北京大学

马尔可夫决策过程和动态规划

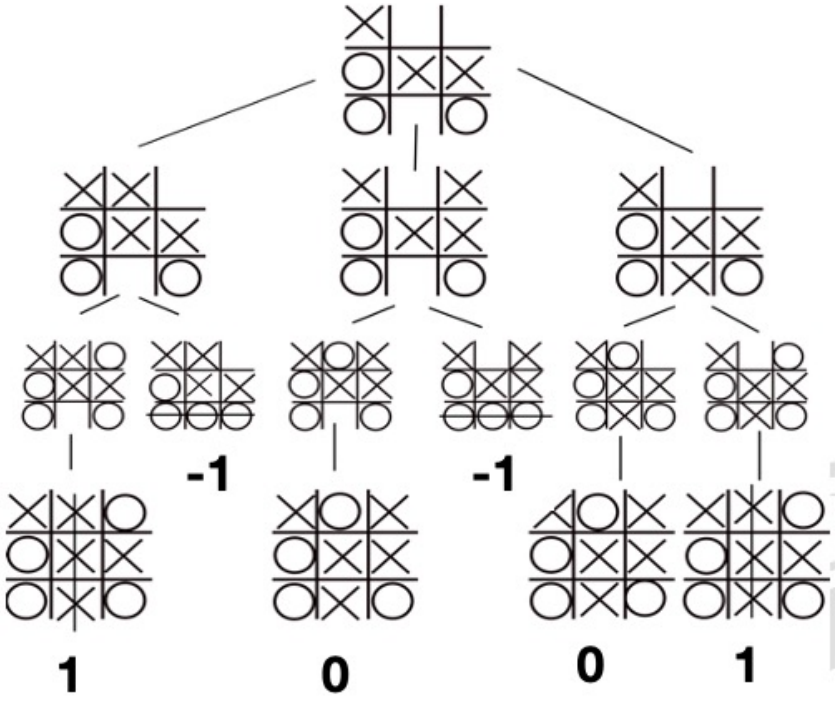
MDP and Dynamic Programming

内容主要来自：Reinforcement Learning An Introduction Richard S. Sutton Andrew G. Barto

主讲人：李文新 2023年春

本讲主写人 李文新

上节课讲了三件事

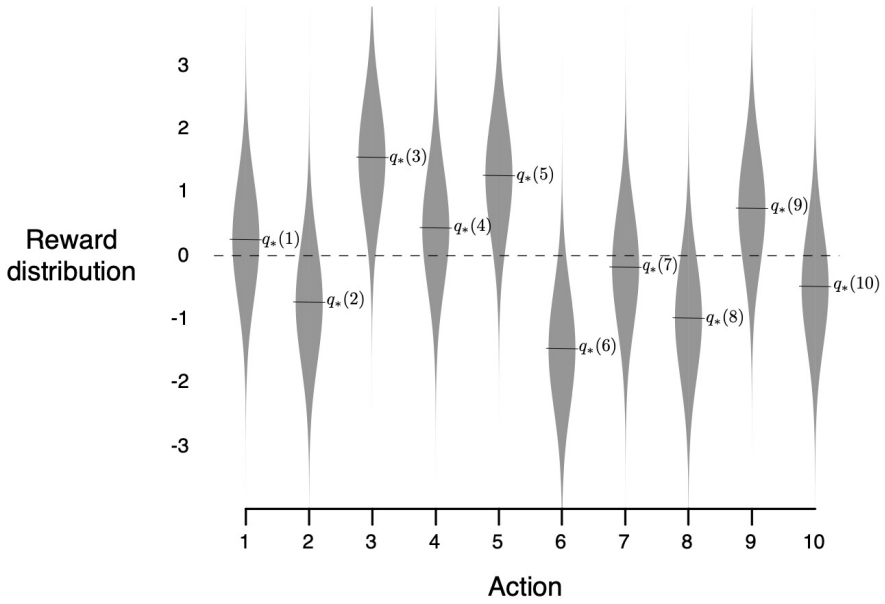


井字棋

第一件：

我们通过井字棋的例子，展示了如何应用强化学习方法，在不对敌人做任何假设的情况下，通过反复对敌演练，习得一个应敌之策。

上节课讲了三件事



多臂老虎机问题

第二件：

通过多臂老虎机问题，展示了强化学习面临的最大挑战是如何平衡探索和利用的关系。一方面我们要探索未知世界以期发现更好的收益路径，另一方面还要控制风险，采用已知收益较好的动作使得累积收益有一定保障。

上节课讲了三件事

第三件：给出了强化学习方法的问题模型和解的形态。

环境（问题模型）

1. 初始状态 S_0(state)
2. 动作 A(action)
3. 状态转移 P(transition)
4. 终止状态 S_T(terminate state)
5. 奖励 R(reward)

概率化

智能体

1. 策略 π (A|S)
2. 目标（问题的解）

最大化期望累积收益G

内容提要

1. 马尔可夫决策过程 Markov decision process (MDP)
   - $MDP < S, A, P, R, \gamma >$
   - 马尔可夫性
   - 贝尔曼方程
2. P, R已知，用动态规划方法求解最优策略
   - 策略迭代（Policy Iteration）
   - 值迭代（Value Iteration）
   - 广义策略迭代（Generalized Policy Iteration）
3. P, R未知，用采样方法逼近最优策略
   - Bootstrap
   - 蒙特卡洛学习
   - 时序差分学习

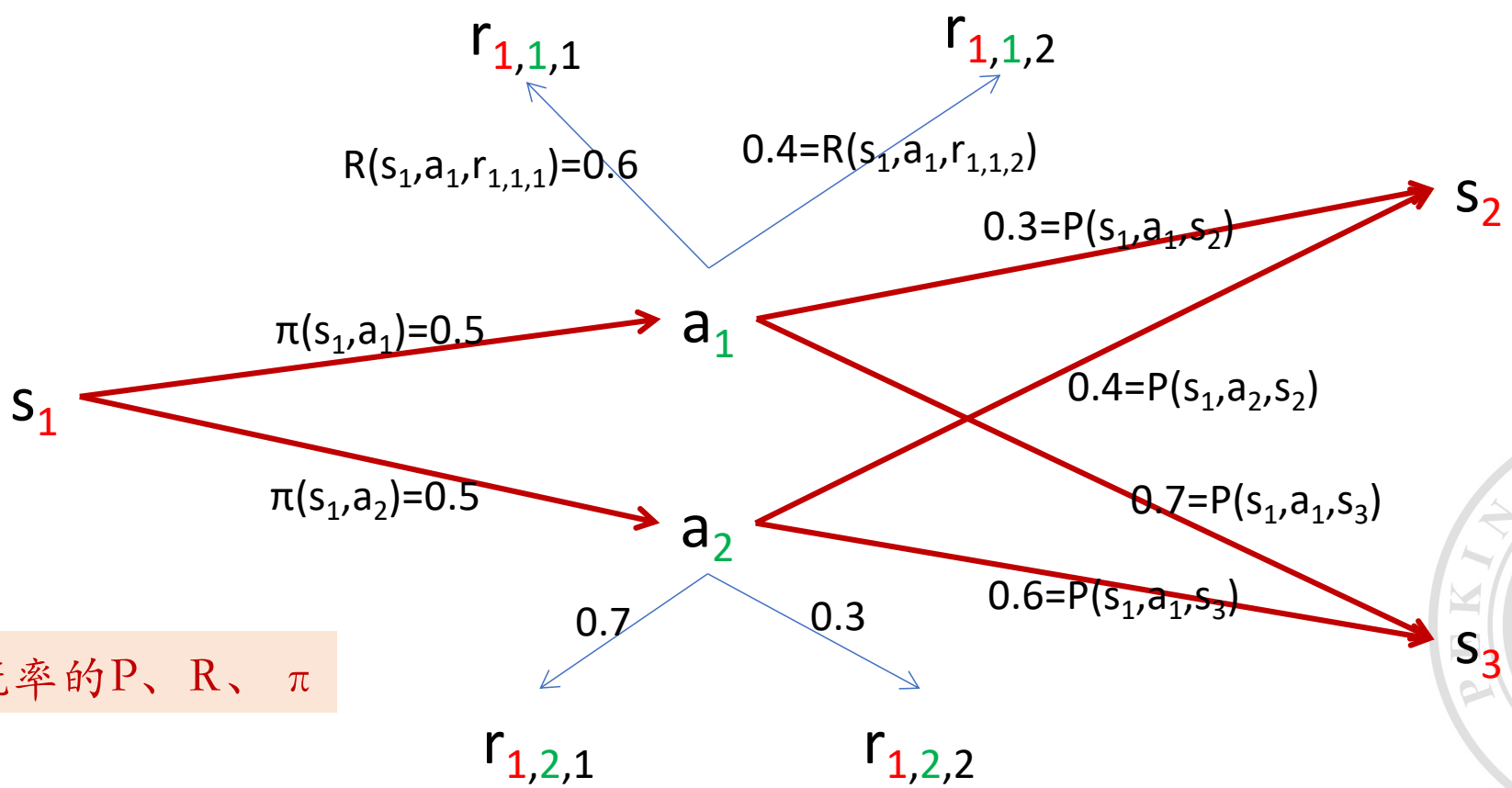
马尔可夫决策过程(MDP)

- 马尔可夫决策过程 Markov decision process (MDP)
  1. 状态集合：S
  2. 动作集合：A
  3. 状态转移函数 P: <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
  4. 奖励函数R: <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是奖励

马尔可夫性：S_{t+1}只和S_t有关和S_{t-1},S_{t-2},S_{t-3},…无关

- 在当前状态S下，状态转移模型P，奖励函数R都与S之前的状态和动作无关。
- 有限马尔可夫决策过程是有限的状态S、动作A和奖励R集合
- 本次课程限定在有限 MDP问题，但是方法和思路是更普适的，可以拓展到无限情况

状态、动作、状态转移和奖励示意图



有概率的P、R、 π

几个概念

- G_t是时间步t以来的累积收益值

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

- 状态价值函数 $V_\pi(s)$

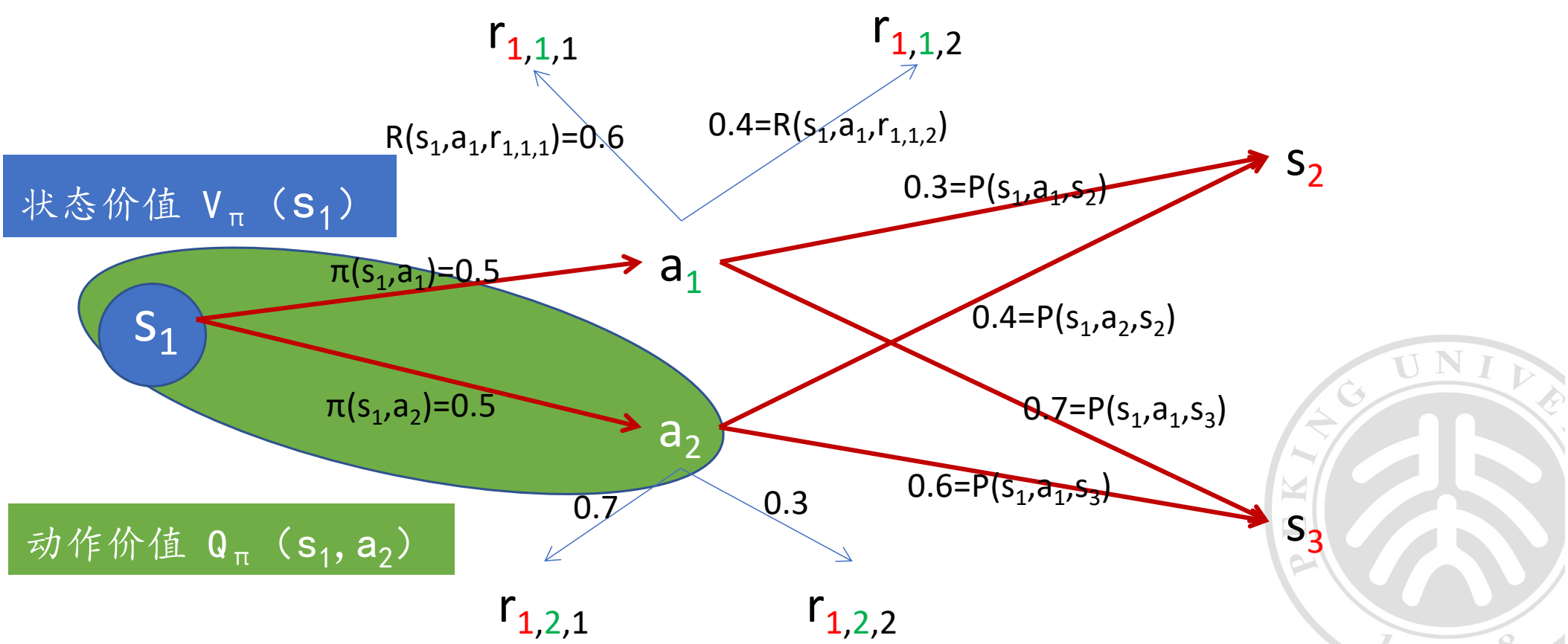
$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$

- 动作价值函数 $Q_\pi(s, a)$

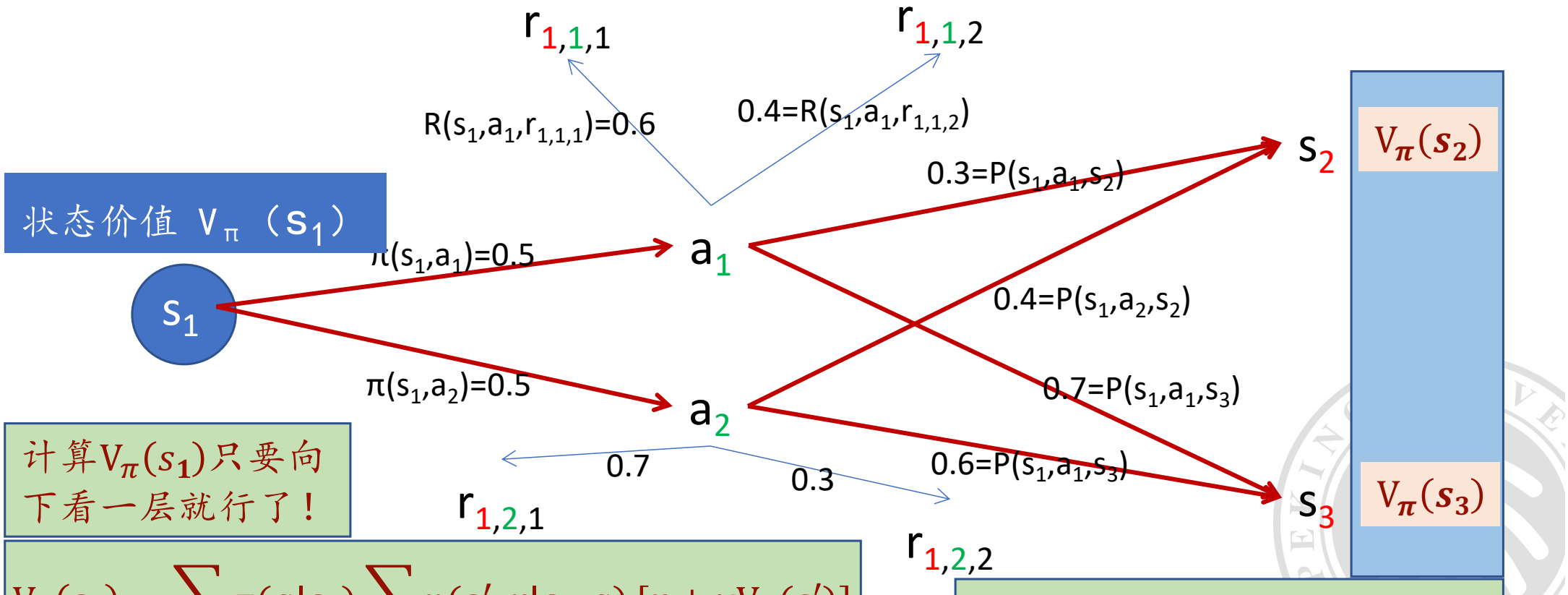
$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s, A_t = a]$$

Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P44,46*

状态价值V_π和动作价值q_π



马尔可夫性意味着什么？

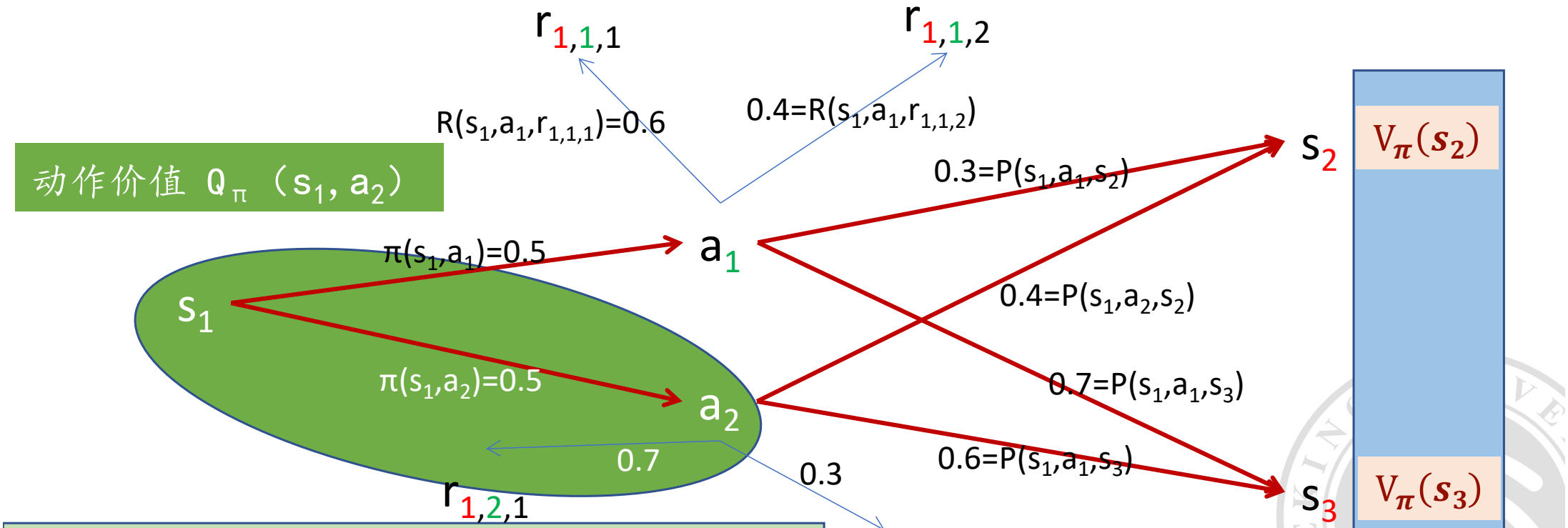


计算$V_\pi(s_1)$只要向下看一层就行了！

$$V_\pi(s_1)=\sum_{a\text{加权以求期望}} \pi(a|s_1)\sum_{s',r\ \text{一共有4种组合}} p(s',r|s_1,a)\,[r+\gamma V_\pi(s')]$$

$V_\pi(s_2)$和$V_\pi(s_3)$ 与s_1没有关系

马尔可夫性意味着什么？



$$Q_\pi(s_1, a_2) = \sum_{s',r} p(s', r|s_1, a_2)[r + \gamma V_\pi(s')]$$

$V_\pi(s_2)$和$V_\pi(s_3)$ 与s_1没有关系

Bellman期望方程

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

$$Q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')]$$

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

$$V_\pi(s) = \sum_a \pi(a|s)\, Q_\pi(s,a)$$

马尔可夫性保证这个递推方程成立

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

状态价值和最优策略

定义：$V^*(s) = \max_{\pi} V_\pi(s)$ 所有可能的策略下，对当状态S下最优的状态价值

- 状态价值（等价于最优状态价值）就是从状态 S 出发所能获得的最大累积收益。
- 最优策略π^*是从状态S出发执行该策略可以获得状态价值的策略。

状态价值只有一个，最优策略可以有多个纯策略或者最优纯策略的任意组合。

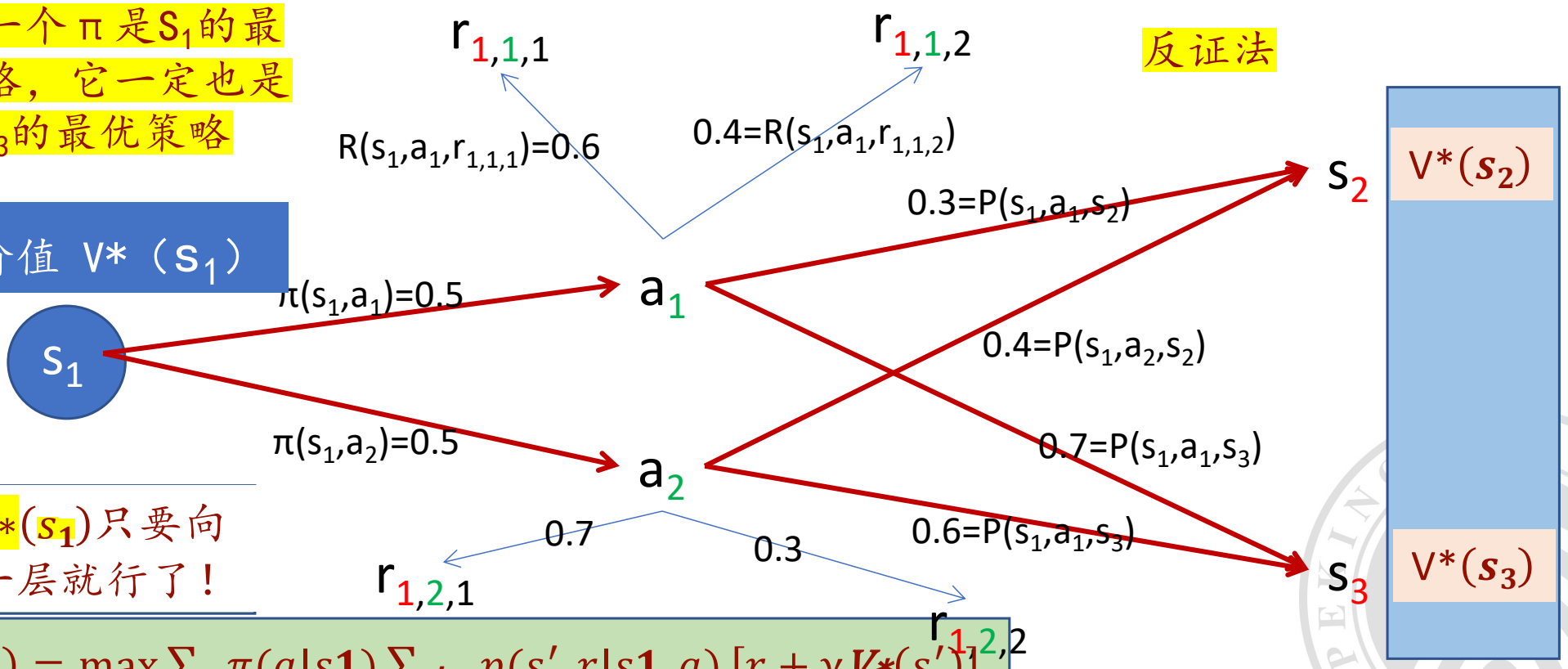
Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P59,60,63*

马尔可夫性意味着什么？

如果一个π是S_1的最优策略，它一定也是S_2和S_3的最优策略

反证法

状态价值 V*（S_1）



计算V*(s_1)只要向下看一层就行了！

$$V^*(s1) = \max_{\pi} \sum_a \pi(a|s1) \sum_{s',r} p(s',r|s1,a)\,[r + \gamma V^*(s')]$$

$$= \max_{a} \sum_{s',r} p(s',r|s_1,a)\,[r + \gamma V^*(s')]$$

V*(s_2)和V*(s_3) 与S_1没有关系

动作价值和最优策略

定义：$\boldsymbol{Q}^*(s,a) = \max_{\pi} \boldsymbol{Q}_{\pi}(s,a)$ 所有可能的策略下，对S和A下最优的动作价值

- 动作价值（等价于最优动作价值）就是从状态 S 出发，采取动作a，所能获得的最大累积收益。
- 最优策略是从状态S出发，采取动作a，之后执行该策略可以获得动作价值的策略。

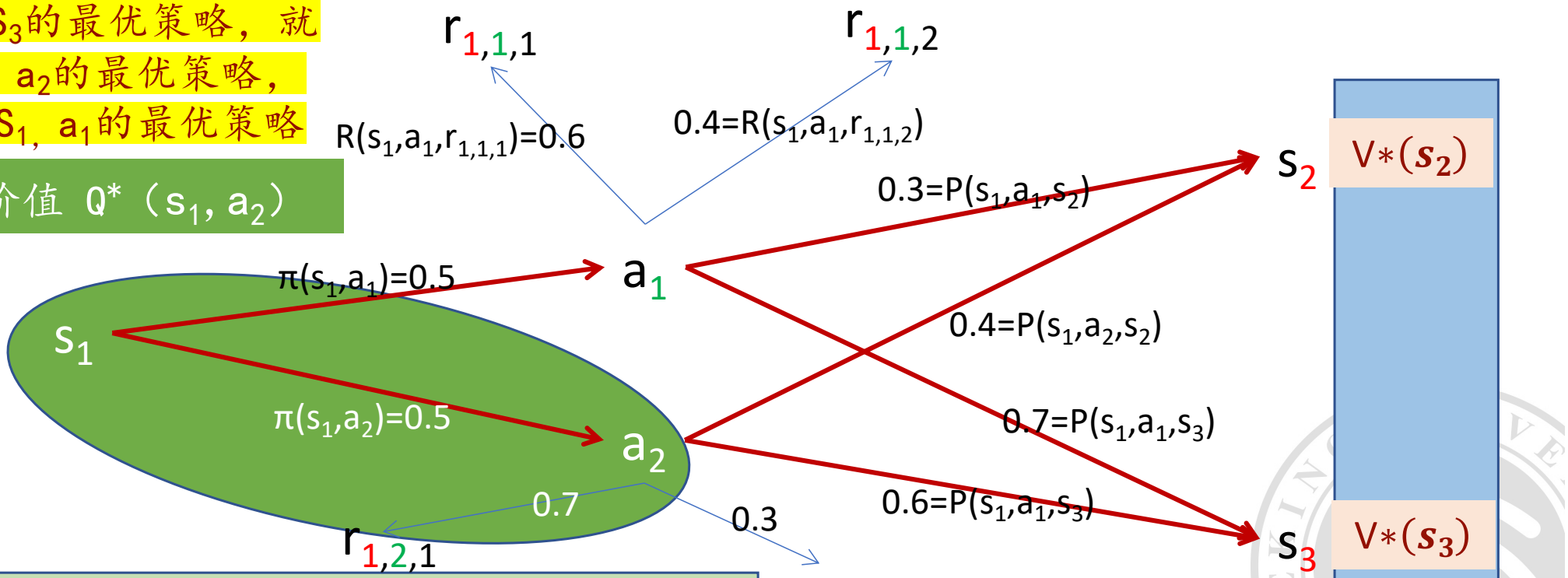
动作价值只有一个，最优策略可以有多个纯策略或者最优纯策略的任意组合。

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

马尔可夫性意味着什么？

S_2和S_3的最优策略，就是S_1, a_2的最优策略，也是S_1, a_1的最优策略

动作价值 $Q^*(s_1, a_2)$



$$Q^*(s_1, a_2) = \sum_{s',r} p(s', r|s_1, a_2)[r + \gamma V*(s')]$$

$V*(s_2)$和$V*(s_3)$ 与S_1没有关系

Bellman最优方程

$$V^*(s) = \max_\pi V_\pi(s)$$
$$Q^*(s,a) = \max_\pi Q_\pi(s,a)$$
$$\Rightarrow$$
$$V^*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V^*(s')] = \max_a Q^*(s,a)$$
$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q^*(s',a')]$$
$$\forall s \in S, a \in A(s), s' \in S^+$$

最优价值只有一个，最优策略可以有多个纯策略或者最优纯策略的任意组合

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

内容提要

1. 马尔可夫决策过程 Markov decision process (MDP)
   - $MDP < S, A, P, R, \gamma >$
   - 马尔可夫性
   - 贝尔曼方程
2. P, R已知，用动态规划方法求解最优策略
   - 策略迭代（Policy Iteration）
   - 值迭代（Value Iteration）
   - 广义策略迭代（Generalized Policy Iteration）
3. P, R未知，用采样方法逼近最优策略
   - Bootstrap
   - 蒙特卡洛学习
   - 时序差分学习

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论文里对动态规划的说明(1954)

- 动态规划理论一开始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，它的状态(state)被描述为一组状态变量。决策(decisions)等价于状态变量的转移(transformations)。决策与状态转移是互相独立的，在前一段决策的结果的基础上实施后面的决策，整个过程的目标是最大化最终状态参数的相关函数。

Bellman R. *The theory of dynamic programming*[J]. Bulletin of the American Mathematical Society, 1954, 60(6): 503-515.

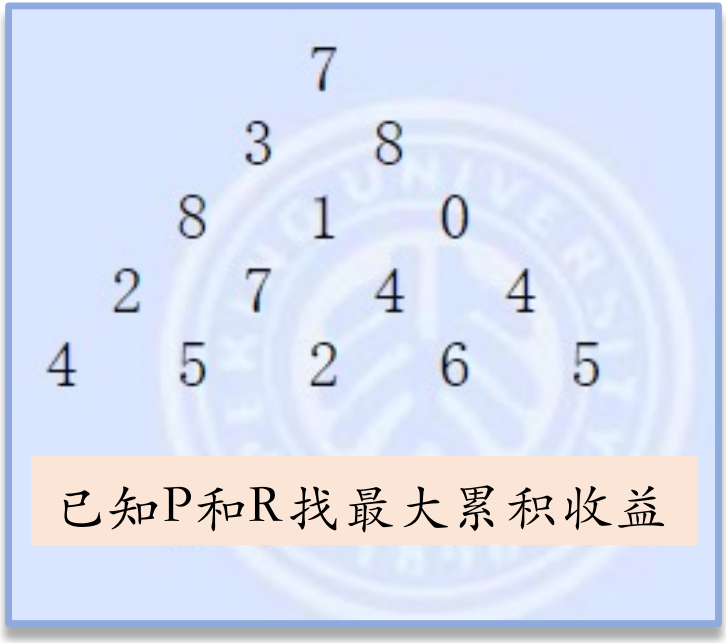
Bellman论文里对动态规划的说明续(1954)

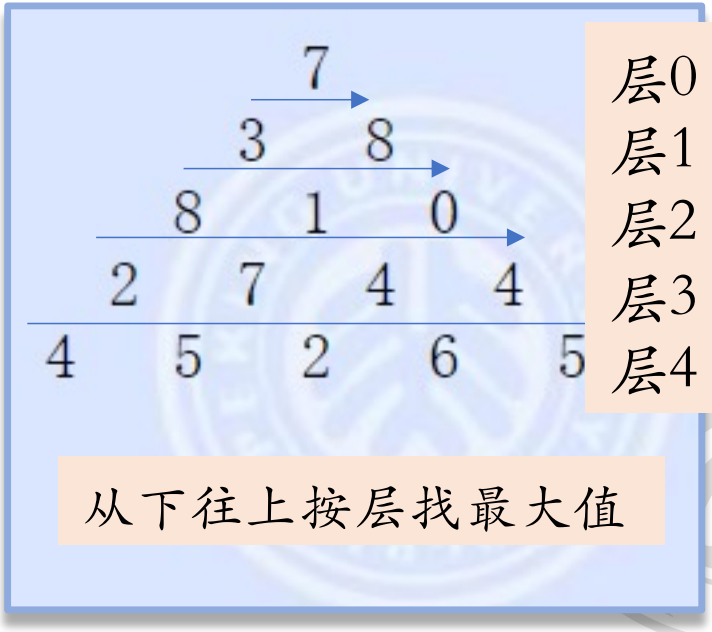
- 考虑一个中间状态。做决策时，我们并不关心如何从初状态到中间状态的过程，之后的决策都只从这个中间状态开始。这就叫做无后效性（马尔可夫性产生的）
- 最优原则(principle of optimality)：
  - 一个最优策略有这样的性质——无论初始状态和最初的几个决策是什么，剩余决策一定构成一个与之前决策产生的状态相关的一个最优策略。
  - 形式化描述：一个策略$\pi(a|s)$能达到状态s下的最优值$v_\pi(s) = v_*(s)$，当且仅当：对于从状态s出发可达到的任意状态s'，策略π能达到新状态s'的最优值$v_\pi(s') = v_*(s')$。

Bellman R. *The theory of dynamic programming*[J]. Bulletin of the American Mathematical Society, 1954, 60(6): 503-515.

数字三角形

- 在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。

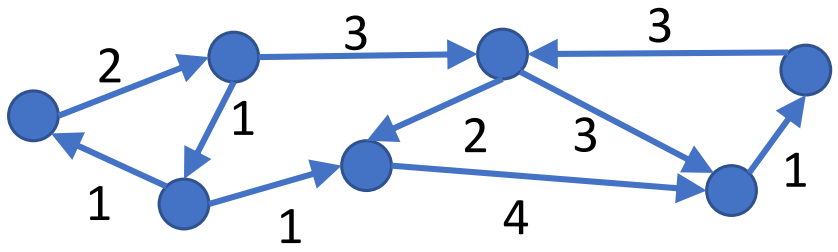




Paragraph referenced from poj 1163 *"The Triangle"*

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（起点）。现在要计算从源（起点）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。（n个点）
- 最多n-1个阶段，第一次得到最短路为1步的点的最短路；依次类推，最多n-1步；如果还能再减小，就是有和为负的环。




```
for 每一个结点v ∈ V
    Distance(v) := +∞
Distance(s) := 0        ← 源到源距离为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
```

s：源/起点
v、u：所有顶点
v和u是的相邻顶点

记录s到v的最短路径

内容提要

1. 马尔可夫决策过程 Markov decision process (MDP)
   - $MDP < S, A, P, R, \gamma >$
   - 马尔可夫性
   - 贝尔曼方程

策略估值（Policy Evaluation）
策略提升（Policy Improvement）
策略迭代（Policy Iteration）

2. P, R已知，用动态规划方法求解最优策略
   - 策略迭代（Policy Iteration）通过策略改进得到最优策略
   - 值迭代（Value Iteration）通过求最优状态价值得到最优策略
   - 广义策略迭代（Generalized Policy Iteration）
3. P, R未知，用采样方法逼近最优策略
   - Bootstrap
   - 蒙特卡洛学习
   - 时序差分学习

1 策略估值（Policy Evaluation）

- 定义：对于任意策略π，计算在此策略下的状态值函数V_π

- （1）策略估值中Bellman方程的赋值形式

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \quad (1)$$

- （2）迭代更新规则：v_π是这个更新规则的不动点

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \quad (2)$$

K代表阶段，一个阶段里所有状态都有一个估值

注：(1)式中的v_π下标表示是对策略π的**准确稳定**状态值函数，(2)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**

Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P60*

迭代策略估值（iterative policy evaluation）

- 完全回溯（full backup）：

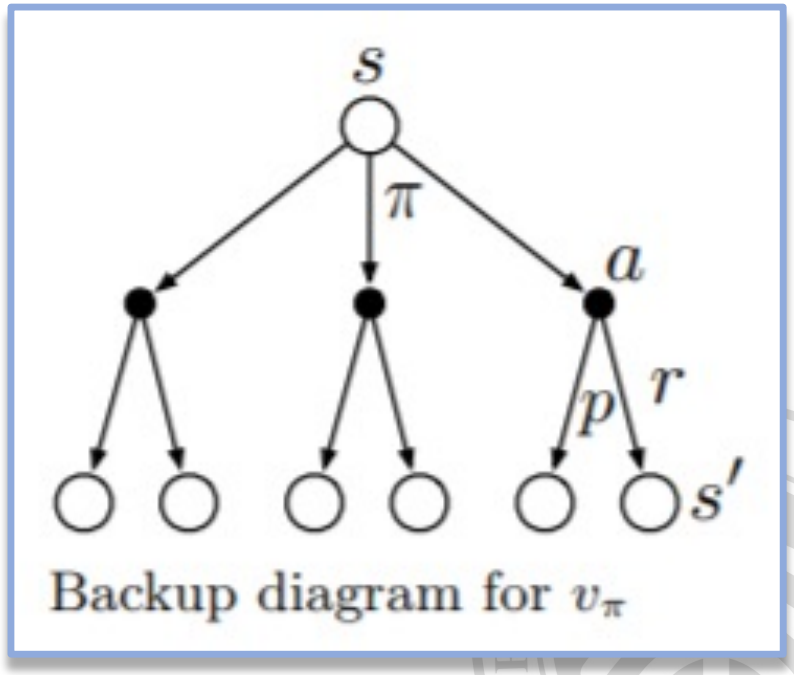
  依据被评估策略 π 所有可能的一步转移，用$v_k(s)$依次计算$v_{k+1}(s), \forall s \in S$

- 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

新一轮估值和上一轮估值中最大误差的状态估值



Backup diagram for v_π

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 $V \approx v_\pi$

联想单源最短路Bellman-Ford算法每次使状态转移前后一致，但是后面的可能又被更新了，所以不能一次结束更新

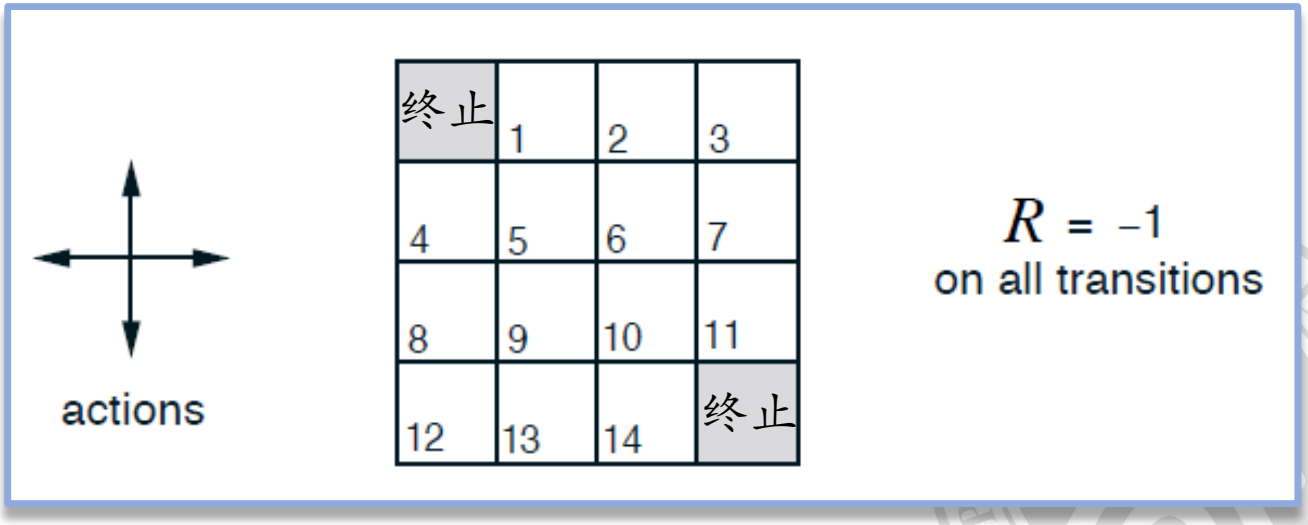
Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p61*

v_π的存在性和唯一性保障条件

- v_π的存在性和唯一性保障条件
  - $\gamma < 1$ 如果是无限步，$\gamma < 1$可以想象成远处传递上来的值为0或者依据策略π，所有状态最终能保证终止如果是有限步比照Bellman-Ford算法
- 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。
    证明留待后面的值迭代收敛性证明
  - 计算依据的是同一个策略π

策略估值计算的例子

- 下图为网格地图，假设agent只能移动到地图上有数字的格子，问题的MDP模型如下
  - 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。
  - 动作集合$A = \{上, 下, 左, 右\}$
  - $\gamma = 1$
  - R： -1;



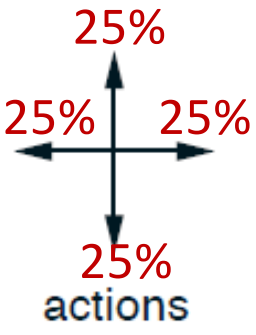
- P 转移概率： $p(5,-1|6,左) = 1$ ； $p(终止,-1|1,左) = 1$ 确定性转移

s' r | s a　　s' r | s a

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

策略估值计算的例子

- 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}



s_{t1}	1	2	3
4	5	6	7
8	9	10	11
12	13	14	s_{t2}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

状态价值表

$R = -1$
on all transitions

注：右上角矩阵中给出了$k \to \infty$时的稳定值函数，可以找到 $v_\pi(11)$

$$q_\pi(11, 下) = p(s_{t2}, -1|11, 下) * [r + \gamma v_\pi(s_{t2})] = 1 * (-1) = -1$$

$$q_\pi(7, 下) = p(11, -1|7, 下) * [r + \gamma v_\pi(11)] = 1 * [-1 + 1 * (-14)] = -15$$

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

策略估值计算的例子

- 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15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，向下待在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？只写出计算式。

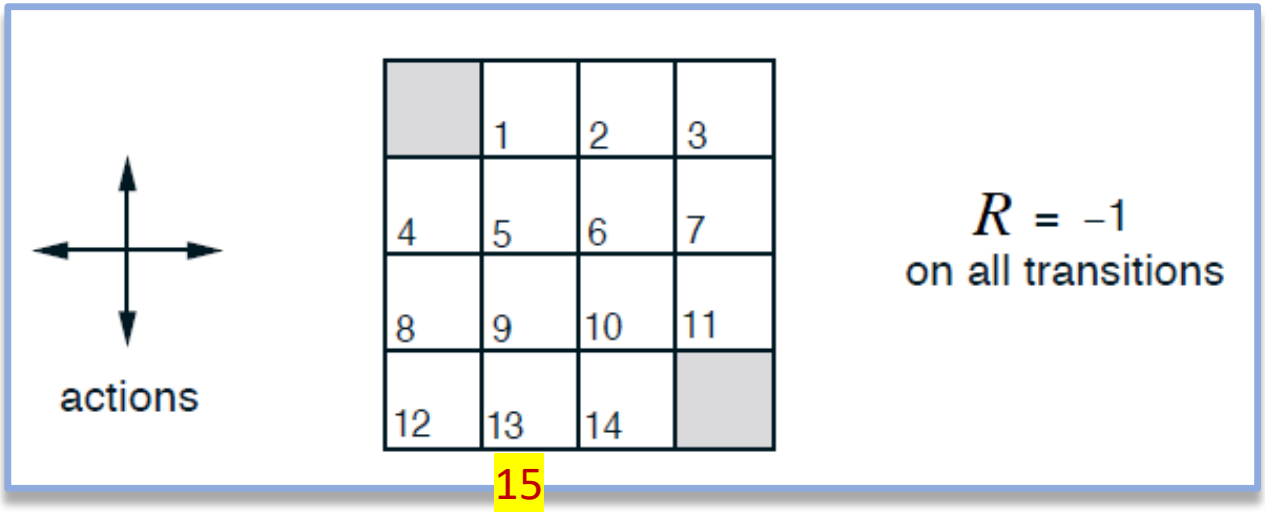


Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. p61*

答案

- 问二答案：
- $v_\pi(15) = \pi(\text{上}|15) * p(13,-1|15,\text{上}) * [-1 + 1 * v_\pi(13)]$
  $+ \pi(\text{下}|15) * p(15,-1|15,\text{下}) * [-1 + 1 * v_\pi(15)]$
  $+ \pi(\text{左}|15) * p(12,-1|15,\text{上}) * [-1 + 1 * v_\pi(12)]$
  $+ \pi(\text{右}|15) * p(14,-1|15,\text{上}) * [-1 + 1 * v_\pi(14)]$

r γ

$= -1 + 0.25 * (-20 - 22 - 14 + v_\pi(15))$

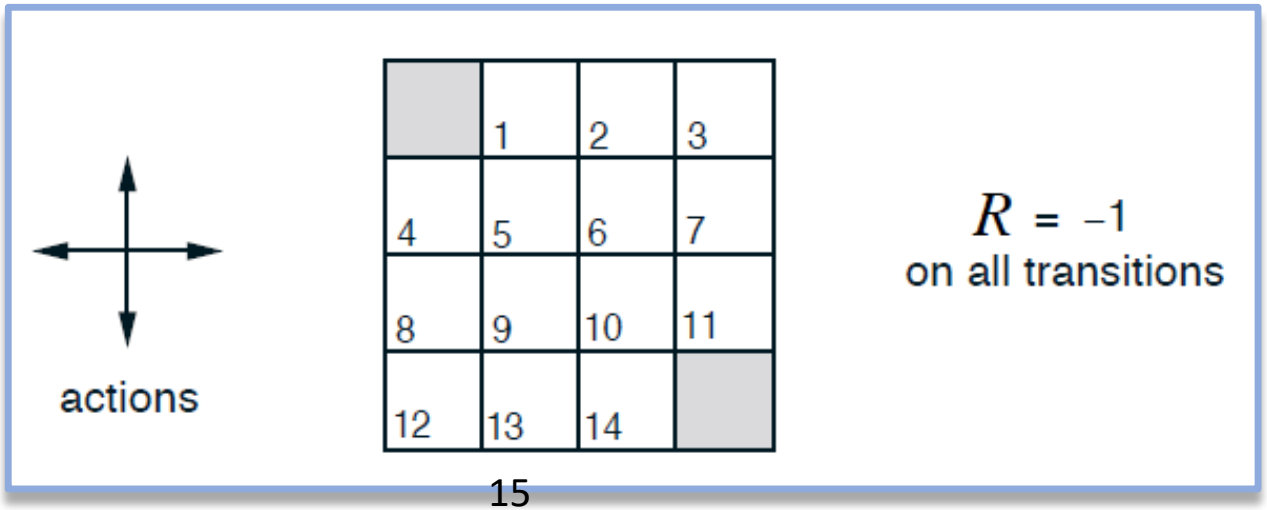
$(1-0.25) * v_\pi(15) = -15$

$v_\pi(15) = -20$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

策略估值计算的例子

- 问三：如果第13号格子向下转移到15号，$v_\pi(15)$等于多少？



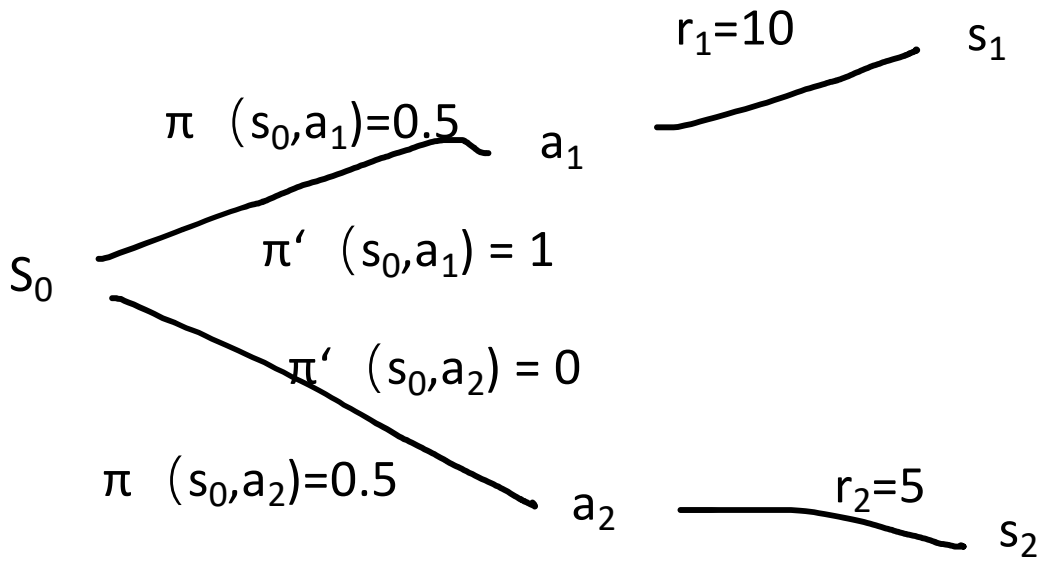
问三答案：

13号向下转移到15，经过计算，恰好$v_\pi(13) = v_\pi(15) = -20$

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

2 策略提升（Policy Improvement）

策略提升的机会



$V(s_1) = V(s_2)=0$　　结束状态

$V_\pi(s_0) = 7.5$

$q_\pi(s_0, a_1) = 10$

$q_\pi(s_0, a_2) = 5$

$\pi'(s_0) = a_1$

$q_\pi(s_0, \pi'(s_0)) = 10 > V_\pi(s_0) = 7.5$

$V_{\pi'}(s_0) = 10 > V_\pi(s_0) = 7.5$

2 策略提升（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
- 可以根据状态估值计算动作估值

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

- 贪心选择动作产生新策略

$$\pi'(s) \doteq argmax_a q_\pi(s,a)$$

策略提升方法

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

Formulas referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P63*

2 策略提升总结

- 在某个状态s下，找到一个比π好的q，在所有遇到状态s时，用q替代原来的π，得到新的π’
- 因为其他状态的动作都保持不变，对于所有状态，$V_{\pi'}(s)$一定比$V_{\pi}(s)$要好，策略得到提升
- 进而考虑在所有s下，贪心选择q最好的动作，完成一次更有成效的策略提升
- 如果有多于一个并列最好的q，可以在这些q里随机选
- 策略提升后，状态价值对于新的策略估计的就不准确了，需要重新估计……

3 策略迭代（Policy Iteration）

- 定义：交替进行迭代策略估值和策略提升，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提升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

E：Evaluation估值
I：improvement提升

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

where $\xrightarrow{\text{E}}$ denotes a policy *evaluation* and $\xrightarrow{\text{I}}$ denotes a policy *improvement*.

- 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
- 在有限MDP问题中，$\mathcal{S}$，$\mathcal{A}$，$\mathcal{R}$ 的取值都有有限个

策略迭代算法伪代码

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*

策略提升结束条件

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

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

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

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

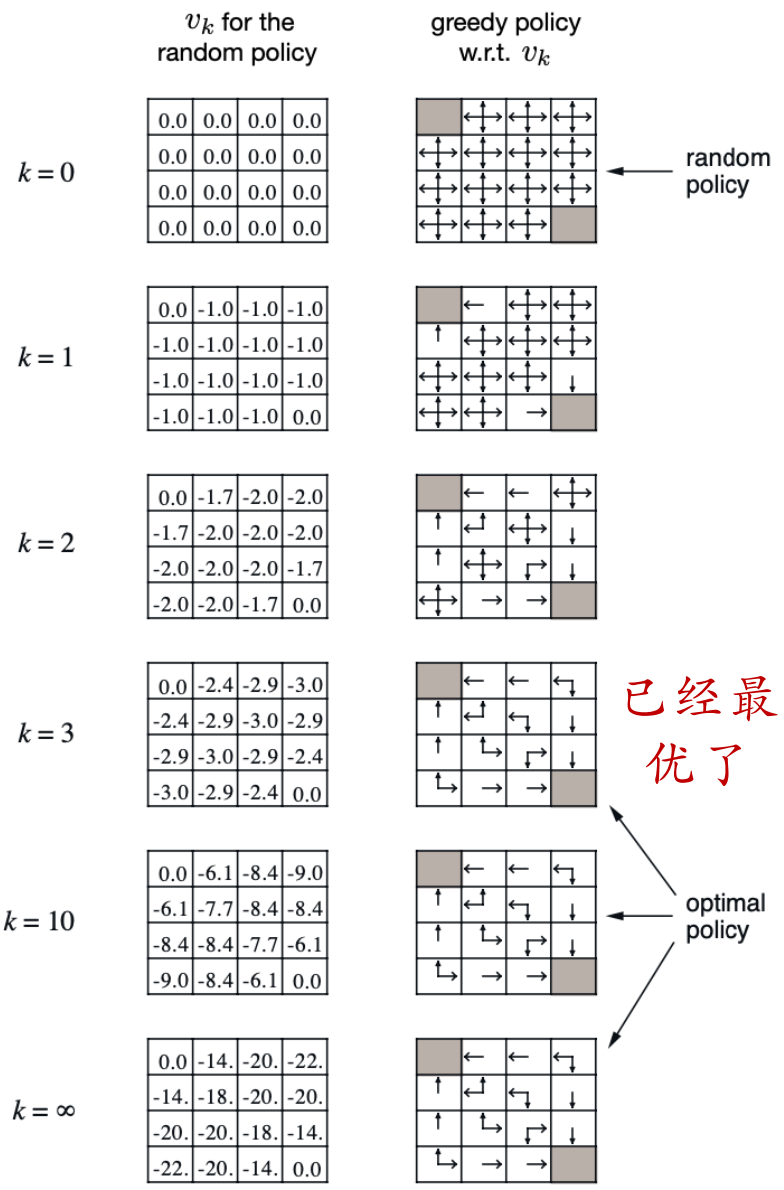
内容提要

1. 马尔可夫决策过程 Markov decision process (MDP)
   - $MDP < S, A, P, R, \gamma >$
   - 马尔可夫性
   - 贝尔曼方程
2. P, R已知，用动态规划方法求解最优策略
   - 策略迭代（Policy Iteration）通过策略改进得到最优策略
   - 值迭代（Value Iteration）通过求最优状态价值得到最优策略
   - 广义策略迭代（Generalized Policy Iteration）
3. P, R未知，用采样方法逼近最优策略
   - Bootstrap
   - 蒙特卡洛学习
   - 时序差分学习

1 值迭代

- 策略迭代的问题
  - 策略估值耗费时间
  - 在右图的例子中后面的迭代对策略提升都没有帮助
  - 可以提前截断估值
- 一种极端的矫正方式
  - 只做一次估值、不需要估值收敛就开始策略提升
  - 这就是值迭代方法

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$



1 值迭代（Value Iteration）

- 所以除了策略迭代，还可以从另一个角度看值迭代
  - 既然改进策略就是贪心的选值最大的，那不如就直接求v的最大估值，求到最优值，再贪心
- 值迭代是另一种计算最优策略的方法.
- 基本思想是
  - 定义一个状态的价值是从它出发所能达到的最优价值
  - 计算所有状态所能达到的最优值，
  - 贪心地选择能获得最优价值的动作 – 最优动作

1 值迭代　算法伪代码

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

之前策略估值没有这项Max

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

1 值迭代

Policy iteration (using iterative policy evaluation)

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

策略迭代

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)

策略估值

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

策略提升

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

值迭代

贝尔曼更新

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

将两者结合，可以迭代估值若干轮，提升一次，再若干轮，提升一次

值迭代算法的收敛性分析

- 我们说值迭代算法最终会收敛到贝尔曼最优方程的唯一解，为什么呢?
- 这里引入一些数学思想。
- 一个基本概念是 contraction 压缩. 简单讲一个压缩变化是一个带有一个参数的函数, 当我们给它两个不同的输入时，它会输出两个距离更近的数值（常数倍于原距离）。
- 例如“除以2”是一个压缩变换, 因为当我们把两个数都除以2，他们间的距离也是原来的一半。注意，“除以2”这个函数有一个不动点, 就是 0, 这个点除以2，还是它自己。
- 从这个例子中，我们可以得到关于压缩变化的两个重要性质：

① 压缩变化只有一个不动点; 如果有两个，那么输入这两个不动点，输出的值的距离不会改变，所以它不是压缩变换;

② 当函数作用于任何参数，它的值应该离不动点更近了 (因为不动点的值不变), 所以反复使用压缩变换，会最终到达不动点;

值迭代算法的收敛性分析

证明：令 $\Delta = \max_s |V_{i+1}(s) - V_i(s)| = ||V_{i+1} - V_i||$ 一轮值迭代中估值变化最大值

向量距离的定义

$||V|| = \max_s |V(s)|$

$$V_{i+1}(s) = \max_a (R + \gamma V_i(s'))$$

$$V_{i+2}(s) \le \max_a (R + \gamma V_i(s') + \gamma\Delta)$$ 上一轮$V_i(s')$的更新量

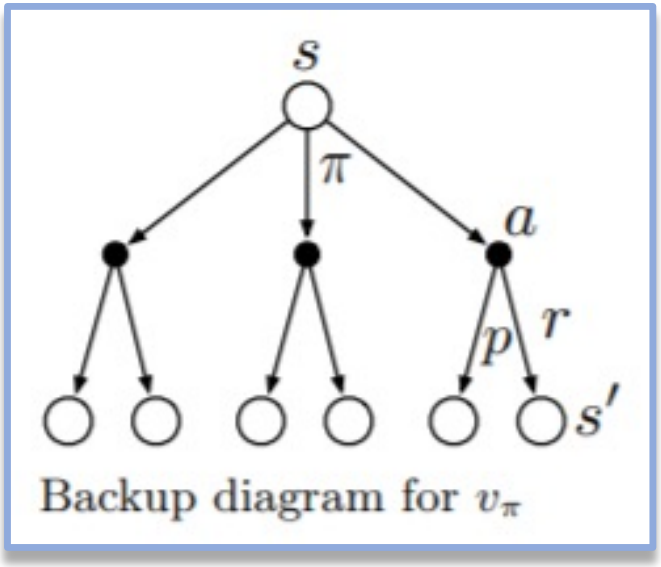
$$V_{i+2}(s) - V_{i+1}(s) \le \max_a (R + \gamma V_i(s') + \gamma\Delta) - \max_a (R + \gamma V_i(s'))$$

$$\le \max_a (R + \gamma V_i(s') + \gamma\Delta - R + \gamma V_i(s')) = \gamma\Delta$$

所以，$||BV_{i+1} - BV_i|| \le \gamma ||V_{i+1} - V_i||$，贝尔曼更新是压缩变换

策略估值和值迭代的回溯图

- 策略估值（Policy Evaluation）
- 值迭代（Value Iteration）



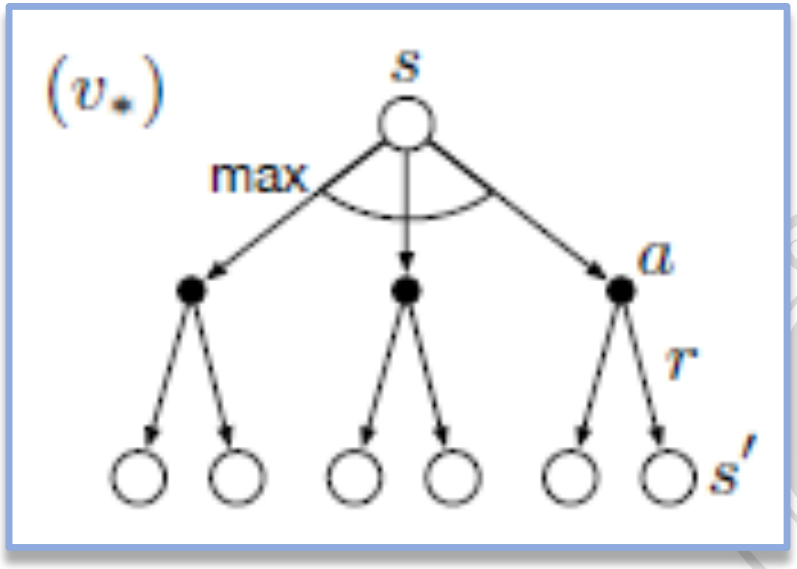


Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P47,51*

内容提要

1. 马尔可夫决策过程 Markov decision process (MDP)
   - $MDP < S, A, P, R, \gamma >$
   - 马尔可夫性
   - 贝尔曼方程
2. P, R已知，用动态规划方法求解最优策略
   - 策略迭代（Policy Iteration）通过策略改进得到最优策略
   - 值迭代（Value Iteration）通过求最优状态价值得到最优策略
   - 广义策略迭代（Generalized Policy Iteration）
3. P, R未知，用采样方法逼近最优策略
   - Bootstrap
   - 蒙特卡洛学习
   - 时序差分学习

1 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

不用每次都更新所有状态，
可以有选择地更新部分状态

- 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$v_{k_1}(s_1), v_{k_2+1}(s_2), \cdots, v_{k_n}(s_n)$

1 异步动态规划（Asynchronous Dynamic Programming）

- 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

2 广义策略迭代（Generalized Policy Iteration）

- 定义：策略估值和策略提升交互的一般思想，几乎所有强化学习方法都能用广义策略迭代（Generalized Policy Iteration） 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

2 广义策略迭代（Generalized Policy Iteration）

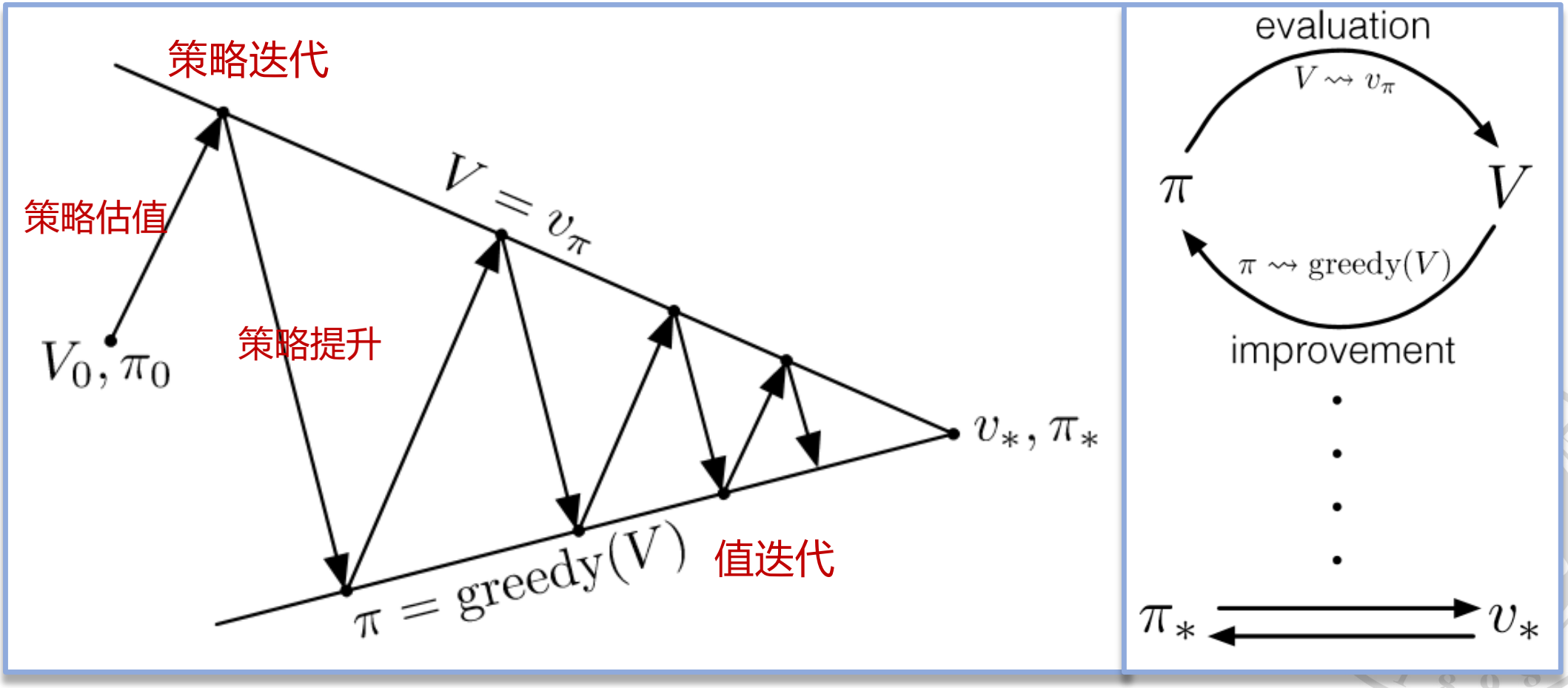


Figure referenced from *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. P70*

动态规划方法小结（1/3）

- 这一部分主要介绍了用动态规划的方法解MDP问题
- 策略估值通过反复迭代计算一个给定的策略下能达到的总收益
- 策略提升根据规定策略的估值贪心地改进原策略
- 结合这两者我们得到两种流行的动态规划方法：策略迭代和值迭代
- 这两种方法在给定MDP问题时都能得到最优策略和最优价值

动态规划方法小结（2/3）

- 经典动态规划扫描整个状态空间，用后续状态值的加权和更新每个状态的值。更新时，用后继状态可能出现的概率和后继状态的值。
- 更新公式使用贝尔曼方程
- 当使用更新公式时，状态的值不再有变化就说明所有状态及其后继的值都满足贝尔曼方程

动态规划方法小结（3/3）

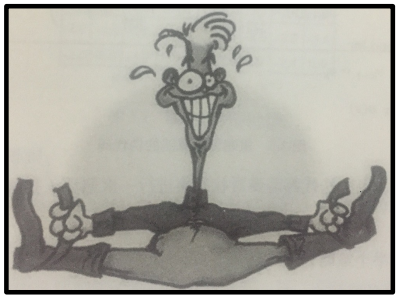
- General Policy Iteration 广义策略迭代在策略估值和策略提升两个进程之间交替进行
- 一个进程负责策略估值，使得值越来越接近中间策略的估值
- 另一个进程负责根据估值贪心地提升策略
- 尽管一个进程的进行会使另一个进程的值跑偏，但是总的来说他们奔向一个共同的目标: 一个稳定的策略和一个稳定的估值，即最优策略和最优价值
- 一定条件下， GPI 被证明可以收敛到最优策略和最优价值。

内容提要

1. 马尔可夫决策过程 Markov decision process (MDP)
   - $MDP < S, A, P, R, \gamma >$
   - 马尔可夫性
   - 贝尔曼方程
2. P, R已知，用动态规划方法求解最优策略
   - 策略迭代（Policy Iteration）
   - 值迭代（Value Iteration）
   - 广义策略迭代（Generalized Policy Iteration）
3. P, R未知，用采样方法逼近最优策略
   - Bootstrap
   - 蒙特卡洛学习：直接采样到终止状态、估算V(S)
   - 时序差分学习：每走一步得到R和S‘，用V(S’)来估算一次V(S)，R大则V涨、R小则V跌

bootstrap

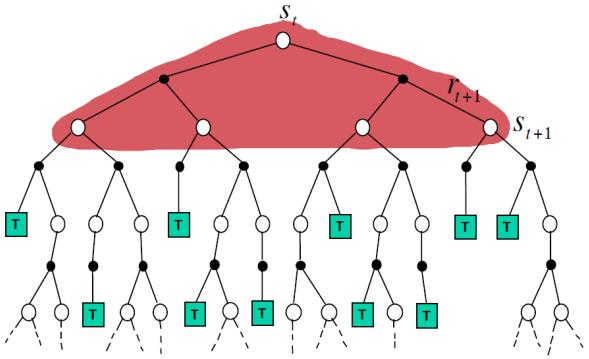
- 最后，我们可以注意到，计算一个状态的估值是依据它的后继状态的估值的，我们称之为bootstrap。
- 许多强化学习的方法使用bootstrap 方法。
- 动态规划方法假设环境已知（P，R已知）
- 蒙特卡洛学习方法假设环境未知（P，R未知），也不使用bootstrap.
- 时序差分学习方法假设环境未知（P，R未知），但是使用bootstrap.
- 环境已知和bootstrap是彼此独立的，当然也可以对他们进行有趣组合。



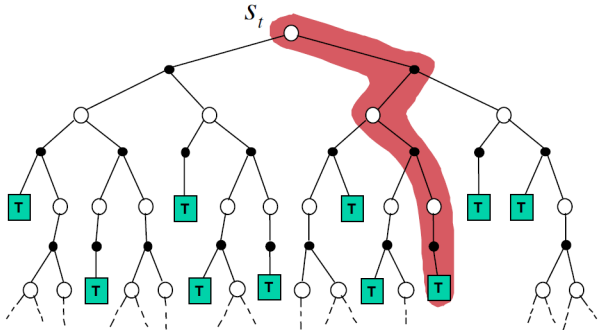
Bootstrap直译为靴带，源自18世纪小说《巴龙历险记》(Adventures of Baron Munchausen)中的短语”pull oneself up by one’s bootstrap”，Baron落入湖中沉到湖底，在绝望的时候用靴子上的带子把自己拉了上来。

学习方法：蒙特卡洛和时序差分

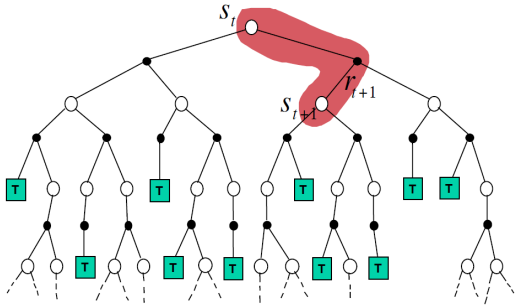
动态规划更新形式



蒙特卡洛更新形式



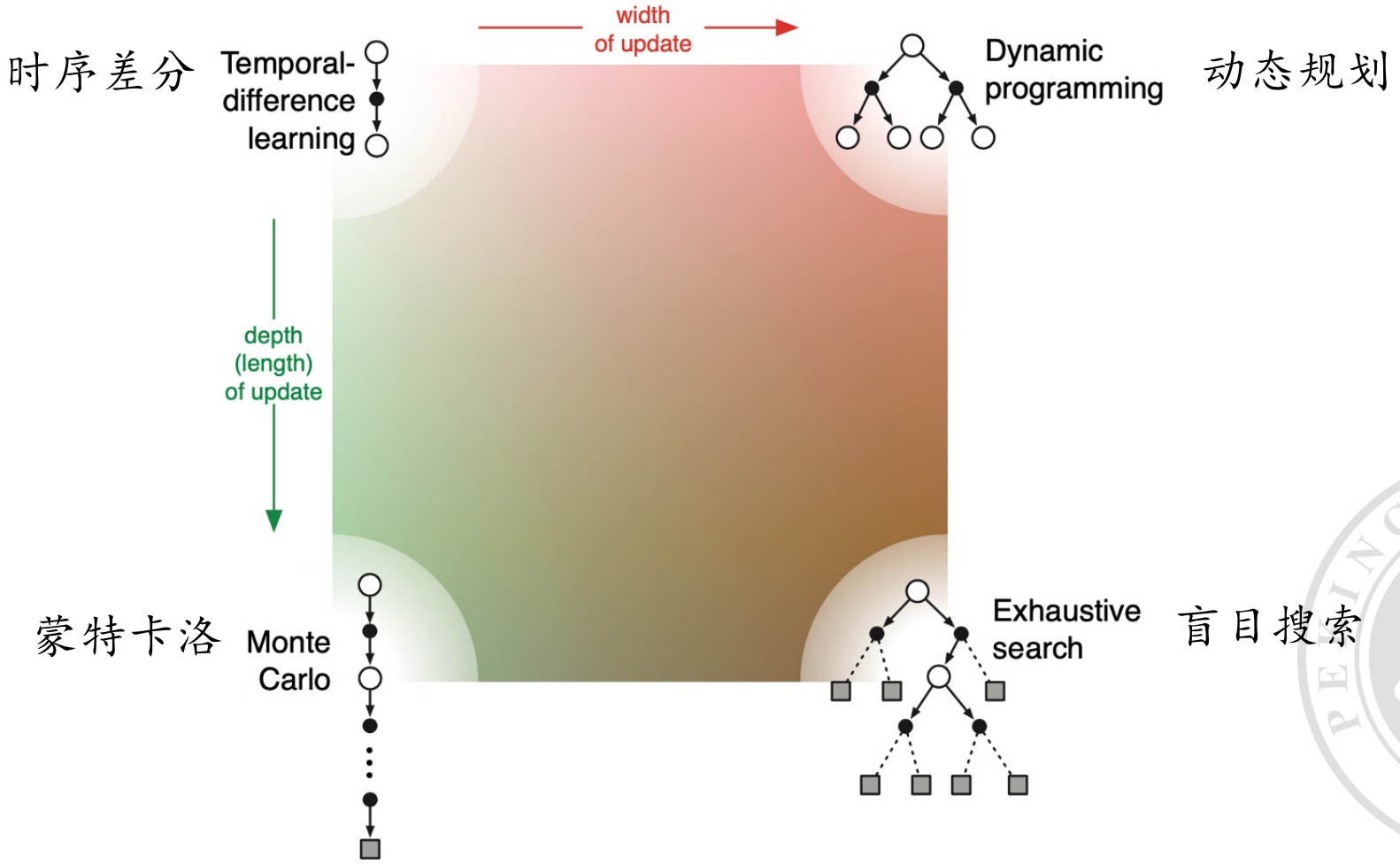
时序差分更新形式



三种方法价值更新方式比较

学习算法	Bootstrap	Sampling	V_π更新
动态规划	✓	✘	$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(s')]$
蒙特卡洛	✘	✓	$V_\pi(s) \leftarrow V_\pi(s) + \alpha[G_t - V_\pi(s)]$
时序差分	✓	✓	$V_\pi(s) \leftarrow V_\pi(s) + \alpha[R_{t+1} + \gamma V_\pi(s') - V_\pi(s)]$

ES、DP、MC、TD的比较



时序差分

动态规划

蒙特卡洛

盲目搜索