强化学习

Reinforcement learning

第三节

动态规划

Planning by Dynamic Programming

张世周

前情回顾

- 强化学习被建模成马尔可夫决策过程 $< S, A, P, R, \gamma >$
- 马尔可夫决策过程的求解转换为求解贝尔曼期望/最优方程
- 贝尔曼期望方程是一个线性方程（闭式解），复杂度 $O(n^3)$；

$$v_\pi(s) = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$$

- 贝尔曼最优方程是一个非线性方程（无闭式解）；

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

- 有许多间接的解贝尔曼方程的方法，比如：动态规划、蒙特卡罗评估、时间差分学习

Outlines

- 1.1 介绍动态规划
- 1.2 策略评估
- 1.3 策略迭代
- 1.4 价值迭代
- 1.5 动态规划的拓展
- 1.6 压缩映射

介绍动态规划

什么是动态规划（Dynamic Programming）

动态规划算法是解决复杂问题的一个（类）方法（思想），算法通过把复杂问题分解为子问题，通过求解子问题进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。（通常是以空间换时间）

quora上有这样一个问题:

How should I explain dynamic programming to a 4-year-old?

底下有个42K赞同的答案，是这样说的:

writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper

"What's that equal to?"

***counting* "Eight!"**

writes down another "1+" on the left

"What about that?"

***quickly* "Nine!"**

"How'd you know it was nine so fast?"

"You just added one more"

"So you didn't need to recount because you remembered there were eight! *Dynamic Programming* is just a fancy way to say 'remembering stuff to save time later'"

介绍动态规划

什么时候用到动态规划？

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：第一个特性是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；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。（以空间换时间）

马尔科夫决策过程（MDP）具有上述两个属性：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。

介绍动态规划

我们用动态规划算法来求解一类称为“规划”（Planning）的问题。“规划”指的是在了解整个MDP的基础上求解最优策略，也就是清楚模型结构的基础上：包括状态/动作空间、转换矩阵、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。

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$<S, A, P, R, \gamma>$ ，要求确定最优价值函数 v_{*} 和最优策略 π_{*} 。

动态规划的其他应用

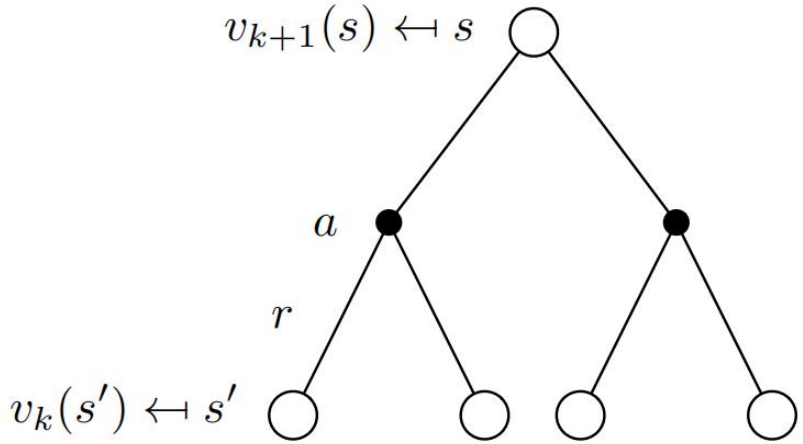
- 调度算法（装配线/动态车辆调度/作业调度）
- 字符串算法（如序列比对---最小编辑距离/最长公共子序列）
- 图算法（例如最短路径算法）
- 图模型（例如维特比算法）
- 生物信息学（如晶格模型）

迭代法策略评估

- 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：反向迭代应用Bellman期望方程
- 具体方法：同步反向迭代，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 $v_k(s')$ 计算并更新该状态第k+1次迭代中使用的价值 $v_k(s)$，其中s’是s的后继状态。
- 我们在之后会讨论异步反向迭代
- 收敛到 v_π 将在本节课结束时得到证明（压缩映射）

策略评估

迭代法策略评估（2）

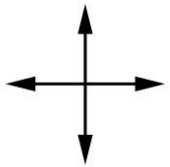


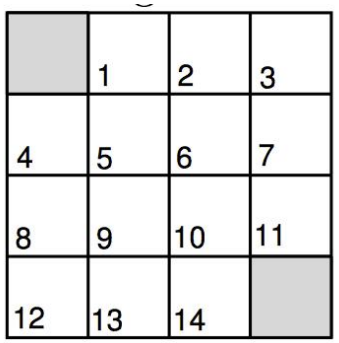
$$v_{k+1}(s) = \sum_{a \in A} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

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

策略评估

在格子世界中评估随机策略





$r = -1$
on all transitions

格子世界的情况如下：

状态空间S：如图。S1 - 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行为；

转移概率P：任何试图离开方格世界的动作其位置将不会发生改变，其余条件下将100%地转移到动作指向的状态；

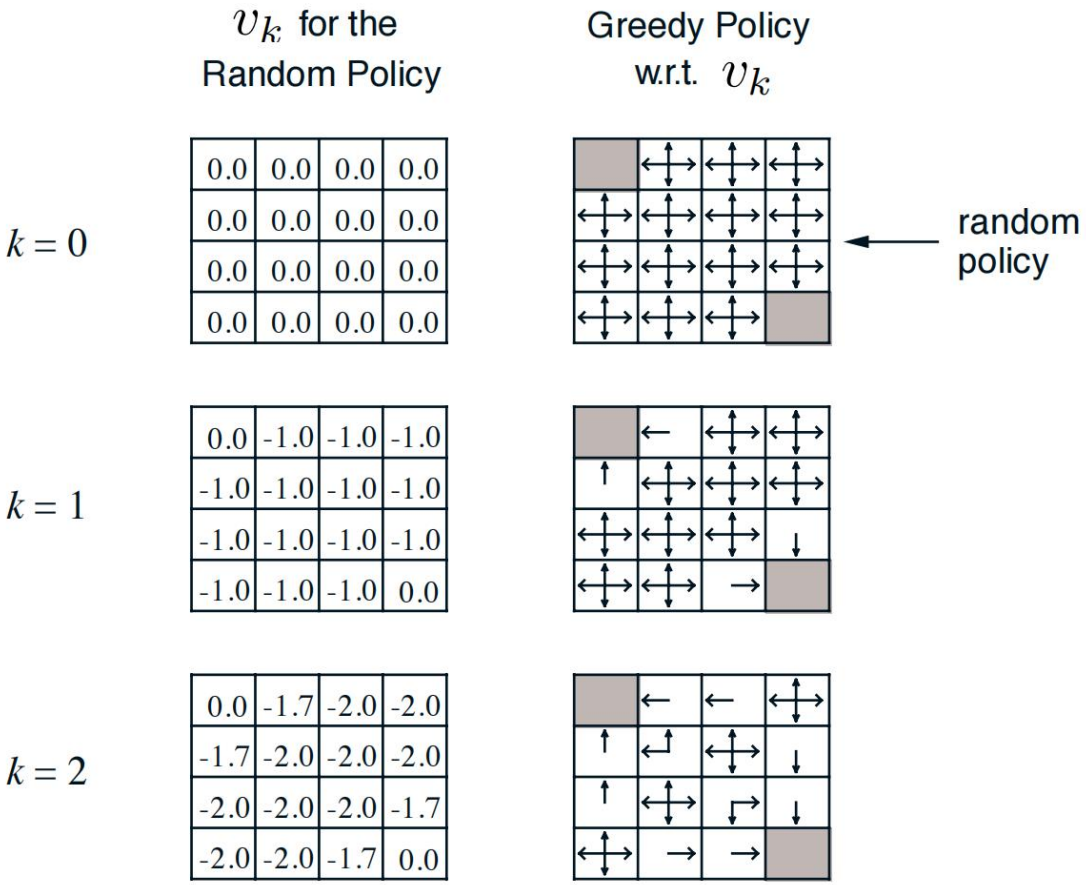
即时奖励R：任何在非终止状态间的转移得到的即时奖励均为-1，进入终止状态即时奖励为0；

折扣系数γ：1；

当前策略π：Agent采用随机行动策略，在任何一个非终止状态下有均等的几率采取任一移动方向这个行为，即 $\pi(n|\bullet) = \pi(e|\bullet) = \pi(s|\bullet) = \pi(w|\bullet) = 1/4$。

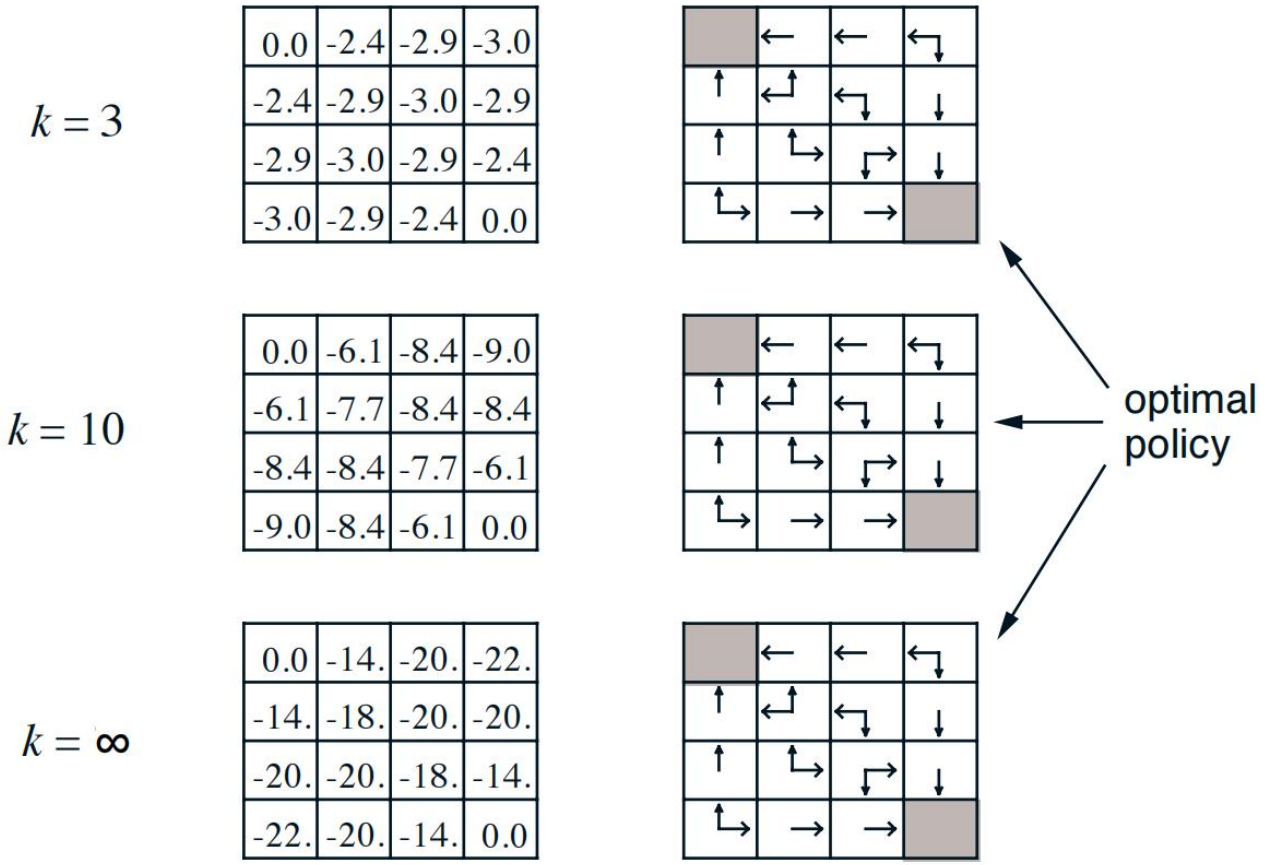
策略评估

格子世界中的迭代策略评估



策略评估

格子世界中的迭代策略评估（2）



策略迭代

如何改善策略

分为两步：首先我们在一个给定的策略下迭代更新价值函数

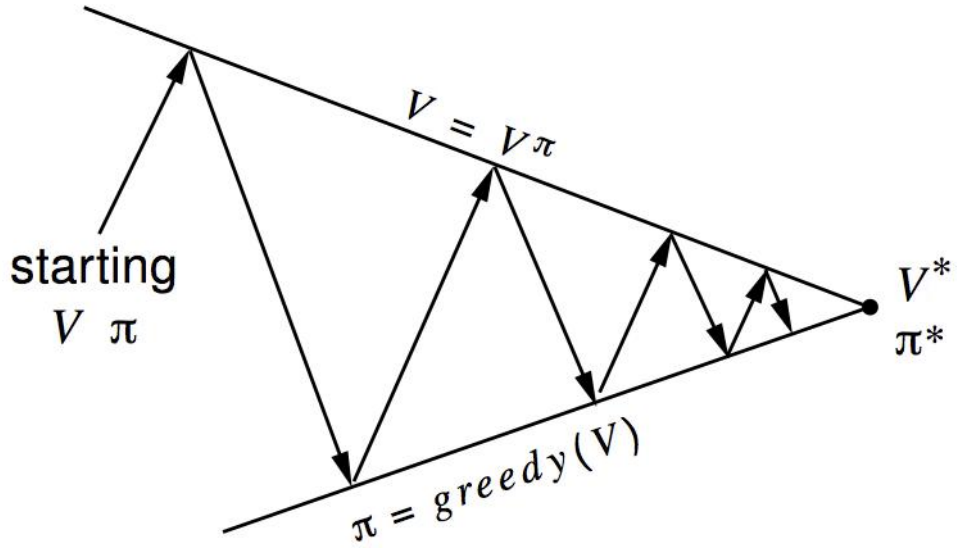
$$v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$$

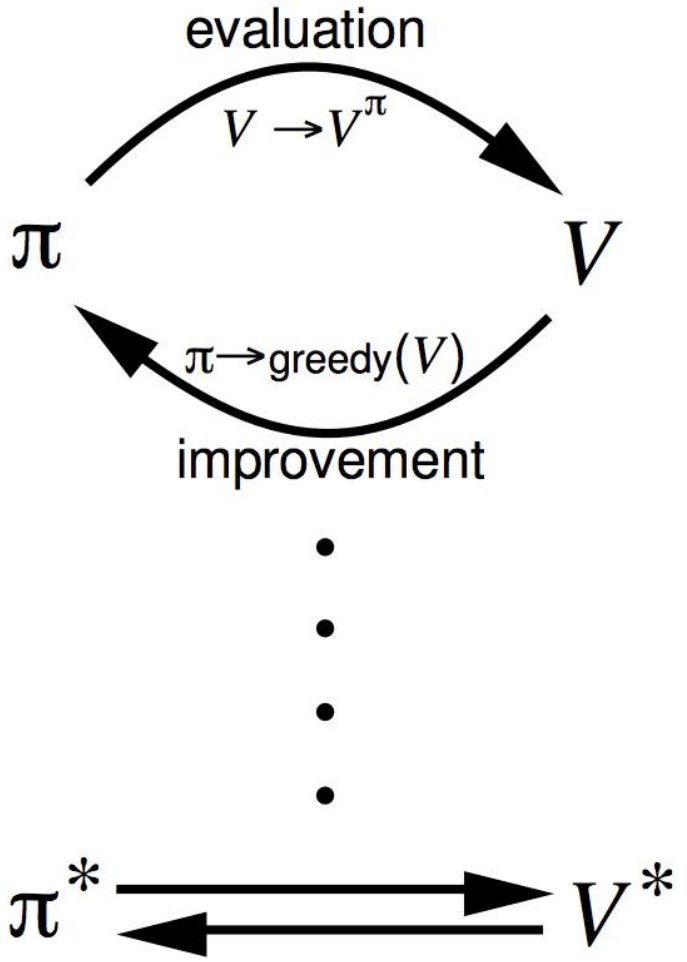
随后，在当前策略基础上，贪婪地选取行为，使得后继状态价值增加最多：

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

在刚才的格子世界中，基于给定策略的价值迭代最终收敛得到的策略就是最优策略，但通过一个回合的迭代计算价值联合策略改善就能找到最优策略不是普遍现象。通常，还需在改善的策略上继续评估，反复多次。不过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，
如此反复多次，最终得到最优策略 π^*和最优状态价值函数V^*

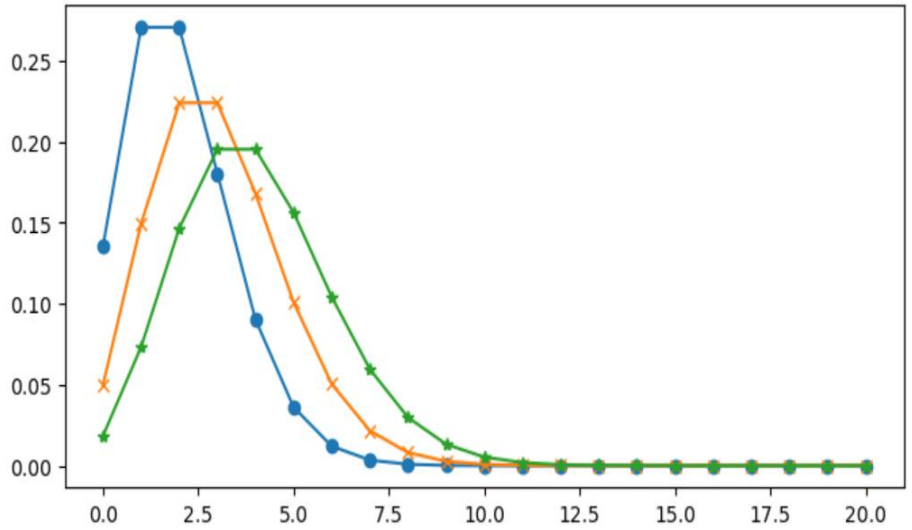
策略迭代

杰克租车问题


问题描述：杰克管理一家有两个地点的租车公司。每一天，一些用户会到一个地点租车。如果杰克有可用的汽车，便会将其租出，并从全国总公司那里获得10美元的收益。如果他在那个地点没有汽车，便会失去这一业务。租出去的汽车在还车的第二天变得可用。为了保证每辆车在需要的地方使用，杰克在夜间在两个地点之间移动车辆，移动每辆车的代价为2美元。我们假设每个地点租车与还车的数量是一个泊松随机变量，即数量为n的概率为 $\frac{\lambda^n}{n!}e^{-\lambda}$ 。假设租车的期望在两个地点分别为3和4，而还车的期望分别为3和2。为了简化问题，我们假设任何一个地点有不超过20辆车，并且每天最多移动5辆车。折扣率为0.9。

策略迭代

杰克租车问题



已知：

状态空间：2个地点，每个地点最多20辆车供租赁

行为空间：每天下班后最多转移5辆车从一处到另一处；

即时奖励：每租出1辆车奖励10元，必须是有车可租的情况；不考虑在两地转移车辆的支出。

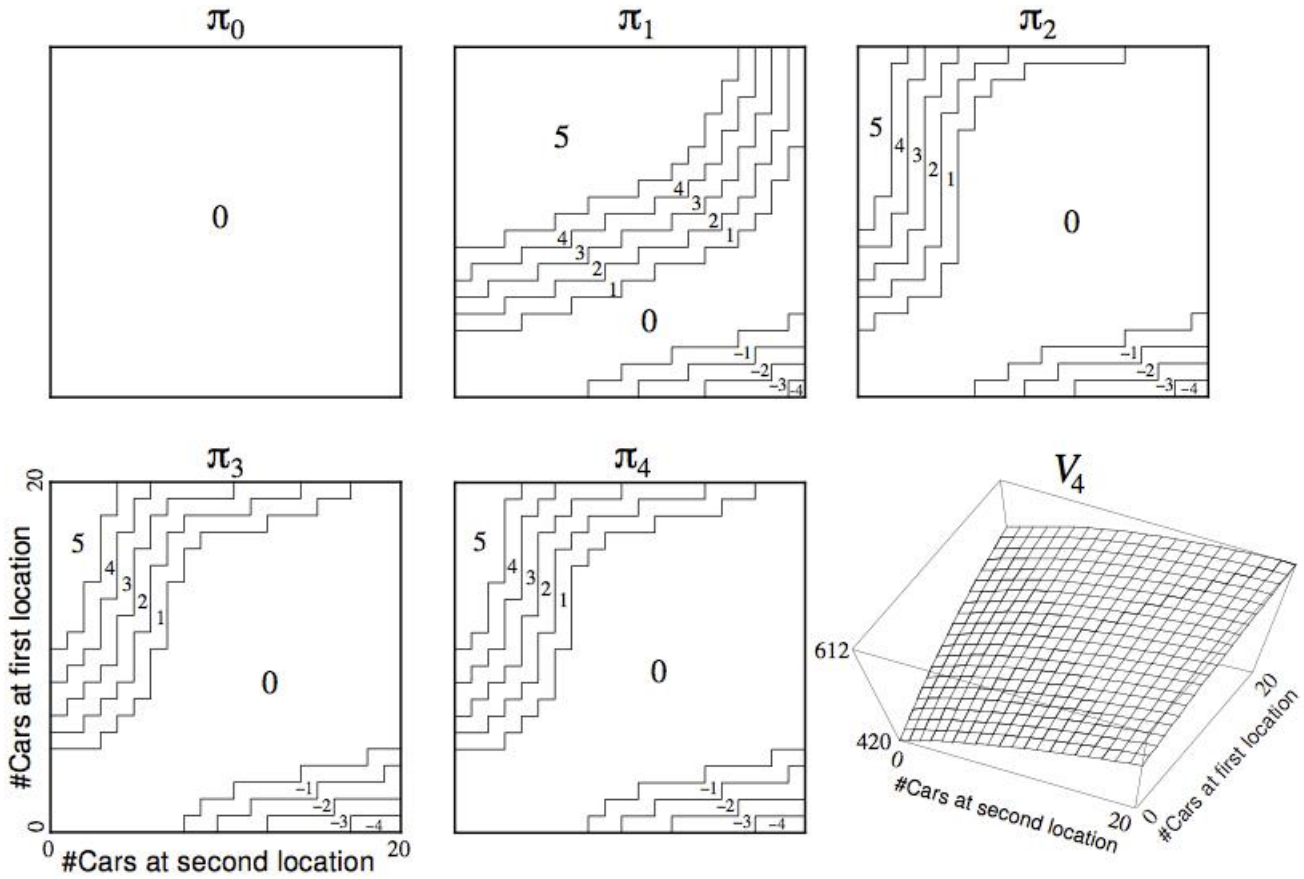
转移概率：求租和归还是随机的，但是满足泊松分布 $\frac{\lambda^n}{n!}e^{-\lambda}$ 。第一处租赁点平均每天租车请求3次，归还3次；第二处租赁点平均每天租车4次，归还2次。

衰减系数 γ：0.9；

问题：怎样的策略是最优策略？

策略迭代

杰克租车问题的策略迭代过程



策略迭代

策略改善定理

1. 考虑一个确定的策略：$a = \pi(s)$

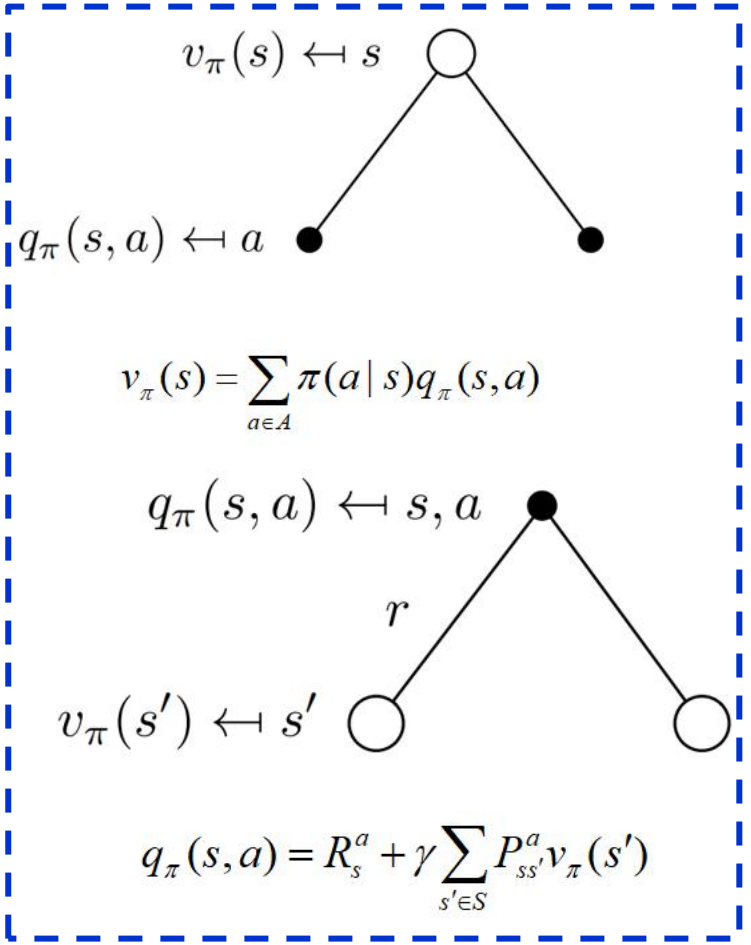
2. 通过贪婪计算优化策略：$\pi'(s) = \arg\max_{a \in A} q_\pi(s,a)$

- 一步迭代改善状态s的q值

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- 因此，也会提高价值函数，使

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



策略迭代

策略改善——理论证明（2）

3、当提升停止时满足以下公式

$$q_\pi(s,\pi'(s)) = \max_{a\in A} q_\pi(s,a) = q_\pi(s,\pi(s)) = v_\pi(s)$$

因此这就满足了Bellman最优方程

$$v_\pi(s) = \max_{a\in A} q_\pi(s,a)$$

此时π就是一个最优策略

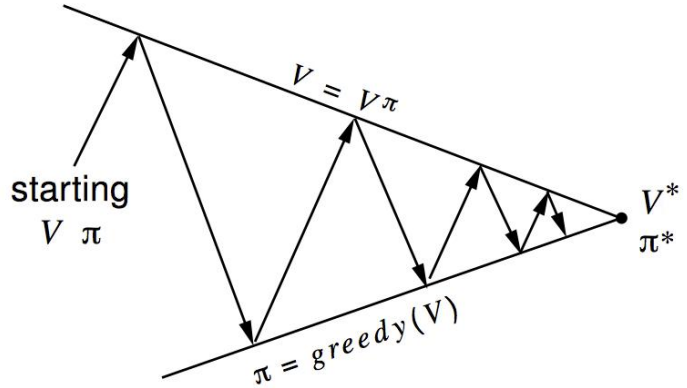
修饰过的策略迭代（Modified Policy Iteration）

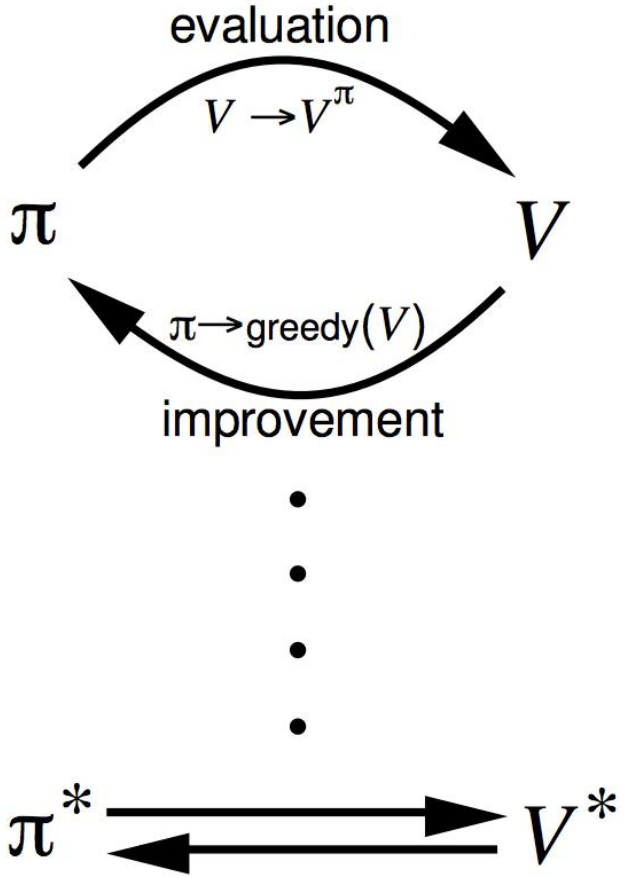
很多时候，策略的更新较早就收敛至最优策略，而状态价值的收敛要慢很多，是否有必要一定要迭代计算直到状态价值得到收敛呢？

有时候不需要持续迭代至最优价值函数，可以设置一些条件提前终止迭代，比如设定一个ε，比较两次迭代的价值函数平方差；直接设置迭代次数；以及每迭代一次更新一次策略等（值迭代）。

策略迭代

广义策略迭代





策略评估: 估计 v_π

通过任何策略评估算法

策略改善: 生成$\pi' \geq \pi$

通过任何策略改善算法

价值迭代（Value Iteration）

最优性定理

最优策略可以被分解为两部分：

（1）从状态s到后继状态s’采取了最优行为 A_* ；

（2）在状态s’时遵循最优策略。

最优性定理

一个策略能够使得状态s获得最优价值，当且仅当：
对于从状态s出发，任意可达的状态s’，该策略能够使得状态s’的价值是最优价值：

价值迭代

确定性的价值迭代

- 如果我们知道子问题 $v_*(s')$ 的解
- 然后 $v_*(s)$ 可以通过“一步前瞻”找到

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

- 价值迭代的思想是迭代地应用这些更新
- 从最后的奖励开始，然后反向工作
- 仍然适用于循环的随机MDP

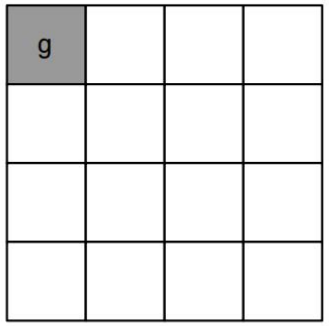
$$R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

代表一步迭代策略评估；max操作，代表进行了一次策略改善

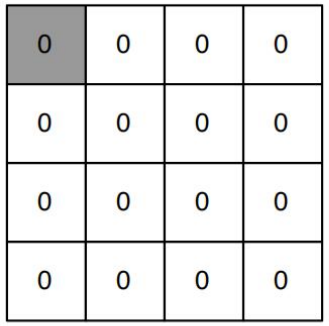
价值迭代

示例：最短路径

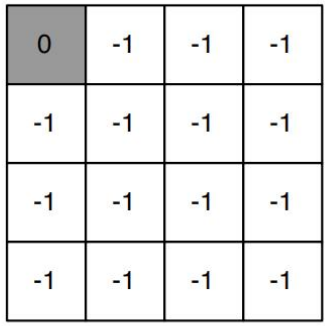
$$v_*(s) \leftarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$



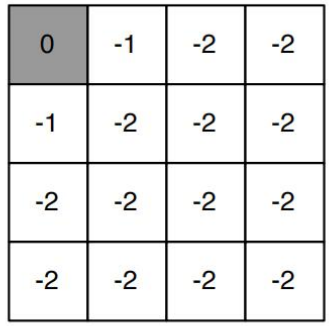
Problem



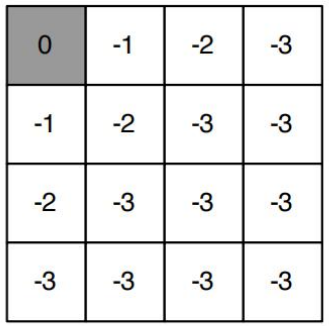
V_1



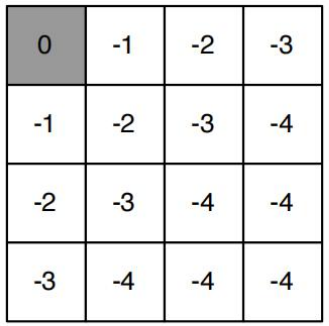
V_2



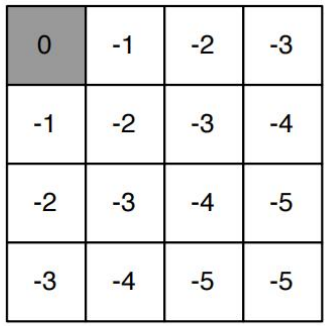
V_3



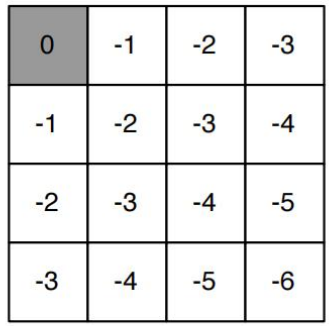
V_4



V_5



V_6



V_7

价值迭代

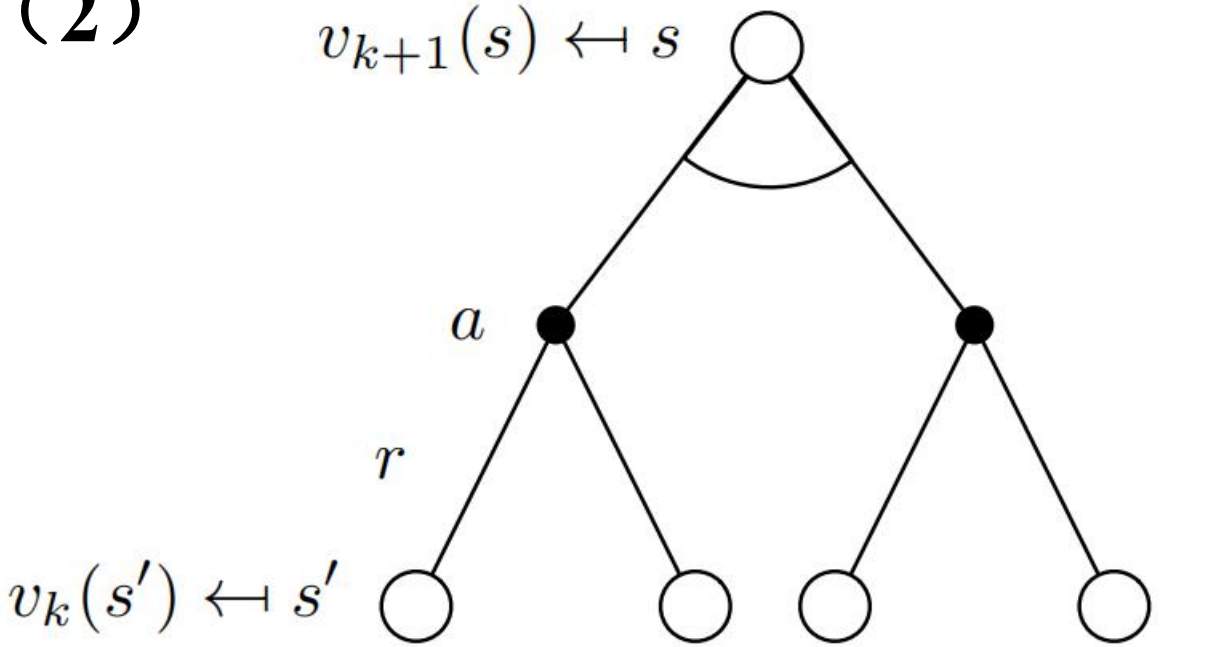
- 问题：寻找最优策略π
- 解决方案：从初始状态价值开始同步迭代计算，最终收敛，整个过程中没有遵循任何策略。

$$v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$$

- 使用同步反向迭代：对于第 k+1次迭代，所有的状态s的价值$v_{k+1}(s)$，使用$v_k(s')$计算，其中s’是s的后继状态。
- 注意：与策略迭代不同，在价值迭代过程中，算法不会给出明确的策略，迭代过程中得到的价值函数，不对应任何策略。

价值迭代

价值迭代（2）



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\boldsymbol{a}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{a}} \mathbf{v}_k$$

价值迭代在实践中的应用实例

https://artint.info/demos/mdp/vi.html

价值迭代

同步动态规划算法

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

- 算法基于状态价值函数 $v_\pi(s)$ 或 $v_*(s)$
- 对于m个动作和n个状态，每次迭代的复杂性 $O(mn^2)$
- 也可以应用于作用值函数 $q_\pi(s,a)$ 或 $q_*(s,a)$ 复杂度是 $O(m^2n^2)$

动态规划的拓展

异步动态规划

- 到目前为止描述的DP方法使用的是同步备份
- 换句话说就是所有的状态都是并行备份
- 异步DP以任何顺序单独备份状态
- 对于每个选定的状态，选择使用适当的备份
- 可以显著减少计算量
- 如果所有状态能够持续被选择，则保证收敛

异步动态规划

异步动态规划的三个简单思想：

- 就地动态规划（In-Place DP）
- 重要状态优先更新(Prioritised Sweeping)
- 实时动态规划(Real-time DP)

动态规划的拓展

就地动态规划

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。

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

$$v_{old} \leftarrow v_{new}$$

同步价值迭代

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

就地价值迭代

重要状态优先更新

对那些重要的状态优先更新。使用Bellman error来确定哪些状态是比较重要的。Bellman error 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。Bellman error越大，越有必要优先更新。对那些Bellman error较大的状态进行备份。这种算法使用优先级队列能够较得到有效的实现。

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

Bellman error

动态规划的拓展

实时动态规划

（1）更新那些仅与智能体相关的状态；

（2）使用智能体的经验来指导更新状态的选择；

（3）每个时间步 $S_t, \boldsymbol{S_{t+1}}, \boldsymbol{R_{t+1}}$ ， 备份状态$\mathbf{S_t}$

$$v(S_t) \leftarrow \max_{a \in A} \left(R^a_{S_t} + \gamma \sum_{s' \in S} P^a_{S_t s'} v(s') \right)$$

有些状态虽然理论上存在，但在现实中几乎不会出现。St是实际与Agent相关或者说Agent经历的状态，可以省去关于那些仅存在理论上的状态的计算。

动态规划的拓展

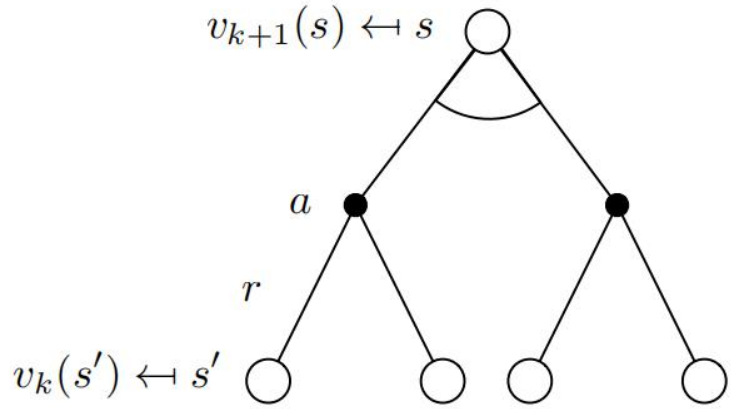
全宽备份(Full-width Backup)

（1）动态规划使用全宽备份。

（2）使用DP算法，对于每一次状态更新，都要考虑到其所有后继状态及所有可能的行为，同时还要使用MDP中的状态转移矩阵、奖励函数（信息）。

（3）DP的这一特点决定了其对中等规模（百万级别的状态数）的问题较为有效，但是对于更大规模的问题，会带来Bellman维数灾难。

（4）即使一次备份代价也会非常昂贵。

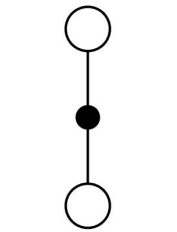


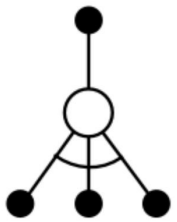
动态规划的拓展

采样备份（Sample Backups）

（1）因此在面对大规模MDP问题时，需要寻找更加实际可操作的算法，主要的思想是采样备份。

（2）这类算法的优点是不需要完整掌握MDP的条件（例如奖励机制、状态转移矩阵等），通过Sampling（采样）可以打破维度灾难，反向更新状态函数的开销是常数级别的，与状态数无关。





动态规划的拓展

近似动态规划

- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w_k}) \right)$$

  - Train next value function $\hat{v}(\cdot, \mathbf{w_{k+1}})$ using targets $\{\langle s, \tilde{v}_k(s) \rangle\}$

压缩映射

一些技术问题

- 我们怎么知道值迭代收敛到 v_*？
- 或者迭代策略评估收敛到 v_π？
- 策略迭代收敛到 v_*？
- 得到的解是唯一的吗？
- 这些算法的收敛速度有多快？
- 这些问题可以由压缩映射定理来回答。

压缩映射

价值函数空间（value function space）

- 价值函数上的向量空间 V有|S| 维，这个空间中的每个点都完全指定了一个值函数 v(s)
- Bellman 备份对这个空间中的点有什么作用？
- 我们将证明它使价值函数更接近
- 因此备份必然会收敛到唯一解上

压缩映射

值函数∞-范数

- 我们将通过 ∞ 范数度量状态值函数 **u** 和 **v** 之间的距离
- 换句话说就是状态值之间的最大差异
- 价值函数的度量空间是完备空间

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

压缩映射

贝尔曼期望备份是一个压缩映射

- 定义 **Bellman** 期望备份算子 T^{π}，

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

- 这个算子是一个 γ 收缩，即它使价值函数至少接近 γ，

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

压缩映射定理

压缩映射定理

对于完备（封闭的）度量空间V下的，压缩映射算子 T(v)，其中 T 是 γ-收缩，

- T 收敛到一个唯一的不动点（Banach不动点定理）
- 以收敛率为 γ的线性速度收敛

对于**完备**的**metric space** $\langle M, d \rangle$，如果 $f : M \mapsto M$ 是它的一个**压缩映射**，那么

* 在该metric space中，存在唯一的点 x_* 满足 $f(x_*) = x_*$ 。

* 并且，对于任意的 $x \in M$，定义序列 $f^2(x) = f(f(x))$，$f^3(x) = f(f^2(x))$ ，$\cdots$，$f^n(x) = f(f^{n-1}(x))$，该序列会收敛于 x_* 即 $\lim_{n \to \infty} f^n(x) = x_*$

压缩映射

策略评估和策略迭代的收敛

- 贝尔曼期望算子 T^{π} 有一个唯一的不动点
- v_{π} 是 T^{π} 的不动点（根据贝尔曼期望方程）
- 由压缩映射定理知，迭代策略评估收敛于 v_{π} ，策略迭代收敛于 v_{*}

压缩映射

贝尔曼最优备份是收敛的

- 定义贝尔曼最优备份算子 ，T^*

$$T^*(v) = \max_{a \in A} R^a + \gamma P^a v$$

- 这个算子是一个 γ 收缩，即它使值函数至少接近 γ（类似于之前的证明，证明略）

$$\left\| T^*(u) - T^*(v) \right\|_\infty \leq \gamma \left\| u - v \right\|_\infty$$

压缩映射

价值迭代的收敛

- 贝尔曼最优算子 T^* 有一个唯一的不动点
- v_* 是 T^* 的不动点（通过贝尔曼最优方程）
- 由压缩映射定理得知，价值迭代收敛于 v_*

The End