

CSE 190 – Intro to Deep RL

Classical Control, Pre-deep Learning

Prithviraj Ammanabrolu

Thanks to David Silver's DeepMind RL Course and Sheila McIlraith's Planning Course at UofT. Some slides were adapted from there.

Logistics

For project proposals

- What task you're doing
- What env you intend to use? Are you making a sim yourself?
 - What are time estimates for how hard that would be
- Why is this interesting? If you make an agent in this env, who will care?
- Initial ideas on how to solve it (what will you do if X LLM doesn't work?)

Generally speaking the presentation will not be graded, but we will give feedback and expect a revised proposal slide deck to be submitted by the end of the week. That will be graded.

Examples: <https://shorturl.at/9Z8oY>

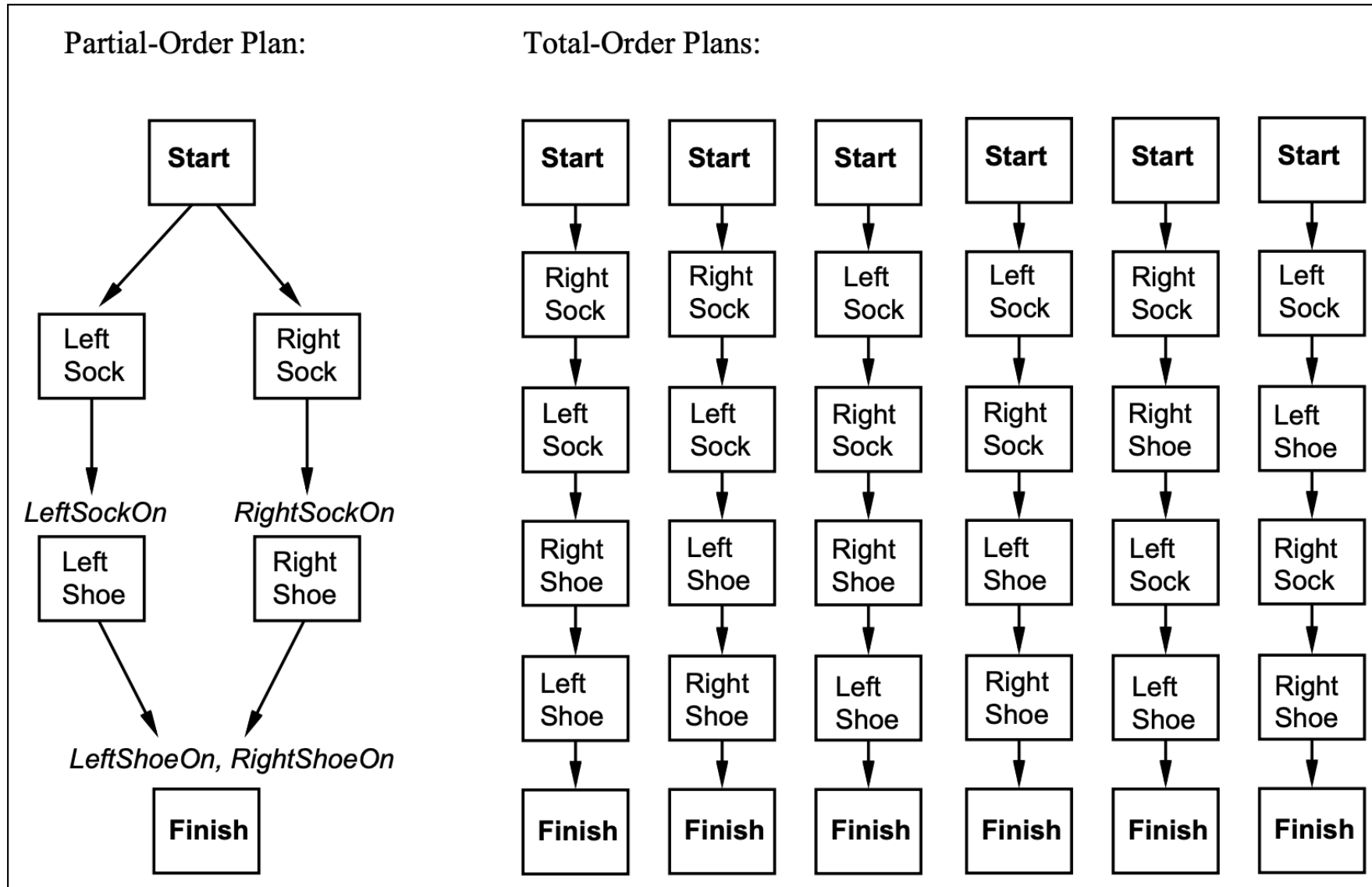
Forward Search

- Some deterministic implementations of forward search:
 - breadth-first search
 - depth-first search
 - best-first search (e.g., A^*)
 - greedy search
- Breadth-first and best-first search are sound and complete But they usually aren't practical, requiring too much memory
 - Memory requirement is exponential in the length of the solution
- In practice, more likely to use depth-first search or greedy search
 - Worst-case memory requirement is linear in the length of the solution
 - In general, sound but not complete
 - But classical planning has only finitely many states
 - Thus, can make depth-first search complete by doing loop-checking

Backward Search

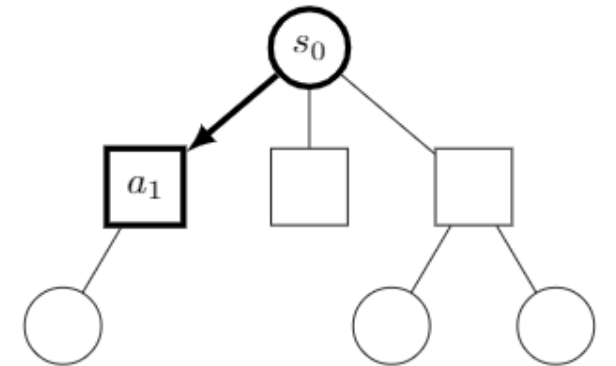
- For forward search, we started at the initial state and computed state transitions
 - new state = $T(s, a)$
- For backward search, we start at the goal and compute inverse state transitions
 - new set of subgoals = $T^{-1}(g, a)$
- To define $T^{-1}(g, a)$, must first define relevance: An action a is relevant for a goal g if
 - a makes at least one of g 's literals true, $g \cap \text{effects}(a) \neq \emptyset$
 - a does not make any of g 's literals false, $g \cup \text{effects}^{-}(a) = \emptyset$ and $g \cap \text{effects}^{+}(a) = \emptyset$

Total Order and Partial Order Plans



Monte Carlo Tree Search

- 4 phases of building out and simulating paths along a search tree
- Various forms of this used in everything from Alpha Zero to modern LLM inference
- For arbitrary problem with start state s_0 and actions a_i
- All states have attributes:
 - Total simulation reward $Q(s)$ and
 - Total no. of visits $N(s)$



Improvements to MCTS Components

- Improvements are possible for each of the parts I talked about
- Think about that it would take to improve selection / expansion phases

Upper Confidence Trees (UCT)

- A way of improving the selection phase by treating selection as a multi-arm bandit problem: which possible action to select that maximizes the possible payout (reward) in the future

$$\text{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c \sqrt{\frac{\ln N(v)}{N(v_i)}}$$

Upper Confidence Trees (UCT)

- A way of improving the selection phase by treating selection as a multi-arm bandit problem: which possible action to select that maximizes the possible payout (reward) in the future

$$\text{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c \sqrt{\frac{\ln N(v)}{N(v_i)}}$$

Exploit

Upper Confidence Trees (UCT)

- A way of improving the selection phase by treating selection as a multi-arm bandit problem: which possible action to select that maximizes the possible payout (reward) in the future

$$\text{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c \sqrt{\frac{\ln N(v)}{N(v_i)}}$$

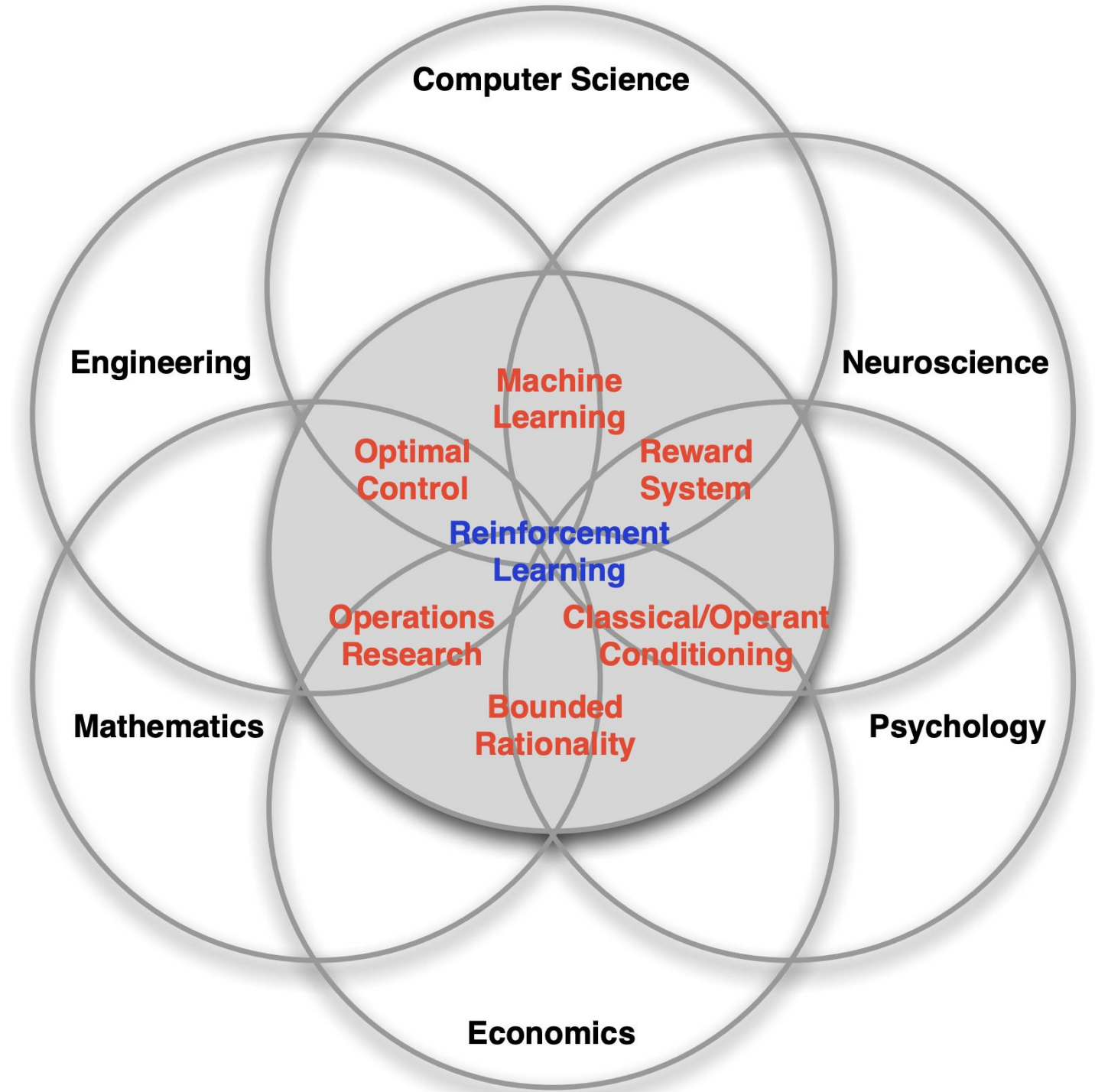
Exploit

Explore

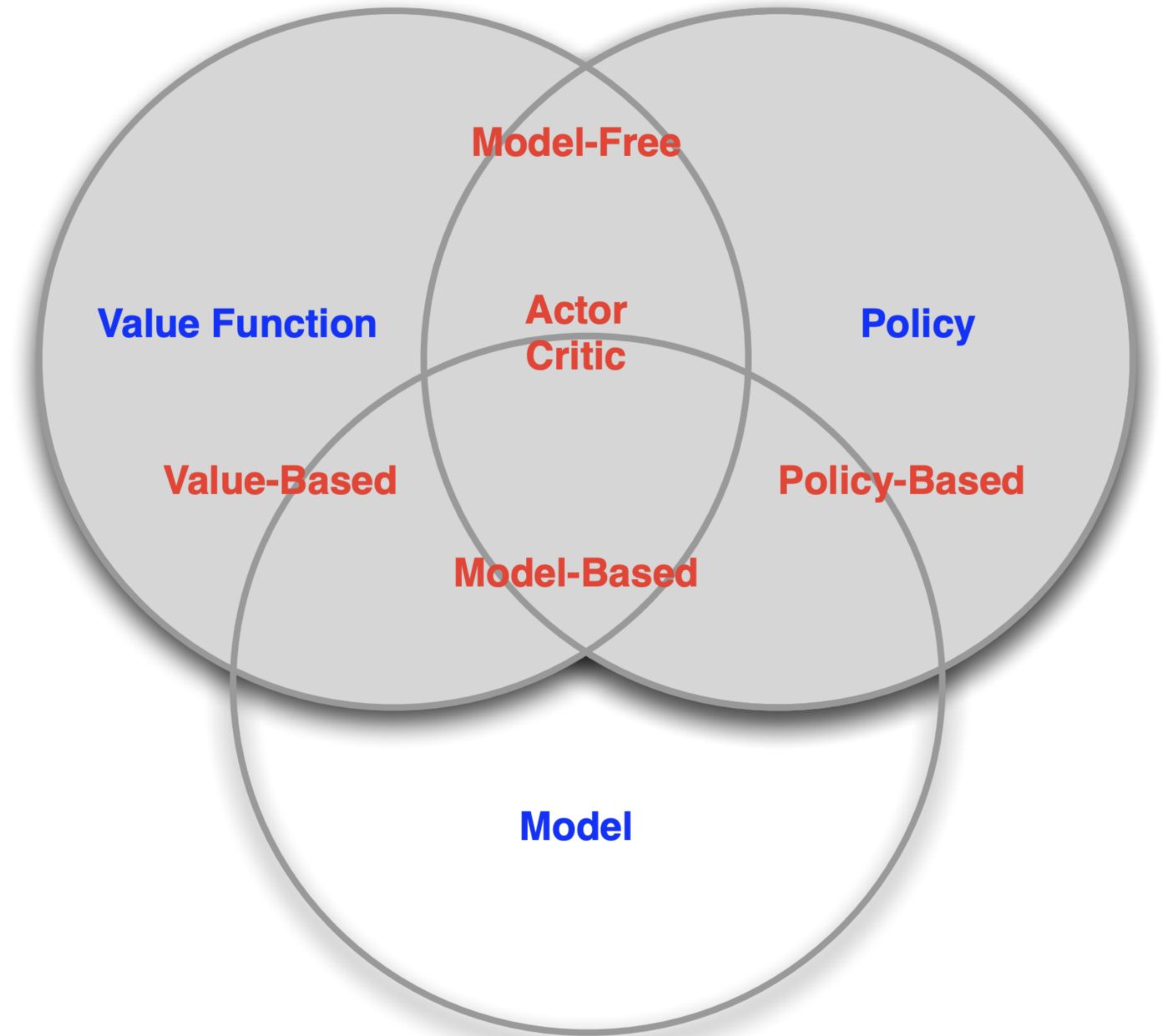
Why Reinforcement Learning?

- Reinforcement Learning:
 - The environment is initially unknown
 - The agent interacts with the environment
 - The agent improves its policy
- Planning:
 - A model of the environment is known
 - The agent performs computations with its model (without any external interaction)
 - The agent improves its policy a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Origins of RL



RL Agent Taxonomy



Terminology

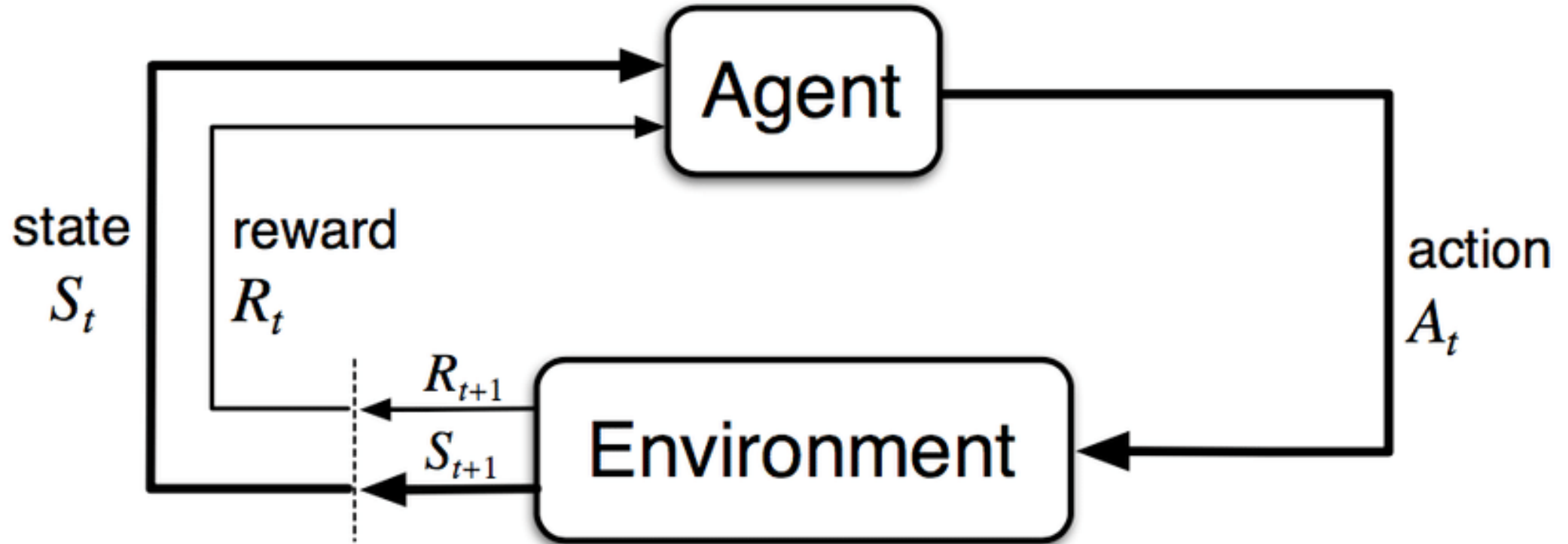
- Policy: agent's behavior function
 - Finding optimal policy known as the control problem
- Value function: how good is each state and/or action
 - Finding optimal value function is known as the prediction problem
- Model: agent's representation of the environment

More Terminology on Types of RL

- Model free ← will build up to today
- Model based

- On Policy ← will build up to today
 - Learn directly from your experiences “on the job”
- Off policy
 - Learn from someone else’s behavior

Markov Decision Process



Formal MDP Definition

A Markov Decision Process is a tuple $\langle S, A, T, R, \gamma \rangle$

- S is a finite set of states
- A is a finite set of actions
- T is a state transition probability matrix,
 $T_{ss'}^a = P[S_{t+1} = s' \mid S_t = s, A_t = a]$
- R is a reward function, $R_s^a = E[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.

Returns and Discounting

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

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

- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$
- $\gamma \sim 0$ is “myopic”, $\gamma \sim 1$ is “far-sighted”
- Why discount?
 - Mathematically convenient, avoids infinite returns
 - Animal/human/investment banker’s behavior shows preference for immediate reward

Formal Definition of Policy

- Distribution of action over states: $\pi(a|s) = P [A_t = a | S_t = s]$
- Policy depends only on current state not history, this is the Markov property bit of MDP (how do people get around this for cases where history does matter)
- Theorem (abridged): There always exists an optimal policy for a given finite MDP. It follow the optimal value function.

Formal Definition of Value Function

- State value: expected return starting from state s , and then following policy π
 - $v_{\pi}(s) = E_{\pi} [G_t | S_t = s]$
- Action value: is the expected return starting from state s , taking action a , and then following policy π
 - $q_{\pi}(s, a) = E_{\pi} [G_t | S_t = s, A_t = a]$

Dynamic Programming

- Building up to RL first requires understanding Dynamic Programming
- Dynamic sequential or temporal component to the problem Programming optimizing a “program”, i.e. a policy
- A method for solving complex problems by breaking them down into subproblems
 - Solve the subproblems → Combine solutions to subproblems

When to use DP

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

- Optimal substructure:
 - Principle of optimality applies
 - Optimal solution can be decomposed into subproblems
- Overlapping subproblems:
 - Subproblems recur many times
 - Solutions can be cached and reused
- Markov decision processes satisfy both properties Bellman equation gives recursive decomposition Value function stores and reuses solutions

Prediction vs Control

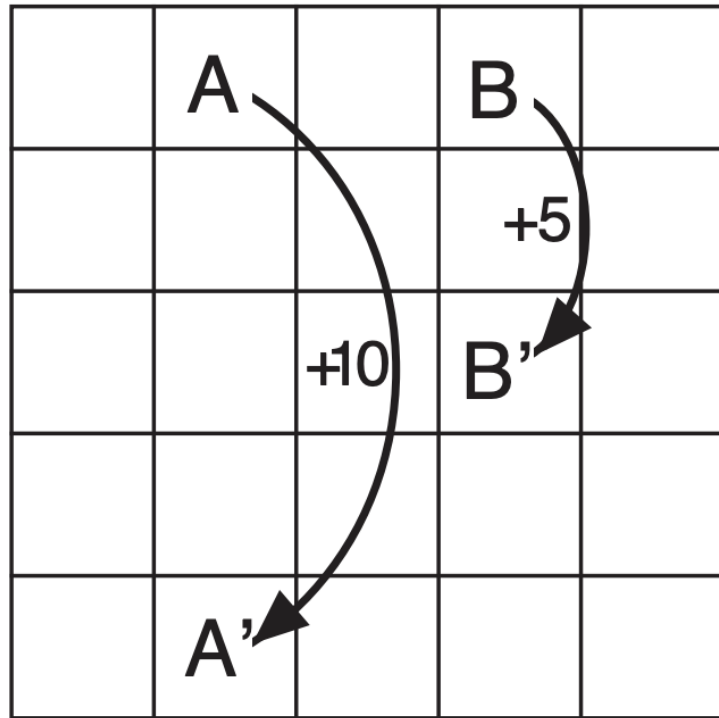
Two problems in RL

- Prediction is the problem of evaluating how good any given state is for getting rewards given a policy
- Control is the problem of selecting actions that give you a policy that maximizes reward

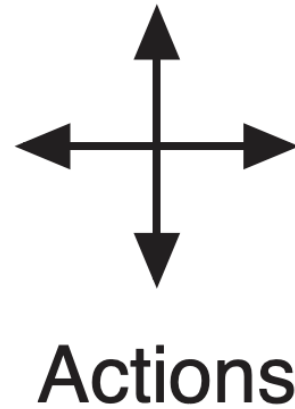
Planning via DP

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP
- For prediction:
 - Input: MDP $\langle S, A, T, R, \gamma \rangle$ and policy π
 - Output: value function v_π
- For control:
 - Input: MDP $\langle S, A, T, R, \gamma \rangle$
 - Output: optimal value function v^* and: optimal policy π^*

Prediction Example



(a)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

Bellman Expectation

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

$$v_{\pi}(s) = E_{\pi} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s]$$

- The action-value function can similarly be decomposed,

$$q_{\pi}(s, a) = E_{\pi} [R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

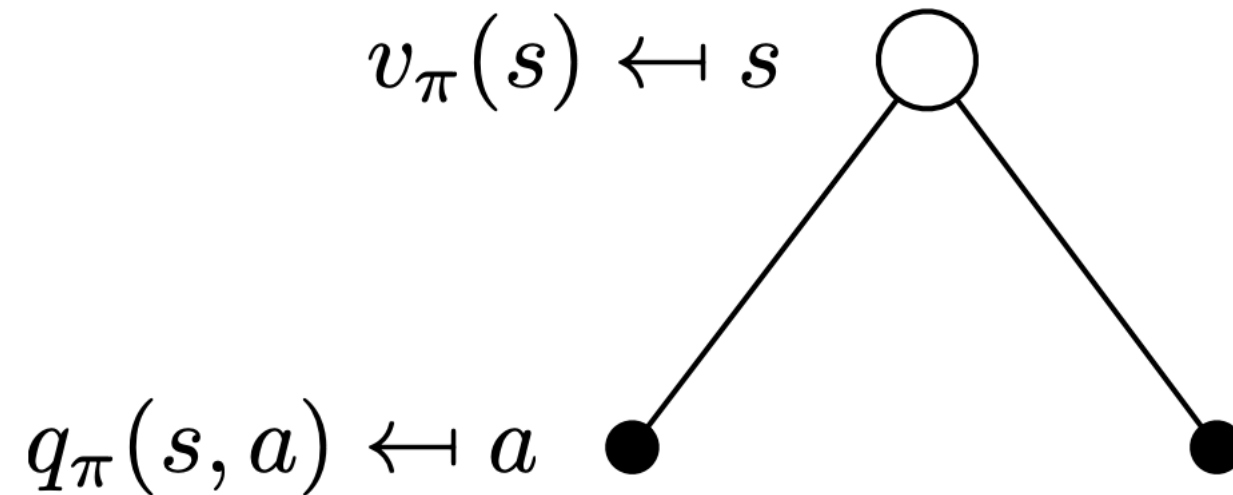
- No closed form solution (in general)

Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
$$V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_\pi$$
- Using synchronous backups,
 - At each iteration $k + 1$
 - For all states $s \in S$ Update $v_{k+1}(s)$ from $v_k(s')$, where s' is a successor state of s

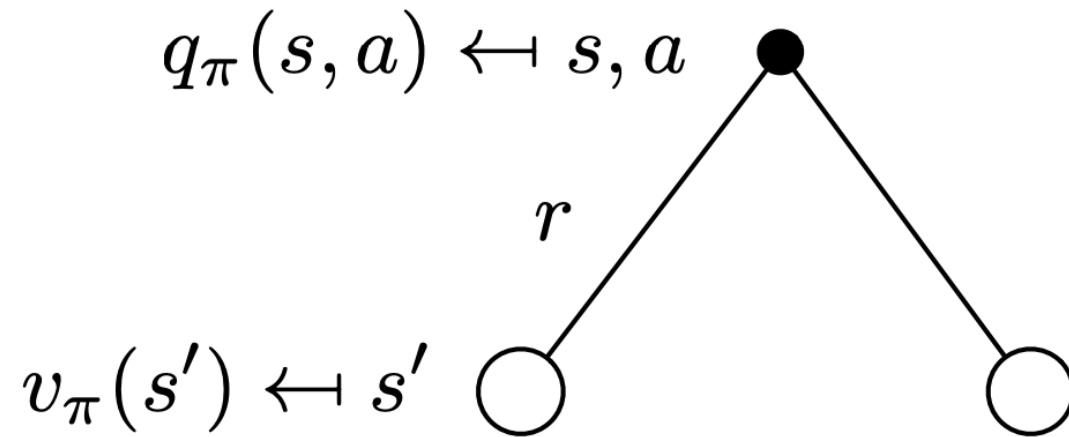
Policy Evaluation

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



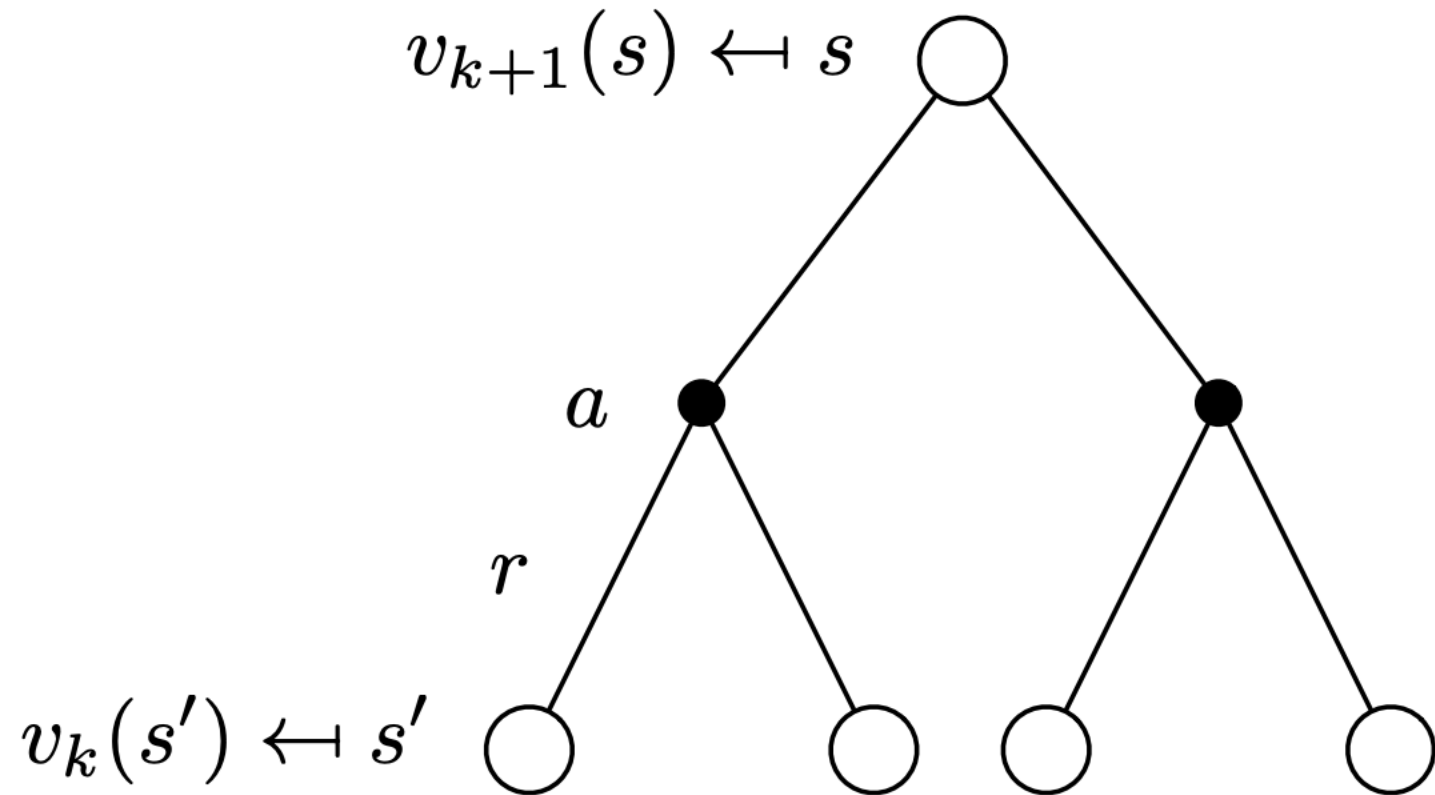
Policy Evaluation

$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a v_{\pi}(s')$$

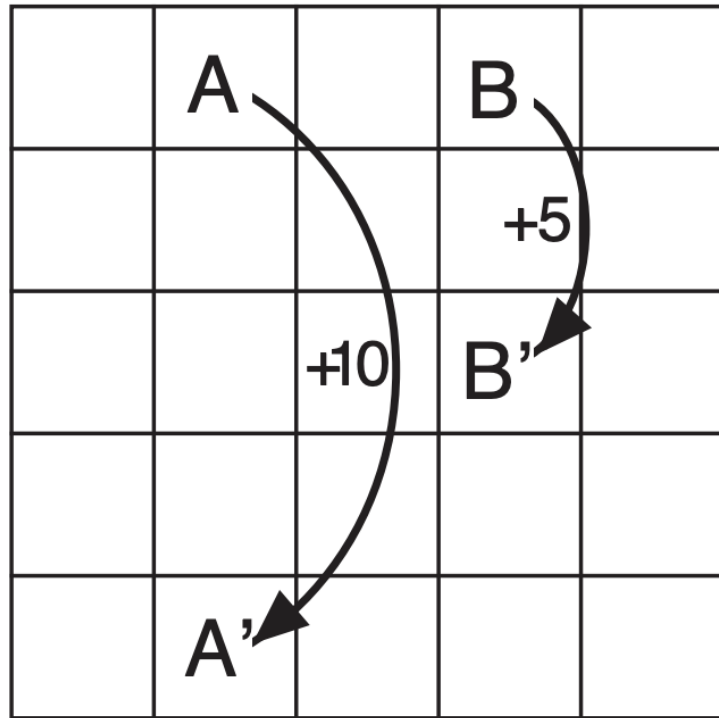


Policy Evaluation

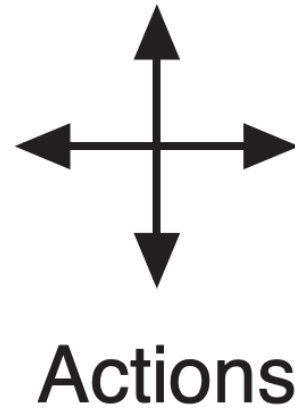
$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} T_{ss'}^a v_k(s'))$$



Prediction Example



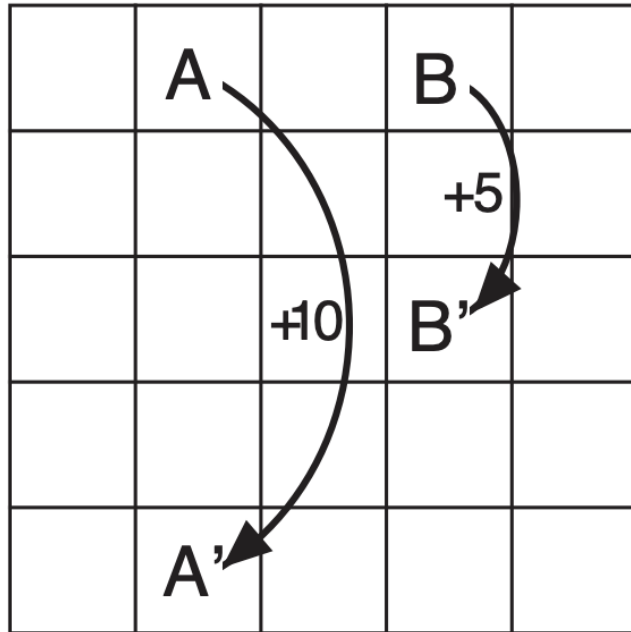
(a)



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

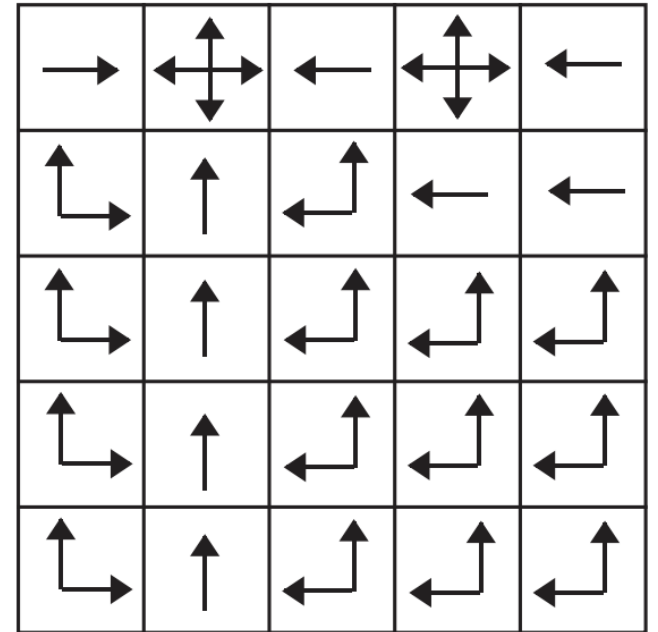
Control Example



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



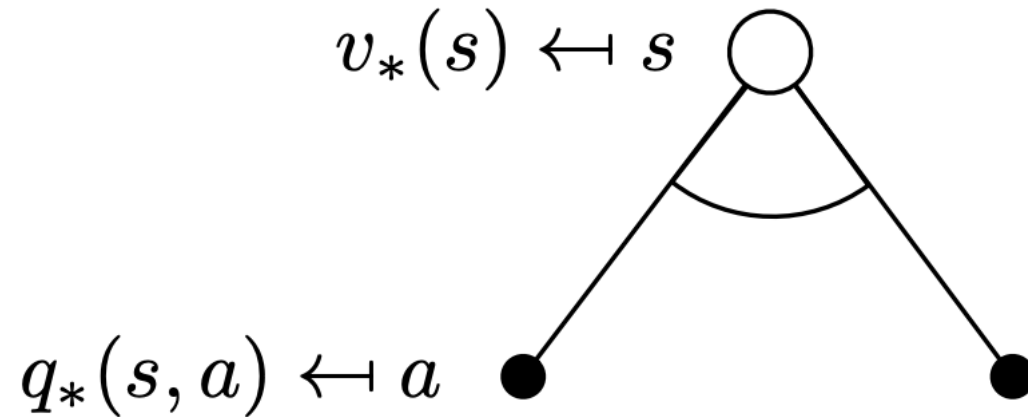
c) π_*

Bellman Optimality Equation

- Optimal state value: $v^*(s) = \max_{\pi} v_{\pi}(s)$
- Optimal action value: $q^*(s,a) = \max_{\pi} q_{\pi}(s,a)$
- Optimal policy: $\pi^*(s) = \operatorname{argmax}_a q^*(s,a)$

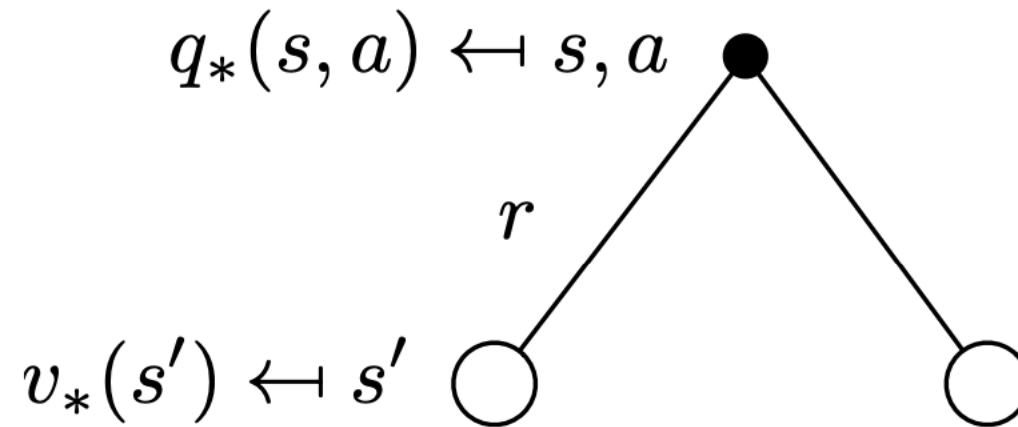
Bellman Optimality Equation

$$v^*(s) = \max_a q^*(s', a')$$



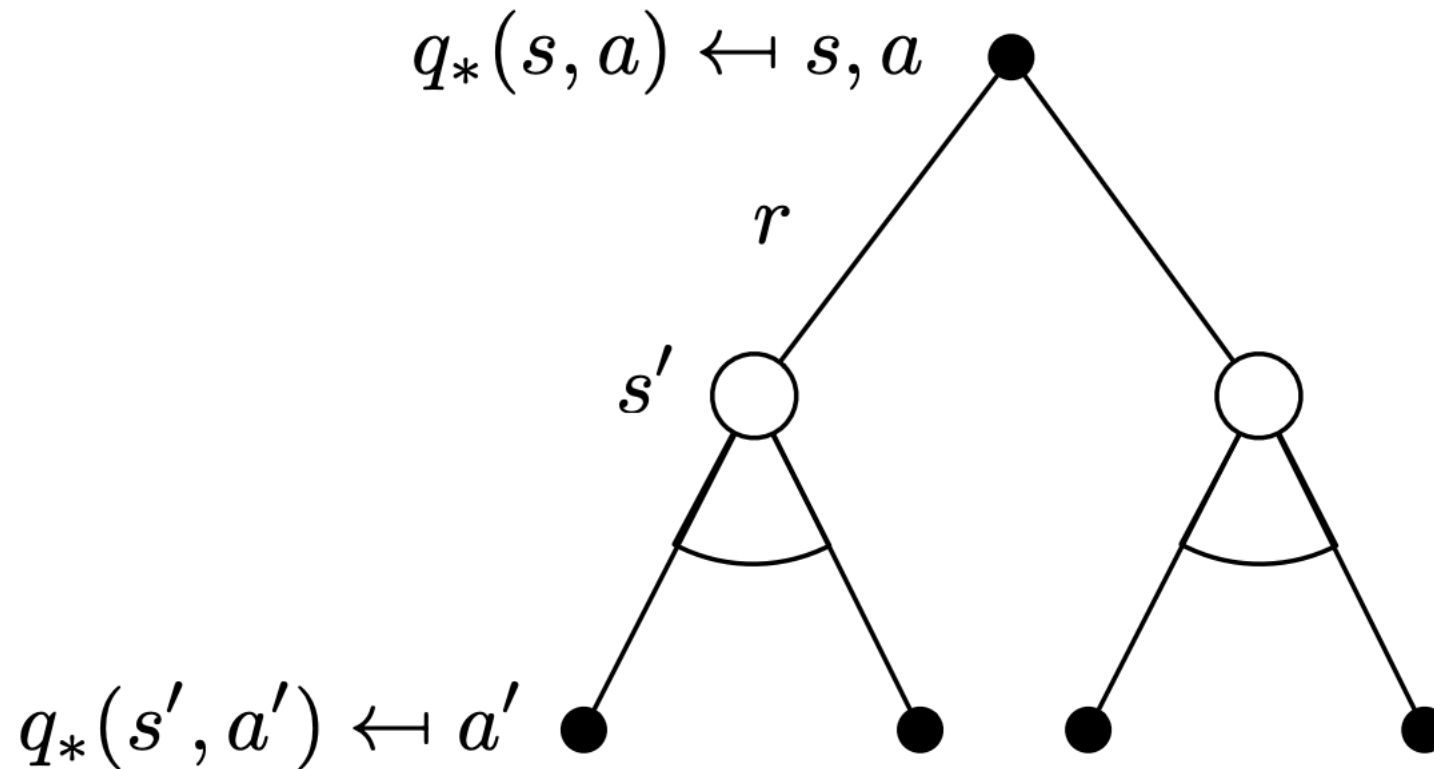
Bellman Optimality Equation

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a v^*(s')$$



Bellman Optimality Equation

$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} T_{ss'}^a \max_{a'} q^*(s',a')$$



Bellman Optimality Equation

- Optimal state value: $v^*(s) = \max_{\pi} v_{\pi}(s)$
- Optimal action value: $q^*(s,a) = \max_{\pi} q_{\pi}(s,a)$
- Optimal policy: $\pi^*(s) = \operatorname{argmax}_a q^*(s,a)$

- $q^*(s,a) = R^a_s + \gamma \sum_{s' \in S} T^a_{ss'} \max_a q^*(s',a')$
- $v^*(s) = \max_{a \in A} q^*(s, a)$

Policy Iteration

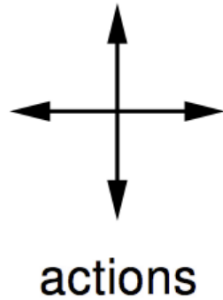
Given a policy π

- Evaluate the policy π $v_{\pi}(s) = E [R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$
- Improve the policy by acting greedily with respect to v_{π}
 $\pi' = \text{greedy}(v_{\pi})$
- Converting back and forth between prediction and control
- Start with random policy, eval it, improve value, improve policy

Value Iteration

- Similar to Policy Iteration but start with random value function, recursively improve it
- Exercise to figure out equations if you start with random value instead of policy

Put it together



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

$r = -1$
on all transitions

- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, ..., 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy $\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$

\mathcal{V}_k for the
Random Policy

Greedy Policy
w.r.t. \mathcal{V}_k

$k = 0$

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

	↕	↕	↕
↕	↕	↕	↕
↕	↕	↕	↕
↕	↕	↕	

← random
policy

$k = 1$

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

	←	↕	↕
↑	↕	↕	↕
↕	↕	↕	↓
↕	↕	→	

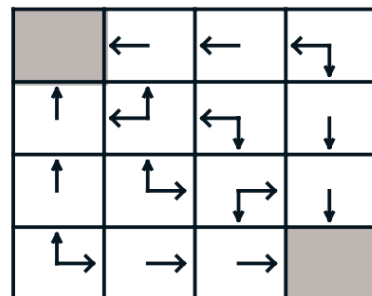
$k = 2$

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

	←	←	↕
↑	↖	↕	↓
↑	↕	↘	↓
↕	→	→	

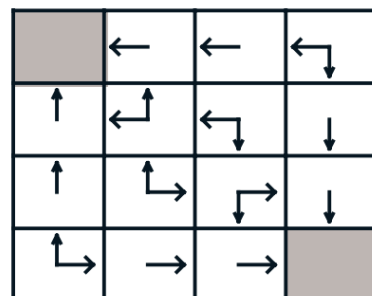
$k = 3$

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



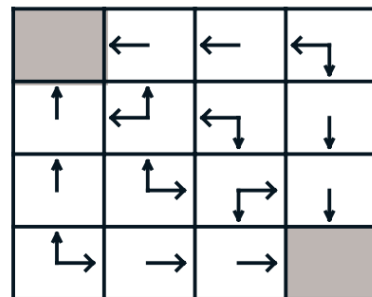
$k = 10$

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

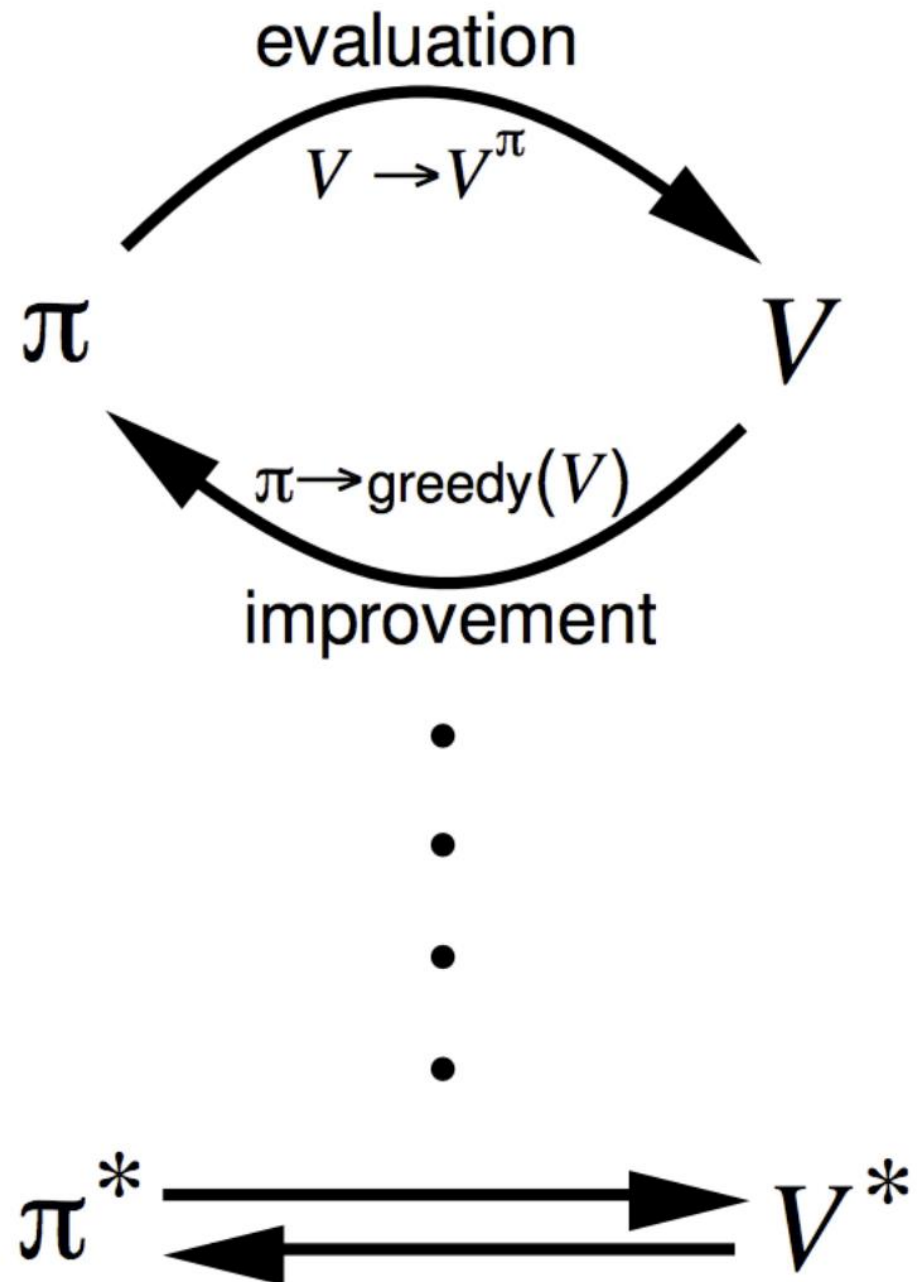
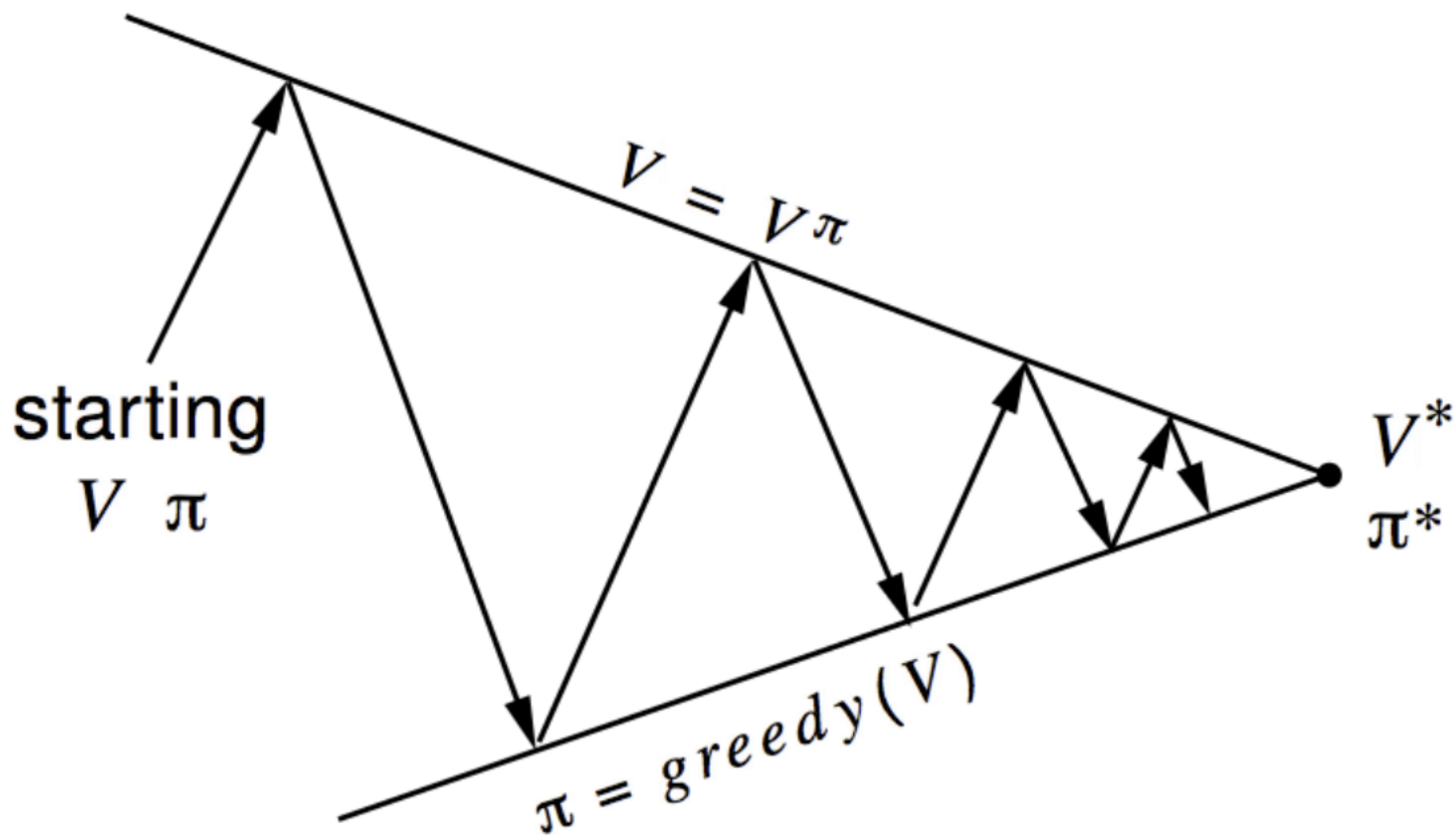


optimal
policy



Generalized Policy Iteration

- Both are iterative versions of this



DP Limitations

- DP uses full-width backups
- For each backup Every successor state and action is considered
- Using knowledge of the MDP transitions and reward function DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's curse of dimensionality
- Number of states $n = |S|$ grows exponentially with number of state variables Even one backup can be too expensive

Model Free RL via Sample Backups

- Model Free RL: optimize value of unknown MDP
- Using sample rewards and sample transitions $\langle S, A, R', S' \rangle$
Instead of reward function R and transition dynamics T
- Advantages: Model-free: no advance knowledge of MDP required
Breaks the curse of dimensionality through sampling
- Cost of backup is constant, independent of $n = |S|$

Experience Based Learning

- Many real world problems are better suited to being solved by RL as opposed to DP based planning
- All the examples of agents we talked about first class
 - Robots in your home
 - Video games harder than tic tac toe
 - Language

Monte Carlo Control

- Greedy policy improvement over $V(s)$ requires model of MDP

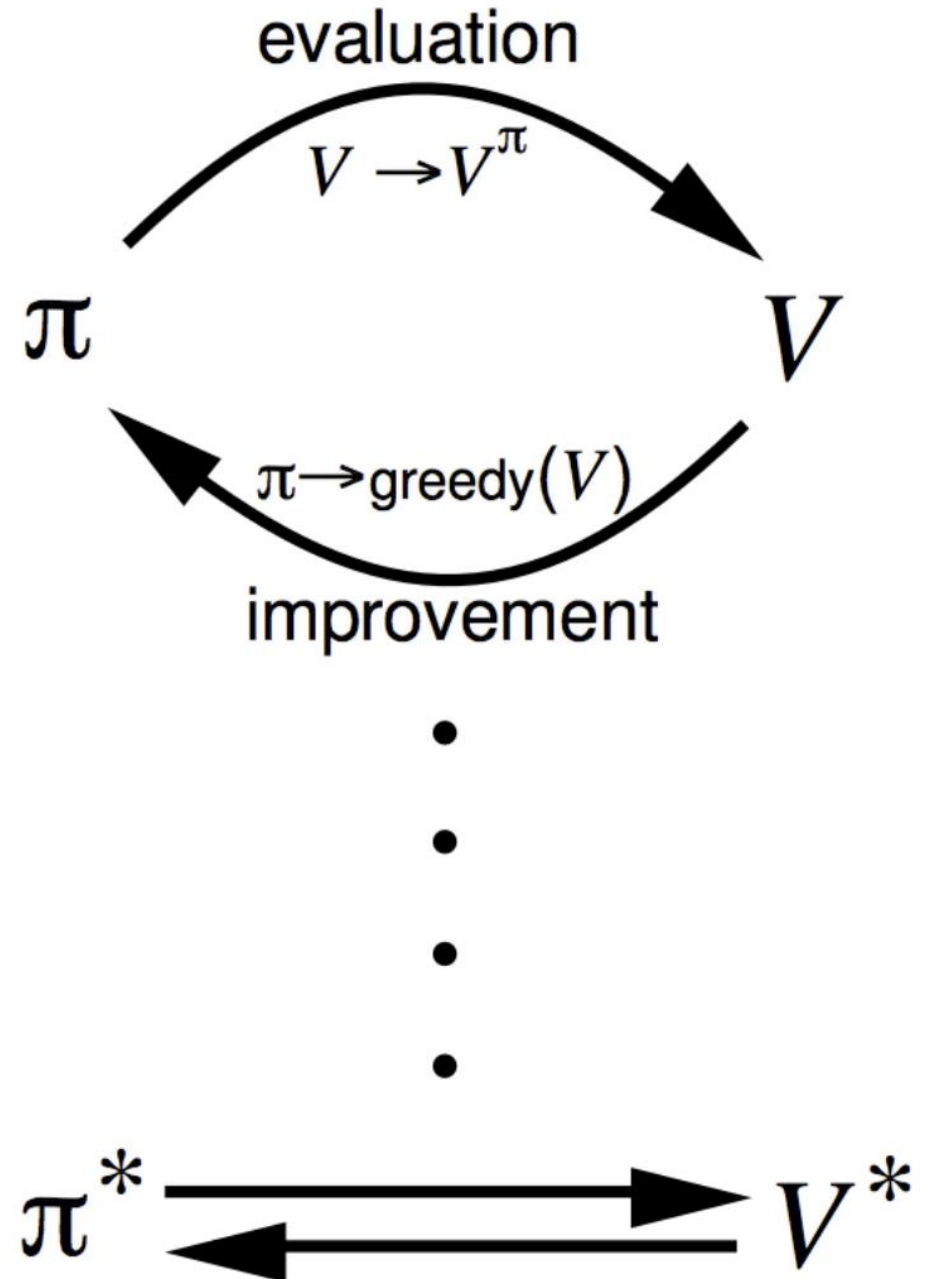
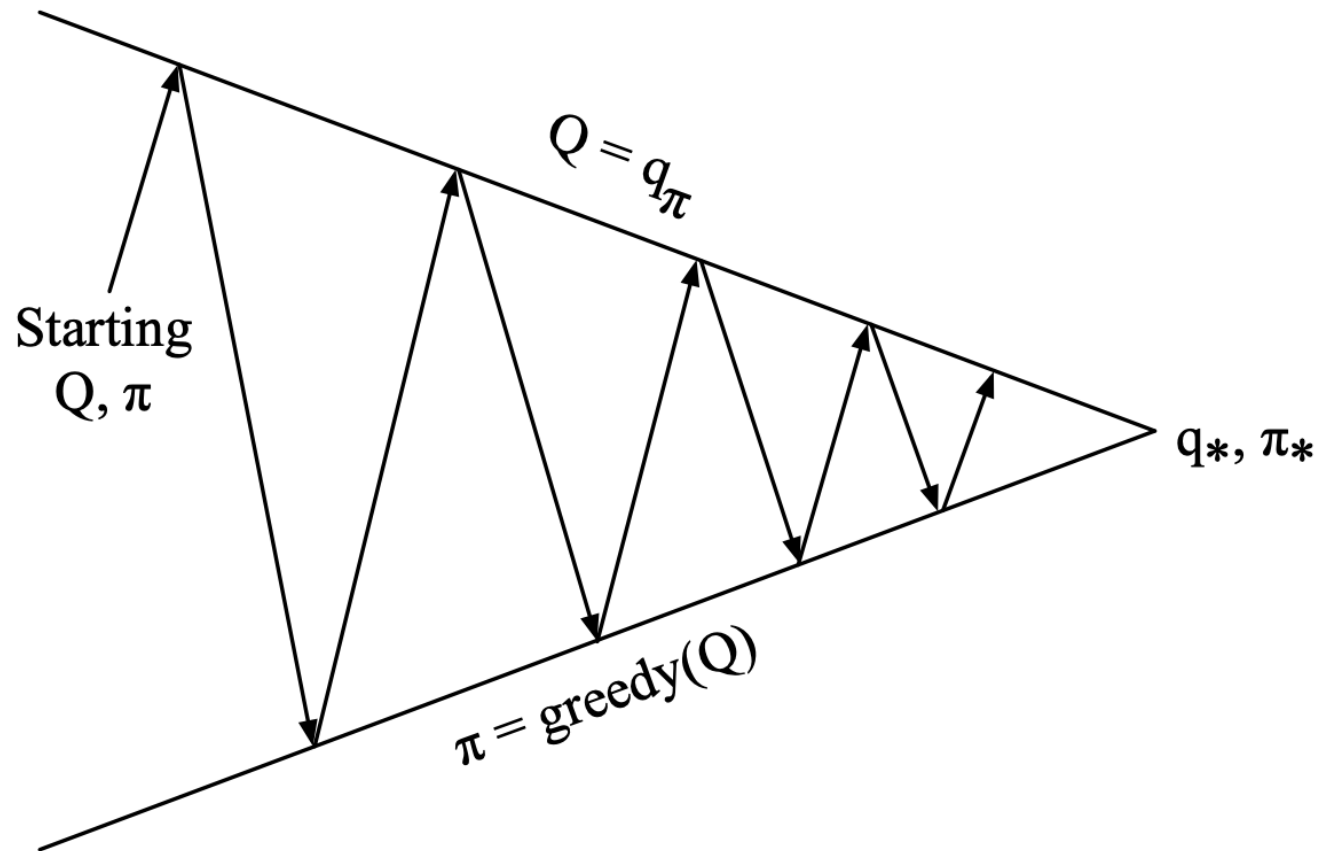
$$\pi'(s) = \operatorname{argmax}_{a \in A} R^a_s + \gamma \sum_{s' \in S} T^a_{ss'} V'(s')$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

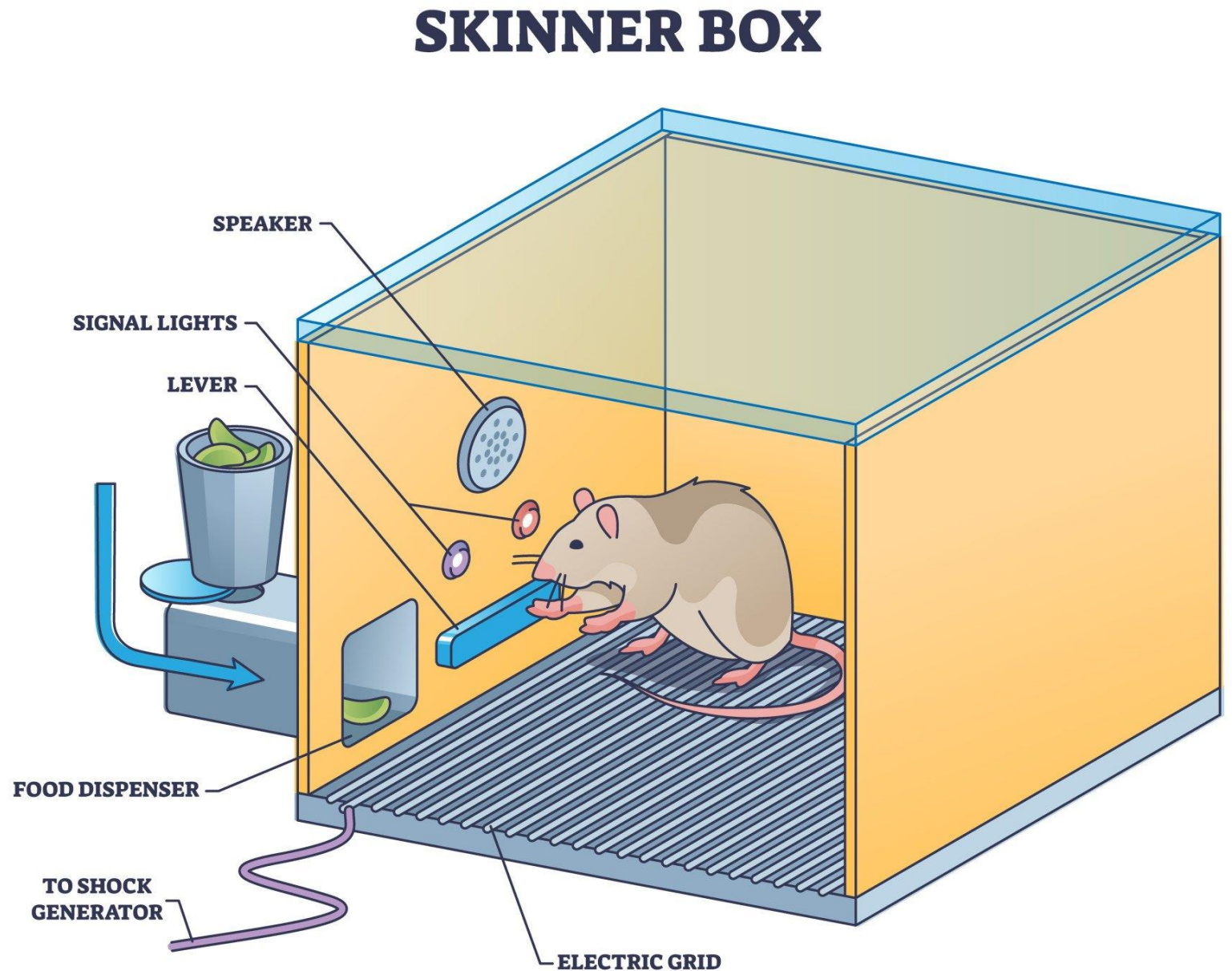
- Learn this Q by function approximation using the experiences you've gathered by Monte Carlo sampling

Generalized Policy Iteration Monte Carlo Evaluation



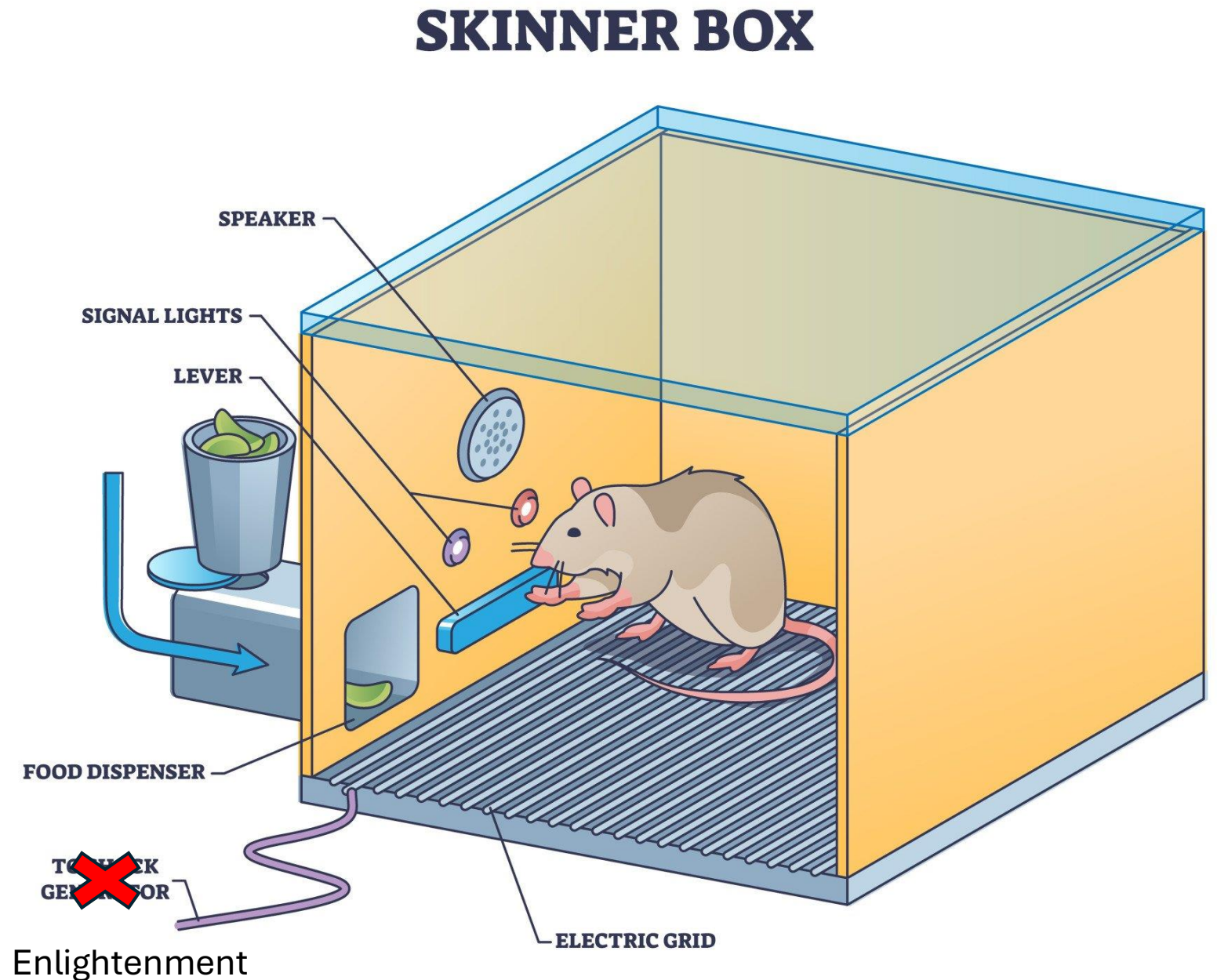
Greedy Policy Improvement Limitations

- Greedy doesn't let you always explore all the actions you need



Greedy Policy Improvement Limitations

- Greedy doesn't let you always explore all the actions you need



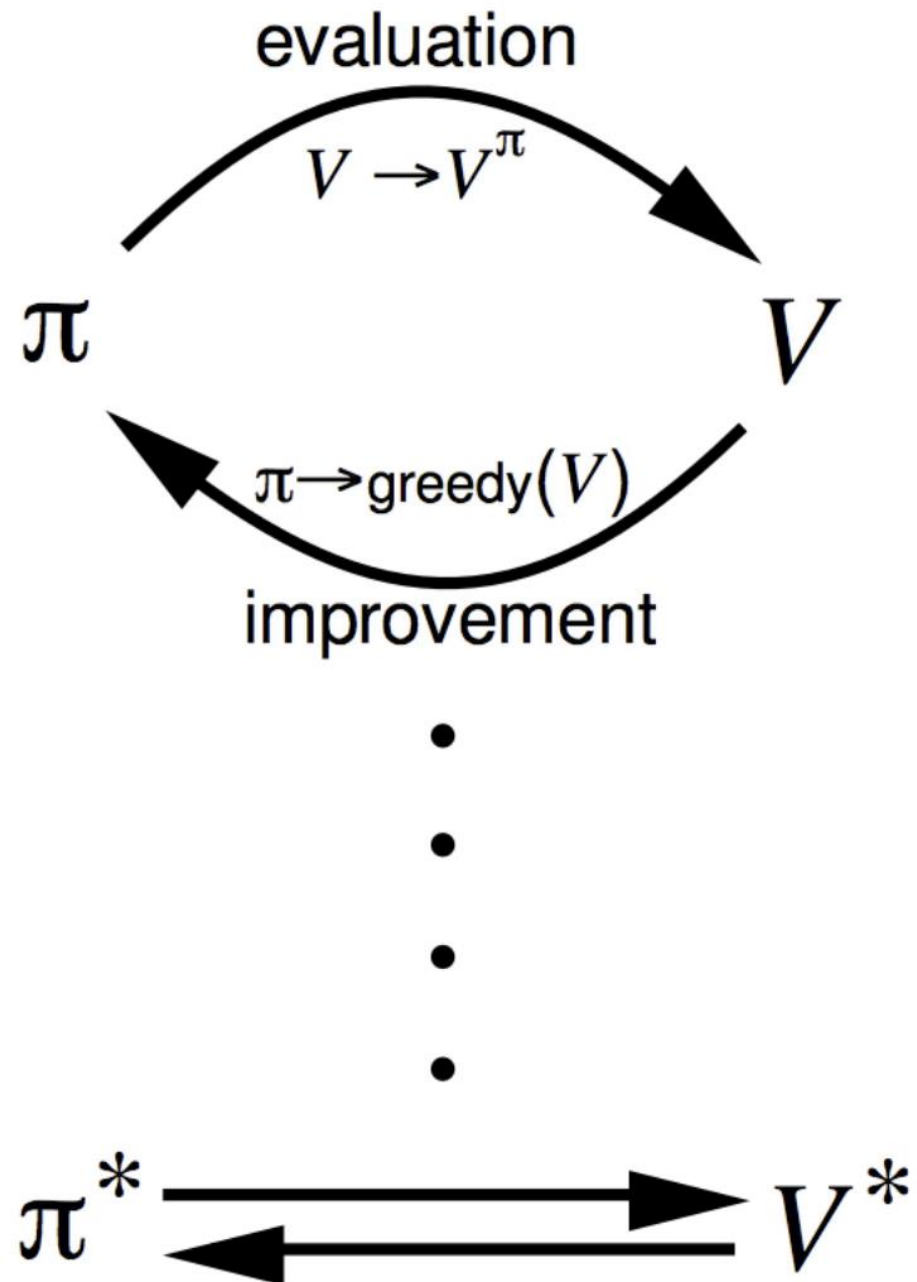
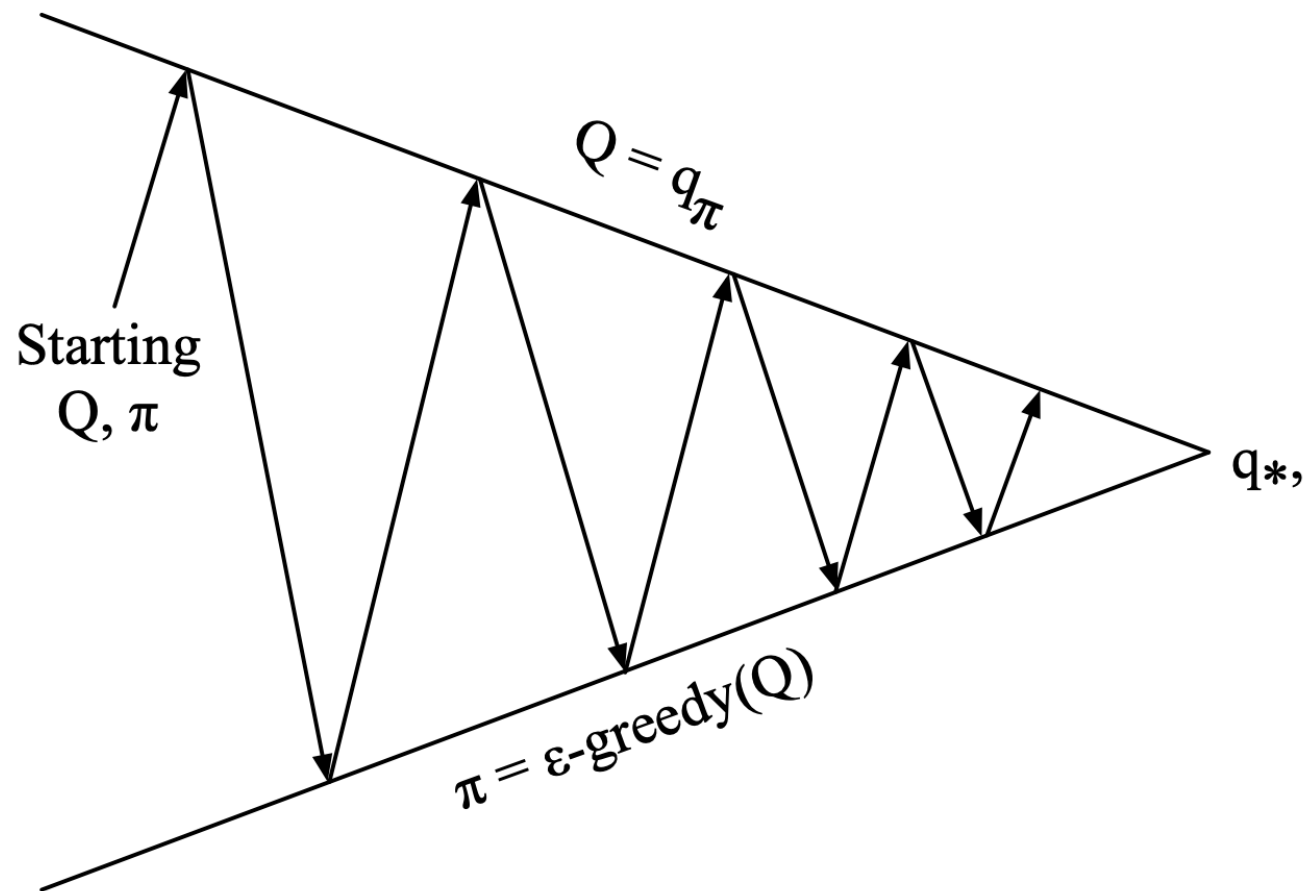
ϵ -greedy exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ – choose the greedy action
- With probability ϵ choose an action at random

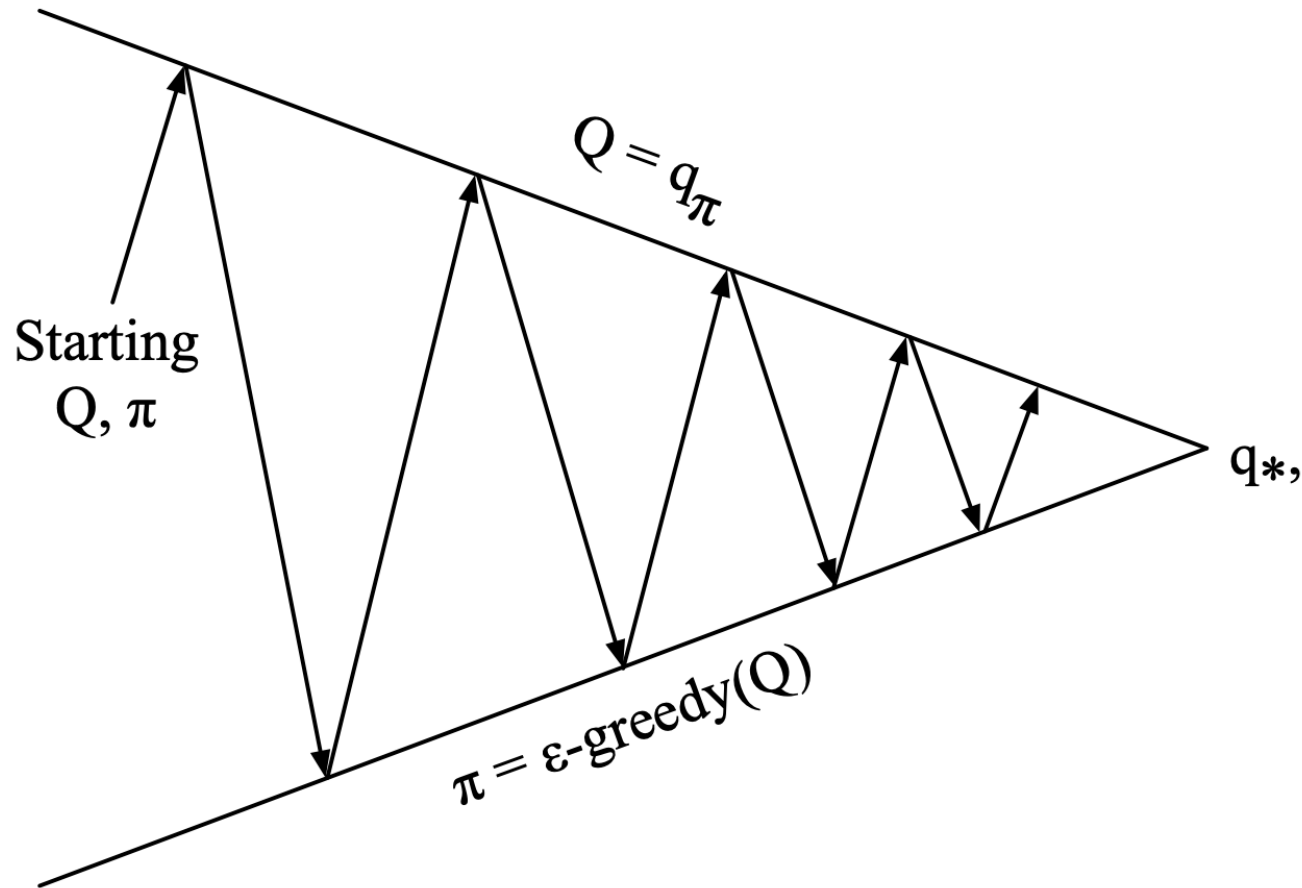
$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Generalized Policy Iteration

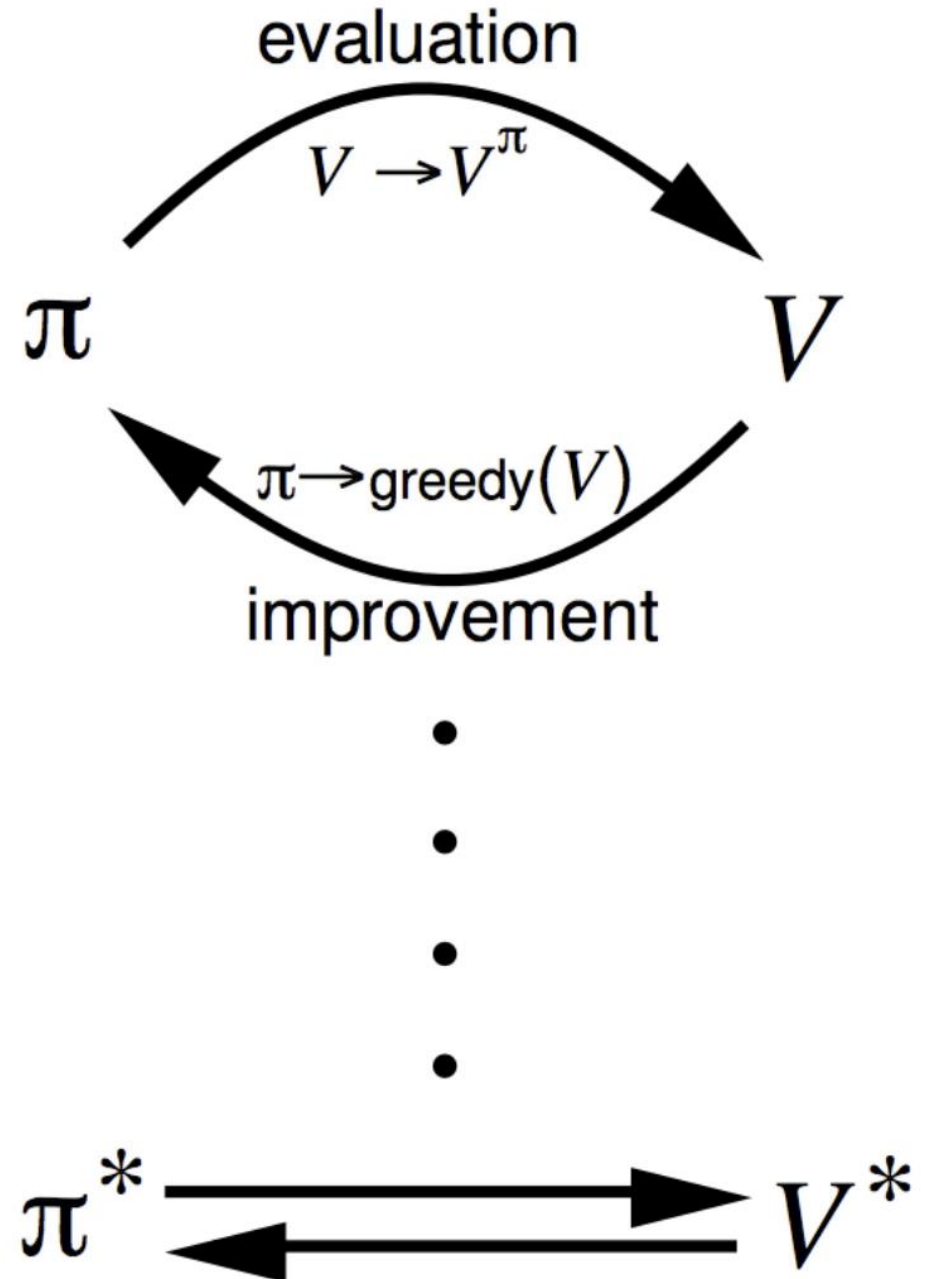
Monte Carlo Evaluation



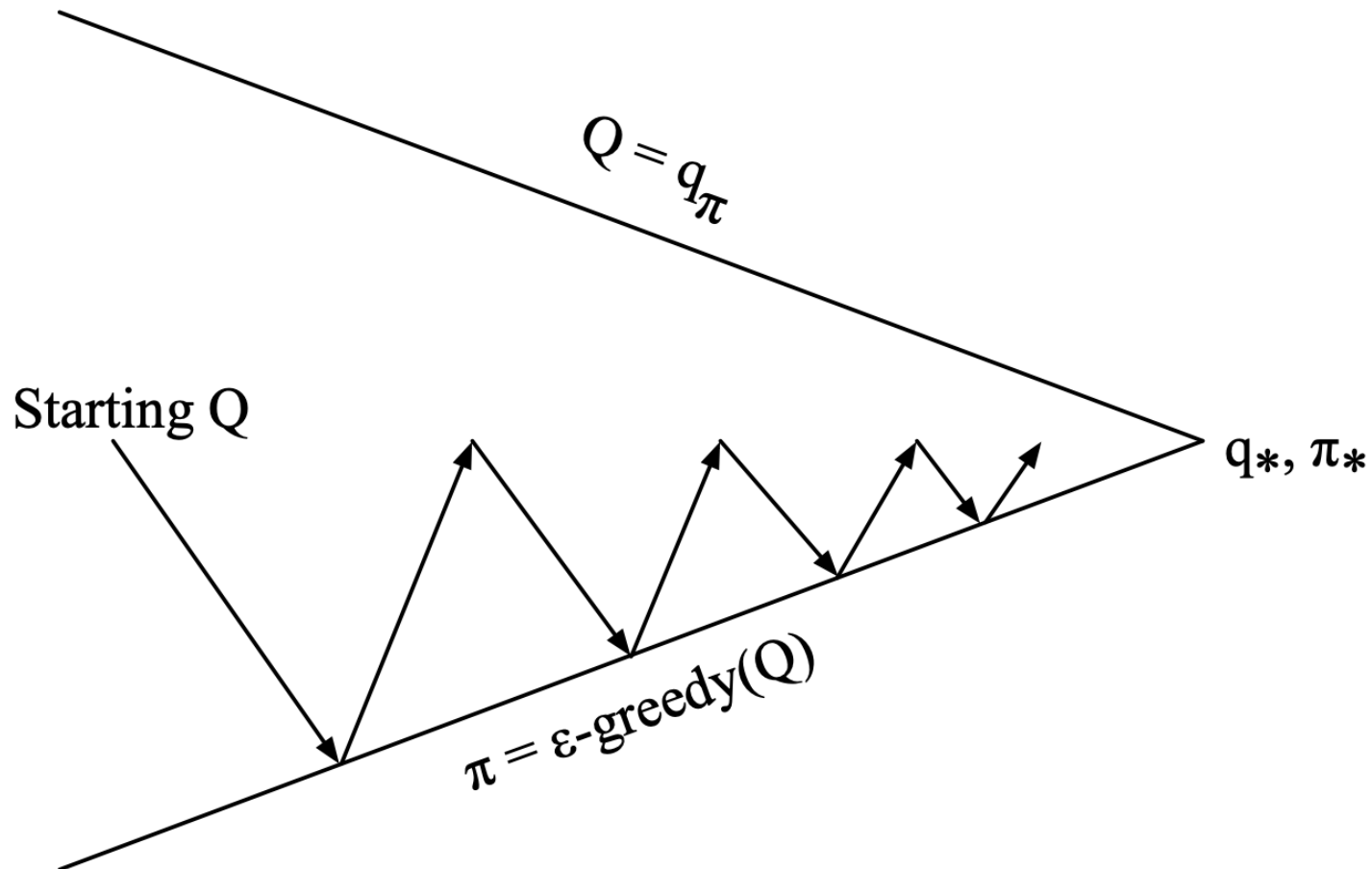
Generalized Policy Iteration Monte Carlo Evaluation



You can't fully evaluate the entire state space each time



Generalized Policy Iteration with Fn Approximation + Monte Carlo Eval



You can't fully evaluate the entire state space each time

