作对开始生成大量的回合，然后可以删除探索开始的要求。

一种常见的软政策是 ϵ - 贪婪策略。 ϵ - 贪婪策略是一种随机策略，它有更高的机会选择贪婪行动（Greedy Action），并且采取任何其他行动的概率相同非零。这里，贪婪动作是指动作值最大的动作。特别地，假设 $\epsilon \in [0, 1]$ 。相应的 ϵ - 贪婪策略具有以下形式：

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

其中 $|\mathcal{A}(s)|$ 表示与 s 相关的动作的数量。

当 $\epsilon = 0$ 时， ϵ - 贪婪变为贪婪。当 $\epsilon = 1$ 时，采取任何行动的概率等于 $\frac{\epsilon}{|\mathcal{A}(s)|}$ 。

采取贪婪动作的概率总是大于采取任何其他动作的概率，因为

$$\forall \epsilon \in [0, 1], \quad 1 - \frac{\epsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1) = 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} \geq \frac{\epsilon}{|\mathcal{A}(s)|}$$

虽然 ϵ -贪婪策略是随机的，但我们如何通过遵循这样的策略来选择动作呢？我们首先可以按照均匀分布生成 $[0, 1]$ 中的随机数 x 。如果 $x \geq \epsilon$ ，那么我们选择贪婪动作。如果 $x < \epsilon$ ，那么我们以 $\frac{1}{|\mathcal{A}(s)|}$ 的概率随机选择 $\mathcal{A}(s)$ 中的一个动作（我们可以再次选择贪婪动作）。这样，选择贪婪动作的总概率为 $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$ ，选择任何其他动作的概率为 $\frac{\epsilon}{|\mathcal{A}(s)|}$ 。

5.4.2 算法说明

为了将 ϵ -贪婪策略融入到 MC 学习中，我们只需要将策略提升步骤从贪婪改为 ϵ -贪婪即可。

特别是，MC Basic 或 MC Exploring Starts 中的策略提升步骤旨在解决：

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

其中 Π 表示所有可能策略的集合。我们知道 (5.4) 的解是一个贪心策略：

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

其中 $a_k^* = \operatorname{argmax}_a q_{\pi_k}(s, a)$ 。

现在，政策提升步骤改变为解决：

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

其中 Π_ϵ 表示具有给定值 ϵ 的所有 ϵ -贪婪策略的集合。通过这种方式，我们迫使策略变成 ϵ -贪婪。 (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 $a_k^* = \operatorname{argmax}_a q_{\pi_k}(s, a)$ ，通过上述改变，我们得到了另一种算法，称为 MC ϵ -Greedy。该算法的细节在算法 5.3 中给出。在这里，采用每次访问策略来更好地利用样本。

算法 5.3：MC ϵ -Greedy (MC Exploring Starts 的变体)

初始化：初始策略 $\pi_0(a|s)$ 和所有 (s, a) 的初始值 $q(s, a)$ 。对于所有 (s, a) ，
 $\text{Returns}(s, a) = 0$ 且 $\text{Num}(s, a) = 0$ ， $\epsilon \in [0, 1]$

目标：寻找最优策略

对于每个回合，进行

回合生成：

选择一个起始状态 - 动作对 (s_0, a_0) 并确保所有对都可能被选择（这是探索开始条件）

按照当前策略，生成长度为 T 的回合： $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$

每回合初始化： $g \leftarrow 0$

For 回合的每个步骤， $t = T-1, T-2, \dots, 0$ ，**do**

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

$$\text{Returns}(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) + g$$

$$\text{Num}(s_t, a_t) \leftarrow \text{Num}(s_t, a_t) + 1$$

策略评估：

$$q(s_t, a_t) \leftarrow \text{Returns}(s_t, a_t) \div \text{Num}(s_t, a_t)$$

策略提升：

令 $a_k^* = \operatorname{argmax}_a q_{\pi_k}(s_t, a)$ ，且

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

如果在策略提升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。是的，我们的意思是，当给定足够样本时，算法可以收敛到在集合 Π_ϵ 中最优的 ϵ -贪婪策略。而不是，该策略仅在 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，则 Π_ϵ 中的最优策略会接近 Π 中的最优策略。

5.4.3 示例

考虑图 5.5 所示的网格世界示例。目的是为每个状态找到最佳策略。MC ϵ -贪婪算法的每次迭代都会生成一百万步的单回合。在这里，我们特意考虑只有一个回合的极端情况。我们设置 $r_{\text{boundary}} = r_{\text{forbidden}} = -1$ ， $r_{\text{target}} = 1$ ， $\gamma = 0.9$ 。

初始策略是统一策略，采取任何动作的概率相同为 0.2，如图 5.5 所示。经过两次迭代即可得到 $\epsilon = 0.5$ 的最优 ϵ -贪婪策略。尽管每次迭代仅使用单个回合，但策略逐渐改进，因为所有状态 - 动作对都可以访问，因此可以准确估计它们的值。

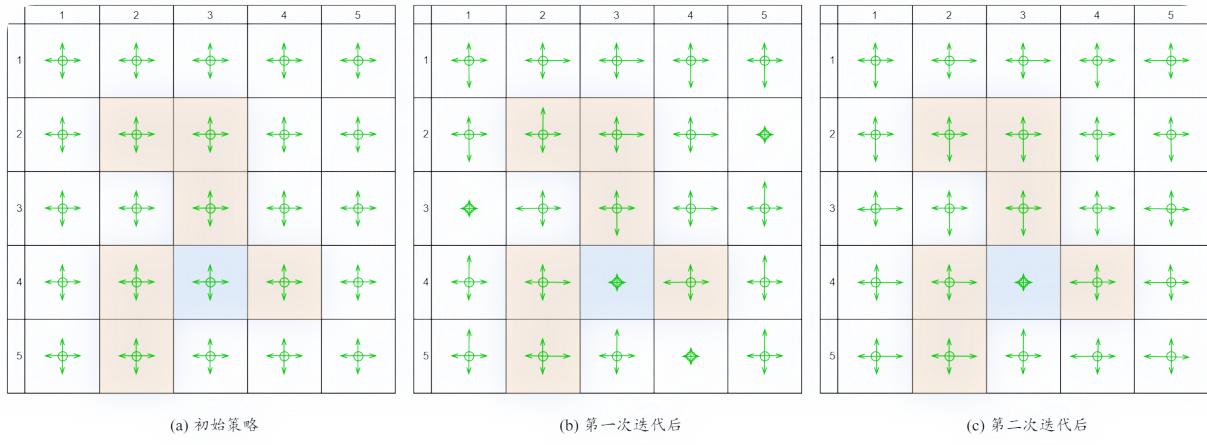


图 5.5

5.5 ϵ - 贪婪策略的探索和利用

探索（Exploration） 和 **利用（Exploitation）** 构成了强化学习的基本权衡。在这里，探索意味着策略可能采取尽可能多的动作。这样，所有的动作都可以被很好地访问和评估。利用意味着改进后的策略应该采取行动价值最大的贪婪行动。然而，由于探索不充分，当前时刻获得的动作值可能不准确，因此我们应该在进行探索的同时不断探索，以避免错过最佳动作。

ϵ - 贪婪策略提供了一种平衡探索和利用的方法。一方面， ϵ - 贪婪策略有更高的概率采取贪婪动作，从而可以利用估计值。另一方面， ϵ - 贪婪策略也有机会采取其他行动，以便继续探索。 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 章介绍的时间差分学习。

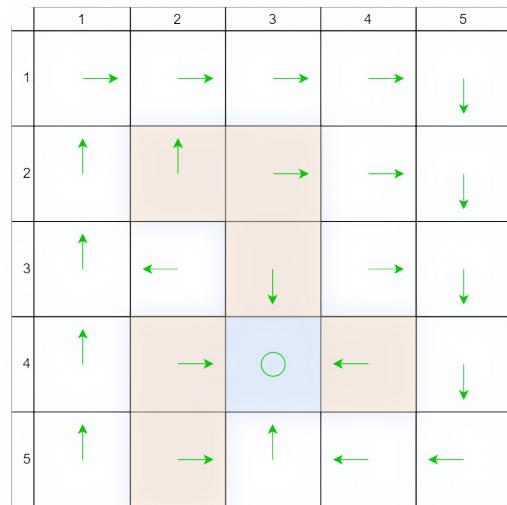
利用与最优性相关，因为最优策略应该是贪婪的。 ϵ - 贪婪策略的基本思想是通过牺牲最优性来增强探索。如果我们想增强利用和最优性，就需要降低 ϵ 的值。然而，如果我们想加强探索，我们需要增加 ϵ 的值。

接下来我们将根据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。奖励设置为 $r_{\text{boundary}} = -1$ 、 $r_{\text{forbidden}} = -10$ 、 $r_{\text{target}} = 1$ 。折扣率为 $\gamma = 0.9$ 。

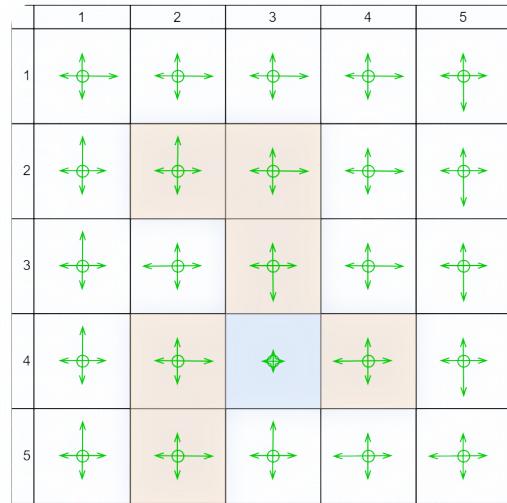
- ϵ - 贪婪策略的最优性

接下来我们证明，当 ϵ 增加时， ϵ - 贪婪策略的最优性会变得更差。

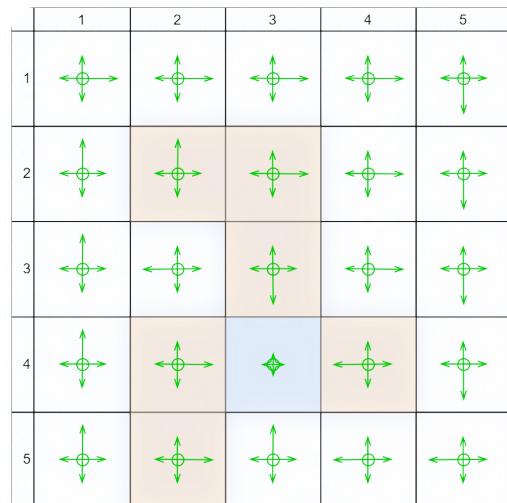
- 首先，贪婪最优策略和相应的最优状态值如图 5.6(a) 所示。一些一致的 ϵ - 贪婪策略的状态值如图 5.6(b) - (d) 所示。这里，如果策略中概率最大的行为相同，则两个 ϵ - 贪婪策略是一致的。



	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) 给定的 ε - 贪婪策略及其状态值: $\varepsilon = 0$ 

	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) 给定的 ε - 贪婪策略及其状态值: $\varepsilon = 0.1$ 

	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) 给定的 ε - 贪婪策略及其状态值: $\varepsilon = 0.2$

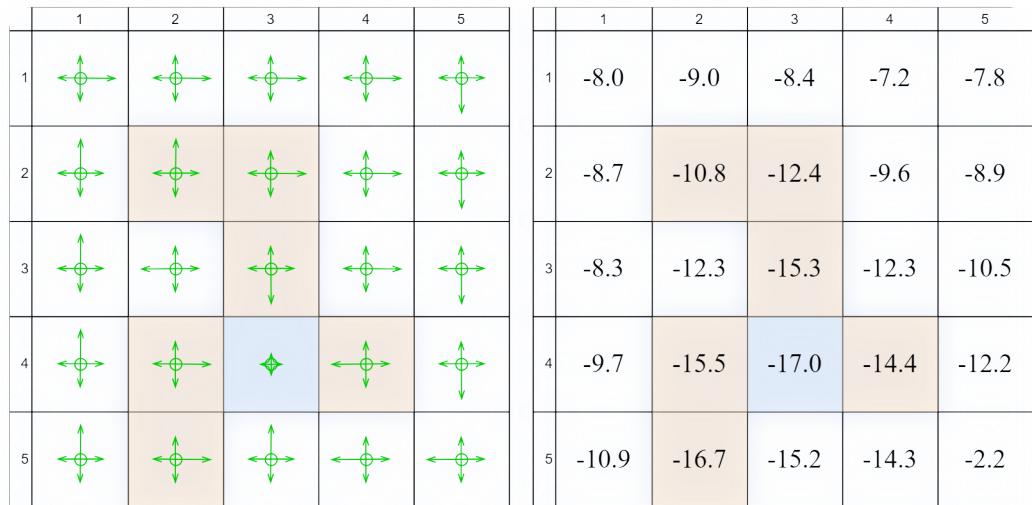
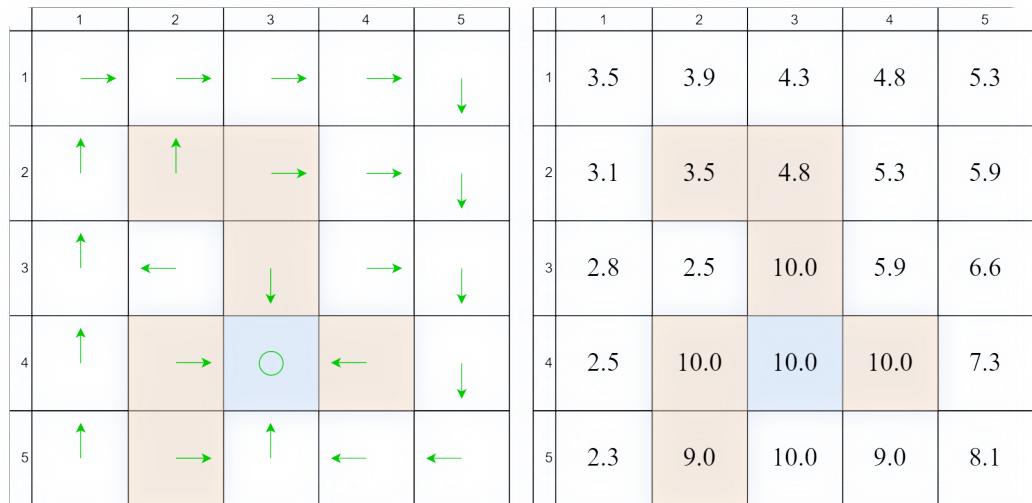
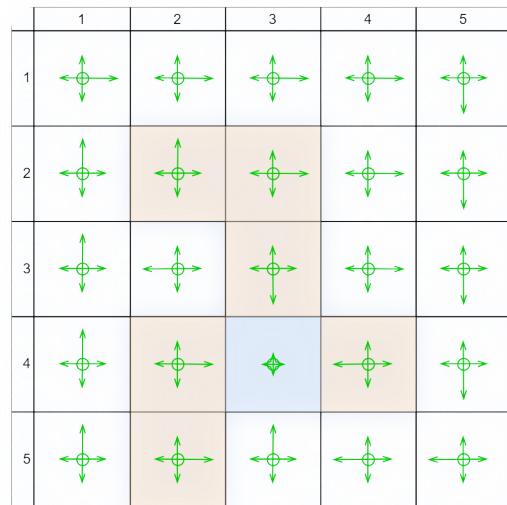
(d) 给定的 ϵ - 贪婪策略及其状态值: $\epsilon = 0.5$

图 5.6

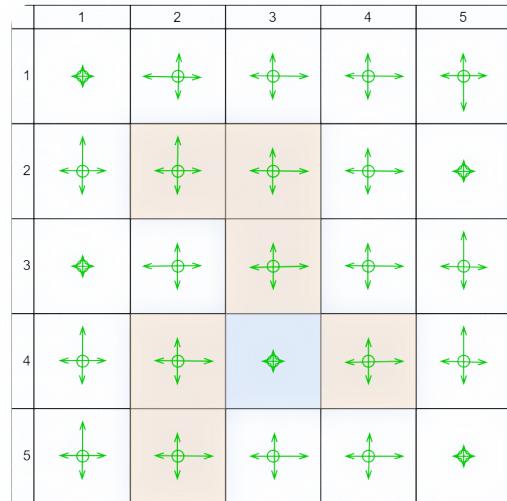
随着 ϵ 值的增加, ϵ - 贪婪策略的状态值下降, 表明这些 ϵ - 贪婪策略的最优性变差。值得注意的是, 当 ϵ 大到 0.5 时, 目标状态的值变得最小。这是因为, 当 ϵ 较大时, 从目标区域出发的智能体可能会进入周围的禁区, 从而以更高的概率获得负奖励。

- 其次, 图 5.7 显示了最优的 ϵ - 贪婪策略 (它们在 Π_ϵ 中是最优的)。当 $\epsilon = 0$ 时, 该策略是贪婪的, 并且是所有策略中最优的。当 ϵ 小至 0.1 时, 最优 ϵ - 贪婪策略与最优贪婪策略一致。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并不一致。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。

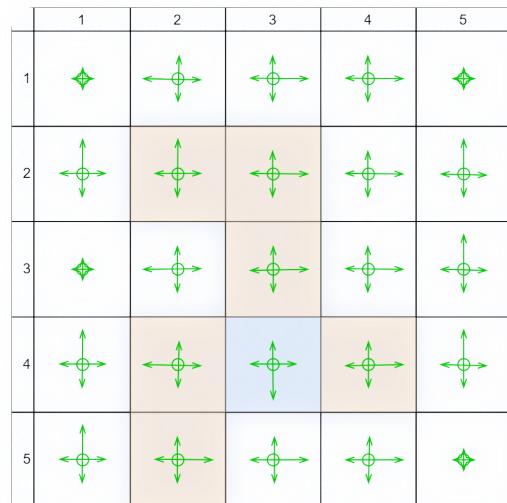
(a) 给定的 ϵ - 贪婪策略及其状态值: $\epsilon = 0$



	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) 给定的 ε - 贪婪策略及其状态值: $\varepsilon = 0.1$ 

	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) 给定的 ε - 贪婪策略及其状态值: $\varepsilon = 0.2$ 

	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
5	-7.9	-10.1	-6.7	-5.1	-3.7

(d) 给定的 ε - 贪婪策略及其状态值: $\varepsilon = 0.5$

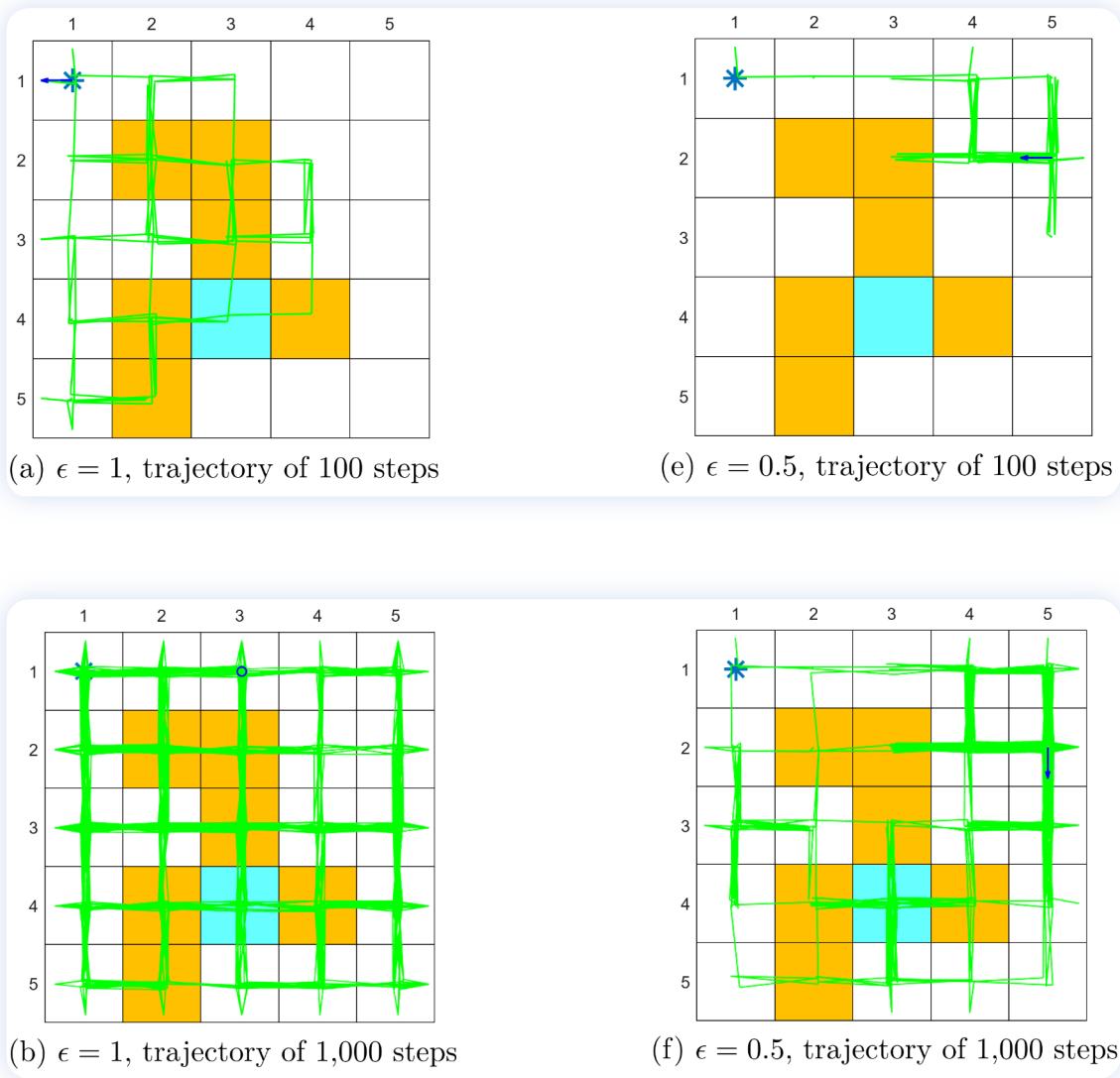
图 5.7

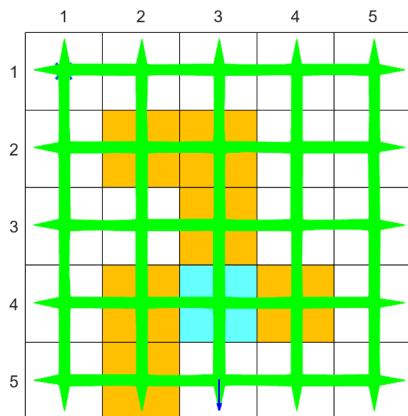
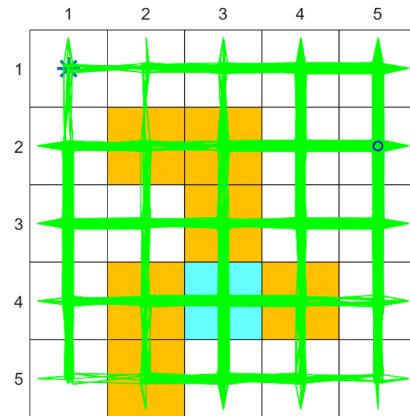
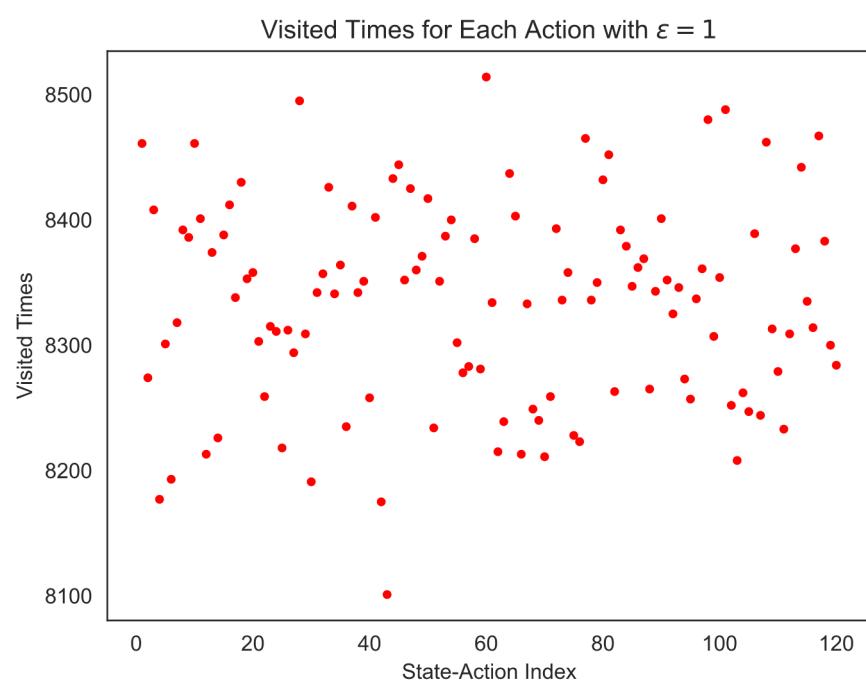
为什么当 ϵ 很大时， ϵ -贪婪策略与最优贪婪策略不一致？我们可以通过考虑目标状态来回答这个问题。在贪婪的情况下，目标状态下的最优策略是保持不变以获得正奖励。然而，当 ϵ 很大时，进入禁区并获得负奖励的几率就很高。因此，这种情况下目标状态的最优策略是逃离而不是保持不变。

- ϵ -贪婪策略的探索能力

接下来我们说明，当 ϵ 大时， ϵ -贪婪策略的探索能力很强。

首先，考虑 $\epsilon = 1$ 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行动的概率都是 0.2。从 (s_1, a_1) 开始，由 ϵ -策略生成的一段回合如图 5.8 (a) - (c) 所示。可以看出，由于策略的探索能力很强，当回合足够长时，这一单个回合可以多次访问所有状态动作对。而且，所有状态 - 动作对被访问的次数几乎是偶数，如图 5.8(d) 所示。



(c) $\epsilon = 1$, trajectory of 10,000 steps(g) $\epsilon = 0.5$, trajectory of 10,000 steps

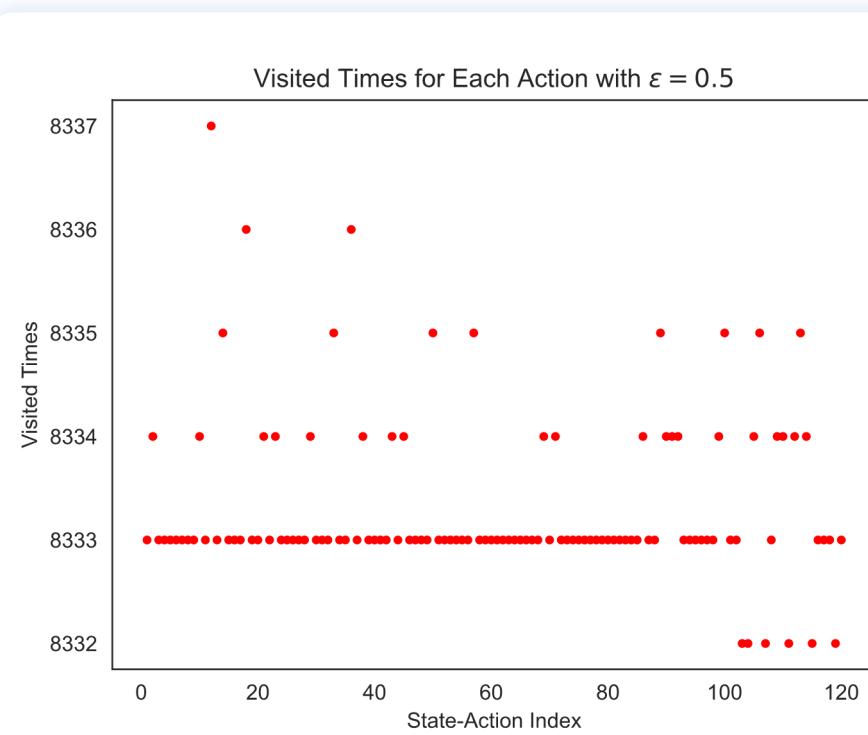


图 5.8

其次，考虑 $\epsilon = 0.5$ 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， ϵ -策略生成的一个回合如图 5.8 (e) - (g) 所示。虽然当回合足够长时，每个动作仍然可以被访问，但访问次数的分布可能极不均匀。例如，假设一个回合有 100 万步，某些动作的访问次数超过 250000 次，而大多数动作的访问次数仅为数百甚至数十次，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也会降低。一种有用的技术是最初将 ϵ 设置得很大以加强探索，然后逐渐减小它以确保最终策略的最优性^{5 6 7}。

-
1. M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, “Learning by playing solving sparse reward tasks from scratch,” in International Conference on Machine Learning, pp. 4344–4353, 2018. [\[1\]](#)
 2. J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, “How to train your robot with deep reinforcement learning: Lessons we have learned,” The International Journal of Robotics Research, vol. 40, no. 4-5, pp. 698–721, 2021. [\[2\]](#)
 3. S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, “Curriculum learning for reinforcement learning domains: A framework and survey,” The Journal of Machine Learning Research, vol. 21, no. 1, pp. 7382–7431, 2020. [\[3\]](#)
 4. C. Szepesv’ Algorithms for reinforcement learning. Springer, 2010. [\[4\]](#)
 5. A. Maroti, “RBED: Reward based epsilon decay,” arXiv:1910.13701, 2019. [\[5\]](#)
 6. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015. [\[6\]](#)
 7. W. Dabney, G. Ostrovski, and A. Barreto, “Temporally-extended epsilon-greedy exploration,” arXiv:2006.01782, 2020. [\[7\]](#)