Temporal Difference Learning

CMPUT 366: Intelligent Systems

S&B §6.0-6.2, §6.4-6.5

Labs & Assignment #3

- Assignment #3 was due **Mar 25 (today)** before lecture
- Today's lab is from **5:00pm to 7:50pm** in **CAB 235**
  - Last-chance lab for late assignments
  - Not mandatory
  - Opportunity to get help from the TAs

Lecture Overview

1. Recap
2. TD Prediction
3. On-Policy TD Control (Sarsa)
4. Off-Policy TD Control (Q-Learning)

Recap: Monte Carlo RL

- **Monte Carlo** estimation: Estimate **expected returns** to a state or action by averaging **actual returns** over **sampled trajectories**

- Estimating **action values** requires either **exploring starts** or a **soft policy** (e.g., ε-greedy)

- **Off-policy learning** is the estimation of value functions for a **target policy** based on episodes generated by a different **behaviour policy**

- **Off-policy control** is learning the **optimal policy** (target policy) using episodes from a **behaviour policy**

Learning from Experience

- Suppose we are playing a blackjack-like game **in person**, but we **don't know the rules**.
    - We know the actions we can take, we can see the cards, and we get told when we win or lose
- **Question:** Could we compute an optimal policy using **dynamic programming** in this scenario?
- **Question:** Could we compute an optimal policy using **Monte Carlo**?
    - What would be the **pros and cons** of running Monte Carlo?

Bootstrapping

	Bootstrapping	No bootstrapping
Learns from experience	TD	MC
Requires full dynamics	DP	

- Dynamic programming **bootstraps**: Each iteration's estimates are based partly on **estimates from previous iterations**
- Each Monte Carlo estimate is based only on **actual returns**

Updates

- Dynamic Programming: $V(S_t) \leftarrow \sum_a \pi(a \mid S_t) \sum_{s',r} p(s', r \mid S_t, a)\left[r + \gamma V(s')\right]$

- Monte Carlo: $V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right]$

- TD(0): $V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] .
\end{aligned}
$$

Monte Carlo: Approximate because of $\mathbb{E}$

Dynamic programming: Approximate because v_Π not known

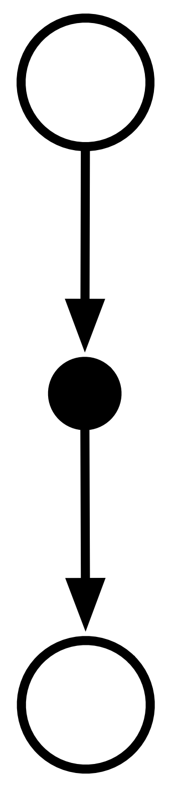
TD(0): Approximate because of $\mathbb{E}$ ***and*** v_Π not known

TD(0) Algorithm

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal



Question: What **information** does this algorithm use?

TD for Control

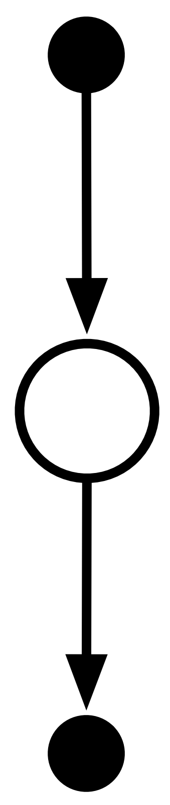
- We can plug TD prediction into the **generalized policy iteration** framework
- Monte Carlo control loop:
   1. Generate an **episode** using estimated π
   2. Update estimates of Q and π
- On-policy TD control loop:
   3. Take an **action** according to π
   4. Update estimates of Q and π

On-Policy TD Control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal



Question: What **information** does this algorithm use?

Question: Will this estimate the Q-values of the **optimal** policy?

Actual Q-Values vs. Optimal Q-Values

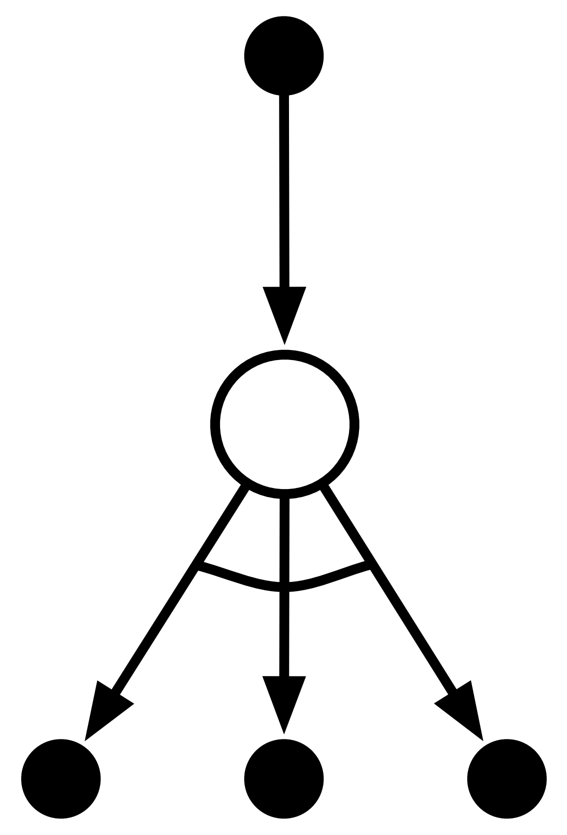
- Just as with on-policy Monte Carlo control, Sarsa does not converge to the **optimal** policy, because it always chooses an **ε-greedy action**
  - And the estimated Q-values are with respect to the **actual actions**, which are ε-greedy
- **Question:** Why is it necessary to choose ε-greedy actions?
- What if we **acted** ε-greedy, but **learned the Q-values** for the optimal policy?

Off-Policy TD Control

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

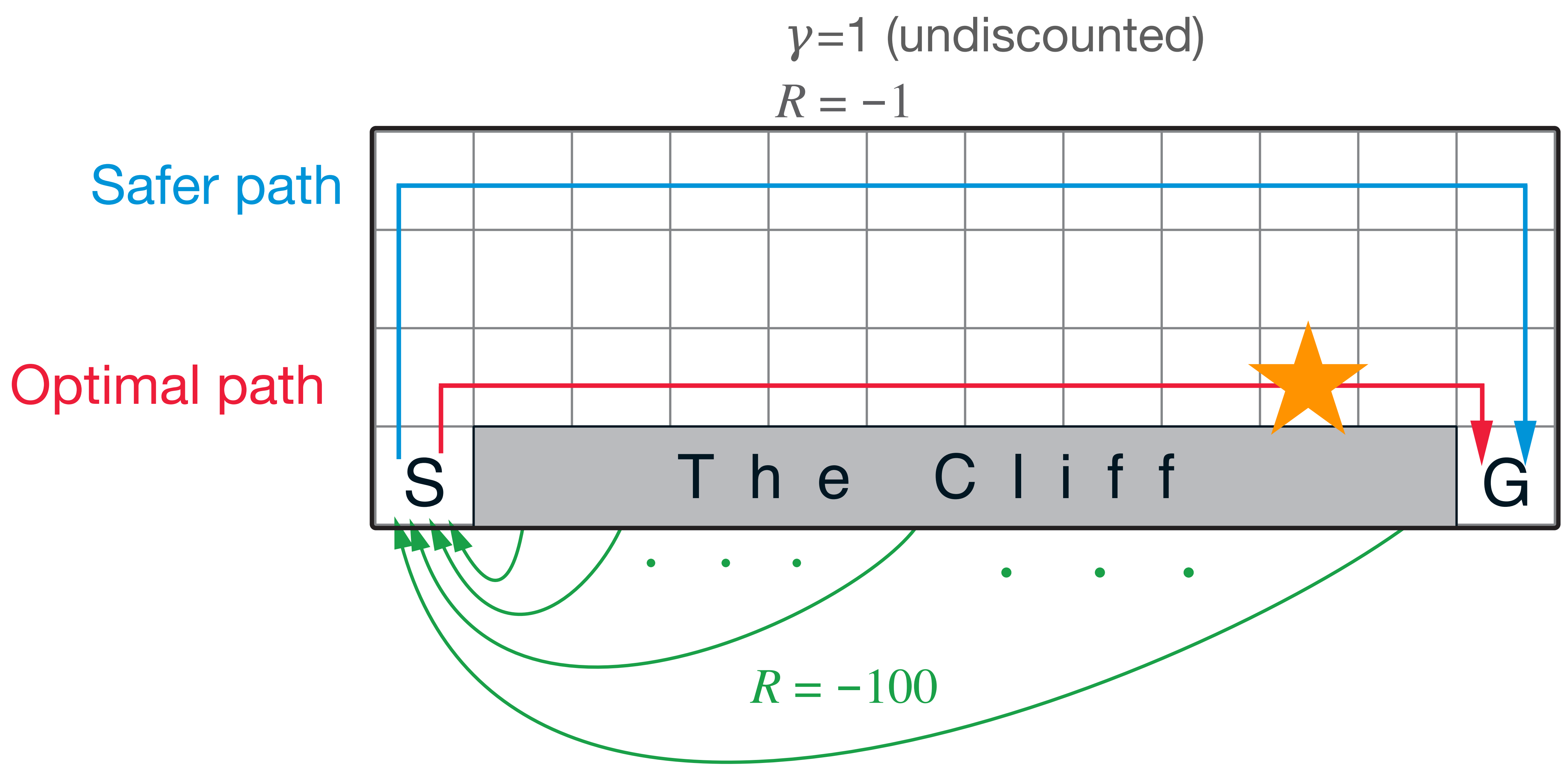
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal



Question: What **information** does this algorithm use?

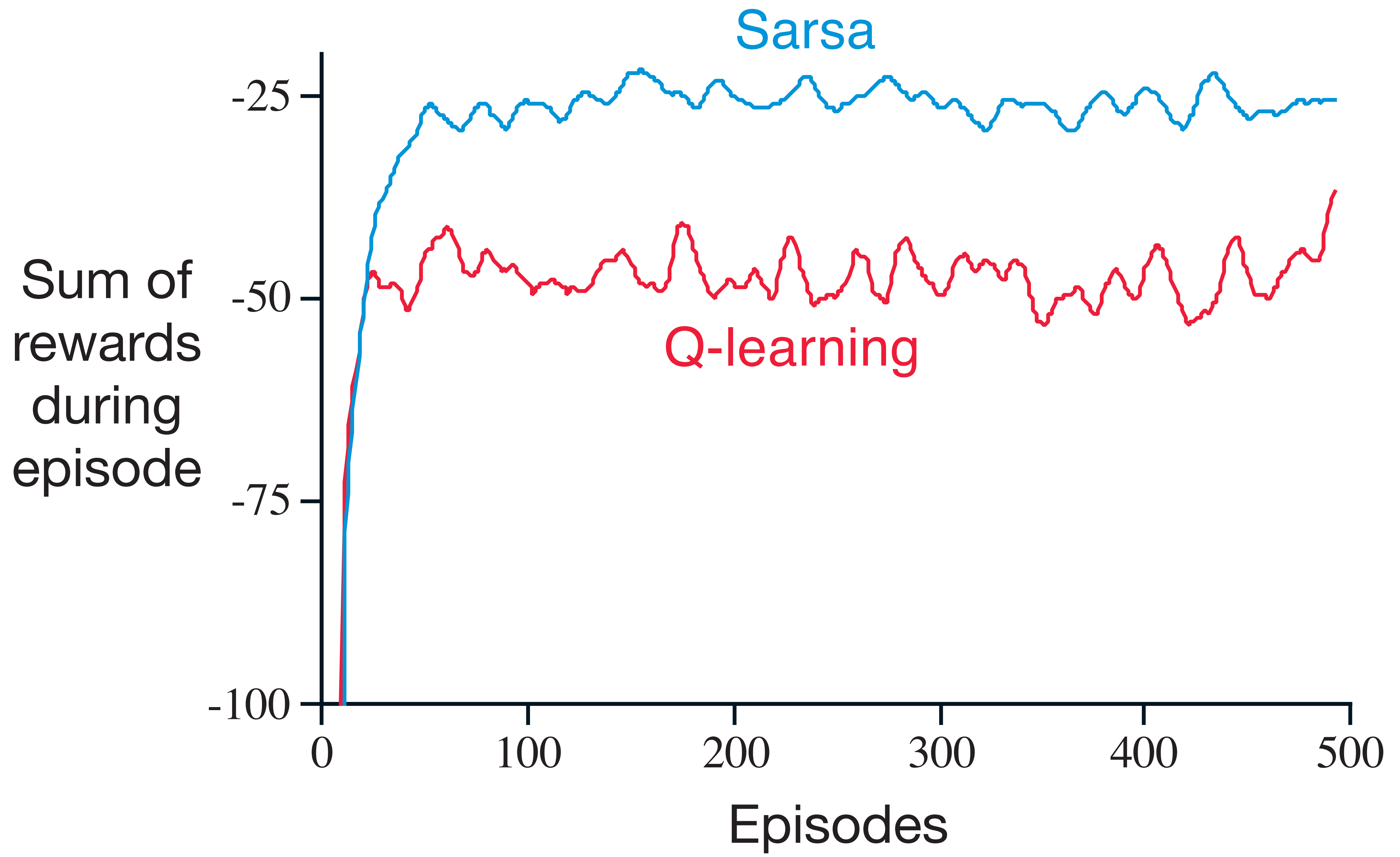
Question: Why aren't we estimating the **policy** π explicitly?

Example: The Cliff



- Agent gets -1 reward until they reach the goal state
- Step into the Cliff region, get reward -100 and go back to start
- **Question:** How will **Q-Learning** estimate the value of **this** state?
- **Question:** How will **Sarsa** estimate the value of **this** state?

Performance on The Cliff



Q-Learning estimates **optimal policy**, but Sarsa consistently **outperforms** Q-Learning. (**why?**)

Summary

- Temporal Difference Learning **bootstraps** *and* learns from **experience**
  - Dynamic programming bootstraps, but doesn't learn from experience (requires full dynamics)
  - Monte Carlo learns from experience, but doesn't bootstrap
- Prediction: **TD(0) algorithm**
- **Sarsa** estimates action-values of **actual ε-greedy policy**
- **Q-Learning** estimates action-values of **optimal** policy while **executing** an **ε-greedy** policy