

6. Markov Decision Processes 1 - Value Iteration

Intro



cs221.stanford.edu/q

Question

How would you get to Mountain View on Friday night in the least amount of time?

bike

drive

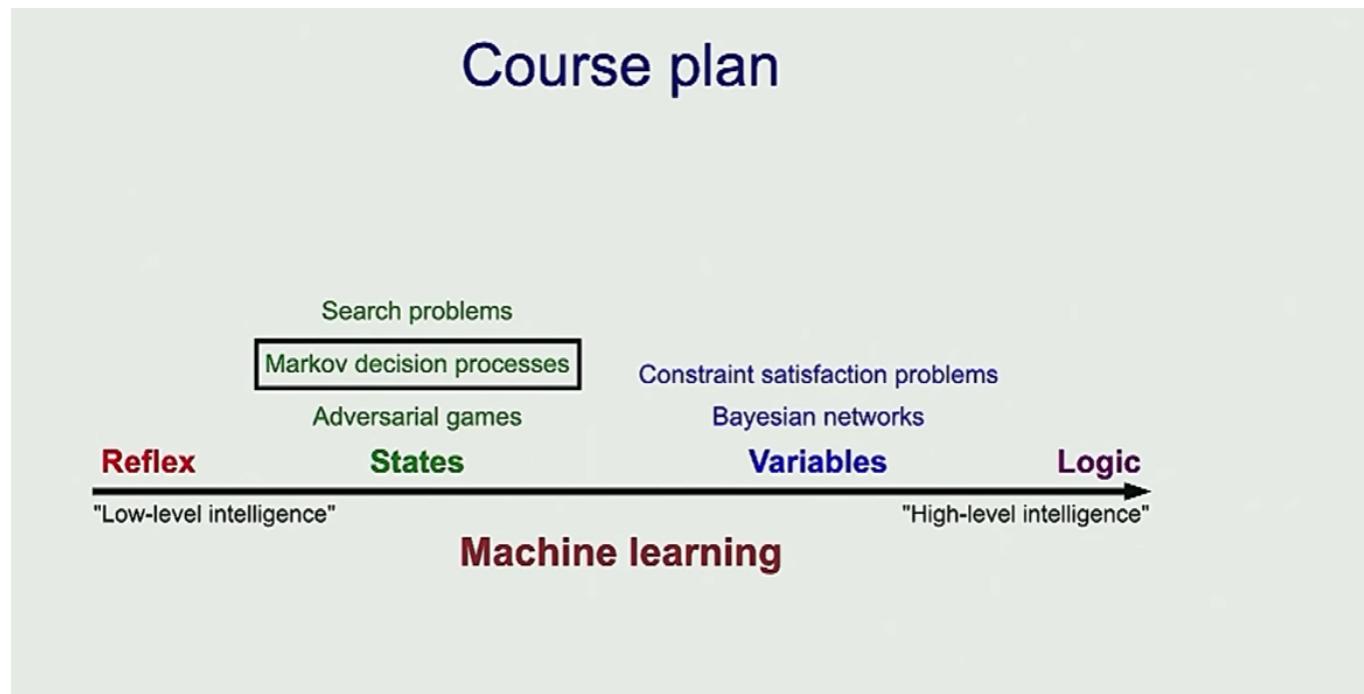
Caltrain

Uber/Lyft

fly

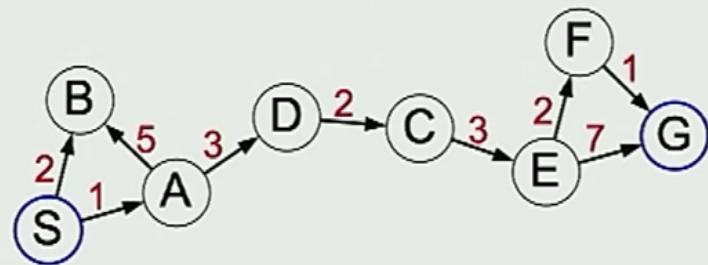
[activate](#) [deactivate](#) [reset](#) [report](#)

There are lot of uncertainties



We talked about search problems, where everything was deterministic. The solution was a sequence of actions

So far: search problems

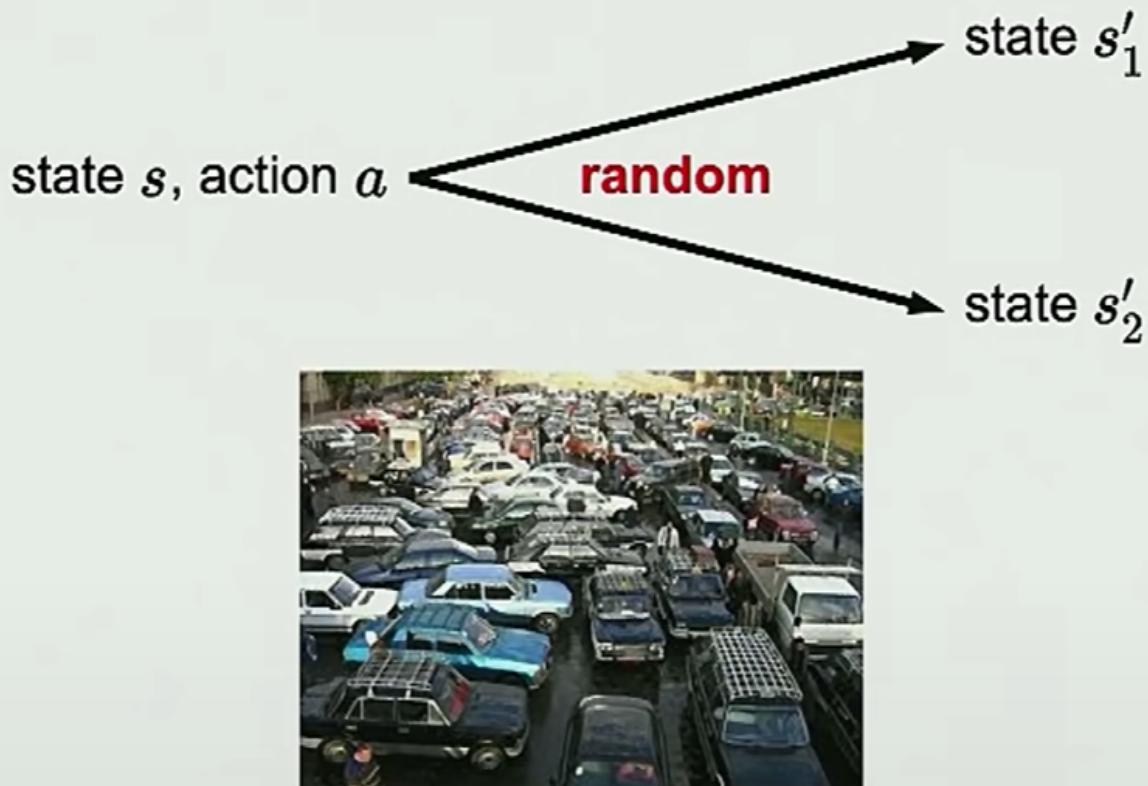


state s , action a **deterministic** → state $\text{Succ}(s, a)$



And now we are talking about this next class of state-based functions, Markov Decision Processes. The idea is that, we take actions, but we may not end up where we expected to, because there's a nature around you and there is going to be uncertainty and stuff that you did not expect.

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



```
// Model  
moveReward = 0 // For every action you take  
passReward = 20 // If get to far green  
volcanoReward = -50 // If fall into volcano  
slipProb = 0.3 // If slip, go in random direction  
discount = 1 // How much to value the future  
  
// CHANGE THIS to 10  
numIters = 10 // # iterations of value iteration
```

1.4	-2.9	-50	20
1.9	1.1	-50	13.8
2	6.5	7.5	13.2

Value: 1.86



(or press ctrl-enter)

CS221 / Autumn 2019 / Liang & Sadich

The dice game

Dice game



Example: dice game

For each round $r = 1, 2, \dots$

- 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.

Start

Stay

Quit

Dice: 2

Rewards: 8

Rewards

Rewards

If follow policy "stay":



Expected utility:

$$\frac{1}{3}(4) + \frac{2}{3} \cdot \frac{1}{3}(8) + \frac{2}{3} \cdot \frac{2}{3} \cdot \frac{1}{3}(12) + \dots = 12$$

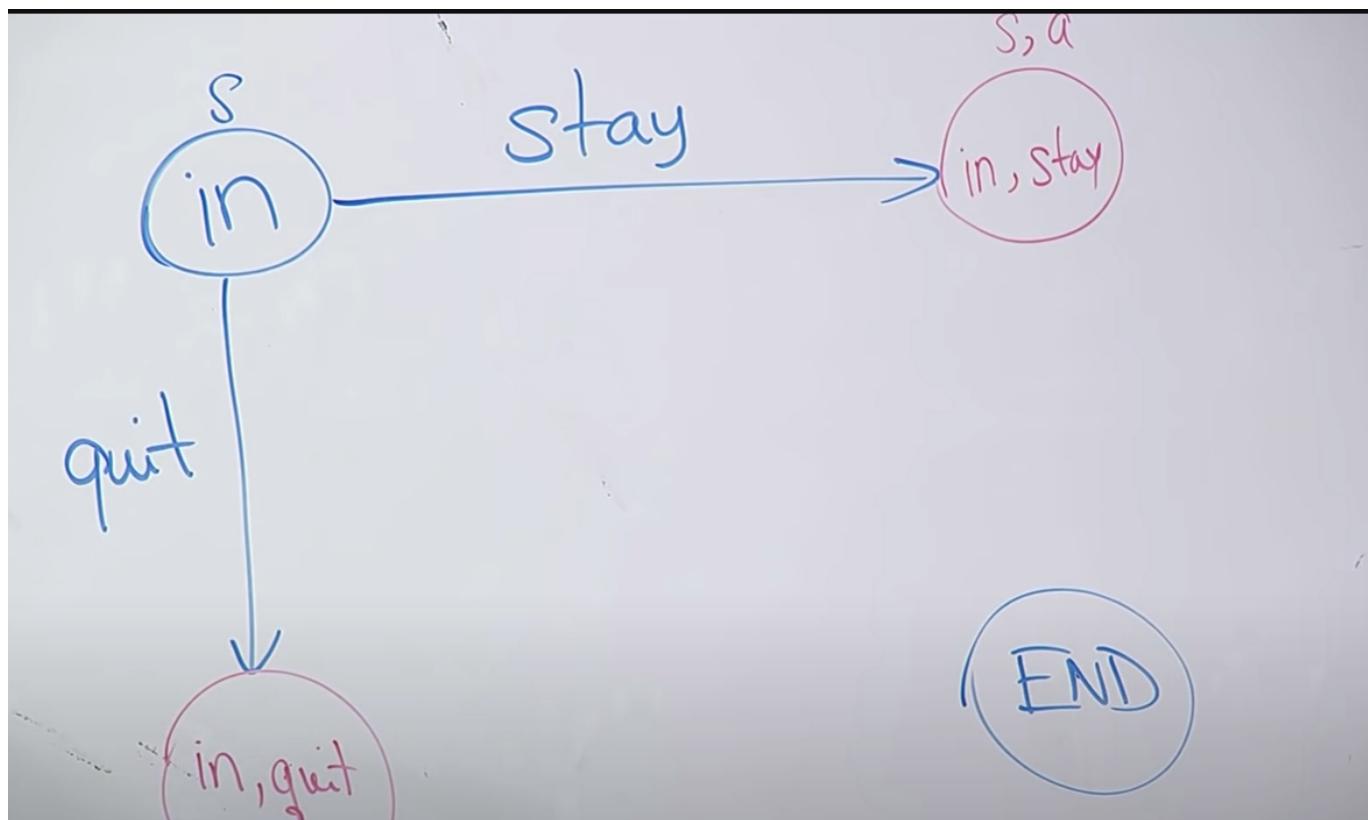
Rewards

If follow policy "quit":



Expected utility:

$$1(10) = 10$$



State: You can either in or out(end of the game) Blue: States Red: "Chance Node", state with actions, flagging

the state that you're in when taking that action From the "Chance Node", we are going to take probabilities

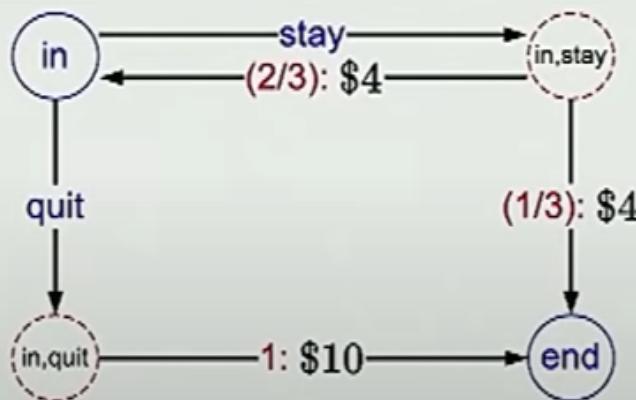
MDP for dice game



Example: dice game

For each round $r = 1, 2, \dots$

- 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



Definition: Markov decision process

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

Actions(s): possible actions from state s

$T(s, a, s')$: probability of s' if take action a in state s

Reward(s, a, s'): reward for the transition (s, a, s')

IsEnd(s): whether at end of game

$0 \leq \gamma \leq 1$: discount factor (default: 1)

Discount factor: How much you value the future

Different from search

Search problems



Definition: search problem

States: the set of states

$s_{\text{start}} \in \text{States}$: starting state

Actions(s): possible actions from state s

$\text{Succ}(s, a)$: where we end up if take action a in state s

Cost(s, a): cost for taking action a in state s

IsEnd(s): whether at end

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

About the goal: In search problems, we want to minimize the cost In Markov Decision Processes, we wanna maximize the reward, similar to real world view 😊

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

What sum up to 1?

The different S' you are going to end up at, when you choose the same action

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 $T(s, a, s') > 0$

Tram Problem Example



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.

If the tram fails, we just gonna wast 2 minutes reward for walk: -1 reward for tram: -2

```

12 def isEnd(self, state):
13     return state == self.N
14 def actions(self, state):
15     # return list of valid actions
16     result = []
17     if state+1<=self.N:
18         result.append('walk')
19     if state*2<=self.N:
20         result.append('tram')
21     return result
22 def succProbReward(self, state, action):
23     # return list of (newState, prob, reward) triples
24     # state = s, action = a, newState = s'
25     # prob = T(s, a, s'), reward = Reward(s, a, s')
26     result = []
27     if action=='walk':
28         result.append((state+1, 1., -1.))
29     elif action=='tram':
30         result.append((state*2, 0.5, -2.))
31         result.append((state, 0.5, -2.))
32     return result
33 def discount(self):
34     return 1.
35 def states(self):
36     return range(1, self.N+1)
37
38 mdp = TransportationMDP(N=10)
39 print(mdp.actions(3))

```

The policy is the solution to MDP problem (but not guaranteed to be the most optimistic solution).

So with a given state, there are many actions we can take, which means many policies, but they are not guaranteed to be optimistic, we have to evaluate those policies and find the best policy via policy evaluation. In summary : Policies are like recipes, and we need to taste-test them (evaluate) to find the most satisfying one!

Policies are action to take!

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
...	...

Find a solution

The solution for search problem was a sequence of paths, but that's when everything is deterministic. But in MDP the way we are defining the solution is using the notion of policy.

The policy is a path/action to take, but it is not guaranteed to be optimistic, we have to evaluate it.

The policy is a function that takes any state and returns an action we can take? $\Pi(s) \Rightarrow \text{Action}$

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
...	...

Policy Evaluation

So ideally I would like to find the best policy to give me the best/right solution But first, we need to tell how good a policy is, and that is the **Policy Evaluation**

A policy from state S is denoted by $\pi(S)$

Utility

Evaluating a policy



Definition: utility

Following a policy yields a **random path**.

The **utility** of a policy is the (discounted) sum of the rewards on the path (this is a random quantity).

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, end]	16
...	...



Definition: value (expected utility)

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

Value: 12

CS221 / Autumn 2019 / Liang & Savitch

15

The utility is just random variables and we cannot optimize on that Utility is a sum of rewards, we are summing them up with discount so Utility a ****discounted sum of rewards ****

utility

$$U: r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

Value $\mathbb{E}[U]$

States
chance nodes

MDP: $T(s, a, s')$
 $R(s, a, s')$

(Discount) $0 \leq \gamma \leq 1$

$\pi(s)$ Policy

I can just sum them up and average them and that gives me value. Yes.

3:06 • Roadmap >

And the value is the expected utility

The value of a policy is the expected utility of that policy And that will be a number

So everytime we run under some policy, we gonnd get different form of utility (becasue of the probability)
But the value won't change Because it is the expectation of all utilities(statictically)

Discounting

The idea of this discount factor is that: I might care about the future differently from how much I care about it now

Discounting



Definition: utility

Path: $s_0, a_1 r_1 s_1, a_2 r_2 s_2, \dots$ (action, reward, new state).

The **utility** with discount γ is

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

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]: $4 + 0 \cdot (4 + \dots) = 4$

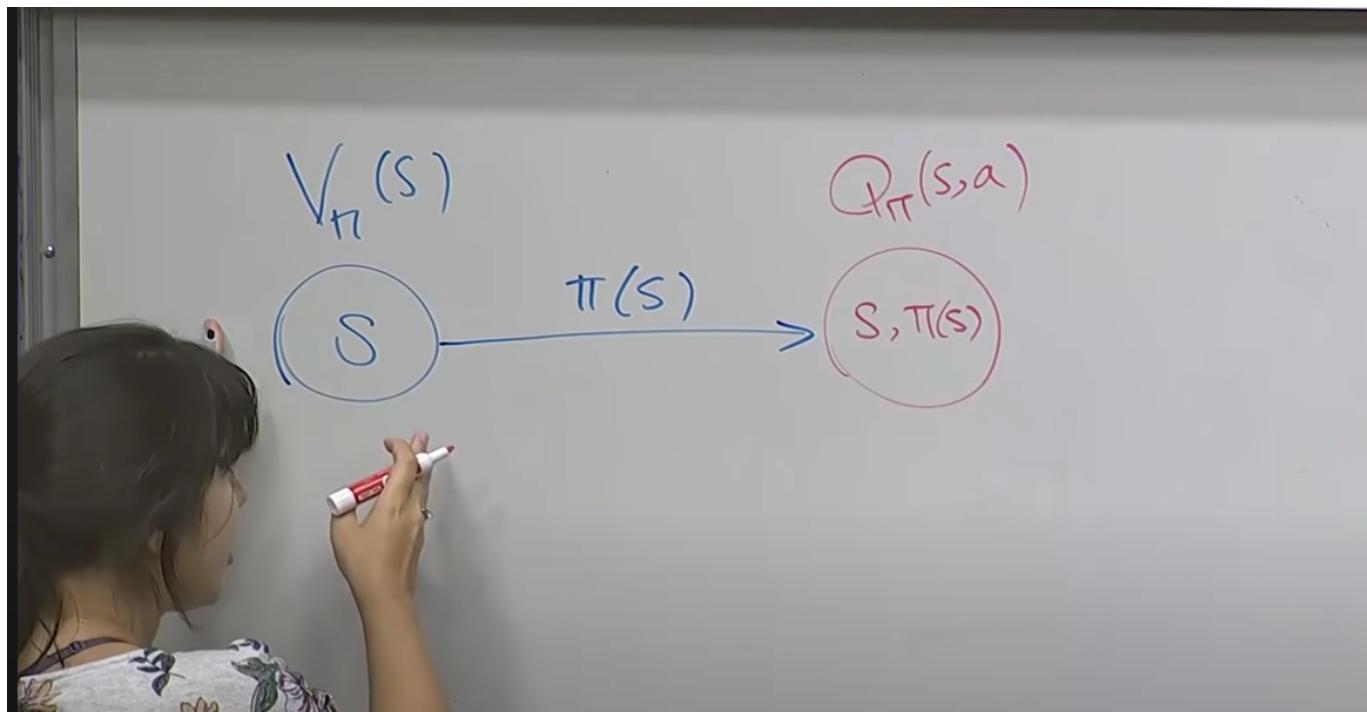
Discount $\gamma = 0.5$ (balanced life):

[stay, stay, stay, stay]: $4 + \frac{1}{2} \cdot 4 + \frac{1}{4} \cdot 4 + \frac{1}{8} \cdot 4 = 7.5$

So if gamma is 1, then the value in the future is the same as it is now IF gamma is 0, then we don't care about the future at all, we live at the moment(greedy) So in reliaty we'll be somewhere in between

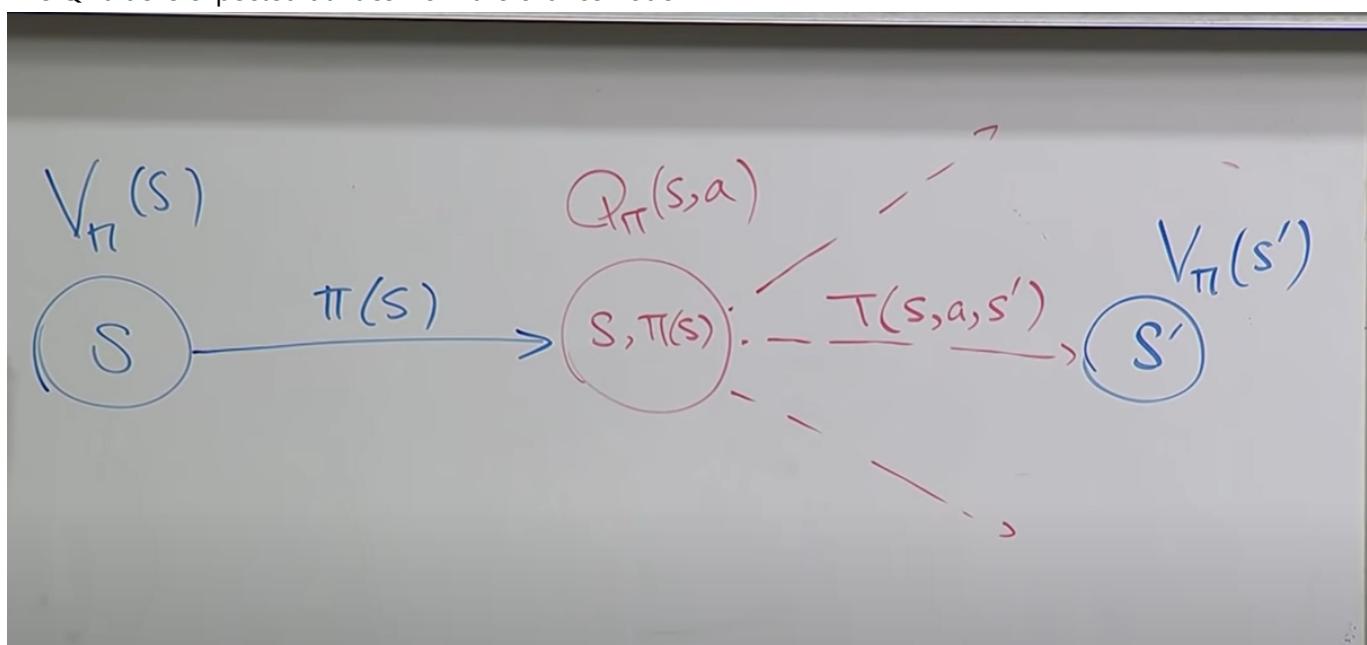
Gamma is not really a hyperparameter that we need to tune, we shall choose the the one that fit for the problem

Policy Evaluation



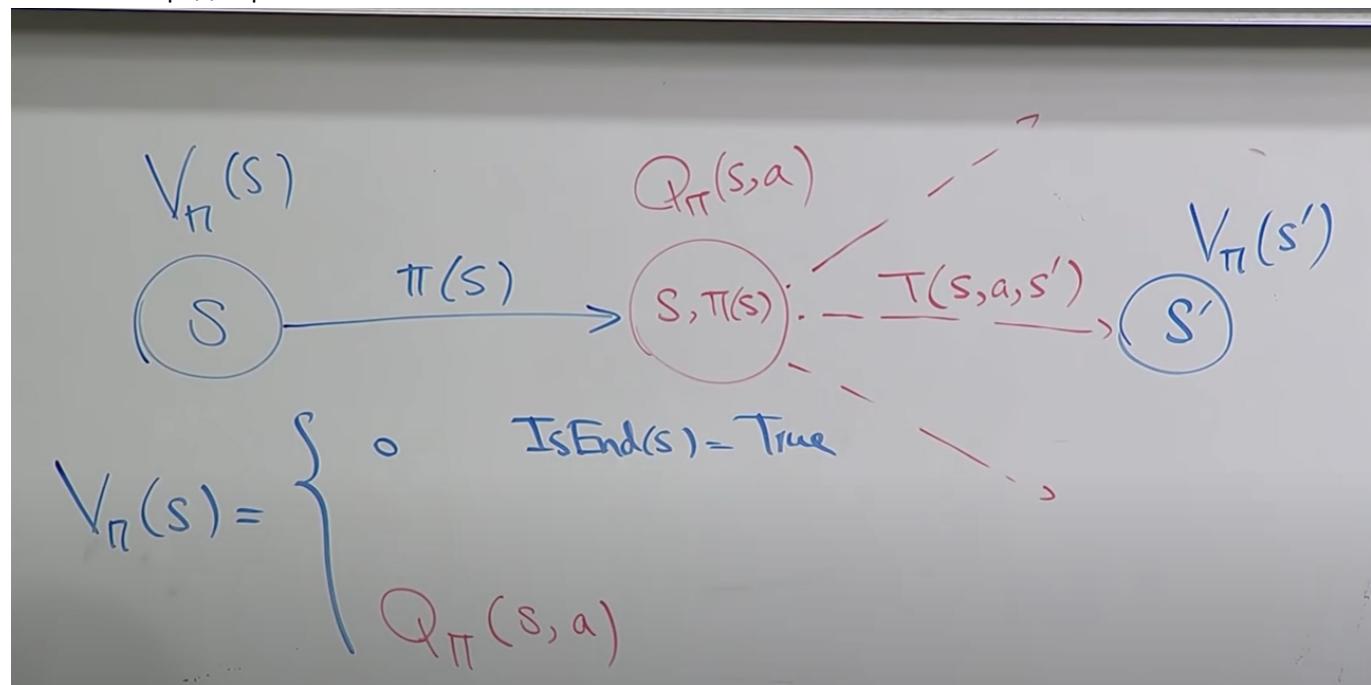
Value is the expected utility from actual state S , IF we take policy $\pi(s)$. When we take policy $\pi(s)$, we will end up at a chance node $Q_\pi(S, a)$, where the state is in between and the probability is gonna come to play and decide the actual state you are gonna reach

The Q value is expected utilities from the chance node

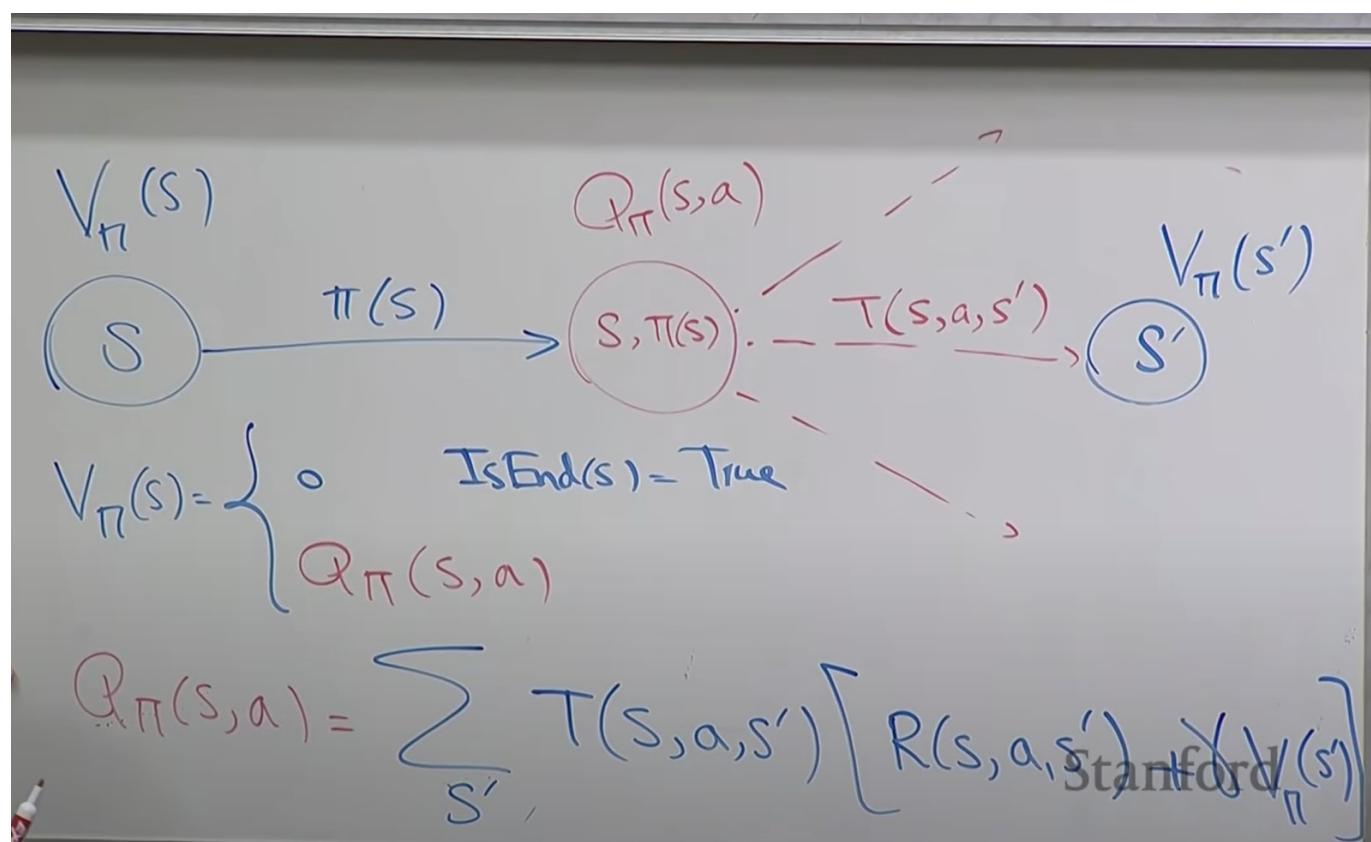


After the nature/probility is played, you are in a new state S' , the value of the new state is $V_\pi(S')$

So, what is $V_{\pi}(s)$ equal to



- Gonna be 0 if I am in a end state
- OR Q_{π} because i already take action a



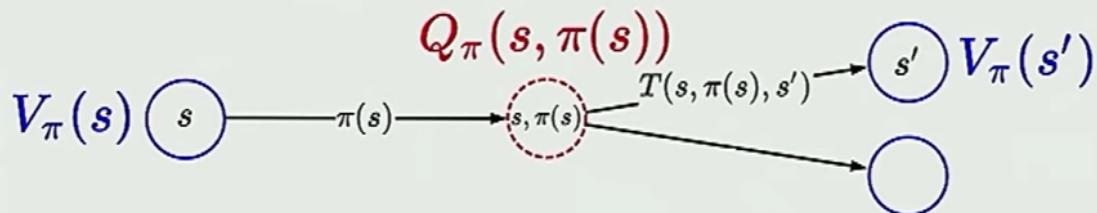
$Q_{\pi}(s, a)$ is the sum of probability of state s' that we could end up at * the utility of this state

Transition(probility) * [immediate reward of taking this action to this state + discounted value(value of new state/future)]

$Q_{\pi}(S, a)$ is the sum of expectations where all the places that I can end up at

Policy evaluation

Plan: define recurrences relating value and Q-value

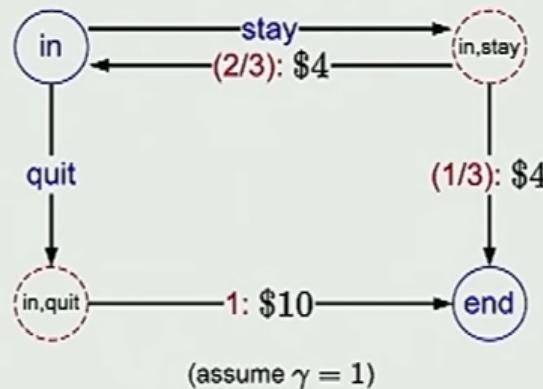


$$V_\pi(s) = \begin{cases} 0 & \text{if } \text{IsEnd}(s) \\ Q_\pi(s, \pi(s)) & \text{otherwise.} \end{cases}$$

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

Let's say somebody comes in and tell me the policy is to stay

Dice game



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

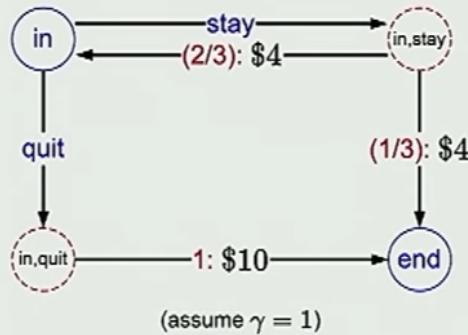
$$V_\pi(\text{end}) = 0$$

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

In this case, can solve in closed form:

$$V_\pi(\text{in}) = \frac{1}{3} 4 + \frac{2}{3} (4 + V_\pi(\text{in}))$$

Dice game



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

$$V_\pi(\text{end}) = 0$$

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

In this case, can solve in closed form:

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

Itratively find the value

In the previous example we just solve the $V_{\pi}(S)$ from the equation (By canceling the same state) However in the reality there can be much more and more states, and we cannot just solve from equation

In this case we can iteratively try to find the value $V_{\pi}(S)$

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 $t = 1, \dots, t_{PE}$:

For each state s :

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

We initialize the values to some numbers close to 0 And we iterate some number of times For every state, we will update the value with that formula, the same equation with the value of the previous step/iteration ?

This allows us to compute new values based on the previous values that I've had I start with close to 0, and update values of all states and keep going iterative updating, and converged to true value

Policy evaluation computation

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

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	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	

How long we should run this?

We can keep track of the difference between the value of previous time step and this time step If that is below some threshold, then we can call it done and believe the right value has been found

Policy evaluation implementation

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

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

We are looking for the diff between iteration t and t-1 and taking max for all possible states, because i want the value to be close for all states

Policy evaluation implementation

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

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

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

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

Only store the last two columns of the table to reduce space complexity

Complexity

Complexity



Algorithm: policy evaluation

Initialize $V_{\pi}^{(0)}(s) \leftarrow 0$ for all states s .

For iteration $t = 1, \dots, t_{PE}$:

For each state s :

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

MDP complexity

S states

A actions per state

S' successors (number of s' with $T(s, a, s') > 0$)

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

Worst case: $S' = S$

Value Iteration

To find the right policy

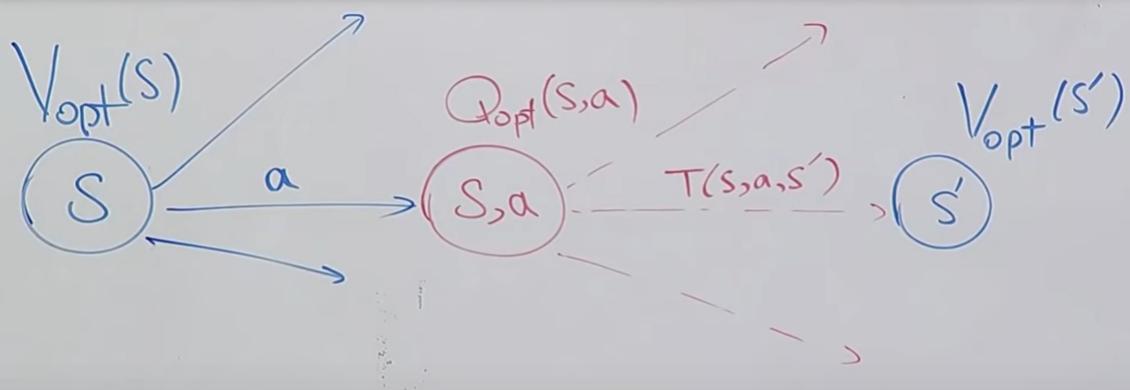
Optimal value and policy

Goal: try to get directly at maximum expected utility



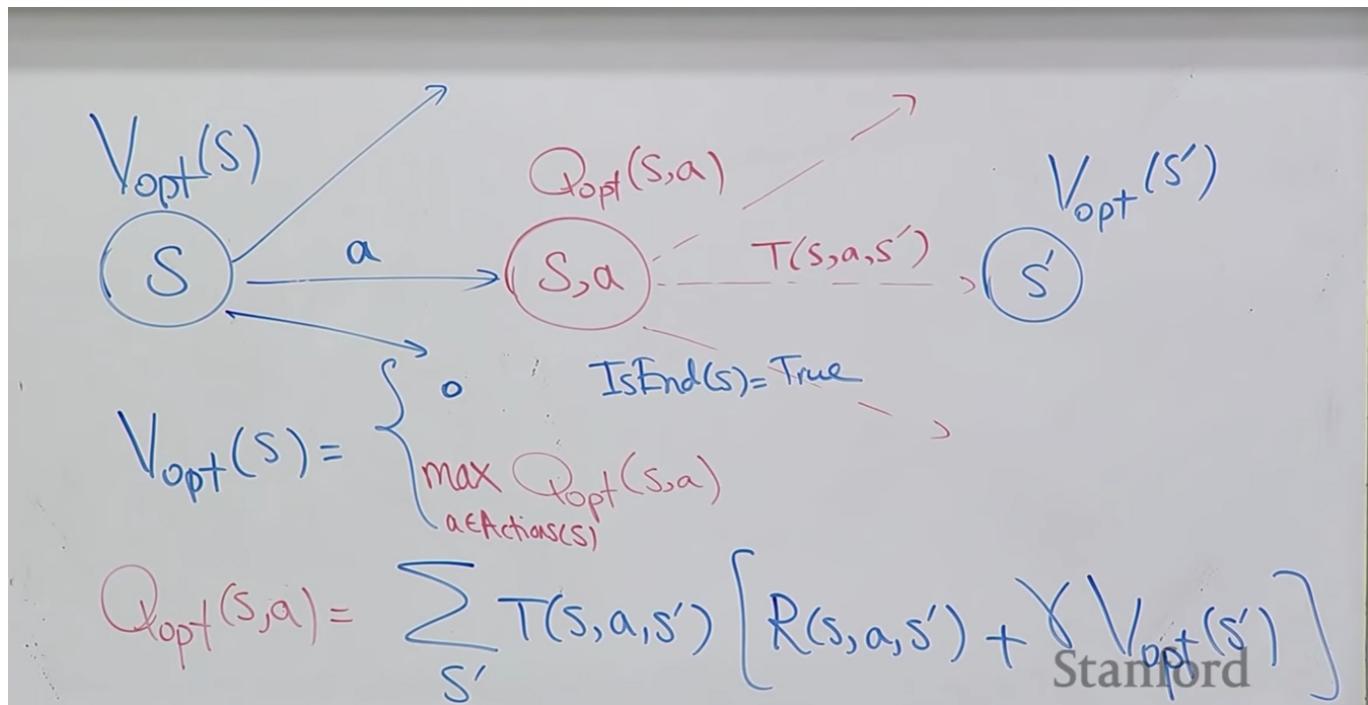
Definition: optimal value

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

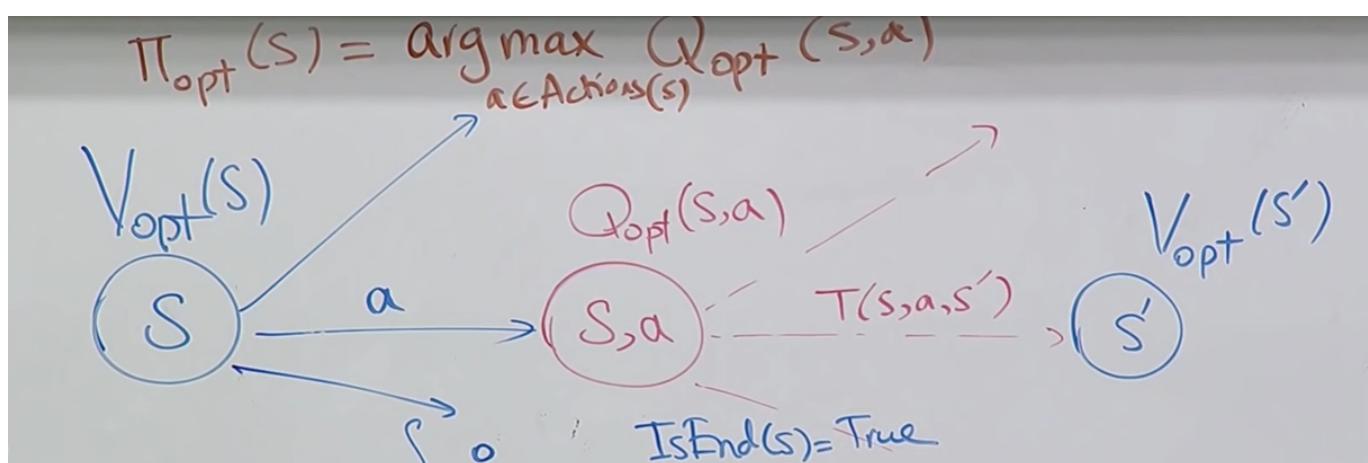


$$Q_{\text{opt}}(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V_{\text{opt}}(s') \right]$$

Stanford



Just pick the maximum of Q_{opt}

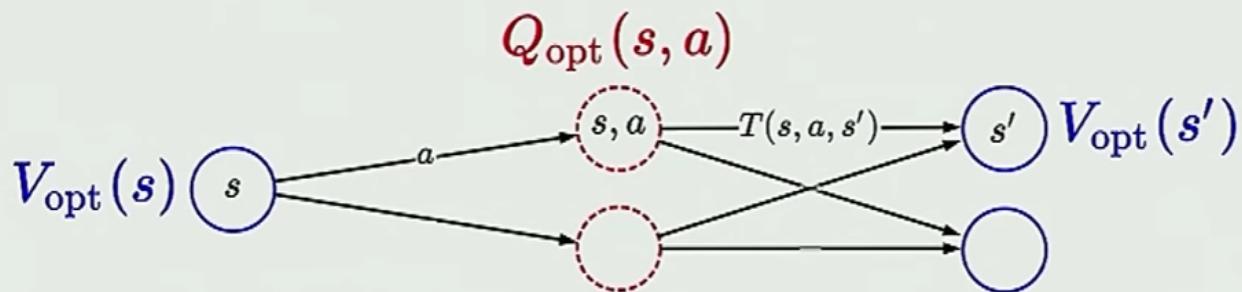


what is Argmax ?

maxf(x) is the maximum value (if it exists) of f(x) as x varies through some domain, while argmaxf(x) is the value of x at which this maximum is attained(达到).

However, there could be more than one x value which gives rise to the maximum f(x) , in which case argmaxf(x) would be this set of values of x instead. 简而言之，最大值和达到最大值时的自变量x值，自变量x可能不唯一

Optimal values and Q-values



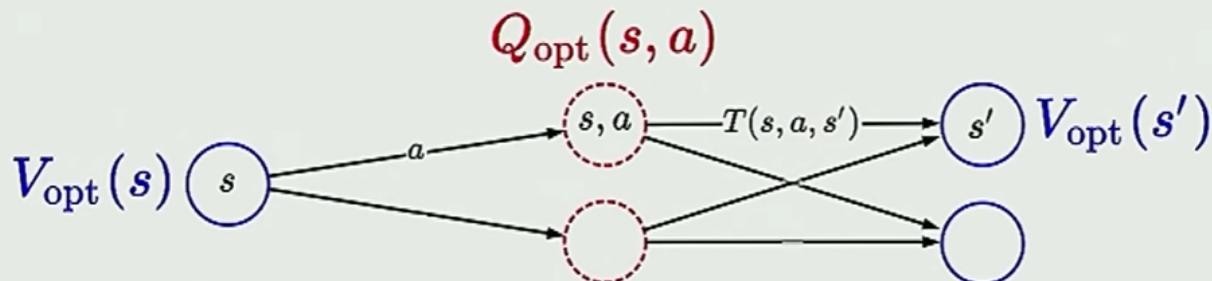
Optimal value if take action a in state s :

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

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:

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

Complexity

Value iteration



Algorithm: value iteration [Bellman, 1957]

Initialize $V_{\text{opt}}^{(0)}(s) \leftarrow 0$ for all states s .

For iteration $t = 1, \dots, t_{\text{VI}}$:

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

Convergence

Convergence



Theorem: convergence

Suppose either

- discount $\gamma < 1$, or
- MDP graph is acyclic.

Then value iteration converges to the correct answer.



Example: non-convergence

discount $\gamma = 1$, zero rewards



If there's no cycles, ur just doing dynamic programming, it's gonna converge, ur gonna find the optimistic one

However if u have cycles, you should make ur discount rate < 1 , otherwise it will never converge (Cuz you will just keep going between cycles)



Summary

- **Markov decision processes** (MDPs) cope with uncertainty
- Solutions are **policies** rather than paths
- **Policy evaluation** computes policy value (expected utility)
- **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

Code

```
from enum import Enum
import os
import sys
from queue import PriorityQueue

sys.setrecursionlimit(100000)

Destination = 10

### Model (Search Problem)
class TransportationProblemMDP(object):

    WALK_REWARD = -1
    TRAM_REWARD = -2

    WALK = "walk"
    TRAM = "tram"

    TRAM_FAIL_PROB = 0.5
    def __init__(self, destination):
        # N number of blocks
        self.destination = destination

    def startState(self) -> int:
        return 1
```

```

def isEnd(self, state):
    return state == self.destination

def actions(self, state):
    ...
    Return a list of valid actions that we can take from the current state
    ...
    result = []
    if(state + 1 <= self.destination):
        result.append(self.WALK)

    if(state * 2 <= self.destination):
        result.append(self.TRAM)

    return result

def succProbAndReward(self, state : int, action: str):
    ...
    Input:
        current state
        Action that we gonna take
    Return a list of (newState, prob, rewards) triples
    Meaning return the a list of:
        the new state we are gonna endup at,
        what probability of it,
        and the reward we get from it

    state = s, action = a, newState = s'
    prob = T(s, a, s'), reward = Reward(s, a, s')
    ...
    result = []
    if(action == self.WALK):
        # It is deterministic, so the prob is 1.
        result.append((state + 1, 1., self.WALK_REWARD))
    elif(action == self.TRAM):
        # There's a 50% chance that the tram could fail
        result.append((state, self.TRAM_FAIL_PROB, self.TRAM_REWARD))
        result.append((state * 2, 1. - self.TRAM_FAIL_PROB, self.TRAM_REWARD))

    return result

def discountFactor(self):
    return 1.

def states(self):
    return range(1, self.destination+1)

# Inference (Algorithms)
def valueIteration(mdp : TransportationProblemMDP):
    # Initialize
    valueDic = {} # state => Vopt[state]
    # Initialize all optimal value with 0
    for state in mdp.states():

```

```
valueDic[state] = 0.

def VQ(state, action):
    return sum(
        transProb * (reward + mdp.discountFactor() * valueDic[newState])
        for newState, transProb, reward in mdp.succProbAndReward(state,
action)
    )

def isConverged(valueDic, newValueDic):
    ...
    We want the maximum value difference of all states to be smaller than the
treashold
    ...
    return max(abs(valueDic[state] - newValueDic[state])) for state in
mdp.states() < 1e-10

while True:
    # Compute the new values (new V) with old values (V)
    newValueDic = {}

    for state in mdp.states():
        if mdp.isEnd(state):
            newValueDic[state] = 0
            break

        newValueDic[state] = max(
            VQ(state, action)
            for action in mdp.actions(state)
        )

    # Check for convergence
    if (isConverged(valueDic, newValueDic)):
        break

    valueDic = newValueDic

    # Read out policy:
    pi = {}
    for state in mdp.states():
        if mdp.isEnd(state):
            pi[state] = "none"
        else:
            # Otherwise it's gonna be the argmax of Q values
            # Which is the input that can maximize Q values
            # And how we gonna find the argmax?
            # - We just gonna try out different actions and see
            pi[state] = max(
                (VQ(state, action), action) for action in mdp.actions(state)
            )[1]

    # Print stuff out
    os.system('clear')
```

```
    print('{:25} {:25} {:25}'.format('state', 'valueDic[state]',  
'pi[state]'))  
  
    for state in mdp.states():  
        print('{:5} {:15} {:55}'.format(state, valueDic[state], pi[state]))  
    input()  
  
mdp = TransportationProblemMDP(destination=10)  
# print(mdp.actions(3))  
# print(mdp.succProbAndReward(3, "tram"))  
valueIteration(mdp)
```