



# 强化学习原理及应用 Reinforcement Learning (RL): Theories & Applications

*DCS6289 Spring 2022*

Chao Yu (余超)

School of Computer Science and Engineering  
Sun Yat-Sen University

# Lecture 2: MDP and Dynamic Programming

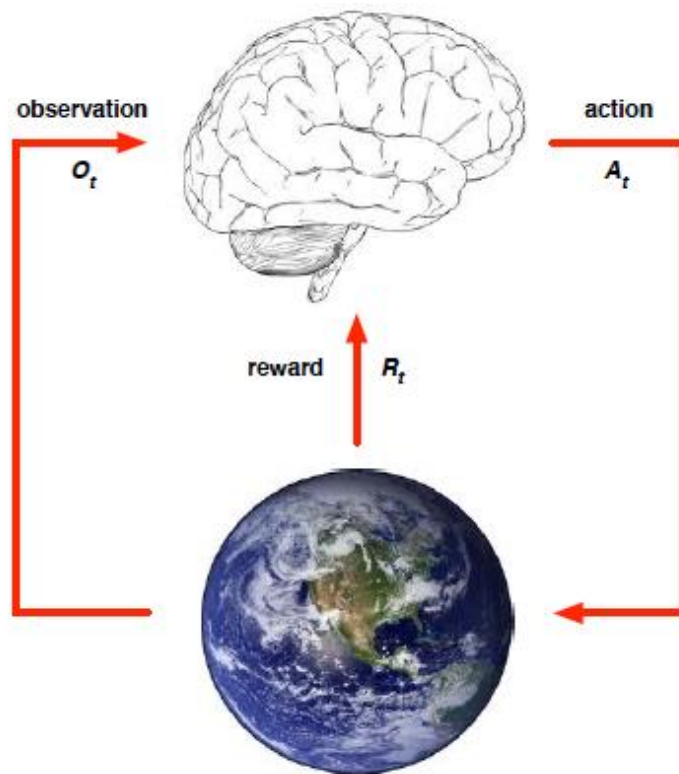
1<sup>st</sup> Mar. 2022

- Last lecture:
  - Course introduction
  - What's RL?
  - Broad applications of RL
  - Why RL?
- This lecture:
  - Basic components of RL: Reward, State, Policy, Model, Value function
  - The formal formulation of an RL problem as a Markov decision process
  - Making good decisions given a Markov decision process

# The RL Problem



“learn to *make good sequences of decisions through trail-and-errors*”



- ❑ At each step  $t$  the agent:
  - ❑ Executes action  $A_t$
  - ❑ Receives observation  $O_t$
  - ❑ Receives scalar reward  $R_t$
- ❑ The environment:
  - ❑ Receives action  $A_t$
  - ❑ Emits observation  $O_{t+1}$
  - ❑ Emits scalar reward  $R_{t+1}$

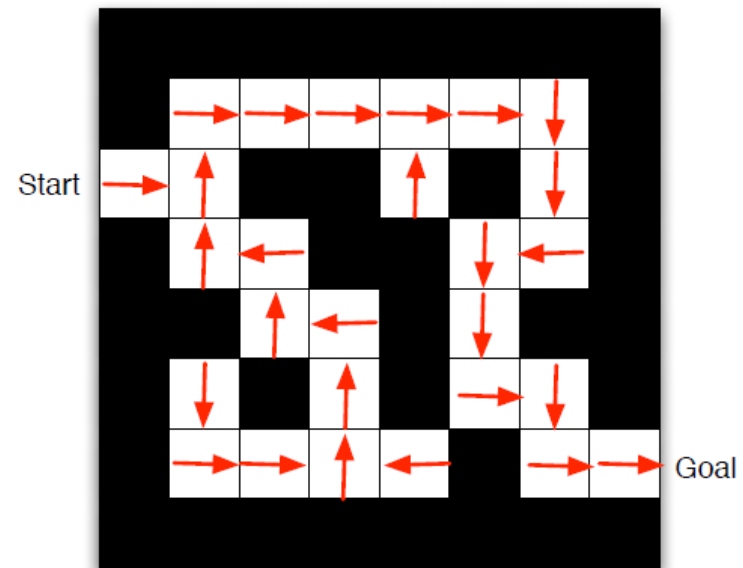
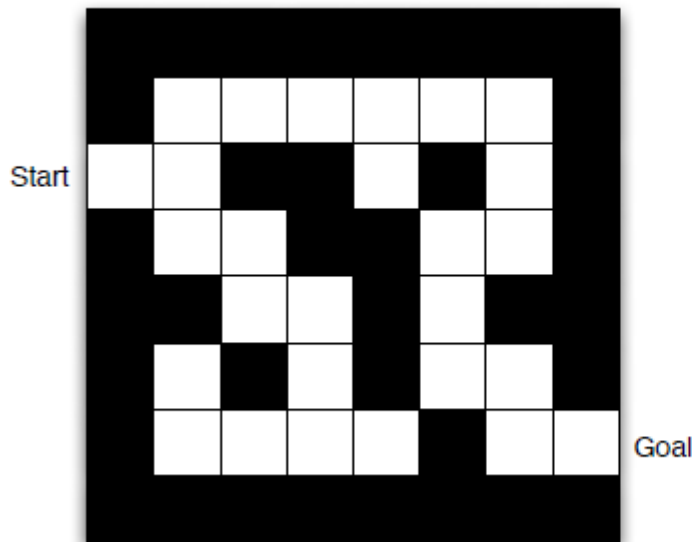
Goal: learn a policy (*i.e.*, a mapping from observations to actions) to maximise total future reward

# Elements of RL Problems - Policy

□ **Policy**: an agent's behaviour function, i.e., a mapping from state to action

□ Deterministic policy:  $a = \pi(s)$

□ Stochastic policy:  $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$




# Elements of RL Problems - Policy

□ **Policy**: an agent's behaviour function, i.e., a mapping from state to action

□ Deterministic policy:  $a = \pi(s)$

□ Stochastic policy:  $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

| $s_1$ | $s_2$ | $s_3$ | $s_4$  | $s_5$ | $s_6$ | $s_7$ |
|-------|-------|-------|--|-------|-------|-------|
|       |       |       |  |       |       |       |

- For the Mars rover example [ 7 discrete states (location of rover); 2 actions: Left or Right], how many deterministic policies are there?

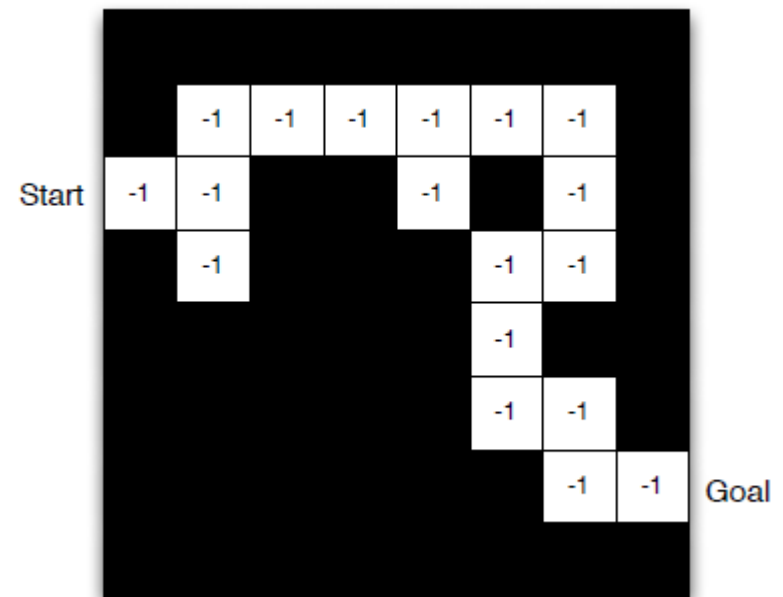
$$2 / 14 / 7^2 / 2^7$$

# Elements of RL Problems - Model

- ❑ **Model**: A model predicts what the environment will do next, i.e., agent's representation of the environment
- ❑  $P$  predicts the next state
- ❑  $R$  predicts the next (immediate) reward

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

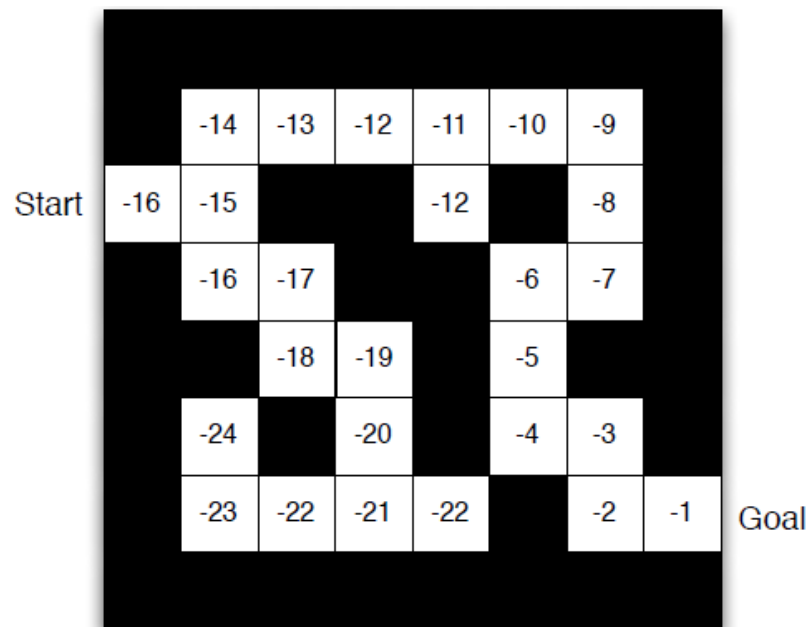


# Elements of RL Problems – Value Function

□ **Value functions:** how good is each state and/or action

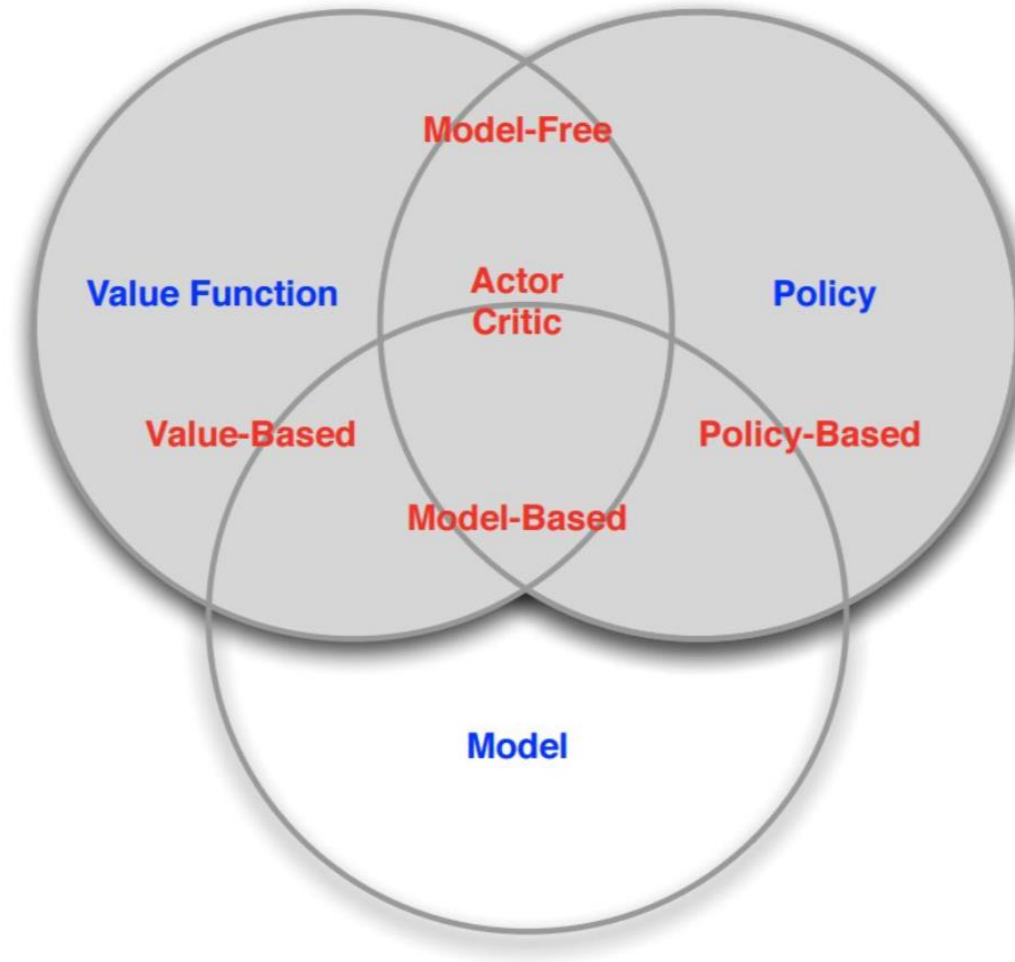
- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore used to select between actions

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$





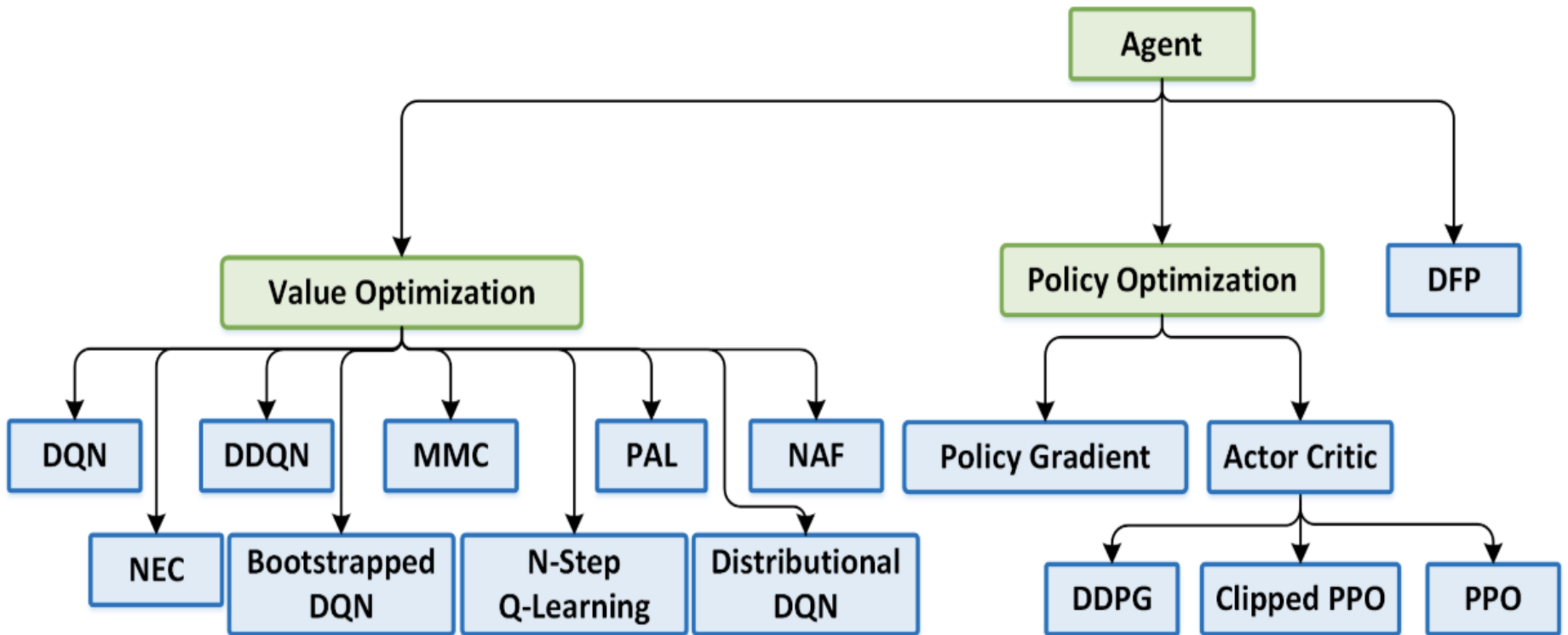
# Categorizing RL Algorithms



[from David Silver's course]



# Categorizing RL Algorithms



<https://blog.csdn.net/gsw404>

Deep RL

# Formulation of an RL Problem



- Markov Process
- Markov Reward Process (MRP)
- Markov Decision Process (MDP)
- Evaluation/Prediction and Improvement/Control in MDP

# Recall: Markov Property

- A **Markov state** contains all useful information from the history, i.e., future is independent of past given present

A state  $S_t$  is **Markov** if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

- **Markov Process** or Markov Chain


- Sequence of random states with Markov property
- $P$  is dynamics/transition model that specifies

$$p(s_{t+1} = s' \mid s_t = s)$$

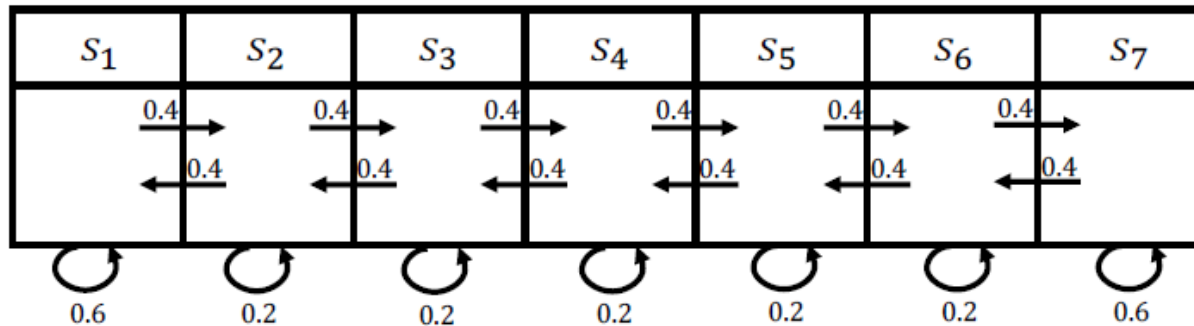
- no rewards, no actions
- $P$  can be expressed as a matrix

$$P = \begin{pmatrix} P(s_1|s_1) & P(s_2|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \cdots & P(s_N|s_N) \end{pmatrix}$$

# Markov Process

| $s_1$ | $s_2$ | $s_3$ | $s_4$  | $s_5$ | $s_6$ | $s_7$ |
|-------|-------|-------|--|-------|-------|-------|
|       |       |       |  |       |       |       |

The Mars rover problem



$$P = \begin{pmatrix} 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0.2 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0.2 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0.2 & 0.4 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

e.g., Sample episodes starting from  $s_4$

- $s_4, s_5, s_6, s_7, s_7, s_7, \dots$
- $s_4, s_4, s_5, s_4, s_5, s_6, \dots$
- $s_4, s_3, s_2, s_1, \dots$

# Markov Reward Process (MRP)

□ **Markov Reward Process** is a Markov Process with rewards

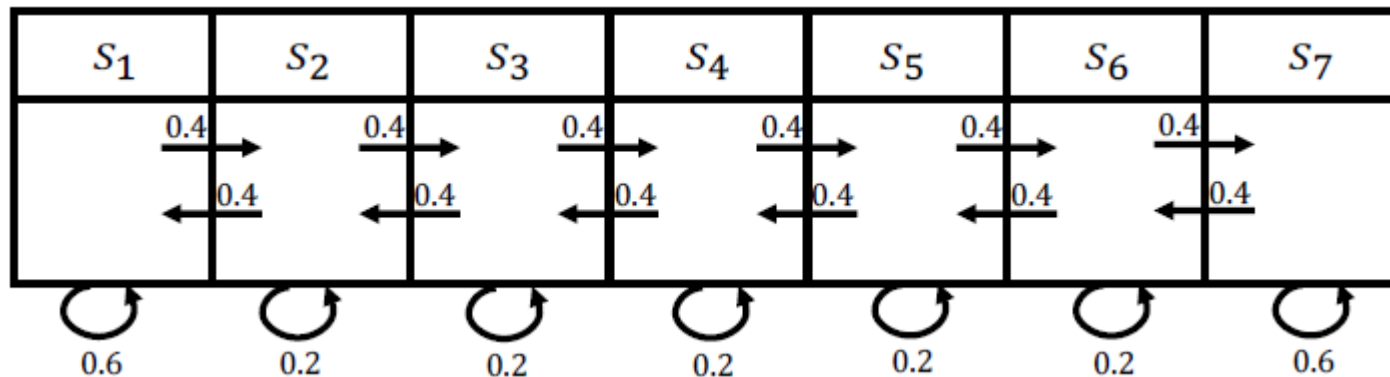
□  $P$  is dynamics/transition model that specifies

$$p(s_{t+1} = s' | s_t = s)$$

□  $R$  is a reward function  $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$

□ Discount factor  $\gamma \in [0, 1]$

□ No actions



Reward: +1 in  $s_1$ , +10 in  $s_7$ , 0 in all other states

# Return & Value Function

- Definition of **Horizon** ( $H$ )

- Number of time steps in each episode

- Can be infinite or finite

- Definition of **Return**

- Discounted sum of rewards from time step  $t$  to horizon  $H$

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1}$$

- Definition of **State Value Function**  $V(s)$

- Expected return from starting in state  $s$

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1} | s_t = s]$$

# Discount Factor

- ❑ Mathematically convenient
  - ❑ avoid infinite returns and values
- ❑ Model humans' behaviors
  - ❑  $\gamma = 0$ : only care about immediate reward
  - ❑  $\gamma = 1$ : future reward is with the same importance

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1}$$



# Computing the Value of MRP

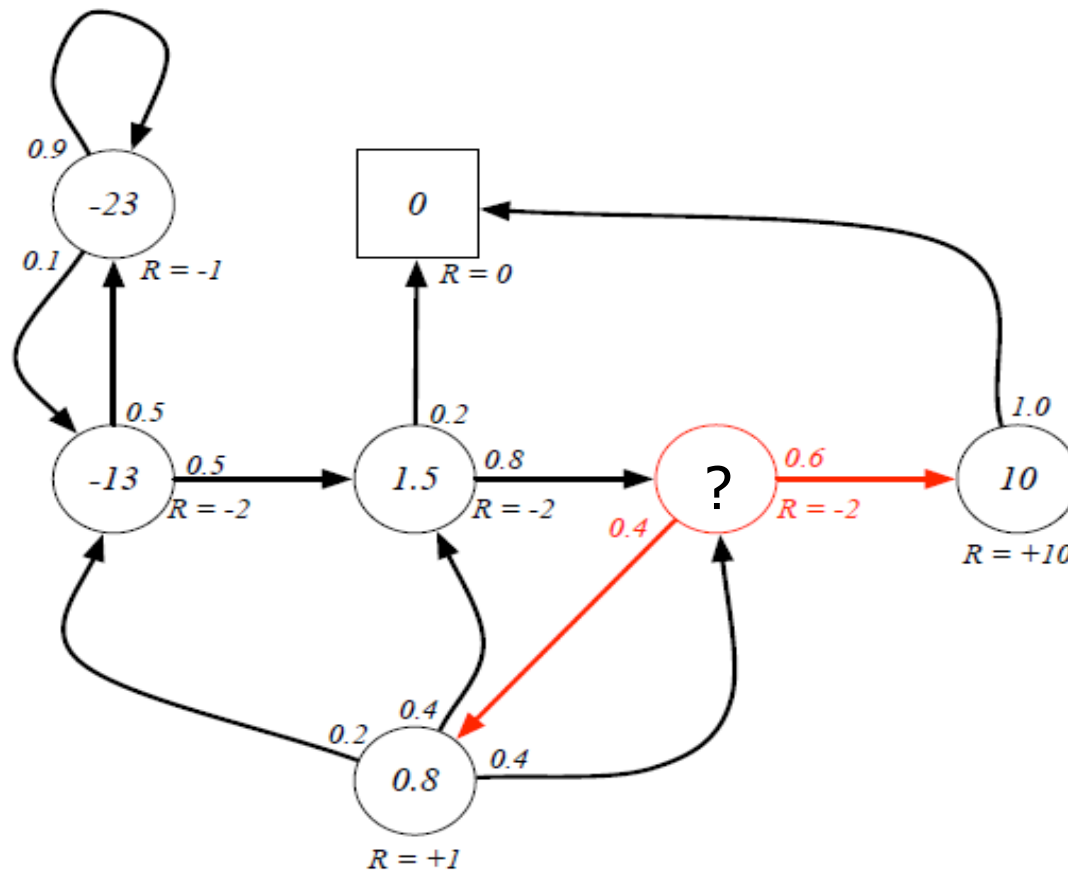
□ MRP value function satisfies the **Bellman Equation**

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{H-1} r_{t+H-1} | s_t = s]$$

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future rewards}}$$

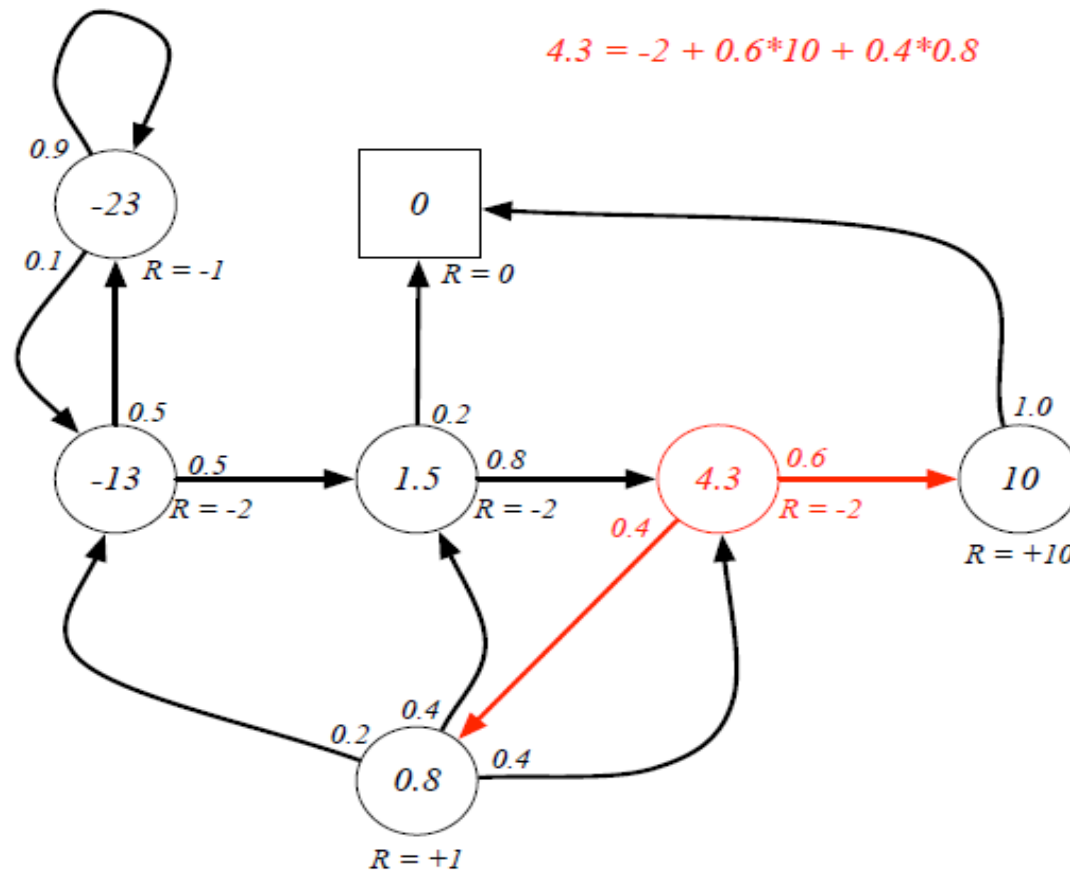
# Computing the Value of MRP

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future rewards}}$$



# Computing the Value of MRP

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future rewards}}$$



# Computing the Value of MRP

- For finite state MRP, we can express  $V(s)$  in a matrix form

$$\begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ \vdots \\ R(s_N) \end{pmatrix} + \gamma \begin{pmatrix} P(s_1|s_1) & \cdots & P(s_N|s_1) \\ P(s_1|s_2) & \cdots & P(s_N|s_2) \\ \vdots & \ddots & \vdots \\ P(s_1|s_N) & \cdots & P(s_N|s_N) \end{pmatrix} \begin{pmatrix} V(s_1) \\ \vdots \\ V(s_N) \end{pmatrix}$$

$$V = R + \gamma PV$$

$$V - \gamma PV = R$$

$$(I - \gamma P)V = R$$

$$V = (I - \gamma P)^{-1}R$$

- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

# Markov Decision Process (MDP)

- **Markov Decision Process** is Markov Reward Process with actions
  - $P$  is dynamics/transition model for each action that specifies
$$P(s_{t+1} = s' | s_t = s, a_t = a)$$
  - $R$  is a reward function  $R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t = s, a_t = a]$
  - Discount factor  $\gamma \in [0, 1]$
  - MDP is a tuple:  $(S, A, P, R, \gamma)$

$$P(s' | s, a_1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad P(s' | s, a_2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The transition matrix for the Mars rover problem ( $a_1$  means moving left, and  $a_2$  means moving right)

- Policy specifies what action to take in each state
  - Can be deterministic or stochastic
  - Usually is a distribution over actions given states

$$\pi(a|s) = P(a_t = a | s_t = s)$$

- Given an MDP and a policy, then

$$R^\pi(s) = \sum_{a \in A} \pi(a|s) R(s, a)$$

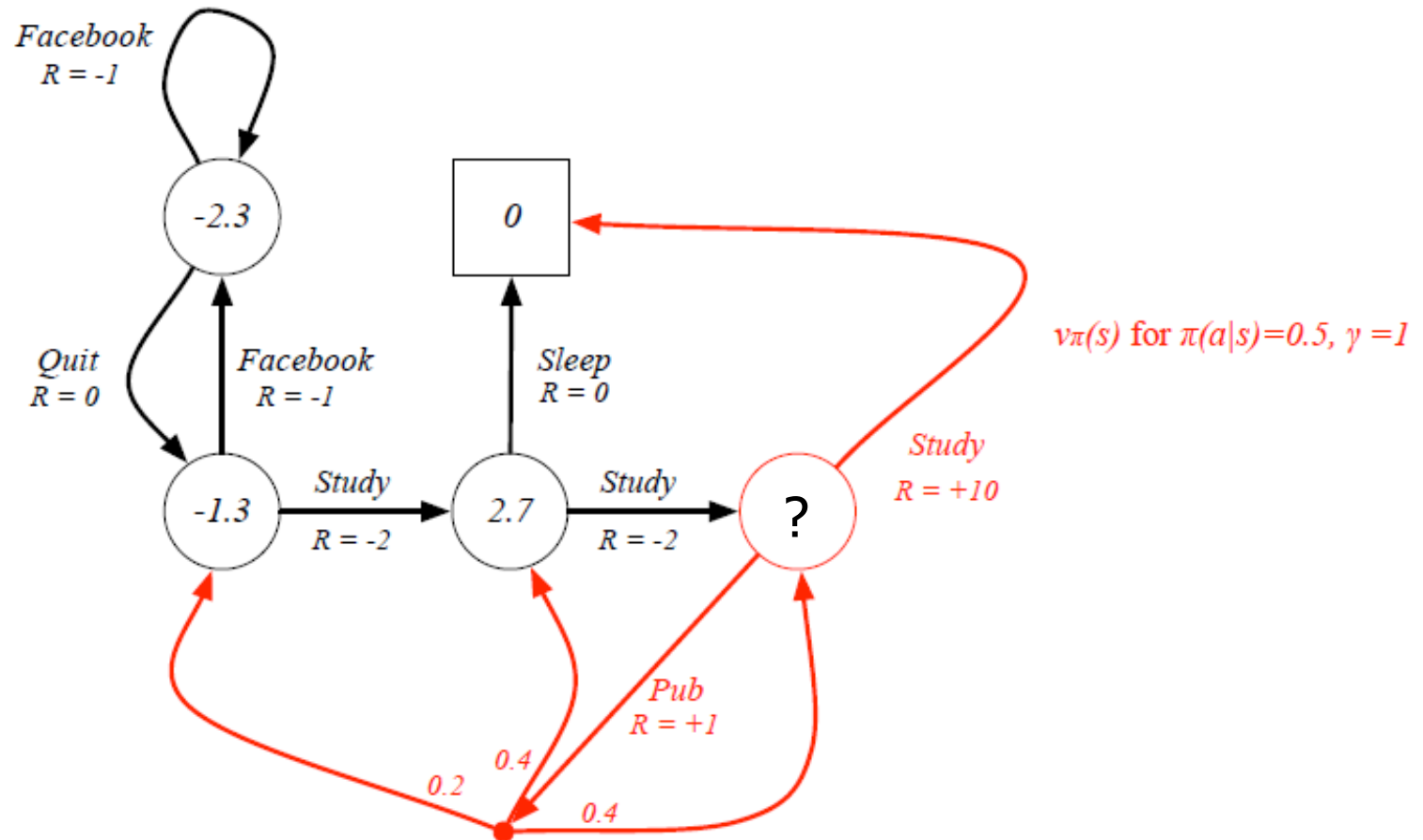
$$P^\pi(s'|s) = \sum_{a \in A} \pi(a|s) P(s'|s, a)$$

- **State-Action Value Q** for a policy

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^\pi(s')$$

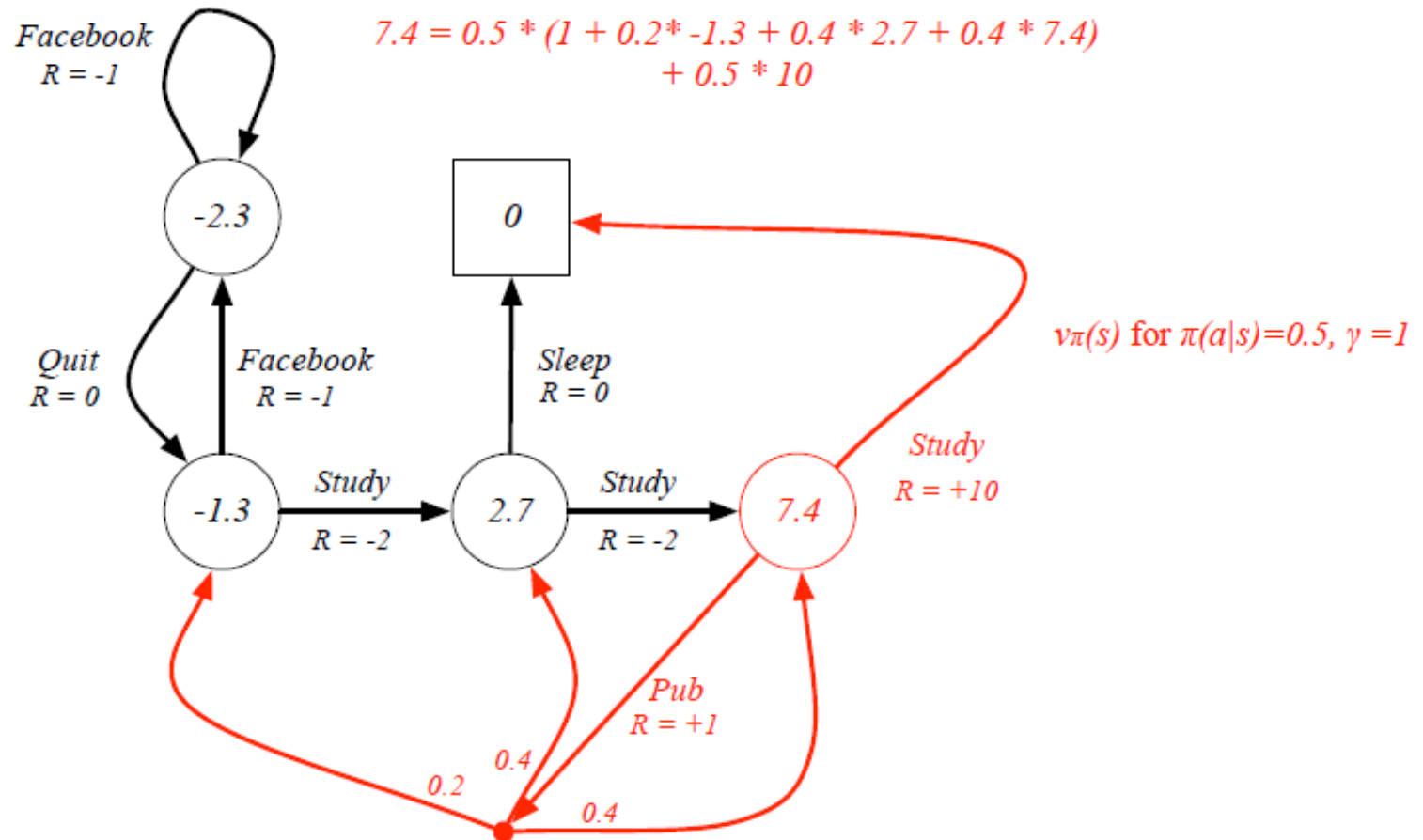
# Computing the Value of MDP

- The value of a state is the value of expected next state plus the reward expected along the way



# Computing the Value of MDP

- The value of a state is the value of expected next state plus the reward expected along the way





- ❑ Compute the **optimal policy**

$$\pi^*(s) = \arg \max_{\pi} V^{\pi}(s)$$

- ❑ There exists a unique optimal value function, which specifies the best possible performance in the MDP
- ❑ Optimal policy for an MDP in an infinite horizon problem is deterministic, but not necessarily unique
- ❑ One option is searching to compute best policy
- ❑ Number of deterministic policies is  $|A|^{|S|}$
- ❑ Policy iteration is generally more efficient than enumeration

# What's Dynamic Programming (DP)?

- ❑ *Dynamic*: sequential or temporal component to the problem
- ❑ *Programming*: optimizing a “program”, i.e. a policy
- ❑ A method for solving complex problems by
  - ❑ breaking them down into subproblems
  - ❑ combining solutions of subproblems
- ❑ DP is a general solution method for problems with two properties:
  - ❑ Optimal solution can be decomposed into subproblems
  - ❑ Subproblems recur many times and solutions can be cached and reused
- ❑ MDP satisfy both properties
  - ❑ Bellman equation gives recursive decomposition
  - ❑ Value function stores and reuses solutions

$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future rewards}}$$

# Solving MDP using DP



- ❑ DP assumes full knowledge of the MDP for planning
- ❑ DP is an iterative solution method to MDP
  - ❑ Policy Iteration (PI)
  - ❑ Value Iteration (VI)
- ❑ PI iterates between the following processes
  - ❑ Policy **evaluation** (prediction): Estimate/predict the expected rewards from following a given policy
  - ❑ Policy **improvement** (control): find a better policy
- ❑ VI iterates between the estimation of value functions and policy optimization, without explicit policy

# Value Function for MDP



- The state-value function  $v(s)$  of an MDP is the expected return starting from state  $s$ , and following policy  $\pi$

$$v^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

$$\text{where } G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

- The action-value function  $q(s,a)$  is the expected return starting from state  $s$ , taking action  $a$ , and then following policy  $\pi$

$$q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, A_t = a]$$

- We have the relation

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s, a)$$

# Bellman Expectation Equation



- The state-value function can be decomposed into immediate reward plus discounted value of the successor state,

$$v^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma v^{\pi}(s_{t+1}) | s_t = s]$$

- The action-value function can similarly be decomposed

$$q^{\pi}(s, a) = E_{\pi}[R_{t+1} + \gamma q^{\pi}(s_{t+1}, A_{t+1}) | s_t = s, A_t = a]$$

# Bellman Expectation Equation for V and Q



$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s) q^{\pi}(s, a)$$

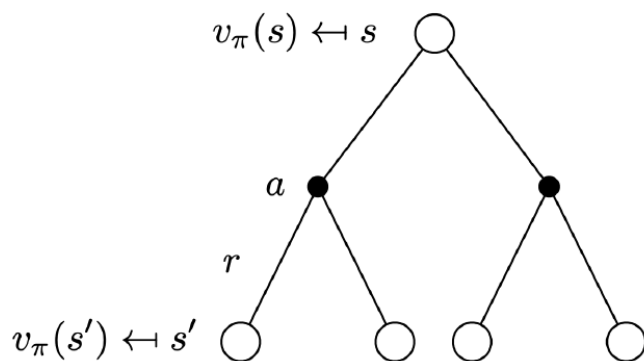
$$q^{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s, a) v^{\pi}(s')$$

Thus

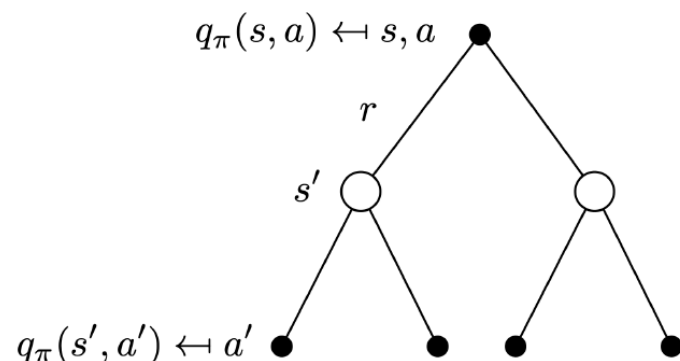
$$v^{\pi}(s) = \sum_{a \in A} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^{\pi}(s'))$$

$$q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') q^{\pi}(s', a')$$

# Backup Diagram for V and Q



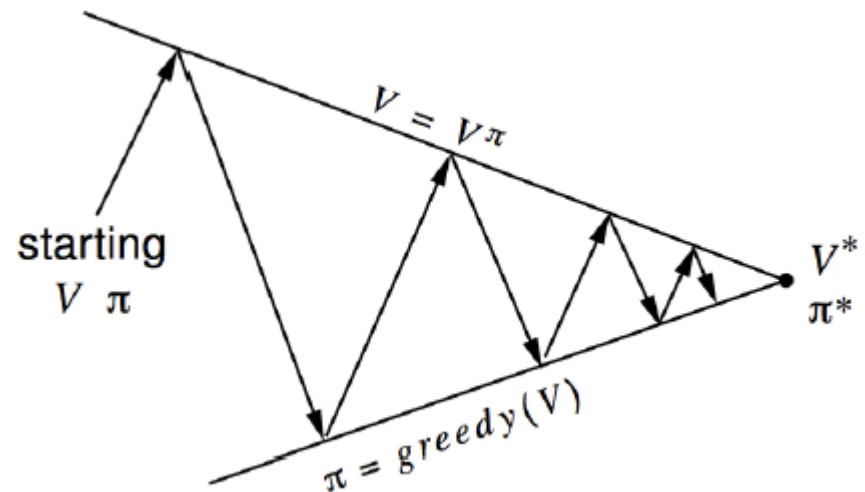
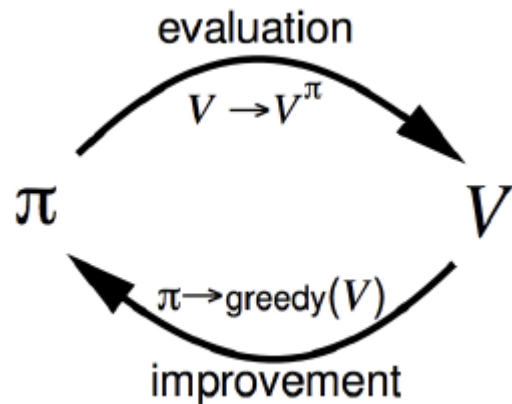
$$v^\pi(s) = \sum_{a \in A} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^\pi(s'))$$



$$q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') q^\pi(s', a')$$

# Policy Iteration (PI)

- Set  $i = 0$
- Initialize  $\pi_0(s)$  randomly for all states  $s$
- While  $i == 0$  or  $\|\pi_i - \pi_{i-1}\|_1 > 0$  (L1-norm, measures if the policy changed for any state):
  - $V^{\pi_i} \leftarrow$  MDP  $V$  function policy **evaluation** of  $\pi_i$
  - $\pi_{i+1} \leftarrow$  Policy **improvement**
  - $i = i + 1$





# Policy Evaluation

- ❑ Objective: evaluate a given policy  $\pi$  for a MDP
- ❑ Output: the value function under policy  $\pi$
- ❑ Solution: iteration on Bellman expectation backup
- ❑ Algorithm: Synchronous backup

- 1 At each iteration  $t+1$   
update  $v_{t+1}(s)$  from  $v_t(s')$  for all states  $s \in \mathcal{S}$  where  $s'$  is a successor state of  $s$

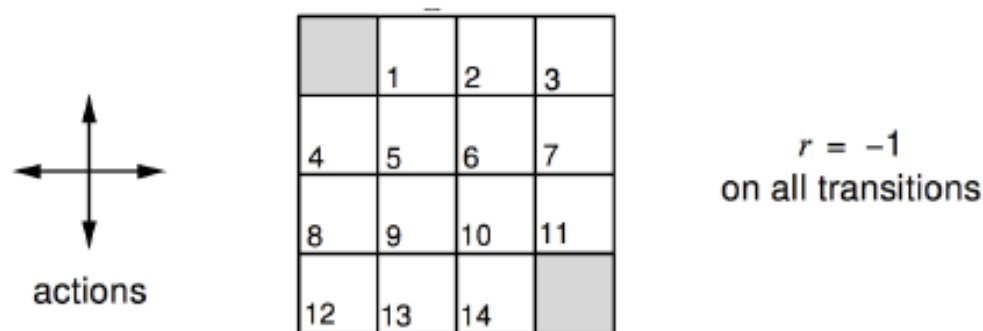
$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_t(s'))$$

- ❑ Convergence:  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v^\pi$

# Policy Evaluation

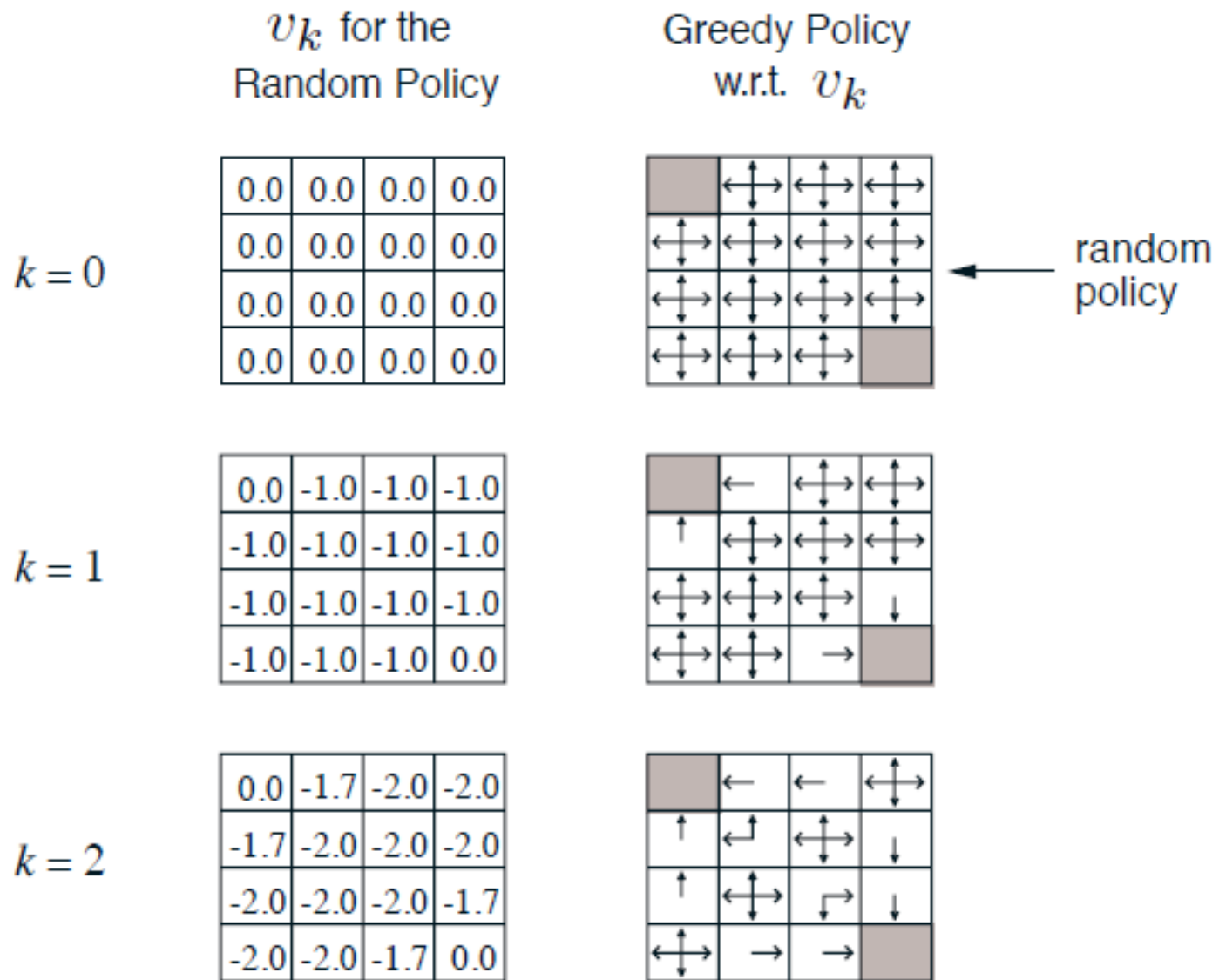
□ Example 4.1 in the Sutton RL textbook.

- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy



$$v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v_t(s'))$$

# Policy Evaluation



# Policy Evaluation



$k = 3$

|      |      |      |      |
|------|------|------|------|
| 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  |

|   |   |   |   |
|---|---|---|---|
|   | ← | ← | ↙ |
| ↑ | ↖ | ↙ | ↓ |
| ↑ | ↗ | ↘ | ↓ |
| ↖ | → | → |   |

optimal policy

# Policy Improvement

□ Consider a deterministic policy  $a = \pi(s)$

□ We improve the policy through

$$\pi'(s) = \arg \max_a q^\pi(s, a)$$

□ 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  $v_{\pi'}(s) \geq v_\pi(s)$

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

# PI as Bellman Operations

□ **Bellman backup operator**  $B^\pi$  for a particular policy is defined as

$$B^\pi V(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s) V(s)$$

$$V^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s) V^\pi(s')$$

□ Policy evaluation amounts to computing the **fixed point** of  $B^\pi$

□ To do policy evaluation, repeatedly apply operator until  $V$  stops changing

$$V^\pi = B^\pi B^\pi \dots B^\pi V$$

□ To do policy improvement,

$$\pi_{k+1}(s) = \arg \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi_k}(s')$$

# Bellman Optimality Equation



- The optimal value function  $v^*$  are reached by the **Bellman optimality equations**

$$v^*(s) = \max_a q^*(s, a)$$

$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$$

thus

$$v^*(s) = \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v^*(s')$$

$$q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a'} q^*(s', a')$$

- ❑ Policy iteration computes optimal value and policy
- ❑ Value iteration is another technique
  - ❑ Maintain optimal value of starting in a state  $s$  if having a finite number of steps  $k$  left in the episode
  - ❑ Iterate to consider longer and longer episodes
- ❑ In other word, we assume we know the solution to subproblems and then find the optimal solution by one-step lookahead

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) v(s') \right)$$



# Value Iteration (VI)

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 |

$V_7$

# Value Iteration (VI)

- ① Objective: find the optimal policy  $\pi$
- ② Solution: iteration on the Bellman optimality backup
- ③ Value Iteration algorithm:

- ① initialize  $k = 1$  and  $v_0(s) = 0$  for all states  $s$
- ② For  $k = 1 : H$ 
  - ① for each state  $s$

$$q_{k+1}(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v_k(s')$$

$$v_{k+1}(s) = \max_a q_{k+1}(s, a)$$

- ②  $k \leftarrow k + 1$
- ③ To retrieve the optimal policy after the value iteration:

$$\pi(s) = \arg \max_a R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v_{k+1}(s')$$

# Value vs Policy Iteration



- ❑ Policy iteration includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges
- ❑ Value iteration includes: **finding optimal value function** + **one policy extraction**. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
- ❑ Finding optimal value function can also be seen as a combination of policy improvement (due to max) and truncated policy valuation (the reassignment of  $v(s)$  after just one sweep of all states regardless of convergence).

# Will DP Converge?



□ Bellman backup operator is a contraction if discount factor when  $\gamma < 1$

□ Contraction Operator

- Let  $O$  be an operator, and  $|x|$  denote (any) norm of  $x$
- If  $|OV - OV'| \leq |V - V'|$ , then  $O$  is a contraction operator

□ If apply it to two different value functions, distance between value functions shrinks after applying Bellman equation to each

- Let  $\|V - V'\| = \max_s |V(s) - V'(s)|$  be the infinity norm

$$\begin{aligned}
 \|BV_k - BV_j\| &= \left\| \max_a \left( R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_k(s') \right) - \max_{a'} \left( R(s, a') + \gamma \sum_{s' \in S} P(s'|s, a') V_j(s') \right) \right\| \\
 &\leq \max_a \left\| \left( R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V_k(s') - R(s, a) - \gamma \sum_{s' \in S} P(s'|s, a) V_j(s') \right) \right\| \\
 &= \max_a \left\| \gamma \sum_{s' \in S} P(s'|s, a) (V_k(s') - V_j(s')) \right\| \\
 &\leq \max_a \left\| \gamma \sum_{s' \in S} P(s'|s, a) \|V_k - V_j\| \right\| \\
 &= \max_a \left\| \gamma \|V_k - V_j\| \sum_{s' \in S} P(s'|s, a) \right\| \\
 &= \gamma \|V_k - V_j\|
 \end{aligned}$$

# Summary of DP methods

| Problem    | Bellman Equation   | Algorithm                   |
|------------|--|-----------------------------|
| Prediction | Bellman Expectation Equation                             | Iterative Policy Evaluation |
| Control    | Bellman Expectation Equation + Greedy Policy Improvement | Policy Iteration            |
| Control    | Bellman Optimality Equation                              | Value Iteration             |

- Algorithms are based on state-value function  $v_{\pi}(s)$  or  $v_*(s)$
- Complexity  $O(mn^2)$  per iteration, for  $m$  actions and  $n$  states
- Could also apply to action-value function  $q_{\pi}(s, a)$  or  $q_*(s, a)$
- Complexity  $O(m^2n^2)$  per iteration

# Asynchronous DP

- ❑ DP methods described so far used *synchronous* backups
  - ❑ i.e. all states are backed up in parallel
- ❑ *Asynchronous* DP backs up states individually, in any order
  - ❑ For each selected state, apply the appropriate backup
  - ❑ Can significantly reduce computation
  - ❑ Guaranteed to converge if all states continue to be selected
- ❑ Three simple ideas for asynchronous DP:
  - ❑ **In-place DP**: only stores one copy of value function
  - ❑ **Prioritised sweeping**: backup the state with the largest remaining Bellman error
$$\left| \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s') \right) - v(s) \right|$$
  - ❑ **Real-time DP**: Use agent's experience to guide the selection of states

# Full-Width Backups

- ❑ DP uses **full-width backups**
  - ❑ For each backup (sync or async), every successor state and action is considered
  - ❑ DP is effective for medium-sized problems (millions of states), but suffers Bellman's curse of dimensionality for large DP, as number of states grows exponentially with number of state variables
- ❑ Sample backups use sample rewards and sample transitions, instead of reward function and transition dynamics
  - ❑ **Model-free**: no advance knowledge of MDP required
  - ❑ Breaks the curse of dimensionality through sampling
  - ❑ Cost of backup is constant, independent of  $n = |\mathcal{S}|$



- ❑ Define MP, MRP, MDP, Bellman operator, contraction, model
- ❑ Know how to compute value function, optimal policy
- ❑ Be able to implement Value Iteration and Policy Iteration
- ❑ Give pros and cons of different policy evaluation approaches
- ❑ Be able to prove contraction properties