Lecture 2
Markov Decision Processes

Outline

- Markov Processes
- Markov Reward Processes
- Markov Decision Processes
- Extensions to MDPs

Markov Processes

Markov Property

"The future is independent of the past given the present"

Definition

A state S_t is *Markov* if and only if

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

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

Markov Process/Markov Chain

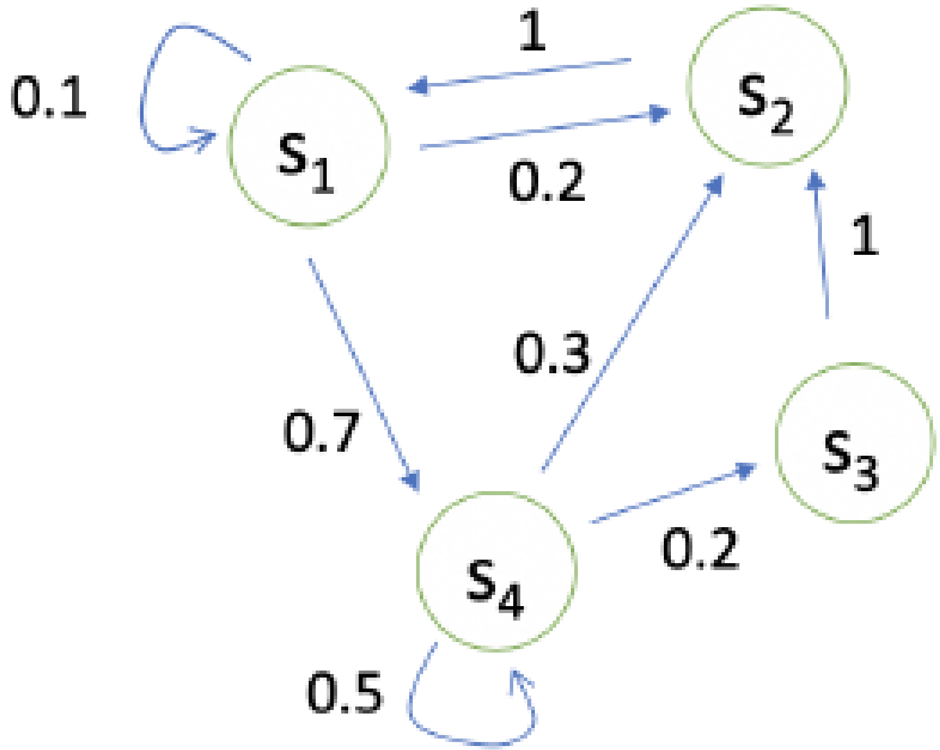
- A Markov process is a memoryless random process.
- For example, a sequence of random states $s_1, s_2, \ldots$ with the Markov property.

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

Example: Markov Chain



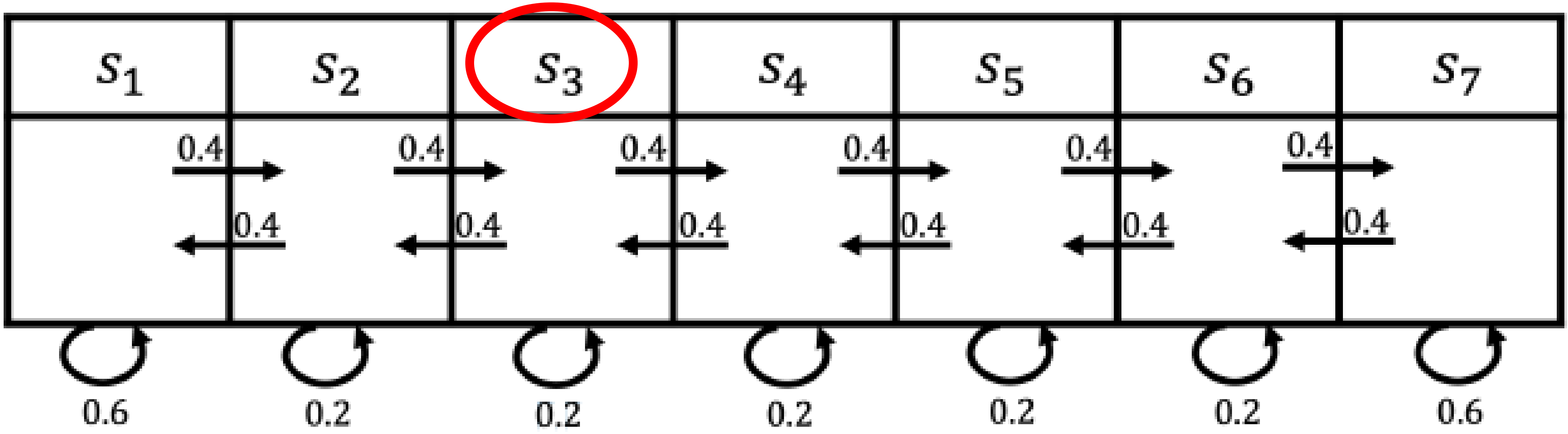
State transition matrix P specifies:

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

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

each row of the matrix sums to 1

Example of Markov Process



Sample episodes starting from $\boldsymbol{s_3}$

- E1: s_3, s_4, s_5, s_4, s_3
- E2: s_3, s_2, s_3, s_2, s_1
- E3: s_3, s_4, s_4, s_5, s_5

Markov Reward Processes

Markov Reward Process(MRP)

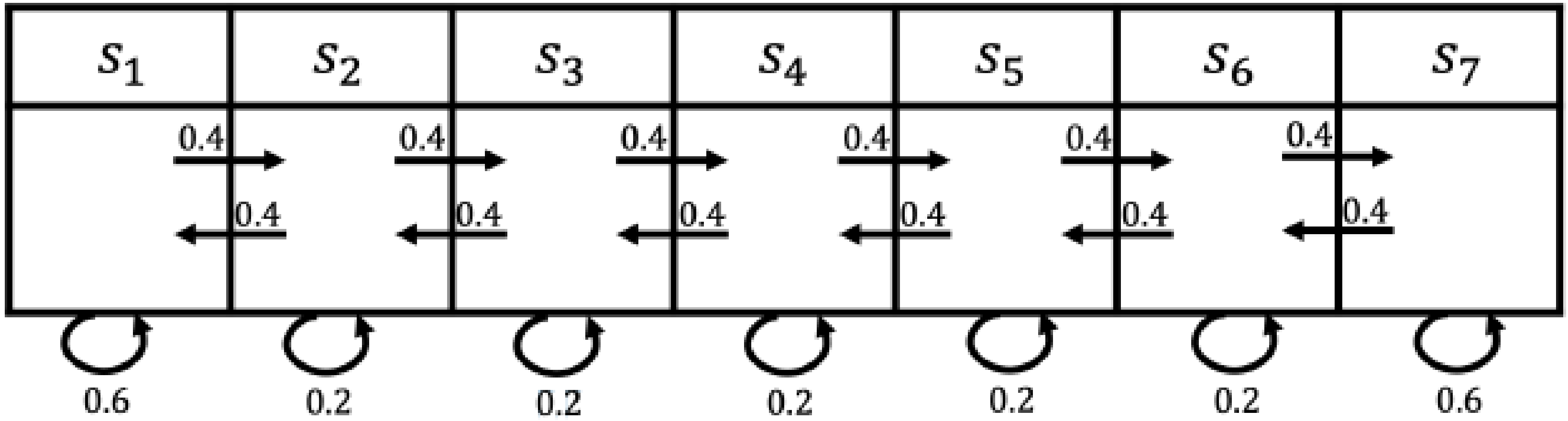
- A Markov reward process is a Markov chain with values.

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, $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Example of MRP



Reward: +5 in $\boldsymbol{s_1}$, +10 in $\boldsymbol{s_7}$, 0 in all other states.

So that we can represent R = [5, 0, 0, 0, 0, 0, 10]

Return

Definition

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

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

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

Why Discount Factor γ

- Mathematically convenient to discount rewards
- Avoids **infinite returns** in cyclic Markov processes
- **Uncertainty about the future** may not be fully represented
- If the reward is financial, **immediate rewards** may earn more interest than delayed rewards
- Animal/human behaviour shows **preference for immediate reward**
- It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences **terminate**.

Value Function

- The value function $v(s)$ gives the long-term value of state s

Definition

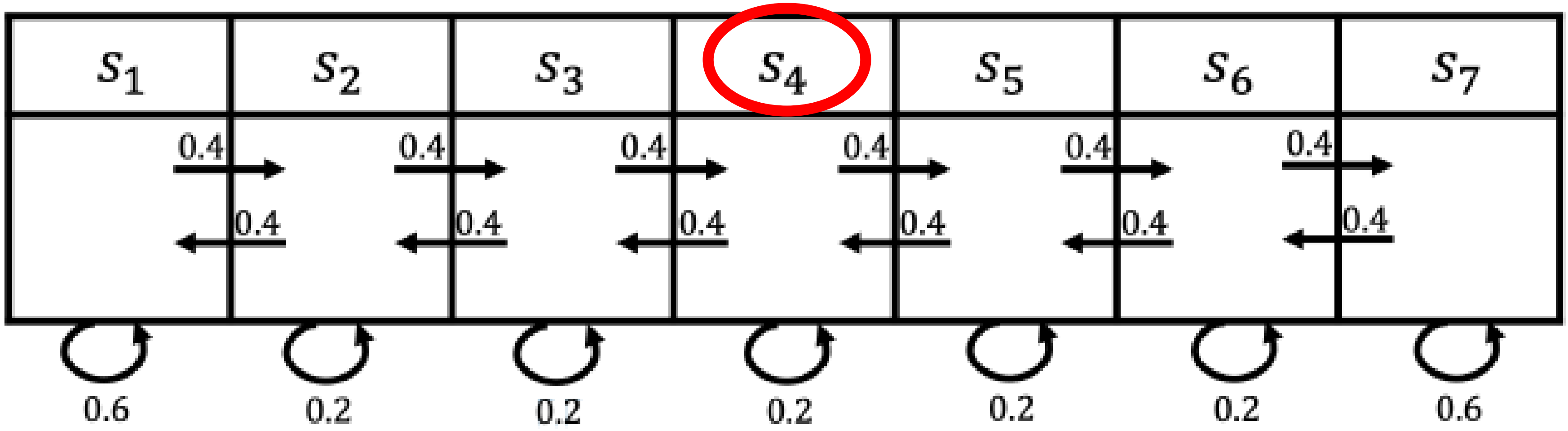
The *state value function* $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + .. | s_t = s]$$

Example of MRP

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



Reward: +5 in s_1, +10 in s_7, 0 in all other states. R = [5, 0, 0, 0, 0, 0, 10]

Sample returns G for a 4-step episodes with $\gamma = 1/2$

- return for $s_4, s_5, s_6, s_7 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 1.25$
- return for $s_4, s_3, s_2, s_1 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 0.625$
- return for $s_4, s_5, s_6, s_6 : 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 0 = 0$

Bellman Equation for MRPs

- The value function can be decomposed into two parts:
  - Immediate reward R_{t+1}
  - Discounted value of successor state $\gamma V(S_{t+1})$

$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t \mid S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + .. \mid s_t = \mathrm{s}] \\
&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ..) \mid s_t = \mathrm{s}] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid s_t = \mathrm{s}] \\
&= \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) \mid s_t = \mathrm{s}]
\end{aligned}
$$

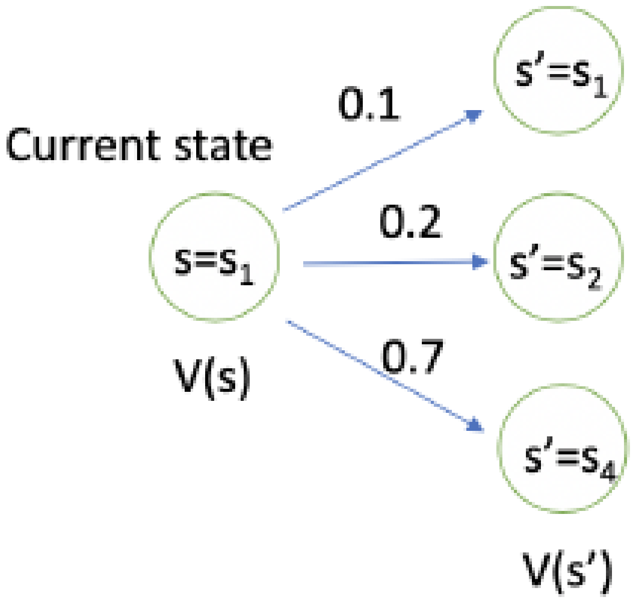
Understanding Bellman Equation

$V(s) = \mathbb{E}[R_{t+1} + \gamma V(s_{t+1}) \mid s_t = s]$

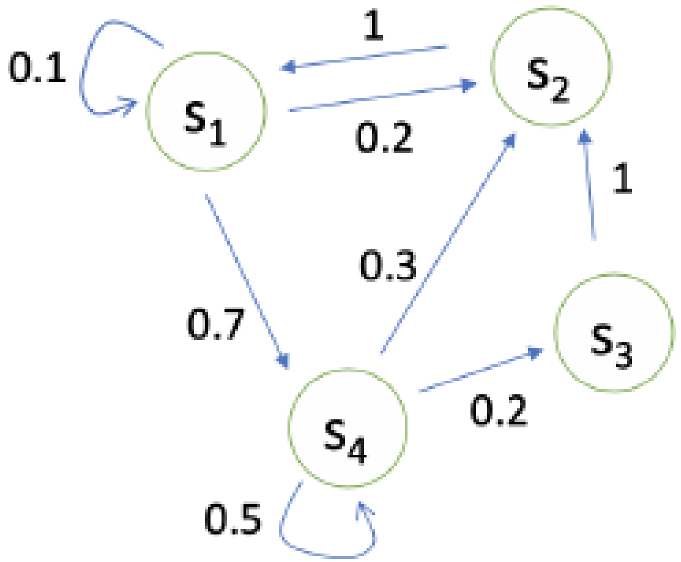
- Bellman equation describes the iterative relations of states

$$V(s) = R(s) + \gamma \sum_{s' \in S} P_{ss'} V(S')$$

Possible next state



Markov Transition matrix



Matrix Form of Bellman Equation for MRP

- Therefore, we can express ***V***(***s***) using the matrix form:

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

$$\boldsymbol{V} = \boldsymbol{R} + \gamma \boldsymbol{P} \boldsymbol{V}$$

Solving the Bellman Equation

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

$$\boldsymbol{V} = \boldsymbol{R} + \gamma \boldsymbol{P}\boldsymbol{V}$$
$$(1 - \gamma P)\boldsymbol{V} = \boldsymbol{R}$$
$$\boldsymbol{V} = (1 - \gamma \boldsymbol{P})^{-1}\,\boldsymbol{R}$$

- Matrix inverse takes the complexity O(N^3) for *N* states
- Only possible for a small MRPs
- Iterative methods for large MRPs:
  - Dynamic Programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Monte Carlo Algorithm for Computing Value of a MRP

Algorithm 1 Monte Carlo simulation to calculate MRP value function

$i \leftarrow 0, G_t \leftarrow 0$
while $i \neq N$ **do**
    generate an episode, starting from state s and time t
    Using the generated episode, calculate return $g = \sum_{i=t}^{H-1} \gamma^{i-t} r_i$
    $G_t \leftarrow G_t + g, i \leftarrow i + 1$
end while
$V_t(s) \leftarrow G_t / N$

For example: to calculate $V(s_4)$ we can generate a lot of trajectories then take the average of the returns:

- Return for $s_4, s_5, s_6, s_7: 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 1.25$
- Return for $s_4, s_3, s_2, s_1: 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 10 = 0.625$
- Return for $s_4, s_5, s_6, s_6: 0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 0 + \frac{1}{8} \times 0 = 0$

Iterative Algorithm for Computing Value of a MRP

Algorithm 2 Iterative algorithm to calculate MRP value function

1: for all states $s \in S, V'(s) \leftarrow 0, V(s) \leftarrow \infty$
2: **while** $||V - V'|| > \epsilon$ **do**
3: $\quad V \leftarrow V'$
4: $\quad$ For all states $s \in S, V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s) V(s')$
5: **end while**
6: return $V'(s)$ for all $s \in S$

Markov Decision Process

Markov Decision Process(MDP)

- A Markov decision process (MDP) is a Markov reward process with **decisions**. It is an environment in which *all states are Markov*.

Definition

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,
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Policy in MDP

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- Policy specifies what action to take in each state
- Give a state, specify a distribution over actions
- Policies are stationary (time-independent), $A_t \sim \pi(\cdot\,|s)$ for any t > 0

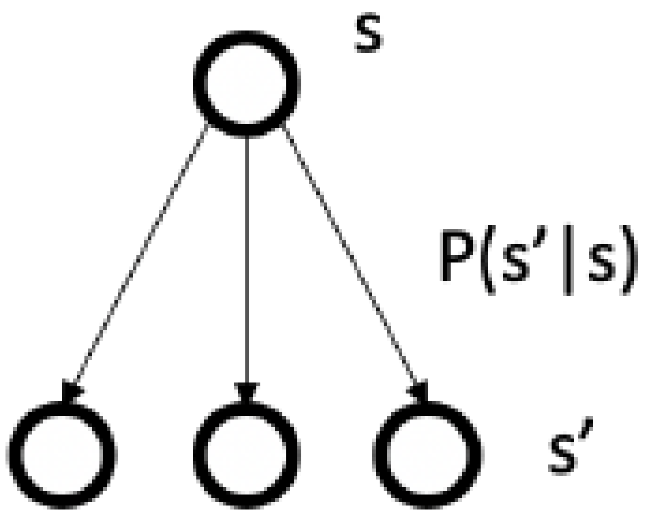
Policy in MDP

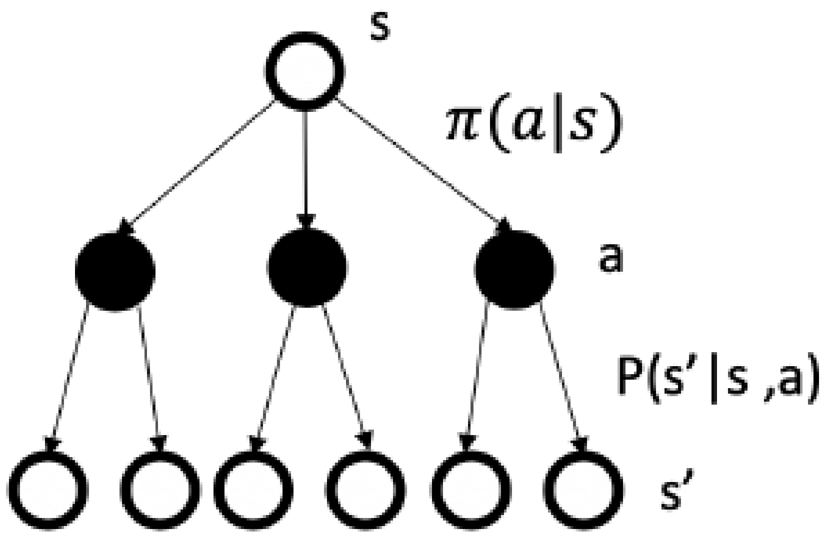
- Given an MDP $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$ and a policy π
- The state sequence $S_1, S_2, \ldots$is a Markov process $\langle S, P^{\pi} \rangle$ The state and reward sequence $S_1, R_2, S_2, R_2, \ldots$ is a Markov reward process $\langle S, P^{\pi}, R^{\pi}, \gamma \rangle$
- Where

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

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

Comparison of MP/MRP and MDP





Value Function in MDP

Definition

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

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

Definition

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

$$q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

Bellman **Expectation Equation**

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

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid s_t = \mathrm{s}]$$

- The action-value function can similarly be decomposed

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid s_t = \mathrm{s}, A_t = a]$$

- We have the relation between $V_\pi(s)$ and $q_\pi(s, a)$

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a)$$

Bellman Expectation Equation for V_π and Q_π

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

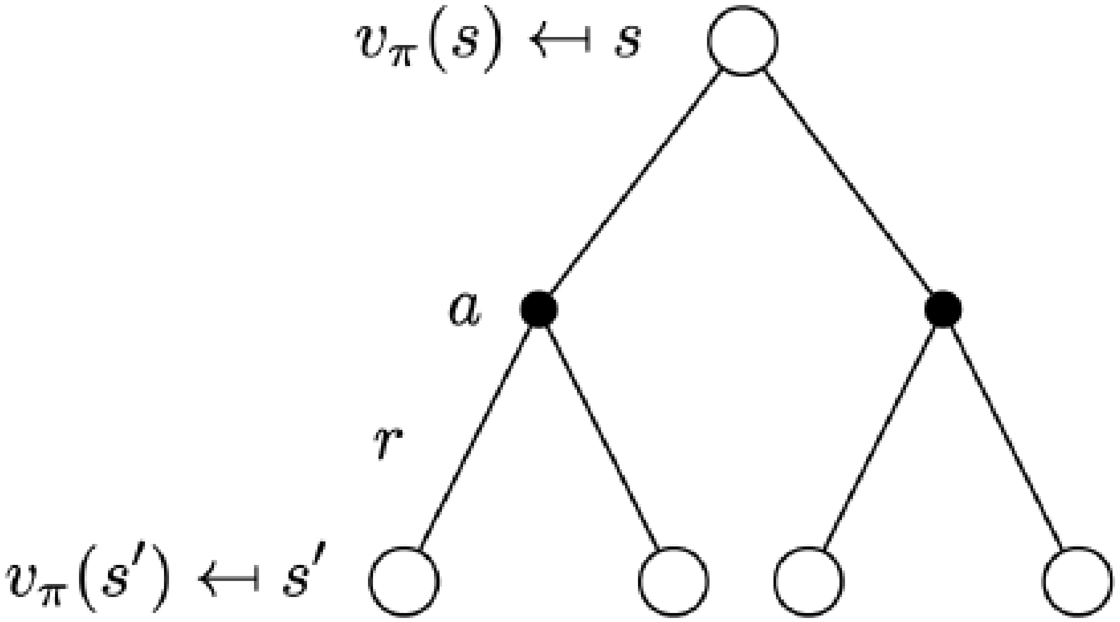
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) V_\pi(s')$$

Thus:

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

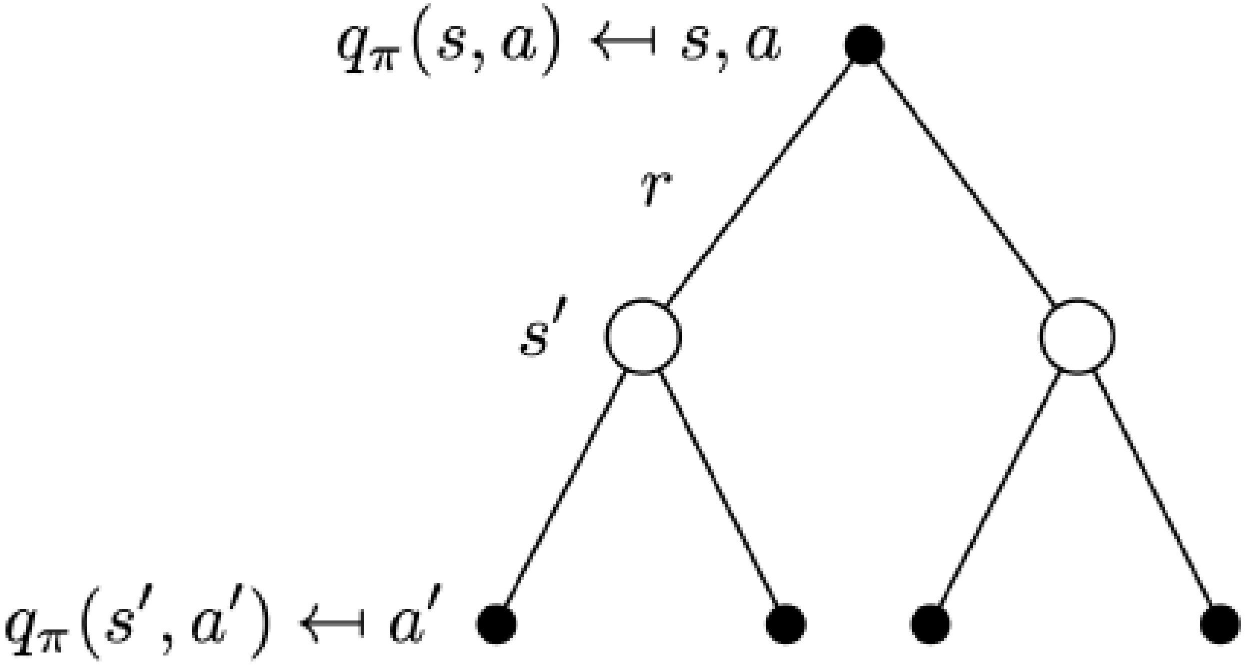
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a' \in A} \pi(a'|s') q_\pi(s',a')$$

Backup Diagram for V_π



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

Backup Diagram for Q_π



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

Policy **Evaluation** in MDP

- Evaluate the value of state given a policy π: compute $V_\pi(s)$
- Also called as (value) prediction

Example: Navigate the boat

随波逐流
浑浑噩噩

Figure: Markov Chain/MRP: Go with river stream

随机应变
优化决策

Figure: MDP: Navigate the boat

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

Actions: *Left* and *Right*
Reward: +5 in s_1, +10 in s_7, 0 in all other states. R = [5, 0, 0, 0, 0, 0, 10]
Let's have a *deterministic policy* π(s) = *Left* and γ = 0 for any state *s*, then what is the value of the policy V_π ?

V_π = [5, 0, 0, 0, 0, 0, 10] since γ = 0

Example: Policy Evaluation

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

R = [5, 0, 0, 0, 0, 0, 10]

- Practice 1: *Deterministic policy* π(s) = *Left* with γ = 0.5 for any state *s*, then what are the state values under the policy?
- Practice 2: *Stochastic policy* $P(\pi(\text{s}) = Left)$ = 0.5 and $P(\pi(\text{s}) = Right)$ = 0.5 and γ = 0.5 for any state *s*, then what are the state values under the policy?
- Iteration *t* :
  - $V_t^{\pi}(s) = \sum_{a \in A} P(\pi(\text{s}) = a)(R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) V_{t-1}^{\pi}(s'))$

Decision Making in MDP

- **Prediction** (evaluate a given policy):
  - Input: MDP $\langle S,A,P,R,\gamma\rangle$ and a policy π or MRP $\langle S, P^{\pi}, R^{\pi}, \gamma\rangle$
  - Output: value function V^{π}

- **Control** (search the optimal policy):
  - Input: MDP $\langle S,A,P,R,\gamma\rangle$
  - Output: optimal value function V^* and optimal policy π^*

Prediction and **control** in MDP can be solved by dynamic programming.

Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

I. Optimal substructure
   - Principle of optimality applies
   - Optimal solution can be decomposed into subproblems

II. Overlapping subproblems
   - Subproblems recur many times
   - Solutions can be cached and reused

Markov decision processes (MDP) satisfy both properties:

- Bellman equation gives recursive decomposition
- Value function stores and reuses solutions

Decision Making in MDP - Prediction

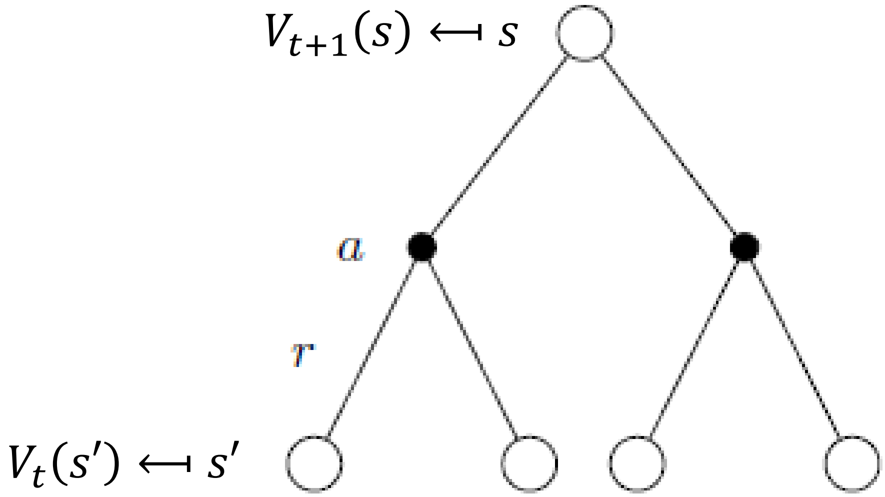
Policy evaluation on MDP

- Objective: Evaluate a **given policy** π for an MDP
- Output: the value function under policy V^{π}
- Solution: iteration on Bellman expectation backup
- Algorithm: Synchronous backup
  - *At each iteration t+1:*
    update $V_{t+1}(s)$ *from* $V_t(s')$ *for all states* $s \in S$ *where* s' *is a successor state of* s*:*

$$V_{t+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) V_t(s'))$$

- Convergence: $V_t \longrightarrow V_2 \longrightarrow \cdots \longrightarrow V_\pi$

Policy evaluation: Iteration on Bellman expectation backup



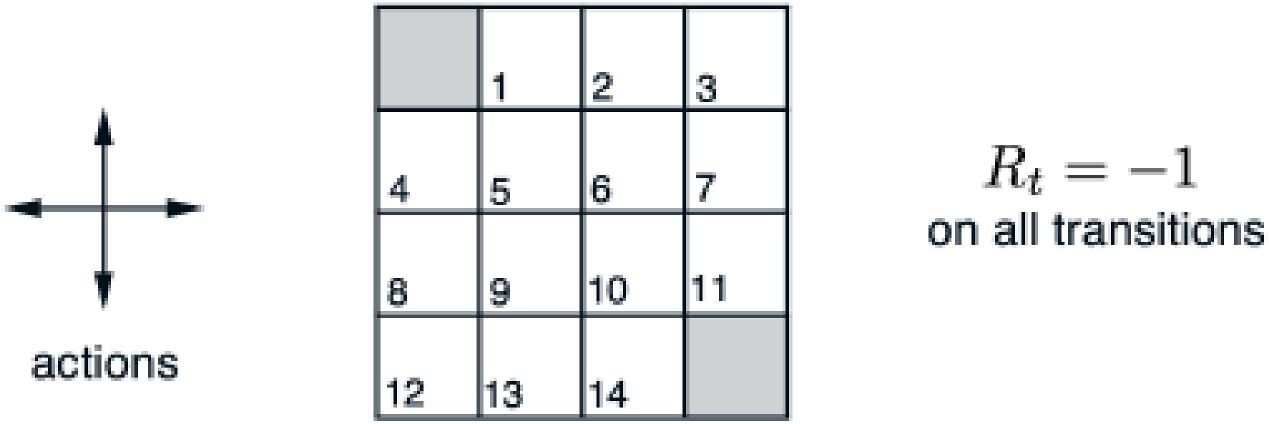
Bellman expectation backup for a particular policy

$$V_{t+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P(s'|s,a)V_t(s'))$$

Or if in the form of MRP $\langle S, P^{\pi}, R^{\pi}, \gamma \rangle$

$$V_{t+1}(s) = R^{\pi}(s) + \gamma P^{\pi}(s'|s)V_t(s')$$

Evaluating a Random Policy in the Small Gridworld



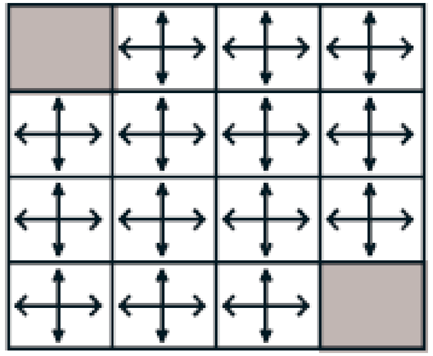
- Undiscounted episodic MDP (γ = 1)
- Nonterminal states 1,…,14
- Two terminal states (two shaded squares)
- Action leading out of grid leaves state unchanged, P(7|7, right) = 1
- Reward is -1 until the terminal state is reach
- Transition is deterministic given the action, e.g., P(6|5, right) = 1
- Uniform random policy $\pi(\mathrm{l}\,|\cdot) = \pi(\mathrm{r}|\cdot) = \pi(\mathrm{u}|\cdot) = \pi(\mathrm{d}|\cdot) = 0.25$

Iterative Policy Evaluation in Small Gridworld

V_t for the Random Policy | Greedy Policy

$t = 0$

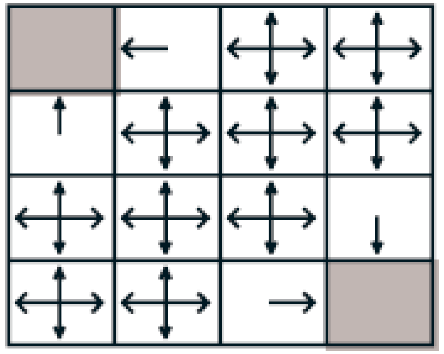
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Random policy

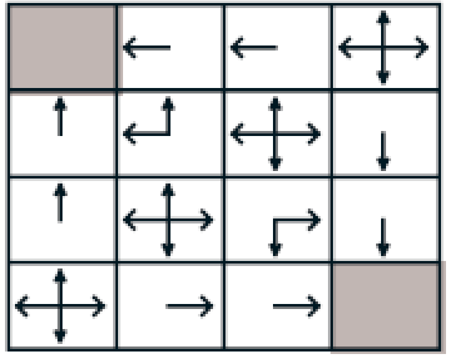
$t = 1$

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



$t = 2$

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

$t = 3$

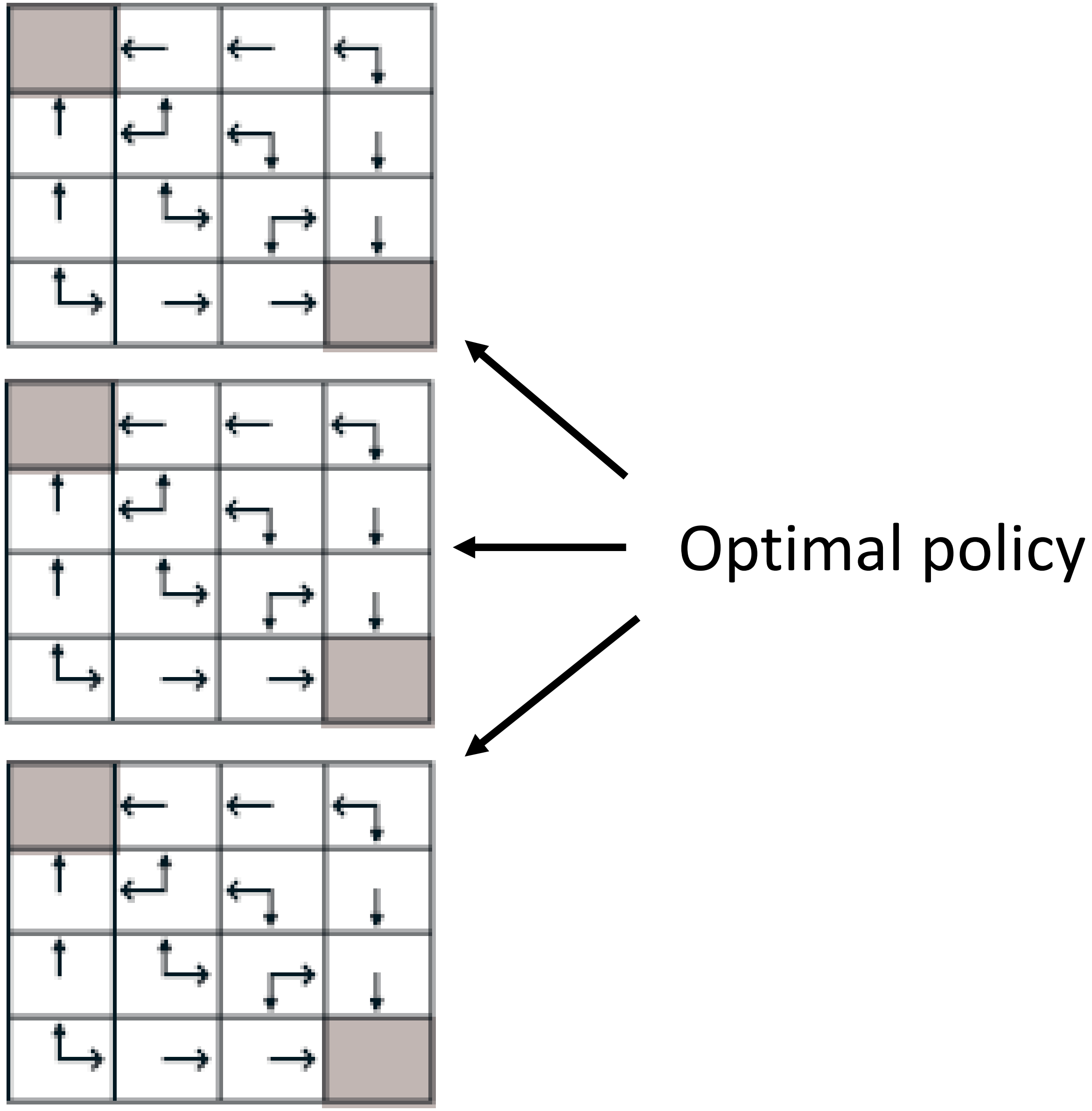
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$t = 10$

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

$t = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Decision Making in MDP - Control

Optimal Value Function

Definition

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

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

- The optimal value function species the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value function.

Optimal Policy

- Define a partial ordering over policies:

$$\pi \geq \pi' \text{ if } \mathrm{v}_{\pi}(s) \geq \mathrm{v}_{\pi'}(s), \forall \mathrm{s}$$

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 Optimal Policy

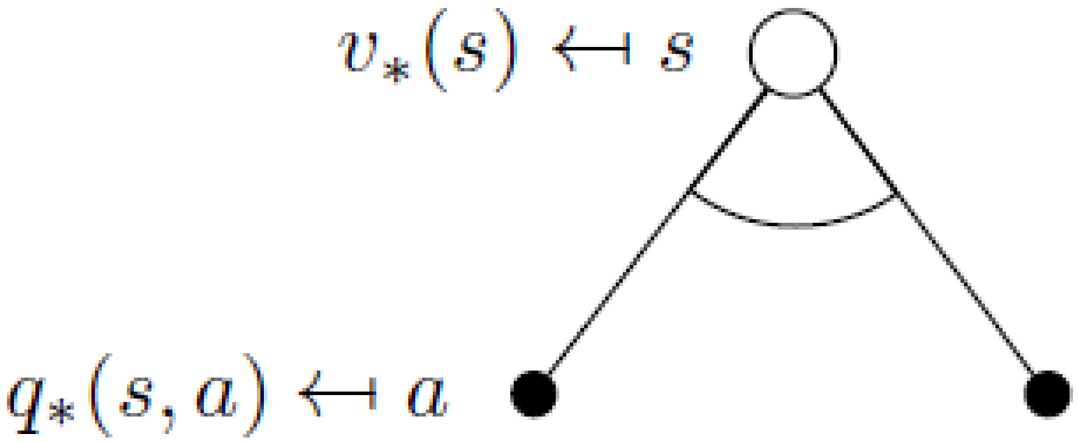
- An optimal policy can be found by maximizing over $q_*(s, a)$

$$\pi_*(a|s) = \begin{cases} 1, & if\ a = argmax_{a \in A}\, q_*(s, a) \\ 0, & otherwise \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q^*(s, a)$, we immediately have the optimal policy

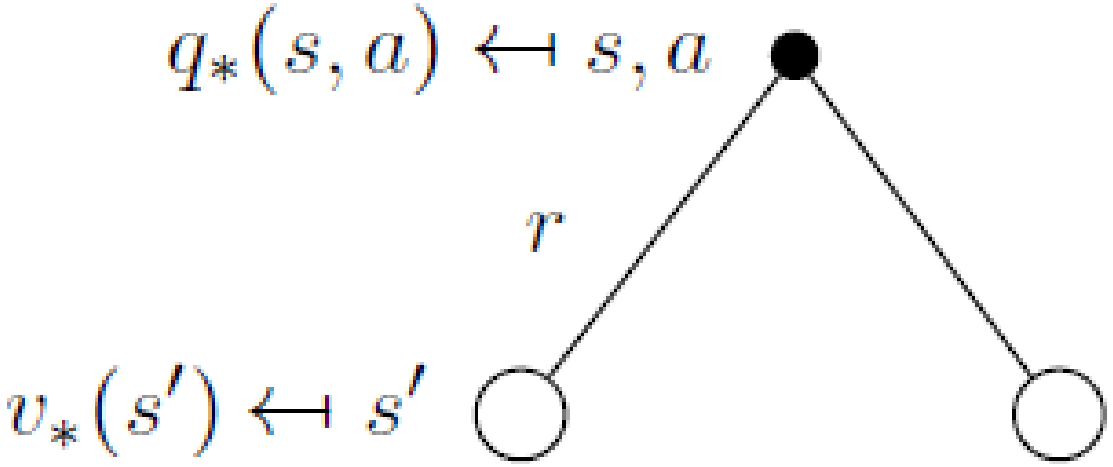
Bellman Optimality Equation for v_*

- The optimal value functions are recursively related by the Bellman optimality equations:



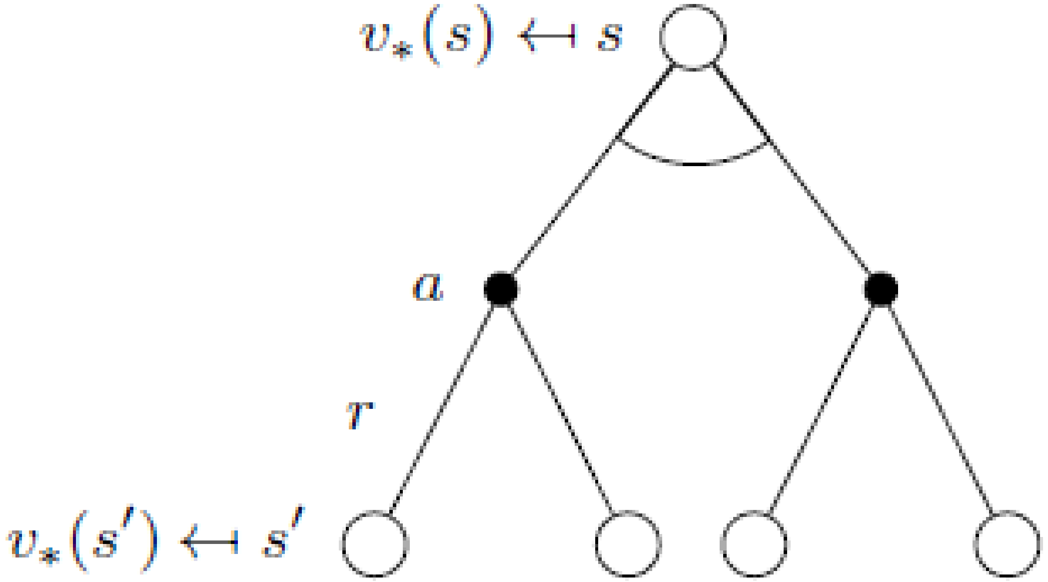
$$v_*(s) = \max_a q_*(s,a)$$

Bellman Optimality Equation for Q_*



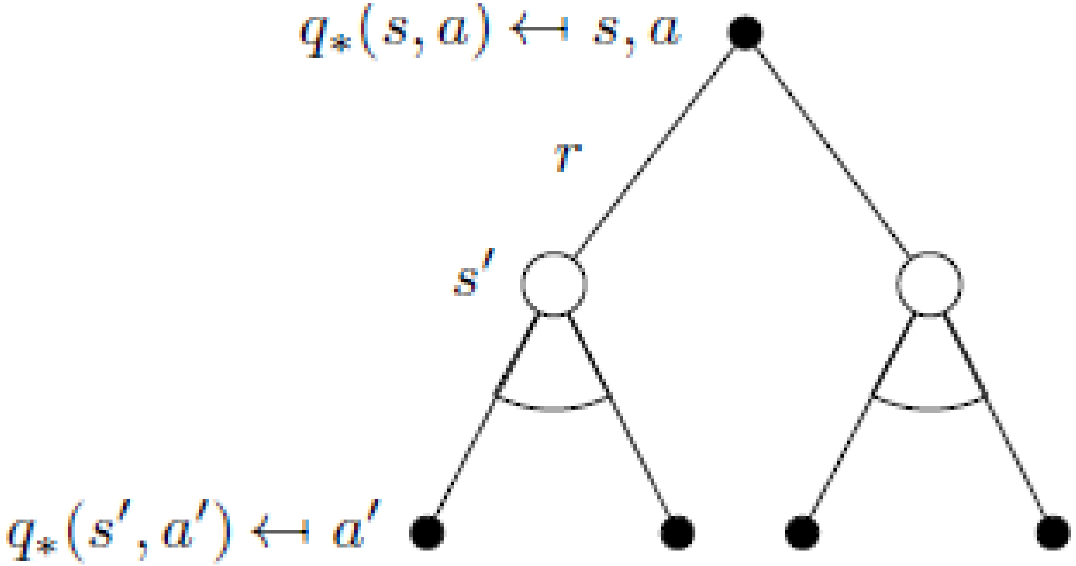
$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) V_*(s')$$

Bellman Optimality Equation for v_* cont.



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

Bellman Optimality Equation for Q_* cont



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

Solving the Bellman Optimality Equation

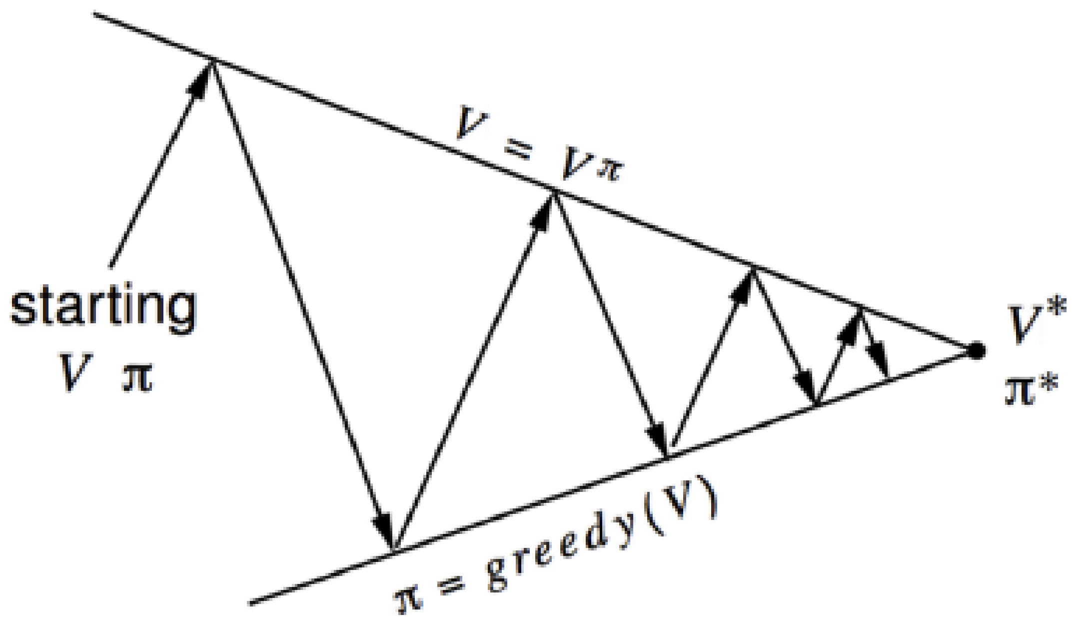
- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods(find Optimal Policy):
  - Policy Iteration
  - Value Iteration
  - Q-learning
  - Sarsa

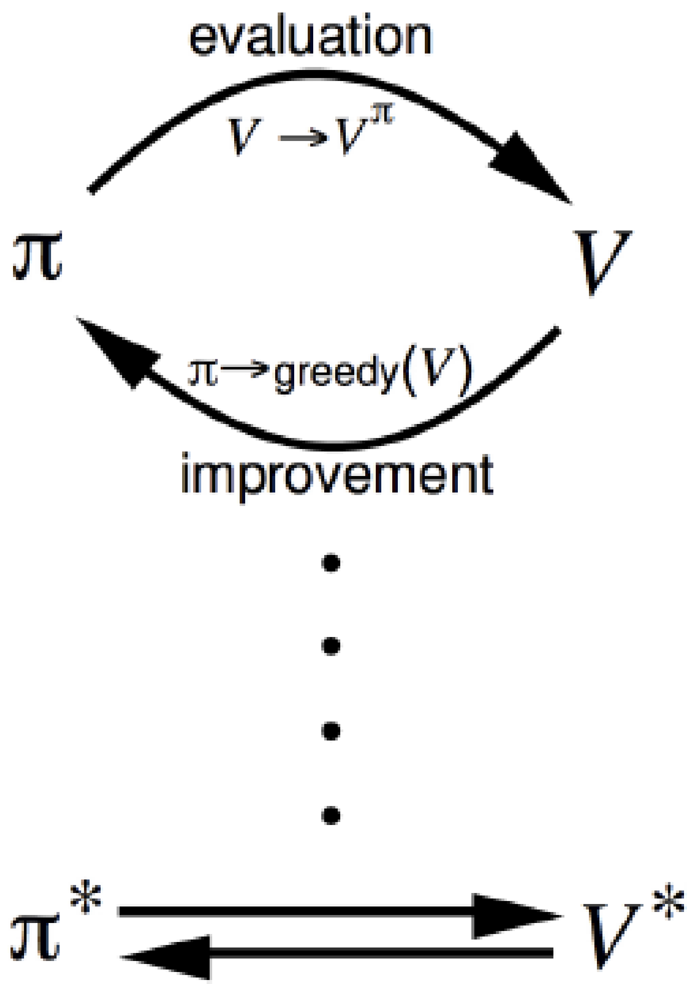
Policy Iteration in MDPs

Improve a Policy through Policy Iteration

- Iterate through the two steps:
  1. Policy evaluation: Evaluate the policy $\longrightarrow \mathrm{V}_{\pi}$
  2. Policy improvement: Generate $\pi' \geq \pi$

$$\pi' = greedy(\mathrm{V}_{\pi})$$





Principle of Optimality

- Any optimal policy can be subdivided into two components:
  - An optimal first action A_*
  - Followed by an optimal policy from successor state S'

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', $v_\pi(s') = v_*(s')$*

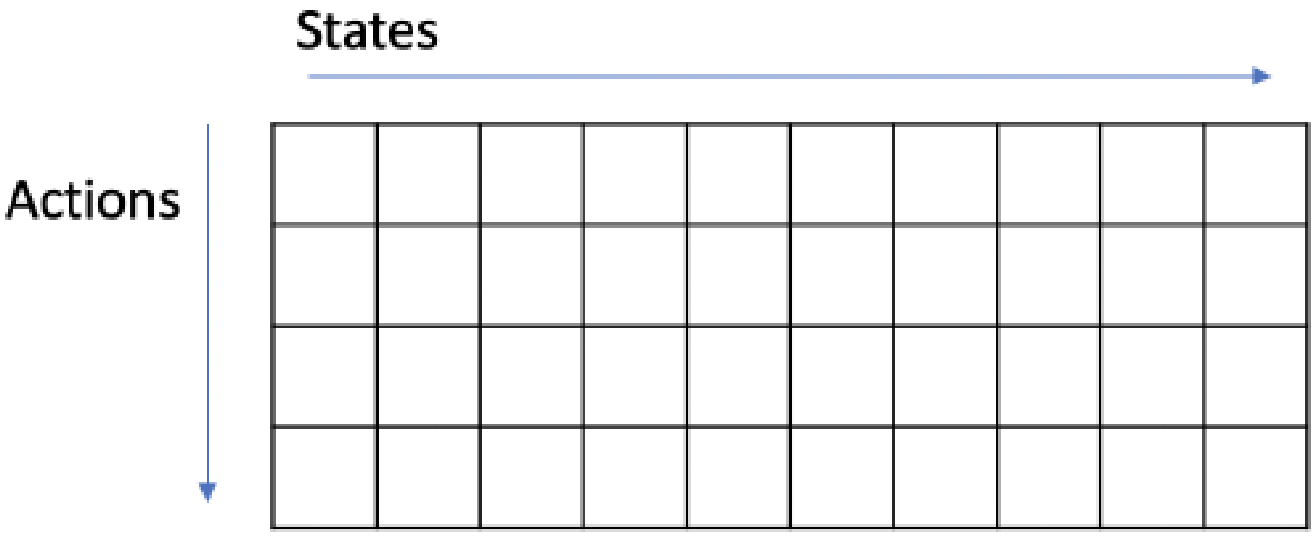
Policy Improvement

- Compute the state-action value of a policy π

$$q_{\pi_i}(s,a) = \; R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s')$$

- Compute new policy π_{i+1} for all $s \in S$ as following:

$$\pi_{i+1} = \underset{a}{argmax}\; q_{\pi i}(s,a)$$



Q-table

Monotonic Improvement in Policy

- Consider a deterministic policy $a = \pi(s)$
- We improve the policy through

$$\pi'(s) = \underset{a}{argmax}\ q_\pi(s, a)$$

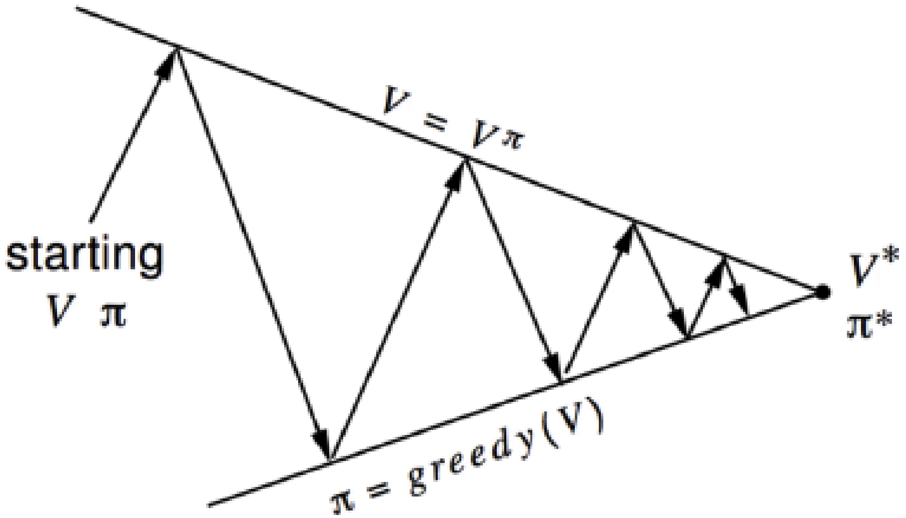
- This improves the value from any state s over one step:

$$q_\pi\big(s, \pi'(s)\big) = max_{a \in A} q_\pi(s, a) \geq q_\pi(s, \pi(s)\ ) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

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

When finish?



- The improvement process stop if,

$$q_\pi(s, \pi'(s)) = max_{a \in A} q_\pi(s, a) = q_\pi(s, \pi(s)\ ) = v_\pi(s)$$

- Thus the **Bellman optimality equation** has been satisfied

$$v_\pi(s) = max_{a \in A} q_\pi(s, a)$$

- Therefore $v_\pi(s) = v_*(s)$ for all s $\in S$, so π is an optimal policy

Bellman Optimality Equation

- The optimal value functions are reached by the Bellman optimality equations:

$$V_*(s) = max_{a \in A} q_*(s, a)$$

$$q_*(s, a) = \ R_s^a + \gamma \sum_{s' \in S} P(s'|s, a) V_*(s')$$

Thus:

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

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

Value Iteration in MDPs

Value Iteration by turning the Bellman Optimality Equation as update rule

- If we know the solution to subproblem $v_*(s')$,which is optimal.
- Then the solution for the optimal $v_*(s)$ can be found by iteration over the following Bellman Optimality backup rule,

$$v(s) \longleftarrow max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P(s'|s,a)v(s'))$$

- The idea of value iteration is to apply these updates iteratively

Algorithm of Value Iteration

- Objective: find the optimal policy
- Solution: iteration on the Bellman optimality backup
- Value Iteration algorithm:
  1. *initialize* $k = 1$ *and* $v_0(s) = 0$ *for all states* s
  2. *For* $k = 1 : H$
     - *For each state s*

$$q_{k+1}(s,a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s,a) \mathrm{v}_k(s')$$

$$V_{k+1}(s) = max_{a \in A} q_{k+1}(s,a)$$

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

$$\pi(s) = \underset{a}{argmax}\, R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \mathrm{v}_{k+1}(s')$$

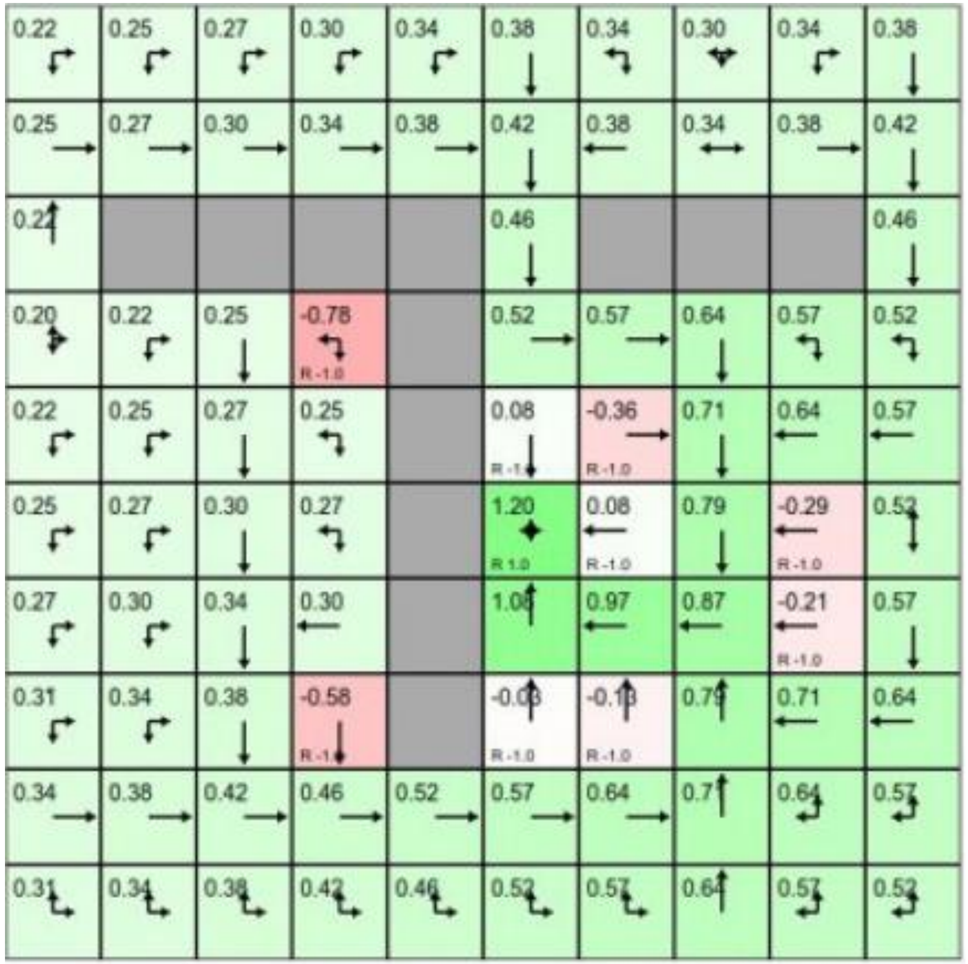
Policy Iteration vs. Value Iteration

- 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 evaluation (the reassignment of v(s) after just one sweep of all states regardless of convergence).

Summary for Prediction and Control in MDP

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

Demo of policy iteration and value iteration



- Policy iteration: Iteration of policy evaluation and policy
- improvement(update)
- Value iteration
- https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Extensions to MDPs

- Asynchronous Dynamic Programming
    - In-Places Dynamic Programming
    - Prioritized Sweeping
    - Real-Time Dynamic Programming
- Approximate Dynamic Programming
- Convergence Problem

Asynchronous Dynamic Programming

- A major drawback to the DP methods is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set.
- If the state set is very large, for example, the game of backgammon has over 10^{20}states. Thousands of years to be taken to finish one sweep.
- Asynchronous DP algorithms are in-place iterative DP that are not organized in terms of systematic sweeps of the state set
- The values of some states may be updated several times before the values of others are updated once.

In-Places Dynamic Programming

- Synchronous value iteration stores two copies of value function:
  1. *for all s in S:*

$$v_{\text{new}}(s) \leftarrow max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)\, v_{\text{old}}(s'))$$

  2. $v_{\text{new}} \leftarrow v_{\text{old}}$

- In-place value iteration only stores one copy of value function:

for all s in S:

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

Prioritized Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$| \, max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \, v(s')) \; - v(s) \, |$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Can be implemented efficiently by maintaining a priority queue

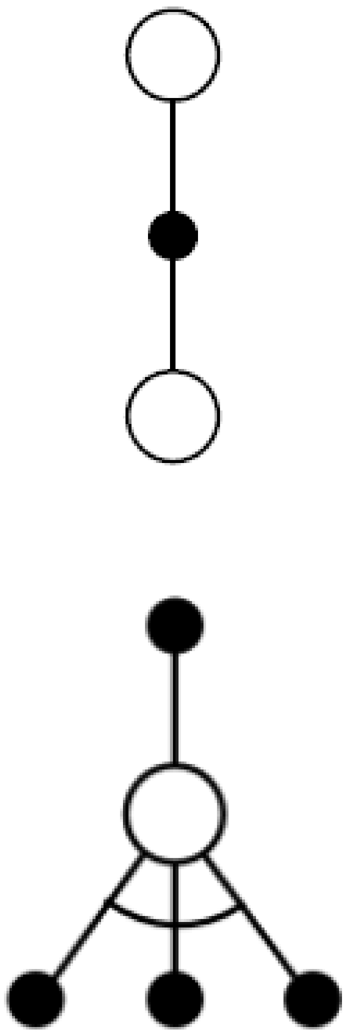
Real-Time Dynamic Programming

- To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP
- The agent's experience can be used to determine the states to which the DP algorithm applies its updates
- We can apply updates to states as the agent visits them. So focus on the parts of the state set that are most relevant to the agent
- After each time-step S_t, A_t, backup the state S_{t} ,

$$v(S_{\mathrm{t}}) \leftarrow max_{a \in A}(R(S_{\mathrm{t}}, a) + \gamma \sum_{s' \in S} P(s'|S_{\mathrm{t}}, a)\, v(s'))$$

Sample Backups

- The key design for RL algorithms such as Q-learning and SARSA in next lectures
- Using sample rewards and sample transitions $\langle S, A, R, S' \rangle$, rather than the reward function R and transition dynamics P
- Benefits:
  - Model-free: no advance knowledge of MDP required
  - Break the curse of dimensionality through sampling
  - Cost of backup is constant, independent of n = |S|



Approximate Dynamic Programming

- Using a function approximator $\hat{v}(s, \boldsymbol{w})$
- Fitted value iteration repeats at each iteration k,
    - Sample state s from the state cache $\tilde{S}$

$$\tilde{\mathrm{v}}_k(s) = max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)\,\hat{\mathrm{v}}(\mathrm{s}, \mathbf{w}))$$

    - Train next value function $\hat{v}(s, \boldsymbol{w}_{k+1})$using targets < s, $\tilde{v}_k$ >.
- Key idea behind the Deep Q-Learning

Convergence Problem

Some Technical Questions

- How do we know that value iteration converges to v_*?
- Or that iterative policy evaluation converges to v_π?
- And therefore that policy iteration converges to v_*?
- Is the solution unique?
- How fast do these algorithms converge?
- These questions are resolved by contraction mapping theorem

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,

$$\|u - v\|_\infty = max_{s \in S} |u(s) - v(s)|$$

Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator T^{π}

$$T^{\pi}(v) = R^{\pi} + \gamma P^{\pi} v$$

- This operator is a γ -contraction, i.e. it makes value functions closer by at least γ

$$\begin{aligned}
\|u - v\|_{\infty} &= \|(R^{\pi} + \gamma P^{\pi} u) - (R^{\pi} + \gamma P^{\pi} v)\|_{\infty} \\
&= \|\gamma P^{\pi}(u - v)\|_{\infty} \\
&\leq \|\gamma P^{\pi} \|u - v\|_{\infty}\|_{\infty} \\
&\leq \gamma \|u - v\|_{\infty}
\end{aligned}$$

Contraction Mapping Theorem

Theorem (Contraction Mapping Theorem)

For any metric space $\mathcal{V}$ that is complete (i.e. closed) under an operator $T(v)$, where T is a γ-contraction,

- *T converges to a unique fixed point*
- *At a linear convergence rate of γ*

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator T^{π} has a unique fixed point
- v_{π} is a fixed point of T^{π} (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges on v_{π}
- Policy iteration converges on v_{*}

Bellman Optimality Backup is a Contraction

- Define the *Bellman optimality backup operator* T^*

$$T^*(v) = max_{a \in A}(R^a + \gamma P^a v)$$

- This operator is a γ -contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$\|T^*(u) - T^*(v)\|_\infty \leq \gamma \|u - v\|_\infty$$

Convergence of Value Iteration

- The Bellman optimality operator T^* has a unique fixed point
- v_* is a fixed point of T^* (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on v_*

Next Lecture: