

Markov Decision Process

I-Chen Wu

- Sutton, R.S. and Barto, A.G., Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998. (Bible for RL)
 - <http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html>
 - Chapters 3-4
- David Silver, Online Course for Deep Reinforcement Learning.
 - <http://www.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html>
 - Chapters 2-3



Outline

- Introduction
- Markov Property
- Markov Process
- Markov Reward Process (MRP)
- Markov Decision Process (MDP)
- Partially Observable Markov Decision Process (POMDP)

The purpose of this chapter:

- Introduce all kinds of Markov processes

Introduction

- Markov decision processes formally describe an environment for reinforcement learning
 - where the environment is fully observable.
 - i.e. The current state completely characterizes the process
 - E.g., 2048.
- Almost all RL problems can be formalized as MDPs, e.g.
 - Optimal control primarily deals with continuous MDPs
 - Partially observable problems can be converted into MDPs
 - Bandits are MDPs with one state



Markov Property

- Markov Property:

- “The future is independent of the past given the present”
- Definition: A state S_t is Markov if and only if
$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

- Comments:

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

- But, what if the history does matter?

- Simply let S_t carry all information of history, $H_t = (S_1, \dots, S_{t-1})$.
 - ▶ E.g., the castling rule for chess.
- Then, it satisfies Markov Property.



Markov Process

- A Markov process is a memoryless random process,
 - i.e. a sequence of random states S_1, S_2, \dots with the Markov property.

Definition:

- A Markov Process (or Markov Chain) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
 - \mathcal{S} is a (finite) set of states
 - \mathcal{P} is a state transition probability matrix (part of the environment),
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

State Transition Matrix

- For a Markov state s and successor state s' , the **state transition probability** is defined by

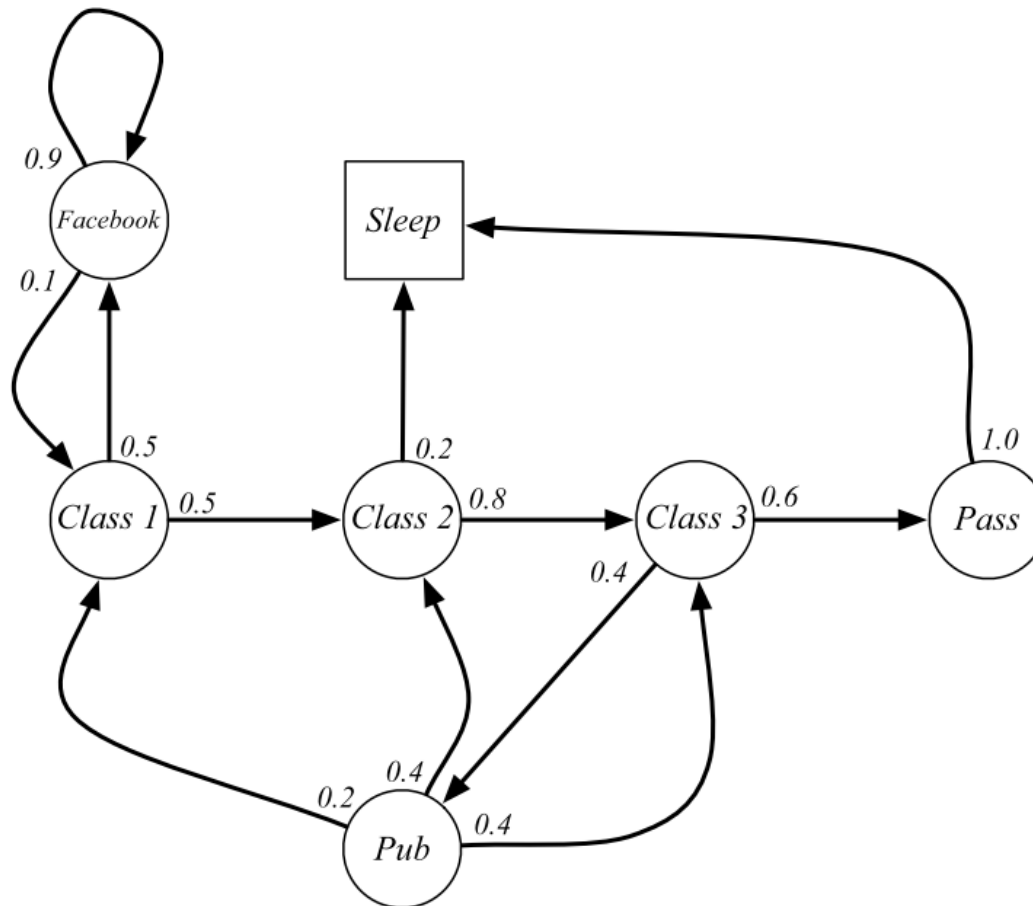
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

- State transition matrix \mathcal{P} : (assume n states)

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \dots & & \dots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

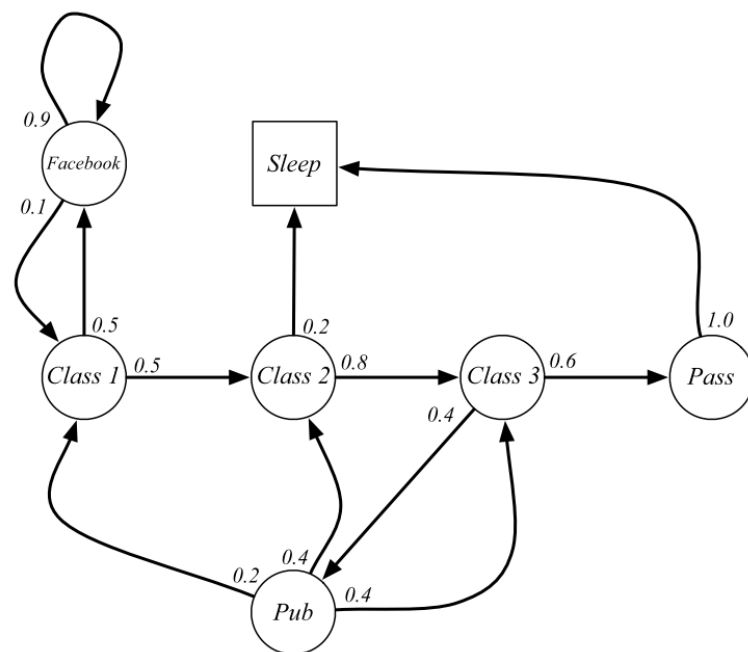
- Each row of matrix sums to 1.

Example: Student Markov Chain



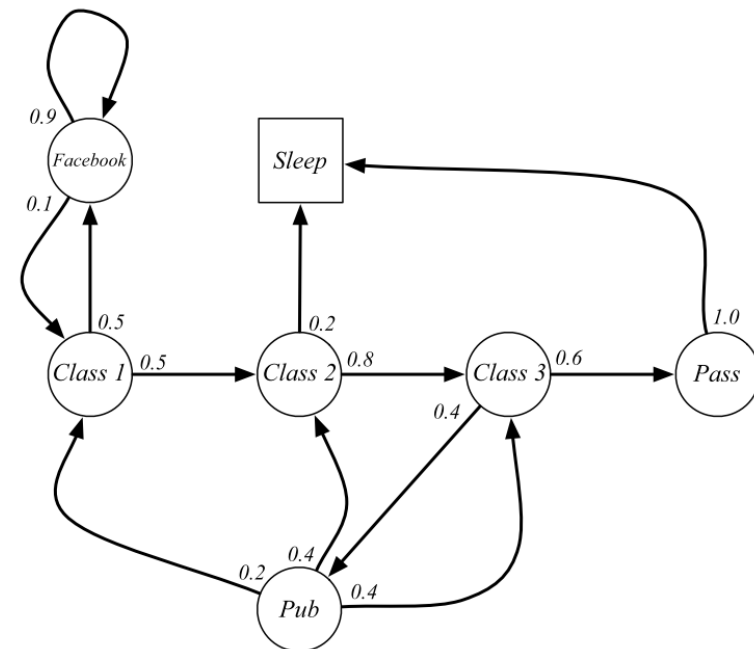
Example: Episodes

- Sample episodes for starting from $S_1 = C1$.
 - S_1, S_2, \dots, S_T
- Examples:
 - C1 C2 C3 Pass Sleep
 - C1 FB FB C1 C2 Sleep
 - C1 C2 C3 Pub C2 C3 Pass Sleep
 - C1 FB FB C1 C2 C3 Pub C1 FB FB
 - FB C1 C2 C3 Pub C2 Sleep



Example: Transition Matrix

$$\mathcal{P} = \begin{matrix} & \begin{matrix} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{matrix} \\ \begin{matrix} C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{matrix} & \left[\begin{array}{cccccc} & & 0.5 & & & 0.5 & \\ & & & 0.8 & & & 0.2 \\ & & & & 0.6 & 0.4 & \\ 0.2 & 0.4 & 0.4 & & & & 1.0 \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{array} \right] \end{matrix}$$



Markov Reward Process (MRP)

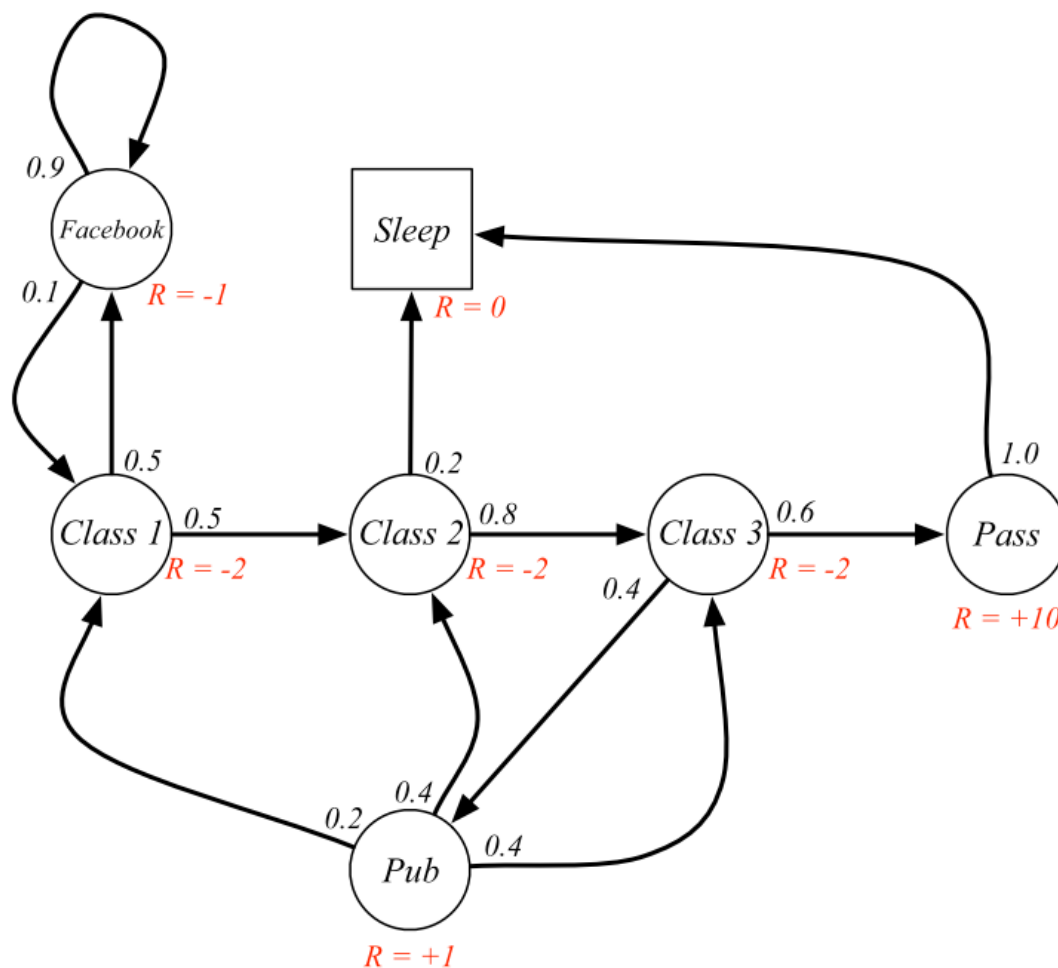
- A Markov reward process is a Markov chain with values.

Definition:

- A **Markov Reward Process** is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 - \mathcal{S} is a finite set of states
 - \mathcal{P} is a state transition probability matrix (part of the environment),
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$
 - \mathcal{R} is a reward function,
$$\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$$
 - γ is a discount factor $\gamma \in [0, 1]$.



Example: Student MRP



Return

Definition

- The return G_t is the total discounted reward from time-step t .

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Notes:

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R is diminishing
 - $\gamma^k R$, after $k + 1$ time-steps.
- This values immediate reward above delayed reward.
- Discount:
 - γ close to 0 leads to "myopic" evaluation
 - γ close to 1 leads to "far-sighted" evaluation



Value Function

- The value function $v(s)$ gives the long-term value of s
- Definition
 - The state value function $v(s)$ of an MRP is the expected return starting from state s
 - $v(s) = \mathbb{E}[G_t | S_t = s]$

Example: Student MRP Returns

● Sample returns for Student MRP:

- Starting from $S_1 = C1$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + \dots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep

$$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8} = -2.25$$

C1 FB FB C1 C2 Sleep

$$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} = -3.125$$

C1 C2 C3 Pub C2 C3 Pass Sleep

$$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots = -3.41$$

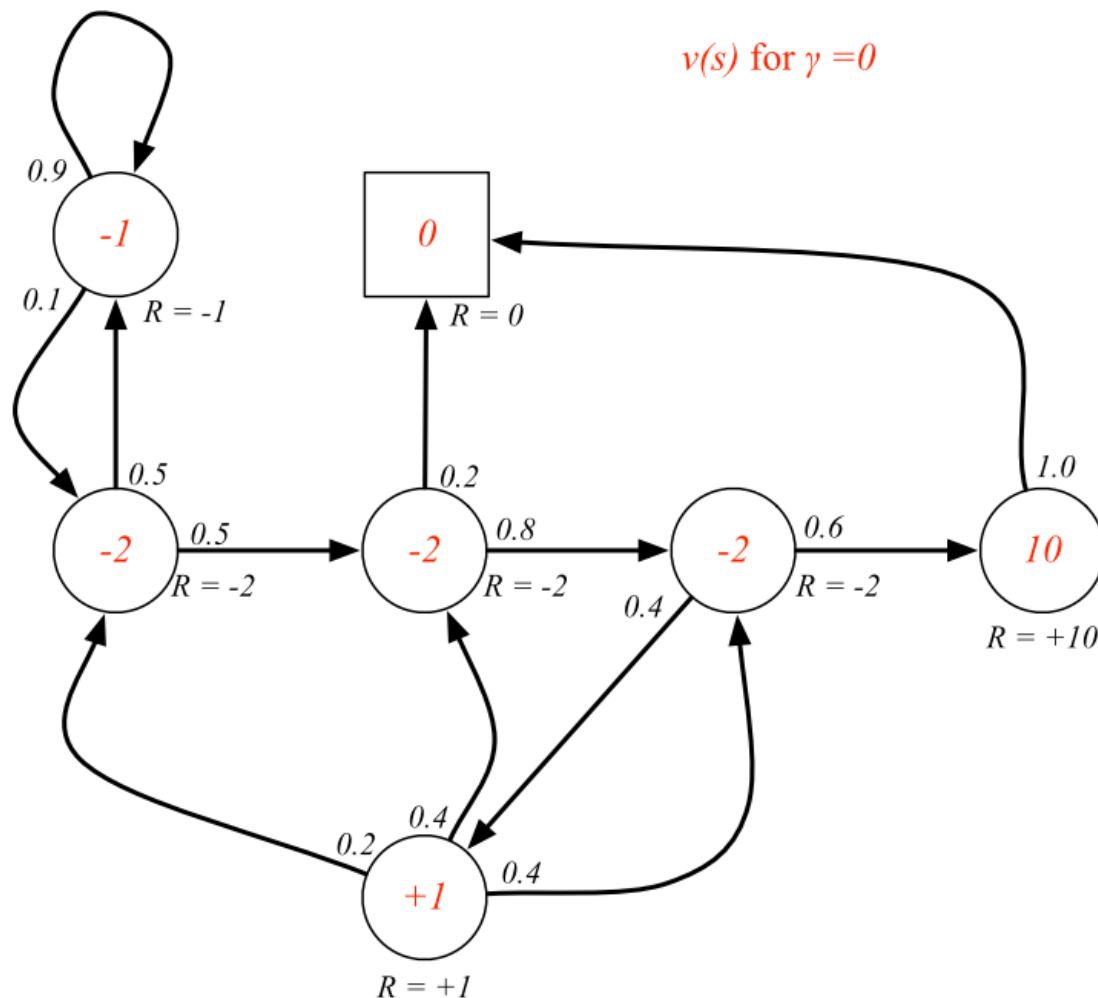
C1 FB FB C1 C2 C3 Pub C1 ...

$$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots = -3.20$$

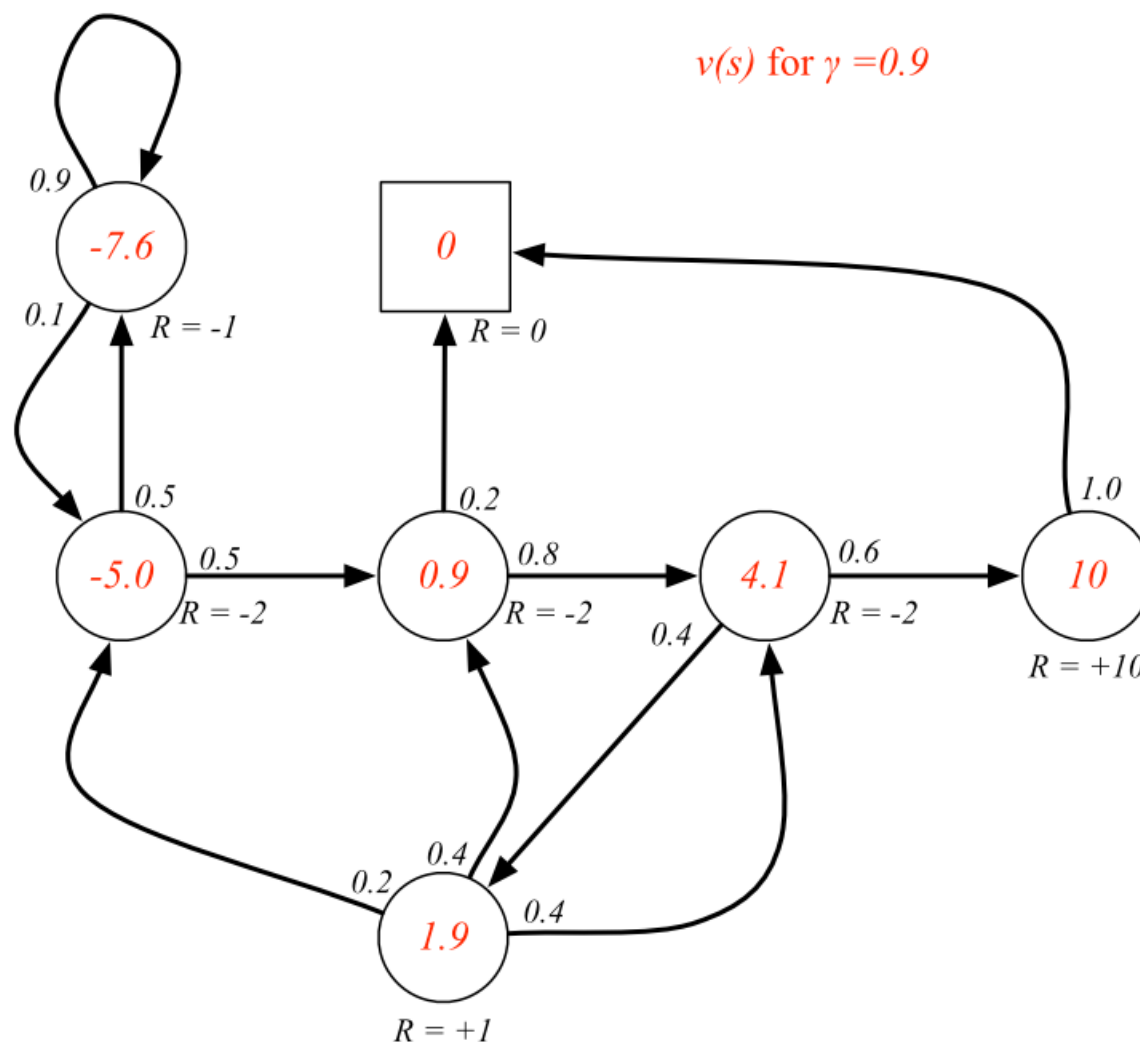
FB FB FB C1 C2 C3 Pub C2 Sleep



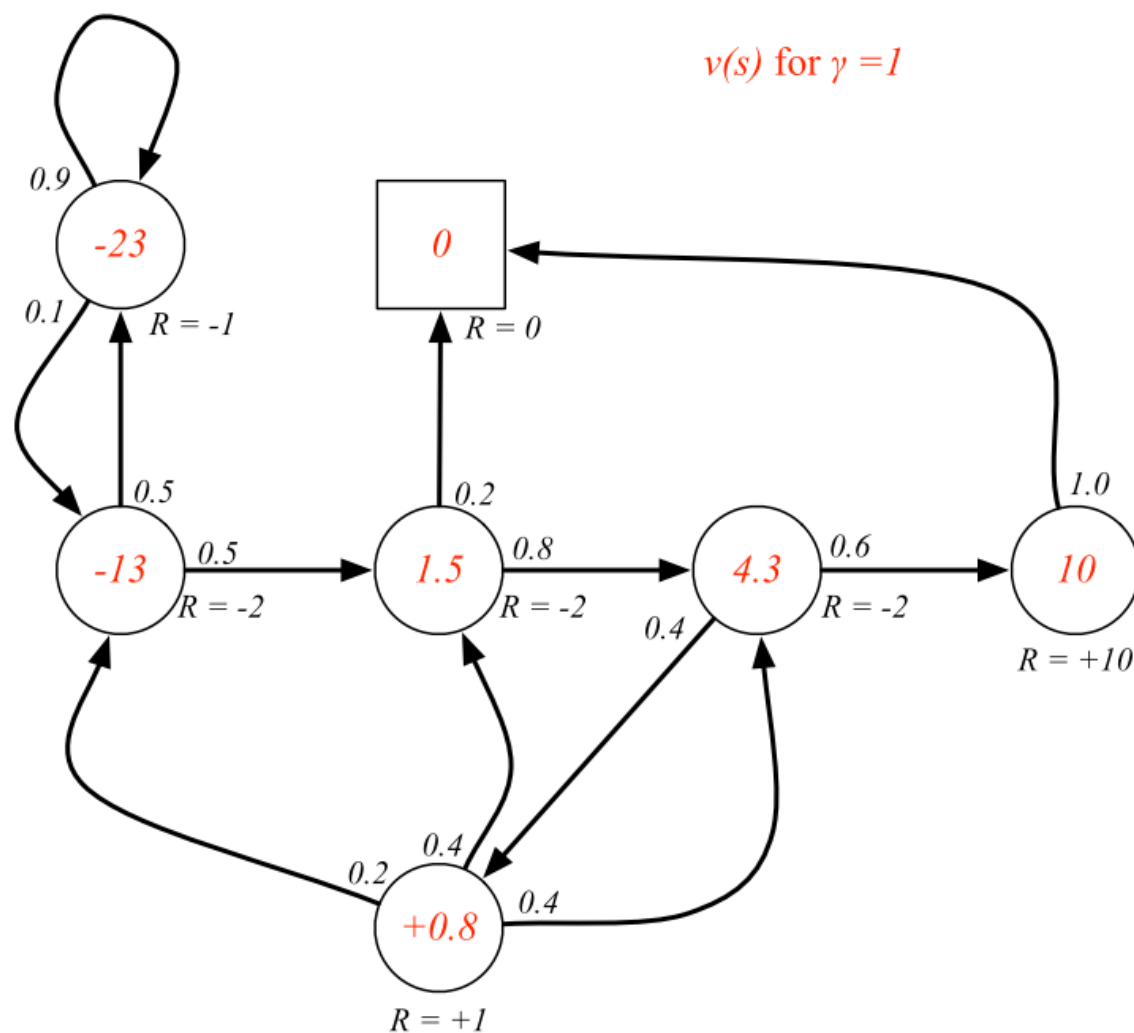
Example: State-Value Function for Student MRP



Example: State-Value Function for Student MRP



Example: State-Value Function for Student MRP



Bellman Equation for MRPs

- The value function can be decomposed into two parts:
 - immediate reward R_{t+1}
 - discounted value of successor state $\gamma v(S_{t+1})$
- $$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s] \end{aligned}$$
- For a transition (s, r, s') , we have

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$



Bellman Equation in Matrix Form

- The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P}v$$

- where v is a column vector with one entry per state.

$$\begin{bmatrix} v(1) \\ \dots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \dots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \dots & \dots & \dots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \dots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

$$v = \mathcal{R} + \gamma \mathcal{P}v$$

$$v = (1 - \gamma \mathcal{P})^{-1} \mathcal{R}$$

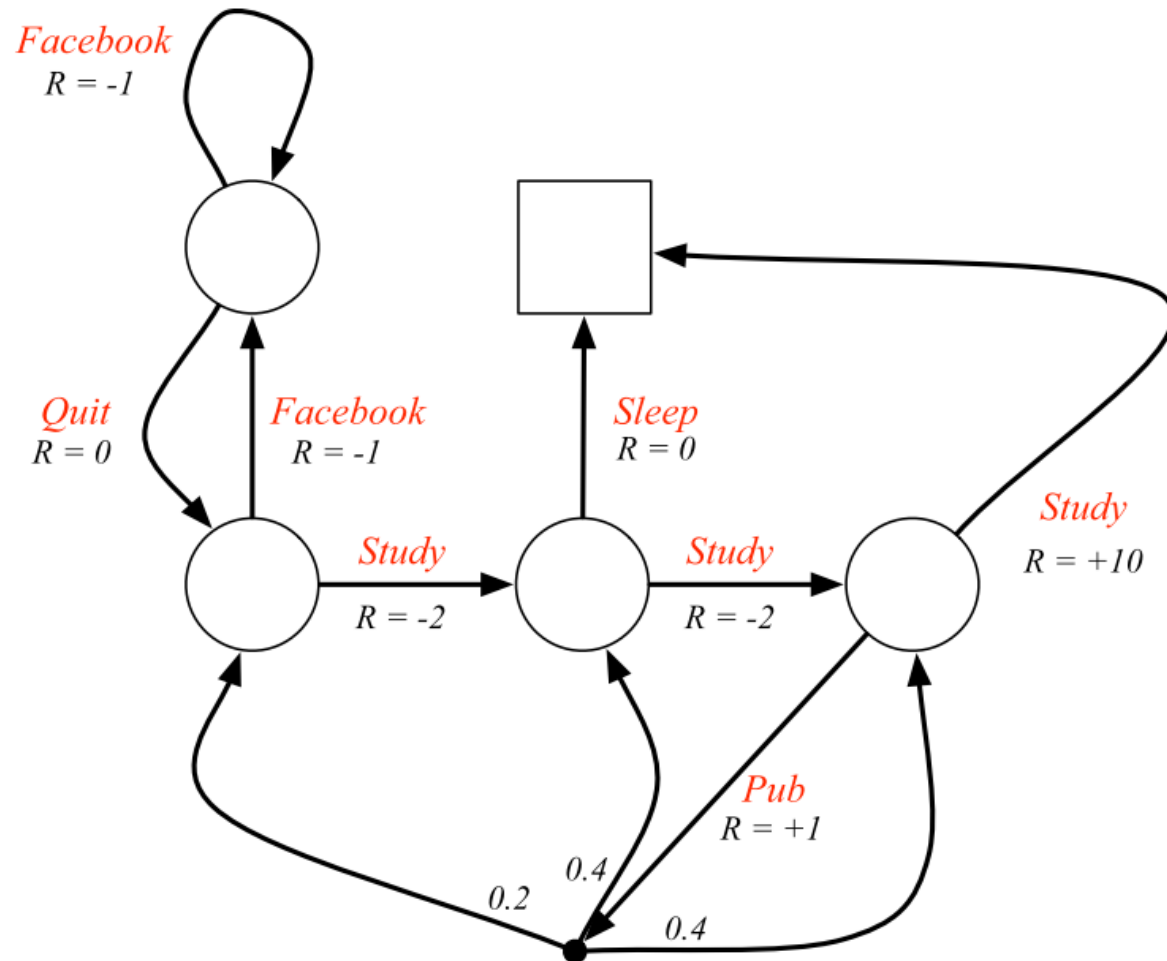
- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
 - Dynamic programming
 - Monte-Carlo evaluation
 - Temporal-Difference learning



Markov Decision Processes (MDP)

- A **Markov Decision Process** is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 - \mathcal{S} is a finite set of states
 - \mathcal{A} is a finite set of actions
 - \mathcal{P} is a state transition probability matrix (part of the environment),
$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
 - ▶ Let \mathcal{P}^a denote the matrix $\mathcal{P}_{\cdot\cdot}^a$.
 - \mathcal{R} is a reward function,
$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$
 - γ is a discount factor $\gamma \in [0, 1]$.

Example: Student MDP



Example: Recycling Robot

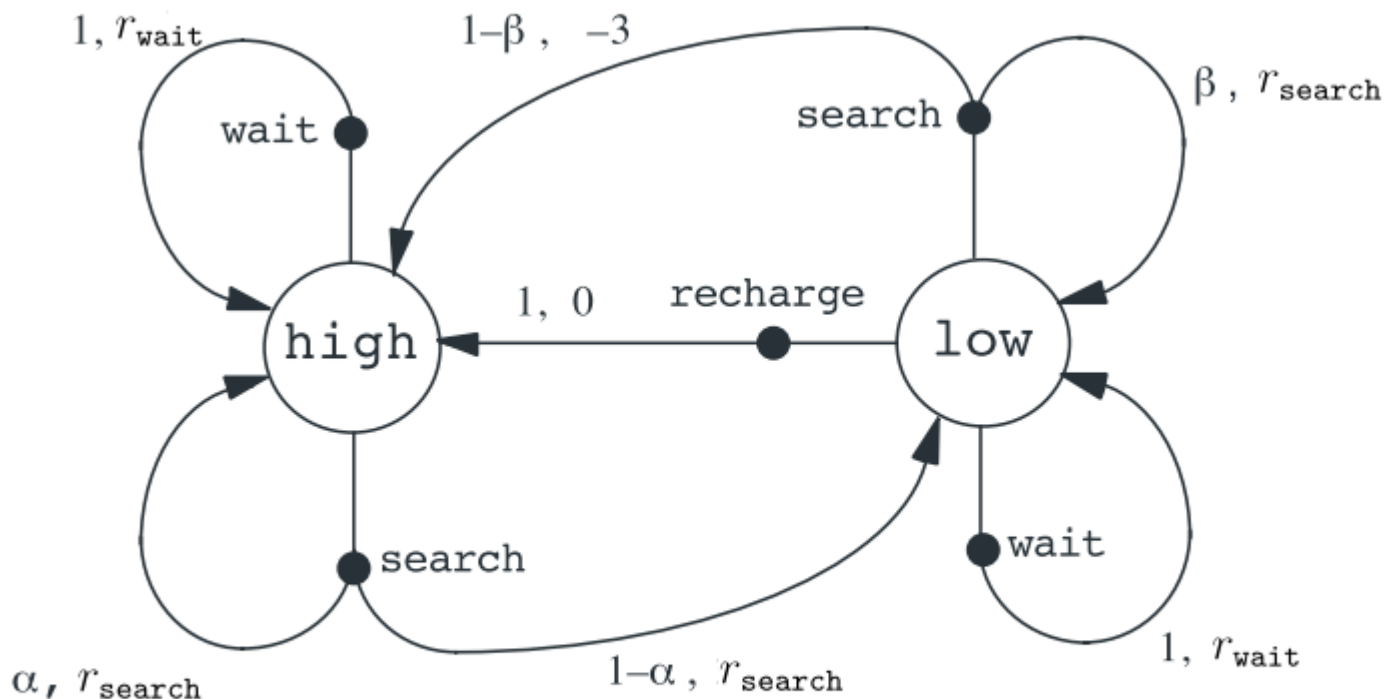


Figure 3.3: Transition graph for the recycling robot example.



Example: Recycling Robot

● Transition and Rewards:

s	s'	a	$p(s' s, a)$	$r(s, a, s')$
high	high	search	α	r_{search}
high	low	search	$1 - \alpha$	r_{search}
low	high	search	$1 - \beta$	-3
low	low	search	β	r_{search}
high	high	wait	1	r_{wait}
high	low	wait	0	r_{wait}
low	high	wait	0	r_{wait}
low	low	wait	1	r_{wait}
low	high	recharge	1	0
low	low	recharge	0	0.



Policies

- A policy is the agent's behavior
 - It is a map from state to action
 - A policy fully defines the behaviour of an agent
 - MDP policies depend on the current state (not the history)
 - ▶ i.e. Policies are stationary (time-independent),
 $A_t \sim \pi(\cdot | S_t), \forall t > 0$
- Policy types:
 - **Deterministic policy:** $a = \pi(s_i)$
 - **Stochastic policy:** $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$
 - ▶ Sometimes, written in $\pi(s, a)$.
- Examples:
 - In 2048: Up/down/left/right
 - In robotics: angle/force/...



Policy and MRP

- Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence S_1, S_2, \dots is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence $S_1, R_2, S_2, R_3, \dots$ becomes a Markov reward process (MRP) $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
 - \mathcal{P}^π is a state transition probability matrix (part of the environment),

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

- \mathcal{R}^π is a reward function,

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

- So, the property of MRP can be applied.

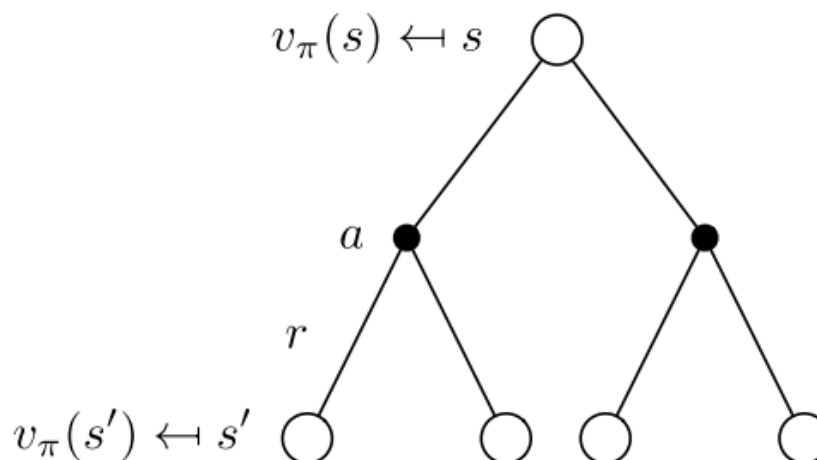
Value Function

- A value function is a prediction of future reward
 - Used to evaluate the goodness/badness of states
 - ▶ therefore to select between actions.
 - Return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$
- Types of value functions under policy π :
 - State value function: the expected return from s .
$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\ &= \mathbb{E}_{\pi}[G_t \mid S_t = s] \end{aligned}$$
 - Q-Value function: the expected return from s taking action a .
$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a]$$
- Examples:
 - In 2048, the expected score from a board S_t .

Bellman Expectation Equation for π

- State value function:

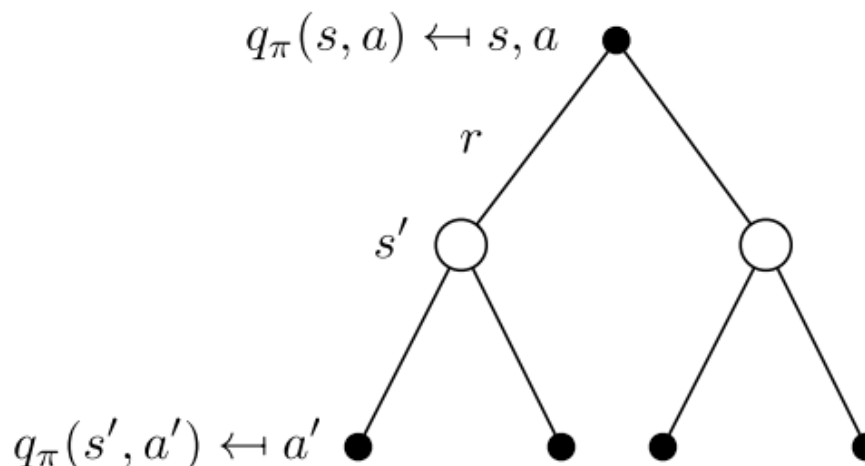
$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\pi}(s') \right)$$



Bellman Expectation Equation for π

- Q value

$$q_{\pi}(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_{\pi}(s', a')$$



Bellman Expectation Equation in Matrix

- The Bellman expectation equation can be expressed concisely using the induced MRP.
- So, it can be solved directly:

$$v_{\pi} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} v_{\pi}$$
$$v_{\pi} = (1 - \gamma \mathcal{P}^{\pi})^{-1} \mathcal{R}^{\pi}$$

Optimal Value Function

- The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

- The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

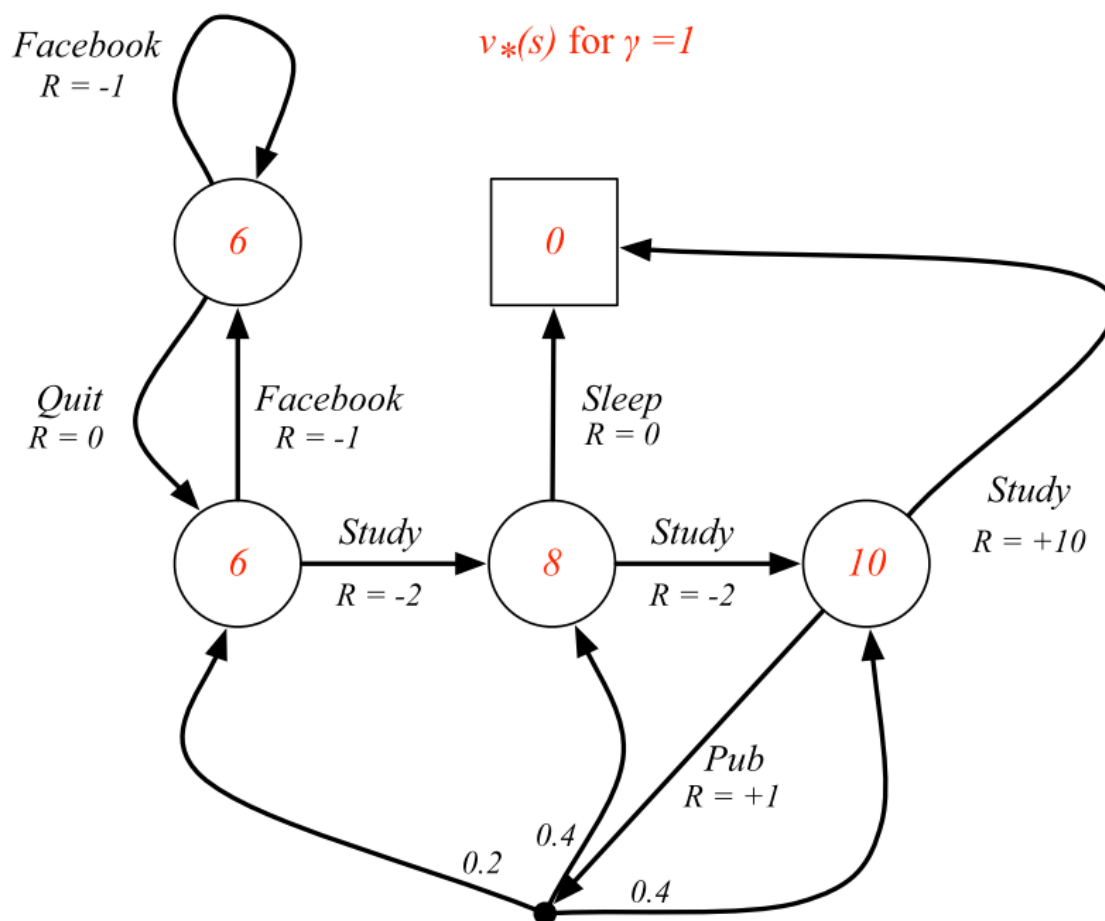
$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- Notes:

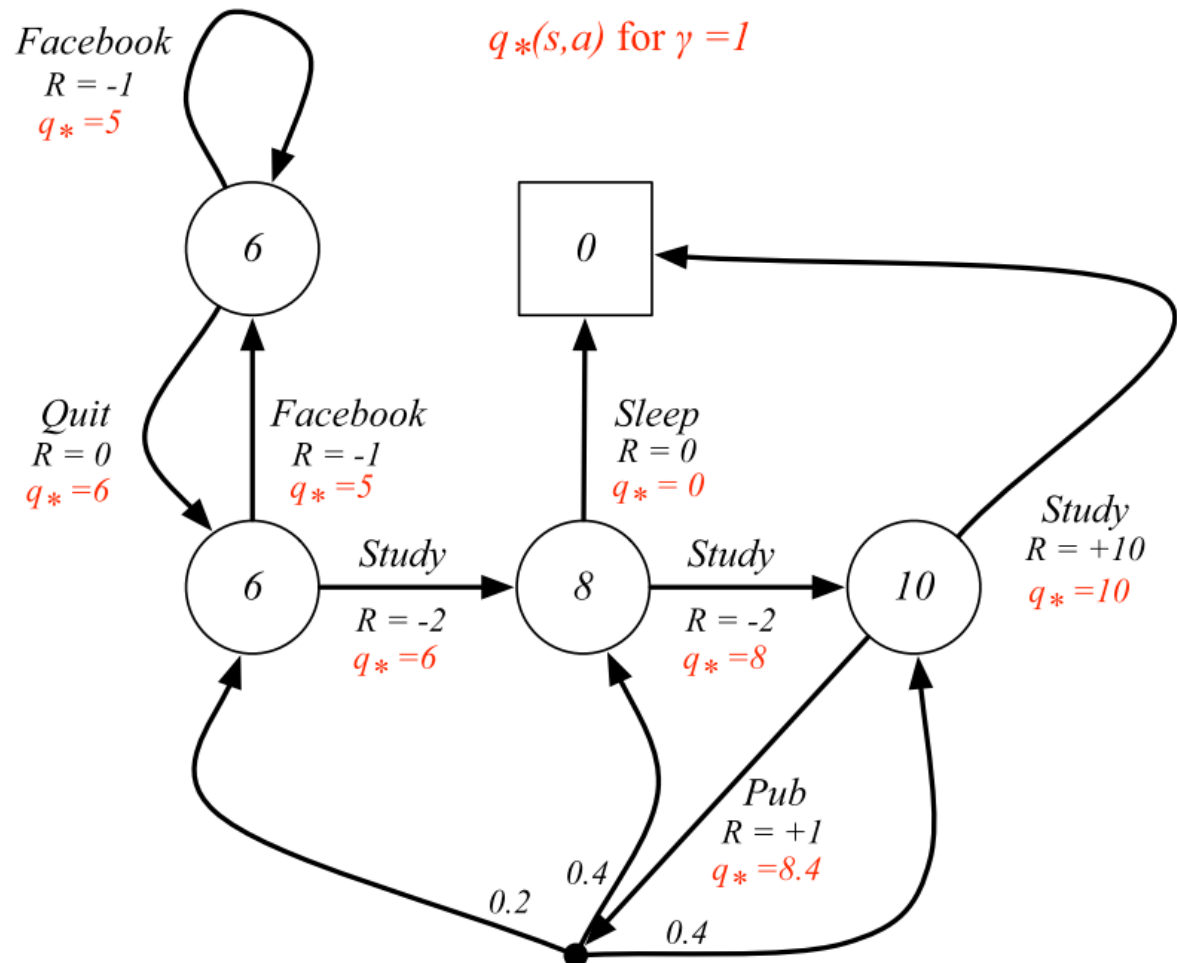
- The optimal value function specifies the best possible performance in the MDP.
- An MDP is “solved” when we know the optimal value fn.



Example: Optimal Value Function for Student MDP



Example: Optimal Action-Value Function for Student MDP



Optimal Policy

- Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_{\pi}(s) \geq v_{\pi'}(s), \forall s$$

- Theorem: For any Markov Decision Process,

- There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$.

- All optimal policies achieve the optimal value function,

$$v_{\pi_*}(s) = v_*(s)$$

- All optimal policies achieve the optimal action-value function,

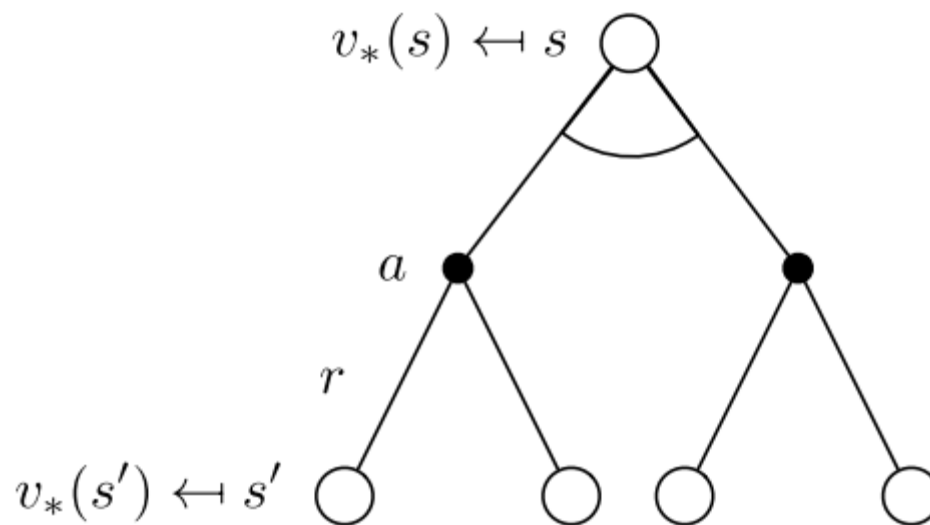
$$q_{\pi_*}(s, a) = q_*(s, a)$$

Finding an Optimal Policy

- An optimal policy can be found by maximizing over $q_*(s, a)$,
 - $\pi(a|s) = 1$, if $a = \operatorname{argmax}_{a \in \mathcal{A}} q_*(s, a)$
 - $\pi(a|s) = 0$, otherwise.
- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy
- What about state value function $v_*(s)$?
 - Similar, but we need to know model, $\mathcal{P}_{ss'}^a$. → not model free.



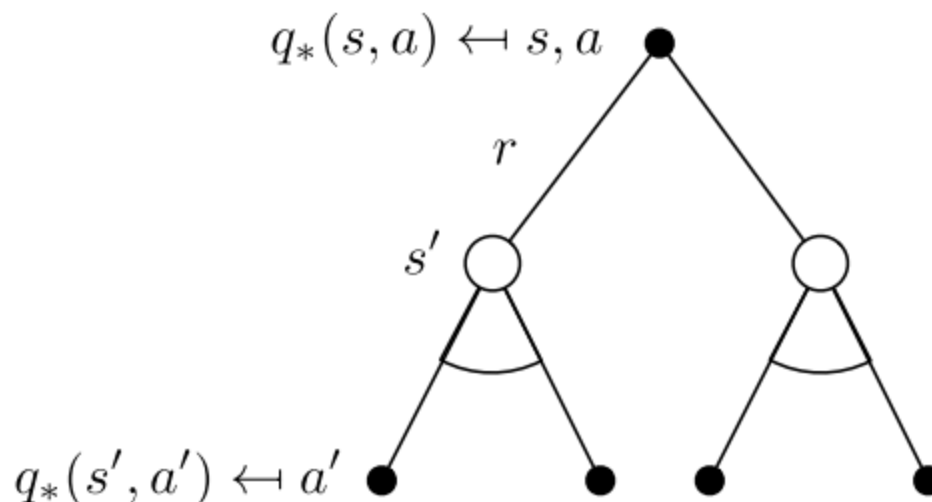
Bellman Optimality Equation for V^*



$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$



Bellman Optimality Equation for Q^*



$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a' \in \mathcal{A}} q_\pi(s, a')$$



Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
 - Value Iteration
 - Policy Iteration
 - Q-learning
 - Sarsa

Extensions to MDPs

- Infinite and continuous MDPs
 - Countably infinite state and/or action spaces
 - ▶ Straightforward
 - Continuous state and/or action spaces
 - ▶ Closed form for linear quadratic model (LQR)
 - Continuous time
 - ▶ Requires partial differential equations
 - ▶ Hamilton-Jacobi-Bellman (HJB) equation
 - ▶ Limiting case of Bellman equation as time-step
- Partially observable MDPs
 - E.g., Mahjong (as we mentioned)
- Undiscounted, average reward MDPs (ignored)



Dynamic Programming

- (Sutton) The term dynamic programming (DP) refers to a collection of algorithms that
 - compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).
- (Silver) A method for solving complex problems by **breaking them down into subproblems**
 - Solve the subproblems,
 - Combine solutions to subproblems
- (Algorithm textbook by Cormen et al.) says
 - DP, like the divide-and-conquer method, solves problems by combining the solutions to subproblems.
 - DP is typically applied to **optimization problems**.
 - Applications:
 - ▶ String algorithms (e.g. sequence alignment)
 - ▶ Graph algorithms (e.g. shortest path algorithms)
 - ▶ Bioinformatics (e.g. lattice models)



Why is DP related?

- Sequential or temporal component to the problem optimizing
 - a “program”, i.e. a policy,
 - values, i.e., state values and state action values
- Like solving LCS (longest common sequence) problem.
 - The optimal actions.
 - The optimal values.

		j	0	1	2	3	4	5	6
i	y_j	B	D	C	A	B	A		
	x_i								
0	x_i	0	0	0	0	0	0	0	
1	A	0	↑	↑	↑	↖1	←1	↖1	
2	B	0	↖1	←1	←1	↑1	↖2	←2	
3	C	0	↑1	↑1	↖2	←2	↑2	↑2	
4	B	0	↖1	↑1	↑2	↑2	↖3	←3	
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3	
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4	
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4	



Requirements for Dynamic Programming

- Dynamic Programming is a very general solution method for problems which have two properties:
 - **Optimal substructure**
 - ▶ Principle of optimality applies
 - ▶ Optimal solution can be decomposed into subproblems
 - **Overlapping subproblems**
 - ▶ Subproblems recur many times
 - ▶ Solutions can be cached and reused
- Markov decision processes satisfy both properties
 - Bellman equation gives recursive decomposition
 - Value function stores and reuses solutions

Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
 - It is used for planning in an MDP
- For **prediction**: evaluate values
 - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
 - Output: **value function** v_π
- For **control**: find the optimal policy.
 - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
 - Output: optimal value function v_*
and: **optimal policy**, π_*



Three Approaches

- Policy Evaluation

- Directly solve Bellman Equation in matrix form (see above)
 - ▶ Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π , it becomes a MRP problem $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$.
- Use Iterative Policy Evaluation

- Policy Iteration

- Value Iteration

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup

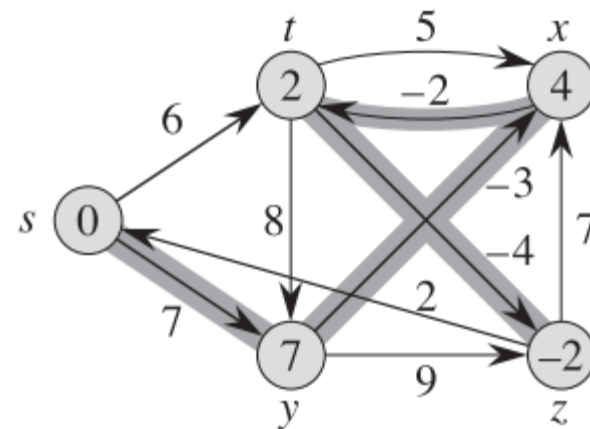
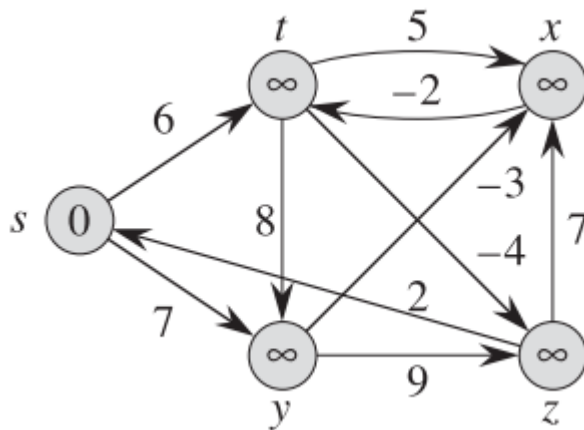
$$v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$$

- Using **synchronous backups**,
 - At each iteration $k + 1$,
 - ▶ for all states $s \in S$, update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s
- Notes:
 - We will discuss asynchronous backups later
 - Convergence to v_π will be proven at the end of the lecture
 - Review **the Bellman-Ford algorithm for the shortest path problem**.



The Shortest Path Problem

- A very simple MDP problem with
 - deterministic state transition \mathcal{P} .
- A good example to get a quick idea about why it works.
(see Cormen's Algorithm textbook)



Algorithms for the Shortest Path Problem

RELAX(u, v, w)

```
1  if  $v.d > u.d + w(u, v)$ 
2       $v.d = u.d + w(u, v)$ 
3       $v.\pi = u$ 
```

- Bellman-Ford Algorithm:

- Simple, but it works.

- Dijkstra Algorithm:

- Complex, but faster.

- Note:

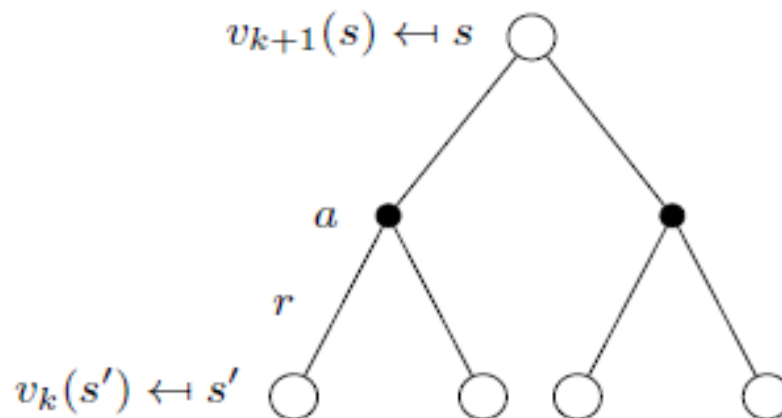
- The concept of Iterative Policy Evaluation is based on Bellman-Ford.

BELLMAN-FORD(G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  for  $i = 1$  to  $|G.V| - 1$ 
3      for each edge  $(u, v) \in G.E$ 
4          RELAX( $u, v, w$ )
5  for each edge  $(u, v) \in G.E$ 
6      if  $v.d > u.d + w(u, v)$ 
7          return FALSE
8  return TRUE
```



Iterative Policy Evaluation

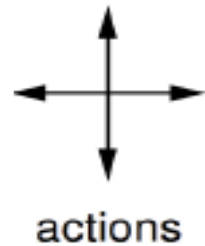


$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\pi}(s') \right)$$

$$\mathbf{v}^{k+1} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} \mathbf{v}^k$$



Example: Evaluating a Random Policy in the Small Gridworld



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$r = -1$
on all transitions

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

$$\pi(n | \cdot) = \pi(e | \cdot) = \pi(s | \cdot) = \pi(w | \cdot) = 0.25$$

















Iterative Policy Evaluation in Small Gridworld (I)

 $k = 0$ v_k for the
Random Policy

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

Greedy Policy
w.r.t. v_k

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

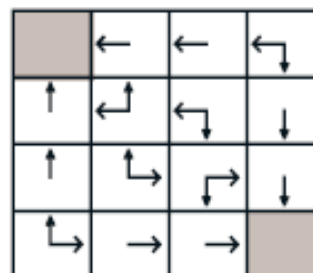
			
			
			
			



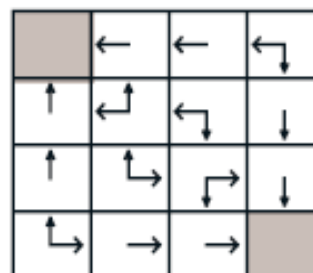
Iterative Policy Evaluation in Small Gridworld (2)

 $k = 3$

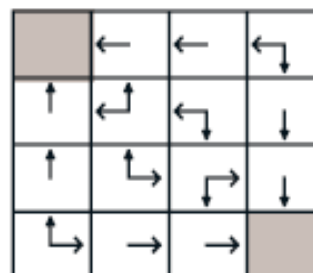
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

 $k = 10$

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



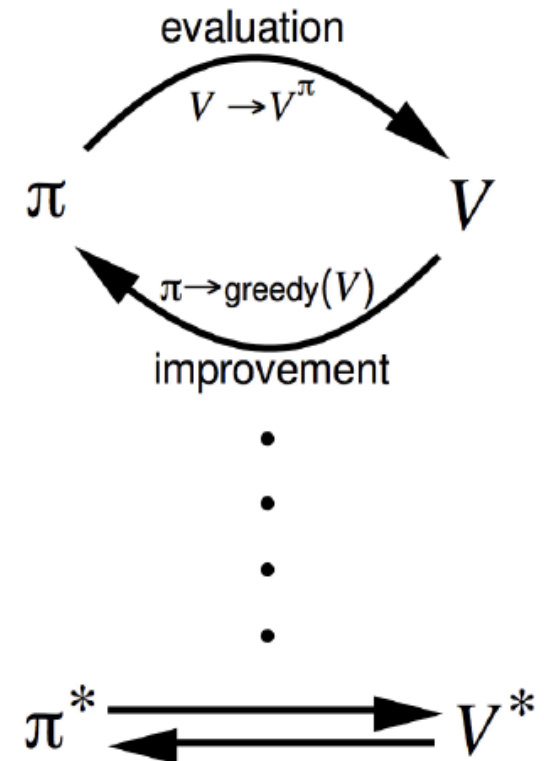
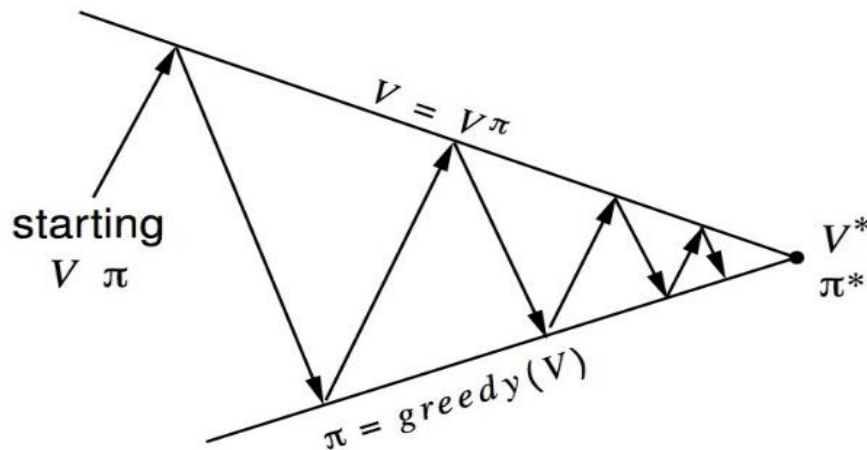
How to Improve a Policy

- Definition of policy improvement
 - Let π and π' be any pair of deterministic policies
 - ▶ For all $s \in S$, “ $\pi(s)$ performs better than $\pi'(s)$ ”. (We will see example)
- Given a policy π
 - Evaluate the policy π
$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$
 - Improve the policy by acting greedily with respect to v_π
$$\pi' = \text{greedy}(v_\pi)$$
- Notes:
 - In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
 - In general, need more iterations of improvement / evaluation
 - But this process of policy iteration always converges to π^*



Policy Iteration

- Policy evaluation \rightarrow Estimate v_π
 - Iterative policy evaluation
- Policy improvement \rightarrow Generate $\pi' \geq \pi$
 - Greedy policy improvement



Proof of Policy Improvement

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

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

- This improves the value from any state s over one step,
$$q_{\pi}(s, \pi'(s)) = \max_{a \in A} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$
- It therefore improves the value function, $v_{\pi'}(s) \geq v_{\pi}(s)$.

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



Converge of Policy Improvement

- If improvements stop,
 - That is, for $q_{\pi}(s, \pi'(s)) = \max_{a \in A} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)$
 - ▶ “ \geq ” becomes “=” when stopping.
- Then the Bellman optimality equation has been satisfied
$$v_{\pi}(s) = \max_{a \in A} q_{\pi}(s, a)$$
- This implies $v_{\pi}(s) = v_*(s)$ for all $s \in S$
- The above proves that π will converge to an optimal policy.

Variations of Policy Iteration

- Questions:

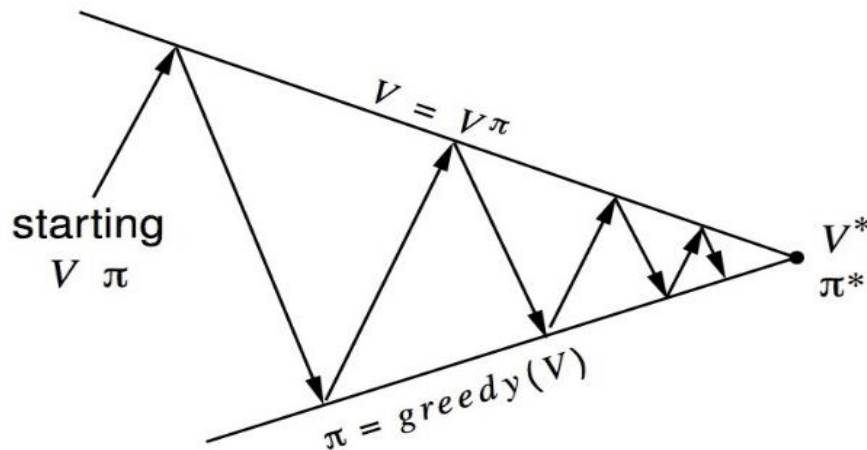
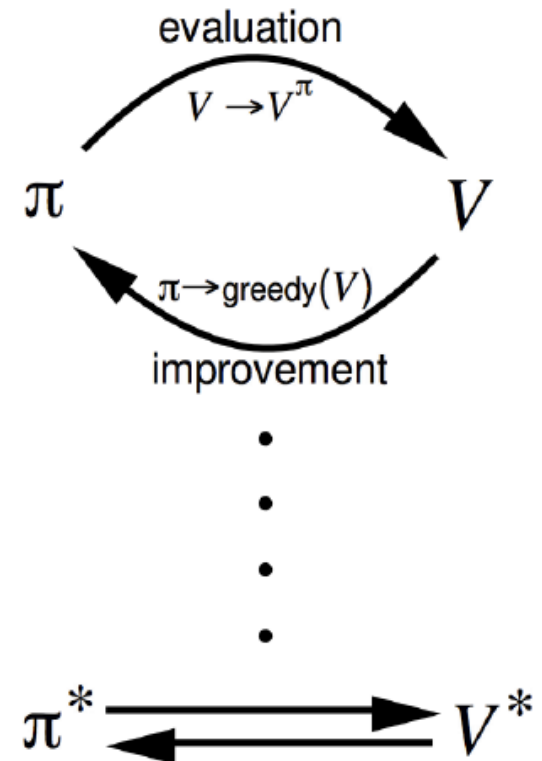
- Does policy evaluation need to converge to v_π ?
- Should we introduce a stopping condition, e.g. ϵ -convergence of value function?
- Simply stop after k iterations of iterative policy evaluation?
 - ▶ For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
 - ▶ Why not update policy every iteration? i.e. stop after $k = 1$

- This is equivalent to value iteration (next section)



Generalized Policy Iteration

- Policy evaluation \rightarrow Estimate v_π
 - **Any** policy evaluation algorithm
- Policy improvement \rightarrow Generate $\pi' \geq \pi$
 - **Any** policy improvement algorithm



Principle of Optimality

- Theorem (Principle of Optimality)
 - A policy $\pi(a|s)$ achieves the optimal value from state s , $v_\pi(s) = v_*(s)$, if and only if
 - For any state s' reachable from s , π achieves the optimal value from state s' , $v_\pi(s') = v_*(s')$

Deterministic Value Iteration

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

$$v_*(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s') \right)$$

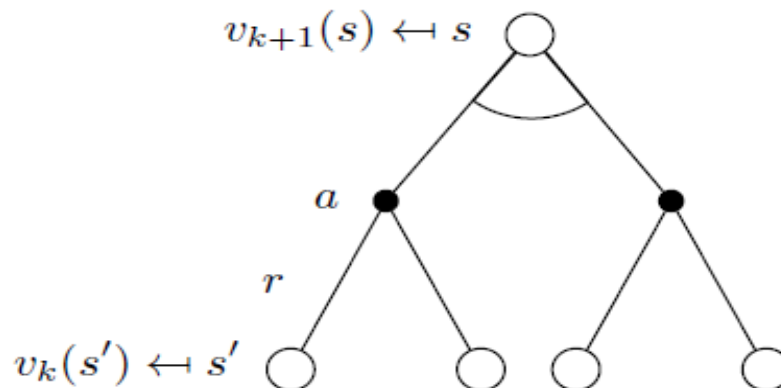
- Intuition:
 - Start with final rewards and work backwards
 - apply these updates iteratively
- Notes:
 - Still works with loopy, stochastic MDPs
 - Like most DP problems. (e.g., shortest path problem)



Value Iteration

- Problem:
 - find optimal policy π
- Solution: **directly find the optimal v_* without π .**
 - iterative application of Bellman optimality backup
$$v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$$
- Using synchronous backups (like Bellman-Ford)
 - At each iteration $k + 1$
 - ▶ For all states $s \in S$
 - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later
- Unlike policy iteration, there is no explicit policy

Value Iteration



$$v_{n+1}(s) = \max_{a \in A} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_n(s') \right)$$

or:

$$V^{(n+1)}(s) = \max_{a \in \mathcal{A}} \left(\mathbb{E}_{s'|s,a} [r + \gamma V^{(n)}(s')] \right)$$



Operator View

- Value iteration update

$$V^{(n+1)}(s) = \max_{a \in \mathcal{A}} (\mathbb{E}_{s'|s,a} [r + \gamma V^{(n)}(s')])$$

- It can be viewed as:

- A function $\mathcal{T}: \mathcal{S} \rightarrow \mathcal{S}$.
- Called **backup operator**.

$$[\mathcal{T}V](s) = \max_{a \in \mathcal{A}} (\mathbb{E}_{s'|s,a} [r + \gamma V(s')])$$
$$V^{(n+1)} = \mathcal{T}V^{(n)}$$

(Let V be an array of $v(s)$)

Algorithm Value Iteration

Initialize $V^{(0)}$ arbitrarily.

for $n = 0, 1, 2, \dots$ until termination condition do

$$V^{(n+1)} = \mathcal{T}V^{(n)}$$

end



Value Function Space

- Consider the vector space V over value functions
 - There are $|S|$ dimensions
 - Each point in this space fully specifies a value function $v(s)$
- What does a Bellman backup do to points in this space?
 - It brings value functions closer
 - Therefore the backups must converge on a unique solution

Value Function ∞ -Norm

- We will measure distance between state-value functions u and v by the ∞ -norm
 - i.e. the largest difference between state values,
$$||U - V||_{\infty} = \max_s |u(s) - v(s)|$$
- Let $\delta = ||(U - V)||_{\infty}$
 - $u(s) - v(s) \leq \delta$ for all s

Contraction for Bellman Optimality Backup

- Bellman optimality backup operator \mathcal{T} is a γ -contraction.

- Proof: Since

$$\max_{a \in \mathcal{A}}(x(a)) - \max_{a \in \mathcal{A}}(y(a)) \leq \max_{a \in \mathcal{A}}(x(a) - y(a))$$

- we have $||\mathcal{T}U - \mathcal{T}V||_{\infty}$
$$\begin{aligned} &= ||\max_{a \in \mathcal{A}}(\mathcal{R}^a + \gamma \mathcal{P}^a U) - \max_{a \in \mathcal{A}}(\mathcal{R}^a + \gamma \mathcal{P}^a V)||_{\infty} \\ &\leq ||\max_{a \in \mathcal{A}}[(\mathcal{R}^a + \gamma \mathcal{P}^a U) - (\mathcal{R}^a + \gamma \mathcal{P}^a V)]||_{\infty} \\ &= ||\gamma \mathcal{P}^a (U - V)||_{\infty} = \gamma ||\mathcal{P}^a (U - V)||_{\infty} \\ &\leq \gamma \delta = \gamma ||(U - V)||_{\infty} \end{aligned}$$

– Note: $(\mathcal{P}_{s::}^a (U - V)) \leq \delta$ for all s

→ $||\mathcal{P}^a (U - V)||_{\infty} \leq \delta$

▸ For \mathcal{P}^a , each row of matrix sums to 1.



Contraction Mapping Theorem

- Backup operator \mathcal{T} is a γ -contraction with modulus $\gamma(< 1)$ under ∞ -norm

$$\|\mathcal{T}U - \mathcal{T}V\|_{\infty} \leq \gamma \|U - V\|_{\infty}$$

- By contraction-mapping principle, it has a fixed point V^*
 - by iterating

$$V, \mathcal{T}V, \mathcal{T}^2V, \dots \rightarrow V^*$$

- Proof:

$$\|\mathcal{T}V - \mathcal{T}V^*\|_{\infty} \leq \gamma \|V - V^*\|_{\infty}$$

- Since $\mathcal{T}V^* = V^*$,

$$\|\mathcal{T}V - V^*\|_{\infty} \leq \gamma \|V - V^*\|_{\infty}$$

- By recurrence,

$$\|\mathcal{T}^n V - V^*\|_{\infty} \leq \gamma \|\mathcal{T}^{n-1} V - V^*\|_{\infty} \leq \dots \leq \gamma^n \|V - V^*\|_{\infty}$$

- Since $\gamma^n \rightarrow 0$, $\|\mathcal{T}^n V - V^*\|_{\infty} \rightarrow 0$.

- That is, $\mathcal{T}^n V \rightarrow V^*$



Policy Evaluation



- Problem: how to evaluate fixed policy π :

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s]$$

- Backwards recursion involves a backup operation

$$V^{(k+1)} = \mathcal{T}^\pi V^{(k)}$$

- \mathcal{T}^π is defined as:

$$[\mathcal{T}^\pi V](s) = \mathbb{E}_{s'|s, a=\pi(s)}[r + \gamma V(s')]$$

- \mathcal{T}^π is also a contraction with modulus γ , sequence

$$V, \mathcal{T}^\pi V, (\mathcal{T}^\pi)^2 V, (\mathcal{T}^\pi)^3 V, \dots \rightarrow V^\pi$$

- $V = \mathcal{T}^\pi V$ is a linear equation that we can solve directly.



Contraction for Bellman Expectation Backup

- Bellman Expectation Backup operator \mathcal{T}^π is a γ -contraction,

- Proof:

$$\begin{aligned} \|\mathcal{T}^\pi U - \mathcal{T}^\pi V\|_\infty &= \|(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi U) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi V)\|_\infty \\ &= \|\gamma \mathcal{P}^\pi (U - V)\|_\infty \\ &\leq \gamma \delta = \gamma \|U - V\|_\infty \end{aligned}$$

– Note:

- ▶ $(\mathcal{P}_{s::}^\pi(U - V)) \leq \delta$ for all s
→ $\|\mathcal{P}^\pi(U - V)\|_\infty \leq \delta$
 - For \mathcal{P}^π , each row of matrix sums to 1.



Policy Iteration: Overview

- Alternate between
 - Evaluate policy $\pi \Rightarrow V^\pi$
 - Set new policy to be greedy policy for V^π
$$\pi(s) = \underset{a}{\operatorname{argmax}} \mathbb{E}_{s'|s,a} [R_{t+1} + \gamma V^\pi(s')]]$$
- Guaranteed to converge to optimal policy and value function in a finite number of iterations, when $\gamma < 1$
- Value function converges faster than in value iteration

Algorithm Policy Iteration

Initialize $\pi^{(0)}$ arbitrarily.

for $n = 1, 2, \dots$ until termination condition do

$$V^{(n+1)} = \text{Solve } [V = \mathcal{T}^{\pi^{(n-1)}} V]$$

$$\pi^{(n)} = \mathcal{G} V^{(n)}$$

\mathcal{G} : a greedy mapping function.

end



Modified Policy Iteration

- Update π to be the greedy policy, then value function with k backups (k -step lookahead)

Algorithm Modified Policy Iteration

Initialize $V^{(0)}$ arbitrarily.

for $n = 1, 2, \dots$ until termination condition do

$$\pi^{(n+1)} = \mathcal{G}V^{(n)}$$

$$V^{(n+1)} = \left(\mathcal{T}^{\pi^{(n+1)}} \right)^k V^{(n)}, \text{ for integer } k \geq 1.$$

end

- $k = 1$: value iteration
- $k = \infty$: policy iteration



Exercise

- What if $\gamma = 1$?