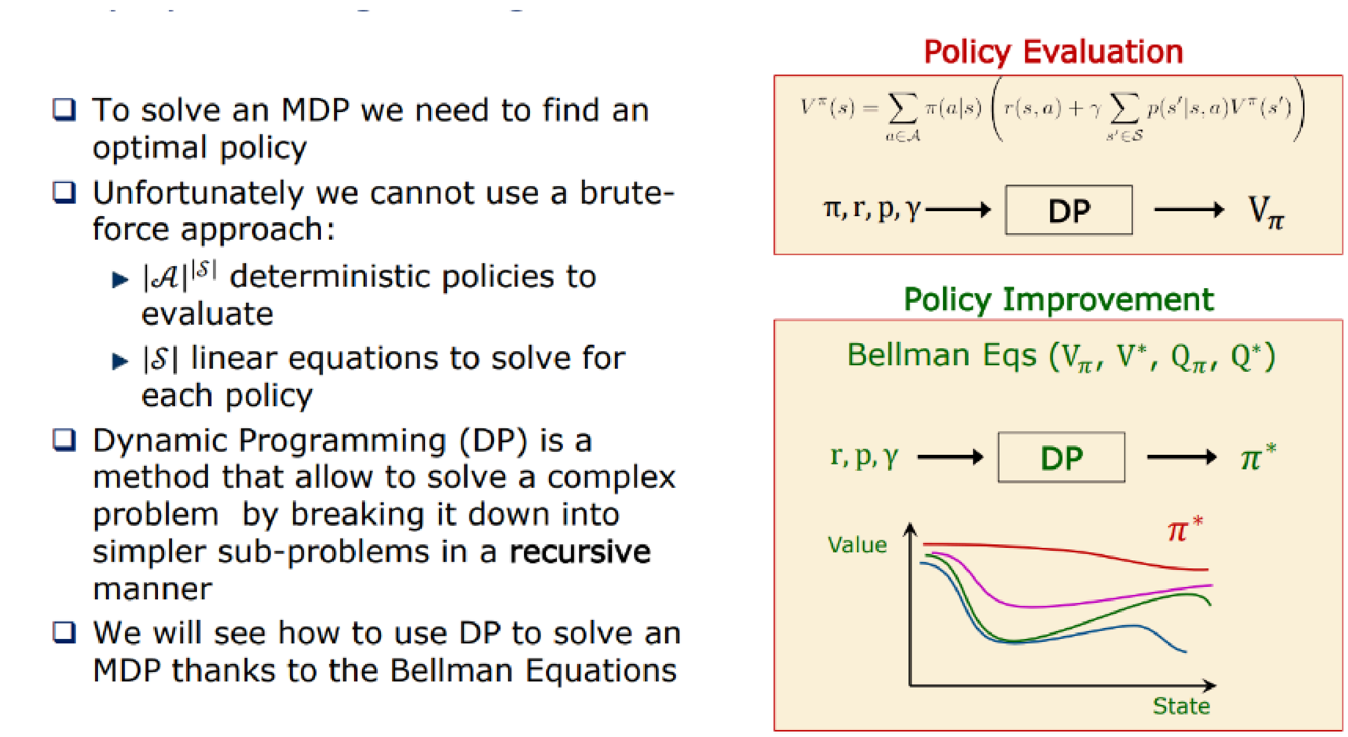
Dynamic Programming

First technique to try to resolve the problem of finding the optimal policy with nonlinear equations and not knowing π^*.
Dynamic programming is approaching the problem recursively. We will see two algorithm:



- To solve an MDP we need to find an optimal policy
- Unfortunately we cannot use a brute-force approach:
  - $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies to evaluate
  - $|\mathcal{S}|$ linear equations to solve for each policy
- Dynamic Programming (DP) is a method that allow to solve a complex problem by breaking it down into simpler sub-problems in a **recursive** manner
- We will see how to use DP to solve an MDP thanks to the Bellman Equations

Policy Evaluation

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)V^\pi(s')\right)$$

$$\pi, r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow V_\pi$$

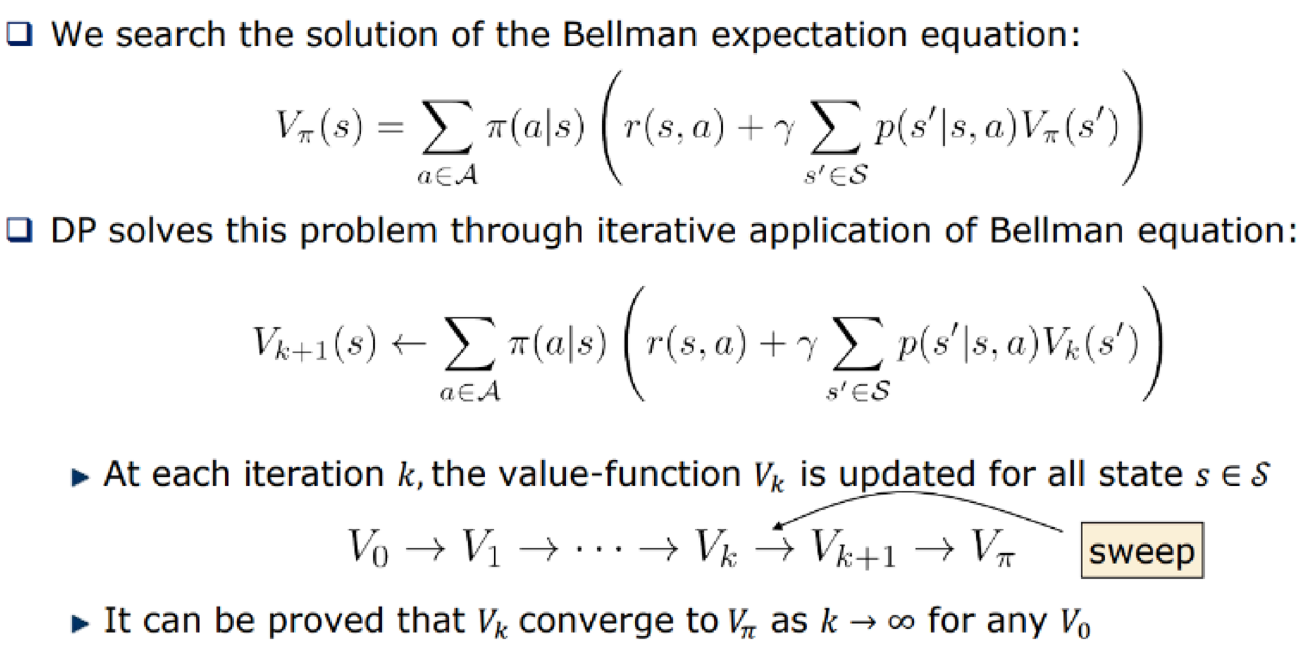
Policy Improvement

Bellman Eqs (V_π, V^*, Q_π, Q^*)

$$r, p, \gamma \longrightarrow \boxed{\text{DP}} \longrightarrow \pi^*$$

Policy Evaluation

The algorithm is simple: if you apply the Bellman Expectation Equation for all the state of the problem and use a specified verify function. You go through all the states applying the Bellman Expectation Equation using all the previous found values as the value function.
This lead to get the true value of the policy function



- We search the solution of the Bellman expectation equation:

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)V_\pi(s')\right)$$

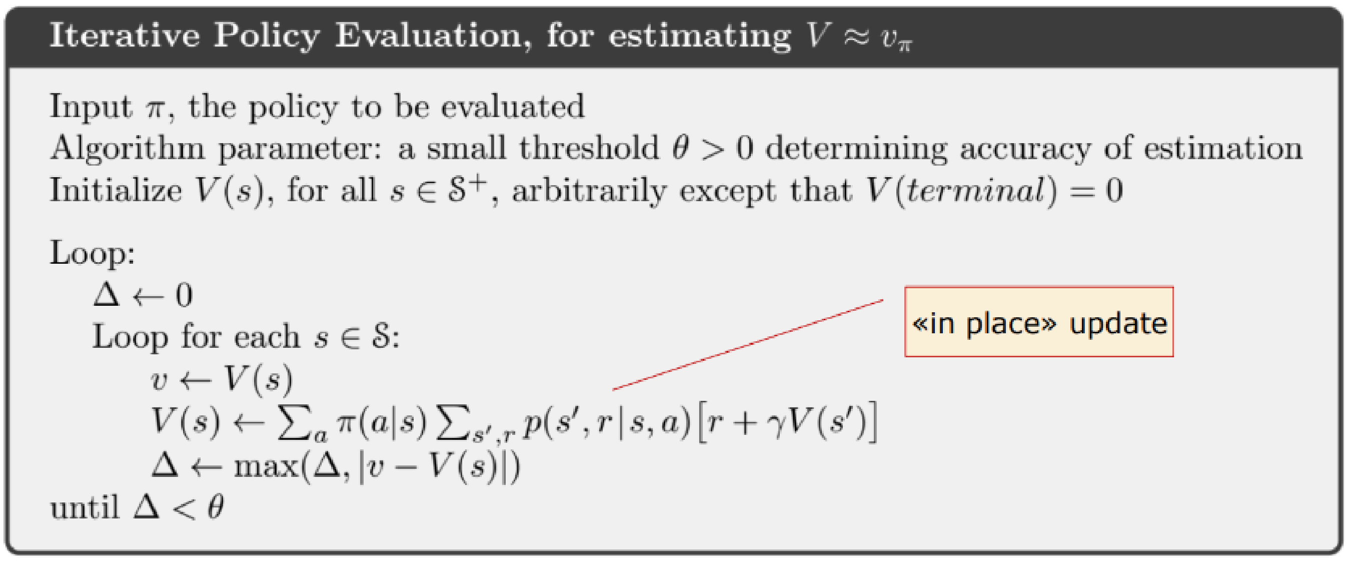
- DP solves this problem through iterative application of Bellman equation:

$$V_{k+1}(s) \leftarrow \sum_{a\in\mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)V_k(s')\right)$$

  - At each iteration k, the value-function V_k is updated for all state $s \in \mathcal{S}$

$$\begin{tikzcd} V_0 \arrow[r] & V_1 \arrow[r] & \cdots \arrow[r] & V_k \arrow[r] & V_{k+1} \arrow[r] \arrow[l, bend right, ""] & V_\pi & \boxed{\text{sweep}} \end{tikzcd}$$

  - It can be proved that V_k converge to V_π as $k \to \infty$ for any V_0



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

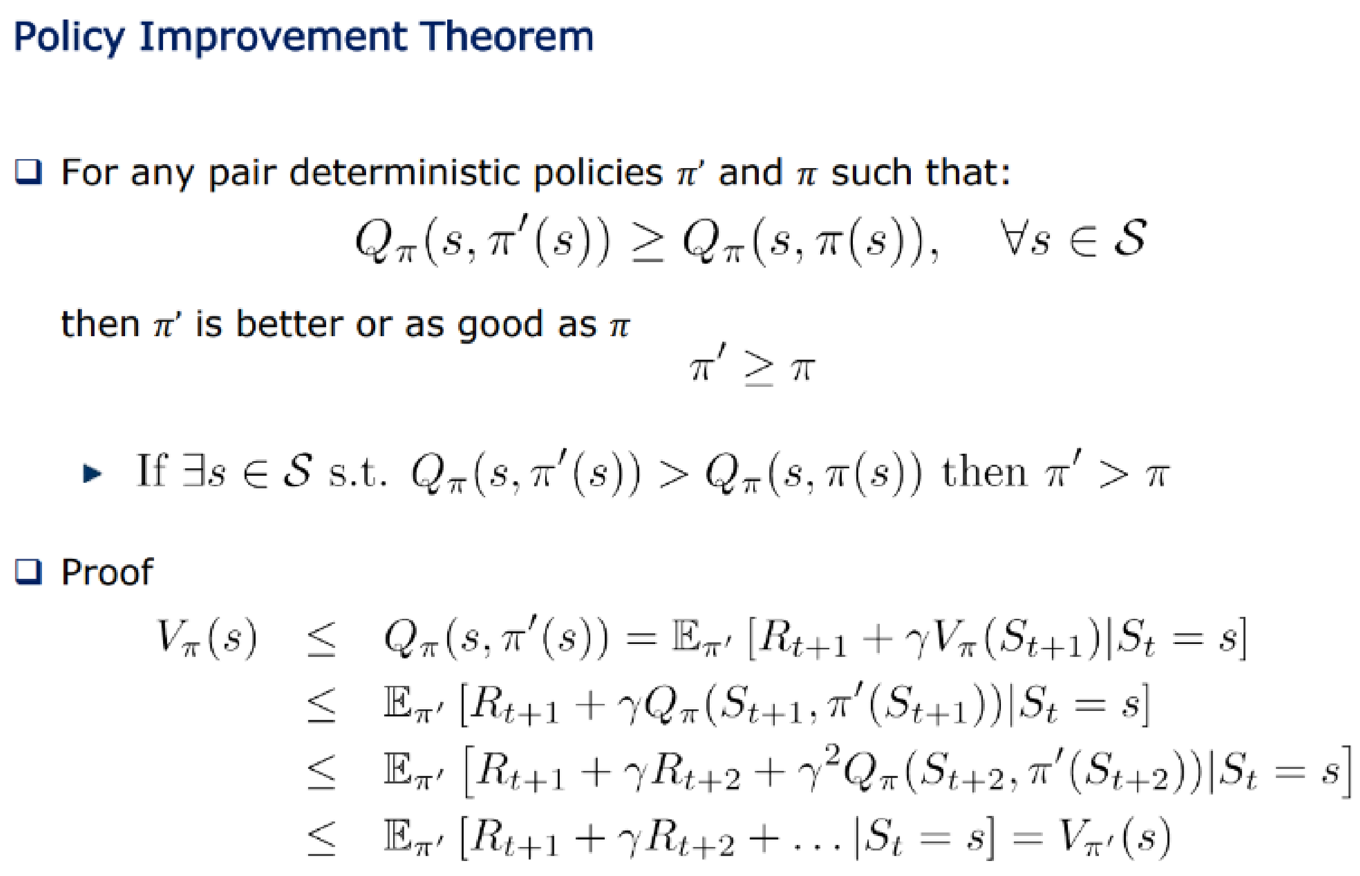
```
Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

«in place» update

Policy Improvement

I can derive the optimal policy computing the optimal policy selecting the action a from the optimal choice.
In dynamic programming I have to do that on a general function π non optimal.
If I apply this greedy algorithm choosing the action greedly from what I know? The answer is that there are two options:

- π' is equal to π it means π is already the optimal policy π^*
- π' is as good as π(easy way to compute the optimal policy)



Policy Improvement Theorem

- For any pair deterministic policies π' and π such that:

$$Q_\pi(s, \pi'(s)) \geq Q_\pi(s, \pi(s)), \quad \forall s \in \mathcal{S}$$

  then π' is better or as good as π

$$\pi' \geq \pi$$

  - If $\exists s \in \mathcal{S}$ s.t. $Q_\pi(s, \pi'(s)) > Q_\pi(s, \pi(s))$ then $\pi' > \pi$

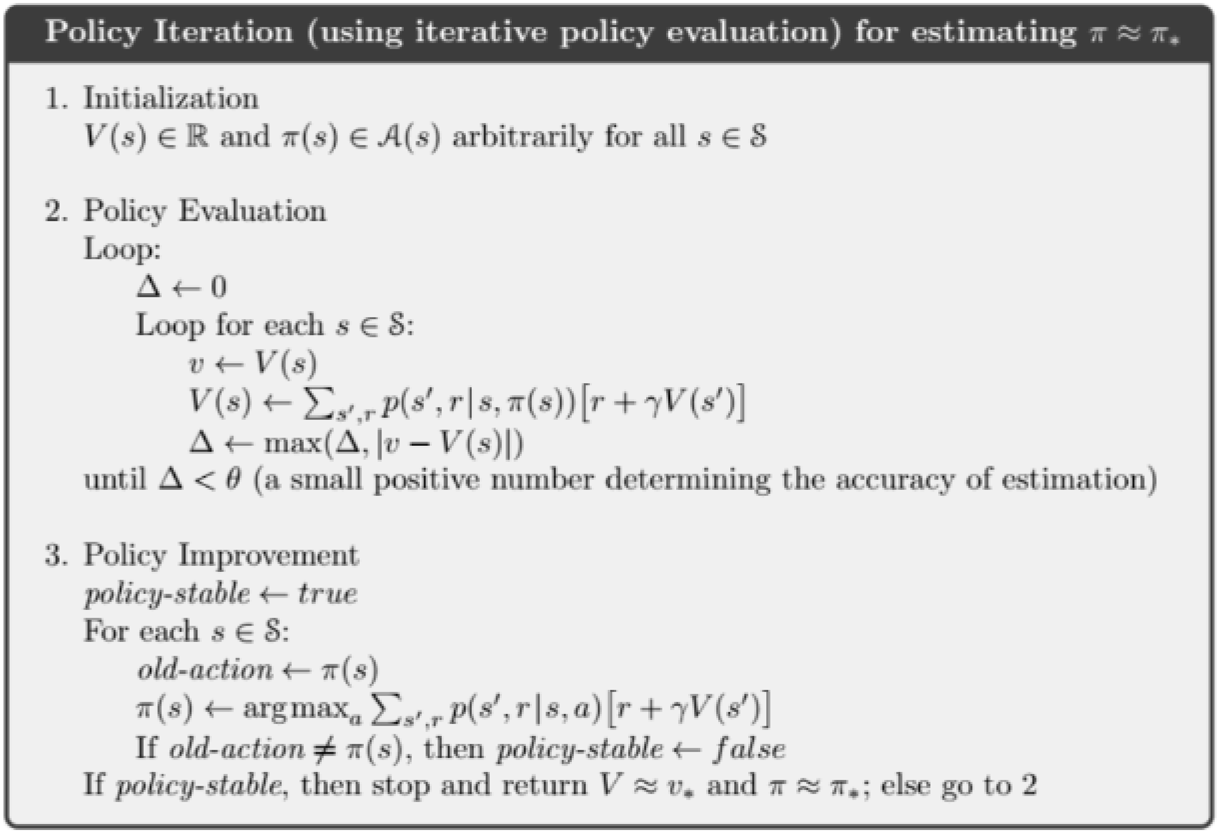
- Proof

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

With this theorem I cannot get something worse only something as good or better than what I already have.

Policy Iteration

Idea is keep iterating the policy evaluation starting from a policy, even random policy. After the first policy evaluation I apply a policy improvement to find a new policy that maximise the initial policy. At this point I repeat these steps until I obtain an optimal policy. At the end you know that you reached π^* when the improvement step is not changing the policy(This is by definition). Guaranteed to reach this result but it is not guaranteed the number of steps to reach it.



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* $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       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
```

To propagate the information of a policy optimization is slow as it takes at east one iteration to get to propagate the information

Generalized Policy Iteration

Instead to alternate evaluation and improvement we try to do a small number of policy evaluation and improvement doing partial evaluation and improvement still converging to the optimal policy.

Value Iteration

Takes the idea to the extreme combining together the update of the policy evaluation and policy improvement. This is done doing the update with respect to Bellman equation considering the greedy version of the policy computed by the policy improvement.

- In the policy evaluation step, only a single sweep of updates is performed:

$$\pi'(s) = \arg\max_a \left\{ r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right\}, \forall s \in \mathcal{S}$$

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi'(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_k(s') \right), \forall s \in \mathcal{S}$$

- Combining them, we simply need to iterate the update of the value function using the Bellman optimality equation:

$$V_{k+1}(s) \leftarrow \max_a \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V_k(s') \right], \forall s \in \mathcal{S}$$

- It can be proved that $\lim_{k \to \infty} V_k = V^*$

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

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

Even simpler as it takes only one cycle to complete every step.

Efficiency of DP

The DP implementation can or cannot be useful in programs.
The Asynchronous Dynamic Programming is one idea to have an improvement. The idea is that in a program tries to update from random state to converge to the solution and search one optimal way. One drawback is that you could not find the best optimal way.
It can scale up to few million of states and then start to struggle. If you have more states you can scale the problem using parallel programming.
Linear programming approaches can be also used instead of DP but they do not typically scale well on larger problems.
DP is extremely effective with not gigantic problems , but doesn't scale well on gigantic

problems. To solve some problems/limitations of DP we can use two families of solutions: Montecarlo's Families and Bandits