Lecture 30

CS 6260 Automated Planning and Learning

Solving MDPs: Policy Iteration and Evaluation

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Some material based on lecture slides for ASU's CSE571 Artificial Intelligence by Siddharth Srivastava.

Limitations of Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s) + \gamma V_k(s')]$$

- $O(S^2 A)$ time per iteration
- Policy may have converged even when values haven't
- Policy iteration is another approach for computing V^* and π^* that addresses these issues

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: go greedy over one-step lookahead converged (but not optimal!) utilities as future values`
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration

- First fix a policy, find its value function using VI until convergence
  - $V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s) + \gamma V_k(s')]$
  - Does this make sense?

Policy Iteration

- Repeat until convergence:
  - First fix a policy, compute its value function using VI until convergence

$$V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s) + \gamma V_k(s')]$$

  - Then, improve the policy using a greedy update:

$$\pi_{i+1}(s) = argmax_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

  - Go back to computing value function of π_{i+1}.

Policy Iteration

- Repeat until convergence:
  - First fix a policy, compute its value function using VI until convergence

$$V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s) + \gamma V_k(s')]$$

Policy Evaluation

  - Then, improve the policy using a greedy update:

$$\pi_{i+1}(s) = argmax_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Policy Improvement

  - Go back to computing value function of π_{i+1}.

Policy Iteration

- Repeat until convergence:
  - First fix a policy, find its value function using VI until convergence

$$V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s) + \gamma V_k(s')]$$

  - Then, improve the policy using a greedy update:

$$\pi_{i+1}(s) = argmax_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

  - Go back to computing value function of π_{i+1}.

Policy Iteration

- Repeat until convergence:
  - First fix a policy, find its value function using VI until convergence

$$V_{k+1}(s) = \sum_{s'} T(s, \pi(s), s')[R(s) + \gamma V_k(s')]$$

Policy Evaluation

  - Then, improve the policy using a greedy update:

$$\pi_{i+1}(s) = argmax_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

Policy Improvement

  - Go back to computing value function of π_{i+1}.

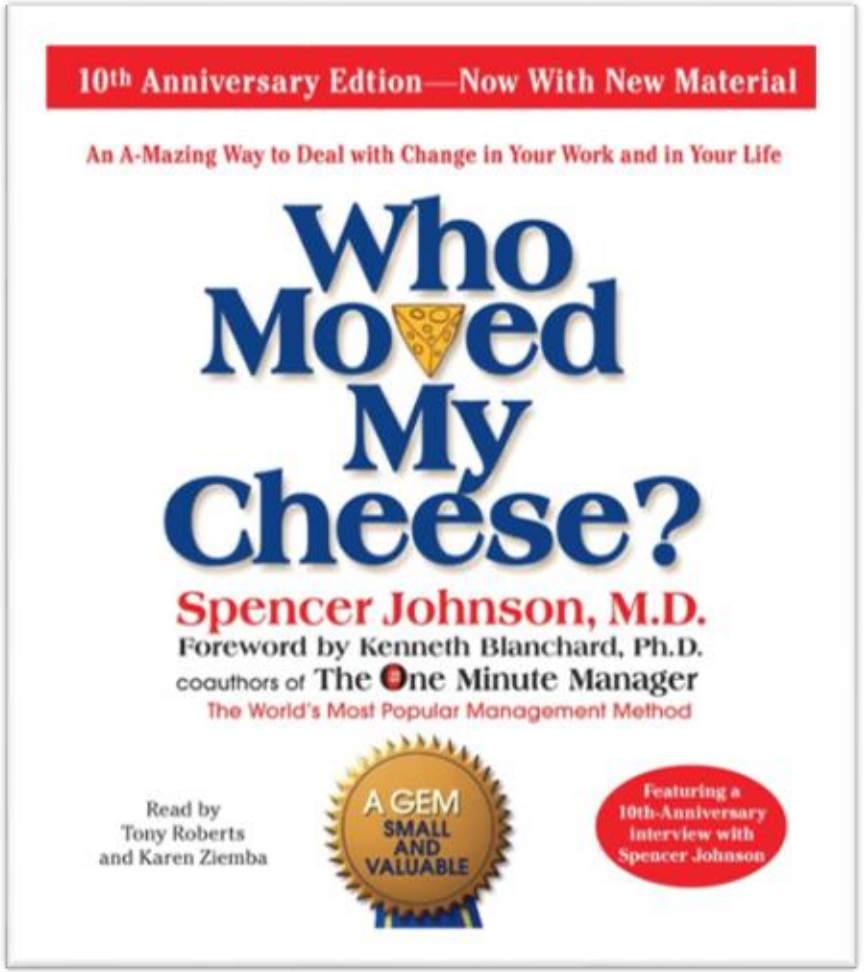
For a Change…

- In the real world, things change

…and we must deal with them

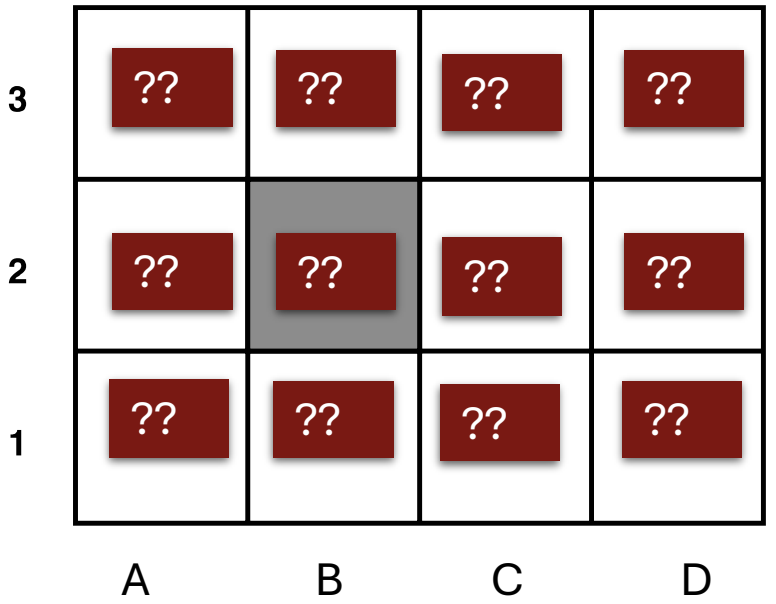
- Adapting to new situations is an essential component of AI

- How would our agent deal with it?

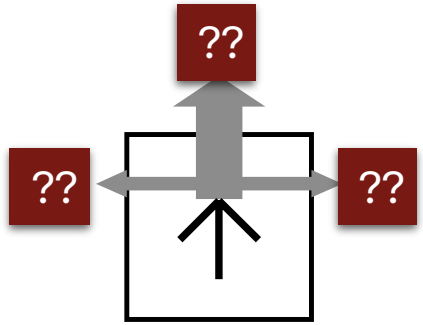


Who Moved My Cheese?
And greased my wheels…
And moved the walls…

- Suppose the agent finds itself in a new environment
- What would you do?



(a)

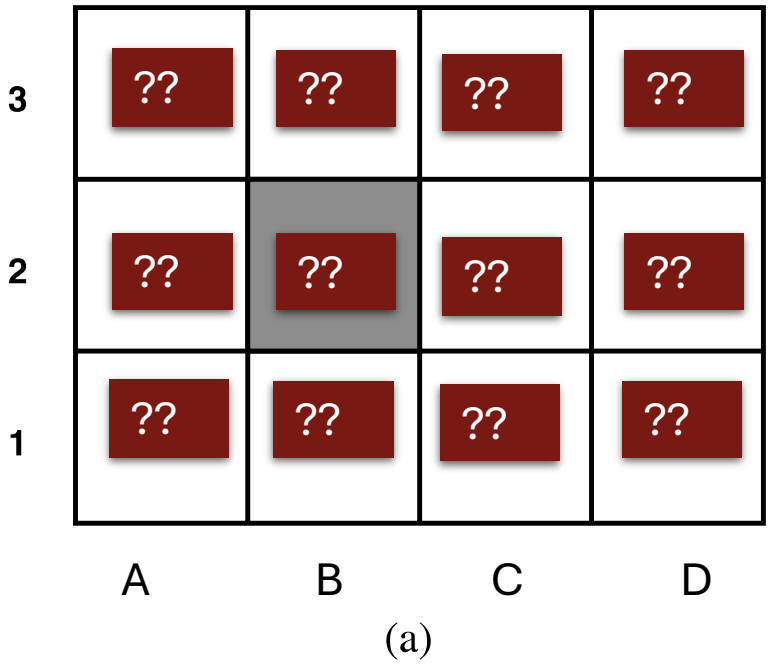


(b)

Map not known to the agent
(or partially known)

Who Moved My Cheese?
And greased my wheels…
And moved the walls…

- Suppose the agent finds itself in a new environment
- Several ideas:
  - Transfer+adapt known knowledge, policy
    - Let me try my last policy first
    - Let me recompute my policy with this change in the environment that I noticed
  - Transfer learned principles
    - In such places, walls are impassable
  - Learn new environment model, then plan
  - Learn while acting in the environment



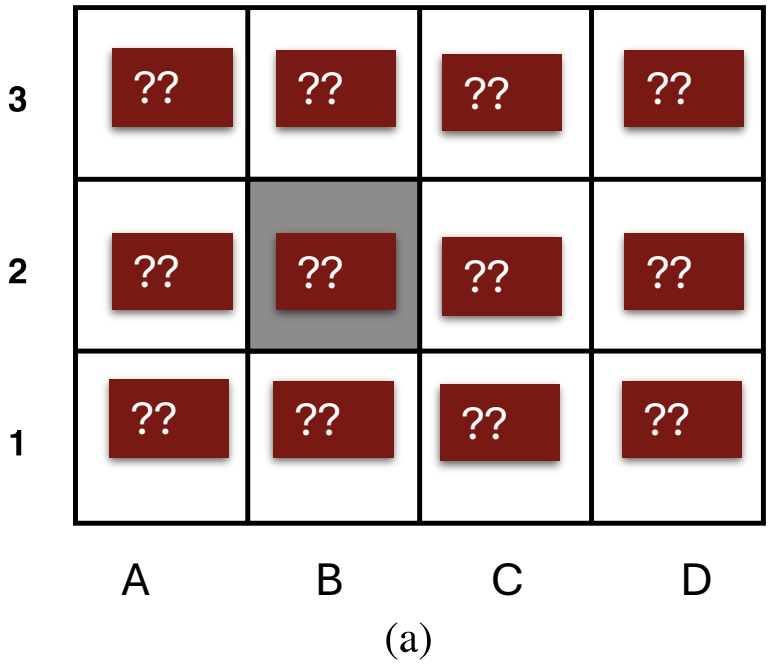
Map not known to the agent
(or partially known)

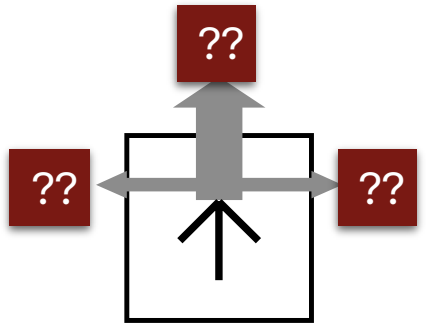
Integrating Planning and Learning

- Reinforcement learning: learning to act based on online feedback from the environment

- Fundamental trade-off:
  - Explore
    - Experiment, learn about the environment
  - Exploit
    - Settle on the learned knowledge, use current best policy to cut (possible) losses

- In pure RL, agent acts in the *real* environment
  - **Not** in a simulator!
  - Model free!

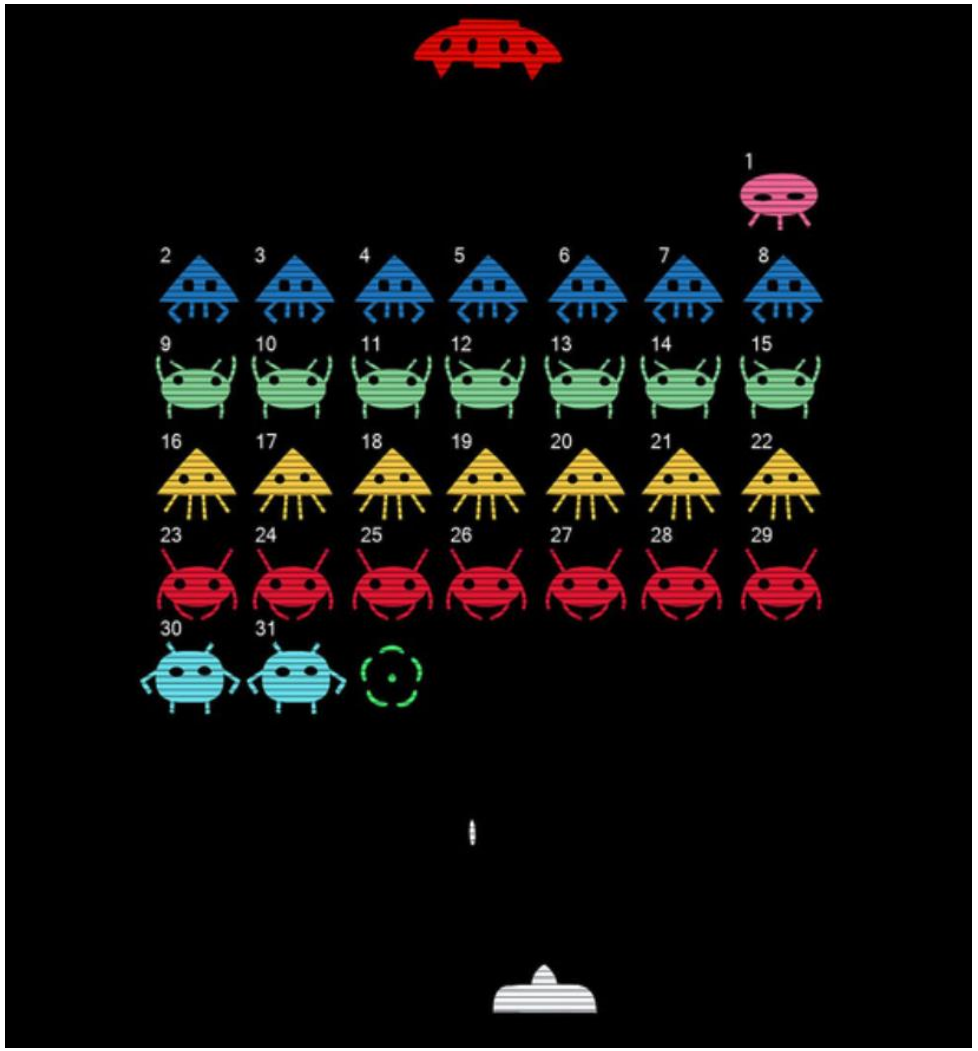




Map not known to the agent
(or partially known)

Approach 1: Learn Model, Then Plan

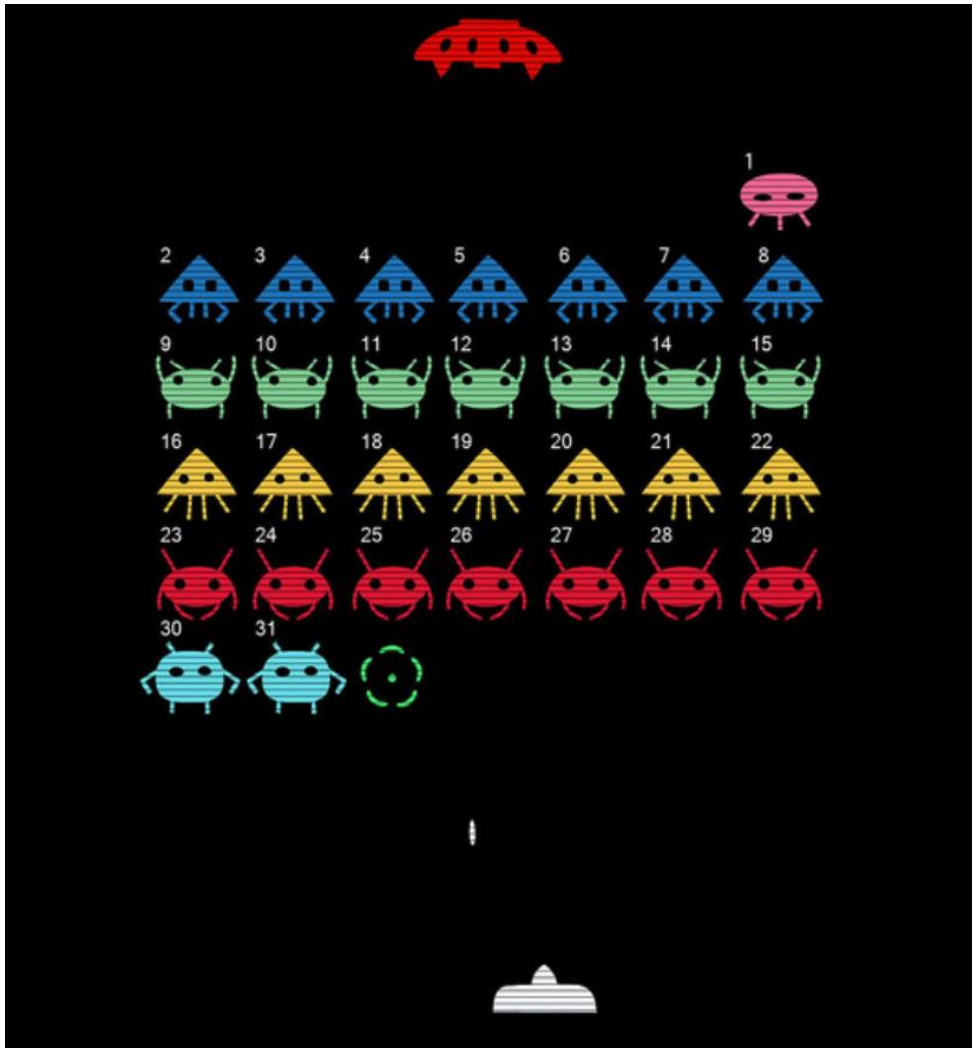
- Advantages of model learning:
  - Easy to transfer to new situations if some information about changes is available
  - Examples:
    - Robot learned to deliver coffee when doors were open
    - New environment: doors closed
    - Can focus learning to opening doors
  - Video games
    - New reward function: need to keep last row alive for 10s!



Approach 1: Learn Model, Then Plan

- Limitations of model learning:
  - Agent needs time to learn
  - While it learns, it doesn't try to reach the goal

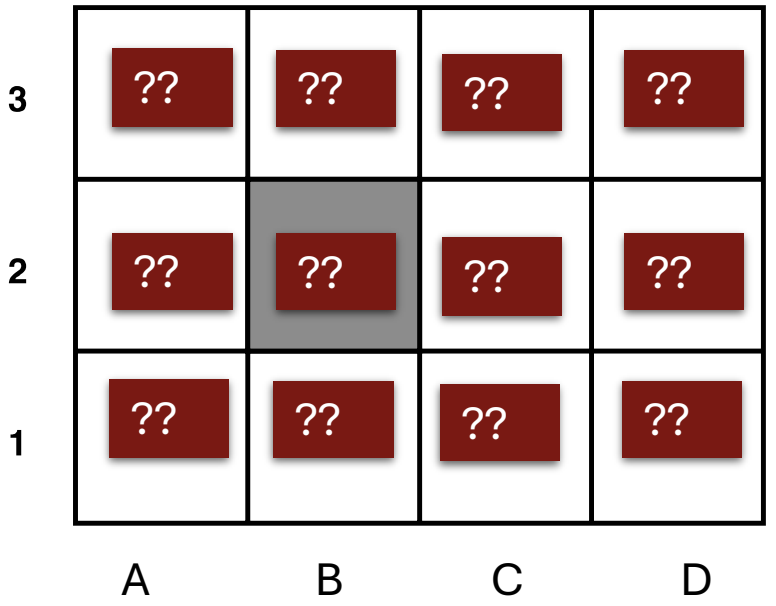
- A bit like learning a new sport…
  no tournaments before you reach a level of competence



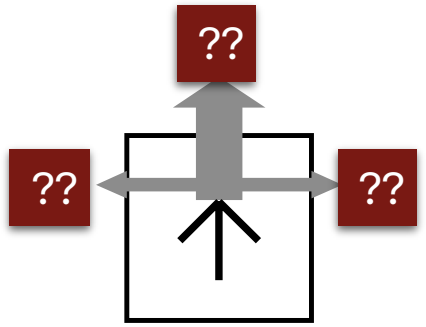
Example of Model Learning

- Training data compiled using an input policy (selecting $A1$):
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, A2, -10$
  - $A1, \rightarrow, B2, 1$
  - $A1, \rightarrow, A2, -10$
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, A2, -10$

Rewards received prior to moving



(a)



(b)

- Learned model:

  $P(B1|A1, \rightarrow) = \frac{3}{7}$, $P(B2|A1, \rightarrow) = \frac{1}{7}$,
  $P(A2|A1, \rightarrow) = \frac{3}{7}$

  Reward: $B1: 0; B2: 1; A2: -10$

Map not known to the agent (or partially known)

Example of Model Learning

- Training data compiled using an input policy:
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, A2, -10$
  - $A1, \rightarrow, B2, 1$
  - $A1, \rightarrow, A2, -10$
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, A2, -10$
- Learned model:

  $P(B1|A1, \rightarrow) = \frac{3}{7}$, $P(B2|A1, \rightarrow) = \frac{1}{7}$,

  $P(A2|A1, \rightarrow) = \frac{3}{7}$

  Reward: $B1: 0; B2: 1; A2: -10$

Assumptions?

Example of Model Learning

- Training data compiled using an input policy:
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, A2, -10$
  - $A1, \rightarrow, B2, 1$
  - $A1, \rightarrow, A2, -10$
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, B1, 0$
  - $A1, \rightarrow, A2, -10$
- Learned model:

  $P(B1|A1, \rightarrow) = \frac{3}{7}$, $P(B2|A1, \rightarrow) = \frac{1}{7}$,

  $P(A2|A1, \rightarrow) = \frac{3}{7}$

  Reward: $B1: 0; B2: 1; A2: -10$

Assumptions

- Set of states is known (representable in some form)
- Set of possible actions is known
- Reward function is deterministic
- Transition probabilities are stationary
- Environment is Markovian

We are still in MDP land…

All approaches we consider today (and most current approaches) make these assumptions

Monte Carlo Policy Evaluation ("Passive" RL)

- Say the agent is given a policy π
  - E.g. driver controls the car
- Still don't know:
  - $P(s'|s,a)$
  - $R(s)$

- Goal: learn the value function, $V^{\pi}(s)$ directly (without learning the transition or the reward function first)

- π can be
  - Explicit (lookup table)
  - Implicit (arbitrary function $S \rightarrow A$)

Why not Use Policy Evaluation?

$$V_0^{\pi}(s) = 0$$

$$V_{k+1}^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s\ ) + \gamma V_k^{\pi}(s')]$$

Monte Carlo Policy Evaluation

- Break up training into episodes
- Episode 1:
  - $A1, \rightarrow, B1, 0$
  - $\mathtt{B}1, \leftarrow, B1, 0$
  - $B1, \leftarrow, A1, 0$
  - $A1, \uparrow, A2, -10$
  - $A2, \uparrow, A3, 20$
  - $A3, \rightarrow, A4, -5$
  - $A4, \rightarrow, A4, 0$
  - $A1, \leftarrow\ A4, 0$
- $V(A1) =$
- $V(A2) =$
- $V(A3) =$
- $V(A4) =$

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:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

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

Append R to $Returns(s)$

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

Sutton & Barto, RL, 1st ed.

Monte Carlo Policy Evaluation

- Break up training into episodes
- Episode 1:
  - $A1, \rightarrow, B1, 0$
  - $\mathtt{B}1, \leftarrow, B1, 0$
  - $B1, \leftarrow, A1, 0$
  - $A1, \uparrow, A2, -10$
  - $A2, \uparrow, A3, 20$
  - $A3, \rightarrow, B3, -5$
  - $B3, \rightarrow, B3, 0$
  - $B3, \leftarrow B3, 0$
- $V(A1) = 5$
- $V(A2) = 15$
- $V(A3) = -5$

Future episode estimates will be incorporated in averaging $V(s)$

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:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

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

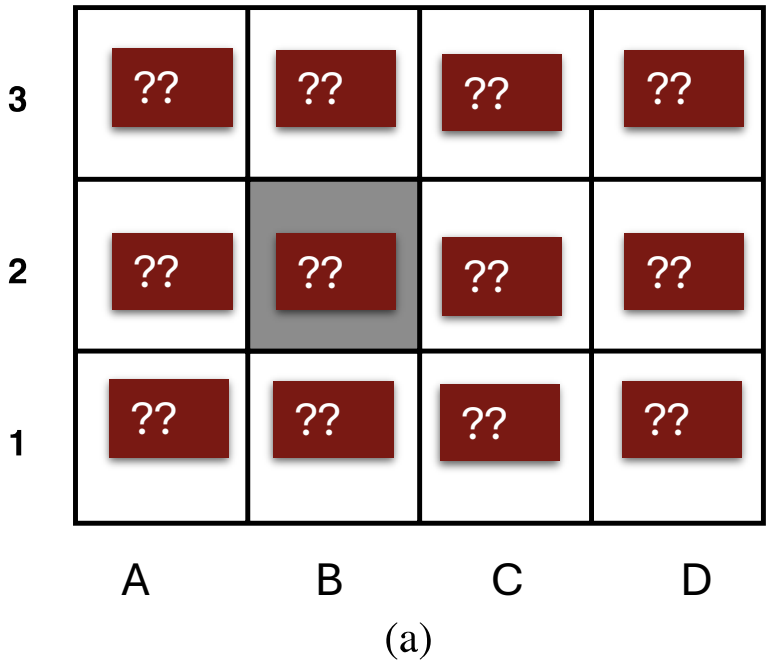
Append R to $Returns(s)$

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

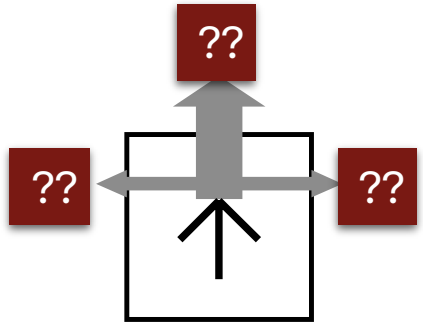
Sutton & Barto, RL, 1st ed.

Monte Carlo Policy Evaluation: Analysis

- Advantage: can focus on a subset of states; learn while collecting data
- Limitation: doesn't utilize state-reachability
  - Suppose in the next episode we get one sample:
  - $C1, \leftarrow, B1, 0$
  - MC evaluation: $V(C1) = 0$
- We already have training data from $B1$
  - $V(C1)$ could have been "bootstrapped" using it



(a)



(b)

Map not known to the agent
(or partially known)

A Simple Fix: Sample-Based Policy Evaluation

- Collect episode samples as before
- Now, bootstrap using the policy evaluation formula

$$V_{k+1}^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s) + \gamma V_k^{\pi}(s')]$$

- Since we don't know T, but we get R, we can bootstrap using current estimate of V:

samples

$$V_{k+1}(s)[1] = R(s) + \gamma V_k(s_1)$$
$$V_{k+1}(s)[2] = R(s) + \gamma V_k(s_2)$$
$$V_{k+1}(s)[3] = R(s) + \gamma V_k(s_3)$$
$$V_{k+1}(s)[4] = R(s) + \gamma V_k(s_4)$$

- $\tilde{V}^{\pi}{}_{k+1}(s) = \frac{1}{n}\sum_i V_{k+1}(s)[i]$

A Simple Fix: Sample-Based Policy Evaluation

- Collect episode samples as before
- Now, bootstrap using the policy evaluation formula

$$V_{k+1}^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s\,) + \gamma V_k^{\pi}(s')]$$

- Since we don't know T, but we get R, we can bootstrap using current estimate of V:
  - $\tilde{V}^{\pi}{}_{k+1}(s) = \frac{1}{n}\sum_i V_{k+1}(s)[i]$

- This utilizes connections between states (bootstraps) but updates V only after episodes
- Agent can't utilize available data until the estimate $\tilde{V}_{k+1}$ is computed

Next Week

- Assignment 4 Discussion
- FOND: Fully Observable Non-Deterministic Planning
- Probabilistic PDDL