Bandit Algorithms and Monte Carlo Methods

Rick Valenzano and Sheila McIlraith

Outline

- General assignment questions
- Recap of where we are
- Finish off bandits
  - UCB, incremental averaging, tracking
- Reinforcement Learning
- Monte Carlo methods
  - On-policy prediction and control

Acknowledgements

- Images from the RL book

- Based on slides by David Silver and Adam White

Assignment

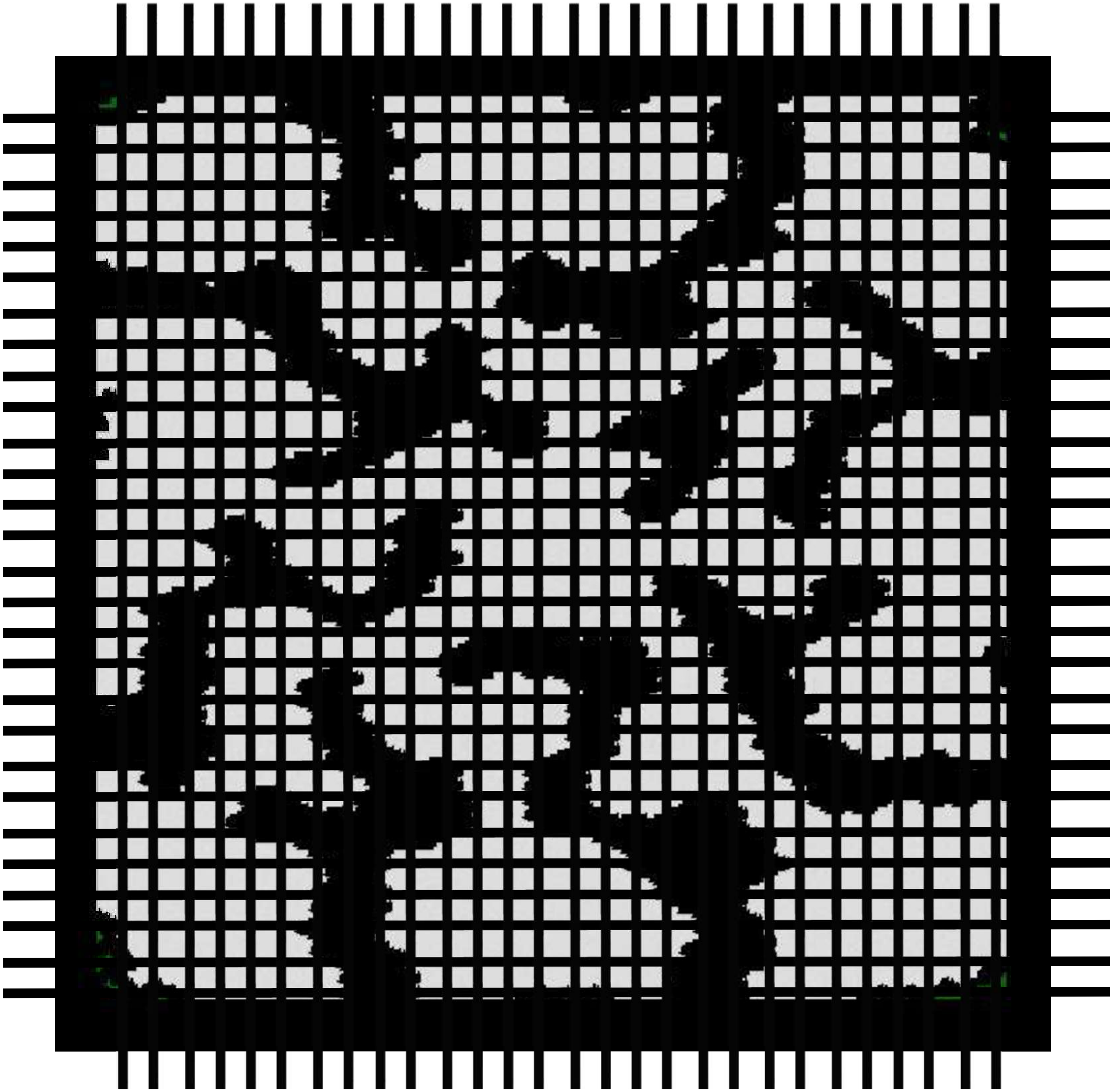
- Grace days
- Questions?

Recap

- Building system to achieve some objective
  - Through some extended interaction with the environment
- Try to model it, then pick an appropriate tool for that model

Recap

- Is the problem deterministic, fully observable, and with a possible finite transition function?
- Try modelling it as a classical planning problem
  - Model it in PDDL and try an off-the-shelf planner
  - Develop a problem specific heuristic, build a system

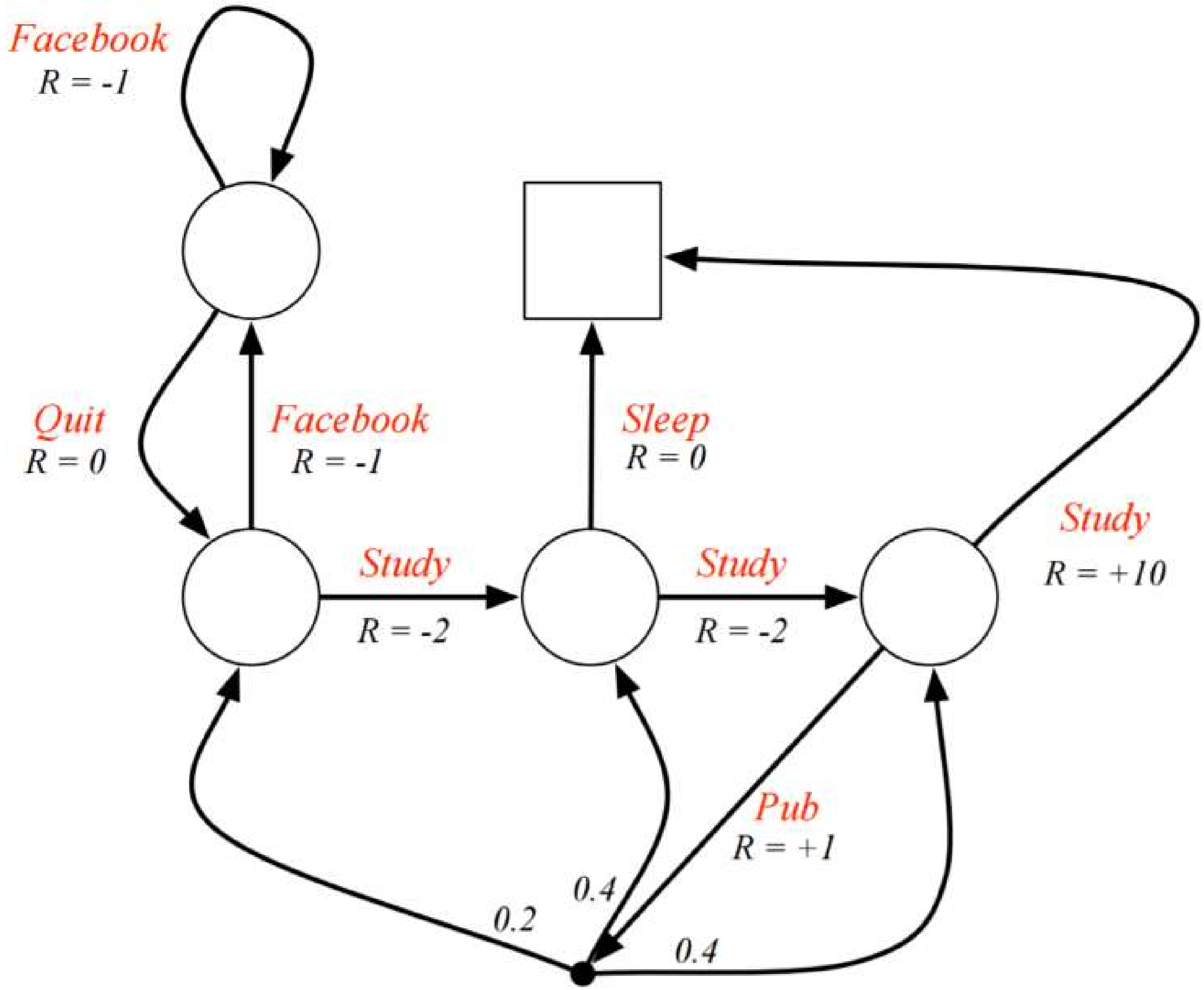


Recap

- Run A*
- Too big for A*? Time constraints?
  - Try suboptimal algorithms like WA* or GBFS
  - Just need to figure out good settings for it

Recap

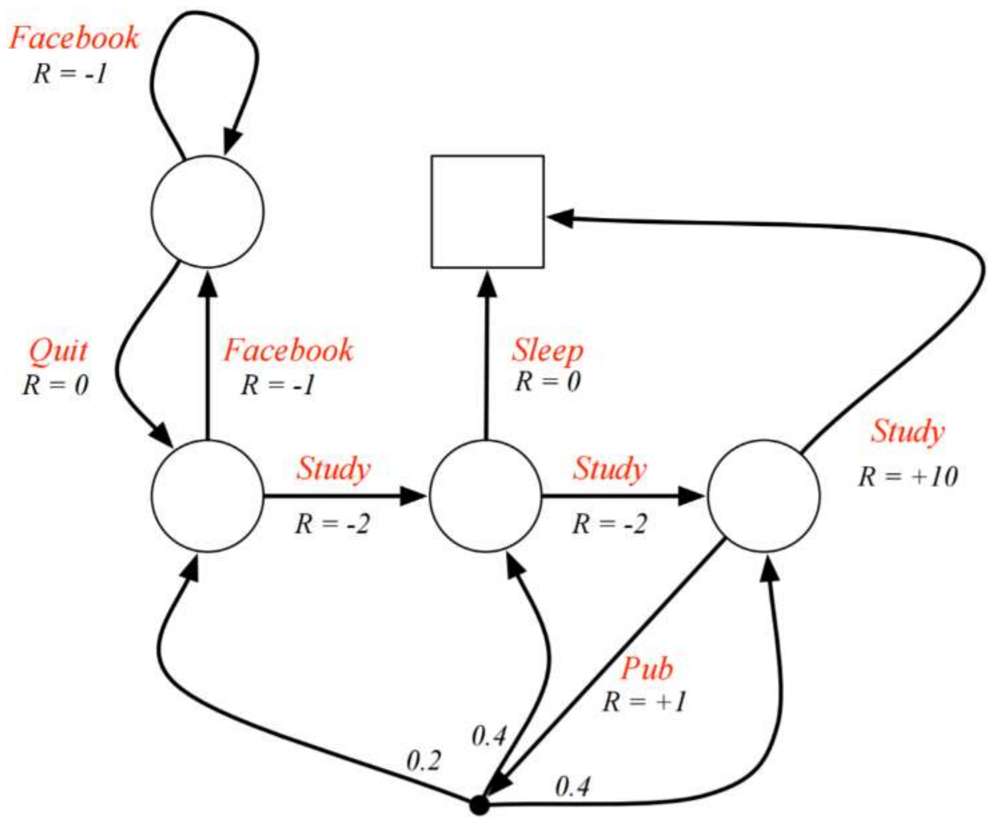
- Fully observable, with discrete and stochastic actions with known transition probabilities?
- Try modelling it as an MDP

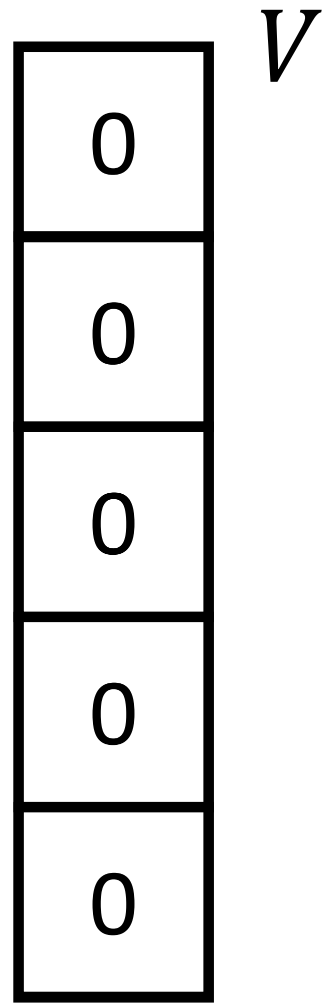

Facebook
R = -1
Quit
R = 0
Facebook
R = -1
Sleep
R = 0
Study
R = -2
Study
R = -2
Study
R = +10
Pub
R = +1
0.2
0.4
0.4

Recap

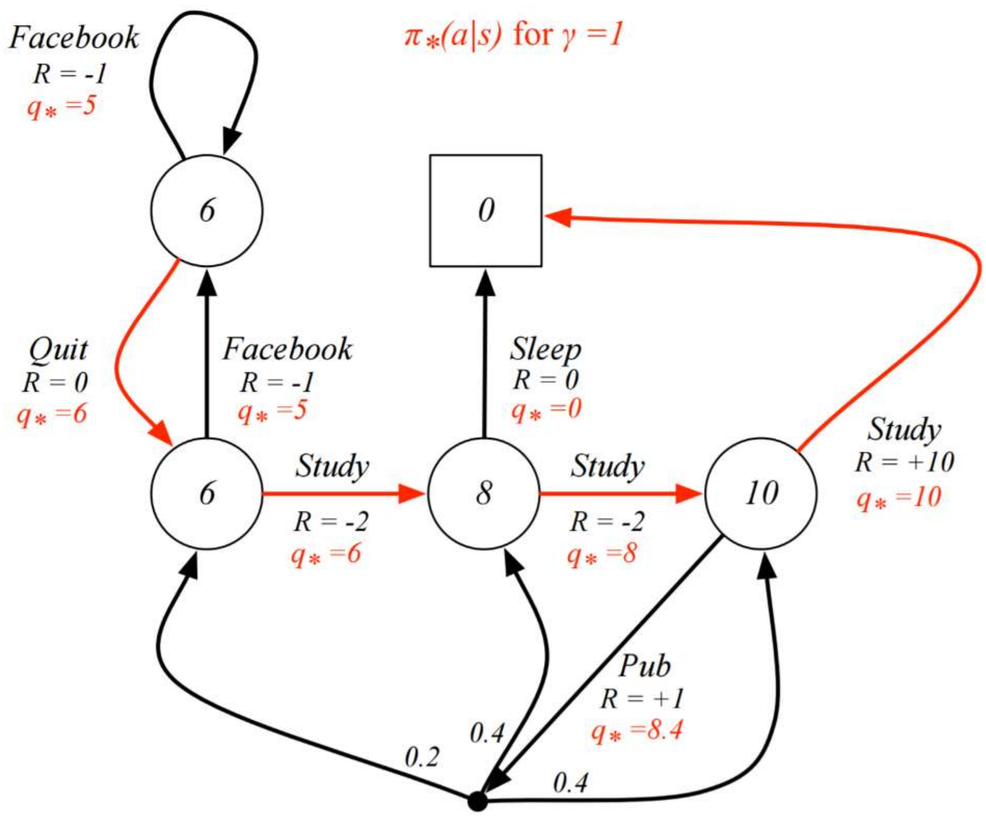
- Model it as an MDP
- Solve using value or policy iteration
- So far, we are assuming we can maintain state value functions in a table
  - **Tabular Methods**

Tabular Methods





Tabular Methods



$$V = v_*$$

6
0
6
8
10

Recap

- To use MDP methods, need a complete model
- But what if we don't know everything?
- What if we don't how reward function works?
  - Single state problem is an MDP
  - **Finish this today**

Preview

- What if the underlying MDP has multiple states?
    - Next two lectures
- What if the state-space is too large?
    - Tabular methods aren’t possible
    - Need approximation methods (next week)

Multi-Armed Bandits

- There are n actions $A = \{a_1, \dots, a_n\}$

- All actions applicable on all of discrete time steps
  - Infinite time steps 1, 2, 3, …
  - On each time step, pick one to execute. Denoted A_t

- $q^*(s, a_i) = q^*(a_i) = \mathrm{E}[R_t | a_i]$

- Agent is trying to maximize total reward over time

Greedy Policy

- Let $Q_t(a_i)$ be the average value of a_i after t steps

- On each step, choose the action with the best average return thus far

$$A_{t+1} = \text{argmax}_{a \in A} Q_t(a)$$

- What are the issues with this approach?

ϵ-Greedy Policy

- Don’t always pick the best looking action
  - May not actually be the best

ϵ-greedy policy:

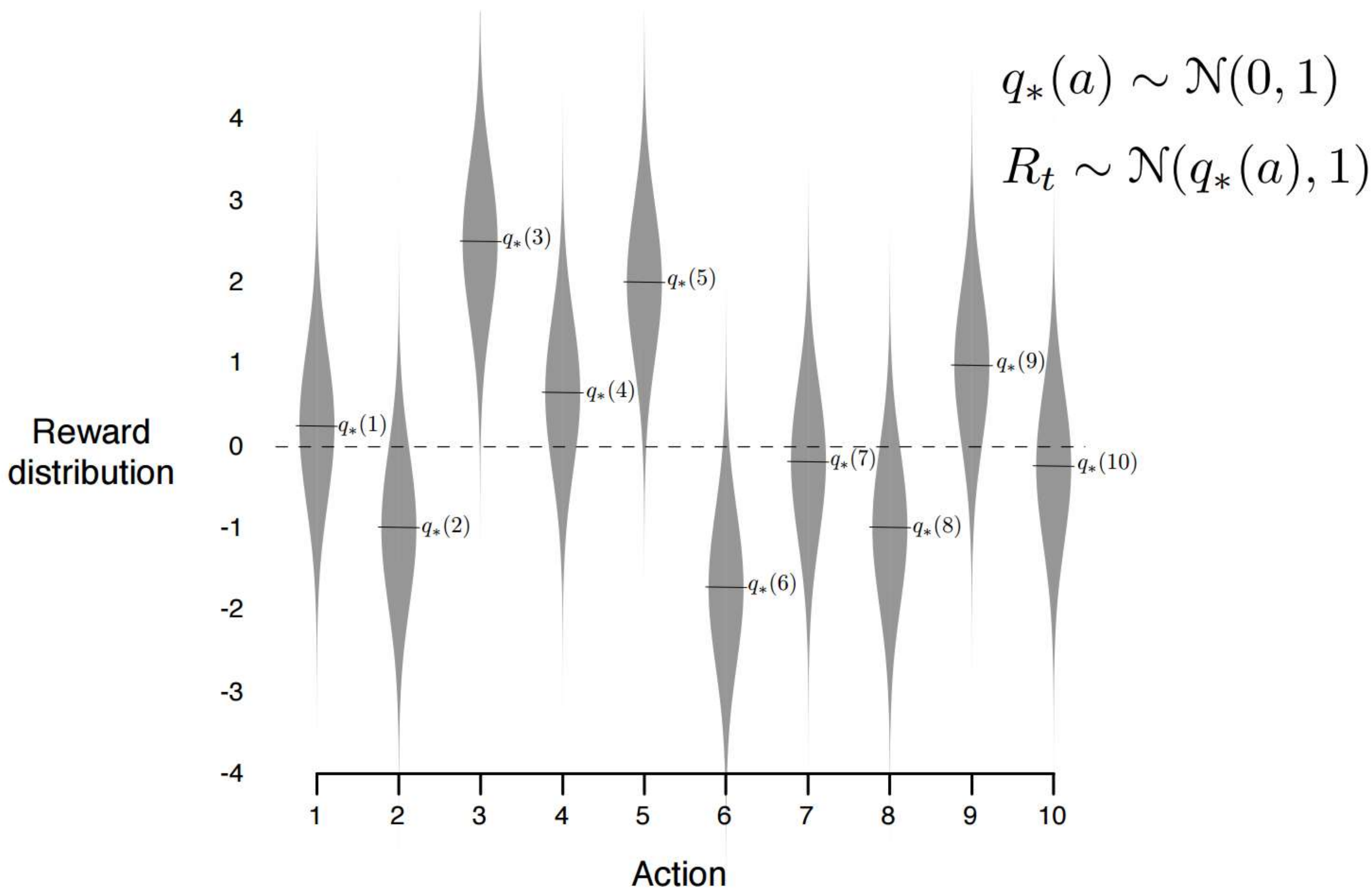
With probability $(1-\epsilon)$:

$$A_{t+1} = \text{argmax}_{a \in A} Q_t(a)$$

With probability ϵ:

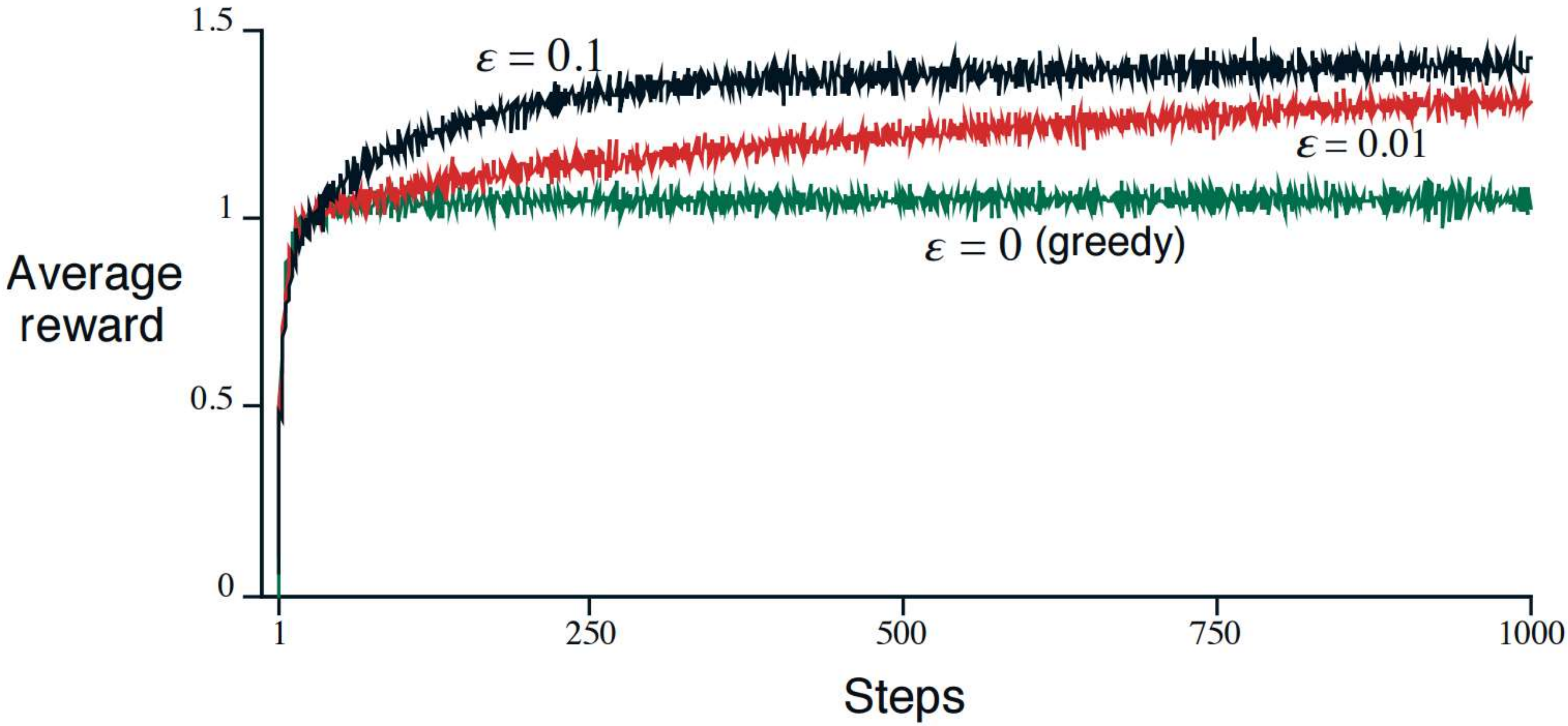
A_{t+1} is selected randomly from A

10-Armed Bandit Testbed

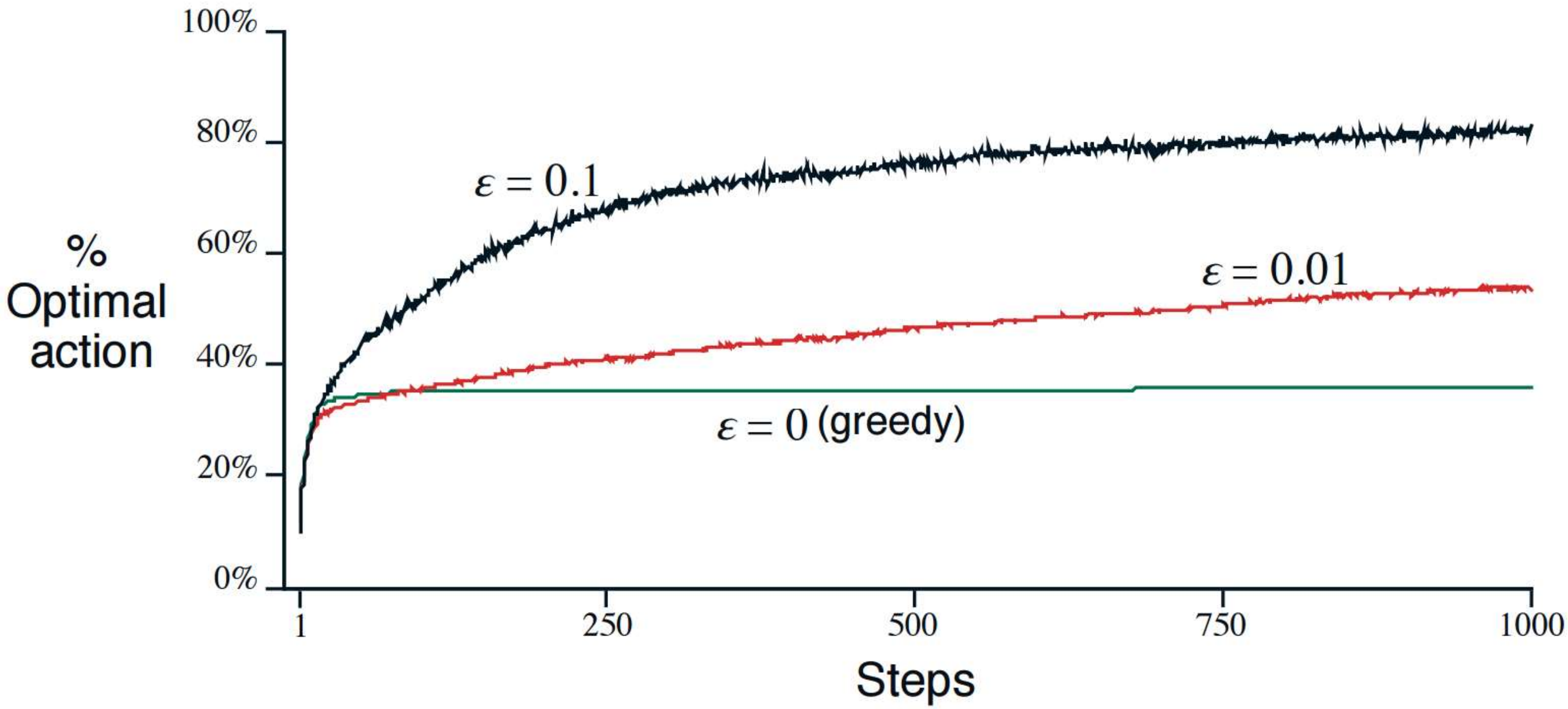


- Made 2,000 such problems

10-Armed Bandit Results



10-Armed Bandit Results



Exploration vs. Exploitation

- When select greedily, agent is **exploiting** its information
- When selects randomly, it is **exploring**
- If we exploit to much, can get stuck with suboptimal values
- If we explore too much, we may be sacrificing a lot of reward that we could have gotten
- Need to balance between the two
  - A central dilemma in reinforcement learning

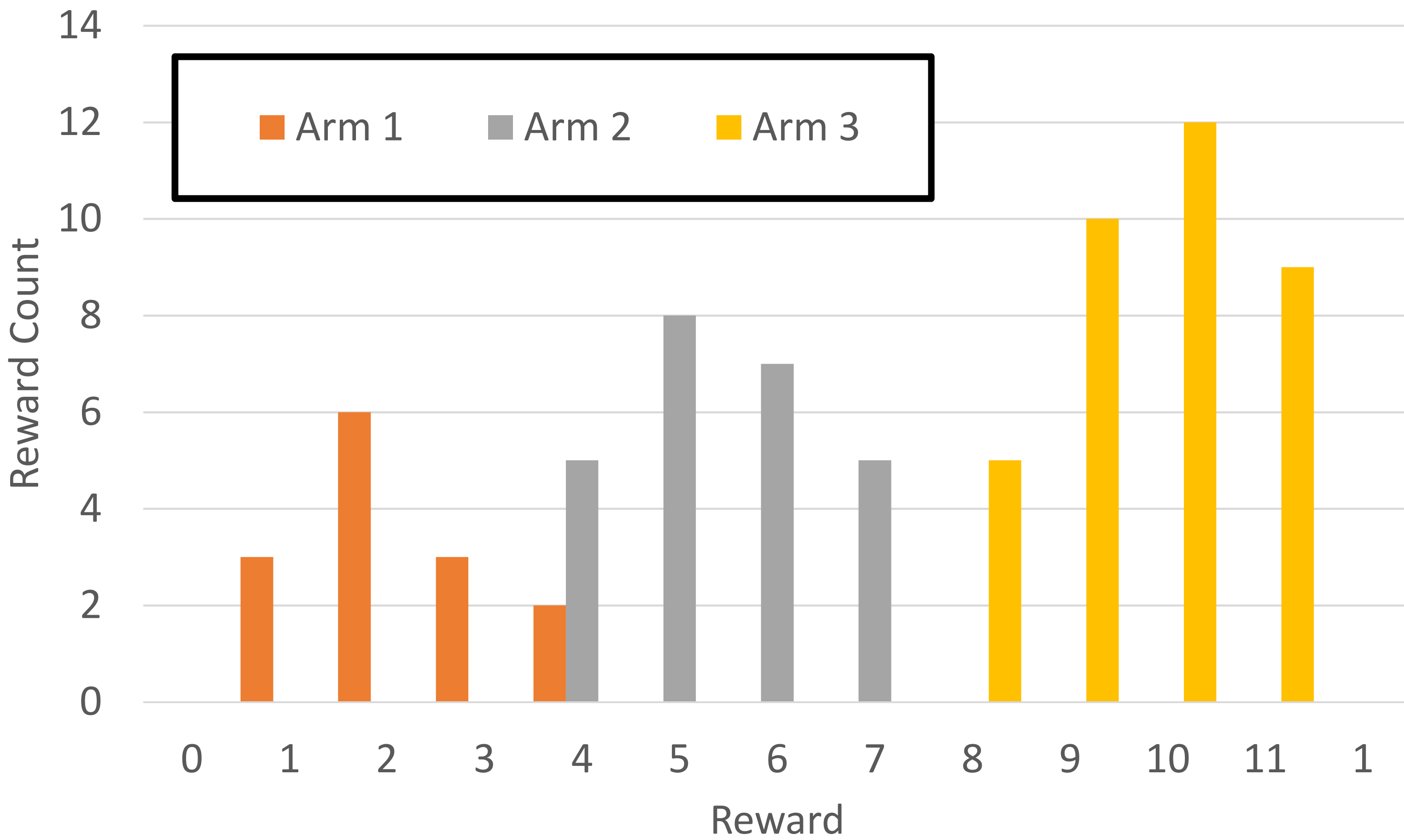
$\boldsymbol{\epsilon}$-Greedy Policy

- Consider case where 10-arms and $\epsilon = 0.1$
  - How often will it select best arm in the limit?

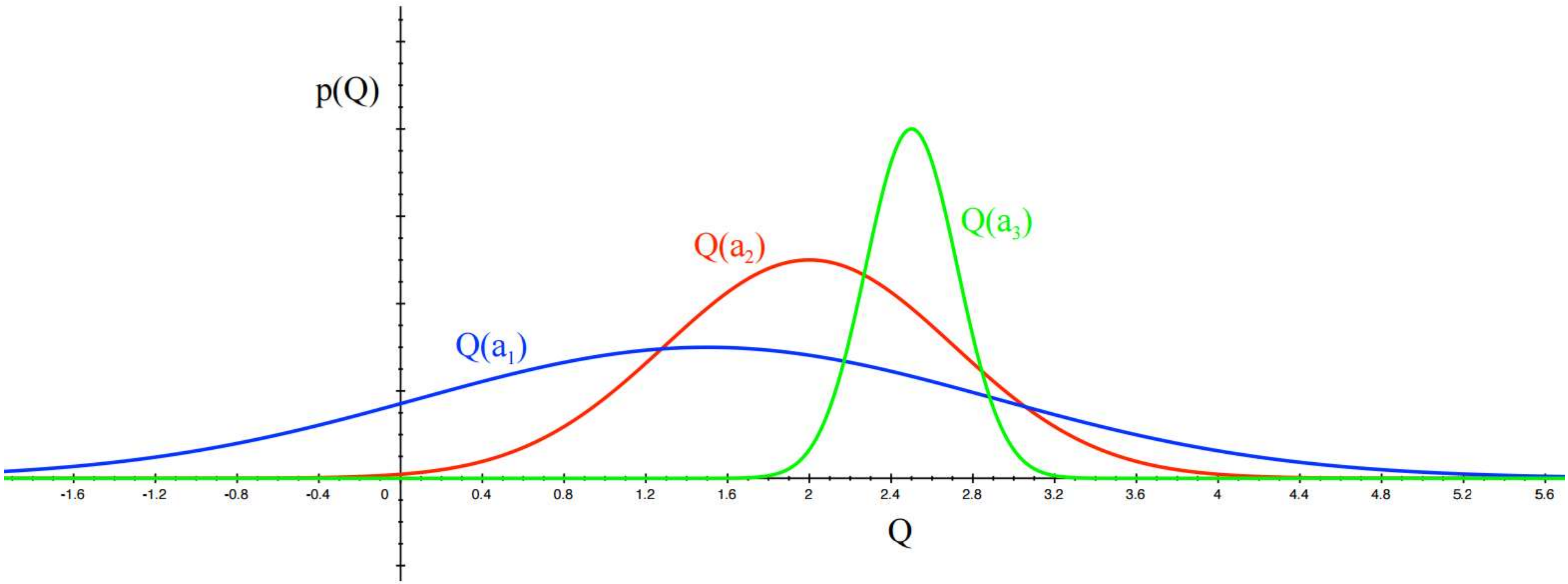
ϵ-Greedy Policy

- Consider case where 10-arms and $\epsilon = 0.1$
  - How often will it select best arm in the limit?
  - With probability 0.91
- Can decrease ϵ over time to converge to right value
  - Must satisfy certain conditions
  - Requires some knowledge about reward function
- Uniform exploration also seems odd

Action-Values



Action-Value Uncertainty



Upper Confidence Bound (UCB)

Estimate a value $U_t(a_i)$ for a_i such that

$$q^*(a_i) \leq Q_t(a_i) + U_t(a_i)$$

with high probability.

UCB Algorithm

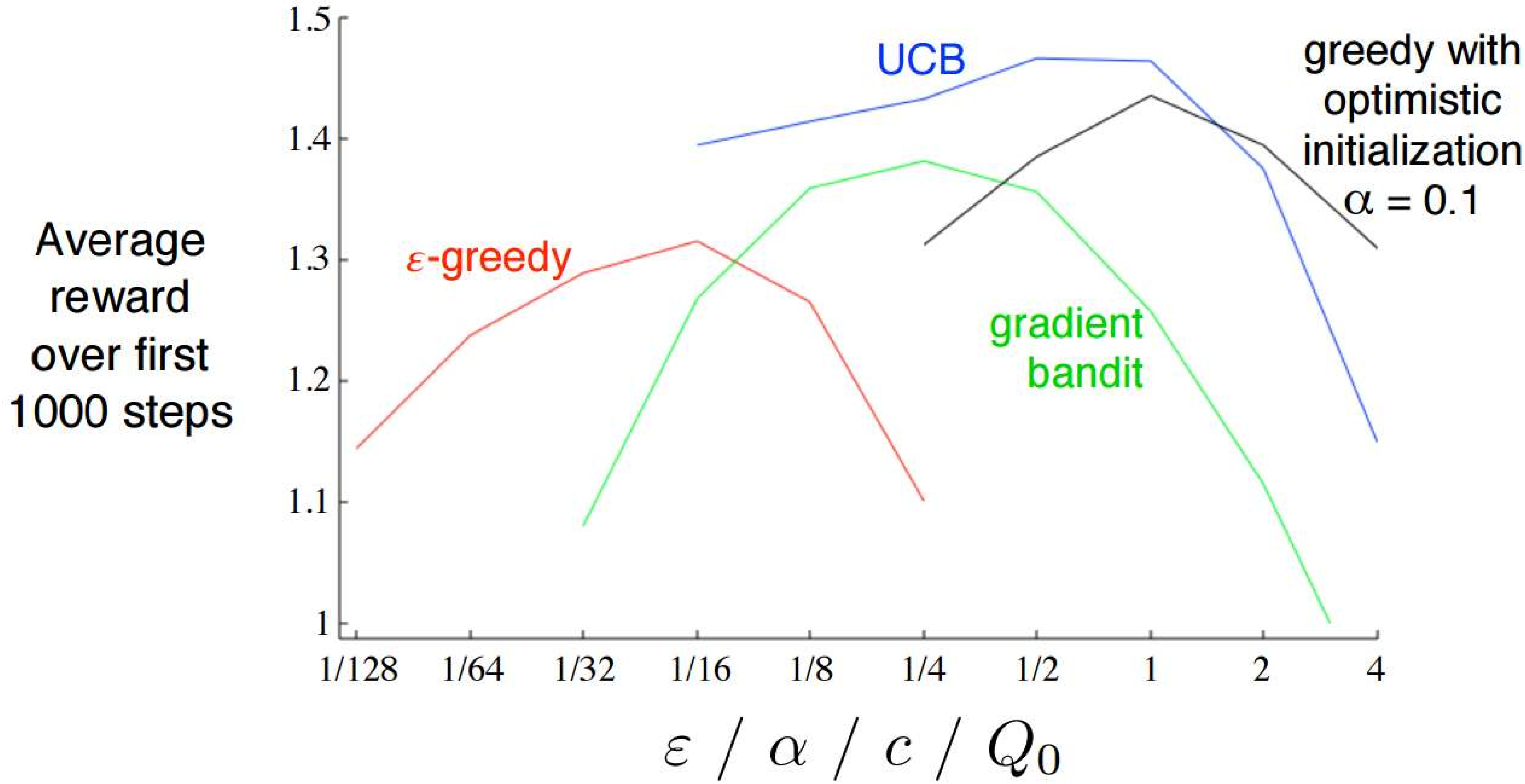
Estimate a value $U_t(a_i)$ for a_i such that

$$A_t = \text{argmax}_{a \in A} \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

$N_t(a)$ is the number of times a was selected

$c > 0$ is a parameter that determines exploration

UCB vs. ϵ-Greedy (and others)



Calculating Average

Standard way to calculate average:

$$Q_t(a) = \frac{1}{N_t(a)} \cdot \sum_{i \in t(a)} R_i$$

In practice, keep track of $N_t(a)$ and the sum

- Some floating point issues

Calculating Average

Standard way to calculate average:

$$Q_t(a) = \frac{1}{N_t(a)} \cdot \sum_{i \in t(a)} R_i$$

- Linear combination of R_t values
  - All rewards have same weight of $1/N_t(a)$
  - Evidence at time $t = 0$ same as at $t = 1{,}000{,}000$

Stationary vs. Non-Stationary

- Assumption that there is an underlying MDP
  - Transition function is not changing
  - **Stationary** problem

- But environments change over time
  - **Non-stationary**
  - Recent evidence is more important

Incremental Average

Incremental way to calculate average:

$$Q_{t+1}(a) = Q_t(a) + \frac{1}{N_t(a)} \cdot (R_{t+1} - q_t(a))$$

In practice, keep track of $N_t(a)$ and q_t(a)

- More robust for floating point arithmetic
- Flexible

Tracking

Use parameter $\alpha \in (0,1]$

$$Q_{t+1}(a) = Q_t(a) + \alpha \cdot (R_{t+1} - q_t(a))$$

If a is always picked, will look like the following:

$$Q_{t+1}(a)$$
$$= (1-\alpha)^t \cdot R_1 + (1-\alpha)^{t-1} + \cdots + \alpha \cdot R_{t+1}$$

Recent evidence is more important

Tracking

Use parameter $\alpha_t(a) \in (0,1]$

$$Q_{t+1}(a) = Q_t(a) + \alpha_t(a) \cdot (R_{t+1} - Q_t(a))$$

Incremental average uses

$$\alpha_t(\alpha) = \frac{1}{t}$$

Tracking

Use parameter $\alpha_t(a) \in (0,1]$

$$Q_{t+1}(a) = Q_t(a) + \alpha_t(a) \cdot (R_{t+1} - Q_t(a))$$

If stationary, Q converge to the true q_* assuming $\alpha_t(a)$ converges to 0 “quickly enough”

State-Value Updates

- Will often use updates of the following

NewEstimate =

LastEstimate + StepSize · (Target – LastEstimate)

- Target is what we want
    - Or an estimate (*i.e.* sample) of what we want

Bandit Recap

- Don’t know the reward function
- Must balance exploit-explore balance
- ϵ-greedy, UCB as solution techniques
- Incremental average calculation
- Stationary vs non-stationary updates

Beyond Bandits

- What if there are more than one state?
- What if we don't even know the transition function?
- For this class, we will at least assume we know the state-space

Reinforcement Learning

- Learn from interacting with environment
  - Could be an actual environment (like a robot)
  - Or a partially specified model
- Take an action, get a reward, and new state
- Learning to map situations to actions (policy)
  - But without a full model
  - Still trying to maximize the reward

Reinforcement Learning

- Learner (agent) not told how to act
  - No teacher, no labels on training examples
  - At least in “pure” form

- “Trial and error” learning

RL Tabular Methods

- Let's assume we can enumerate all possible states
- Can figure out applicable actions in any state
  - Just don't know resulting reward, or even transition is
- Just as in DP, consider two problems:
  1. **Prediction/Evaluation**: how well will a policy do?
  2. **Control**: find a good policy

Monte Carlo Methods

- Estimate value function using sample episodes

	V
s_1	7.5
s_2	3
s_3	-1.2
s_4	0.5
s_5	11

or

		Q
s_1	a_1	10
	a_2	5
s_2	a_1	3
	a_2	0
s_3	a_1	-2.0
	a_2	0
s_4	a_1	0
	a_2	1
s_5	a_1	11
	a_2	11

Monte Carlo Methods

- Estimate value function using sample episodes
- Suitable for **episodic tasks**
  - No matter what actions we take, episode will terminate at some point (could take a long time though)
- Will update on an episode-by-episode basis
  - Not action-by-action

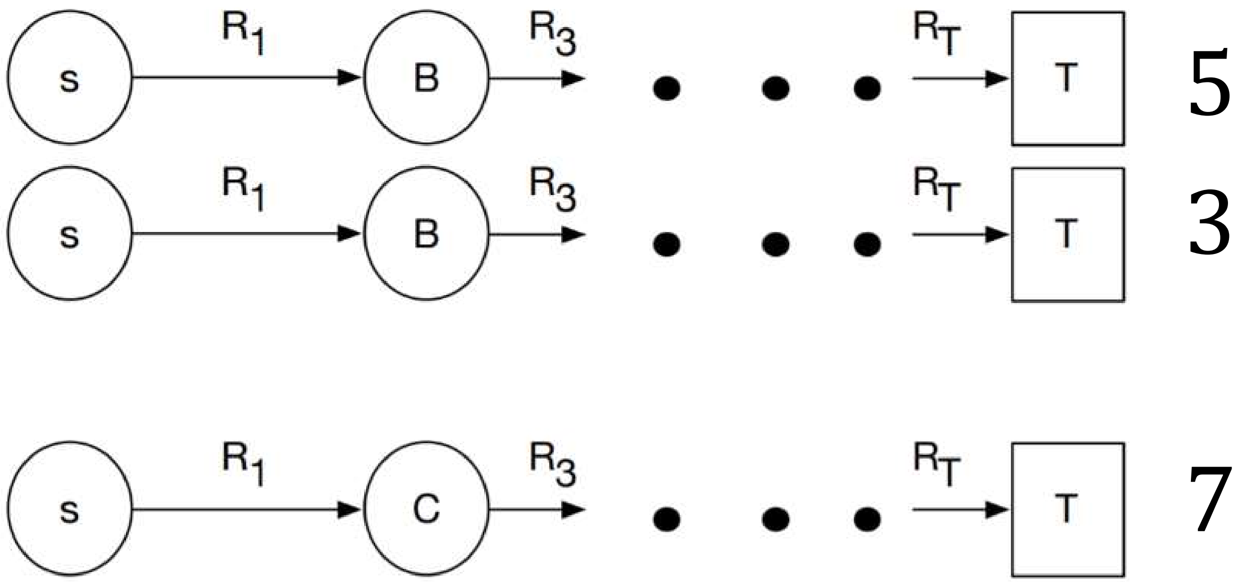
Monte Carlo Prediction

- Recall that G_t is the return we get during an episode after time t

$$v_\pi(s) = \mathrm{E}_\pi[G_t | S_t = s]$$

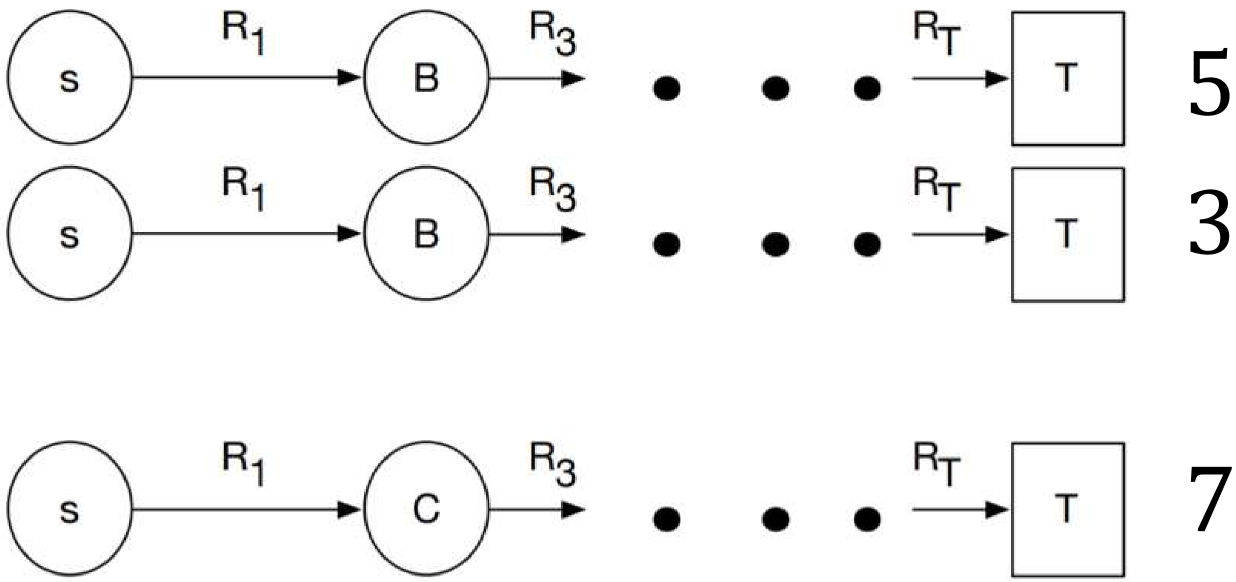
- To estimate $v_\pi(s)$, run episodes starting in s that use policy π, and average returns seen

Monte Carlo Prediction



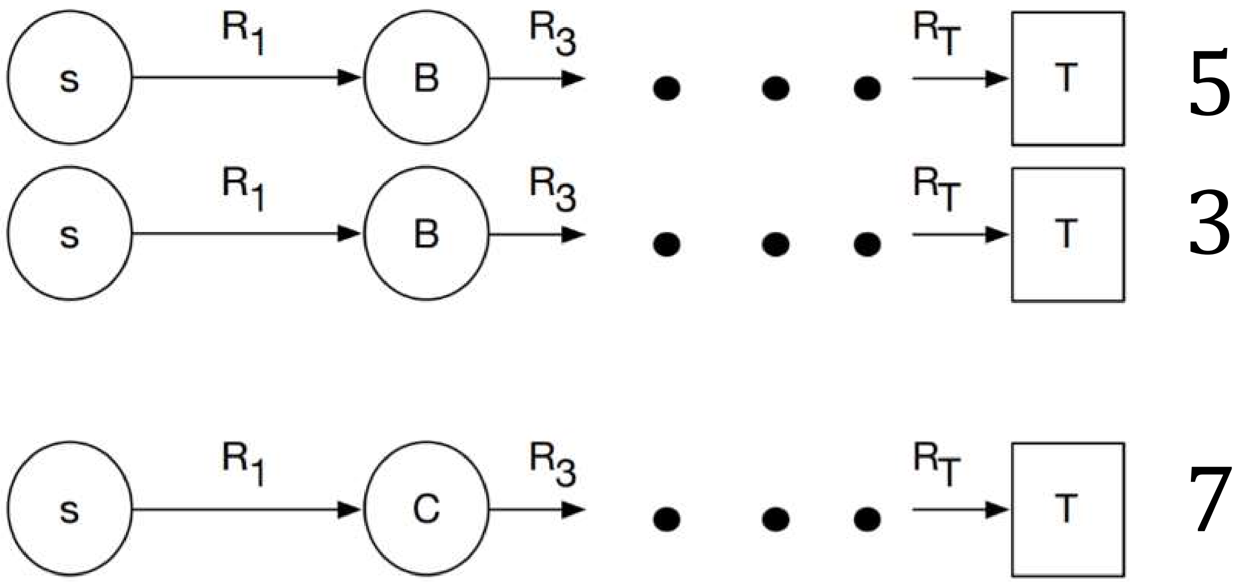
- After three episodes, estimate is …

Monte Carlo Prediction



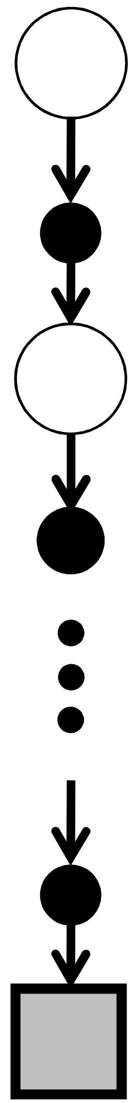
- After three episodes, estimate is … $V_\pi(s) = 5$

Monte Carlo Prediction



- Also have two episodes where B was visited
  - Can update those as well
  - Every episode allows for updates to every visited state

Monte Carlo Backup



First-Visit Monte Carlo Prediction

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

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

Repeat forever:

Generate an episode using π

For each state s appearing in the episode:

$G \leftarrow$ return following the first occurrence of s

Append G to $Returns(s)$

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

Monte Carlo Prediction

- First-visit only updates a state at most once per episode
  - Even if there is “loopy” behaviour
- Can also do **every-visit** where we update for all visits to the state
- Both techniques converge to $v_\pi(s)$ for s if s is visited infinitely often in the limit
  - May need exploration to guarantee this

Exploring Starts

- If π is deterministic, only try one action per state
  - May never reach many states if start in the same place

- **Exploring starts** ensure all states are visited infinitely often in order to guarantee convergence

- Sample episodes such that we start in every state infinitely often

Evaluating State-Action Pairs

- Recall that in DP, could compute $q_\pi(s, a)$ using a lookahead of v_π to the possible transitions
  - Lookahead weighted by the probability of each outcome
  - Needed to know transition probabilities

- Now we don't have transition probabilities
  - Often want q_π since we make decisions based on it
  - So we usually explicitly compute q_π instead of v_π

Evaluating State-Action Pairs

- Can modify first-visit and every-visit MC to update state-action pairs instead
  - Both converge if every pair is visited infinitely often
- Exploring starts for state-action pairs
  - Start with every state-action pair infinitely often

MC vs. DP

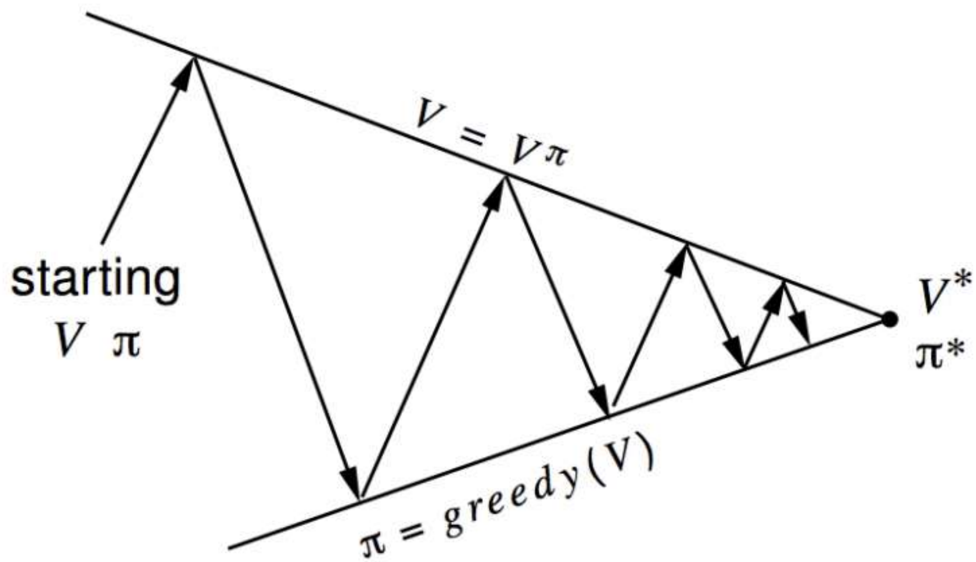
- In MC, only look at outcome that happened
  - Don't look at all outcomes like in DP
  - But means we require exploration
- Do not **bootstrap** in MC
  - DP updates v_π estimates based on other v_π estimates
  - MC only updates based on returns
- Time to update a state in MC does not depend on the number of states or even transitions
  - No **sweeping** like in DP

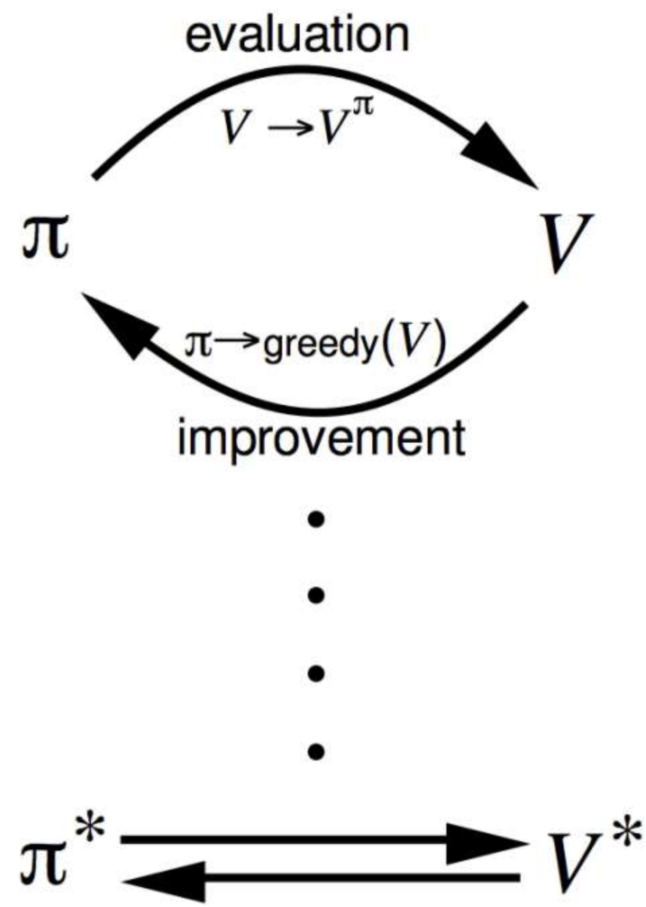
RL Control

- Coming up with a good policy
  - Without model, must do it through experience

- Techniques can be viewed as instances of **Generalized Policy Iteration**

Generalized Policy Iteration





Policy evaluation Estimate v_π

e.g. Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

e.g. Greedy policy improvement

Policy Improvement Theorem

If we have computed v_π, and $q_\pi(s,a) > v_\pi(s)$, then the policy that chooses a will be better.

Or

$$\pi_{k+1} \geq \pi_k$$

if and only if

$$\forall s, q_{\pi_k}\big(s, \pi_{k+1}(s)\big) \geq v_{\pi_k}(s)$$

Monte Carlo Control

- Alternate between policy evaluation and greedy policy improvement

- Just as in DP, don't need to do full evaluation before improvement
  - Can stop if changes are small
  - Or even just after each episode

Monte Carlo Control with ES

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s, a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s, a) \leftarrow$ empty list

Repeat forever:
  Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
  Generate an episode starting from S_0, A_0, following π
  For each pair s, a appearing in the episode:
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s, a)$
    $Q(s, a) \leftarrow \text{average}(Returns(s, a))$
  For each s in the episode:
    $\pi(s) \leftarrow \arg\max_a Q(s, a)$

Exploration for MC Control

- Exploring starts is a strong requirement
  - Not realistic in some cases (like a robot)

- But need to visit every pair to ensure convergence

- Can do this with **soft** policies
  - $\pi(a|s) > 0$ for all s and a

- ϵ-greedy ensures this

Exploration for MC Control

- Exploring starts is a strong requirement
  - Not realistic in some cases (like a robot)

- But need to visit every pair to ensure convergence

- Can do this with **soft** policies
  - $\pi(a|s) > 0$ for all s and a

- ϵ-greedy ensures this
  - Any action sequence of length d starting in s has $\geq \epsilon^d$ probability of happening

MC Control with ϵ-Soft Policies

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s, a) \leftarrow$ arbitrary
  $Returns(s, a) \leftarrow$ empty list
  $\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
  (a) Generate an episode using π
  (b) For each pair s, a appearing in the episode:
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s, a)$
    $Q(s, a) \leftarrow$ average$(Returns(s, a))$
  (c) For each s in the episode:
    $A^* \leftarrow \arg\max_a Q(s, a)$
    For all $a \in \mathcal{A}(s)$:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$$

GPI and $\boldsymbol{\epsilon}$-Greedy Policies

- "Greedy" policy improvement, will now result in another ϵ-greedy policy

- Will now converge to the optimal ϵ-greedy policy
  - Like finding the optimal policy in a new domain where don't always get the action that you want

GPI and ϵ-Greedy Policies

- This control approach is an example of **on-policy learning**
  - Using the learnt policy to generate episodes

- Also have **off-policy learning** techniques
  - Use a **behaviour policy** to generate episodes
  - Learning the **target** policy

Off-Policy Learning

- Behaviour policy is exploratory, ensures that all state-action pairs are tried

- Target policy can be deterministic
  - Means convergence is possible to optimal policy
  - Not just optimal ϵ-soft policy

Off-Policy Learning

- Roughly learn less per episode
- Can only learn from parts of the episode where the behaviour and target policies coincide
  - Or at least, the degree that they are similar
- Can be slower, and harder to generalize

Why Use Off-Policy Learning?

- Can do massively parallel learning
  - Learning about multiple policies at once
- Can learn multiple policies at once
- Can use episodes provided by a human
  - Demonstrating good behaviour
- Learning from batch of episodes

Monte Carlo Recap

- RL is learning from experience
  - Trial and error
- Monte Carlo methods predict using what happened
  - Based on episode results, does not bootstrap
- Need to ensure everything is trying often enough
- Monte Carlo control using GPI in a greedy way
  - On-policy learns best ϵ-soft policy
  - Off-policy can learn best policy, but “learns less” per epsiode