第三节 动态规划（Dynamic Programming）

本讲着重讲解利用动态规划来求解强化学习的问题，动态规划应用于马尔可夫决策过程的规划问题而不是学习问题，我们必须对环境是完全已知的，才能做动态规划，即要需知道状态转移概率和对应的奖励。在已知环境模型的基础上判断一个策略的价值函数，并在此基础上寻找到最优的策略和最优价值函数，或者直接寻找最优策略和最优价值函数。

3.1 动态规划简介

动态规划（简称DP）是解决复杂问题的一种方法，通过把复杂问题分解为子问题，通过求解子问题,组合子问题的解从而得到整个问题的解。在解决子问题的时候，其结果通常需要存储起来被用来解决后续复杂问题。当问题具有下列特性时，通常可以考虑使用动态规划来求解：

- 一个复杂问题的最优解由数个小问题的最优解构成，可以通过寻找子问题的最优解来得到复杂问题的最优解，这个特性又称为最优子结构（optimal substructure）；
- 子问题在复杂问题内重复出现，使得子问题的解可以被存储起来重复利用，这个特性又称为重叠子问题（overlapping subproblem）。

因此，动态规划适合用来解决具有最优子结构和重叠子问题两个特性的问题。

马尔科夫决定过程（MDP）满足上述两个性质：Bellman方程把问题递归为求解子问题，价值函数就相当于存储了一些子问题的解，可以复用。因此可以使用动态规划来求解MDP。我们用动态规划算法来求解一类称为“规划”的问题。“规划”指的是在了解整个MDP的基础上求解最优策略，即清楚环境动力学模型的基础上：包括状态行为空间、状态转移矩阵、奖励等。这类问题不是典型的强化学习问题，我们可以用规划来进行预测和控制。

具体的数学描述如下：

预测： 给定一个MDP $< S, A, P, R, \gamma >$ 和策略 π ，或者给定一个MRP $S, P_\pi, R_\pi, \gamma >$ ，要求输出基于当前策略 π 的价值函数 V_π 。

控制： 给定一个MDP $< S, A, P, R, \gamma >$ ，要求输出最优价值函数 V_* 和最优策略 π_* 。

3.2 策略评估（Policy Evalution）

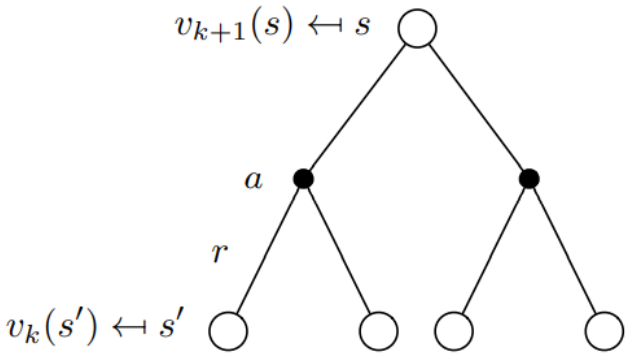
所谓策略评估是指已知马尔可夫决策过程以及要采取的策略 π ，计算价值函数 $V\pi(s)$ 的过程。策略评估在也被称为（价值）预测，即预测我们当前采取的策略最终会产生多少价值。因此，评估一个给定的策略 π ，也就是解决“预测”问题。

迭代策略评估(Iterative Policy Evaluation)

问题：评估一个给定的策略 π ，也就是解决“预测”问题。

解决方案：迭代应用贝尔曼期望方程的备份

具体方法： 同步备份（更新），即在每次迭代过程中，对于第 $k+1$ 次迭代，所有的状态 s 的价值用上一次迭代（第k次）后续状态 s’ 的价值 $v_k(s')$ 来更新本次迭代的状态价值 $v_{k+1}(s)$ ，其中 s’ 是 s 的后继状态。备份图如下：



公式为：

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P\left(s' \mid s, a\right) V_k\left(s'\right) \right)$$

其矩阵形式：

$$V^{k+1} = R^{\pi} + \gamma P^{\pi} V^{k}$$

通过反复迭代最终将收敛至 v_π。在第二讲学生MRP （ $\gamma = 0.9$ ）价值函数计算的例子中，价值函数计算用了同步更新的方法，30次迭代后趋于收敛。

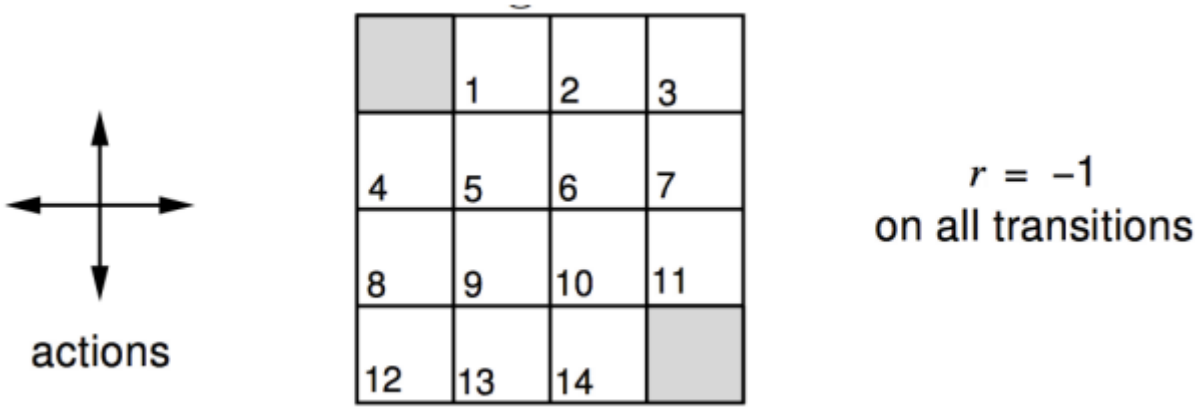
迭代策略评估算法如下图,计算V值时，用的是期望：

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

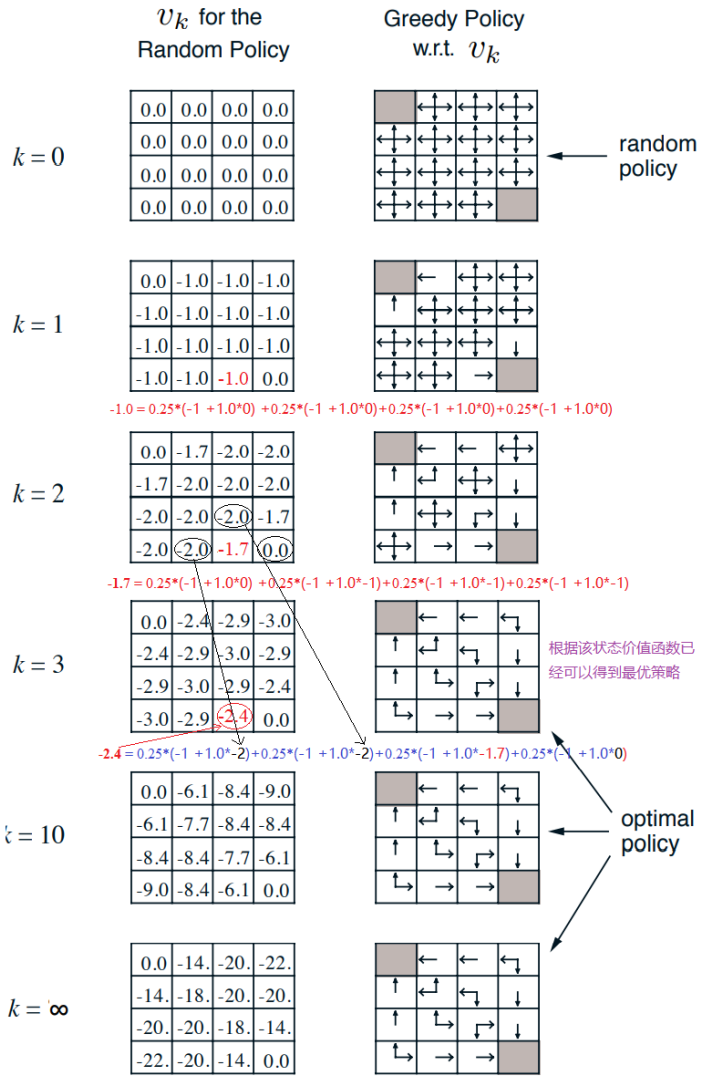
例3-1 格子世界，迭代策略评估例子，评估网格世界的随机策略



已知：
不带折扣的MDP（折扣系数 γ =1）；
状态空间S：{S1, S2, ..., S14}非终止状态，ST终止状态，上图灰色方格所示两个位置；
动作空间A：{n, e, s, w} 对于任何非终止状态可以有东南西北移动四个动作；
转移概率P：任何试图离开方格世界的动作的下一个状态还是原位置，其余条件下将100%地转移到动作指向的状态；
即时奖励R：在任何非终止状态间的转移的即时奖励均为-1，进入终止状态即时奖励为0；

当前策略 π ：智能体采用均匀随机策略，在任何一个非终止状态下有均等的概率的动作向四个方向移动，即

$$\pi(n|\bullet) = \pi(e|\bullet) = \pi(s|\bullet) = \pi(w|\bullet) = 1/4。$$



从上图中可知，在K=3迭代时，计算价值函数用的是上一次迭代k=2后续状态的价值，即-1.7及其周围的三个值。在第三次迭代后，得到最优策略。但是价值函数在一百多次迭代后才收敛。

此外，MDP一定存在一个最优的且是确定的策略，但这并不意味着只有确定策略是最优的。也可以是随机策略是最优的，比如在某个状态s时，有两个动作a1,a2有相同的Q值，即Q(a1,s) = Q(a2,s),那么你可以在a1,a2中随机的选一个。如上图右边最优策略中有两个箭头的状态。

3.3 策略迭代（Policy iteration）

如何改进策略

通过格子世界的例子，我们得到了一个优化策略的方法，分为两步：
首先，在一个给定的策略下评估价值函数，即计算状态价值，

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s]$$

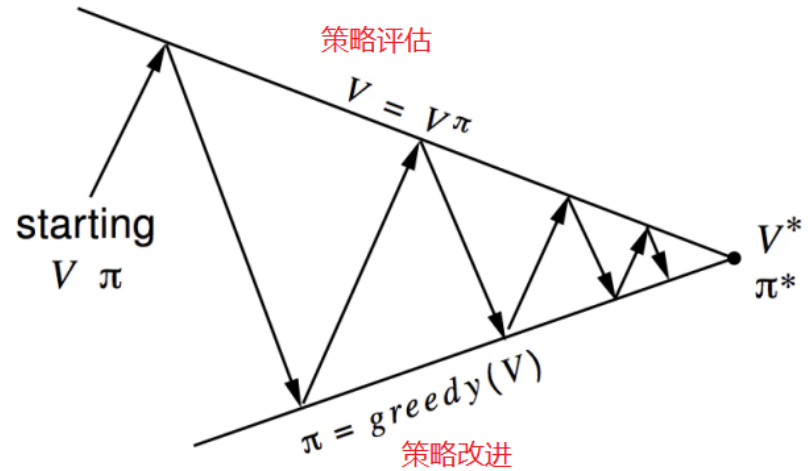
然后，根据价值函数 V_π ，贪婪地选取价值函数最大的行为，从而改进策略，

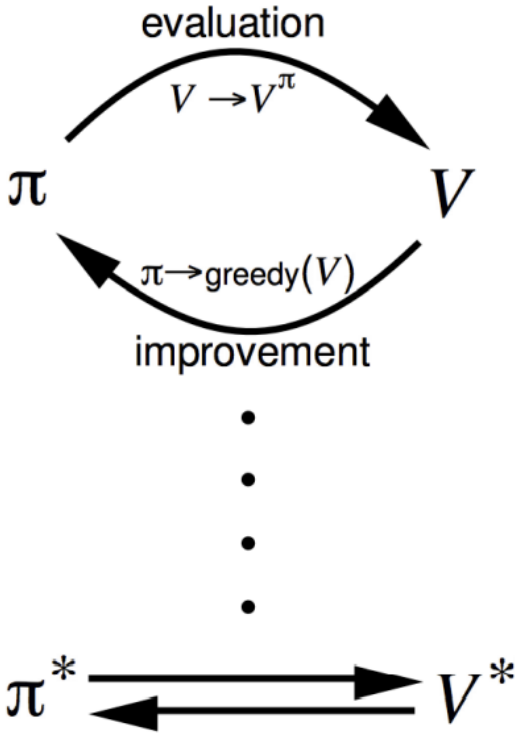
$$\pi' = greedy(V_\pi)$$

在前面的格子世界中，基于给定策略的价值迭代，改进的策略就是最优策略，但通过一个回合的迭代计算价值,策略改进就能找到最优策略不是普遍现象。通常，还需在改进的策略上继续评估，多次迭代：评估策略/改进策略。这种方法总能收敛至最优策略 π^* 。

策略迭代

在当前策略上迭代计算v值，再根据v值贪婪地更新策略，多次重复此过程，最终得到最优策略 π^* 和最优状态价值函数 V^* 。如下图所示，





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

策略改进

证明贪婪策略的迭代将收敛于最优策略和最优状态价值函数。

（1）考虑一个确定的策略： $a = \pi(s)$

（2）通过贪婪计算优化策略：

$$\pi'(s) = \arg\max_{a \in A} Q_\pi(s,a)$$

（3）从任意状态s，通过一步贪婪计算改进了状态s的Q值，即状态s采用贪婪计算，其他状态仍采用原策略。状态s在动作 $\pi'(s)$ 下得到的Q值： $Q_\pi(s,\pi'(s))$ 等于当前策略下状态s所有可能动作得到的最大值: $\max_{a \in A} Q_\pi(s,a)$ 。这个值大于等于使用当前策略 π 得到的行为 $\pi(s)$ 所得出的Q值: $Q_\pi(s,\pi(s))$ ，即状态价值 $V_\pi(s)$ 。

$$Q_\pi(s,\pi'(s)) = \max_{a\in A} Q_\pi(s,a) \geq Q_\pi(s,\pi(s)) = V_\pi(s) \tag{1}$$

（4）因此，价值函数得到改善，即： $V_{\pi'}(s) \geq V_\pi(s)$ ，证明如下：

$$\begin{aligned} v_\pi(s) &\leq Q_\pi(s,\pi'(s)) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma V_\pi(S_{t+2})] \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s] = V_{\pi'}(s) \end{aligned}$$

上式的证明中，只在某一状态下采用贪婪策略，其他状态仍使用原来的策略。例如，从等式到等式后面的小于等于号这一步(标红部分)，首先用了上一讲的全期望，即期望的期望等于期望，把括号里的期望去掉。然后，状态 S_{t+2} 时，用了贪婪策略 $\pi'(S_{t+2})$, 所以小于等于成立。如此递推下去，得到证明。

（5）如果Q值不再改善（即式（1）中的大于等于号变为等于号），则在某一状态下，遵循当前策略采取的行为得到的Q值将会是最优策略下所能得到的最大Q值，

$$Q_\pi(s,\pi'(s)) = \max_{a\in A} Q_\pi(s,a) = Q_\pi(s,\pi(s)) = V_\pi(s) \tag{2}$$

式（2）满足了贝尔曼最优方程，

$$V_\pi(s) = \arg\max_{a\in A} Q_\pi(s,a)$$

因此，当前策略下的状态价值就是最优状态价值，即对于任意状态s， $V_\pi(s) = V_*(s)$, 故当前策略就是最优策略。

策略迭代算法如下图，

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

修改过的策略迭代（Modified Policy Iteration）

很多情况下，策略的更新较早收敛至最优策略，而状态价值的收敛要慢很多，如，例3-1中，K=3策略收敛至最优，但状态价值在k= 150左右收敛。因此，策略评估是否有必要一定要迭代计算直到状态价值收敛至 $V_*(s)$?

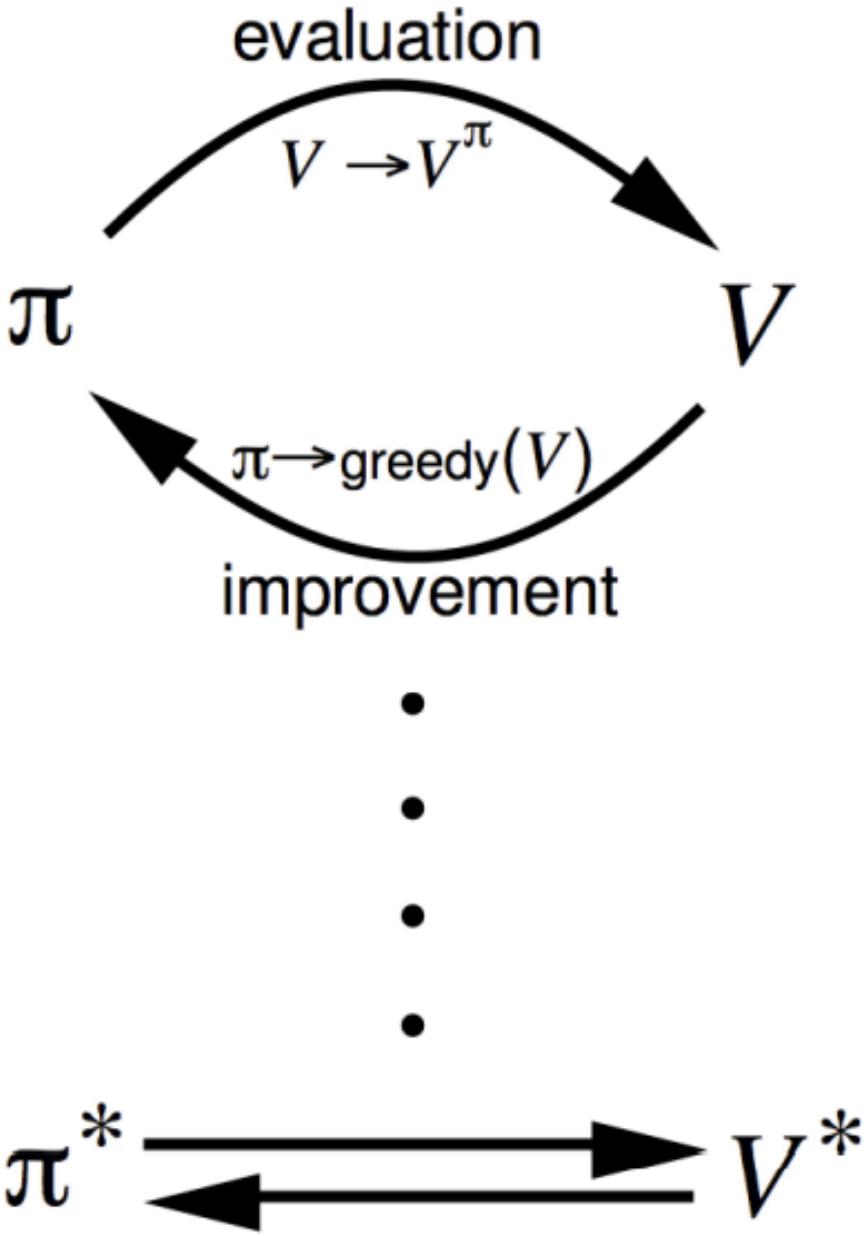
价值函数不需要持续迭代至最优时，可以，

- 或设定一个Ɛ，比较两次迭代的价值函数平方差；
- 或设置迭代次数k，迭代k次后停止,如例3-1，k=3后停止；
- 每迭代一次更新一次策略，k=1，这等价于价值更行，下一节即将介绍。

广义策略迭代（Generalised Policy Iteration）

策略迭代包括两个同时进行且交互的过程，一个使得价值函数与当前策略一致（策略评估），另一个是关于当前价值函数的贪婪策略（策略改善）。在策略迭代中，这两个过程交替进行，每个过程在另一个过程开始之前完成，但这显然不是必需的。 例如，价值迭代（value iteration）中，在每个策略改善之间仅执行一次策略评估迭代。 在异步（asynchronous）动态规划时，评估和提升过程则以更精细的方式交错。 只要两个过程都持续更新所有的状态，那么最终结果通常是相同的，即收敛到最优值函数和最优策略。

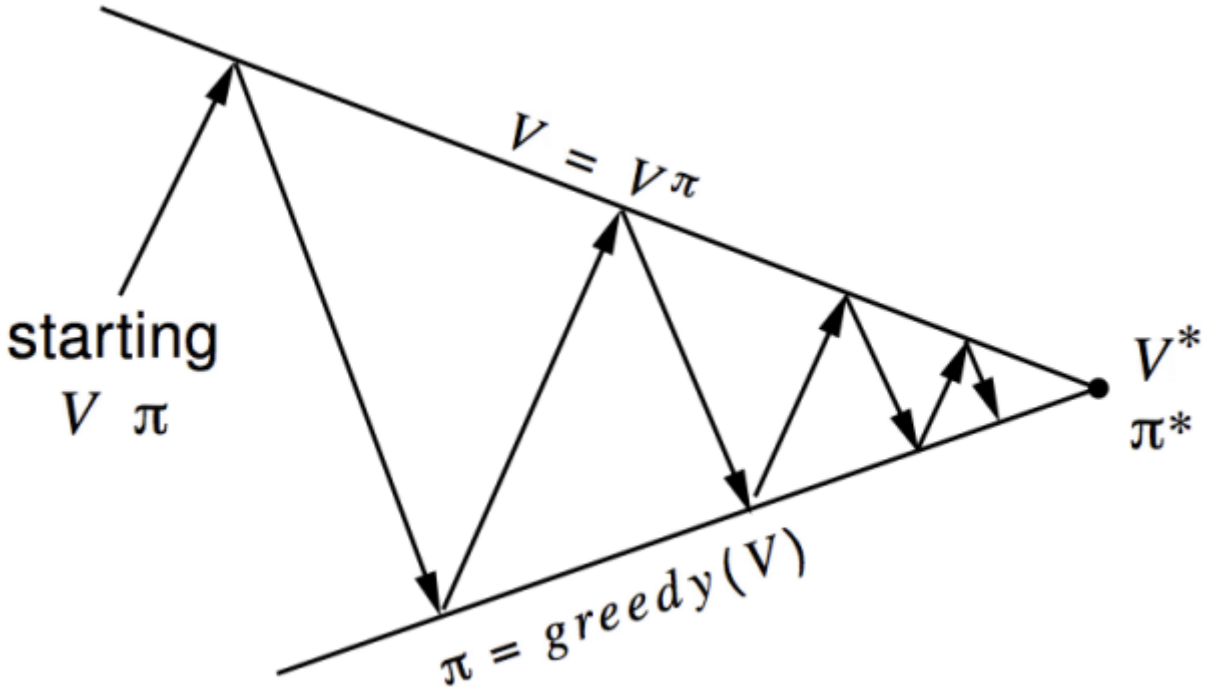
广义策略迭代（Generalized Policy iteration，GPI）指让策略评价和策略提升交互的广义概念，而不依赖于两个过程的粒度（granularity）和其他细节。几乎所有强化学习方法都可以被很好地描述为广义策略迭代。也就是说，所有具有下面特征的都可以称为GPI，即有可识别的策略和价值函数，策略总是关于价值函数被改善，并且价值函数总是朝着策略价值函数逼近，如下图所示，



如果评估过程和改善过程都趋于稳定，即不再发生变化，那么价值函数和策略必须都是最优的。仅当价值函数与当前策略一致时，价值函数才稳定；仅当策略关于当前价值函数贪婪时，策略才稳定。因此，仅当策略被认为关于策略自身的评估函数贪婪时，两个过程才稳定，这意味着满足贝尔曼最优方程，

$$V_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma V_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V_*(s')]$$

我们可以把GPI中的策略评估和策略改善的交互过程看成是两个目标或约束，例如，如下图中的两条线所示，上面的线表示策略评估的（计算策略价值函数）答案，下面的线表示策略改善（关于价值函数的贪婪策略）的答案。每一个过程驱使价值函数（或策略）朝着表示两个目标之一的答案的其中一条线逼近。直接逼近其中一个目标导致远离另一个目标。这个联合过程必然逼近最优总体目标。



3.4 价值迭代（Value iteration）

1.优化原则（Principle of Optimality）

一个最优策略可以被分解为两部分：

- 从状态s到下一个状态s'采取了最优行为 A_*；
- 在下一个状态s'时遵循一个最优策略。

定理：
一个策略 $\pi(a|s)$ 能够使得状态s获得最优价值 $V_\pi(s) = V_*(s)$，当且仅当：
对于可以从状态s到达的任何状态s'，
从状态s'开始，策略 π 能够取得最优价值：$V_\pi(s') = V_*(s')$

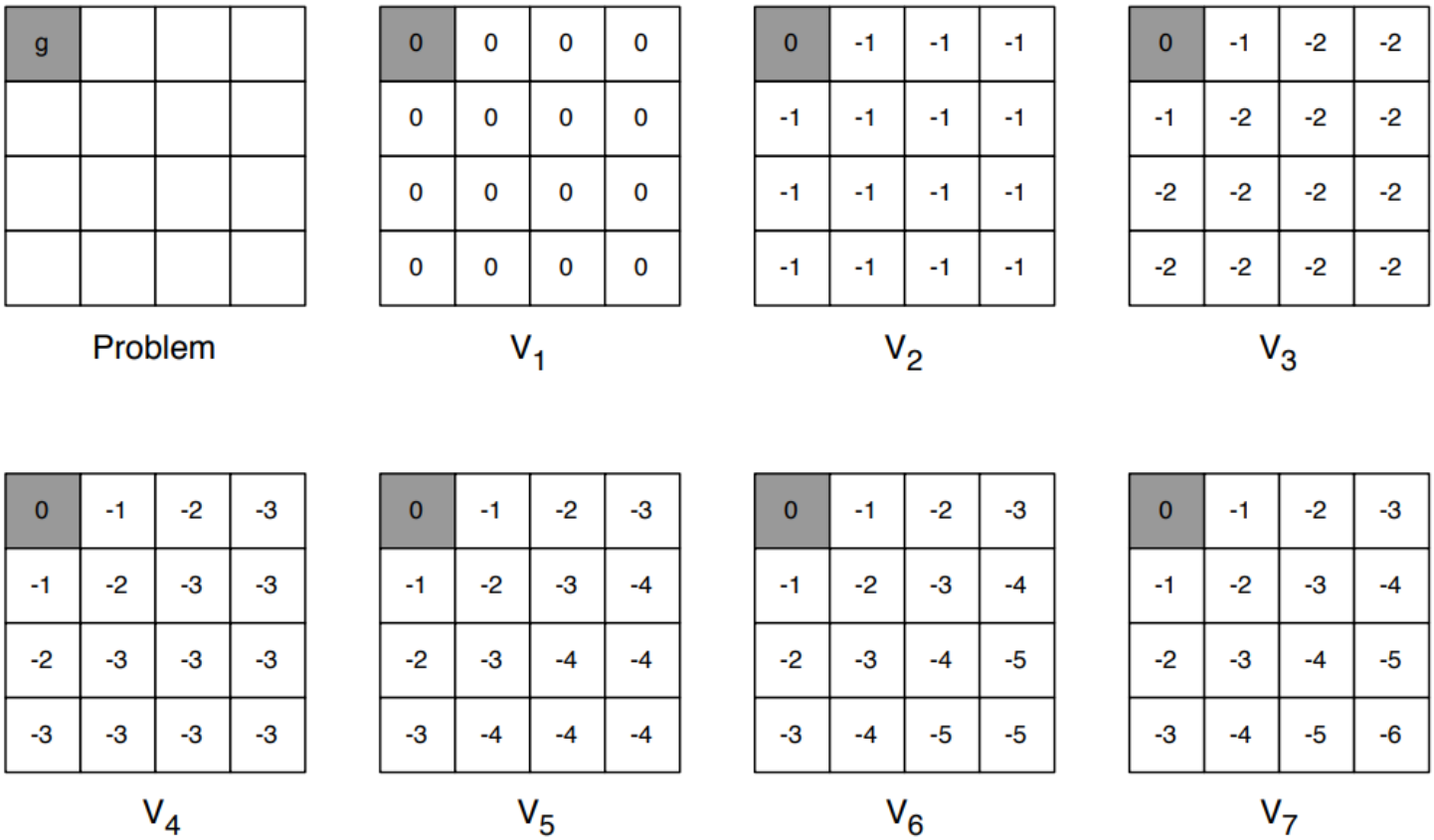
2.确定性的价值迭代（Deterministic Value Iteration）

如果我们知道子问题的解 $V_*(s')$ ， $V_*(s)$ 的解可以通过往前看一步（one-step lookahead）得到，

$$V_*(s) \leftarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')$$

价值迭代的思想就是应用上式这个更新进行不断迭代。这个思想的直觉是：如果我们知道最终目标状态的位置以及最终奖励，那么就可以根据上式，从最终状态开始逐步反向推出所有状态的最优价值。

例3-2，格子的最短路径
在下图中，左上角为终点，要求每个格子到终点的路径最短。从终点开始，反向更新（从邻近终点的状态朝着远离终点的方向）每个状态的价值，每一步只更新相邻状态的价值发生了变化的状态价值。

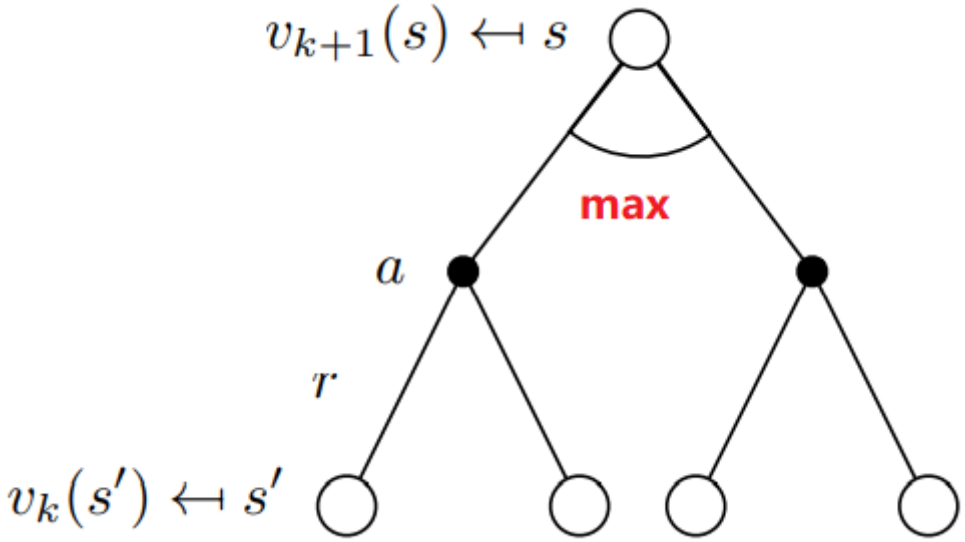


3.价值迭代

问题：寻找最优策略 π
解决方案：迭代地应用贝尔曼最优方程

$$V_1 \rightarrow V_2 \rightarrow \ldots \rightarrow V_*$$

从初始状态价值开始同步迭代计算，最终收敛，整个过程中没有遵循任何策略,价值迭代没有显式的策略，它通过贝尔曼最优方程，隐式地实现了策略改进这一步。
价值迭代的备份图如下：



$$V_{k+1}(s) = \max_{a \in A} \; R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s') \tag{3}$$

$$V_{k+1} = \max_{a \in A} \; R^a + \gamma P^a V_k$$

理解价值迭代的另外一个方法就是通过参考贝尔曼最优方程，贝尔曼最优方程将贝尔曼方程的期望操作变为求最大值操作，即实现了策略改进这一步。策略迭代的更新和价值迭代的更新也是相同的，除了价值迭代需要求关于所有动作的最大值（上面备份图中的max）

价值迭代算法如下，求V值中用的是Max，

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

4.同步动态规划算法

问题	贝尔曼方程	算法
预测	贝尔曼期望方程	策略评估迭代
控制	贝尔曼期望方程+贪婪策略改进	策略迭代
控制	贝尔曼最优方程	价值迭代

- 预测问题：在给定策略下迭代计算价值函数
- 控制问题：策略迭代寻找最优策略问题，先在给定或随机策略下计算状态价值函数，根据状态价值函数贪婪改善策略，多次迭代找到最优策略；价值迭代，全程没有策略参与，也可以获得最优策略，但需要知道状态转移矩阵，即状态s在行为a后到达的所有后续状态及概率(参考式（3）)。

基于状态价值函数或行为价值函数的两种价值迭代算法的时间复杂度都较大，分别为 $O(mn^2)$ 或 $O(m^2n^2)$,其中m为动作数，n为状态数。一种改进方案是使用异步动态规划。一些其他不使用动态规划的方法，将在随后几讲中介绍。

3.5 DP的扩展

1 异步动态规划（Asynchronous Dynamic Programming）

原位动态规划(In-place dynamic programming)： 直接原地更新下一个状态的v值，而不像同步迭代那样需要额外存储新的v值。在这种情况下，按何种次序更新状态价值有时候会比较有意义。

$$V(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right)$$

重要状态优先更新(Priortised Sweeping)： 使用贝尔曼误差的幅值来指导状态的选择。贝尔曼误差反映的是当前的状态价值与更新后的状态价值差的绝对值。贝尔曼误差越大，越有必要优先更新。这种方法可以使用优先级队列有效的实现。

实时动态规划（Real-time dynamic programming）： 更新那些仅与智能体关系密切的状态，同时使用个体的经验来指导更新状态的选择。

2 全宽度备份（full-width backups）

在全宽度备份方法中，每一次更新，所有的后继状态和动作都要被考虑在内，并且需要已知MDP的转移矩阵与奖励函数。DP对中等规模（百万级别的状态数）的问题较为有效。对于大规模问题，DP遭遇贝尔曼维度的诅咒（Bellman’s
curse of dimensionality）

3.近似动态规划

近似价值函数，通过使用近似函数，如神经网络，虽然不完全精确但却够用的近似价值函数。

3.6 压缩映射（contraction mapping）

1 一些技术问题

- 我们如何知道价值迭代收敛至最优价值函数 V_* ？
- 或者策略评估迭代收敛至 V_π ？
- 策略迭代收敛至 V_* ？
- 解是唯一的吗?
- 这些算法的收敛速度有多快?
- 所有的这些问题可以被压缩映射解决。

2 价值函数空间

- 考虑价值函数的向量空间 $\mathbb{V}$
- 维度为 |S|
- 这个空间中的每一个点完全指定了一个价值函数 $v(s)$
- 贝尔曼备份对这个空间中的点做了什么?
- 我们将会展示贝尔曼备份使得价值函数之间更接近
- 因此，备份必须收敛于一个唯一解

3 价值函数的无穷大范数

我们将使用 $\infty - norm$ 来测量状态价值函数u和v的距离，例如，状态价值之间的最大差距如下，

$$||u - v||_\infty = \max_{s \in S} |u(s) - v(s)|$$

4 贝尔曼期望备份是一个压缩映射

定义贝尔曼期望备份的操作 T^π ，

$$T^\pi(v) = R^\pi + \gamma P^\pi v$$

这个操作是一个 γ 压缩， 如，它使得价值函数至少 γ 接近，

$$
\begin{aligned}
||T^{\pi}(u) - T^{\pi}(v)||_{\infty} &= ||\ (R^{\pi} + \gamma P^{\pi} u) - (R^{\pi} + \gamma P^{\pi} v)\ ||_{\infty} \\
&= ||\gamma P^{\pi}(u - v)||_{\infty} \\
&\leq ||\gamma P^{\pi}||u - v||_{\infty}\ ||_{\infty} \\
&\leq \gamma ||u - v||_{\infty}
\end{aligned}
$$

5 压缩映射定理（contraction mapping theorem）

又称为Banach fixed-point theorem

定理： 对于任何度量空间 V 在操作 $T(v)$ 是完备的（如，封闭）, 其中T是一个 γ 压缩，

- T收敛于一个唯一固定点
- 以一个线性收敛速率 γ 收敛

6 策略评估和策略迭代的收敛

- 曼期望备份的操作 T^{π} 有一个唯一固定点
- v_{π} 是 T^{π} 的一个固定点（通过贝尔曼期望方程）
- 通过压缩映射定理
- 策略评估迭代收敛至 V_{π}
- 策略迭代收敛至 V_{*}

7 贝尔曼最优备份是一个压缩

- 定义贝尔曼最优备份操作 T^{*} ,

$$
T^{*}(v) = R^{\pi} + \gamma P^{\pi} v
$$

- 这个操作是一个 γ 压缩， 如，它使得价值函数至少 γ 接近（证明同前面相似），

$$
||T^{*}(u) - T^{*}(v)||_{\infty} \leq \gamma ||u - v||_{\infty}
$$

8 价值函数迭代的收敛

- 曼期望备份的操作 T^{*} 有一个唯一固定点
- v_{*} 是 T^{*} 的一个固定点（通过贝尔曼最优方程）
- 通过压缩映射定理
- 价值迭代收敛至 V_{*}

关于压缩映射与强化学习相关的内容，打算后面写一个详细的介绍。

参考

1.David Silver第3课

2.叶强《David Silver强化学习公开课中文讲解及实践》
3.Richard Sutton 《Reinforcement Learning A Introduction》