1. 背景

上一章所介绍的 value iteration 算法和 policy iteration 算法有一个共同点，即它们都是基于模型的算法（即概率 p 是给定的），统称为 model-based reinforcement learning。而我们考虑下面一个例子：

抛硬币，每次得到的结果记为 X，当正面朝上时记为 X=1，当反面朝上时记为 X=-1，求$E(X)$？

■ 方法一：model-based

我们已知在该事件中的概率分布为：$P(X=1)=0.5, P(X=-1)=0.5$

则根据定义得到$E(X)=\sum_x xp(x)=1\times 0.5+(-1)\times 0.5=0$

而在实际应用中，我们难以得到如此精确的概率分布，即我们无法精确构建模型

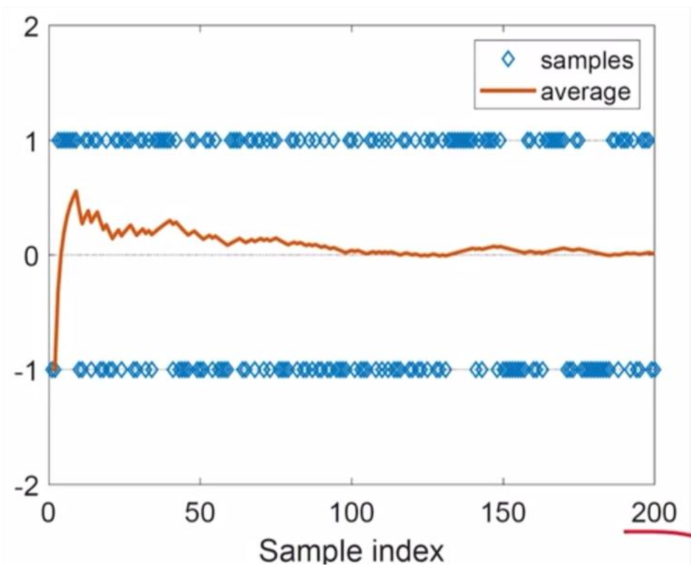
■ 方法二：model-free

该方法的思想是基于大数定理得来的，即经过多次的实验采样，近似求得期望

假设做了 N 次实验$\{x_1, x_2, x_3, \ldots, x_N\}$，则有：

$$E(X) \approx \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

这就是蒙特卡洛方法的思想，结果如下图所示：



大数定理：对于随机变量 X，若存在独立同分布的样例$\{x_i\}_{i=1}^N, \bar{x} = \frac{1}{N}\sum_{i=1}^N x_i$，则有：

$$E(\bar{x}) = E(X)$$
$$D(\bar{x}) = \frac{1}{N}D(X)$$

2. MC Basic 算法：最基本的蒙特卡洛算法

核心思想：将 policy iteration 中 model-based 的部分替换为 model-free，即为 MC Basic 算法

首先来回顾一下 policy iteration 算法：

给定一个初始策略π_0，基于该策略进行后续的更新

- 第一步，policy evaluation：根据当前策略计算出 state value

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

- 第二步，policy improvement：根据计算出的 state value，对策略进行更新

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

写成元素形式为：

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

而在此表示式中，$\sum_r p(r|s,a)r + \gamma\sum_{s'} p(s'|s,a)v_k(s')$即$q_{\pi_k}(s,a)$是 model-based 的部分，我们要将这部分替换为 model-free，考虑 action value 的定义：

$$q_{\pi_k}(s,a) = E(G_t|S_t = s, A_t = a)$$

而求期望这个过程，可以采取大数定理进行近似估计：

（1） 从我们要求的(s,a)开始，跟随 policy，生成一个 episode（有限的 trajectory）

（2） 将该 episode 的 return 记为$g(s,a)$

（3） 则$g(s,a)$为G_t的一个采样

（4） 循环得到一个采样序列$\{g^{(i)}(s,a)\}$，则

$$q_{\pi_k}(s,a) = E(G_t|S_t = s, A_t = a) \approx \frac{1}{N}\sum_{i=1}^{N} g^{(i)}(s,a)$$

MC Basic 的思想为：当模型无法构建时，我们可以使用大量的数据（经验）

该算法的描述为：

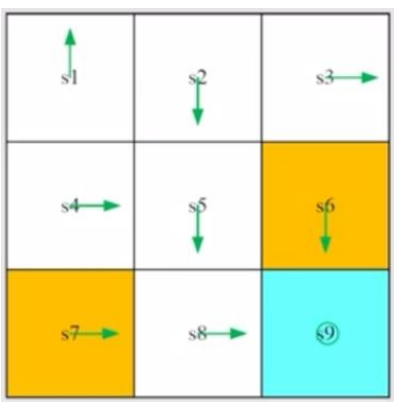
- **Step 1: policy evaluation.** This step is to obtain $q_{\pi_k}(s,a)$ for all (s,a). Specifically, for each action-state pair (s,a), run an infinite number of (or sufficiently many) episodes. The average of their returns is used to approximate $q_{\pi_k}(s,a)$.
- **Step 2: policy improvement.** This step is to solve $\pi_{k+1}(s) = \arg\max_\pi \sum_a \pi(a|s) q_{\pi_k}(s,a)$ for all $s \in \mathcal{S}$. The greedy optimal policy is $\pi_{k+1}(a_k^*|s) = 1$ where $a_k^* = \arg\max_a q_{\pi_k}(s,a)$.

该算法的伪代码为：

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

While the value estimate has not converged, for the kth iteration, 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) following π_k
      MC-based policy evaluation step:
      $q_{\pi_k}(s,a) =$ average return of all the episodes starting from (s,a)
    Policy improvement step:
    $a_k^*(s) = \arg\max_a q_{\pi_k}(s,a)$
    $\pi_{k+1}(a|s) = 1$ if $a = a_k^*$, and $\pi_{k+1}(a|s) = 0$ otherwise

例子：有初始策略如下图，$r_{boundary} = r_{forbidden} = -1, r_{target} = 1, \gamma = 0.9$



对状态s_1的策略进行更新：

（1） 第一步：policy evaluation，计算$q_{\pi_0}(s_1,a)$

由于此时策略是确定的，故一个(s,a)对只可能得到一种结果，所以直接求解即可，不需要大量采样求平均值

从(s_1,a_1)出发得到的 episode 为$s_1 \overset{a_1}{\rightarrow} s_1 \overset{a_1}{\rightarrow} s_1 \overset{a_1}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_1,a_1) =- 1+\gamma(-1)+\gamma^2(-1)+\ldots =- 10$$

从(s_1,a_2)出发得到的 episode 为$s_1 \overset{a_2}{\rightarrow} s_2 \overset{a_3}{\rightarrow} s_5 \overset{a_3}{\rightarrow} s_8 \overset{a_2}{\rightarrow} s_9 \overset{a_5}{\rightarrow} s_9 \overset{a_5}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_1,a_2) = 0+\gamma(0)+\gamma^2(0)+\gamma^3(1)+\gamma^4(1)+\ldots = 7.29$$

从(s_1,a_3)出发得到的 episode 为$s_1 \overset{a_3}{\rightarrow} s_4 \overset{a_2}{\rightarrow} s_5 \overset{a_3}{\rightarrow} s_8 \overset{a_2}{\rightarrow} s_9 \overset{a_5}{\rightarrow} s_9 \overset{a_5}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_1,a_3) = 0+\gamma(0)+\gamma^2(0)+\gamma^3(1)+\gamma^4(1)+\ldots = 7.29$$

从(s_1,a_4)出发得到的 episode 为$s_1 \overset{a_4}{\rightarrow} s_1 \overset{a_1}{\rightarrow} s_1 \overset{a_1}{\rightarrow} ..$，所以有：

$$q_{\pi_0}(s_1,a_4) =- 1+\gamma(-1)+\gamma^2(-1)+\ldots =- 10$$

从(s_1,a_5)出发得到的 episode 为$s_1 \overset{a_5}{\rightarrow} s_1 \overset{a_1}{\rightarrow} s_1 \overset{a_1}{\rightarrow} ..$，所以有：

$$q_{\pi_0}(s_1,a_5) = 0+\gamma(-1)+\gamma^2(-1)+\ldots =- 9$$

（2） 第二步：policy improvement，根据贪心算法，对策略进行更新

由以上结果可知：$q_{\pi_0}(s_1,a_2) = q_{\pi_0}(s_1,a_3)$是最大的，故根据贪心算法，新策略为：$\pi_1(a_2|s_1) = 1$ 或$\pi_1(a_3|s_1) = 1$

对状态s_3的策略进行更新：

（1） 第一步：policy evaluation，计算$q_{\pi_0}(s_3,a)$

从(s_3,a_1)出发得到的 episode 为$s_3 \overset{a_1}{\rightarrow} s_3 \overset{a_2}{\rightarrow} s_3 \overset{a_2}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_3,a_1) =- 1+\gamma(-1)+\gamma^2(-1)+\ldots =- 10$$

从(s_3,a_2)出发得到的 episode 为$s_3 \overset{a_2}{\rightarrow} s_3 \overset{a_2}{\rightarrow} s_3 \overset{a_2}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_3,a_2) =-1+\gamma(-1)+\gamma^2(-1)+\ldots=-10$$

从(s_3,a_3)出发得到的 episode 为$s_3 \overset{a_3}{\rightarrow} s_6 \overset{a_3}{\rightarrow} s_9 \overset{a_5}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_3,a_3) =-1+\gamma(1)+\gamma^2(1)+\ldots=8$$

从(s_3,a_4)出发得到的 episode 为$s_3 \overset{a_4}{\rightarrow} s_2 \overset{a_3}{\rightarrow} s_5 \overset{a_3}{\rightarrow} s_8 \overset{a_2}{\rightarrow} s_9 \overset{a_5}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_3,a_4) = 0+\gamma(0)+\gamma^2(0)+\gamma^3(1)+\gamma^4(1)+\ldots=7.29$$

从(s_3,a_5)出发得到的 episode 为$s_3 \overset{a_5}{\rightarrow} s_3 \overset{a_2}{\rightarrow} s_3 \overset{a_2}{\rightarrow} \ldots$，所以有：

$$q_{\pi_0}(s_3,a_5) = 0+\gamma(-1)+\gamma^2(-1)+\ldots=-9$$

（2） 第二步：policy improvement，根据贪心算法，对策略进行更新

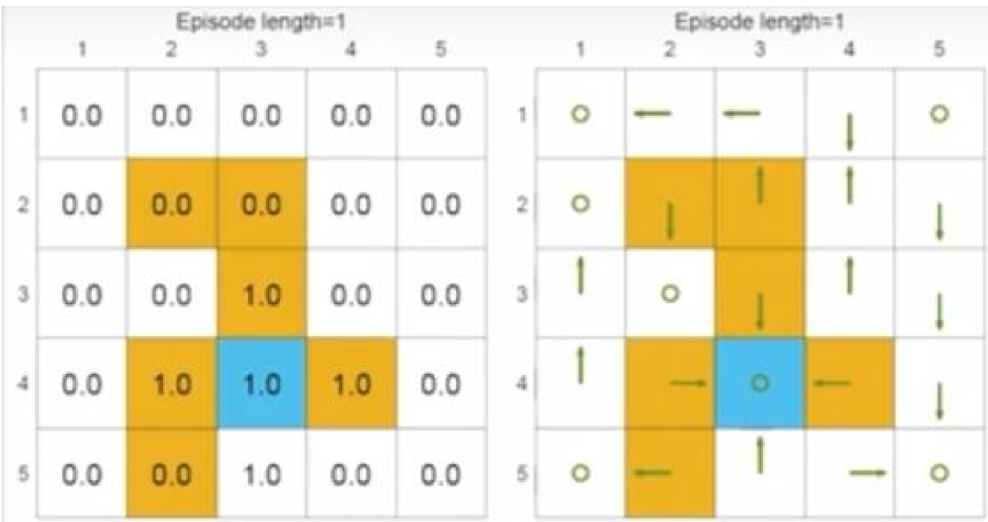
由以上结果可知：$q_{\pi_0}(s_3,a_3)$是最大的，故根据贪心算法，新策略为：$\pi_1(a_3|s_3)=1$

在这个例子中，episode 的长度是设置为无穷的。现在我们来看看 episode 的长度对策略更新的影响：

该例子为一个 5*5 的网格世界，$r_{boundary}=-1, r_{target}=1, \gamma=0.9, r_{forbidden}=-10$

若 episode 的长度为 1，即该 episode 只进行一个动作，表示为：$s \overset{a}{\rightarrow} s', r$

则策略更新如图所示：



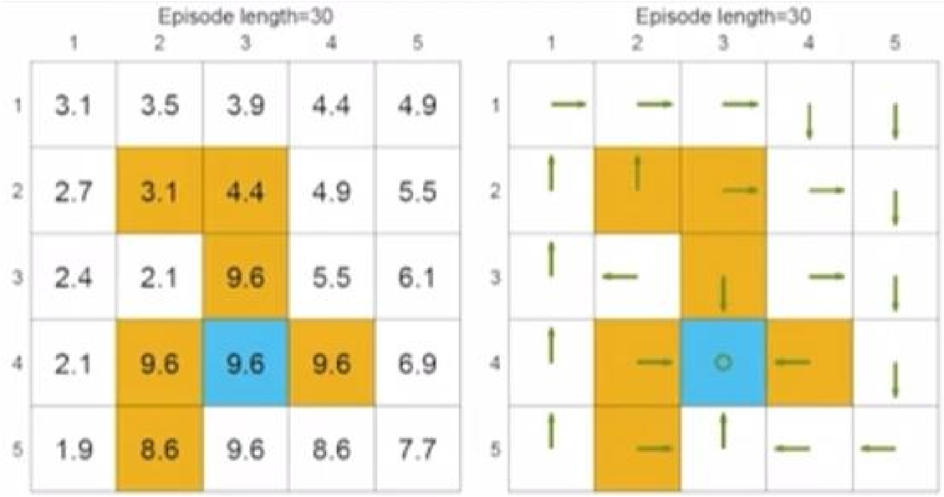
若若 episode 的长度为 2, 即该 episode 只进行两个动作, 表示为: $s \overset{a_1}{\rightarrow} s_1, r_1 \overset{a_2}{\rightarrow} s_2, r_2$

则策略更新如图所示:

episode 的长度为 14 的情况:

episode 的长度为 15 的情况:

episode 的长度为 30 的情况：



episode 的长度为 100 的情况：

通过这些例子我们可以看出：

- 当 episode 的长度比较短时，只有接近 target 的 state 能找到正确的 policy，且 state value 不为 0
- 随着 episode 的长度慢慢增长，距离 target 更近的 state 比距离 target 更远的 state 更早找到正确的 policy 且 state value 不为 0
- episode 的长度应该足够长
- 但是 episode 的长度也不需要无限长

3. MC exploring starts 算法

该算法是在 MC Basic 算法的基础上进行改进，回顾一下 MC Basic 算法：

MC Basic 算法是通过采样多个(s,a)对出发得到的 episode 的 return，计算期望，求得$q_{\pi_k}(s,a)$。考虑以下 episode：

$$
\begin{array}{rl}
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \ldots & \text{[original episode]} \\
s_2 \xrightarrow{a_4} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \ldots & \text{[episode starting from } (s_2, a_4)] \\
s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \ldots & \text{[episode starting from } (s_1, a_2)] \\
s_2 \xrightarrow{a_3} s_5 \xrightarrow{a_1} \ldots & \text{[episode starting from } (s_2, a_3)] \\
s_5 \xrightarrow{a_1} \ldots & \text{[episode starting from } (s_5, a_1)]
\end{array}
$$

我们可以将每一个(s,a)对当作一个起点，这样我们在计算某个$g(s,a)$时，可以根据其他的值来进行计算（采用了动态规划的思想）：

例如当要计算$g(s_1,a_2)$时，可以根据$g(s_2,a_4)$来进行计算，$g(s_1,a_2) = r_1 + \gamma g(s_2,a_4)$；要计算$g(s_2,a_4)$时，可以根据$g(s_1,a_2)$来进行计算，$g(s_2,a_4) = r_2 + \gamma g(s_1,a_2)$

此时会发现一个问题，$g(s_1,a_2)$既可以根据$g(s_2,a_4)$来计算，也可以根据$g(s_2,a_3)$来计算，这时就有两种策略：

- 策略一：first-visit method：即对每个(s,a)对，第一次访问到它的时候，只用第一次访问到的时候后面的g值来进行计算，即在此例子中只采用$g(s_2,a_4)$来计算$g(s_1,a_2)$
- 策略二：every-visit method：即对每个(s,a)对，每次访问到的时候，都用它后面的g值来进行计算，即在此例子中第一次访问到(s_1,a_2)时，采用$g(s_2,a_4)$来进行计算；第二次访问到到(s_1,a_2)时，采用$g(s_2,a_3)$来进行计算

该算法与 MC Basic 算法另一个不同点在于：在 policy improvement 这步中，MC Basic 算法是通过先多次采样$g(s,a)$值后，再计算平均值，求得$q_{\pi_k}(s,a)$，在这种方法下，agent 需要等到所有 episode 收集完毕才能进行下一步的更新；而在 MC exploring starts 算法中，每采样到一个$g(s,a)$值就对$q_{\pi_k}(s,a)$进行更新，提高效率

MC exploring starts 算法的伪代码为：

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

For each episode, do
  Episode generation: Randomly select a starting state-action pair (s_0, a_0) and ensure that all pairs can be possibly selected. Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$.
  Policy evaluation and policy improvement:
  Initialization: $g \leftarrow 0$
  For each step of the episode, $t = T-1, T-2, \ldots, 0$, do
    $g \leftarrow \gamma g + r_{t+1}$
    Use the first-visit strategy:
    If (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1})$, then
      $Returns(s_t, a_t) \leftarrow Returns(s_t, a_t) + g$
      $q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$
      $\pi(a|s_t) = 1$ if $a = \arg\max_a q(s_t, a)$

在计算g值时，是从后往前计算，而不是从前往后计算，便于提高效率（动态规划思想，上面介绍过的）

该算法在实践中难以实现，因为通过这种算法，我们要计算每个(s,a)对的 action value，必须保证在每个(s,a)对都进行了采样，尽管有些(s,a)对可以依赖其他(s,a)对进行计算，但我们无法保证从某个(s,a)对出发，可以遍历所有的(s,a)对，它是依赖于环境和模型的

4. MC epsilon-greedy 算法

为了解决 MC exploring starts 算法的缺陷，提出 MC epsilon-greedy 算法，该算法在 policy improvement 步骤上进行改进，具体方法如下：

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

在该算法中，$\varepsilon\in[0,1]$，$|A(s)|$表示一个状态能进行的所有 action 的数量。对于根据贪心算法求得的 action（即$q_{\pi_k}(s,a)$最大的 action），其概率为 $1-\frac{\varepsilon}{|A(s)|}(|A(s)|-1)$；对于其他 $|A(s)|-1$ 个 action，它们的概率均为$\frac{\varepsilon}{|A(s)|}$

- 在该算法中，greedy action 的概率一定大于等于其他 action 的概率：$1-\frac{\varepsilon}{|A(s)|}(|A(s)|-1)=1-\varepsilon+\frac{\varepsilon}{|A(s)|}\geq\frac{\varepsilon}{|A(s)|}$
- 为什么采用该算法？用于平衡 exploitation 和 exploration
- 当$\varepsilon=1$ 时，该算法即为贪心算法，此时的 exploitation（充分利用性）更强，exploration（探索性）更弱
- 当$\varepsilon=0$ 时，该算法即为均匀分布，每个 action 的概率相等，此时的 exploitation（充分利用性）更弱，exploration（探索性）更强

在原来的算法中，policy improvement 是个贪心算法：

$$\pi_{k+1}(s)=arg\max_{\pi}\sum_{a}\pi(a|s)\,q_{\pi_k}(s,a)$$

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

$$a_k^*(s)=arg\max_{a}q_k(a,s)$$

而在该算法中：

$$\pi(a|s)=\begin{cases}1-\frac{\varepsilon}{|A(s)|}(|A(s)|-1), a=a_k^*(s)\\ \frac{\varepsilon}{|A(s)|}, a\neq a_k^*(s)\end{cases}$$

该算法的伪代码为：

Initialization: Initial guess π_0 and the value of $\epsilon \in [0,1]$
Aim: Search for an optimal policy.

For each episode, do

Episode generation: Randomly select a starting state-action pair (s_0, a_0). Following the current policy, generate an episode of length T: $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$.

Policy evaluation and policy improvement:

Initialization: $g \leftarrow 0$

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

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

Use the every-visit method:

If (s_t, a_t) does not appear in $(s_0, a_0, s_1, a_1, \dots, s_{t-1}, a_{t-1})$, then

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

$q(s_t, a_t) = \text{average}(Returns(s_t, a_t))$

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

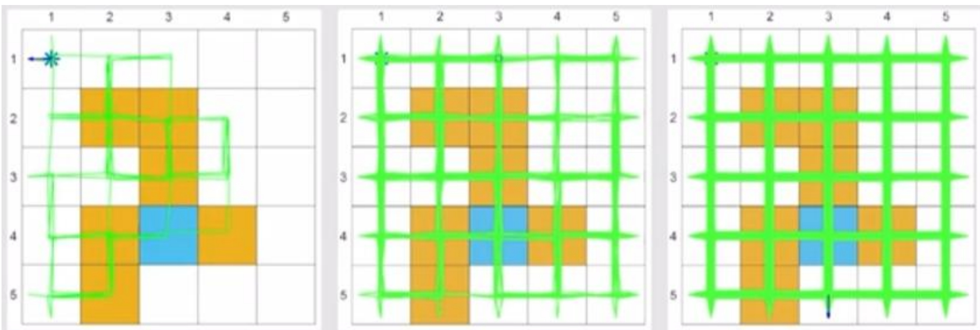
$$\pi(a|s_t) = \begin{cases} 1 - \frac{|\mathcal{A}(s_t)|-1}{|\mathcal{A}(s_t)|}\epsilon, & a = a^* \\ \frac{1}{|\mathcal{A}(s_t)|}\epsilon, & a \neq a^* \end{cases}$$

该算法的前面步骤与 MC exploring starts 算法一致。在该算法中，采用 every-visit 方法，即每次遇见(q,s)对，都使用其后面的g值对其进行计算

该算法使得更新过程中具有了一定的探索性，从一个(s,a)对出发即可得到包含所有(s,a)对的 episode（若该 episode 足够长），而不用像 MC exploring starts 算法一样，从每个(s,a)对出发以保证每个(s,a)对均被采样

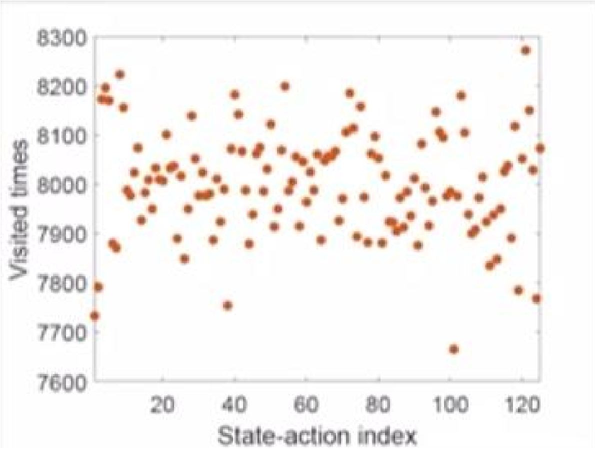
例子：

当$\varepsilon = 0$ 时，该算法为均匀分布，每个 action 的概率相等，分别设置 episode 的长度为 100、1000 和 10000，情况如图所示：

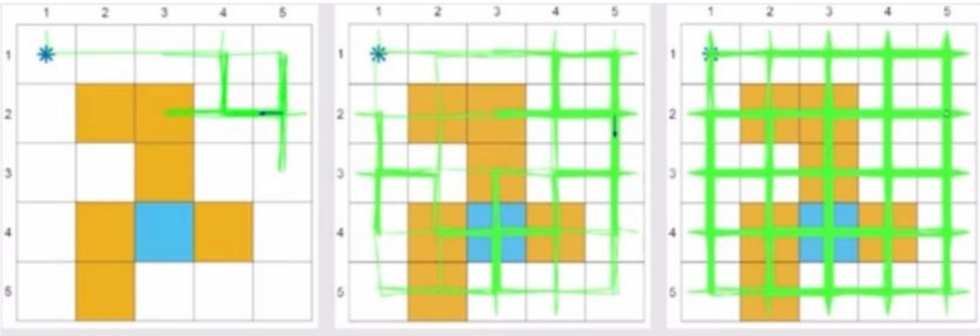


(a) 100 steps (b) 1000 steps (c) 10000 steps

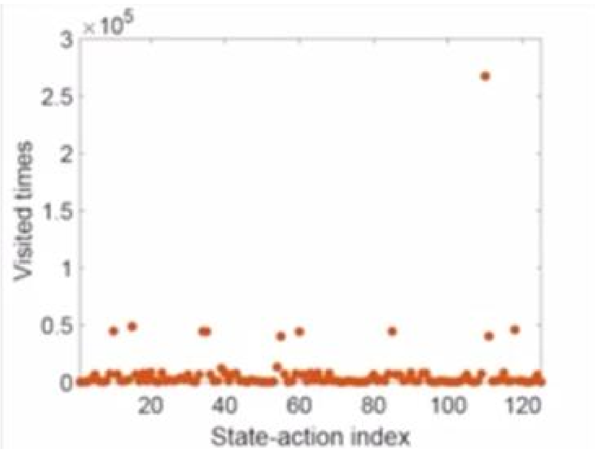
当 episode 的值为 1000000 时，统计每个(s,a)对被访问的次数，一共有 25 个 state，每个 state 有 5 个 action，故一共有 125 个(s,a)对，情况如图所示：



当ε比较小时，其探索性也比较小，episode 的长度分别为 100、1000、10000 和 1000000 的情况如图所示：



(a) 100 steps (b) 1000 steps (c) 10000 steps

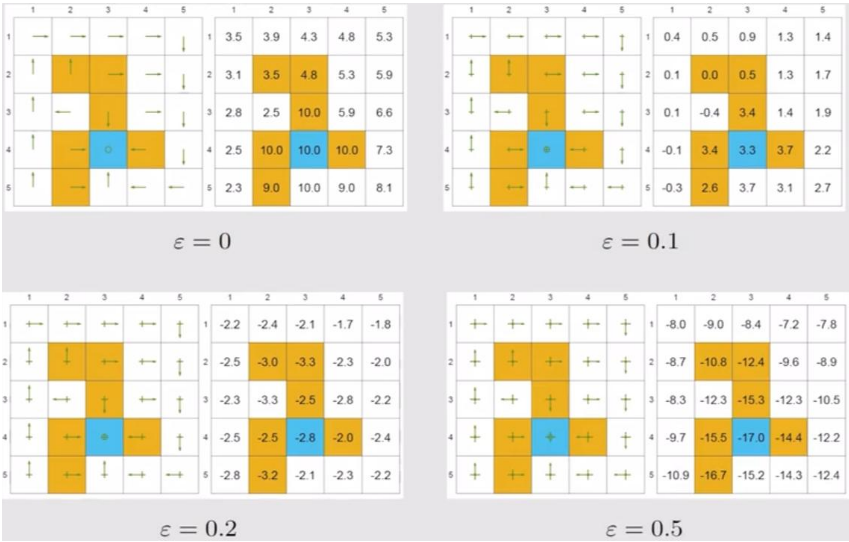


可以看到，此时探索性小了很多，某些(s,a)对被访问的次数特别多，大部分(s,a)对被访问的次数较少

由以上可知：

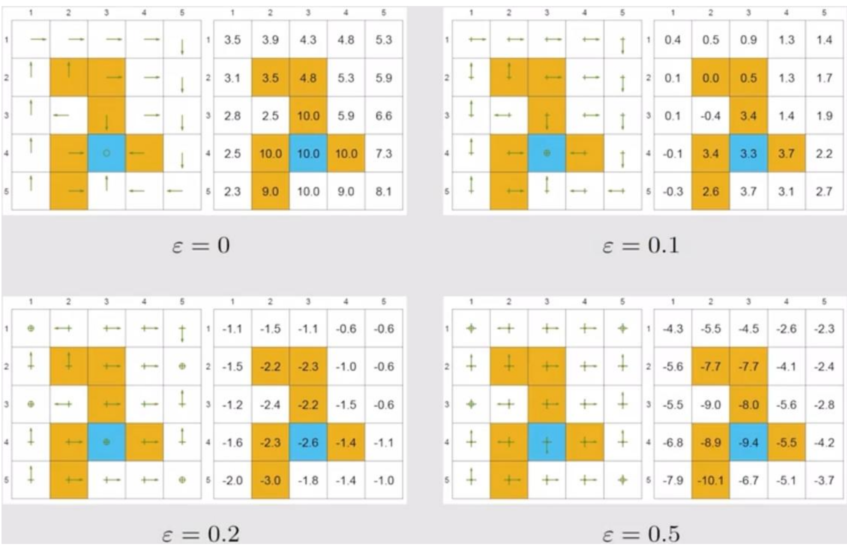
- 该算法确实具有一定的探索能力，以至于不需要 exploring starts 条件
- 但是该算法也损失了它的最优性，它所找到的最优策略不是真正的最优策略，只能通过控制ε来接近最优策略

例子 1：



通过逐渐增大ε值，我们可以看到 state value 是在不断减小的，但是这 4 个策略是 consistent（一致的），因为后面三个 policy，每个 state 在实际最优 action 的概率最大，即若将它们转变为 greedy，即可得到第一个 policy

例子 2：



而在该例子中，只有第二个 policy 与实际最优 policy 是一致的，后面两个 policy 效果都没有那么好

通过上述两个例子说明，在实际应用中ε的值不能取得太大，要适当选择