Markov Decision Process

Hwanjo Yu

POSTECH

http://di.postech.ac.kr/hwanjoyu

Search problem

Markov decision processes

Reinforcement learning & games

Constraint satisfaction problems

Markov networks

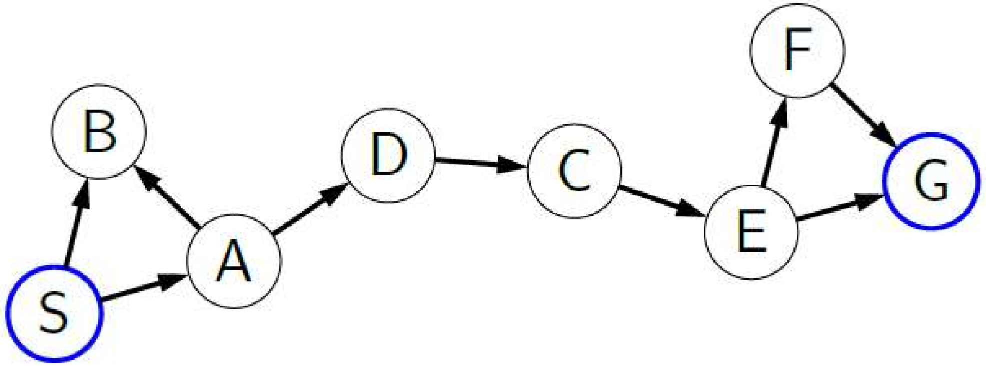
Bayesian networks

Reflex **States** **Variables** **Logic**

"Low-level intelligence" "High-level intelligence"

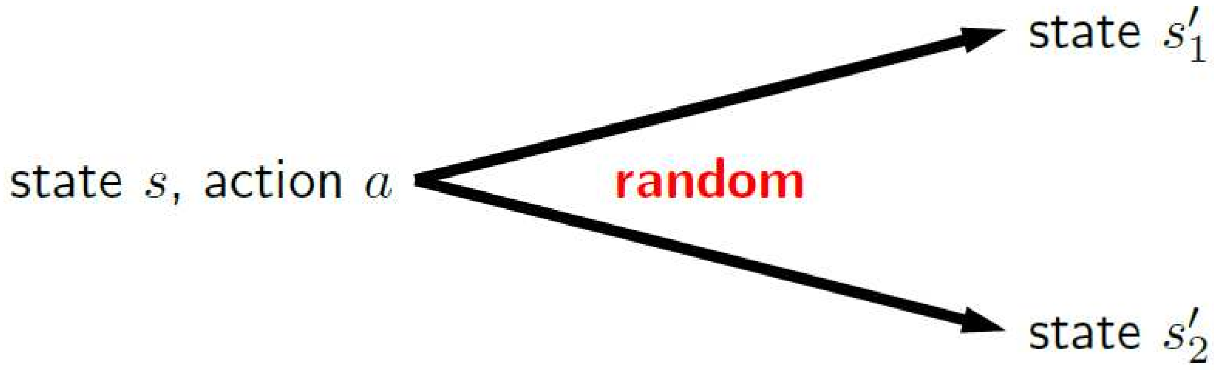
Machine learning

So far: search problems



state s, action a $\xrightarrow{\textbf{deterministic}}$ state $\text{Succ}(s, a)$

Uncertainty in the real world



Applications

- Robotics: decide where to move, but actuators can fail, hit unseen obstacles, etc.
- Resource allocation: decide what to produce, don't know the customer demand for various products
- Agriculture: decide what to plant, but don't know weather and thus crop yield

Volcano crossing



		-50	20
		-50	
2			

Roadmap

MDP modeling

Policy evaluation

Policy iteration

Value iteration

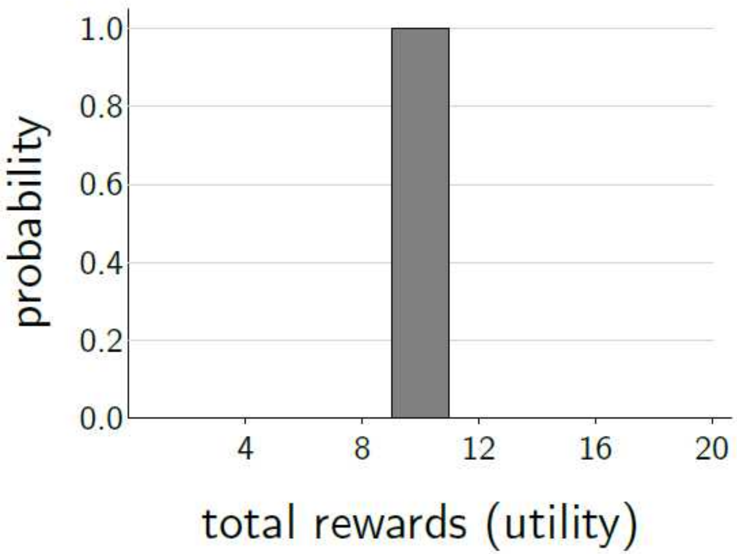
Dice game

Example: dice game

- For each round r = 1, 2, …
  - You choose stay or quit.
  - If quit, you get $10 and we end the game.
  - If stay, you get $4 and then I roll a 6-sided dice.
    - If the dice results in 1 or 2, we end the game.
    - Otherwise, continue to the next round.

Rewards

If follow policy “quit”:

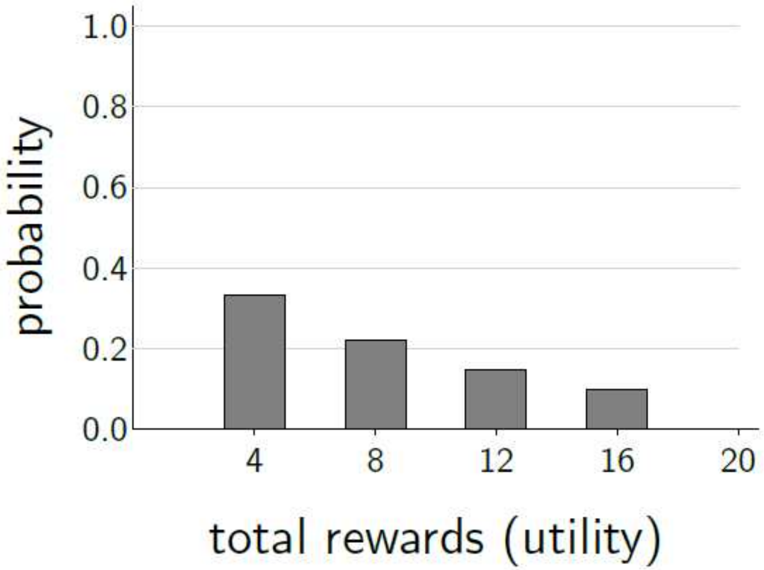


Expected utility:

$$1(10) = 10$$

Rewards

If follow policy "stay":



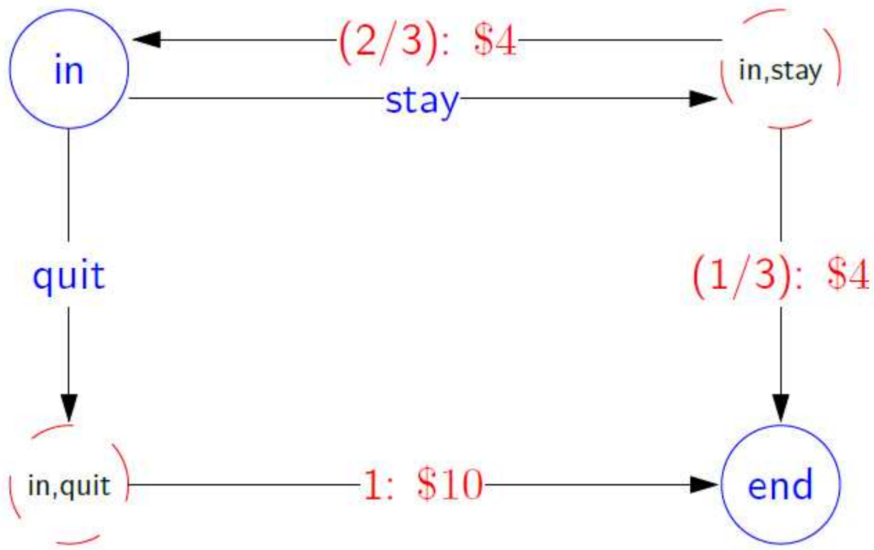
Expected utility:

$$4 + \frac{2}{3}(4) + \frac{2}{3}\cdot\frac{2}{3}(4) + \frac{2}{3}\cdot\frac{2}{3}\cdot\frac{2}{3}(4) + \cdots = 12$$

MDP for dice game

Example: dice game

- For each round r = 1, 2, …
  - you choose stay or quit.
  - If quit, you get \$10 and we end the game.
  - If stay, you get \$4 and then I roll a 6-sided dice.
    - If the dice results in 1 or 2, we end the game.
    - Otherwise, continue to the next round.



Markov Decision Process (MDP)

Definition: Markov Decision Process

- States: the set of states
- $s_{\text{start}} \in \text{States}$: starting state
- $\text{Actions}(s)$: possible actions from state s
- $T(s, a, s')$: probability of s' if take action a in state s
- $\text{Reward}(s, a, s')$: reward for the transition (s, a, s')
- $\text{IsEnd}(s)$: whether s is an end of game
- $0 \leq \gamma \leq 1$: discount factor (default: 1)

Search problems

Definition: search problem

- States: the set of states
- $s_{\text{start}} \in \text{States}$: starting state
- $\text{Actions}(s)$: possible actions from state s
- $\text{Succ}(s, a)$: where we end up if take action a in state s
- $\text{Cost}(s, a)$: cost for taking action a in state s
- $\text{IsEnd}(s)$: whether s is an end of game
- $0 \leq \gamma \leq 1$: discount factor (default: 1)

$\text{Succ}(s, a) \Rightarrow T(s, a, s')$
$\text{Cost}(s, a) \Rightarrow \text{Reward}(s, a, s')$

Transitions

Definition: transition probabilities

- The transition probabilities $T(s, a, s')$ specify the probability of ending up in state s' if taken action a in state s.

Example: transition probabilities

s	a	s'	$T(s, a, s')$
In	quit	end	1
In	stay	in	2/3
In	stay	end	1/3

Probabilities sum to one

Example: transition probabilities

s	a	s'	$T(s,a,s')$
In	quit	end	1
In	stay	in	2/3
In	stay	end	1/3

- For each state s and action a:

$$\sum_{s' \in \text{States}} T(s,a,s') = 1$$

- Successors: s' such that $T(s,a,s') > 0$

Transportation example

Example: transportation

- Street with blocks numbered 1 to n.
- Walking from s to $s + 1$ takes 1 minute.
- Taking a magic tram from s to $2s$ takes 2 minutes.
- How to travel from 1 to n in the least time?
- Tram fails with probability 0.5.

What is a solution?

Search problem: path (sequence of actions)

MDP:

Definition: policy

- A policy π is a mapping from each state $s \in \text{States}$ to an action $a \in \text{Actions}(s)$.

Example: volcano crossing

s	$\pi(s)$
(1,1)	S
(2,1)	E
(3,1)	N
…	…

Roadmap

MDP modeling

Policy evaluation

Policy iteration

Value iteration

Evaluating a policy

Definition: utility

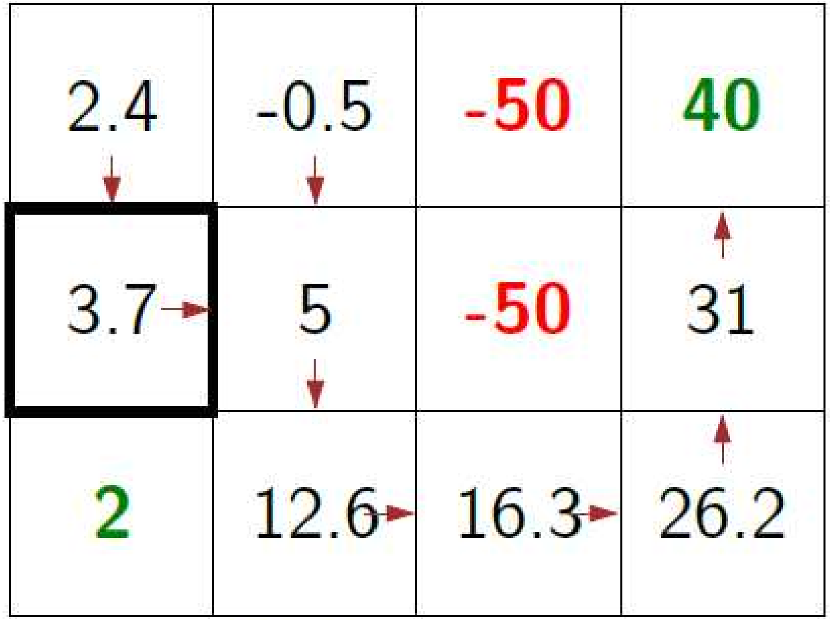
- Following a policy yields a **random path (an episode)**.
- The utility of a policy is the (discounted) sum of the rewards on the path (this is a random quantity). For example of the dice game,

Path	Utility
[in; stay, 4, end]	4
[in; stay, 4, in; stay, 4, in; stay, 4, end]	12
[in; stay, 4, in; stay, 4, end]	8
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	16

Definition: value (expected utility)

- The **value** of a policy is the **expected utility**.

Evaluating a policy: volcano crossing



a	r	s
		(2,1)
E	-0.1	(2,2)
S	-0.1	(3,2)
E	-0.1	(3,3)
E	-50.1	(2,3)

Value: 3.7

Utility: -50.4

Discounting

Definition: utility

- Path: $s_0 a_1 r_1 s_1 a_2 r_2 s_2$, ... (action, reward, new state).
- The **utility** with discount γ is

$$u_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$$

- Discount $\gamma = 1$ (save for the future):
  - [stay, stay, stay, stay]: $4 + 4 + 4 + 4 = 16$
- Discount $\gamma = 0$ (live in the moment):
  - [stay, stay, stay, stay]: $4 + 0 \cdot (4 + \cdots) = 4$
- Discount $\gamma = 0.5$ (balanced life):
  - [stay, stay, stay, stay]: $4 + \frac{1}{2} \cdot 4 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 4 = 7.5$

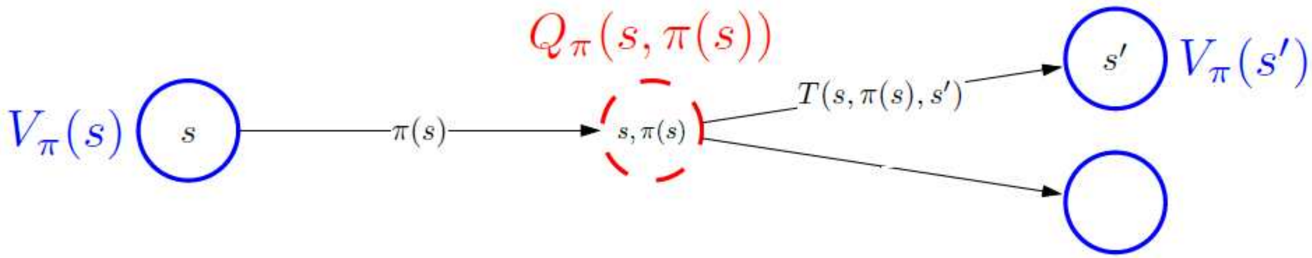
Policy evaluation

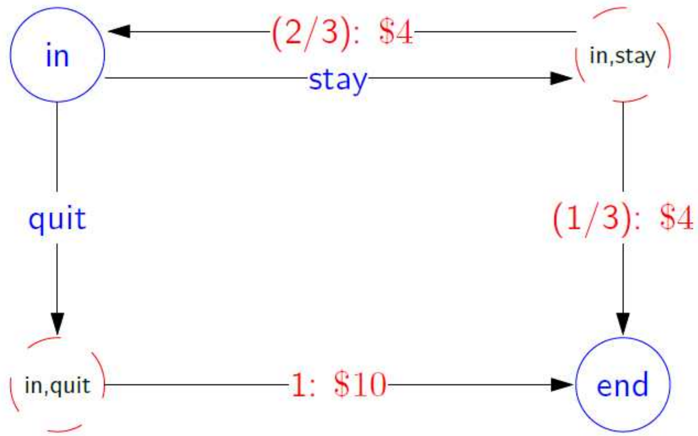
Definition: value of a policy

- Let $V_\pi(s)$ be the expected utility received by following policy π from state s (labeling the state nodes)

Definition: Q-value of a policy

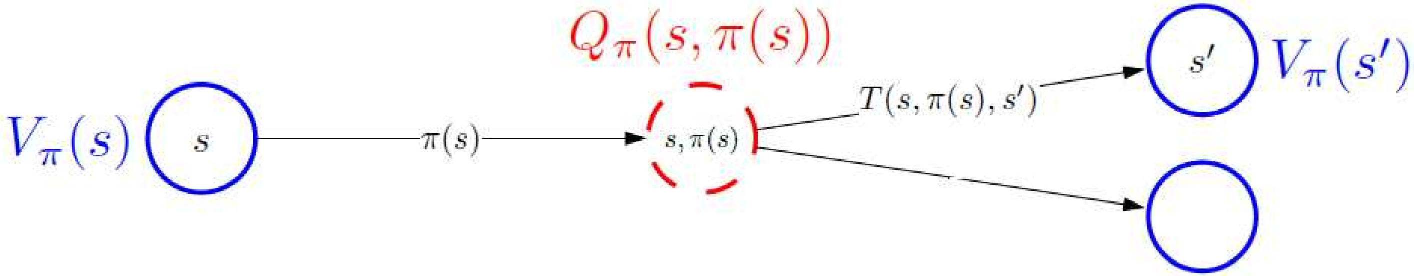
- Let $Q_\pi(s, a)$ be the expected utility of taking action a from state s, and then following policy π (labeling the chance nodes)





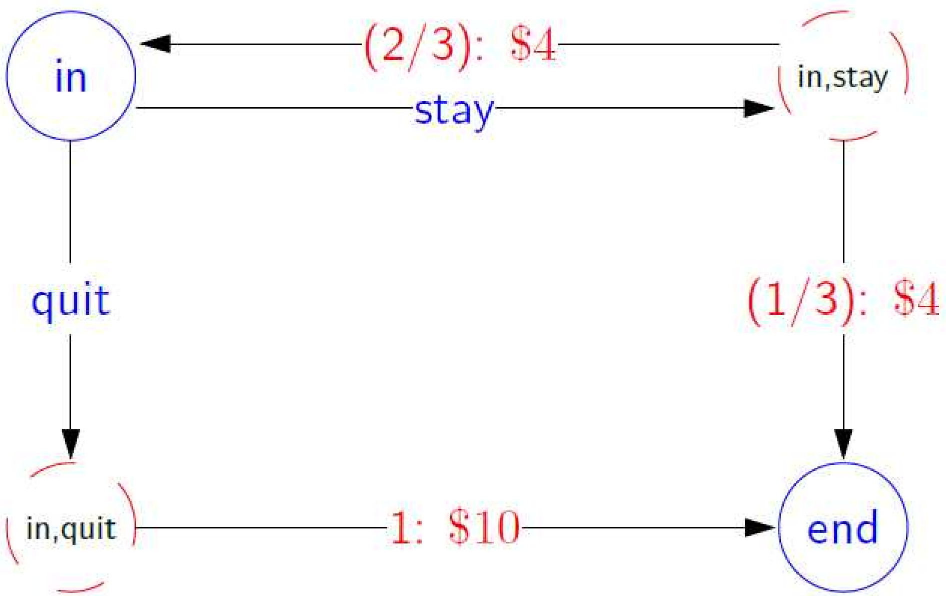
Policy evaluation

Plan: define recurrences relating value and Q-value



- $V_\pi(s) = \begin{cases} 0 & \text{If IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$
- $Q_\pi(s, a) = \sum_{s'} T(s, a, s')[\text{Reward}(s, a, s') + \gamma V_\pi(s')]$

Dice game



Let π be the "stay" policy: $\pi(\text{in}) = \text{stay}$

- $V_\pi(\text{end}) = 0$
- $V_\pi(\text{in}) = \frac{1}{3}\big(4 + 1 \cdot V_\pi(\text{end})\big) + \frac{2}{3}\big(4 + 1 \cdot V_\pi(\text{in})\big)$

In this case, can solve in closed form:

- $V_\pi(\text{in}) = 12$

Policy evaluation

Key idea: iterative algorithm

- Start with arbitrary policy values and repeatedly apply recurrences to converge to true values.

Algorithm: policy evaluation

- Initialize $V_\pi^{(0)}(s) \leftarrow 0$ for all states s.
- For iteration $t = 1, \ldots, t_{\mathrm{PE}}$:
  - For each state s:

$$V_\pi^{(t)}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[\mathrm{Reward}(s, \pi(s), s') + \gamma V_\pi^{(t-1)}(s')]$$

Policy evaluation implementation

- How many iterations (t_{PE})? Repeat until values don't change much:

$$\max_{s \in States} |V_\pi^{(t)}(s) - V_\pi^{(t-1)}(s)| \leq \epsilon$$

- Don't store $V_\pi^{(t)}$ for each iteration t, need only last two:

$$V_\pi^{(t)} \text{ and } V_\pi^{(t-1)}$$

Complexity

Algorithm: policy evaluation

- Initialize $V_\pi^{(0)}(s) \leftarrow 0$ for all states s.
- For iteration $t = 1, \dots, t_{\text{PE}}$:
  - For each state s:

$$V_\pi^{(t)}(s) \leftarrow \underbrace{\sum_{s'} T(s,\pi(s),s')[\text{Reward}(s,\pi(s),s') + \gamma V_\pi^{(t-1)}(s')]}_{Q_\pi^{(t-1)}(s,\pi(s))}$$

MDP complexity

- S states
- A actions per state
- S' successors (number of s' with $T(s,a,s') > 0$)

Time: $O(t_{\text{PE}} S S')$

Policy evaluation on dice game

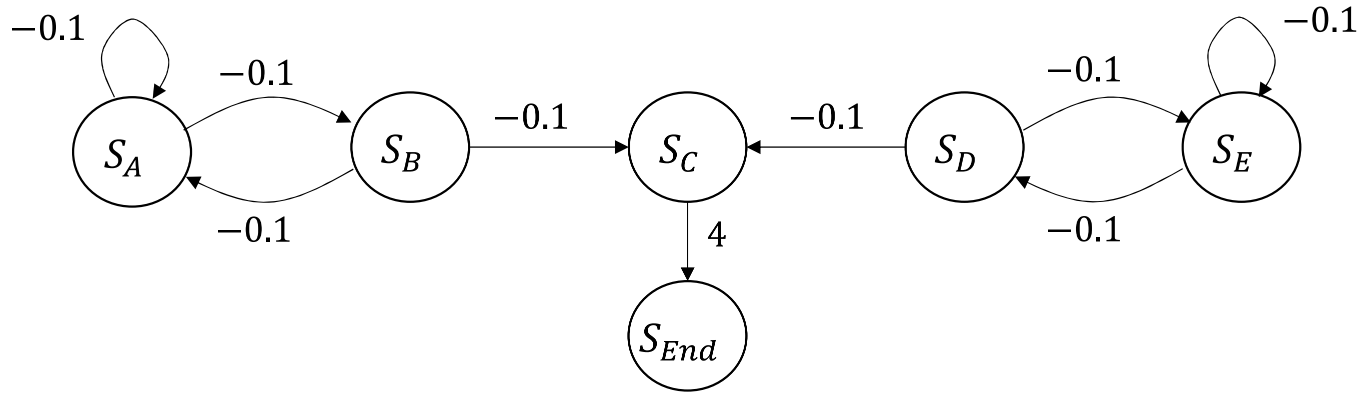
Let π be the "stay" policy: $\pi(\text{in}) = \text{stay}$

- $V_\pi^{(t)}(\text{end}) = 0$
- $V_\pi^{(t)}(\text{in}) = \frac{1}{3}(4 + V_\pi^{(t-1)}(\text{end})) + \frac{2}{3}(4 + 1 \cdot V_\pi^{(t-1)}(\text{in}))$

s	end	in	
$V_\pi^{(t)}$	0.00	12.00	($t = 100$ iterations)

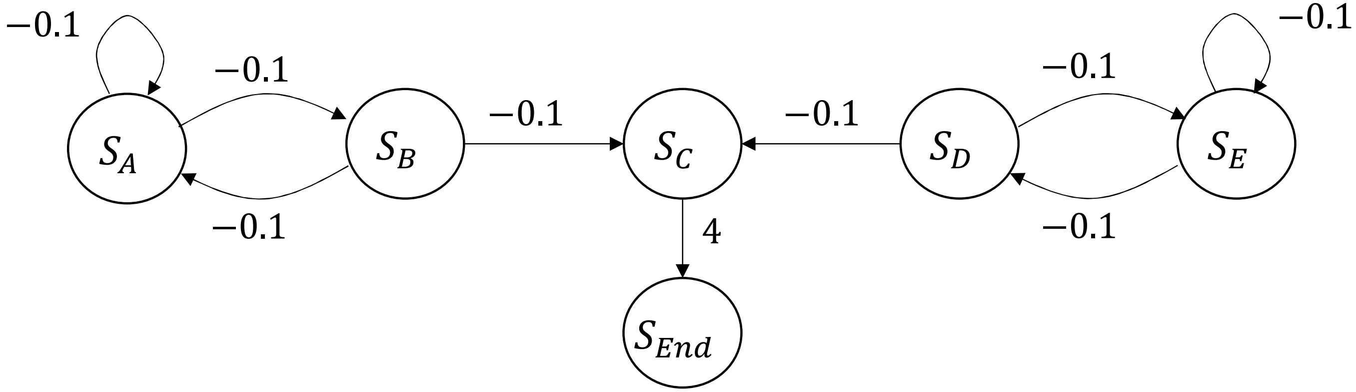
Converges to $V_\pi(\text{in}) = 12$

Policy evaluation example



- $\pi(s) =$ "$move$"
- $T(s, \pi(s), s') = 0.5$ except $T(s_C, \pi(s), s_{End}) = 1$
- $\text{Reward}(s, \pi(s), s') = -0.1$ except $\text{Reward}(s_C, \pi(s), s_{End}) = 4$

Policy evaluation computation



$$\boxed{V_\pi^{(t)}(s)} \Rightarrow$$

iteration t

state s

0	-0.1	-0.2	0.7	1.1	1.6	1.9	2.2	2.4	2.6
0	-0.1	1.8	1.8	2.2	2.4	2.7	2.8	3	3.1
0	4	4	4	4	4	4	4	4	4
0	-0.1	1.8	1.8	2.2	2.4	2.7	2.8	3	3.1
0	-0.1	-0.2	0.7	1.1	1.6	1.9	2.2	2.4	2.6

Summary so far

- MDP: graph with states, chance nodes, transition probabilities, rewards
- Policy: mapping from state to action (solution to MDP)
- Value of policy: expected utility over random paths
- Policy evaluation: iterative algorithm to compute value of policy

Roadmap

MDP modeling

Policy evaluation

Policy iteration

Value iteration

Policy improvement

So far: policy evaluation computes value of a fixed policy π

Goal: improve π to something slightly better π_{new}

Recall: $Q_\pi(s,a)$ is the expected utility of first taking action a in state s, and then following π

Algorithm: policy improvement

Input: value of policy V_π

Output: new policy π_{new}

- For each state s:
  - Compute $Q_\pi(s,a)$ from $V_\pi(s)$ for each a
  - $\pi_{\text{new}}(s) = arg \max_{a \in \text{Actions}(s)} Q_\pi(s,a)$

Policy improvement

Example: dice game

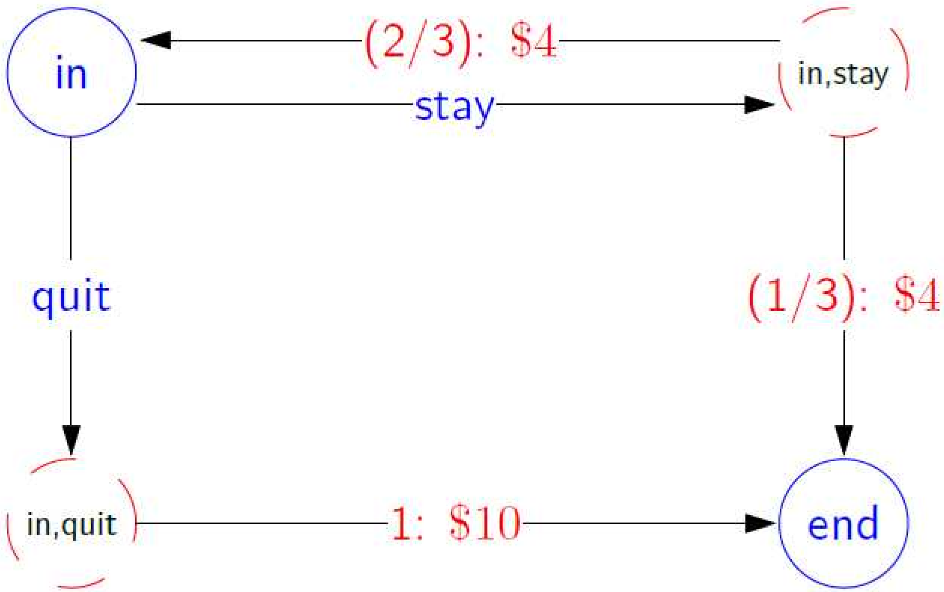
Suppose $\pi(\text{in}) = \text{quit}$.

Step 1:

- $Q_\pi(\text{in}, \text{quit}) = 10$
- $Q_\pi(\text{in}, \text{stay}) = \frac{2}{3}(4 + 10) + \frac{1}{3}(4 + 0) \cong 10.67$

Step 2:

- $\pi_{\text{new}}(\text{in}) = \text{stay}$



Policy improvement

Algorithm: policy improvement

Input: value of policy V_π

Output: new policy π_{new}

- For each state s:
  - Compute $Q_\pi(s, a)$ from $V_\pi(s)$ for each a
  - $\pi_{\text{new}}(s) = \arg\max_{a \in \text{Actions}(s)} Q_\pi(s, a)$

MDP complexity

- S states
- A actions per state
- S' successors (number of s' with $T(s, a, s') > 0$)

Time: $O(SAS')$

Policy iteration

Idea: rinse and repeat

Algorithm: policy iteration

- $\pi \leftarrow \text{arbitrary}$
- For $t = 1, \ldots, t_{\text{PI}}$ (or until π stops changing):
  - Run *policy evaluation* to compute V_π
  - Run *policy improvement* to get π_{new}
  - $\pi \leftarrow \pi_{\text{new}}$

Time: $O(t_{\text{PI}}(t_{\text{PE}} SS' + SA\,S'))$

Implementation trick: **warm start** policy evaluation with previous V_π

Roadmap

MDP modeling

Policy evaluation

Policy iteration

Value iteration

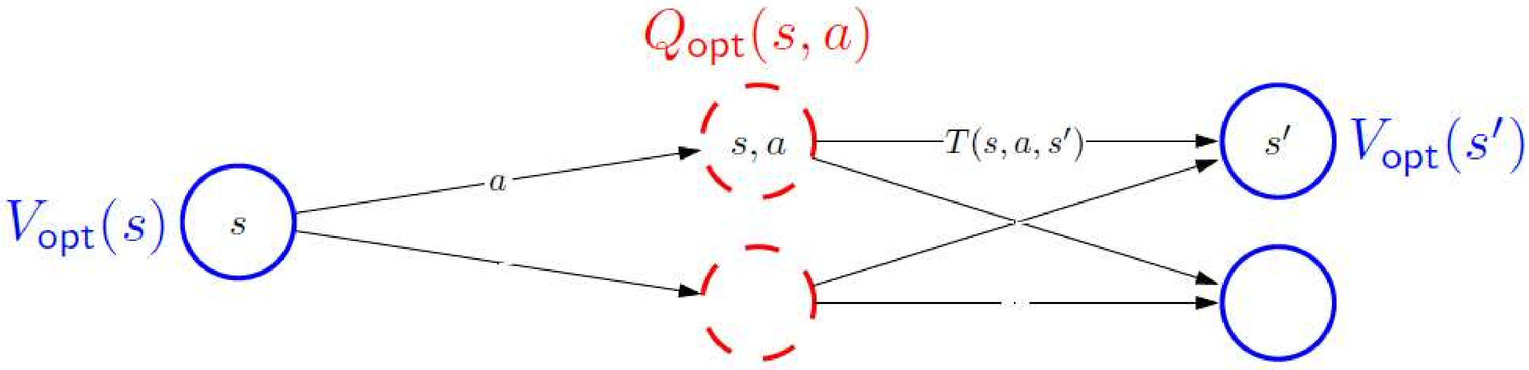
Optimal value and policy

Goal: try to get directly at maximum expected utility

Definition: optimal value

- The **optimal value** $V_{\text{opt}}(s)$ is the maximum value attained by any policy.

Optimal values and Q-values



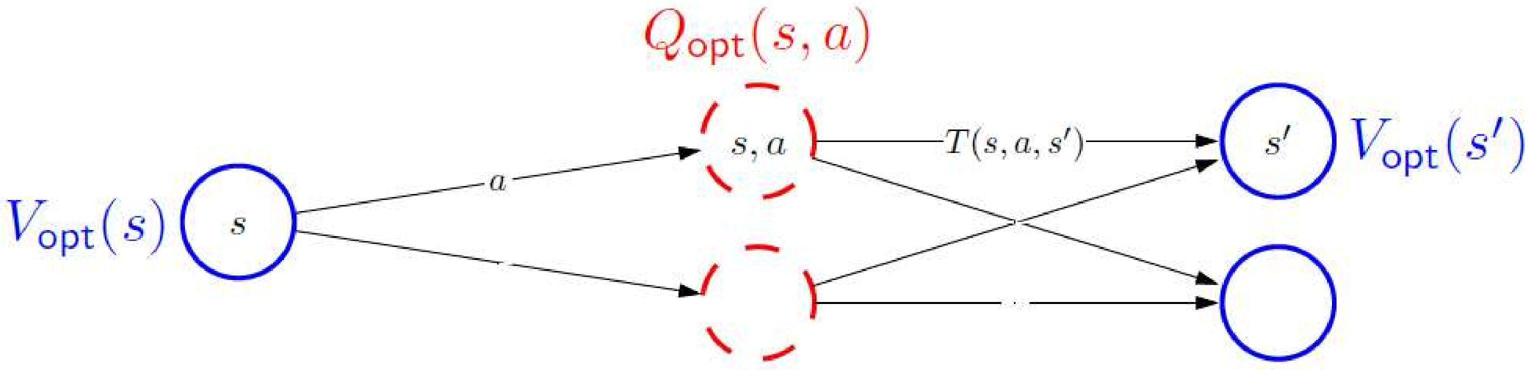
Optimal value if take action a in state s:

$$Q_{\text{opt}}(s,a) = \sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')]$$

Optimal value from state s:

$$V_{\text{opt}}(s) = \begin{cases} 0 & \text{If IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s,a) & \text{otherwise.} \end{cases}$$

Optimal policies



Given Q_{opt}, read off the optimal policy:

$$\pi_{\text{opt}}(s) = \arg\max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s, a)$$

Value iteration

Algorithm: value iteration [Bellman, 1957]

- Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s.
- For iteration $t = 1, \ldots, t_{\text{VI}}$:
  - For each state s:

$$V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')]}_{Q_{\text{opt}}^{(t-1)}(s,a)}$$

Time: $O(t_{\text{VI}} SA S')$

Value iteration: dice game

s	end	in
$V_{\text{opt}}^{(t)}$	0.00	12.00 ($t = 100$ iterations)
$\pi_{\text{opt}}(s)$	-	stay

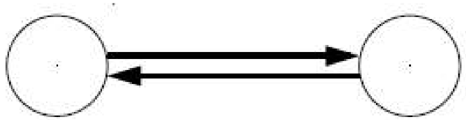
Convergence

Theorem: convergence

- Suppose either
  - discount $\gamma < 1$, or
  - MDP graph is acyclic.
- Then value iteration and policy iteration both converge to the correct answer.

Example: non-convergence

- discount $\gamma = 1$, zero rewards



Summary of algorithms

- Policy evaluation: $(\text{MDP}, \pi) \rightarrow V_\pi$
- Policy improvement: $(\text{MDP}, V_\pi) \rightarrow \pi_{\text{new}}$
- Policy iteration: $\text{MDP} \rightarrow (V_{\text{opt}}, \pi_{\text{opt}})$
- Value iteration: $\text{MDP} \rightarrow (V_{\text{opt}}, \pi_{\text{opt}})$

Unifying idea

Algorithms:

- Search DP computes $\text{FutureCost}(s)$
- Policy evaluation computes policy value $V_\pi(s)$
- Value iteration computes optimal value $V_{\text{opt}}(s)$

Recipe:

- Write down recurrence (e.g., $V_\pi(s) = \ldots V_\pi(s') \ldots$)
- Turn into iterative algorithm (replace mathematical equality with assignment operator)

Summary

- **Markov decision processes** (MDPs) cope with uncertainty
- Solutions are **policies** rather than paths
- **Policy evaluation** computes policy value (expected utility)
- **Policy iteration** and **value iteration** computes optimal value (maximum expected utility) and optimal policy
- Main technique: write recurrences -> algorithm
- Next time: reinforcement learning - when we don't know rewards, transition probabilities