Dynamic Programming

presented by Jason TOKO

Outline

- 1、Introduction
- 2、Policy Evaluation
- 3、Policy Improvement
- 4、Policy Iteration
- 5、Value Iteration
- 6、Asynchronous Dynamic Programming

Introduction

Introduction

- DP algorithm is used for planning in an MDP
- It assumes full knowledge of MDP
- Prediction Problem:
  - Input: MDP and policy π
  - Output: value function v_π
- Control Problem:
  - Input: MDP
  - Output: optimal value function v_*

    optimal policy π_*

Introduction

- Key idea of DP:
  - Use value functions to organize and structure the search for good policies
  - Turn the Bellman equation into update rules

Policy Evaluation

Policy Evaluation

- Policy Evaluation is used to compute v_π for an arbitrary policy π
- Bellman expectation equations:

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

Policy Evaluation

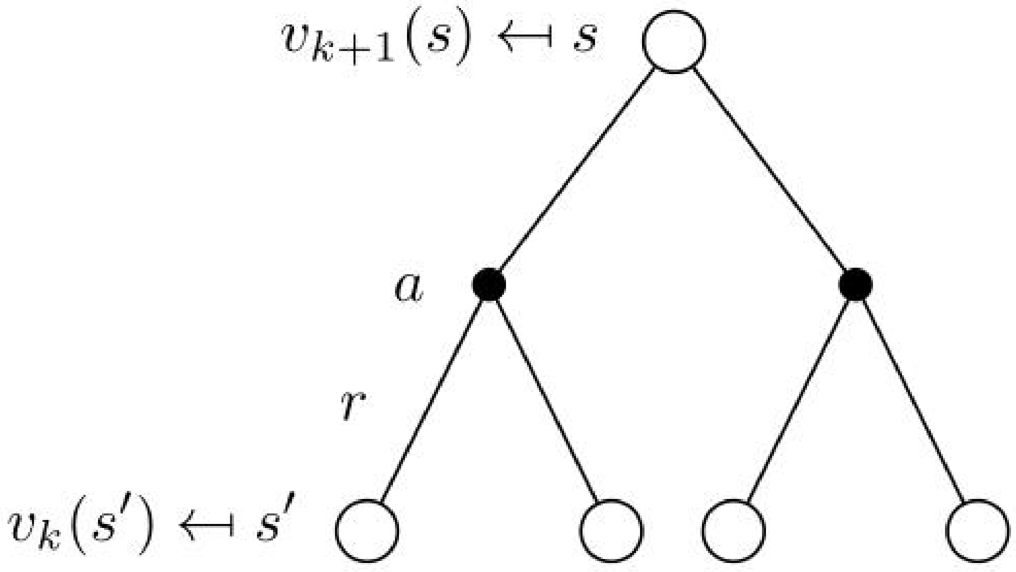
- Direct solution: |S| simultaneous linear equations in |S| unknowns
- Iteration solution: iterative application of Bellman expectation backup (synchronous)

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

- The sequence $\{v_k\}$ can converge to v_π as $k \to \infty$

$$v_1 \to v_2 \to v_3 \to \ldots \to v_k \to \ldots \to v_\pi$$

Iterative Policy Evaluation



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

Iterative Policy Evaluation

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 $s \in \mathcal{S}$:

        $v \leftarrow V(s)$

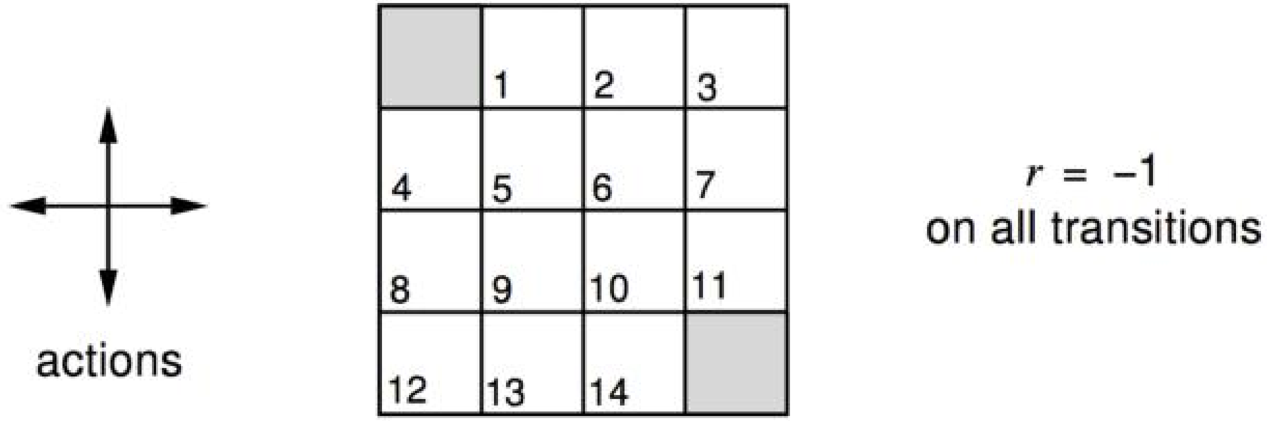
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$

        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Example: Small Gridworld



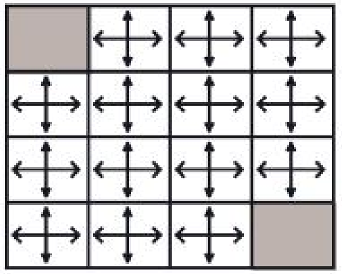
- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1,…,14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is −1 until the terminal state is reached
- Agent follows uniform random policy

Example: Small Gridworld

v_k for the Random Policy | Greedy Policy w.r.t. v_k

$k = 0$

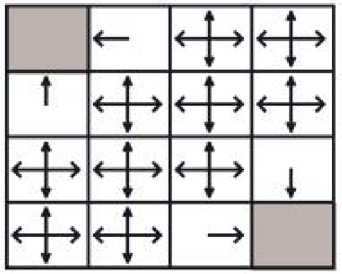
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

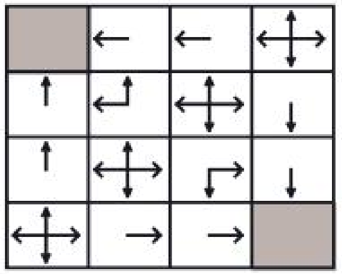
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Example: Small Gridworld

$k = 3$

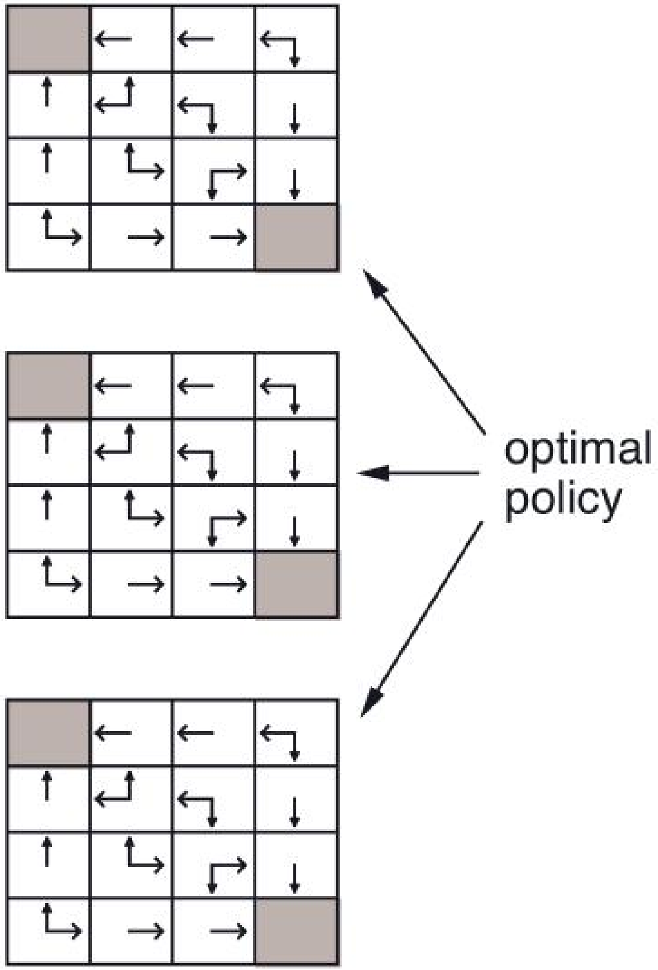
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

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



Policy Improvement

Policy Improvement

- Use value function to help find better policies
- How to do it?

$q_\pi(s,a)$

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

$v_\pi(s)$

  - And compare it to

Policy Improvement

- **Policy Improvement Theorem**
- *If it is satisfied for all s ∈ S that*

  $$q_\pi(s, \pi'(s)) \geq v_\pi(s).$$

- *Then we have conclusion for all s ∈ S that*

  $$v_{\pi'}(s) \geq v_\pi(s).$$

- Thus $\pi' \geq \pi$

Policy Improvement

- **Proof:**

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

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can improve the policy by acting **greedily**

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_\pi(s, a)$$

- This improves the value from any state s over one step

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

- It therefore in $v_{\pi'}(s) \geq v_\pi(s)$ alue function,

Policy Improvement

- If the new greedy policy π' is as good as the old policy π

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

- Then the Bellman optimality equation has been satisfied

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

- Therefore $v_\pi(s) = v_*(s)$ and $\pi = \pi_*$
- **Conclusion**: Policy improvement must give us a strictly better policy except when the original policy is already optimal

Policy Improvement

- If a stochastic policy…
- If there are several actions achieve maximum…
- Example…

Policy Iteration

Policy Iteration

- When we combine policy evaluation and policy improvement :
- Given a policy π
  - Evaluate the policy π

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

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

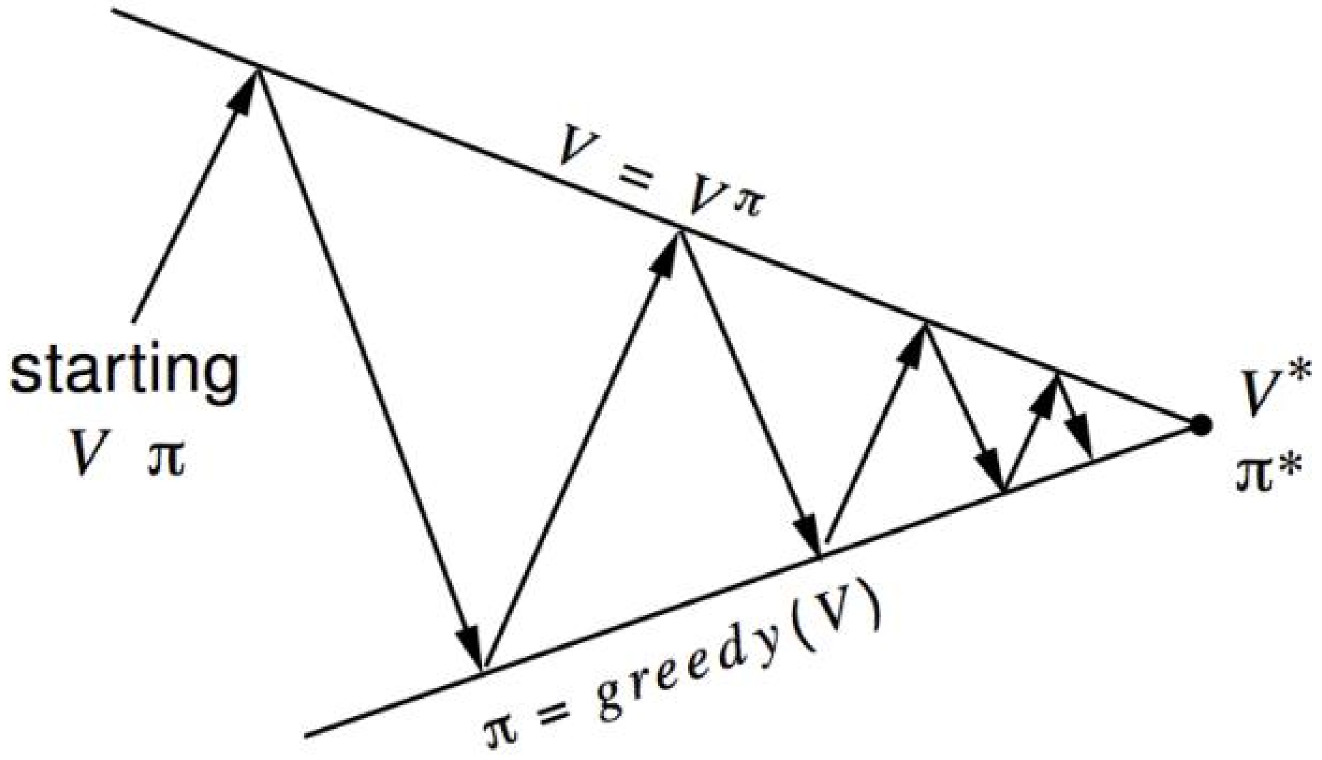
Policy Iteration

- And we repeat this process again and again…
- Then we will get a sequence:

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

- The policy will be strictly improved
- This must converge in a finite number of iterations

Policy Iteration



Policy Iteration

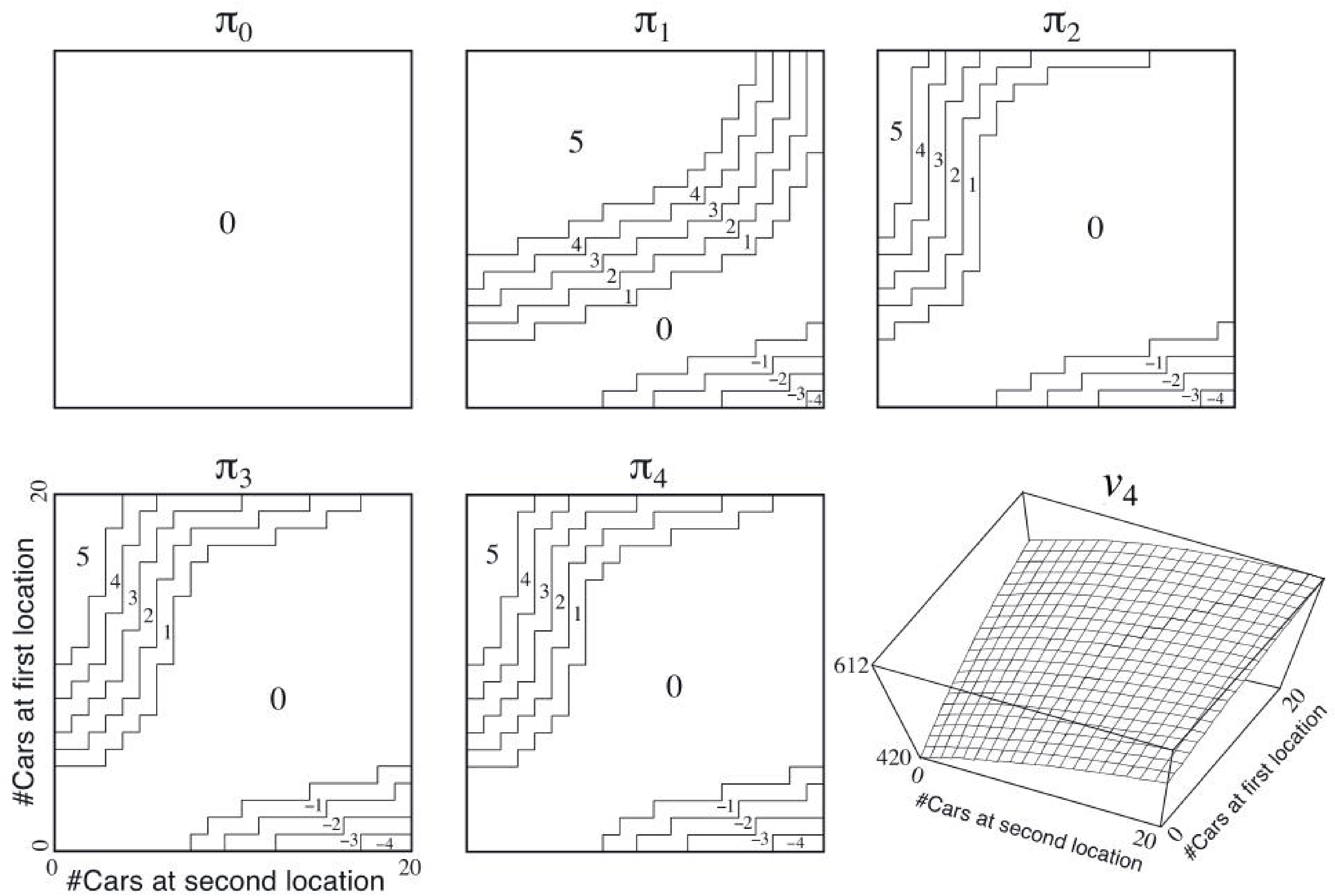
Policy iteration (using iterative policy evaluation)

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

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

Example: Jack's Car Rental



Modified Policy Iteration

- Does policy evaluation need to converge to v_π ?
- **Modification:**
- ①Introduce a stopping condition
  - e.g. stop when $\max_{s \in \mathcal{S}} |v_{k+1}(s) - v_k(s)|$ is sufficiently small
- ②Stop after k iterations of iterative policy evaluation
  - If k=1, this is equivalent to value iteration~

Value Iteration

Value Iteration

- If we combine the policy improvement and one-step policy evaluation, we can get a backup

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

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

- For arbitrary v_0, the sequence $\{v_k\}$ can converge to v_*

$$v_0 \to v_1 \to v_2 \to \dots \to v_k \to \dots \to v_*$$

Value Iteration

- Some points:
  - Unlike policy iteration, there is no explicit policy, thus the algorithm only acts on the value space.
  - Intermediate value functions v_k may not correspond to any policy

Value Iteration

- Another way of understanding: turn the Bellman optimal equation into an update rule

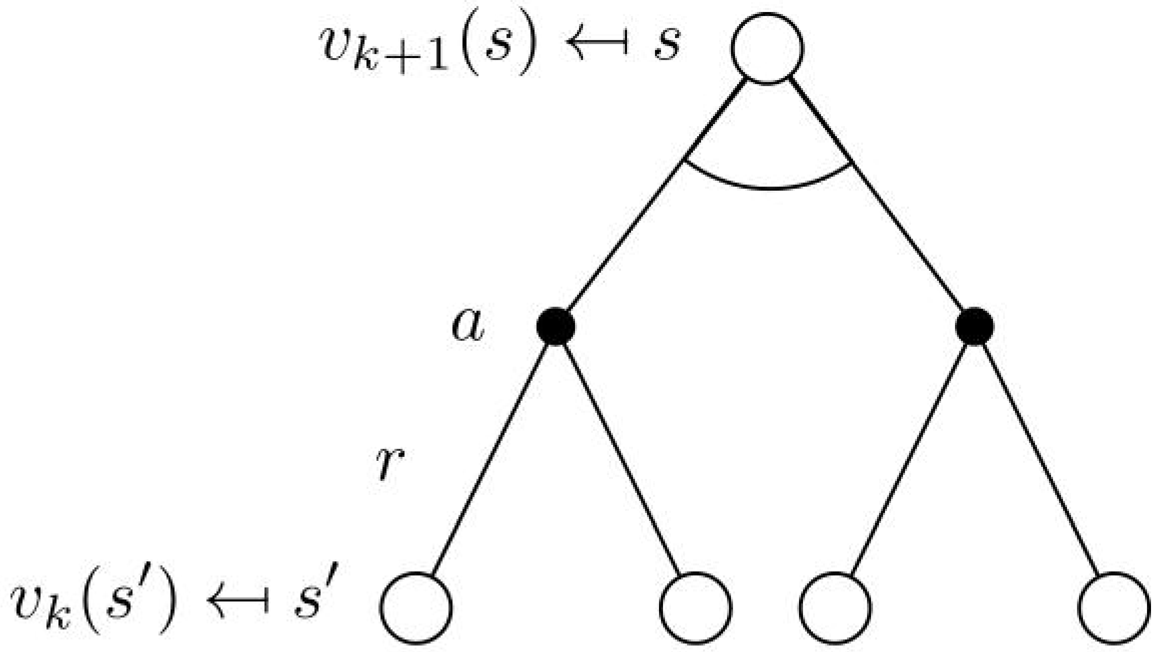
$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big] \end{aligned}$$

$$\Downarrow$$

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

Value Iteration

- Bellman optimality backup (synchronous)



Value Iteration

Value iteration

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

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

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

Value Iteration

- Recursive decomposition :
- If we know the solution to subproblems $v_*(s')$
- Then solution $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \longleftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right]$$

Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Asynchronous DP

Asynchronous DP

- Synchronous DP backs up all states in parallel
- Asynchronous DP backs up states individually, in any order
- Can significantly reduce computation
- Guaranteed to converge if all states continue to be selected

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function

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

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

- In-place value iteration only stores one copy of value function

$$v(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v(s')]$$

In-Place Dynamic Programming

- For the in-place algorithm:
  - It usually converges faster than the synchronous one since it use new data as soon as they are available
  - The order in which states are backed up has a significant influence on the rate of convergence

Prioritised Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma v(s')] - v(s) \right|$$

- Back up the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup

Real-Time Dynamic Programming

- Let an agent actually experience the MDP~
- Use agent's experience to guide the selection of states

$$v(S_t) \leftarrow \max_a \sum_{s',r} p(s',r \mid S_t,a)[r + \gamma v(s')]$$

- The latest value and policy information can also guide the agent's decision-making

Thank You!

π_*

$\{v_k\}$

$v_*(s')$

v_0

$|S|$

$k \to \infty$

$v_0 \to v_1 \to v_2 \to \ldots \to v_k \to \ldots \to v_*$

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

$v_\pi(s) = v_*(s)$

$\pi' \geq \pi$

$\pi = \pi_*$

$$v_{new}(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{old}(s')]$$

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

$$v(S_t) = \max_a \sum_{s',r} p(s', r \mid S_t, a)[r + \gamma v(S_t)]$$

$$\left| \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v(s')] - v(s) \right|$$