Author：Liu Jian

Time：2021-09-01

# Dynamic Programming

动态规划是什么 (算法中的动态规划即递推+cache)： The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP 模型的动态特性已知).

Classical DP algorithms 在强化学习的应用中存在很大的限制，因为其假设 a perfect model 和 great computational expense，但在理论上是非常重要的，因为其为本书中将要介绍的其他方法的理解打下了重要的基础 (In fact, all of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.)

为了便于讨论，假设环境为**有限 MDP**，即 we assume that its state, action, and reward sets, $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$, are finite, and that its dynamics are given by a set of probabilities $p(s',r|s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$, $r \in \mathcal{R}$, and $s' \in \mathcal{S}^+$ ($\mathcal{S}^+$ is S plus a terminal state if the problem is episodic).

我们使用动态规划计算价值函数：DP algorithms are obtained by turning Bellman equations into assignments, that is, into update rules for improving approximations of the desired value functions (即基于 Bellman equations 得到递推/更新方程，再迭代 cache 计算价值函数，这也是这类方法叫动态规划的原因).

## 1 Policy Iteration

策略迭代 (Policy Iteration) 算法包含策略评估 (Policy Evaluation，也称 prediction problem) 和策略改进 (Policy Improvement) 两个迭代步。

---

策略评估即 how to compute the state-value function $v_\pi$ for an arbitrary policy $\pi$. 由 Bellman equation, for all $s \in \mathcal{S}$:

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

The existence and uniqueness of $v_\pi$ are guaranteed as long as either $\gamma < 1$ or eventual termination is guaranteed from all states under the policy $\pi$. 若 环境的动态特性完全已知，则上式给出了 a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns (the $v_\pi(s)$, $s \in \mathcal{S}$). 我们可以选择直接求解这个方程组，但这里，我们采用迭代的方式求解，定义更新公式，for all $s \in \mathcal{S}$:

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

注意，对于 terminal state (若存在)，其价值函数总置为 $0$。Indeed, the sequence $v_k$ can be shown in general to converge to $v_\pi$ as $k \to \infty$ under the same conditions that guarantee the existence of $v_\pi$. 类似地，我们也可以得到 action-value functions 的迭代更新公式。

显然，上述迭代策略评估算法还可以发展出 in-place 版本：With two arrays, the new values can be computed one by one from the old values without the old values being changed. Alternatively, you could use one array and update the values “in place,” that is, with each new value immediately overwriting the old one. 可以想见的是 in-place 版本通常比 two-array 版本收敛得更快，且各 state 的更新顺序对收敛速率也会有影响。一般我们默认 in-place 版本。

策略改进 (Policy Improvement)：对已有策略 $\pi$ 进行改进提高，得到新的策略 $\pi'$.

首先介绍 **policy improvement theorem**：Let $\pi$ and $\pi'$ be any pair of **deterministic policies (确定性策略)** such that, for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geqslant v_\pi(s)$$

（注意，不等式左边 $q_\pi(s, \pi'(s))$ 表示的是在状态 $s$ 下，根据策略 $\pi'$ 选择 action 后，接下来依旧按照策略 $\pi$ 而不是 $\pi'$ 进行操作得到的期望 return；此外，该定理针对的是确定性策略，$\pi'(s)$ 返回的是 action，而随机性策略 $\pi'(a|s)$ 返回的是一个数值，表示选择 action $a$ 的概率).Then the policy $\pi'$ must be as good as, or better than, $\pi$, i.e. for all $s \in \mathcal{S}$,

$$v_{\pi'}(s) \geqslant v_\pi(s)$$

且对某个 state，若前者不等号严格成立，即 $q_\pi(s, \pi'(s)) > v_\pi(s)$, 则后者不等号也严格成立 $v_{\pi'}(s) > v_\pi(s)$.

下面**证明 policy improvement theorem**:

$$\begin{aligned} v_\pi(s) \quad &\leqslant q_\pi(s, \pi'(s)) \\ &= \mathbb{E}\Big[R_{t+1} + \gamma G(S_{t+1}) \Big| S_t = s, A_t = \pi'(s)\Big] \end{aligned}$$

其中，$S_{t+1}, R_{t+1}$ 服从环境动态特性 $p(S_{t+1}, R_{t+1}|S_t = s, A_t = \pi'(s))$，即在状态 $s$ 下，第 $t$ 步采用策略 $\pi'$ 选择 action $\pi'(s)$，环境反馈 reward $R_{t+1}$ 并跳转到下一状态 $S_{t+1}$；接下来，由策略 $\pi$ 控制，即 return $G(S_{t+1})$ 是根据策略 $\pi$ 得到的，因此其期望就是 $v_\pi(S_{t+1})$，则：

$$= \mathbb{E}\Big[R_{t+1} + \gamma v_\pi(S_{t+1}) \Big| S_t = s, A_t = \pi'(s)\Big]$$

其中，给定 $S_{t+1}$ 后，$v_\pi(S_{t+1})$ 为常数，不受外围期望 $\mathbb{E}$ 的影响，而 $S_{t+1}, R_{t+1}$ 则是 agent 在状态 $s$ 下采用策略 $\pi'$ 后环境的反馈和变化，因此：

$$= \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma v_\pi(S_{t+1}) \Big| S_t = s\Big]$$

类似上述分析，我们有：

$$\begin{aligned} v_\pi(s) \quad &\leqslant \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma v_\pi(S_{t+1}) \Big| S_t = s\Big] \\ &\leqslant \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \Big| S_t = s\Big] \\ &= \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma \mathbb{E}\Big[R_{t+2} + \gamma v_\pi(S_{t+2}) \Big| S_{t+1}, A_{t+1} = \pi'(S_{t+1})\Big] \Big| S_t = s\Big] \\ &= \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \Big| S_t = s\Big] \\ &\vdots \\ &\leqslant \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \Big| S_t = s\Big] \\ &= v_{\pi'}(s) \end{aligned}$$

证毕。

> **policy improvement theorem 的一个推论**如下：an original deterministic policy, $\pi$, and a changed policy, $\pi'$, that is identical to $\pi$ except that $\pi'(s) = a \neq \pi(s)$. For states other than $s$, $s' \neq s$, $q_\pi(s', \pi'(s')) \geqslant v_\pi(s')$ holds because the two sides are equal. Thus, if $q_\pi(s, \pi'(s)) > v_\pi(s)$, then the changed policy is indeed better than $\pi$.

> 上面的讨论针对**确定性策略**，**随机策略 (stochastic policy)**的情况也类似 (**确定性策略的情况可视为随机策略情况的特例**)：
>
> 若 for all $s \in \mathcal{S}$,
>
> $$\mathbb{E}_{a\sim\pi'(a|s)}[q_\pi(s,a)] = \sum_a \pi'(a|s) q_\pi(s,a) \geqslant v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$

> (注意，$\mathbb{E}_{\pi'}, \mathbb{E}_{a\sim\pi'(a|s)}$ 是两个不同的记号，前者取强化学习中的含义，后者取概率论中的含义)，则 for all $s \in \mathcal{S}$,
>
> $$v_{\pi'}(s) \geqslant v_\pi(s)$$
>
> 即 $\pi' \geqslant \pi$ .
>
> 证明类似：
>
> $$\begin{aligned} v_\pi(s) \quad &\leqslant \textstyle\sum_{A_t} \pi'(A_t|s) q_\pi(s, A_t) \\ &= \textstyle\sum_{A_t} \pi'(A_t|s) \sum_{S_{t+1}, R_{t+1}} p(S_{t+1}, R_{t+1}|s, A_t)(R_{t+1} + \gamma v_\pi(S_{t+1})) \\ &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \middle| S_t = s\right] (\text{头一步使用策略}\pi'\text{，后续使用策略}\pi) \\ &\leqslant \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \textstyle\sum_{A_{t+1}} \pi'(A_{t+1}|S_{t+1}) q_\pi(S_{t+1}, A_{t+1}) \middle| S_t = s\right] \\ &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \textstyle\sum_{A_{t+1}} \pi'(A_{t+1}|S_{t+1}) \sum_{S_{t+2}, R_{t+2}} p(S_{t+2}, R_{t+2}|S_{t+1}, A_{t+1})(R_{t+2} + \gamma v_\pi(S_{t+2})) \middle| S_t = s\right] \\ &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \middle| S_t = s\right] (\text{头两步使用策略}\pi'\text{，后续使用策略}\pi) \\ &\vdots \\ &\leqslant \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \middle| S_t = s\right] (\text{所有步均使用策略}\pi') \\ &= v_{\pi'}(s) \end{aligned}$$

基于 policy improvement theorem，我们可基于已知策略 $\pi$ 构造一个新的策略 $\pi'$ 使新策略不差于原策略：for all state $s \in \mathcal{S}$,

$$\begin{aligned} \pi'(s) \quad &\triangleq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \textstyle\sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s')) \end{aligned}$$

其中，原策略 $\pi$ 的价值函数由前面介绍的 Policy Evaluation 得到。可以看到，构造的新策略 $\pi'$ 满足 policy improvement theorem 的条件，

$$v_\pi(s) = q_\pi(s, \pi(s)) \leqslant \max_a q_\pi(s, a) = q_\pi(s, \pi'(s))$$

因此 $\pi' \geqslant \pi$. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called **policy improvement**.

进一步，若新策略 $\pi'$ 与原策略 $\pi$ 的价值函数相等 $v_{\pi'}(s) = v_\pi(s)$， for all $s \in \mathcal{S}$，则策略 $\pi'$ 和 $\pi$ 均为最优策略. 由前可知：

$$\begin{aligned} \pi'(s) \quad &\triangleq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \textstyle\sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s')) \\ &= \arg\max_a \textstyle\sum_{s', r} p(s', r|s, a)(r + \gamma v_{\pi'}(s')) \\ &= \arg\max_a q_{\pi'}(s, a) \end{aligned}$$

又由 $v_{\pi'}(s) = q_{\pi'}(s, \pi'(s))$，则：

$$\begin{aligned} v_{\pi'}(s) \quad &= q_{\pi'}(s, \pi'(s)) \\ &= \max_a q_{\pi'}(s, a) \\ &= \max_a \textstyle\sum_{s', r} p(s', r|s, a)(r + \gamma v_{\pi'}(s')) \end{aligned}$$

满足 Bellman optimality equation，证毕. 可以看到，Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

> 当 $\pi$ 是随机性策略时，我们依然可以令 for all state $s \in \mathcal{S}$:
>
> $$\begin{aligned} \pi'(s) \quad &\triangleq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \textstyle\sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s')) \end{aligned}$$

> 此外，若最大值有多个 action 都能达到，新构造的策略 $\pi'$ 既可以是确定性的也可以是随机性的：if there are several actions at which the maximum is achieved—then we need not select a single action from among them. Instead, each maximizing action can be given a portion of the probability of being selected in the new greedy policy.
>
> 构造的新策略 $\pi'$ 满足 policy improvement theorem 的条件:
>
> $$v_\pi(s) = \sum_a \pi(a|s)q_\pi(s,a) \leqslant \max_a q_\pi(s,a) = q_\pi(s,\pi'(s))$$
>
> 因此 $\pi' \geqslant \pi$.
>
> 进一步，若新策略 $\pi'$ 与原策略 $\pi$ 的价值函数相等 $v_{\pi'}(s) = v_\pi(s)$， for all $s \in \mathcal{S}$，则策略 $\pi'$ 和 $\pi$ 均为最优策略，证明过程类似，不再赘述.

---

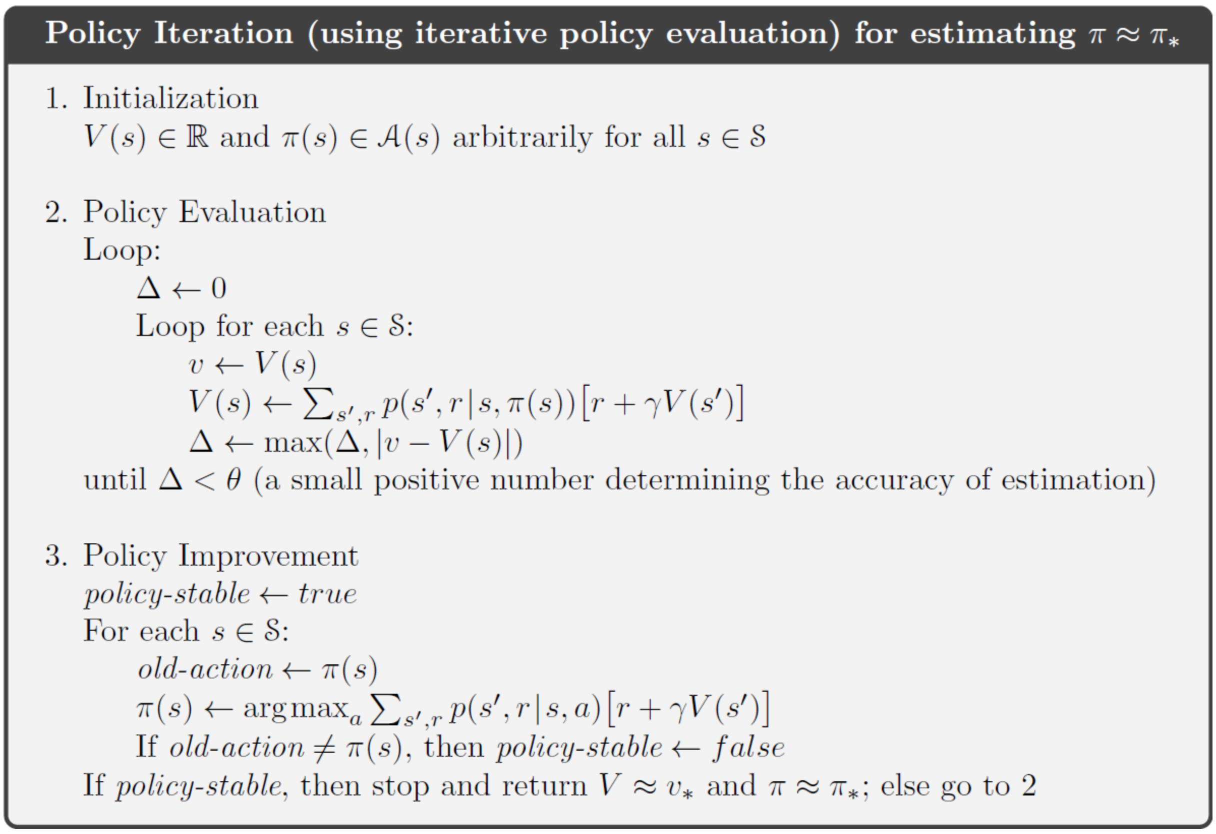
策略迭代 (Policy Iteration)

Once a policy, $\pi$, has been improved using $v_\pi$ to yield a better policy, $\pi'$, we can then compute $v_{\pi'}$ and improve it again to yield an even better $\pi''$. We can thus obtain a sequence of monotonically improving policies and value functions:

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

where $\xrightarrow{\text{E}}$ denotes a policy evaluation and $\xrightarrow{\text{I}}$ denotes a policy improvement. Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and the optimal value function in a finite number of iterations.

Note that each policy evaluation, itself an iterative computation, is started with the value function for the previous policy. This typically results in a great increase in the speed of convergence of policy evaluation (presumably because the value function changes little from one policy to the next).



**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 $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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* $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       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
```

## 2 Value Iteration

Value iteration 有两种推导方式，一种是从 policy iteration 推得，一种是直接从 Bellman 最优方程推得：

- policy iteration 包含 policy evaluation 和 policy improvement 两个迭代步。此外，每一次 policy evaluation 也通过迭代计算实现，会多次扫描所有状态，直到 evaluation 达到一定精度为止。这里，我们在进行 policy evaluation 时只进行一轮更新，此时 evaluation 还没有完全收敛就进行 policy improvement。将 policy evaluation 的一轮更新和 policy improvement 整合为一步可得：

  $$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big]$$

  for all $s \in \mathcal{S}$，此即为 value iteration 的迭代更新公式。For arbitrary $v_0$, the sequence $v_k$ can be shown to converge to $v_*$ under the same conditions that guarantee the existence of $v_*$.
- 由 Bellman optimality equation $v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$ 直接给出迭代更新公式：

  $$v_{k+1}(s) \triangleq \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big]$$

-

对于 action values $q_{k+1}(s,a)$ 我们也可以得到类似的 value iteration update。

---

我们称迭代计算时对所有 state 的一轮扫描为一次 sweep：

- policy evaluation 的一次 sweep：

  $$\text{for all } s \in \mathcal{S}$$
  $$v_{k+1}(s) = \sum_{s',r} p(s',r|s,\pi(s))\Big[r + \gamma v_k(s')\Big]$$

  注意，为了得到满足一定精度的 $v_\pi$，policy evaluation 包含多次 sweep。
- policy improvement 则只需进行一次如下的 sweep：

  $$\text{for all } s \in \mathcal{S}$$
  $$\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big]$$

  policy evaluation 的一次 sweep + policy improvement 的一次 sweep 可整合为如下的一次 sweep：

  $$\text{for all } s \in \mathcal{S}$$
  $$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big]$$

  **下面我们不妨直接称这样的一次 sweep 为 policy improvement (下文 value iteration 伪代码中的 policy evaluation 就是采用的这种做法，而不是像前面返回的得到 $\pi_{k+1}(s)$)**。注意，policy evaluation 中的 $v_k$ 均是对策略 $\pi$ 价值函数的近似，而 policy improvement 中的 $v_k$ 是指第 $k$ 个策略 $\pi_k$ 的价值函数，二者含义不同。

**可以看到，evaluation 和 improvement 每次 sweep 的更新公式区别只在于对 action 进行最大寻优而不是直接选择 $\pi(s)$ (the max operation is the only difference between these updates)**。基于上述内容，**我们可以对策略迭代、价值迭代以及更一般的 truncated policy iteration algorithms 总结如下**：Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement. Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep. **In general, the entire class of truncated policy iteration algorithms can be thought of as sequences of sweeps, some of which use policy evaluation updates and some of which use value iteration updates. Because the max operation is the only difference between these updates, this just means that the max operation is added to some sweeps of policy evaluation**. All of these algorithms converge to an optimal policy for discounted finite MDPs.

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

## 3 Asynchronous Dynamic Programming

前文介绍的动态规划算法的缺点是以 sweep 为 block，一次 sweep 意味着对所有的 state 更新一遍，当 state 很大时计算很困难 (A major drawback to the DP methods that we have discussed so far is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set)。

异步动态规划算法是什么：Asynchronous DP algorithms are **in-place** iterative DP algorithms **that are not organized in terms of systematic sweeps of the state set**. These algorithms update the values of states in any order whatsoever, using whatever values of other states happen to be available. The values of some states may be updated several times before the values of others are updated once. 异步动态规划算法是一种**就地**迭代 DP 算法，它不再以 sweep 为单位进行一轮迭代更新，即不要求在下次更新某个状态前把除了该状态以外的其他所有状态都更新一遍，在其他状态的价值函数更新一次之前，某些状态的价值函数可能会被更新多次。可以看到，异步动态规划算法以某个状态的更新为 block 进行构建，即：

$$\text{select } s \in \mathcal{S}, \quad \begin{array}{c} v_{k+1}(s) = \sum_{s',r} p(s',r|s,\pi(s))\Big[r + \gamma v_k(s')\Big] \\ \text{or} \\ v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big] \end{array}$$

构建前文介绍的同步动态规划算法的 block 为：

$$\text{for all } s \in \mathcal{S}, \quad \begin{array}{c} v_{k+1}(s) = \sum_{s',r} p(s',r|s,\pi(s))\Big[r + \gamma v_k(s')\Big] \\ \text{or} \\ v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big] \end{array}$$

异步动态规划算法的收敛性保证：To converge correctly, however, an asynchronous algorithm must continue to update the values of all the states: it can't ignore any state after some point in the computation. 为了保证收敛，异步动态算法需要能**持续地对所有状态进行更新**，即无论在哪一时间步之后，所有的状态都会被选中更新，虽然它们更新的频次可能有高有低。

异步动态规划算法举例：

1. value iteration 的异步版本：one version of asynchronous value iteration updates the value, in place, of only one state, $s_k$, on each step, $k$, using the value iteration update (这里的 $s_k$ 是指第 $k$ 次价值迭代更新时选择的 state，而不是第 $k$ 种 state). 收敛性：If $0 \leqslant \gamma < 1$, asymptotic convergence to $v_\pi$ is guaranteed given only that all states occur in the sequence $\{s_k\}$ an infinite number of times (the sequence could even be random). In the undiscounted episodic case, it is possible that there are some orderings of updates that do not result in convergence, but it is relatively easy to avoid these.
2. Similarly, it is possible to intermix policy evaluation and value iteration updates to produce a kind of asynchronous truncated policy iteration.

DP 算法的好处：

1. Of course, avoiding sweeps does not necessarily mean that we can get away with less computation. It just means that an algorithm does not need to get locked into any hopelessly long sweep before it can make progress improving a policy. 具体而言：
    - We can try to take advantage of this flexibility by selecting the states to which we apply updates so as to improve the algorithm's rate of progress (通过灵活选择更新 state，可以提高算法改进的速率). We can try to order the updates to let value information propagate from state to state in an efficient way (提高 state 间信息传播的效率). Some states may not need their values updated as often as others. We might even try to skip updating some states entirely if they are not relevant to optimal behavior. Some ideas for doing this are discussed in Chapter 8.
2. Asynchronous algorithms also make it easier to intermix computation with real-time interaction. 具体而言：
    - To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP. The agent's experience can be used to determine the states to which the DP algorithm applies its updates. At the same time, the latest value and policy information from the DP algorithm can guide the agent's decision making. For example, we can apply updates to states as the agent visits them. This makes it possible to focus the DP algorithm's updates onto parts of the state set that are most relevant to the agent. This kind of focusing is a repeated theme in reinforcement learning.

---

对上面介绍的三种 DP 方法可总结如下：

(策略迭代) Policy iteration consists of two simultaneous, interacting processes, one making the value function consistent with the current policy (policy evaluation), and the other making the policy greedy with respect to the current value function (policy improvement). In policy iteration, these two processes alternate, each completing before the other begins, but this is not really necessary.// (价值迭代) In value iteration, for example, only a single iteration of policy evaluation is performed in between each policy improvement.// (异步动态规划) In asynchronous DP methods, the evaluation and improvement processes are interleaved at an even finer grain. In some cases a single state is updated in one process before returning to the other. As long as both processes continue to update all states, the ultimate result is typically the same—convergence to the optimal value function and an optimal policy.

我们统称所有的方法为 Generalized Policy Iteration (GPI)：We use the term generalized policy iteration (GPI) to refer to the general idea of letting policy-evaluation and policy improvement processes interact, independent of the granularity (间隔尺寸，[岩] 粒度) and other details of the two processes.或者说 GPI is the general idea of two interacting processes revolving around an approximate policy and an approximate value function.

- One process takes the policy as given and performs some form of policy evaluation, changing the value function to be more like the true value function for the policy. The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function. Although each process changes the basis for the other, overall they work together to find a joint solution: a policy and value function that are unchanged by either process and, consequently, are optimal. 个人感觉 policy-evaluation process 和 policy improvement process 迭代前进到达最优的过程与坐标上升法类似。
- Asynchronous DP methods are in-place iterative methods that update states in an arbitrary order, perhaps stochastically determined and using out-of-date information. Many of these methods can be viewed as fine-grained forms of GPI.

关于收敛性：In some cases, GPI can be proved to converge, most notably for the classical DP methods that we have presented in this chapter. In other cases convergence has not been proved, but still the idea of GPI improves our understanding of the methods.

## 4 Efficiency of Dynamic Programming

DP may not be practical for very large problems, but compared with other methods for solving MDPs (比如，direct search，Linear programming methods 等), DP methods are actually quite efficient：

- If $n$ and $k$ denote the number of states and actions, this means that a DP method takes a number of computational operations that is less than some polynomial function of $n$ and $k$. A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is $k^n$.

- In this sense, DP is exponentially faster than any direct search in policy space could be, because direct search would have to exhaustively examine each policy to provide the same guarantee.
- Linear programming methods can also be used to solve MDPs, and in some cases their worst-case convergence guarantees are better than those of DP methods. But linear programming methods become impractical at a much smaller number of states than do DP methods (by a factor of about $100$). For the largest problems, only DP methods are feasible.

DP is sometimes thought to be of limited applicability because of the curse of dimensionality, the fact that the number of states often grows exponentially with the number of state variables. 但维度诅咒是问题本身固有的困难，而不是只针对 DP 方法，即并不是因采用 DP 算法而导致的。In fact, DP is comparatively better suited to handling large state spaces than competing methods such as direct search and linear programming.

In practice, DP methods can be used with today’s computers to solve MDPs with millions of states. Both policy iteration and value iteration are widely used, and it is not clear which, if either, is better in general. In practice, these methods usually converge much faster than their theoretical worst-case run times, particularly if they are started with good initial value functions or policies.

On problems with large state spaces, asynchronous methods and other variations of GPI can be applied in such cases and may find good or optimal policies much faster than synchronous methods can because relatively few states occur along optimal solution trajectories.

---

Although DP ideas can be applied to problems with continuous state and action spaces, exact solutions are possible only in special cases. A common way of obtaining approximate solutions for tasks with continuous states and actions is to quantize the state and action spaces and then apply finite-state DP methods. The methods we explore in Part II are applicable to continuous problems and are a significant extension of that approach.

Finally, we note one last special property of DP methods. All of them update estimates of the values of states based on estimates of the values of successor states. That is, they update estimates on the basis of other estimates. We call this general idea **bootstrapping**. Many reinforcement learning methods perform bootstrapping, even those that do not require, as DP requires, a complete and accurate model of the environment. In the next chapter (Chapter 5 Monte Carlo Methods) we explore reinforcement learning methods that do not require a model and do not bootstrap. In the chapter after that (Chapter 6 Temporal-Difference Learning) we explore methods that do not require a model but do bootstrap. These key features and properties are separable, yet can be mixed in interesting combinations.

---