Policy Iteration & Monte Carlo Prediction

CMPUT 366: Intelligent Systems

S&B §4.3-4.4, 5.0-5.2

Lecture Outline

1. Recap & Logistics
2. Policy Iteration
3. Monte Carlo Prediction

Assignment #3

- Assignment #3 is due **Mar 29 (next Monday)** at 11:59pm
- Reminder that TAs are available during office hours 5 days/week to help
- **TensorFlow tutorial** in today's office hour:
  - Wednesday Mar 24 at **2:00pm**
  - see eClass for Google Meet link

Recap: Value Functions

State-value function

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

Action-value function

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

Recap: Bellman Equations

Value functions satisfy a **recursive consistency condition** called the **Bellman equation**:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \Big] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big] \end{aligned}$$

- v_π is the **unique solution** to π's (state-value) Bellman equation
- There is also a Bellman equation for π's **action-value function**

In-Place Iterative Policy Evaluation

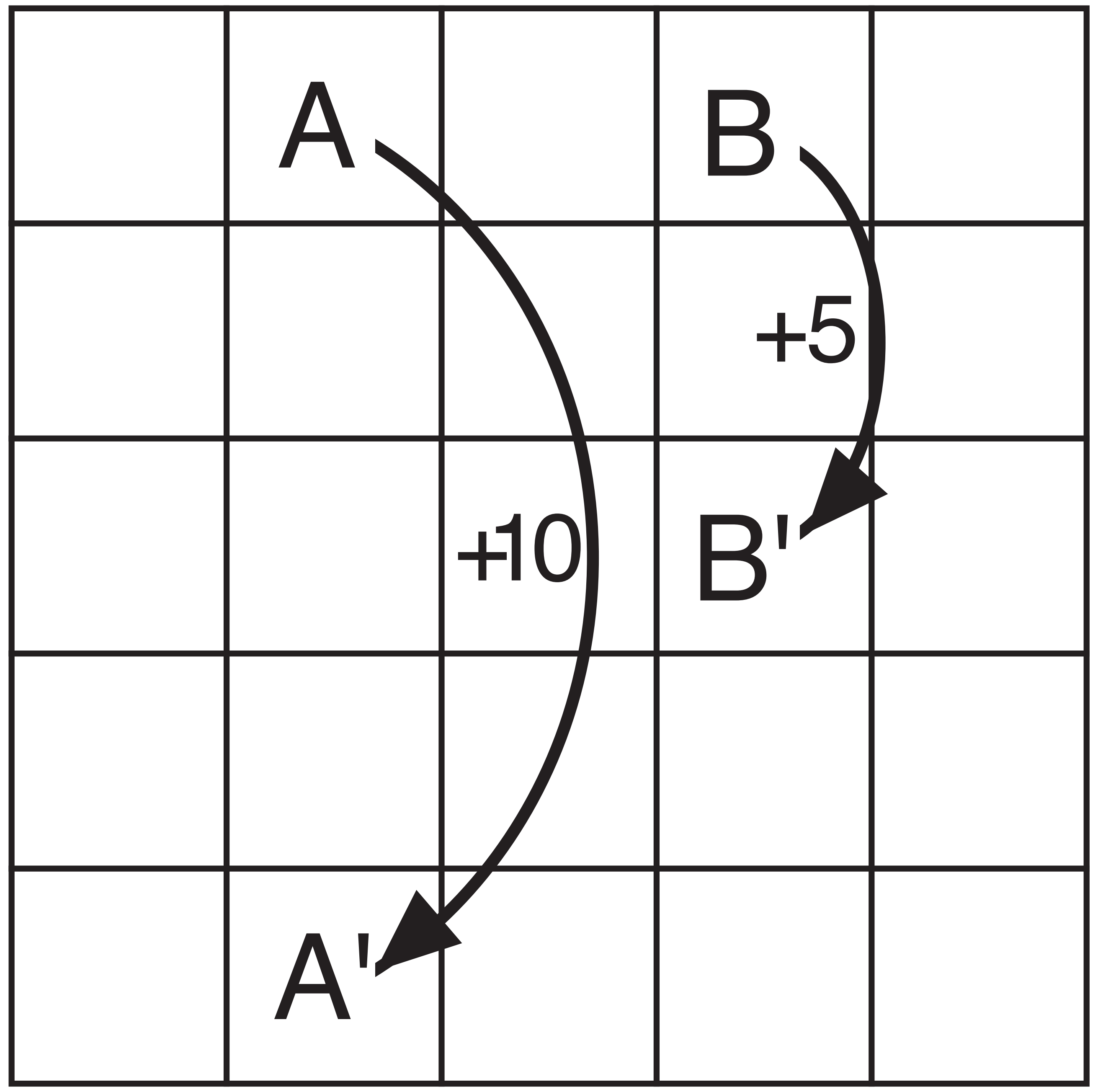
Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
    $\Delta \leftarrow 0$
    Loop for each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \big[r + \gamma V(s') \big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

- The updates are **in-place**: we use new values for $V(s)$ **immediately** instead of waiting for the current sweep to complete (**why?**)
- These are **expected updates**: Based on a weighted average (expectation) of **all possible next states** (**instead of what?**)

Iterative Policy Evaluation

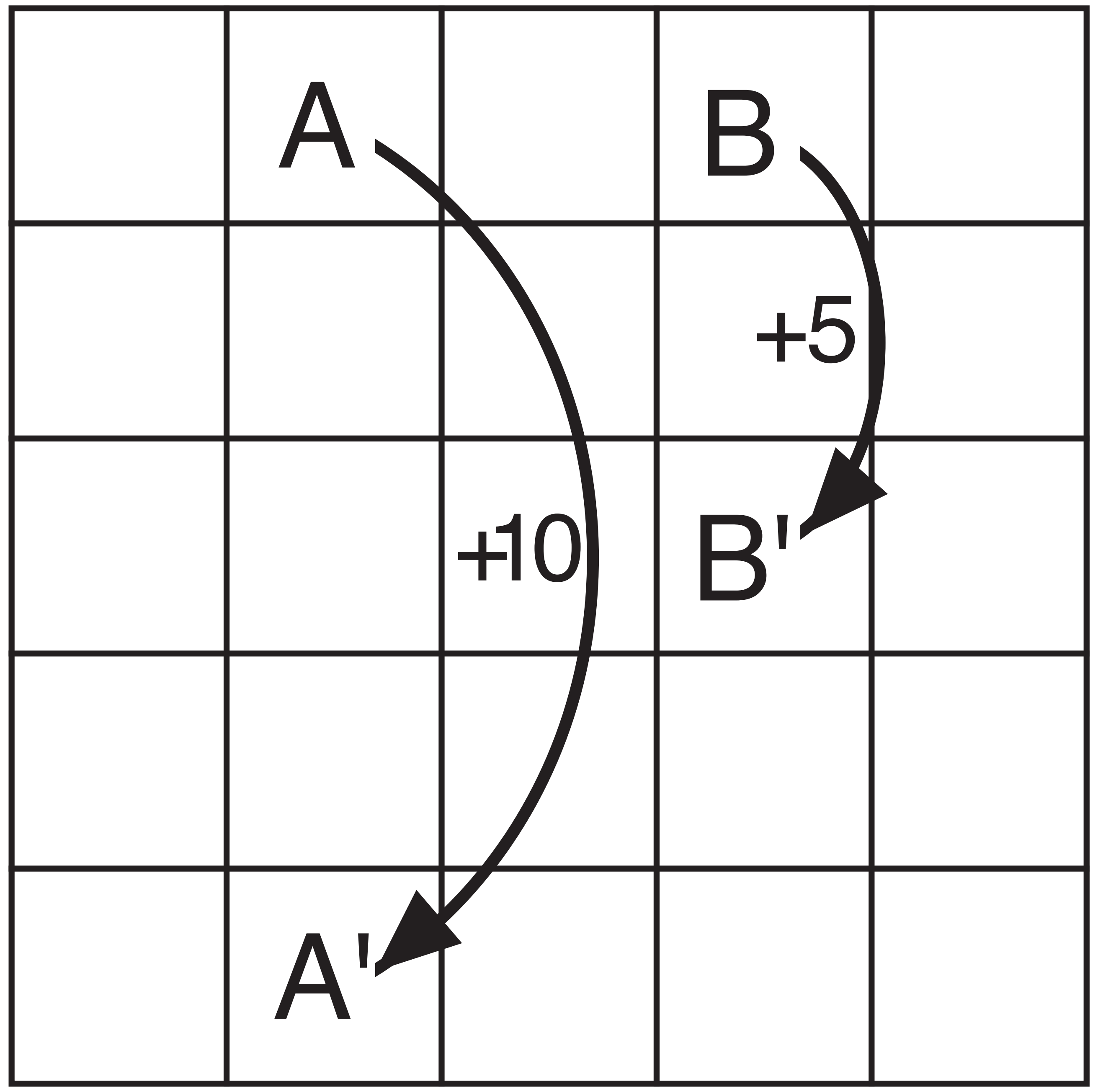


Reward dynamics

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	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0

V at $k = 0$

Iterative Policy Evaluation in GridWorld

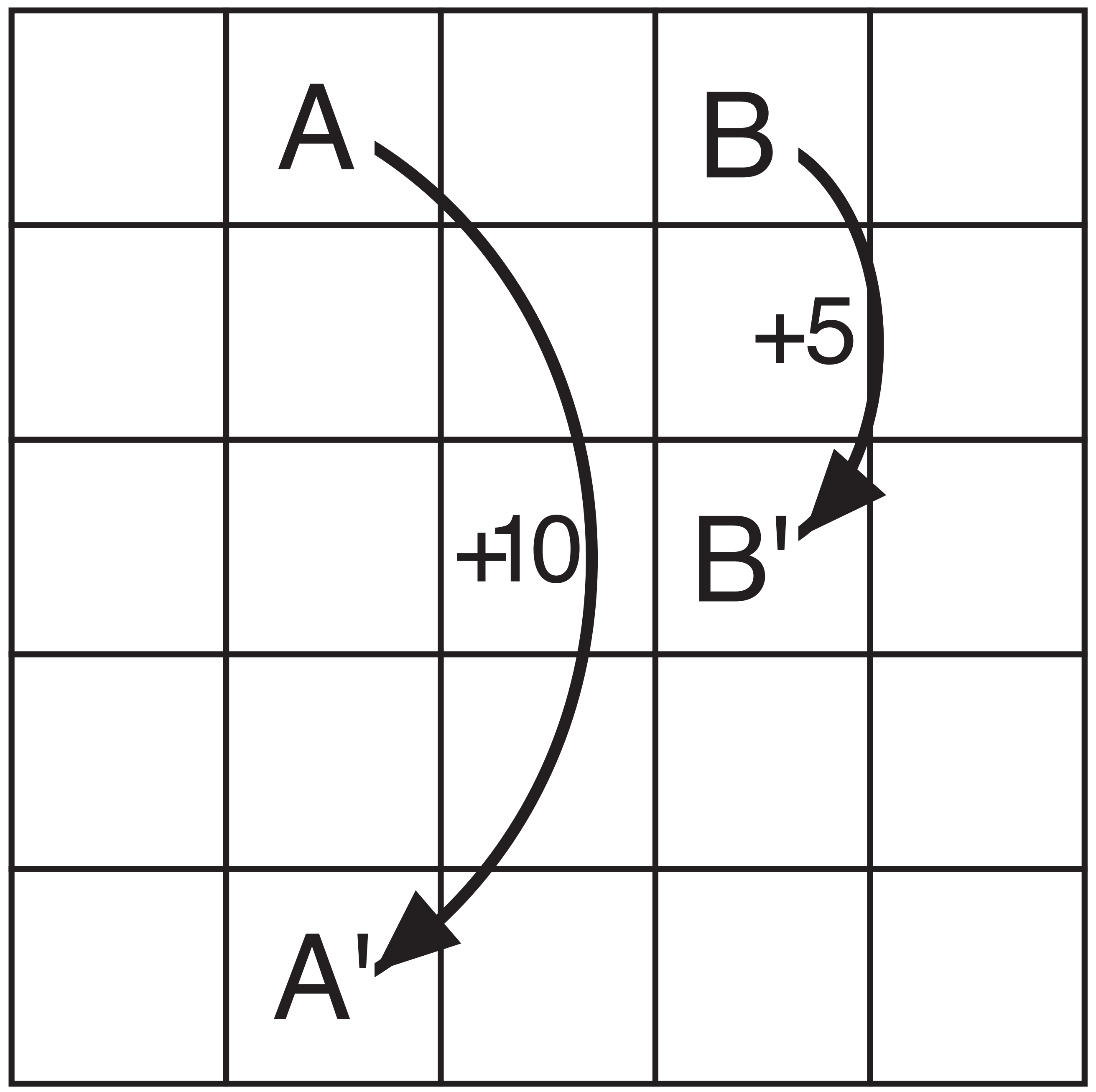


Reward dynamics

-0.5	10	2	5	0.6
-0.3	2.1	0.9	1.3	0.2
-0.3	0.4	0.3	0.4	-0.1
-0.3	0.0	0.0	0.1	-0.2
-0.5	-0.3	-0.3	-0.3	-0.6

V at $k = 1$

Iterative Policy Evaluation in GridWorld

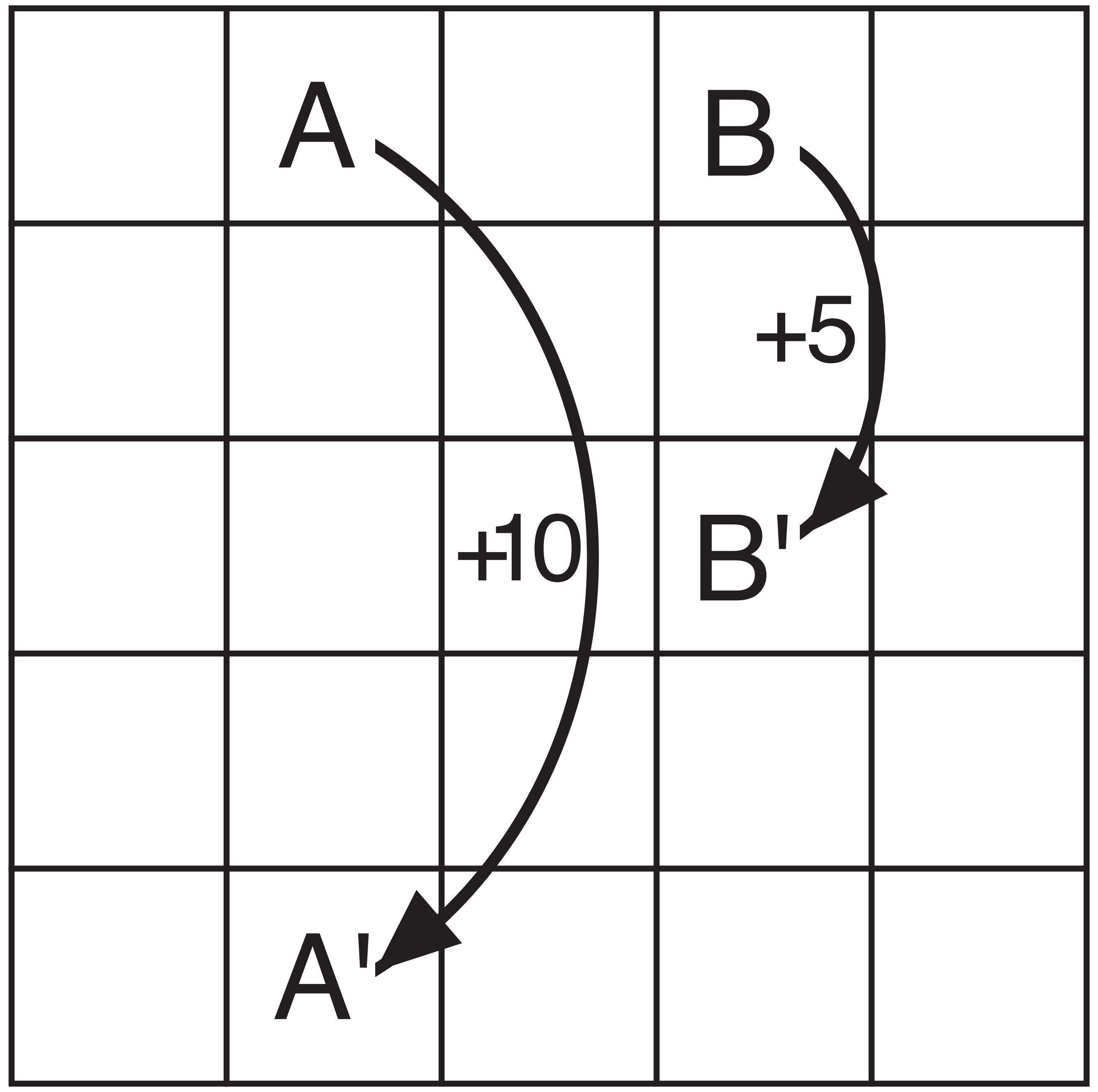


Reward dynamics

1.4	9.7	3.7	5.3	1.0
0.4	2.5	1.8	1.7	0.4
-0.2	0.6	0.6	0.5	-0.1
-0.5	0.0	0.0	0.0	-0.5
-1.0	-0.6	-0.5	-0.5	-1.0

V at $k = 2$

Iterative Policy Evaluation in GridWorld



Reward dynamics

3.4	8.9	4.5	5.3	1.5
1.6	3.0	2.3	1.9	0.6
0.1	0.8	0.7	0.4	-0.4
-1.0	-0.4	-0.3	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

V at $k = 10\,000$

Policy Improvement Theorem

Theorem:
Let π and π' be any pair of deterministic policies.

If $q_\pi(s, \pi'(s)) \geq v_\pi(s) \quad \forall s \in \mathcal{S}$,

then $v_{\pi'}(s) \geq v_\pi(s) \quad \forall s \in \mathcal{S}$.

If you are never worse off **at any state** by following π' for **one step** and then following π forever after, then following π' **forever** has a higher expected value **at every state**.

Policy Improvement Theorem Proof

$v_\pi(s) \leq q_\pi(s, \pi'(s))$

Policy Improvement Theorem Proof

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \Bigm| S_t = s\Big] \\
&\le \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \Bigm| S_t = s\Big] \\
&\vdots \\
&\le \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \Bigm| S_t = s\Big] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Greedy Policy Improvement

Given any policy π, we can construct a new greedy policy π' that is guaranteed to be **at least as good**:

$$\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_\pi(s')\big].
\end{aligned}$$

- If this new policy is **not better** than the old policy, then $v_\pi(s) = v_{\pi'}(s)$ for all $s \in \mathcal{S}$ (**why?**)
- Also means that the new (and old) policies are **optimal** (**why?**)

Policy Iteration

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

This is a lot of iterations!
Is it necessary to run to completion?

Value Iteration

Value iteration interleaves the estimation and improvement steps:

$$v_{k+1}(s) \doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a\right]$$
$$= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_k(s')\right]$$

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Policy Iteration Summary

- An **optimal policy** has higher state value than any other policy **at every state**
- A policy's state-value function can be computed by **iterating** an **expected update** based on the Bellman equation
- Given any policy π, we can compute a **greedy improvement** π' by choosing highest expected value action based on v_π
- **Policy iteration:** Repeat:
  Greedy improvement using v_π, then recompute v_π
- **Value iteration:** Repeat:
  Recompute v_π by assuming greedy improvement at every update

Example: Blackjack

- Player gets two cards, dealer gets 1
- Player can `hit` (get a new card) as many times as they like, or `stick` (stop hitting)
- After the player is done, the dealer hits / sticks according to a fixed rule
- Whoever has the most points (sum of card values) wins
- But, if you have more than 21 points, you **lose immediately** ("bust")

Simulating Blackjack

- Given a policy for the player, it is **very easy** to simulate a game of Blackjack
- **Question:** Is it easy to **compute** the full **dynamics**?
- **Question:** Is it easy to run **iterative policy evaluation**?

Experience vs. Expectation

- In order to compute **expected updates**, we need to know the exact **probability** of **every** possible transition
- Often we don't have access to the full probability distribution, but we do have access to **samples of experience**

1. **Actual experience:** We want to learn based on interactions with a **real environment**, without knowing its dynamics
2. **Simulated experience:** We can **simulate** the dynamics, but we don't have an **explicit representation** of transition probabilities, or there are **too many states**

Monte Carlo Estimation

- Instead of estimating expectations by a **weighted sum** over **all possibilities**, estimate expectation by **averaging** over a **sample** drawn from the distribution:

$$\mathbb{E}[X] = \sum_x f(x)x \approx \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{where } x_i \sim f$$

Monte Carlo Prediction

- Use a large sample of episodes generated by a policy π to estimate the state-values $v_\pi(s)$ for each state s
    - We will consider only episodic tasks for now
- **Question:** What is the return G_t for state $S_t = s$ in a given episode?
- We can estimate the expected return $v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$ by averaging the returns for that state in every episode containing a visit to s

First-visit Monte Carlo Prediction

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

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

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

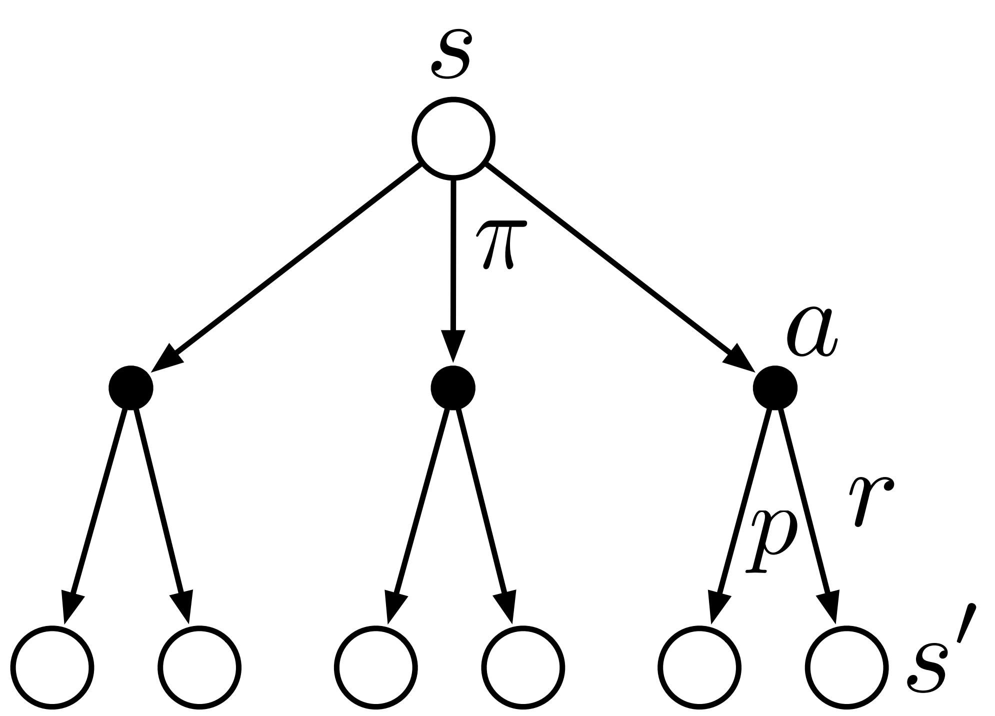
$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

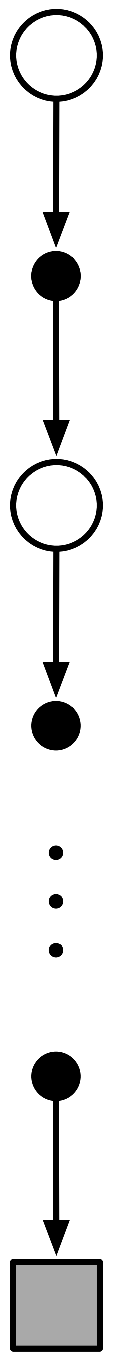
Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Monte Carlo vs. Dynamic Programming



- **Iterative policy evaluation** uses the estimates of the **next state's** value to update the value of this state
  - Only needs to compute a **single transition** to update a state's estimate
- **Monte Carlo** estimate of each state's value is **independent** from estimates of **other states'** values
  - Needs the **entire episode** to compute an update
  - Can focus on evaluating a **subset of states** if desired



Summary

Monte Carlo estimation estimates values by averaging returns over **sample episodes**

- Does not require access to full model of **dynamics**
- Does require access to an entire **episode** for each sample