规划算法之动态规划

人工智能实验室

理查德·贝尔曼(Richard Bellman) 美国数学家

- **生平贡献**
  - **首先提出“维数灾难”**：维度增加时，问题规模呈指数级增长
  - **Bellman方程**：用于最优控制理论。解决多阶段决策过程(multi-stage decision process)。是动态规划的数学基础。
  - **动态规划(Dynamic Programming)**：最早用于最优控制理论中，研究问题的优化方法。之后被抽象简化放入算法框架，幸运地成为了算法课程考察的一个知识点。

Paragraph referenced from Bellman, Richard. "The theory of dynamic programming." Bulletin of the American Mathematical Society 60.6 (1954): 503-515. Figure from wiki *“Dynamic Programming”*

Bellman在他的自传中解释“动态规划”一词的由来：

- “我在1950年秋季在兰德度过了第一个任务是为多阶段决策过程找一个名字，一个有趣的问题是，动态规划这个名称是怎么来的？20世纪50年代不是数学的好年头。在华盛顿我们有一位非常有趣的绅士，威尔逊，他是国防部长，而且对“研究”这个词有一种病态的恐惧和仇恨——我没有轻描淡写地使用这两个词，而是很精准地在使用它们。如果有人胆敢在他面前提“研究”这个术语，他的脸会涨大，会发红，人会变得极为暴躁。你完全可以想象到他对数学这个词会产生什么样的感受。兰德公司受雇于空军，威尔逊又是空军的上司。因此，我觉得我必须采取一些措施，不让威尔逊和空军知道我正在兰德公司里悄悄地搞数学。”

Paragraph referenced from *https://people.ricam.oeaw.ac.at/d.kalise/files/Richard_Bellman_Eye_of_the_Hurricane.pdf*

动态规划的由来（续）

- “我能起什么样的名称？首先，我对计划(planning)、决策(decision making)、思考(thinking)感兴趣。但是对于计划(planning)，由于各种原因，并不是一个好词。因此我决定使用“规划”(programming)一词。我想让大家知道这是动态(dynamic)的，多阶段，随时间变化。我想这真是个一石二鸟的好主意，用了传统物理意义上的词，也即动态，又表达了精确含义……”
- “因此，我认为动态规划(dynamic programming)是一个好名字，甚至国会议员都不能反对。我就这样把它作为我研究的保护伞。”

Paragraph referenced from *https://people.ricam.oeaw.ac.at/d.kalise/files/Richard_Bellman_Eye_of_the_Hurricane.pdf*

但是罗素和诺维格在他们著作中反驳……

- “这是不可能的，因为他第一篇使用该术语的论文(Bellman, 1952)，还要早于威尔逊成为国防部长的年份(1953年)。”
- Harold J. Kushner在他的一篇讲话中回忆了Bellman。不过他是引用Kushner的话：“另一方面，当我问他同样的问题时，他回答说，他试图通过动态添加来更新Dantzig的线性规划。也许这两个动机都是真的。”

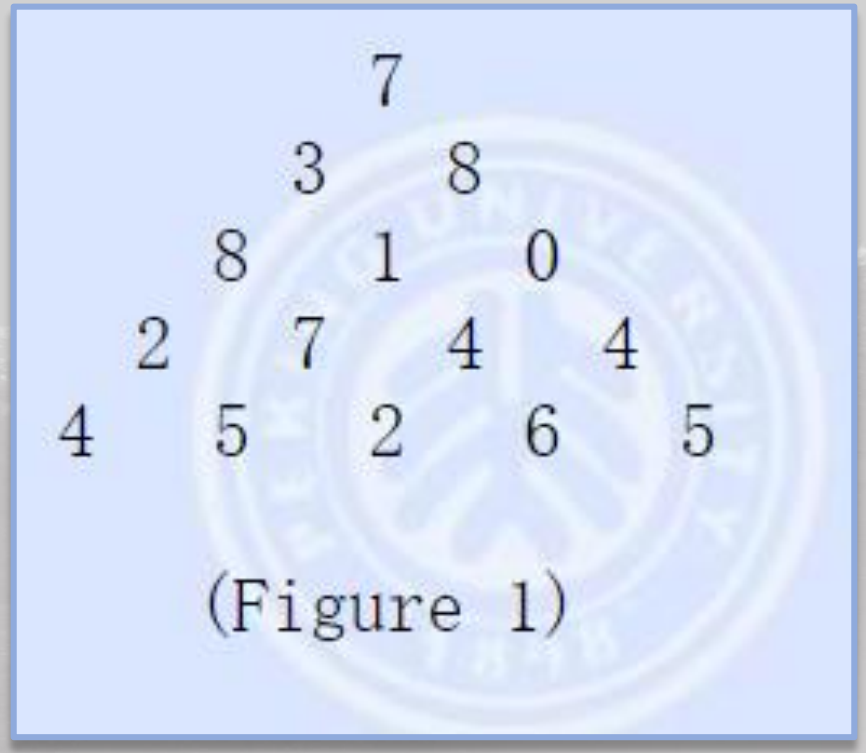
Paragraph referenced from *Wikipedia “Dynamic programming”*

在编程算法中的动态规划

- 三个重要概念
  - 最优子结构
  - 边界
  - 状态转移方程
    - 动态规划中的核心部分
    - 实质上是Bellman方程的变形形式
- 特点
  - 用递推代替递归
  - 用空间换时间

编程算法中的动态规划应用一

- 数塔问题（poj1163）
- 在右边的数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。路径上的每一步都只能往左下或右下走。
- 只需要求出这个最大和即可，不必给出具体路径。



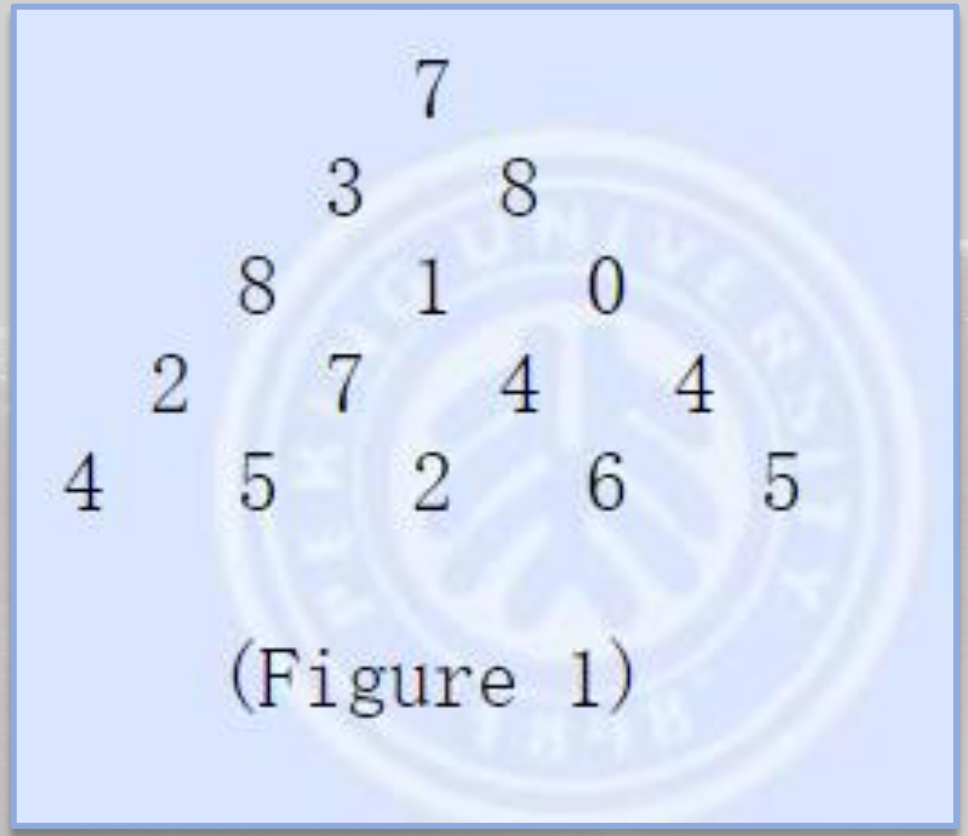
(Figure 1)

Paragraph referenced from poj 1163 *"The Triangle"*

数塔问题（poj1163）伪代码

- 在数字三角形中找一条从顶部到底边的路径，使得路径上经过的数字之和最大。路径上的每一步都只能往左下或右下走。

```
Input num, r, c. // num是二维数组，r是行数，c是列数
Initialize a two-dimension array dp[r][c], whose size is the same as num // dp是一个二维数组，存放的是以第r行第c列的结点为根的子树的最大和
for i<-0 to n-1
        dp[r-1][i] = num[r-1][i] // 复制最后一行数
for i<-(r-2) to 0 by -1
        for j<-0 to n-1
                if i>= j
                        dp[i][j]=max(dp[i+1][j],dp[i+1][j+1]+num[i][j])
return dp[0][0]
```



(Figure 1)

Paragraph referenced from poj 1163 *"The Triangle"*

应用二：单源最短路径问题伪代码 Bellman-Ford算法

- 给定一个带权有向图G=(V,E)，其中每条边的权是一个实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到其他所有各顶点的最短路径长度。这里的长度就是指路上各边权之和。这个问题通常称为单源最短路径问题。

```
for 每一个结点v ∈ V
        Distance(v) := +∞
Distance(s) := 0
循环 n-1 次
        for 每一条路径 (u,v) ∈ E
                if Distance(u) + weight(u,v) < Distance(v) then
                        Distance(v) := Distance(u) + weight(u,v)
for 每一条路径 (u,v) ∈ E
        if Distance(u) + weight(u,v) < Distance(v)
        then 中止程序并且返回 “找到负循环”
返回 执行成功
```

Bellman论文里对动态规划的说明(1954)

- 动态规划理论一开始创造出来是为了解决**多阶段决策过程**(multi-stage decision process)中的数学问题：我们有一个物理系统，任何时刻这个系统的**状态**(state)能被描述为一组量，我们称为状态参数或状态变量。有时，这些状态变量可能被过程本身所决定。这些**决策**(decisions)等价于状态变量之间的**转移**(transformations)，决策的选择与转移的选择是互相独立的。用前一段决策的结果来指导后面的决策，整个过程的目标是最大化**终状态参数**的相关函数。
- （作为一个数学家，）传统步骤是把所有可行策略算一遍，然后取最大值……但是作为一个“实用”的人，我们却要看到维度过大的代价——这会叫一台现代计算机器瑟瑟发抖的——因为有太多信息了。

Bellman论文里对动态规划的说明续(1954)

- 考虑一个中间状态。由于从初状态到中间状态的过程，我们并不关心，之后的决策都只从这个中间状态开始。这就叫做**无后效性**。
- **最优原则**(principle of optimality)：
  - Bellman论文的描述：一个最优策略有这样的性质——无论初状态和最初的几个决策是什么，剩余决策一定构成一个与之前决策产生的状态相关的一个最优策略。
  - 形式化描述：一个策略$\pi(a|s)$能达到状态s下的最优值$v_\pi(s) = v_*(s)$，当且仅当：对于从状态s出发可达到的任意状态s'，策略π能达到新状态s'的最优值$v_\pi(s') = v_*(s')$。

大纲

- 动态规划（Dynamic Programming）与MDP问题
- 策略迭代（Policy Iteration）
  - 策略估值（Policy Evaluation）
  - 策略优化（Policy Improvement）
- 值迭代（Value Iteration）
- 异步动态规划(Asynchronous Dynamic Programming)
- 广义策略迭代（Generalized Policy Iteration）
- 动态规划效率（Efficiency of Dynamic Programming）

回顾MDP几个要素

- 马尔科夫决策过程(Markov Decision Process)
  - 状态(state)
    - 某一时刻的游戏局面，如围棋的某个局面
    - 特殊的状态：初始状态和结束状态
  - 动作(action)
    - 游戏玩家在某个游戏状态下可以采取某个动作，如在一个围棋局面下游戏玩家可以选择某个落子位置
  - 状态转移(transition)
    - 某个游戏状态下，在玩家采取某个游戏动作后，下一游戏状态的概率分布
  - 奖赏(reward)
    - 某个游戏状态下，在玩家采取某个游戏动作后，玩家得到的分数

回顾MDP的数学定义

- 马尔科夫决策过程的数学定义
  - M = <S, A, P, R, γ>
    - S: 状态集合
    - A: 动作集合
    - P: 状态转移函数 <S, A, S> -> $\mathcal{R}^+$， P(s, a, s') = Pr[s'|s, a]， s和a是当前状态和动作，s'是下一状态
    - R: 奖赏函数 <S, A, $\mathcal{R}^+$ > -> $\mathcal{R}^+$ ， R(s, a, r) = Pr[r|s, a]， s和a是当前状态和动作，r是奖赏
    - γ: 折扣因子
- 游戏策略的定义
  - π:策略函数 <S, A> -> $\mathcal{R}^+$ ， π描述的是状态s下采取动作a的概率分布
- 玩家的游戏目标
  - 寻找最优的游戏策略π，使得从游戏初始状态s_1到游戏终结状态s_n的期望累积收益$\mathrm{E}[\sum_{i=1}^{n}\gamma^i r_i\,|M,\pi]$最大

回顾MDP的相关数学定义

- G_t是时间步t以来的总折旧回馈值。有

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{6.1}$$

- 状态值函数$v_\pi(s)$

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s] \tag{6.2}$$

- 动作状态值函数$q_\pi(s,a)$

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s, A_t = a] \end{aligned} \tag{6.3}$$

Bellman方程

Bellman期望方程

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_\pi(s')] \quad (6.4)$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (6.5)$$

$$\forall s \in S, a \in A(s), s' \in S^+$$

Bellman最优方程

$$v_*(s) = \max_\pi v_\pi(s) \quad (6.6)$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \quad (6.7)$$

$$\Rightarrow$$

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

$$= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \quad (6.8)$$

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \,\middle|\, S_t = s, A_t = a\right]$$

$$= \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \quad (6.9)$$

$$\forall s \in S, a \in A(s), s' \in S^+$$

Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P59,60,63*

最优状态值函数与最优动作状态值函数

$$
\begin{aligned}
v_*(s) &= \max_{\pi} v_\pi(s) \\
&= \max_{\pi} \sum_{a \in A} \pi(a|s) q_\pi(s,a) \\
&= \max_{a} q_*(s,a)
\end{aligned}
\tag{6.10}
$$

动态规划与MDP问题

- 动态规划能适用的问题的性质
  - **最优子结构性质**
  - **无后效性**：MDP蕴含的马尔科夫性，具有无后效性
  - **子问题重叠性质**：增加空间记录子问题的解，避免重复计算
- 动态规划需要假设完全环境，即完全可观察（Fully observable）；MDP的状态可见性是信息完全的
- 优化Bellman方程就是一个最优子结构，其迭代形式使得它可以用子问题表示；MDP具有的马尔科夫性，满足无后效性
- 结论：MDP可以用动态规划的方法解，得到最优策略

目标

预测（predict）

- 给定条件：
  - $MDP < S, A, P, R, \gamma >$和策略$\pi$
  - 或$MDP < S, P^{\pi}, R^{\pi}, \gamma >$
- 输出：基于当前策略π的状态值函数V_{π}

控制（control）

- 给定条件：
  - $MDP < S, A, P, R, \gamma >$
- 输出：最优状态值函数V_{*}和最优策略π_{*}

解决MDP思路

- 使用动态规划方法解决MDP
- 转移方程用Bellman方程的赋值形式给出
- 使用迭代更新的方法来逼近最优值

策略估值（Policy Evaluation）

注：(6.4)式中的v_π下标表示是对策略π的**准确稳定**状态值函数，(6.11)式中的v_k表示迭代策略估值过程中，第k次迭代得到的值函数，依据的策略是同一个π，此时值函数可能**不准确**

- 定义：对于任意策略π，计算在此策略下的状态值函数V_π
- 策略估值中Bellman方程的赋值形式

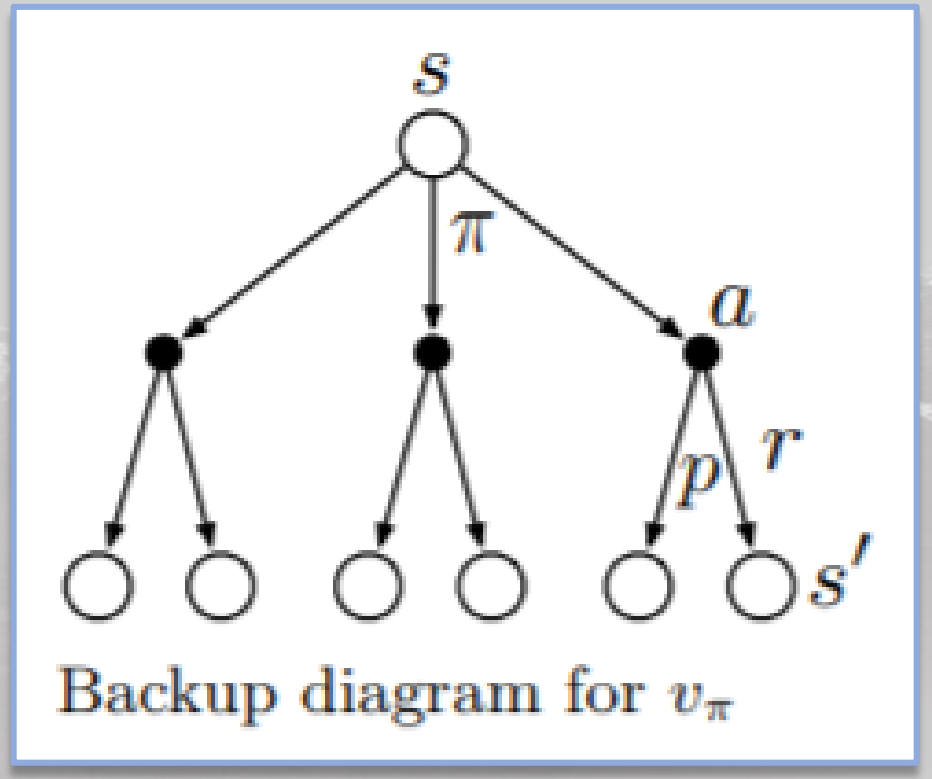
$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{6.4}$$

- 迭代更新规则：v_π是这个更新规则的不动点

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned} \tag{6.11}$$

迭代策略估值（iterative policy evaluation）

- **完全回溯（full backup）**：依据被评估策略π所有可能的一步转移，用$v_k(S)$依次计算$v_{k+1}(s), \forall s \in S$
- 迭代停止条件：测试$max_{s \in S}|v_{k+1}(s) - v_k(s)|$，当这个量足够小时停止
- 迭代内循环计算
  - 同步迭代
  - 异步迭代



Backup diagram for v_π

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p49*

迭代内循环计算

同步迭代（Synchronous iteration）

- 已有$v_k(s_1), v_k(s_2), \cdots, v_k(s_n)$
- 用以上值依次计算$v_{k+1}(s_1), v_{k+1}(s_2), \cdots, v_{k+1}(s_n)$
- 需要双数组分别存储旧值和新值

异步迭代（Asynchronous iteration）

- 用$v_k(s_1), v_k(s_2), \cdots, v_k(s_n)$计算$v_{k+1}(s_1)$
- 用$v_{k+1}(s_1), v_k(s_2), \cdots, v_k(s_n)$计算$v_{k+1}(s_2)$
- 用$v_{k+1}(s_1), v_{k+1}(s_2), \cdots, v_k(s_n)$计算$v_{k+1}(s_3)$
- …
- 只需单数组原位更新，可以更快地收敛。之后会用到类似思想。

迭代策略估值伪代码

Iterative policy evaluation

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  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 $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p61*

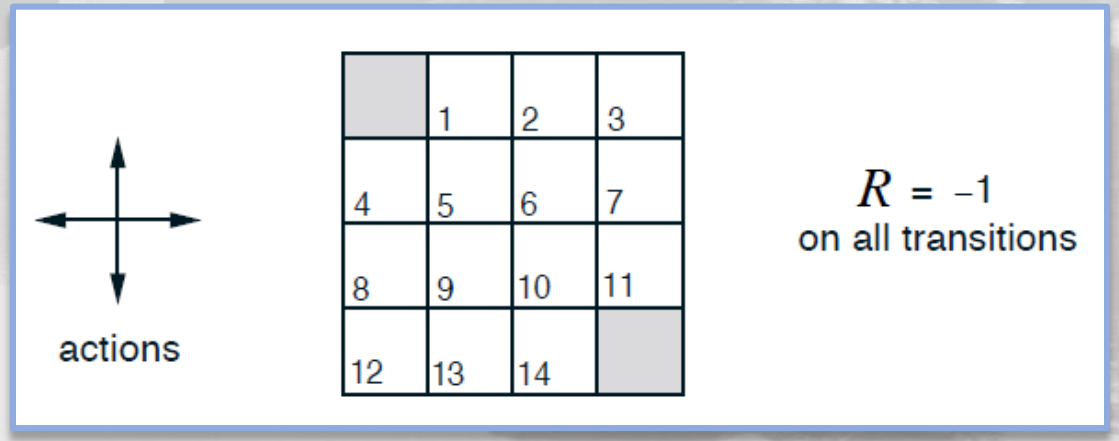
策略估值

- v_π的存在性和唯一性保障条件
  - $\gamma < 1$
  - 或者依据策略π，所有状态最终能保证终止

- 迭代策略估值
  - 在上述存在性和唯一性保障条件下
  - 可以确保序列$\{v_k\}$在$k \to \infty$时能收敛，k是迭代次数，且收敛于v_π。也即在迭代足够多次之后，能得到一个稳定的关于策略π的值函数v_π。
  - 计算依据的是同一个策略π

策略估值计算

- 下图为格点地图，假设agent只能移动到地图上有数字的格子。
- 这个问题的MDP模型
  - 不包含终止状态的有限状态集合$S = \{1,2,\cdots,14\}$。终止状态为agent移动到左上/右下的阴影格。

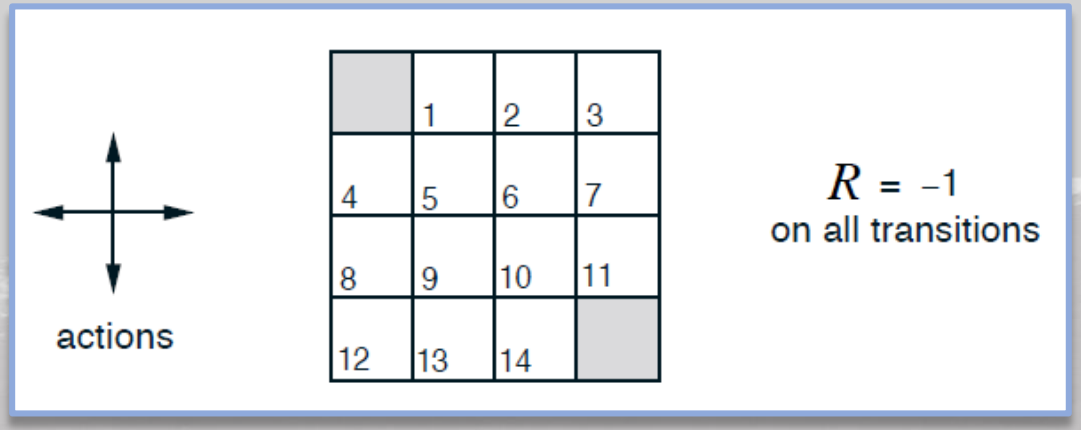


- 这个问题的MDP模型（续）
  - 动作集合$A = \{上, 下, 左, 右\}$
  - 这里我们的目标是熟悉计算，而非确定某种策略。为了计算简便，我们暂时不考虑折旧（$\gamma = 1$）
  - R
    - 达到终状态：1
    - 未达到终状态：-1
  - P转移概率：
    - $p(5, -1|6, 左) = 1$
    - $p(终止, 1|1, 左) = 1$

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p61*

策略估值计算（续）

- 问一：在以上叙述中，如果π是等概率随机策略，那么$q_\pi(11, 下)$和$q_\pi(7, 下)$分别等于多少？
- 记终止状态左上角s_{t1}，右下角s_{t2}



- $q_\pi(11, 下) = p(s_{t2}, 1|11, 下) * r = 1 * 1 = 1$
- $q_\pi(7, 下) = p(11, -1|7, 下) * r = 1 * (-1) = -1$

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p61*

策略估值计算（续）

- 问二：假设在原格点地图标号为13的格子正下方加一个新格，标号为15，并且在状态14上施加动作，其转移为：向左进入第12号格子，向上进入13号，向右进入14号，向下进入15号。假设其他状态转移与原来相同，则对于等概率随机策略π来说，$v_\pi(15)$等于多少？
- $v_\pi(15) =$
  $\pi(上|15) * p(13,-1|15,上) * (-1)$
  $+ \pi(下|15) * p(15,-1|15,下) * (-1)$
  $+ \pi(左|15) * p(12,-1|15,上) * (-1)$
  $+ \pi(右|15) * p(14,-1|15,上) * (-1)$
  $= \frac{-1-1-1-1}{4} = -1$

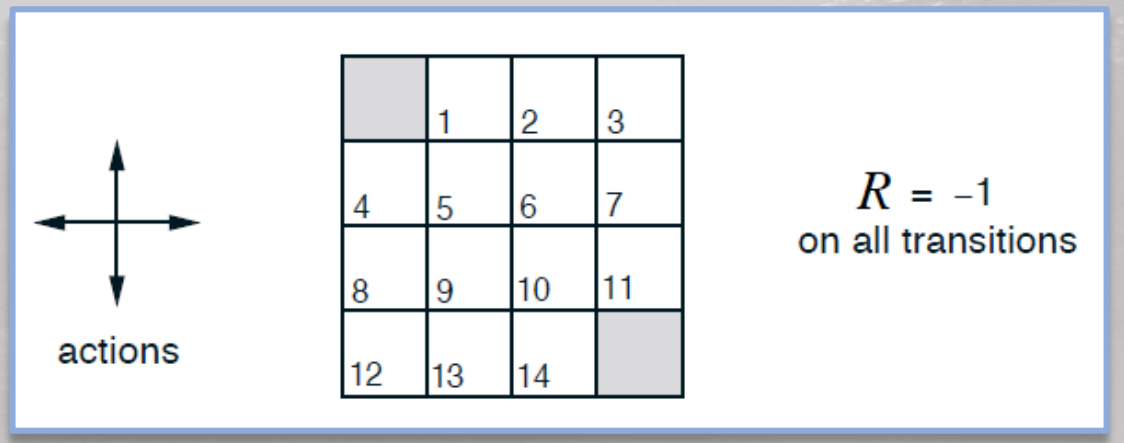
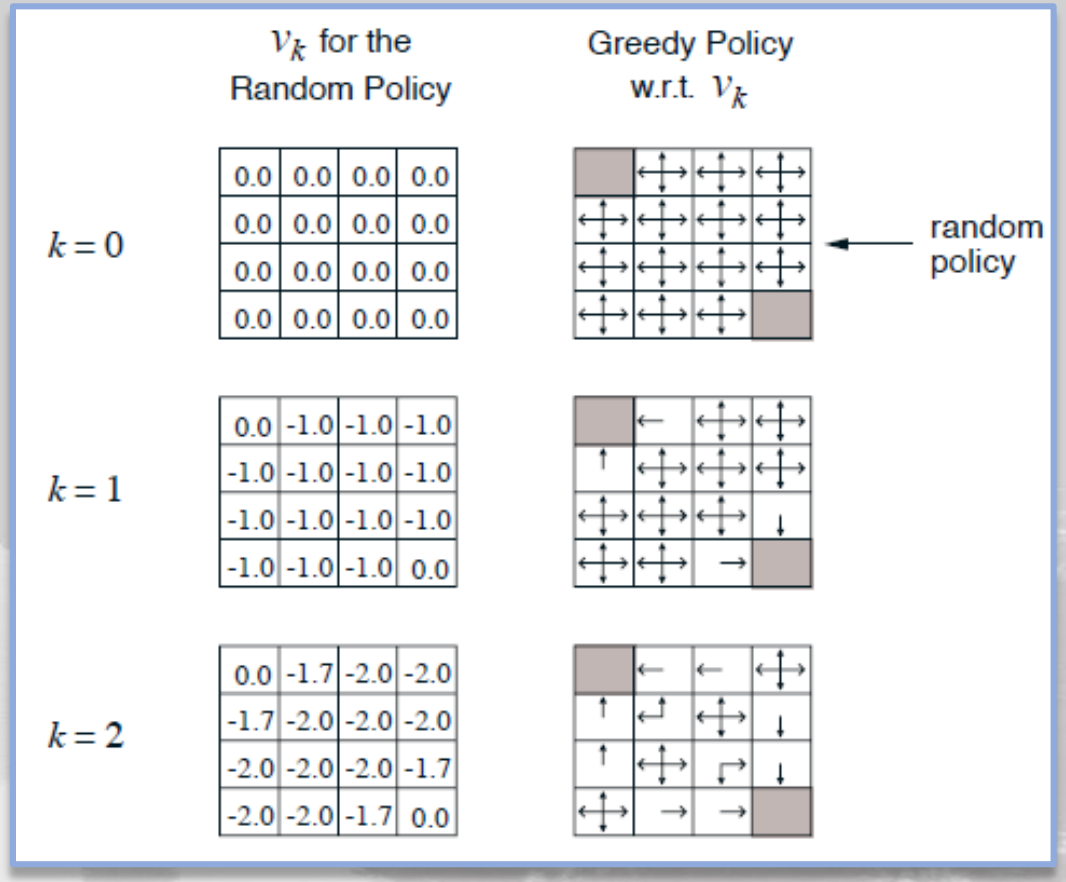


Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p61*

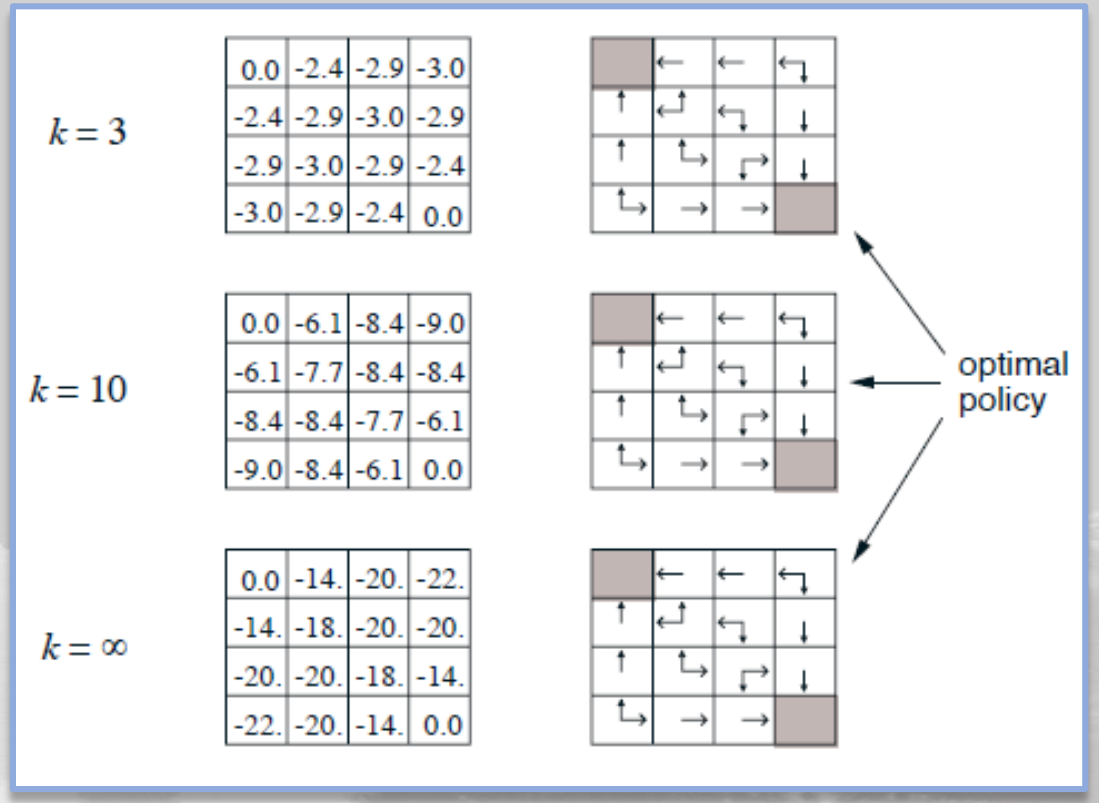
迭代策略估值例子



- 在小的格点世界中的迭代策略评估的收敛。
  - 左侧是随机策略(所有的动作等概率)状态值方程的近似序列。
  - 右侧是对应于值函数估计的贪心策略序列(箭头显示的是所有能使状态值达到最大值的动作)。

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p62*

迭代策略估值例子续



- 在小的格点世界中的迭代策略评估的收敛。
  - 最后的策略只能保证比随机策略稍有进步，但在这个例子中，在第三次迭代后的所有策略，都是最优的。

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p62*

策略优化（Policy Improvement）

- 定义：在原策略基础上，根据原策略计算得到的值函数，贪心地选择动作使得新策略的估值优于原策略，或者一样好
- 策略估值中的计算为寻找最优策略服务

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (6.5)$$

- 如果有$q_\pi(s,\pi'(s)) \geq v_\pi(s)$，那么$\pi'$一定比$\pi$好，至少不劣于。等价于下式。问题：如何用$q_\pi(s,\pi'(s)) \geq v_\pi(s)$推出下式？

$$v_{\pi'}(s) \geq v_\pi(s) \quad (6.12)$$

定义证明$v_{\pi'}(s) \geq v_{\pi}(s)$

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p63*

策略优化

- 贪心选择动作产生新策略

$$\begin{aligned}\pi'(s) &\doteq argmax_a q_\pi(s,a)\\ &= argmax_a \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a]\\ &= argmax_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]\end{aligned} \tag{6.13}$$

- 假设新策略π'和旧策略π一样好，则有$v_\pi = v_{\pi'}$。

$$\begin{aligned}v_{\pi'}(s) &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]\\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi'}(s')]\end{aligned} \tag{6.14}$$

- 与Bellman最优方程相同，因此$v_{\pi'}$一定是最优状态值函数v^*，则π'和π一定都是最优策略

Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P63,64*
Banach, S. "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales." Fund. Math. 3(1922), 133–181

策略迭代（Policy Iteration）

- 定义：交替进行**迭代策略估值**和**策略优化**，在有限步之后找到最优策略与最优值函数
- 以上交替进行会得到策略序列$\{\pi\}$，由于之前已经证明策略优化中新策略一定优于旧策略，或者一样好，故策略序列单调更优
- 有限MDP只有有限种策略，在进行有限步迭代后，一定能收敛得到最优策略与最优值函数

策略迭代伪代码

Policy iteration (using iterative policy evaluation)

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

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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   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

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p65*

杰克的汽车租赁 Jack's Car Rental

- **状态**：两个租车点，每个点最多能停放20辆车
- **动作**：每天下班后，在两个地方之间移动车辆，从一点向另一点移最多5辆车，每移一辆车需要花费$2（多出的车迁回总公司，在本问题中不考虑）
- **奖励**：每辆租出去的车得$10（如果租车点有车）
- **目标**：得到最优策略。折旧因子$\gamma = 0.9$。
- **转移**：还车和租赁随机。
  - 泊松分布，n还车、租赁的概率为$\frac{\lambda^n}{n!}e^{-\lambda}$。
  - 第一个租车点：平均租赁λ =3，平均还车=3
  - 第二个租车点：平均租赁λ =4，平均还车=2

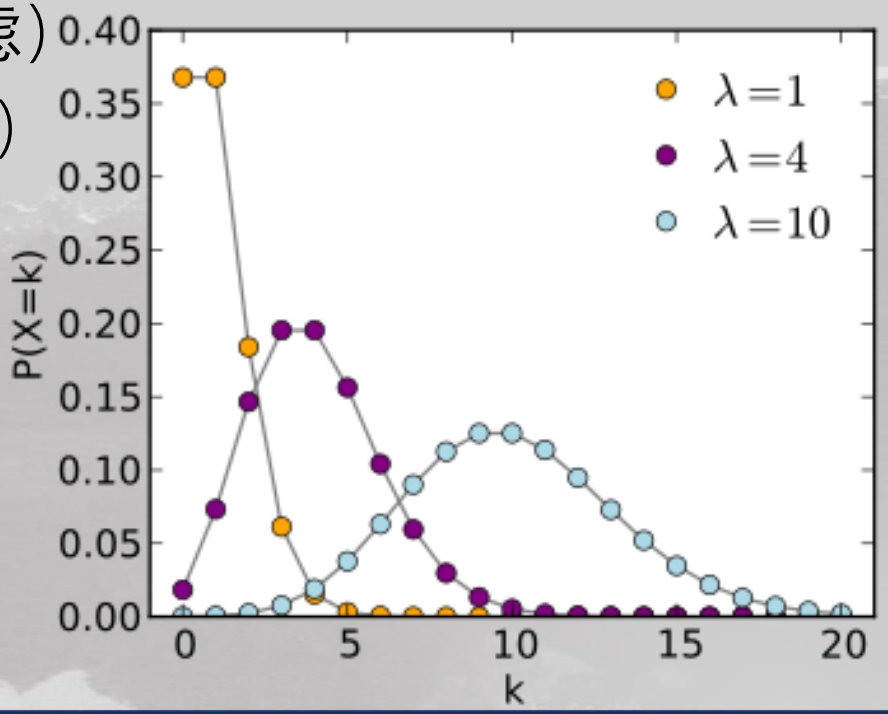


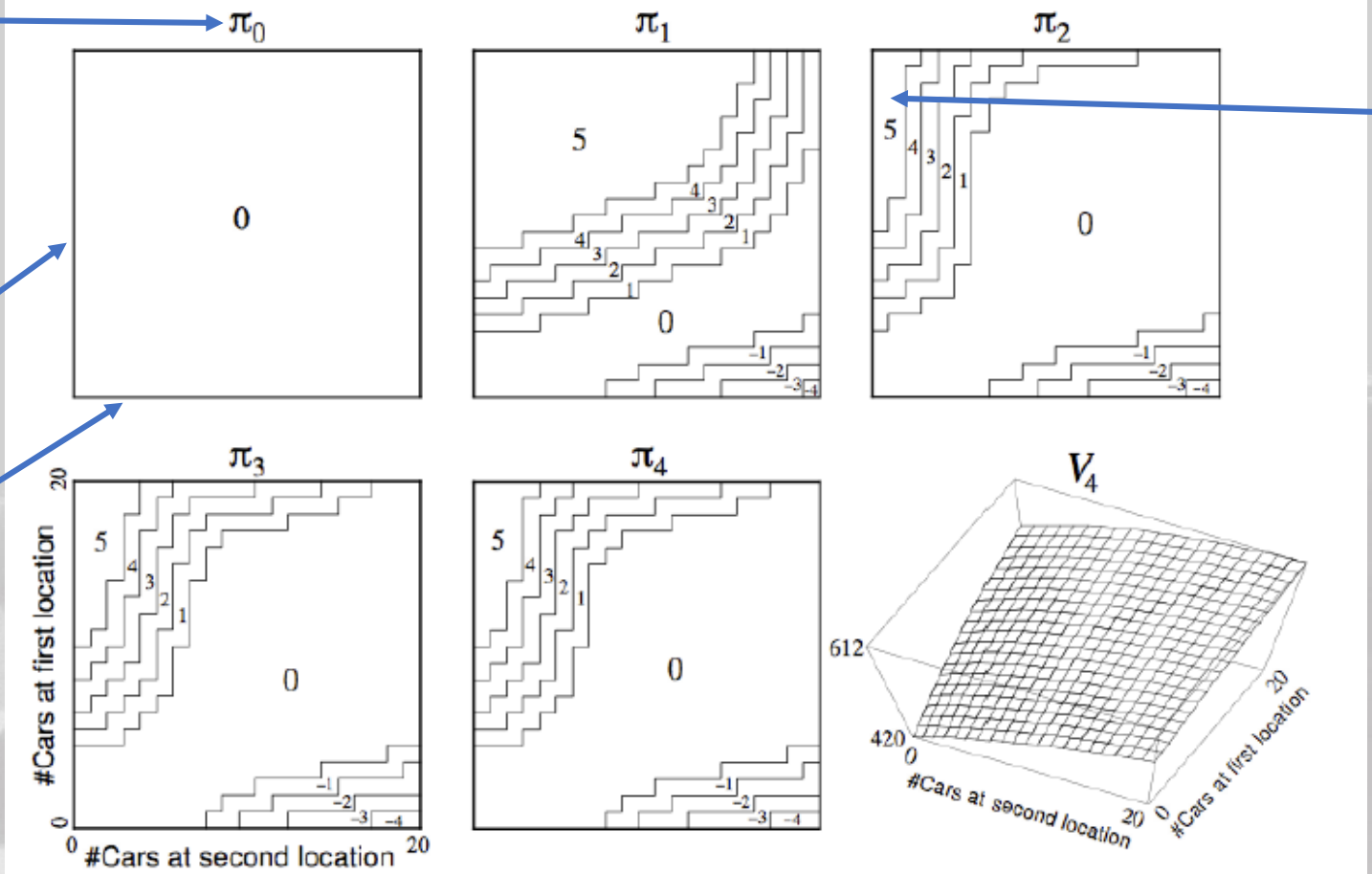
Figure referenced from wiki *"Poisson distribution"*

使用策略迭代分析杰克的汽车租赁问题

每张图上方的π_k，$k = 0,1,2,3,4$，分别表示策略迭代过程中更新的新策略

纵轴表示第一个位置的停车数，从下到上依次增加，大小范围为[0,20]

横轴表示第二个位置的停车数，从左到右依次增加，大小范围为[0,20]



指着5的这个区域表示，当第一个位置和第二个位置的车辆数落在这个区域时，新策略为从第一个位置往第二个位置移动5辆车

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p66*

值迭代（Value Iteration）

- 定义：交替进行**一轮策略估值**和**策略优化**，在有限步之后找到最优策略与最优值函数
- 迭代停止条件：测试$max_{s\in S}|v_{k+1}(s)-v_k(s)|$，当这个量足够小时停止。和迭代策略估值的条件相同。
- 更新规则：结合截断策略估值与策略优化

$$v_{k+1}(s) \doteq \max_a \mathbb{E}_\pi[R_{t+1}+\gamma v_k(S_{t+1})|S_t=s,A_t=a]$$

$$=\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_k(s')] \quad (6.15)$$

Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P67*

值迭代伪代码

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p67*

策略迭代与值迭代对比

策略迭代（Policy Iteration）

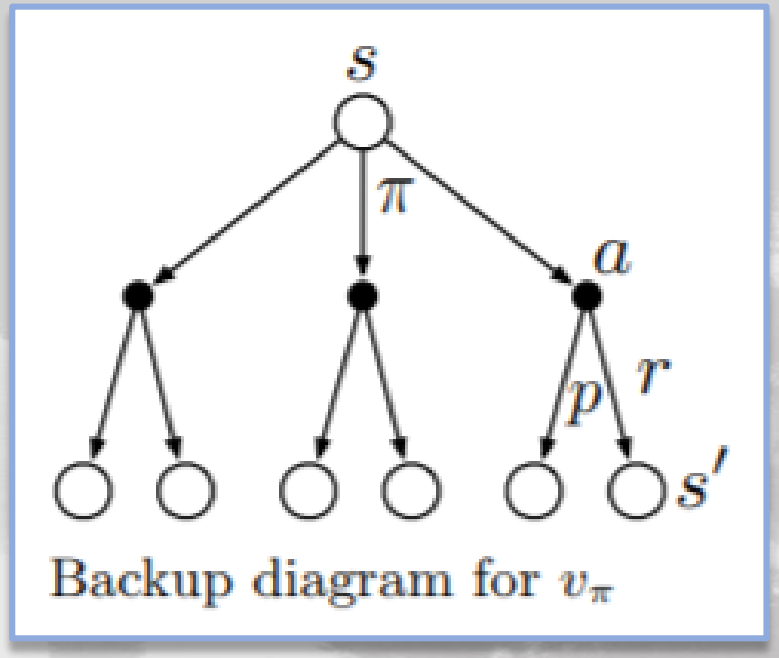
- 策略迭代中的迭代策略估值，会进行多轮策略估值得到稳定的值函数

值迭代（Value Iteration）

- 值迭代中的策略估值只进行一轮，不要求得到稳定的值函数

策略估值和值迭代的回溯图

策略估值（Policy Evaluation）



值迭代（Value Iteration）

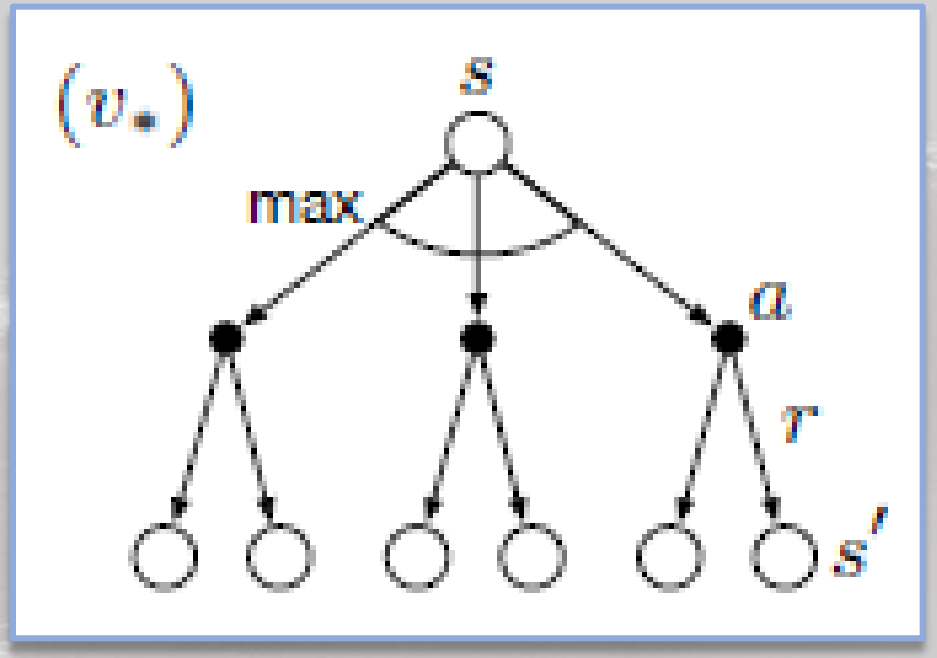


Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P47,51*

迭代方法

迭代方法	策略迭代（Policy Iteration）	值迭代（Value Iteration）
本质	通过不断迭代来优化值函数，从而保证值函数收敛	通过不断迭代来优化值函数，从而保证值函数收敛
过程	迭代策略估值与策略优化交替进行	一轮策略估值与策略优化交替进行
目标	迭代策略估值解决预测问题。策略迭代解决控制问题。	值迭代解决控制问题。
计算	使用Bellman期望方程和贪心	使用Bellman最优化方程

异步动态规划(Asynchronous Dynamic Programming)

- 定义：原位迭代动态规划（单数组），有选择性地优先密集回溯重点状态，不依据状态集合上的同步扫描构建
- 说明
  - 不依据同步扫描构建指的是算法回溯状态值时，不考虑同时、不考虑顺序。一些状态值可能已经回溯多次，而其他的连一次也没有。
  - 但是为了正确地收敛，一定要持续回溯所有状态值
- 计算过程
  - 已有$v_{k_1}(s_1), v_{k_2}(s_2), \cdots, v_{k_n}(s_n)$，其中$k_1, k_2, \cdots, k_n$不一定相等
  - 用以上值有选择地计算，比如在这一步中只计算更新$v(s_2)$的值
  - 计算得到$v_{k_1}(s_1), v_{k_2+1}(s_2), \cdots, v_{k_n}(s_n)$

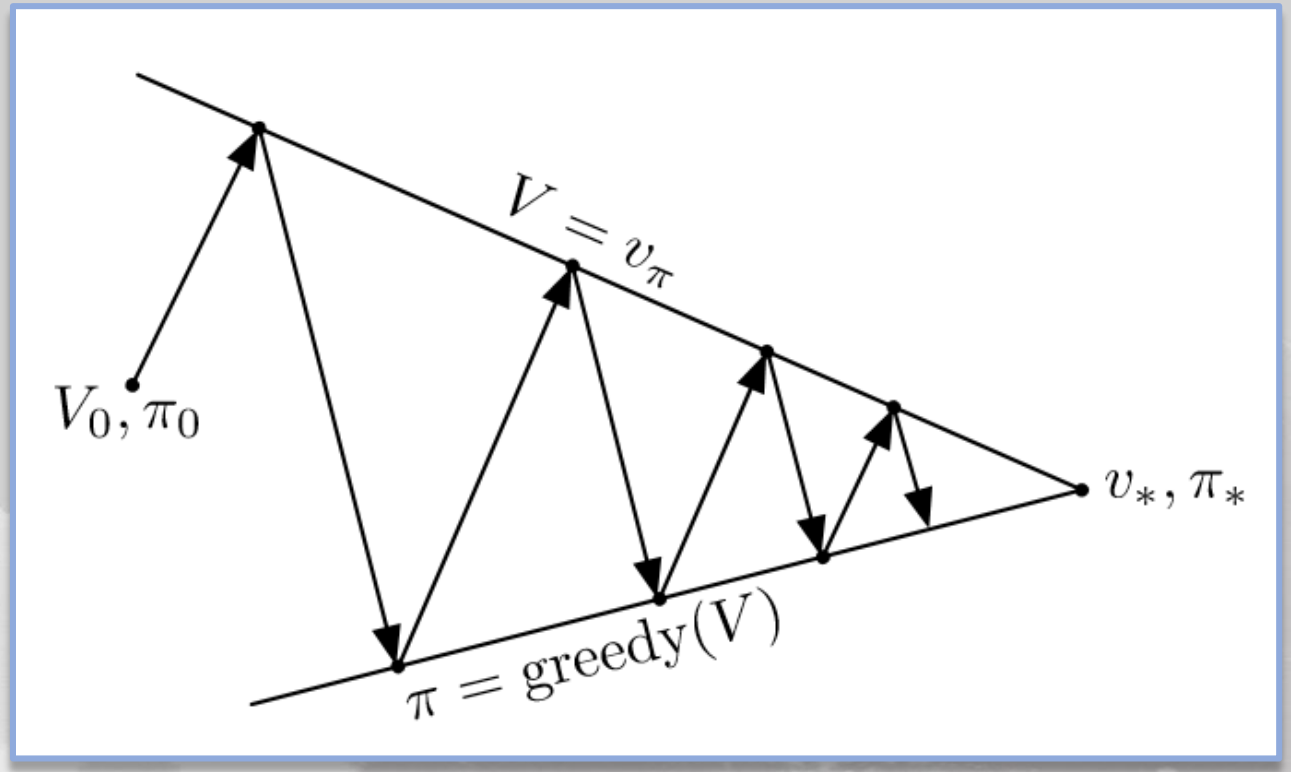
异步动态规划

- 收敛条件
  - $0 \leq \gamma < 1$
  - 则当给定所有状态在序列$\{s_k\}$中出现无穷次，能保证异步收敛到v_*
- 优点：使得计算和实时交互的混合更加简单。
  - 在运行迭代动态规划算法的同时令一个agent也在MDP模型中与环境交互
  - agent的经历（experience）能用来决定动态规划算法优先回溯哪些状态，比如状态集内和agent最相关的一部分状态
  - 来自动态规划算法的最新的值和策略信息能指导agent的决策

广义策略迭代（Generalized Policy Iteration）

- 定义：策略估值和策略优化交互的一般思想，与过程的粒度等其他细节无关。几乎所有强化学习方法都能用GPI描述。
- 不同粒度表现
  - 策略迭代中为迭代策略估值，进行同步计算得到稳定的值函数
  - 值迭代中为一轮策略估值，进行同步计算得到不一定稳定的值函数
  - 异步动态规划选择性地计算，进行异步计算
- GPI可以被看作拮抗与协同
  - 拮（jie2）抗：贪心选择动作产生新策略，会使得值函数发生改变
  - 协同：重新计算值函数使得值与策略相一致
  - 拮抗与协同交替进行从而得到稳定的解决方案：最优策略和最优值函数

广义策略迭代



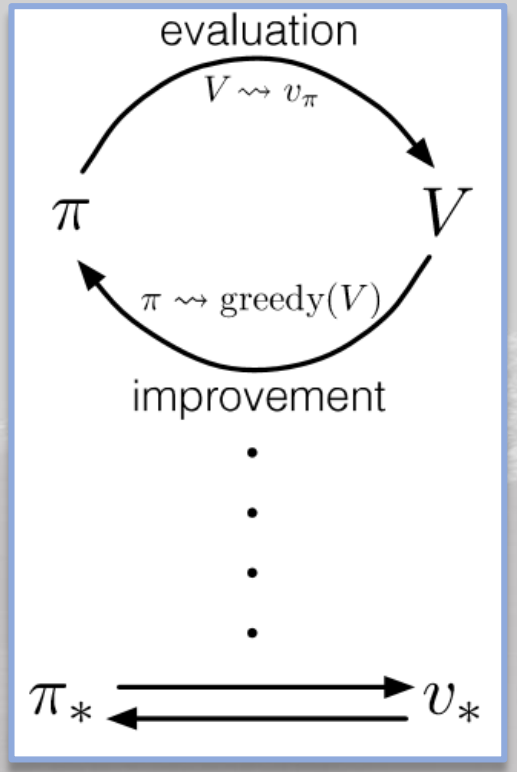


Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P70*

动态规划局限性

- 适用动态规划的问题必须有最优化子结构和无后效性

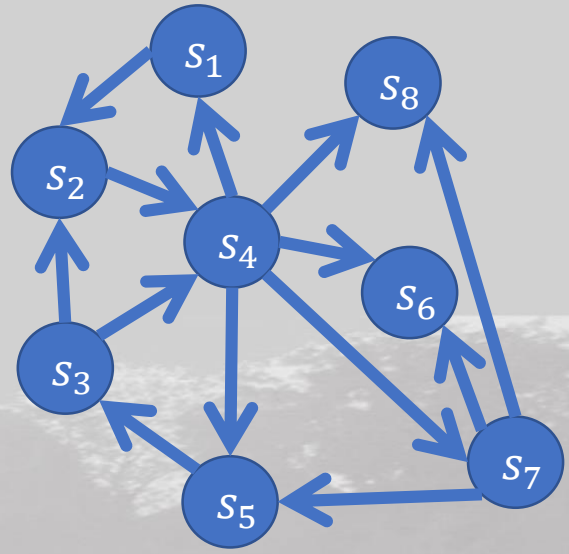
动态规划效率（Efficiency of Dynamic Programming）

- 对于有折旧因子的DMDP(Discounted MDP)
  - 当折旧因子固定时，策略迭代已被证明是强多项式时间
  - 值迭代被证明不是强多项式时间
- **强多项式时间**定义：有形如(s, a)的状态动作对m对，$s \in S$，$a \in A(s)$。如果算法有一个关于运算数的上界，是关于m的多项式，且对于所有MDP都满足这个上界，那么这个算法是强多项式时间的。
- 强多项式时间算法指问题运算时间不取决于输入数据大小，即计算时间不会因为数据很大而变长。

Formula referenced from Grötschel, Martin; László Lovász; Alexander Schrijver (1988). "Complexity, Oracles, and Numerical Computation". Geometric Algorithms and Combinatorial Optimization. Springer. ISBN 0-387-13624-X. P32

动态规划效率（Efficiency of Dynamic Programming）

- 策略迭代时间复杂度
  - 对于**一般MDP**，上界是$O(k^n/n)$。其中n是状态数，k是在一个状态下可以采取的最大动作数。如果将状态看作点，动作看作点与点之间的有向连线，那n就是点的数量，k就是从点连出边的最大数。
  - 对于**有折旧因子的DMDP**(Discounted MDP)，$0 \leq \gamma < 1$，迭代次数的上界为$\frac{n^2(k-1)}{1-\gamma} \cdot \log(\frac{n^2}{1-\gamma})$，每次迭代使用的算术操作数是$O(n^2k)$。最差情况下的时间复杂度为$O(n^4k^2 \log n)$
  - 对于有折旧因子的DMDP，策略迭代已经被证明是强多项式时间的。



上图中，有$n = 8$个状态，其中从s_4状态连出的边最多，为$k = 5$。

Formula referenced from Ye, Yinyu. "The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate." Mathematics of Operations Research36.4 (2011): 593-603.

动态规划效率（Efficiency of Dynamic Programming）

- 值迭代时间复杂度
  - 对于有折旧因子的DMDP，值迭代已被证明不是强多项式时间
  - 迭代次数上界为

$$\frac{n^2 k L(P, c, \gamma) \log\left(\frac{1}{1-\gamma}\right)}{1-\gamma}$$

  - 其中$L(P, c, \gamma)$是DMDP输入数据在线性规划形式下的bit数。
  - 每次迭代的时间复杂度是$O(n^2 k)$

Formula referenced from Ye, Yinyu. "The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate." Mathematics of Operations Research36.4 (2011): 593-603.

策略迭代和值迭代哪一个收敛更快？

- 实际上不能很好地区分策略迭代和值迭代哪个更好。这些方法在实际运用的最差情况下，运行时间都要比理论上的值要小，两个收敛地都比理论值要快。
- 直观来看，策略迭代中用精确的估值来进行策略优化，而值迭代则提高了策略优化的运行次数比重，并不能脱离问题以及初始值来比较二者谁更优。
- 此外，二者对初始值的依赖性较大。对于值迭代，选取一个“好的”初始值将大大加快收敛速度。这可能会成为改进值迭代的一个思路。

动态规划效率（Efficiency of Dynamic Programming）

- 动态规划用于解决MDP
  - 在最差情况下的时间复杂度是关于状态数n和动作数k的多项式
  - 在状态数和动作数非常大的问题中不适用
  - 存在维数灾难——状态数通常随着状态变量数呈现指数型增长。但是状态集过大是问题本身的固有性质，并不是使用动态规划方法而产生的。
- 其他方法
  - 策略空间直接搜索：直接搜索的总策略数为k^n
  - 线性规划：能实际运用的状态数上限比动态规划还要小得多，约为1:100

课后思考：二人对抗取硬币游戏

- 给定n堆排成一行的硬币，从头到尾各放有$v_1, v_2, \ldots, v_n$个硬币，其中n是偶数。我们采取回合制来玩这个游戏。在每一回合中，玩家可以选择两边的硬币堆中的任意一堆。比如对于先开始的玩家，他的选择只能是，取走v_1还是v_n的硬币；如果他选择取走v_1，那另一个玩家的选择就是取走v_2还是v_n的硬币。
- 作为玩家的你，要想方设法取得最大可能数额的硬币。①两人一组尝试一下。②只有一个人时，假设你是第一个取走硬币的人，思考一下如何取走最大可能数额的硬币。（注：此时应考虑对手和你一样，才思敏捷充满智慧）
- 两组简单数据：①5 3 7 10；②8 15 3 7

二人对抗取硬币游戏分析

- 根据游戏描述，我们有$v_1, v_2, \ldots, v_n$个硬币，其中n是偶数。先来写动态转移方程：假设$dp[m][k]$表示的是从第m堆到第k堆的硬币，第一个人按以上方式取得的最大值，有$k-m$是正的偶数。那么最简单的想法是

$$dp[m][k] = \max(\max(v_m, v_k) + dp[m+1][k-1],\\ v_m + dp[m+2][k],\\ v_k + dp[m][k-2])$$

- 但是考虑到对手和你一样聪明，他也会动态规划！所以还要继续考虑。

二人对抗取硬币游戏分析续

- 考虑到$\sum_{i=m}^{k} v_i$是固定的，若取走第m堆，那么对手就会想方设法让你取到$\min(dp[m+2][k], dp[m+1][k-1])$，这样他就可以尽可能多的拿到硬币。所以我们的动态转移方程就变成了

$$dp[m][k] = \max(v_m + \min(dp[m+2][k], dp[m+1][k-1]),$$
$$v_k + \min(dp[m+1][k-1], dp[m][k-2]))$$

参考文献

- Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Cambridge: MIT press, 2017: 59-71
- Eddy, S. R., What is dynamic programming?, Nature Biotechnology, 22, 909-910 (2004).
- Bellman, Richard. "On the theory of dynamic programming." *Proceedings of the National Academy of Sciences* 38.8 (1952): 716-719.
- Bellman R. The theory of dynamic programming[J]. Bulletin of the American Mathematical Society, 1954, 60(6): 503-515.
- https://people.ricam.oeaw.ac.at/d.kalise/files/Richard_Bellman_Eye_of_the_Hurricane.pdf

参考文献

- Ye, Yinyu. "The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate." *Mathematics of Operations Research*36.4 (2011): 593-603.
- Hollanders, Romain, Jean-Charles Delvenne, and Raphaël M. Jungers. "The complexity of policy iteration is exponential for discounted markov decision processes." *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012.
- Feinberg, Eugene A., and Jefferson Huang. "The value iteration algorithm is not strongly polynomial for discounted dynamic programming." *Operations Research Letters* 42.2 (2014): 130-131.

参考文献

- Grötschel, Martin; László Lovász; Alexander Schrijver (1988). "Complexity, Oracles, and Numerical Computation". Geometric Algorithms and Combinatorial Optimization. Springer. ISBN 0-387-13624-X.
- Banach, S. "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales." Fund. Math. 3(1922), 133–181
- https://en.wikipedia.org/wiki/Banach_fixed-point_theorem（压缩收敛到唯一不动点）