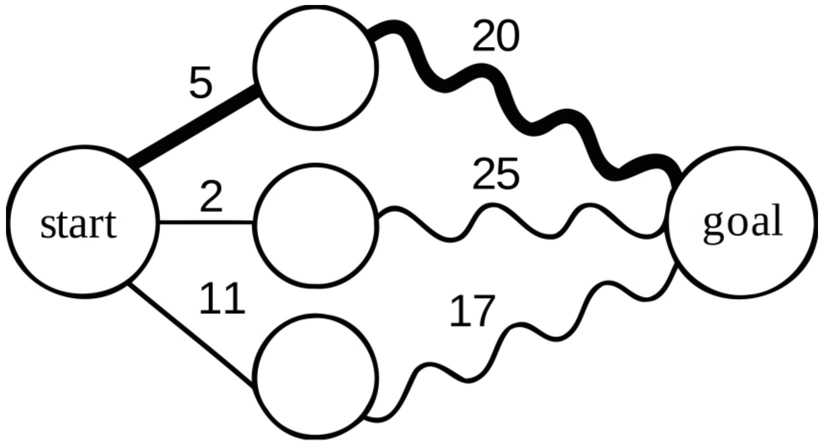
Dynamic Programming

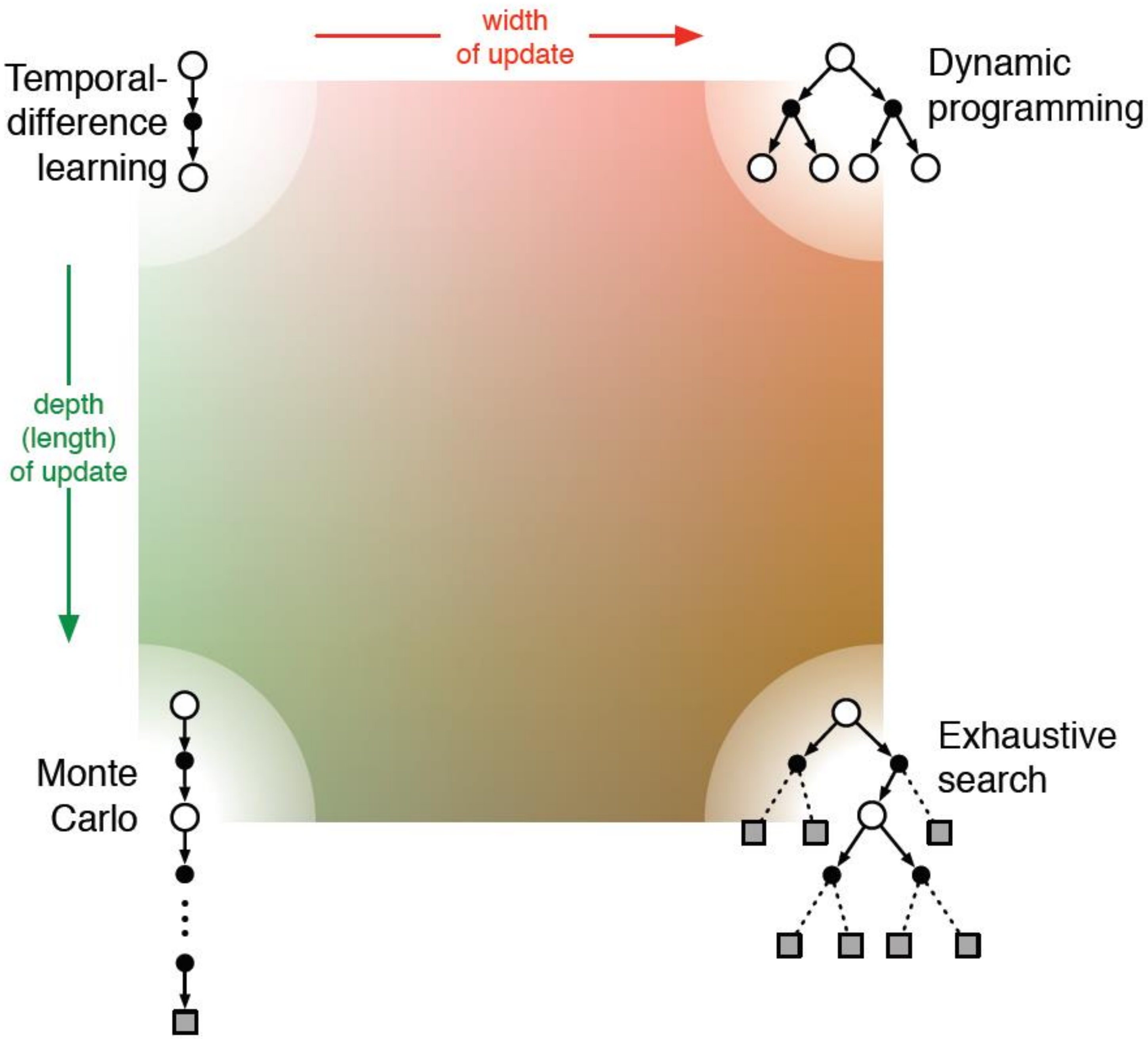
Prof. Kuan-Ting Lai

2020/4/10

Dynamic Programming

- Dynamic Programming is for problems with two properties:
  1. Optimal substructure
     - Optimal solution can be decomposed into subproblems
  2. Overlapping subproblems
     - Subproblems recur many times
     - Solutions can be cached and reused
- Examples:
  – Shortest Path, Hanoi Tower ,…….
  – Markov Decision Process





Sutton, Richard S.; Barto, Andrew G.. Reinforcement Learning (Adaptive Computation and Machine Learning series) (p. 189)

Dynamic Programming for MDP

- Bellman equation gives recursive decomposition
- Value function stores and reuses solutions
- Dynamic programming assumes full knowledge of the MDP
- Used for Model-based Planning

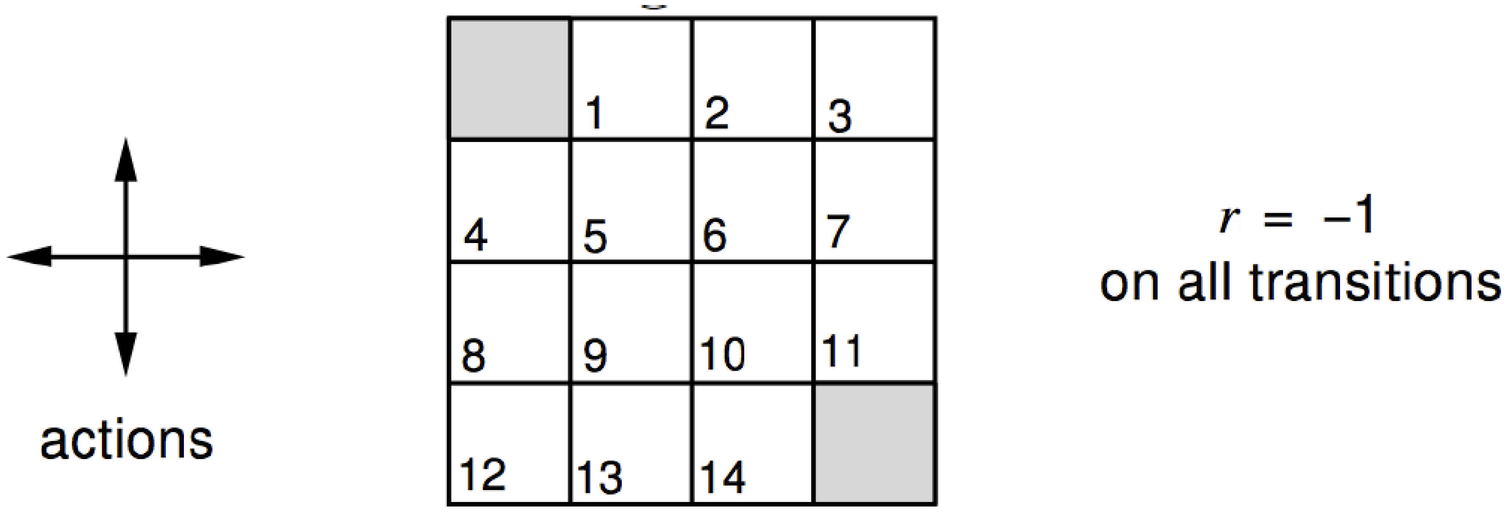
Policy Evaluation (Prediction)

- Calculate the state-action function V_π for an arbitrary policy π
- Can be solved iteratively

$$v_{k+1}(S) \leftarrow E_\pi[R_{t+1} + \gamma v_k(S_{t+1})]$$

Policy Evaluation in Small Grid World

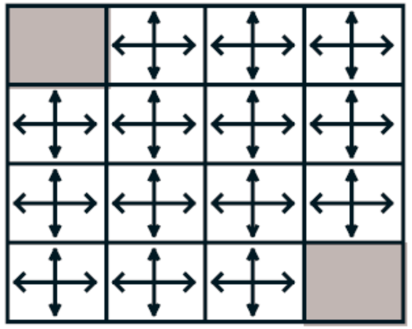
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached



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

$k = 0$

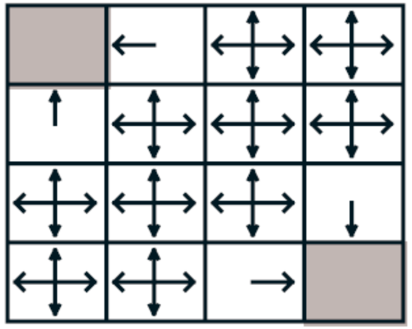
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

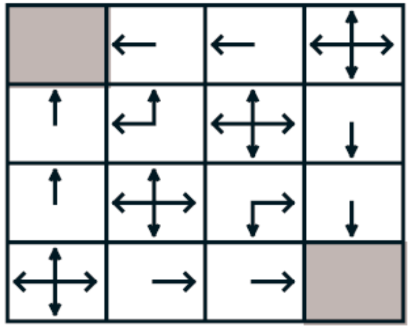
$k = 1$

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



v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 3$

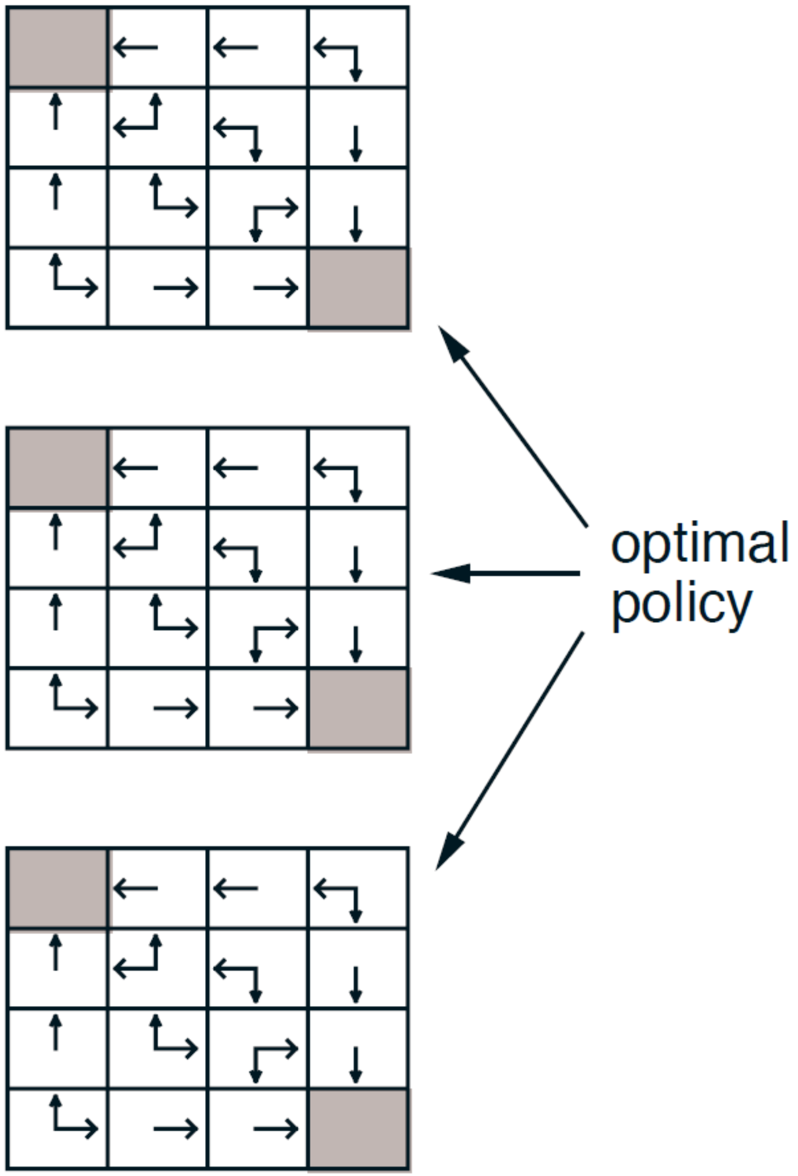
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



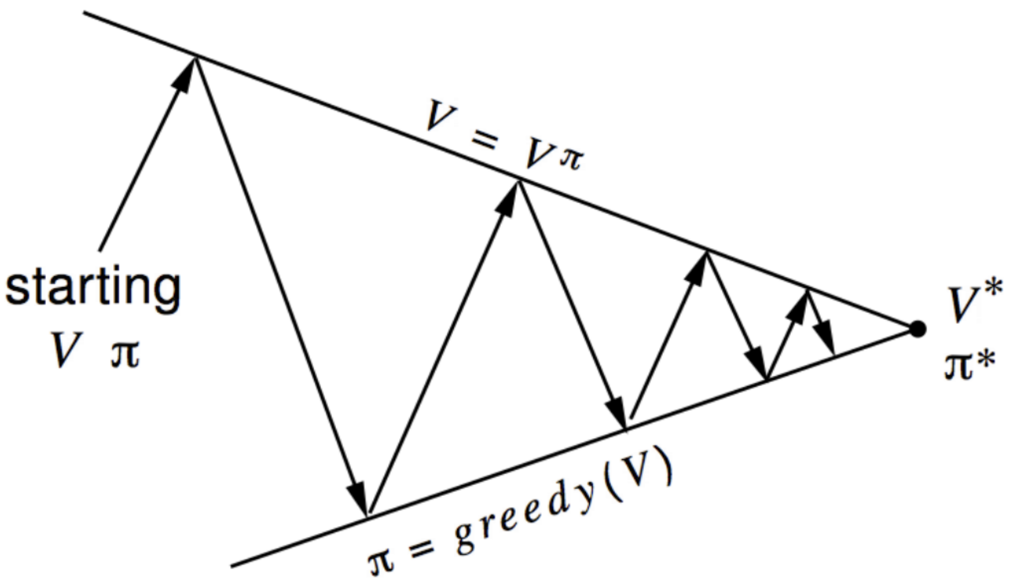
How to Improve a Policy

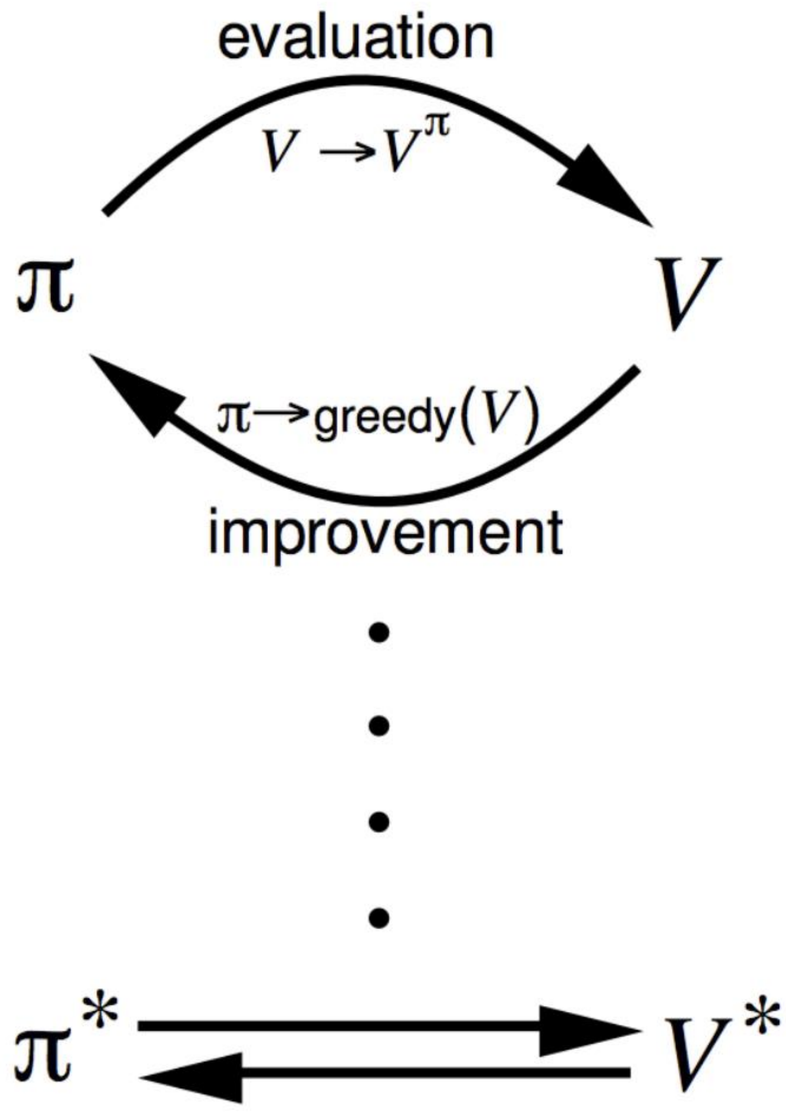
1. Evaluate the policy
   - $v_\pi(s) = E[R_{t+1} + R_{t+2} + \cdots | S_t = s]$
2. Improve the policy by acting greedily with respect to v
   - $\pi' = greedy(v_\pi)$

- This process of policy iteration always converges to π'

Policy Iteration

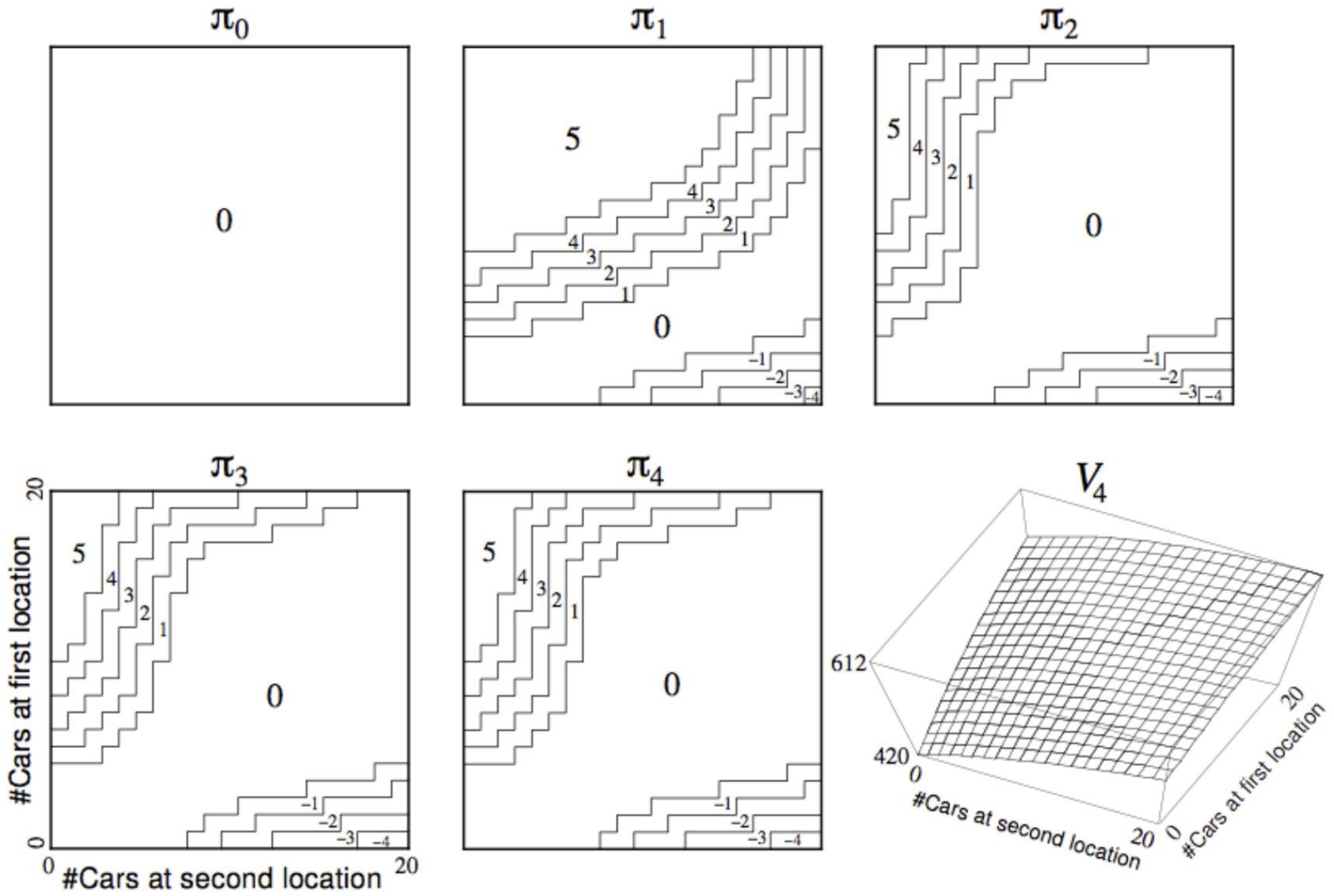
- Policy evaluation Estimate v_π
- Policy improvement Generate $\pi' \geq \pi$





Jack's Car Rental

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars between locations overnight
- Reward: \$10 for each car rented (must be available)
- Transitions: Cars returned and requested randomly
  - Poisson distribution, n returns/requests with prob $\frac{\lambda^n}{n!}e^{-\lambda}$
  - 1st location: average requests $= 3$, average returns $= 3$
  - 2nd location: average requests $= 4$, average returns $= 2$



π0
0
π1
5
4
3
2
1
0
−1
−2
−3
−4
π2
5
4
3
2
1
0
−1
−2
−3
−4
π3
20
0
#Cars at first location
5
4
3
2
1
0
−1
−2
−3
−4
0
#Cars at second location
20
π4
5
4
3
2
1
0
−1
−2
−3
−4
V4
612
420
0
#Cars at second location
20
0
#Cars at first location
20

Policy Improvement (1)

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

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

- 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 improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

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

Policy Improvement (2)

- If improvements stop,

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

- Then the Bellman optimality equation has been satisfied

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

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Modified Policy Iteration

- Do we need to iteratively evaluate until convergence of v_π?
- Can we simply stop after k iteration?
  - Example: Small grid world achieves optimal policy after k=3 iterations
- Update policy every iteration? => Value Iteration

Value Iteration

- Updating value function v only, don't calculate policy function π
- Policy is implicit built using v

Shortest Path Example

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

Policy Iteration vs. Value Iteration

- Policy iteration

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

- Value iteration

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

Reference

- David Silver, Lecture 3: Planning by Dynamic Programming (https://www.youtube.com/watch?v=Nd1-UUMVfz4&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&index=3)
- Chapter 4, Richard S. Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction,” 2nd edition, Nov. 2018