

Comments from Lecture 2

Deterministic VS Stochastic Policies

- Every MDP has a deterministic optimal policy. Why?
 - We can just act greedily w.r.t the optimal action value function q^*
- Then, why do we talk about stochastic policies?
 - Stochastic actions are needed for exploration (later, when the reward and dynamics will be unknown)

MDPs VS Games

- Rock paper scissors has an optimal **stochastic** policy. Why?
 - It is not an MDP! In MDPs, the dynamics of the environment are assumed Markovian: The next state distribution depends solely on my current state and action, not all my so far moves.
 - To model that the adversary can adapt to our policy so far, then states should be defined as the whole episode since starting playing with that adversary (not as the current time step empty information state). This is not a useful MDP definition since states never approximately repeat. Look at Bayesian Decision Making and Game Theory.
 - For games where the states are indeed Markovian e.g., Go, Backgammon MDPs are fine, backup diagrams work, is just you need to do min-max at the point when you consider the environment dynamics.
- Can the backgammon MDP model an adversary that adapts to our way of playing?
 - No: the dynamics of the environment have been assumed Markovian conditioned on the state: The next state distribution depends solely on my current state and action, not all my so far moves! No adaptation can be modeled.
 - So, for the MDPs we have discussed, the environment can be adversarial, but cannot adapt.

State uncertainty

- Could you think of reasons why it is useful to model state uncertainty?
- In self-driving cars, I am not sure of a detection being a bag or a baby, such uncertainty may signal to go closer and check. This can be thought either as stochastic modeling of a state, or a deterministic modeling of the state, where the state is that I am simply uncertain about my perception and I will do a deterministic action to resolve my uncertainty.
- This is different than the hidden states in POMDPs, where you need to figure out in which state you are in: it's a bag, thus you go ahead with the same speed, or is a baby so you slow down. Here you just go closer to resolve your uncertainty.

Rewards

- Rewards may depend on the state the agent is transitioning from, the state he is transitioning to, the action performed, or all of the above. In our representation, it did not depend on the state you are transitioning to.

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

Rewards

- Rewards may depend on the state the agent is transitioning from, the state he is transitioning to, the action performed, or all of the above. In our representation, it did not depend on the state you are transitioning to.

Recall: **Bellman equation for v_π** when rewards depend on previous state and action

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s'))$$

Bellman equation for v_π when rewards depend on the next state to?

Lecture 3: Planning in Markov decision processes

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

Solving MDPs

- Prediction = Given an MDP (S, A, P, R, γ) and a policy π find the state and action value functions
- Optimal Control = Given an MDP (S, A, P, R, γ) find the optimal policy a.k.a. the *planning* problem - Contrast with the *learning* problem under missing information about rewards and/or dynamics
- We still consider **finite MDPs** (discrete and finite set of states and actions) **with known dynamics and rewards**

Outline

- ❑ Exact solution methods:
 - ❑ Policy iteration
 - ❑ Policy Evaluation (a.k.a. DP)
 - ❑ Policy Improvement
 - ❑ Value iteration (a.k.a. DP)
 - ❑ Linear programming
- ❑ Approximate solution methods:
 - ❑ Asynchronous DP

Policy Evaluation

Policy Evaluation: for a given policy π , compute the state-value function v_π

Recall: **State-value function for policy π**

$$v_\pi(s) = \mathbb{E}_\pi[G_t^\cdot \mid S_t = s] = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$

Recall: **Bellman equation for v_π**

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s'))$$

— a system of $|S|$ simultaneous equations

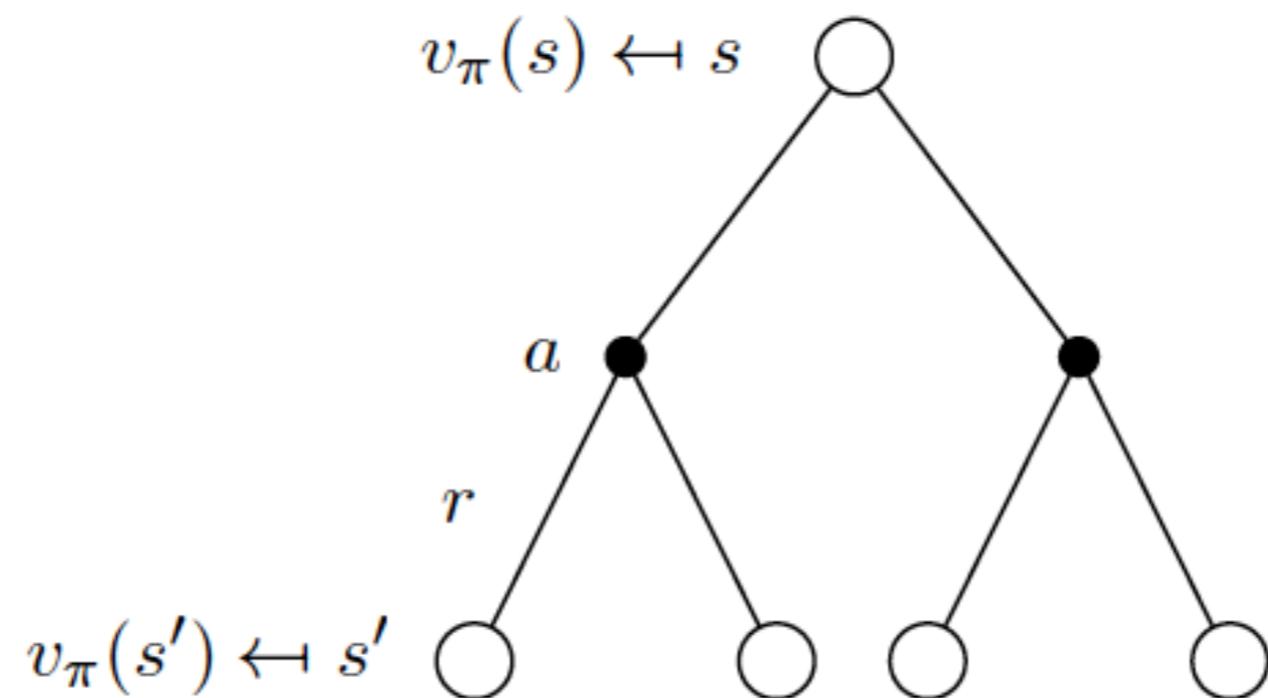
MDPs to MRPs

Under a fixed policy, MDP becomes a Markov Reward Process

$$v_{\pi}(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s'))$$

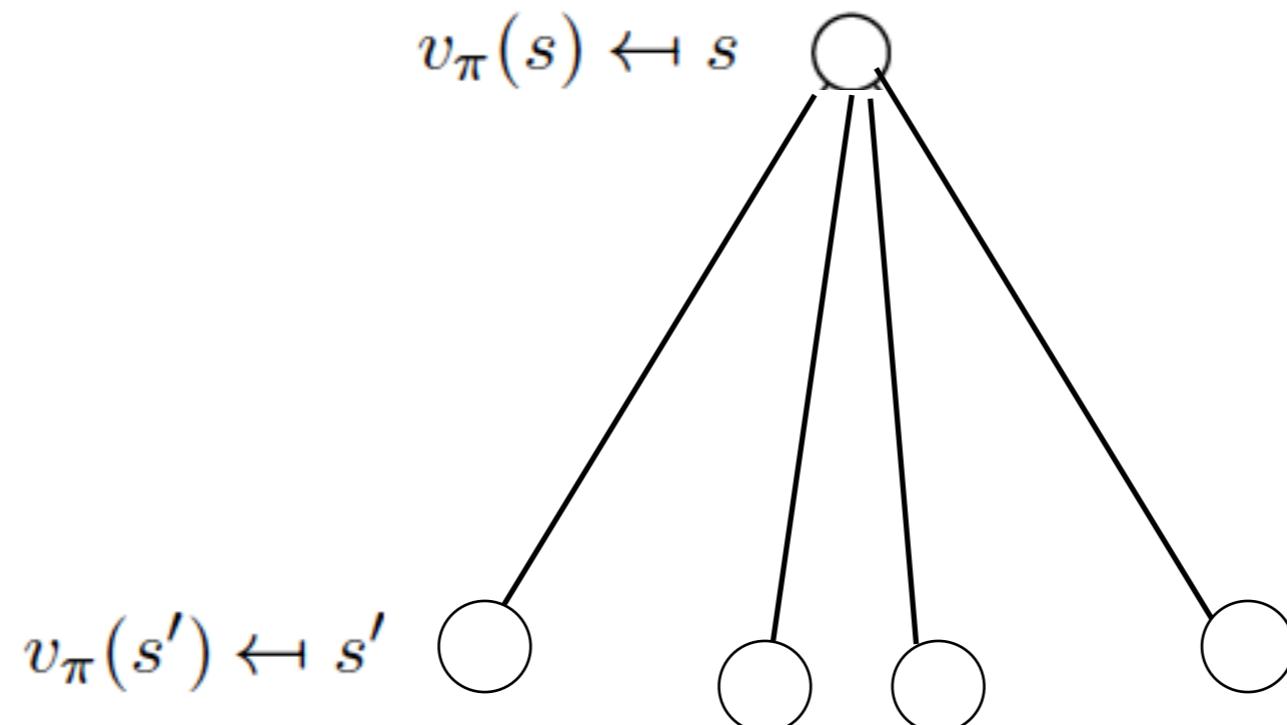
$$\begin{aligned} v_{\pi}(s) &= \sum_{a \in A} \pi(a|s) R_s^a + \gamma \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \\ &= R_s^{\pi} + \gamma \sum_{s' \in S} P_s^{\pi} v_{\pi}(s') \end{aligned}$$

Back up Diagram for state value functions



Back up Diagram for state value functions in MRP

Under a fixed policy, MDP becomes a Markov Reward Process



$$v_\pi(s) = R_s^\pi + \gamma \sum_{s' \in S} P_s^\pi v_\pi(s')$$

Matrix form

The Bellman expectation equation can be expressed concisely using the induced MRP,

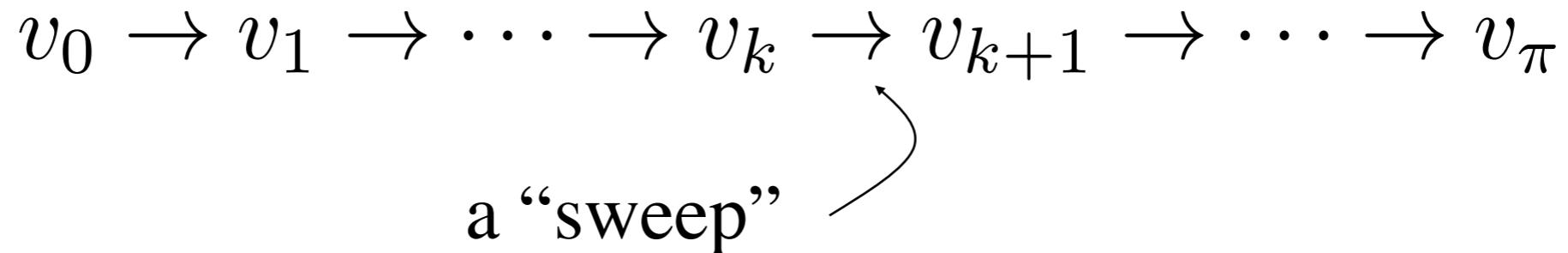
$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

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

complexity $O(N^3)$

Iterative Methods

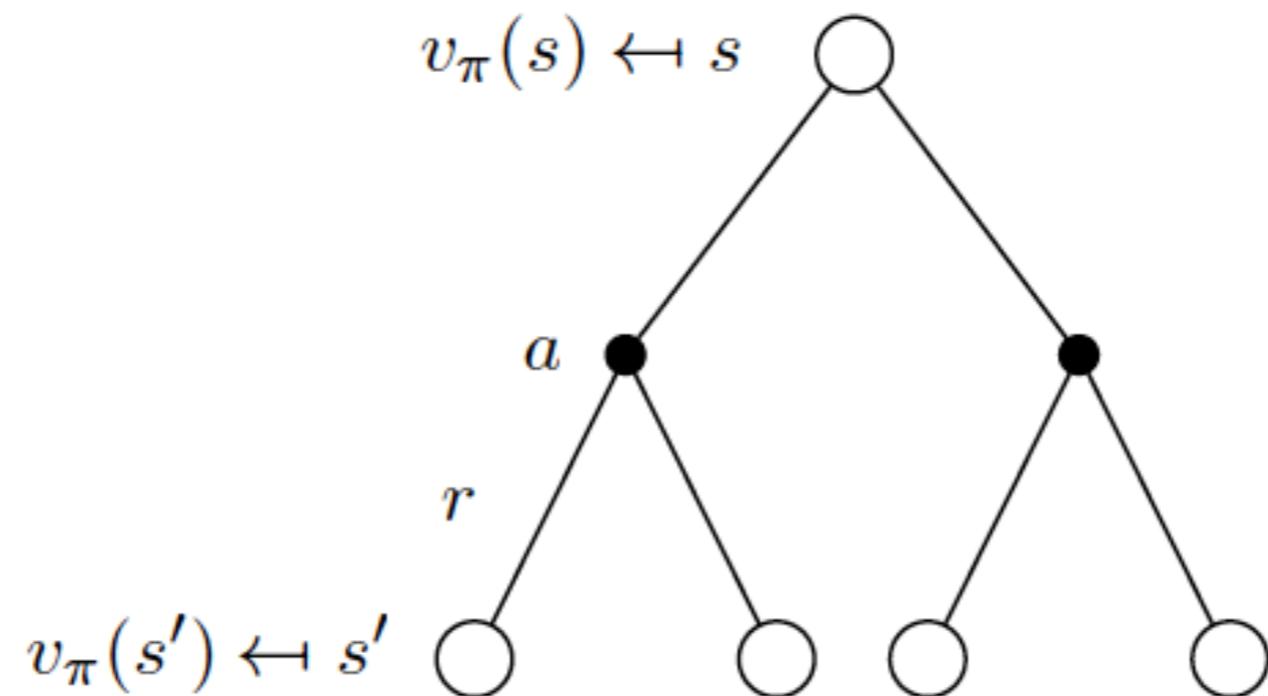


A sweep consists of applying a **backup operation** to each state.

A **full policy-evaluation backup**:

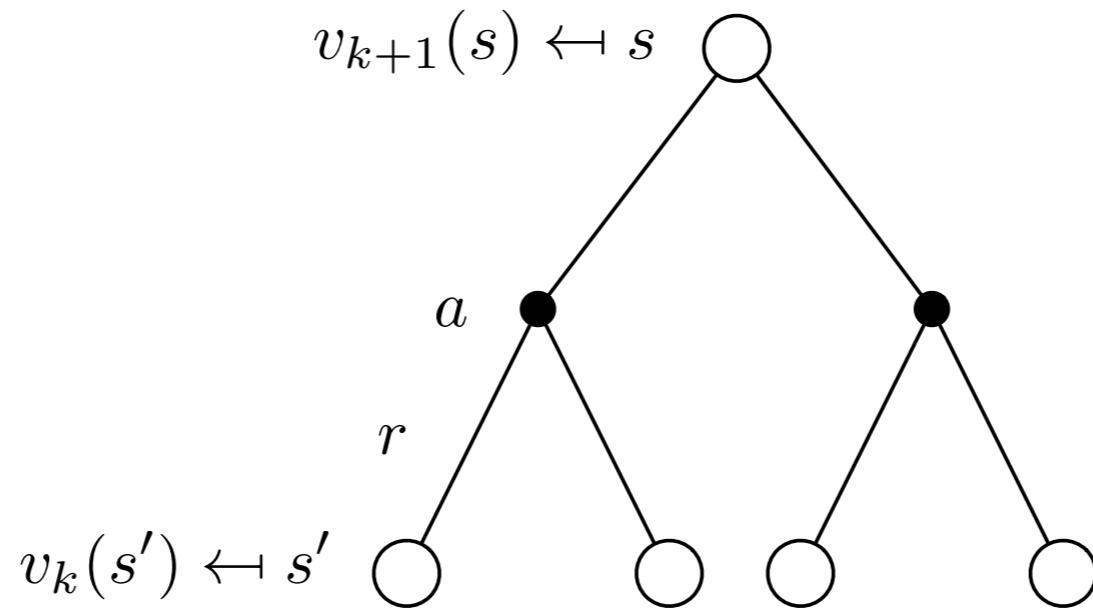
$$v_\pi(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')) \quad \forall s \in S$$

Back up Diagram for state value functions



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

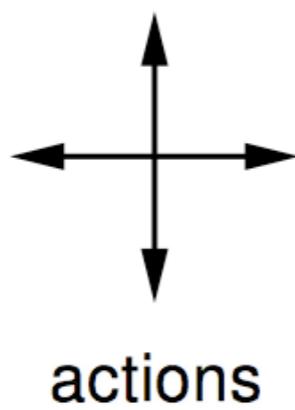
Value function iterative update



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

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

A Small Gridworld



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

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

Iterative Policy Eval for the Small Gridworld

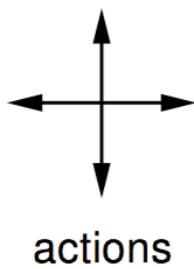
V_k for the
Random Policy

$k = 0$

| | | | |
|-----|-----|-----|-----|
| 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 |

π = equiprobable random action choices

$k = 1$



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

$R = -1$
on all transitions

$\gamma = 1$

$k = 2$

$k = 3$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

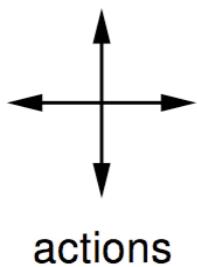
$k = 10$

$k = \infty$

Iterative Policy Eval for the Small Gridworld

V_k for the
Random Policy

π = equiprobable random action choices



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

$R = -1$
on all transitions

$\gamma = 1$

$k = 0$

| | | | |
|-----|-----|-----|-----|
| 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 |

$k = 1$

| | | | |
|------|------|------|------|
| 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 |

$k = 2$

$k = 3$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

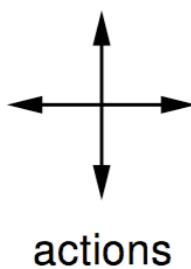
$k = 10$

$k = \infty$

Iterative Policy Eval for the Small Gridworld

V_k for the
Random Policy

π = equiprobable random action choices



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

$R = -1$
on all transitions

$\gamma = 1$

$k = 0$

| | | | |
|-----|-----|-----|-----|
| 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 |

$k = 1$

| | | | |
|------|------|------|------|
| 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 |

$k = 3$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

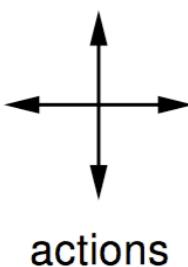
$k = 10$

$k = \infty$

Iterative Policy Eval for the Small Gridworld

V_k for the
Random Policy

π = equiprobable random action choices



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

$R = -1$
on all transitions

$\gamma = 1$

$k = 0$

| | | | |
|-----|-----|-----|-----|
| 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 |

$k = 1$

| | | | |
|------|------|------|------|
| 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 |

$k = 3$

| | | | |
|------|------|------|------|
| 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 |

$k = 10$

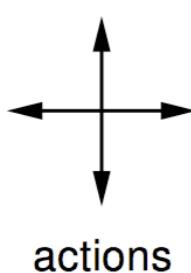
- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

$k = \infty$

Iterative Policy Eval for the Small Gridworld

V_k for the
Random Policy

π = equiprobable random action choices



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

$R = -1$
on all transitions

$\gamma = 1$

$k = 0$

| | | | |
|-----|-----|-----|-----|
| 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 |

$k = 1$

| | | | |
|------|------|------|------|
| 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 |

$k = 3$

| | | | |
|------|------|------|------|
| 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 |

Iterative Policy Evaluation

Input π , the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s)(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Value Function Space

- Consider the vector space \mathcal{V} over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $v(s)$
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions *closer*
- And 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,

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

Bellman Expectation Backup is a Contraction

- Define the *Bellman expectation backup operator* T^π ,

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

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

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

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 Improvement

Suppose we have computed v_π for a deterministic policy π .

For a given state s ,
would it be better to do an action $a \neq \pi(s)$?

It is better to switch to action a for state s if and only if

$$q_\pi(s, a) > v_\pi(s)$$

And, we can compute $q_\pi(s, a)$ from v_π by:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s') \end{aligned}$$

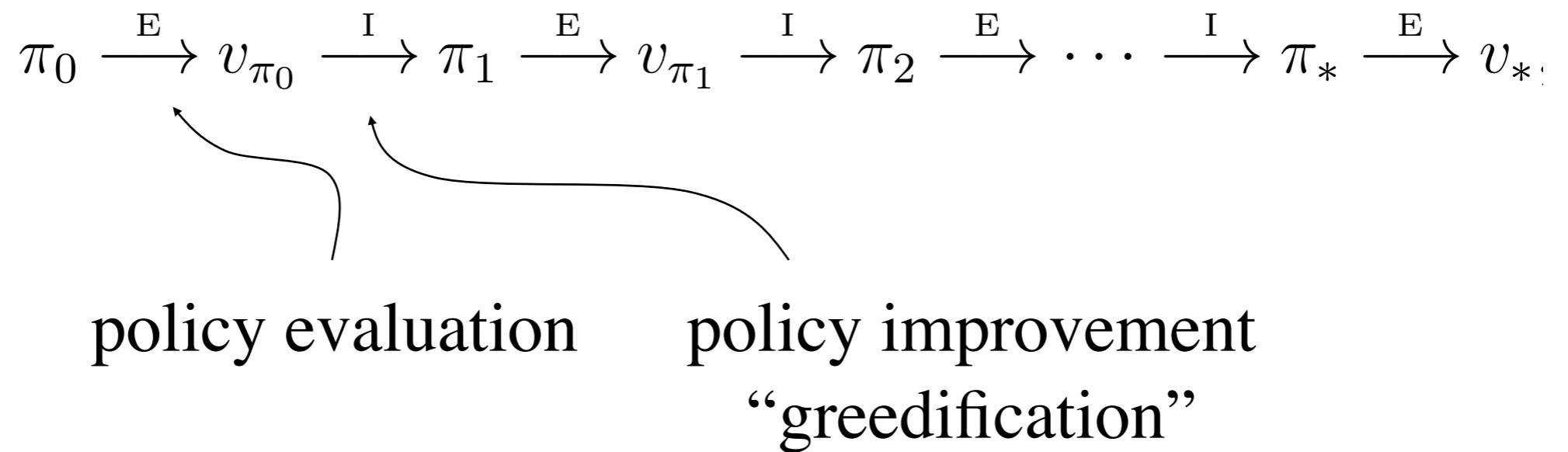
Policy Improvement Cont.

Do this for all states to get a new policy $\pi' \geq \pi$ that is **greedy** with respect to v_π :

$$\begin{aligned}\pi'(s) &= \arg \max_a q_\pi(s, a) \\ &= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(s') \mid S_t = s, A_t = a] \\ &= \arg \max R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')\end{aligned}$$

What if the policy is unchanged by this?
Then the policy must be optimal!

Policy Iteration



Policy Iteration

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s)(R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s'))$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

$$a \leftarrow \pi(s)$$

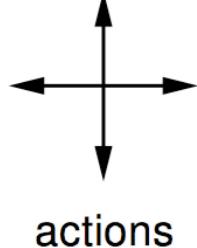
$$\pi(s) \leftarrow \arg \max R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')$$

If $a \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return V and π ; else go to 2

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



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

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

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 |

$k = 0$

Greedy Policy
w.r.t. V_k

| | | | |
|---|---|---|---|
| ↑ | ↔ | ↔ | ↔ |
| ↑ | ↔ | ↔ | ↔ |
| ↑ | ↔ | ↔ | ↔ |
| ↑ | ↔ | ↔ | ↔ |

random
policy

| | | | |
|------|------|------|------|
| 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 |

$k = 1$

| | | | |
|------|------|------|------|
| 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 |

$k = 2$

| | | | |
|------|------|------|------|
| 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 |

$k = 3$

| | | | |
|------|------|------|------|
| 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 |

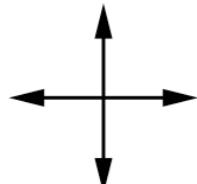
$k = 10$

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

$k = \infty$

Iterative Policy Eval for the Small Gridworld

π = equiprobable random action choices



actions

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

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, . . . , 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

$k = 0$

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 |

$k = 1$

| | | | |
|------|------|------|------|
| 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 |

$k = 3$

| | | | |
|------|------|------|------|
| 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 |

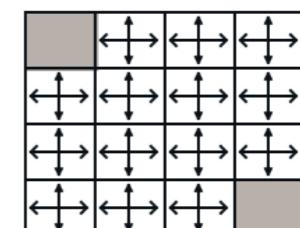
$k = 10$

| | | | |
|------|------|------|------|
| 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 |

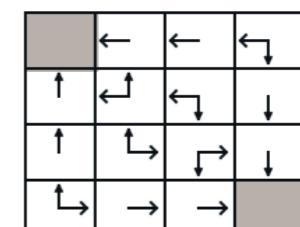
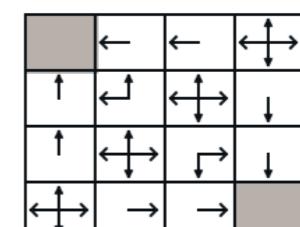
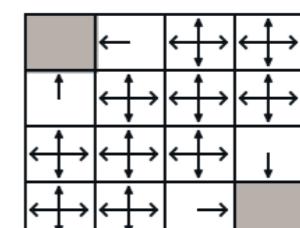
$k = \infty$

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

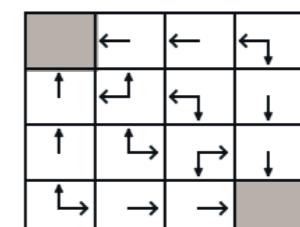
Greedy Policy
w.r.t. V_k



random policy



optimal policy

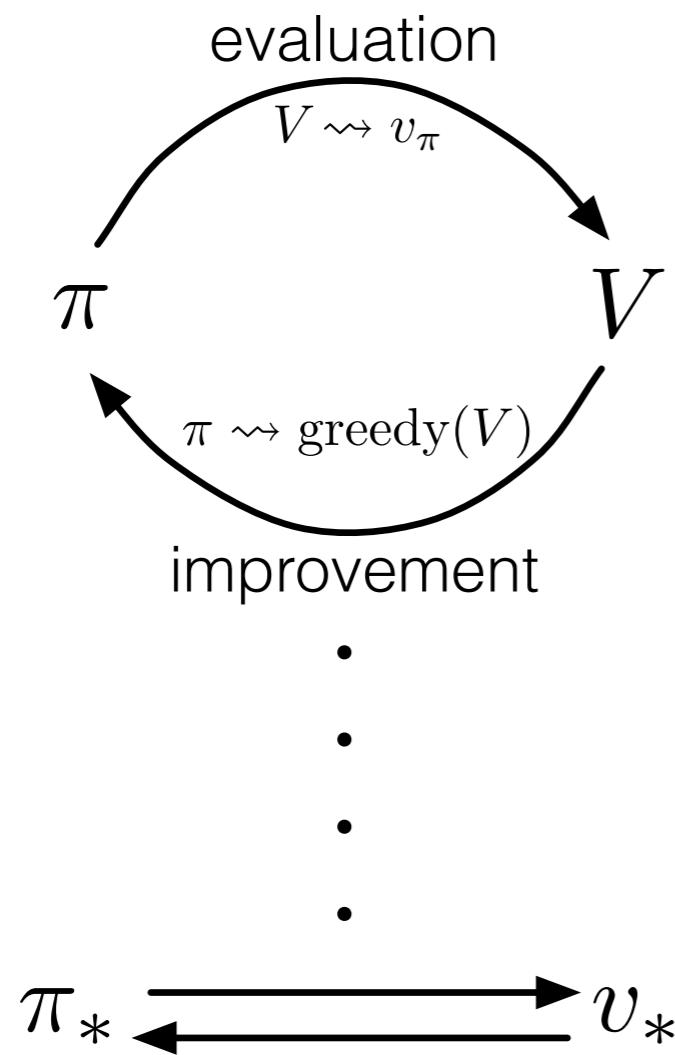


Generalized Policy Iteration

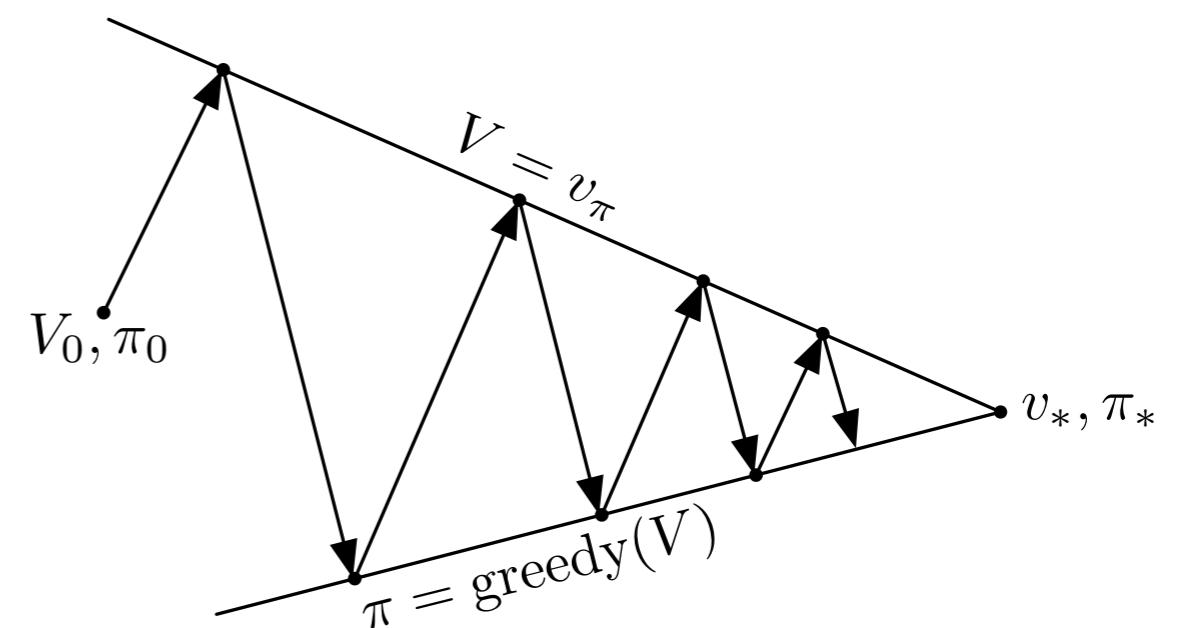
- Does policy evaluation need to converge to v_π ?
- Or should we introduce a stopping condition
 - e.g. ϵ -convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e. stop after $k = 1$
 - This is equivalent to *value iteration* (next section)

Generalized Policy Iteration

Generalized Policy Iteration (GPI):
any interaction of policy evaluation and policy improvement,
independent of their granularity.



A geometric metaphor for convergence of GPI:



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

Value Iteration

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

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

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs

Example: Shortest Path

| g | | | |
|-----|--|--|--|
| | | | |
| | | | |
| | | | |
| | | | |

Problem

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

V_1

| | | | |
|----|----|----|----|
| 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 |

V_2

| | | | |
|----|----|----|----|
| 0 | -1 | -2 | -2 |
| -1 | -2 | -2 | -2 |
| -2 | -2 | -2 | -2 |
| -2 | -2 | -2 | -2 |

V_3

| | | | |
|----|----|----|----|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -3 |
| -2 | -3 | -3 | -3 |
| -3 | -3 | -3 | -3 |

V_4

| | | | |
|----|----|----|----|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -4 |
| -2 | -3 | -4 | -4 |
| -3 | -4 | -4 | -4 |

V_5

| | | | |
|----|----|----|----|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -4 |
| -2 | -3 | -4 | -5 |
| -3 | -4 | -5 | -5 |

V_6

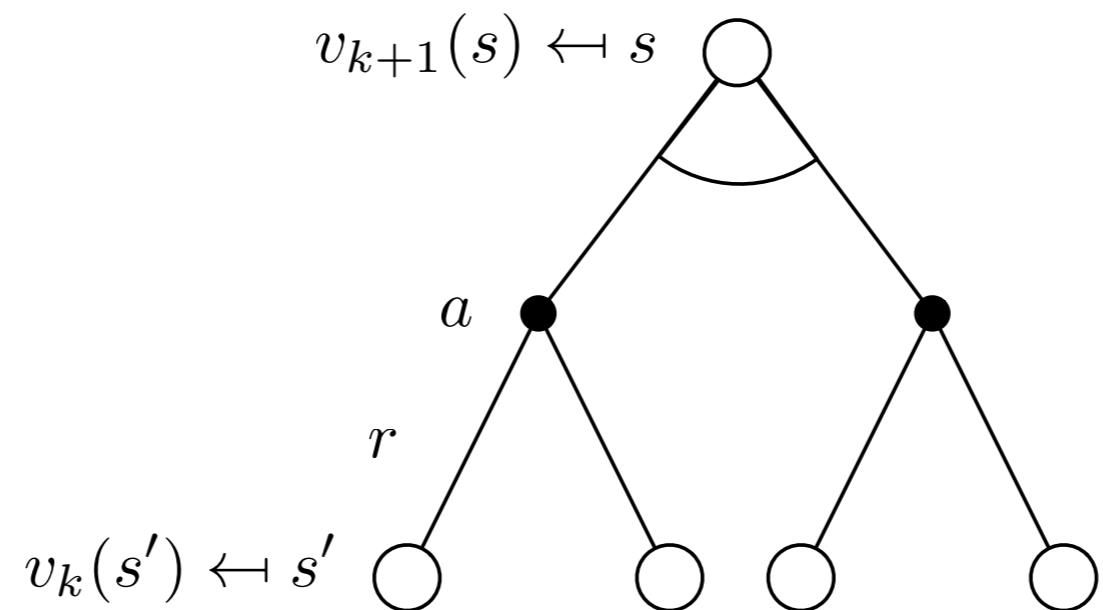
| | | | |
|----|----|----|----|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -4 |
| -2 | -3 | -4 | -5 |
| -3 | -4 | -5 | -6 |

V_7

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Using synchronous backups
 - At each iteration $k + 1$
 - For all states $s \in \mathcal{S}$
 - Update $v_{k+1}(s)$ from $v_k(s')$

Value Iteration (2)



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

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a \mathbf{v}_k$$

Bellman Optimality Backup is a Contraction

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

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

- 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_*

Synchronous Dynamic Programming Algorithms

| 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 |

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2n^2)$ per iteration

Efficiency of DP

- To find an optimal policy is polynomial in the number of states...
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- In practice, classical DP can be applied to problems with a few millions of states.

Asynchronous DP

- ❑ All the DP methods described so far require exhaustive sweeps of the entire state set.
- ❑ Asynchronous DP does not use sweeps. Instead it works like this:
 - Repeat until convergence criterion is met:
 - Pick a state at random and apply the appropriate backup
- ❑ Still need lots of computation, but does not get locked into hopelessly long sweeps
- ❑ Can you select states to backup intelligently? YES: an agent's experience can act as a guide.

Asynchronous Dynamic Programming

Three simple ideas for asynchronous dynamic programming:

- *In-place* dynamic programming
- *Prioritised sweeping*
- *Real-time* dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function
for all s in \mathcal{S}

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

$$v_{old} \leftarrow v_{new}$$

- In-place value iteration only stores one copy of value function
for all s in \mathcal{S}

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

Prioritised Sweeping

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

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

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

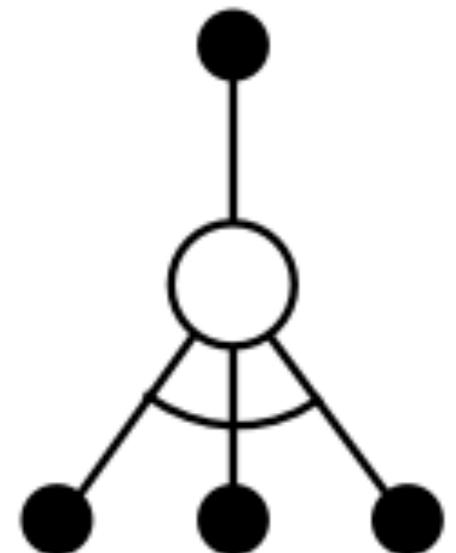
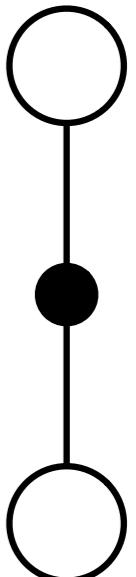
Real-Time Dynamic Programming

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step S_t, A_t, R_{t+1}
- Backup the state S_t

$$v(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_{S_t}^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{S_t s'}^a v(s') \right)$$

Sample Backups

- In subsequent lectures we will consider *sample backups*
- Using sample rewards and sample transitions
 $\langle S, A, R, S' \rangle$
- Instead of reward function \mathcal{R} and transition dynamics \mathcal{P}
- Advantages:
 - Model-free: no advance knowledge of MDP required
 - Breaks the curse of dimensionality through sampling
 - Cost of backup is constant, independent of $n = |\mathcal{S}|$



Approximate Dynamic Programming

- Approximate the value function
- Using a *function approximator* $\hat{v}(s, \mathbf{w})$
- Apply dynamic programming to $\hat{v}(\cdot, \mathbf{w})$
- e.g. Fitted Value Iteration repeats at each iteration k ,
 - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
 - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,

$$\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}(s', \mathbf{w}_k) \right)$$

- Train next value function $\hat{v}(\cdot, \mathbf{w}_{k+1})$ using targets $\{\langle s, \tilde{v}_k(s) \rangle\}$

Linear Programming for solving the Bellman Optimality Equations

- Recall, at value iteration convergence we have

$$\forall s \in S : V^*(s) = \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s')$$

- LP formulation to find V^* :

$$\min_V \sum_s \mu_0(s)V(s)$$

s.t. $\forall s \in S, \forall a \in A :$

$$V(s) \geq R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s')$$

μ_0 is a probability distribution over S , with $\mu_0(s) > 0$ for all s in S .

Theorem. V^* is the solution to the above LP.

Linear Programming for solving the Bellman Optimality Equations

Let F be the Bellman operator, i.e., $V_{i+1}^* = F(V_i)$. Then the LP can be written as:

$$\begin{aligned} \min_V \quad & \mu_0^\top V \\ \text{s.t.} \quad & V \geq F(V) \end{aligned}$$

Monotonicity Property: If $U \geq V$ then $F(U) \geq F(V)$.

Hence, if $V \geq F(V)$ then $F(V) \geq F(F(V))$, and by repeated application, $V \geq F(V) \geq F^2V \geq F^3V \geq \dots \geq F^\infty V = V^*$.

Any feasible solution to the LP must satisfy $V \geq F(V)$, and hence must satisfy $V \geq V^*$. Hence, assuming all entries in μ_0 are positive, V^* is the optimal solution to the LP.

The Dual: optimal action value functions!

$$\begin{aligned} \max_{\lambda} \quad & \sum_{s \in S} \sum_{a \in A} \sum_{s' \in S} \lambda(s, a) P_{ss'}^a R_s^a \\ \text{s.t.} \quad & \forall s' \in S : \sum_{a' \in A} \lambda(s', a') = \mu_0(s) + \gamma \sum_{s \in S} \sum_{a \in A} \lambda(s, a) P_{ss'}^a \end{aligned}$$

■ Interpretation:

- $\lambda(s, a) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a)$
- Equation 2: ensures that λ has the above meaning
- Equation 1: maximize expected discounted sum of rewards
- Optimal policy: $\pi^*(s) = \arg \max_a \lambda(s, a)$

Summary