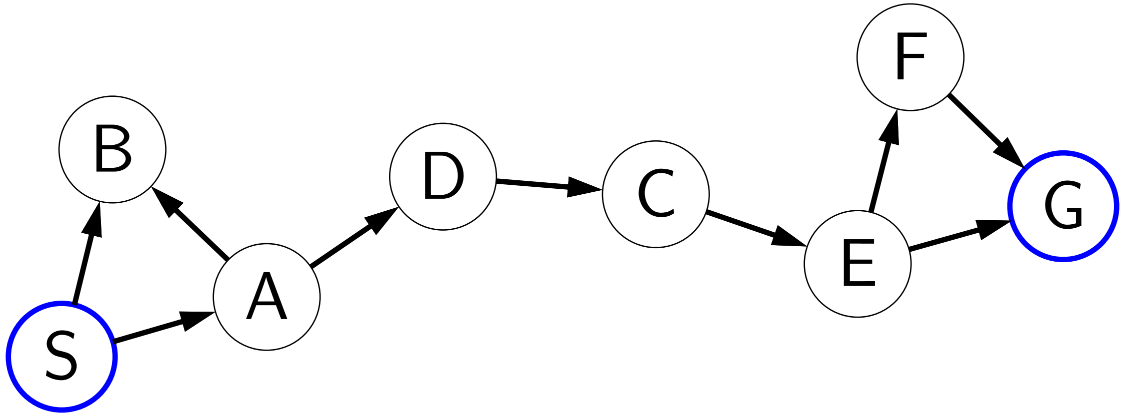
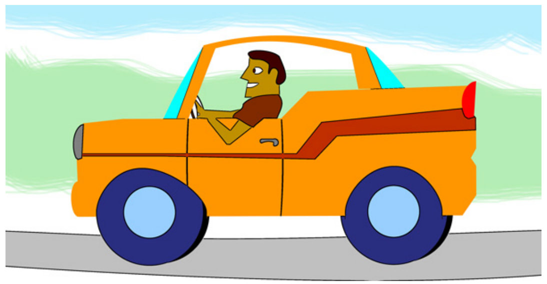
MDPs: overview

So far: search problems



deterministic

state s, action a $\longrightarrow$ state $\text{Succ}(s, a)$



Uncertainty in the real world

state s'_1

state s, action a **random**

state s'_2

History

- MDPs: Mathematical Model for decision making under uncertainty.

- MDPs were first introduces in 1950s-60s.

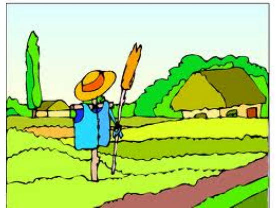
- Ronald Howard's book on Dynamic Programming and Markov Processes

- The term 'Markov' refers to Andrey Makov as MDPs are extensions of Markov Chains, and they allow making decisions (taking actions or having choice).

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

Dice game

Example: dice game

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

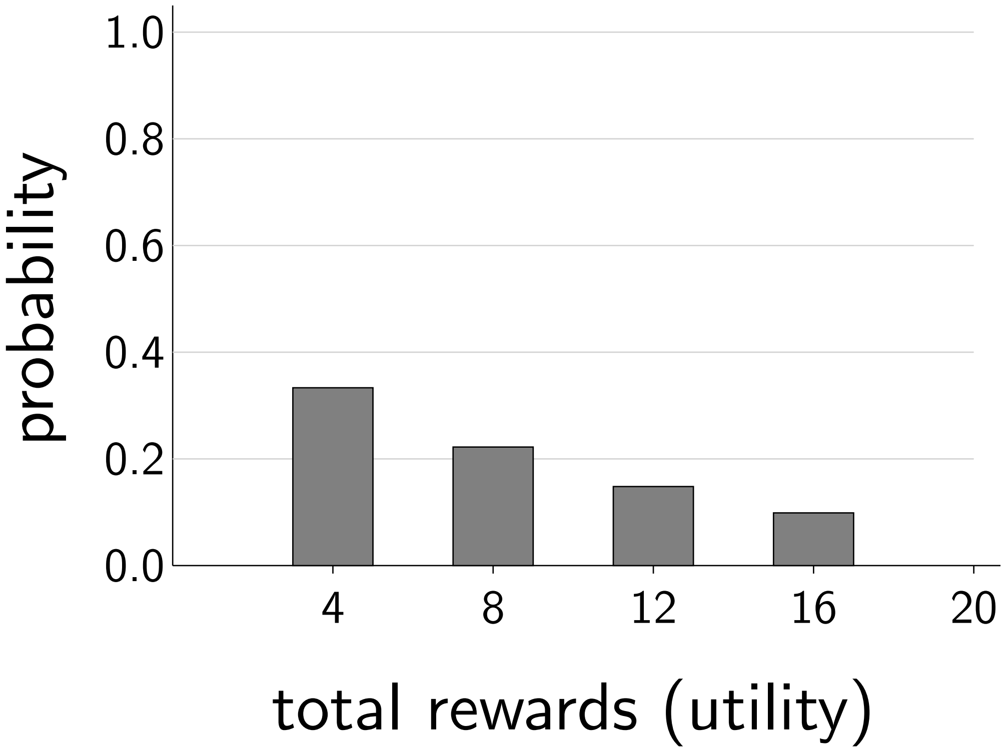
- You choose stay or quit.
- If quit, you get $\$10$ and we end the game.
- If stay, you get $\$4$ and then I roll a 6-sided dice.
  - If the dice results in 1 or 2, we end the game.
  - Otherwise, continue to the next round.

Start Stay Quit

Dice: Rewards: 0

Rewards

If follow policy "stay":

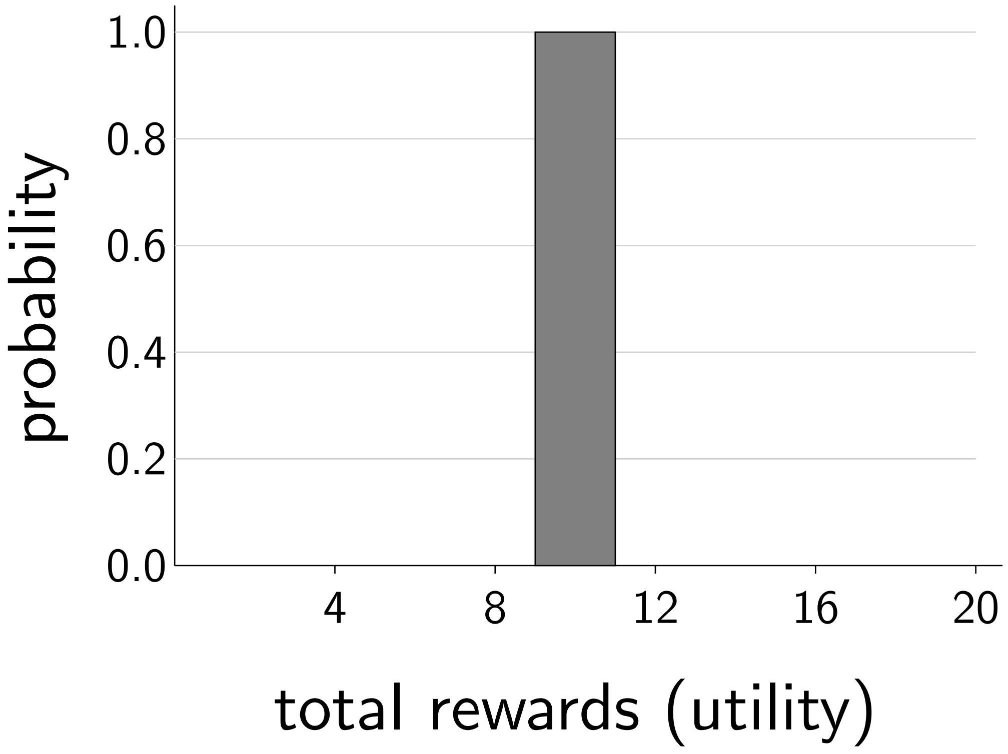


Expected utility:

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

Rewards

If follow policy "quit":



Expected utility:

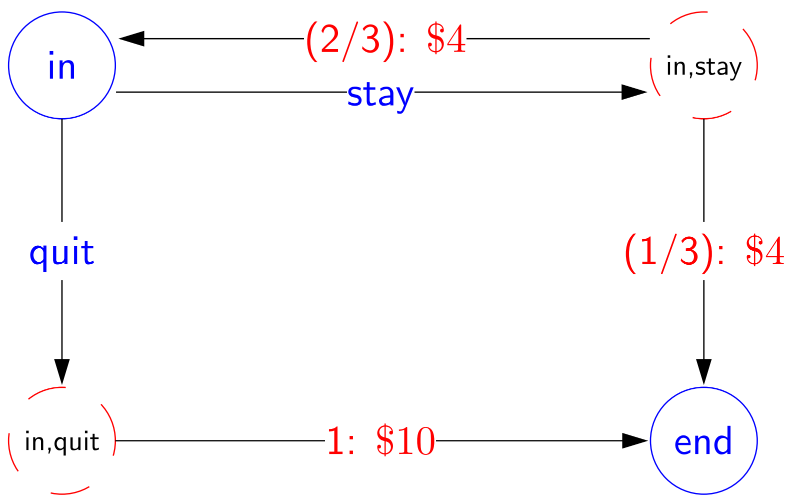
$$1(10) = 10$$

MDP for dice game

Example: dice game

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

- 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

$s_{\text{start}} \in$ 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)

Search problems

Definition: search problem

States: the set of states

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

Actions(s): possible actions from state s

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

- Succ$(s, a) \Rightarrow T(s, a, s')$
- Cost$(s, a) \Rightarrow$ 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.

[semi-live solution]

What is a solution?

Search problem: path (sequence of actions)

MDP:

Definition: policy

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

Example: volcano crossing

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

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 variable).

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 at a state is the **expected** utility.

Value: 12

Evaluating a policy: volcano crossing

Run (or press ctrl-enter)

2.4	-0.5	**-50**	**40**
3.7	5	**-50**	31
2	12.6	16.3	26.2

a	r	s
		(2,1)
E	-0.1	(2,2)
S	-0.1	(3,2)
E	-0.1	(3,3)
E	-0.1	(3,4)
N	-0.1	(2,4)
N	-50.1	(2,3)

Value: 3.73

Utility: -29.99

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

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

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

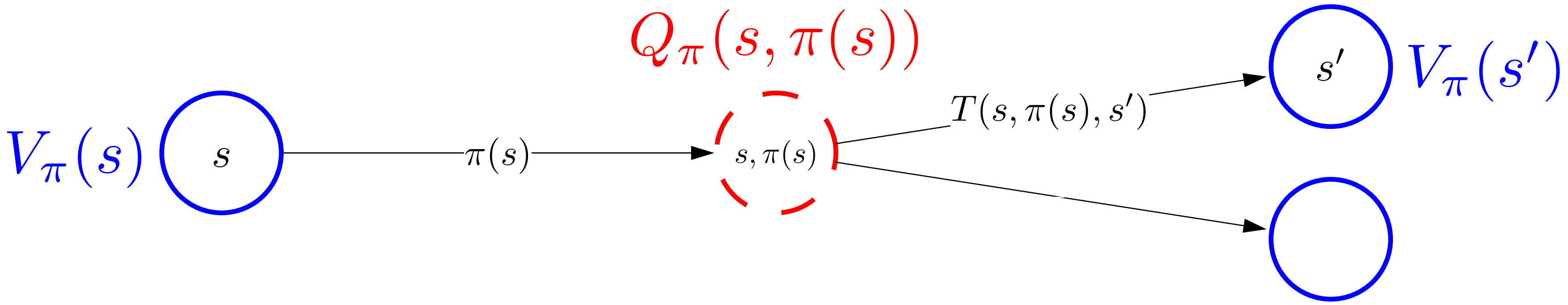
Policy evaluation

Definition: value of a policy

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

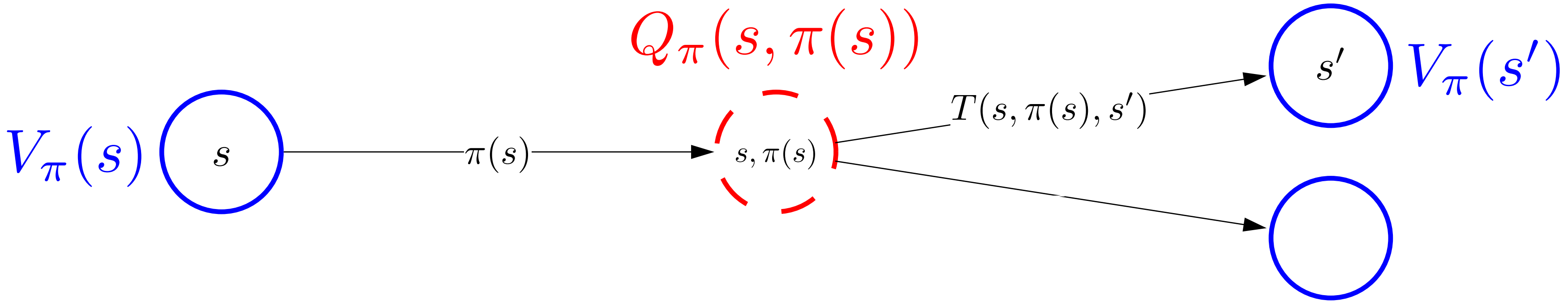
Definition: Q-value of a policy

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



Policy evaluation

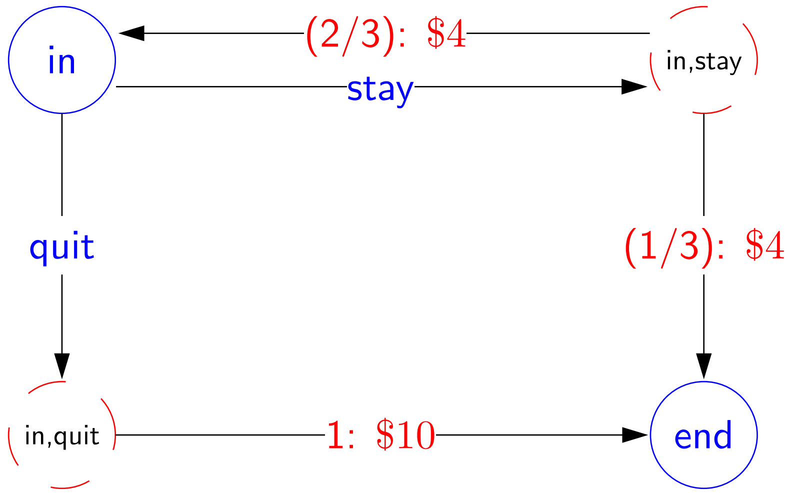
Plan: define recurrences relating value and Q-value



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

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

Dice game



(assume $\gamma = 1$)

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:

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

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

For each state s:

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

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:

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

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:

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

MDP complexity

S states

A actions per state

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

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

Policy evaluation on dice game

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

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

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

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

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

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

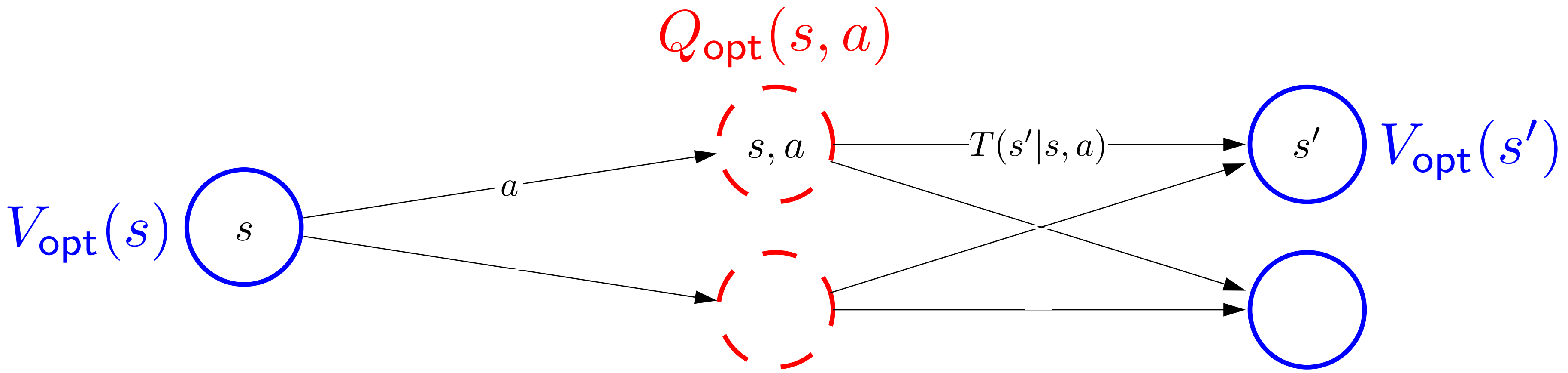
Optimal value and policy

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



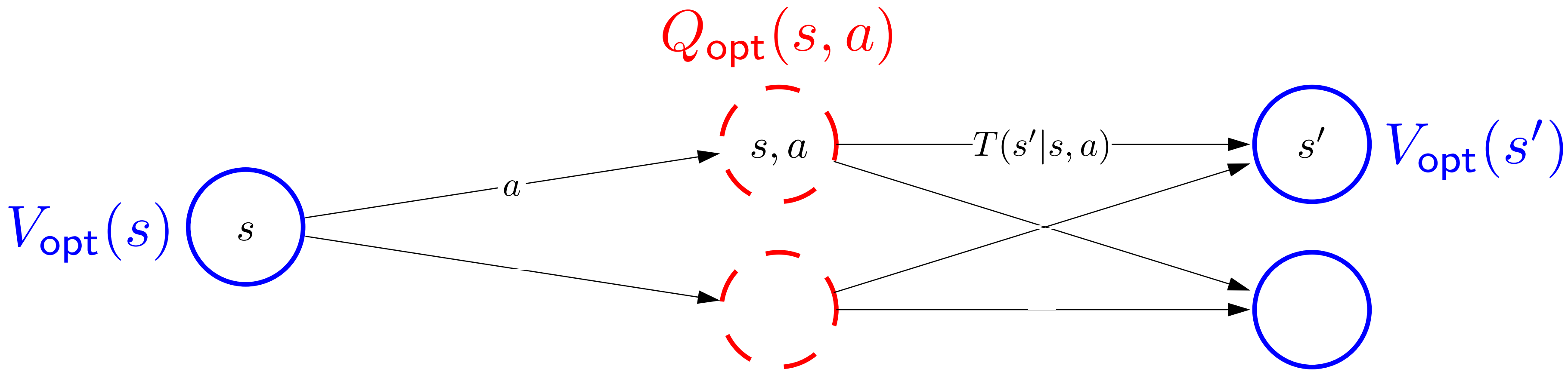
Optimal value if take action a in state s:

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

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

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

[semi-live solution]

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

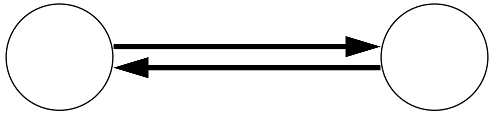
Suppose either

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

Then value iteration converges to the correct answer.

Example: non-convergence

discount $\gamma = 1$, zero rewards



Summary of algorithms

- **Policy evaluation**: (MDP, π) $\rightarrow V_\pi$

- **Value iteration**: MDP $\rightarrow (Q_{\text{opt}}, \pi_{\text{opt}})$

Unifying idea

Algorithms:

- Search DP computes 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., $V_\pi(s) = \cdots V_\pi(s') \cdots$)
- Turn into iterative algorithm (replace mathematical equality with assignment operator)