Markov Decision Process II

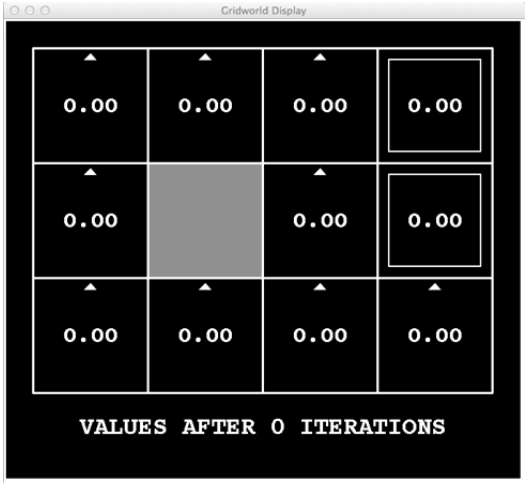
CSE 4711: Artificial Intelligence

Md. Bakhtiar Hasan

Assistant Professor
Department of Computer Science and Engineering
Islamic University of Technology

Problems with Value Iteration

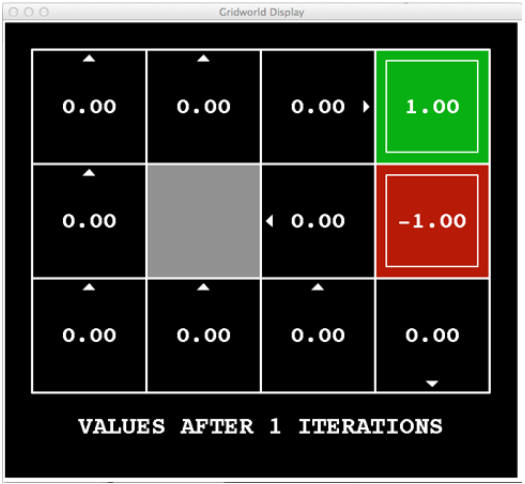
k = 0



Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 1



Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 2

Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 3

Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 4

Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 5

Noise = 0.2
Discount = 0.9
Living reward = 0

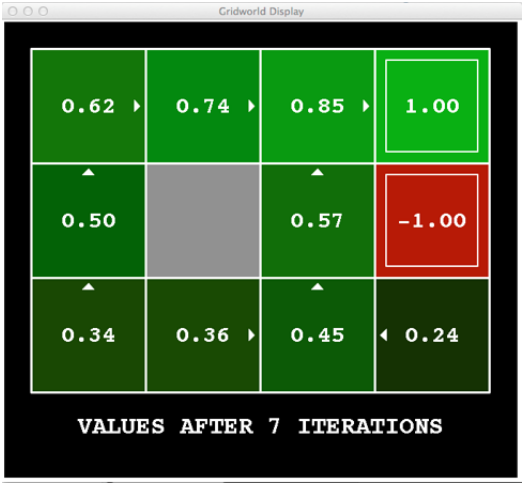
Problems with Value Iteration

k = 6

Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

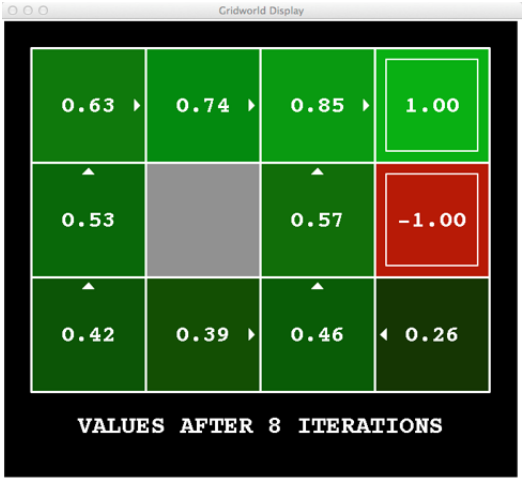
k = 7



Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

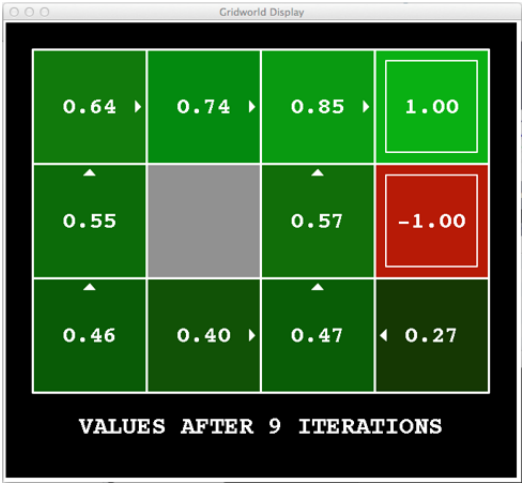
k = 8



Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 9



Noise = 0.2
Discount = 0.9
Living reward = 0

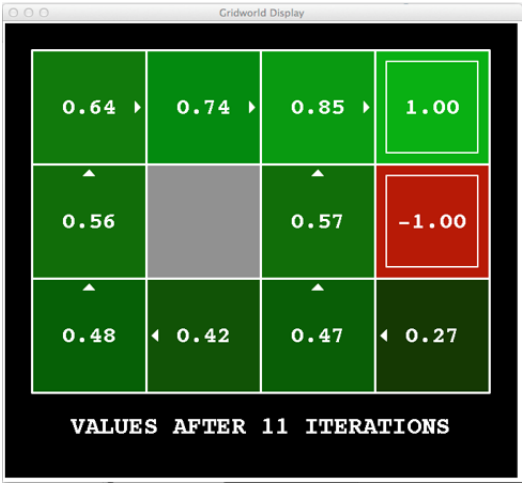
Problems with Value Iteration

k = 10

Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

k = 11



Noise = 0.2
Discount = 0.9
Living reward = 0

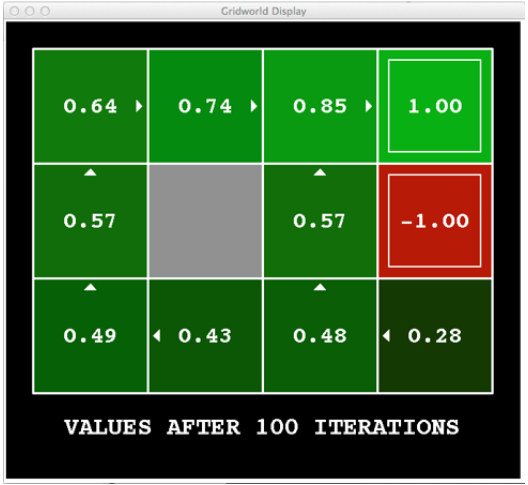
Problems with Value Iteration

k = 12

Noise = 0.2
Discount = 0.9
Living reward = 0

Problems with Value Iteration

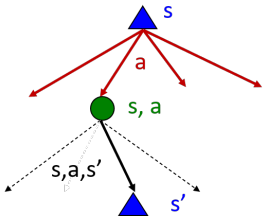
k = 100



Noise = 0.2
Discount = 0.9
Living reward = 0

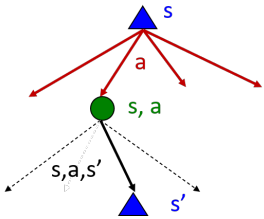
Problems with Value Iteration

- Value iteration repeats the Bellman updates:
  $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$
- Problem 1: It's slow - $O(S^2A)$ per iteration



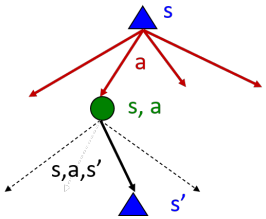
Problems with Value Iteration

- Value iteration repeats the Bellman updates:
  $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$
- Problem 1: It's slow - $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes

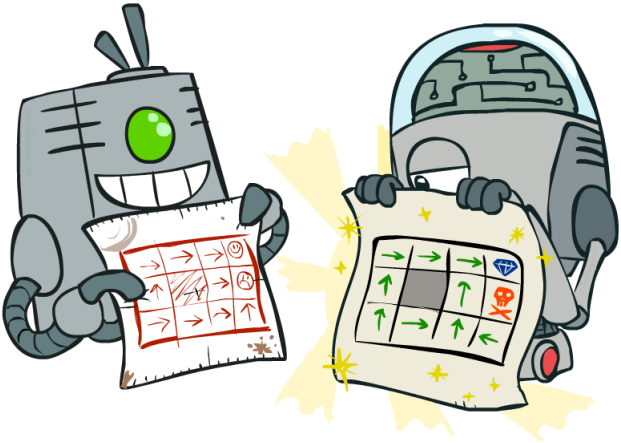


Problems with Value Iteration

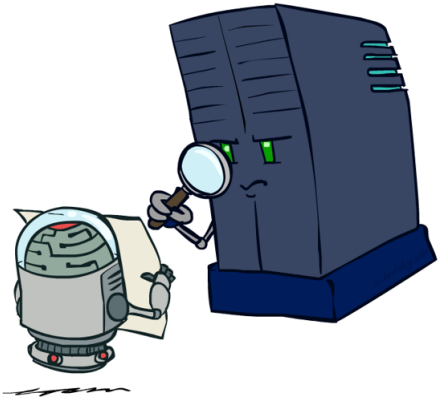
- Value iteration repeats the Bellman updates:
  $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$
- Problem 1: It's slow - $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values



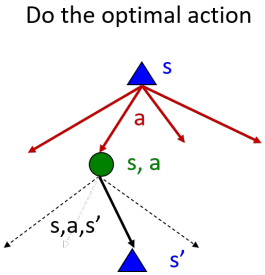
Policy Methods



Policy Evaluation

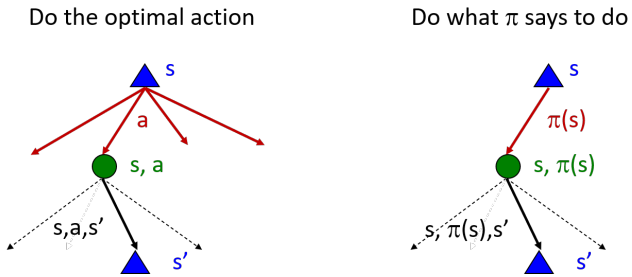


Fixed Policies



- Expectimax trees max over all actions to compute the optimal values

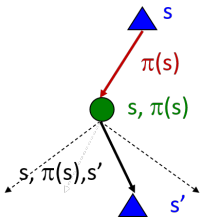
Fixed Policies



- Expectimax trees max over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only one action per state
  - . . . though the tree's value would depend on which policy we fixed

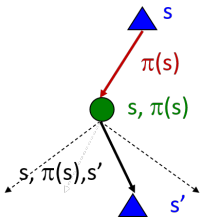
Utilities for a Fixed Policy

- Compute the utility of a state s under a fixed (generally non-optimal) policy
- Define the utility of a state s, under a fixed policy π:
  $V^{\pi}(s)$ =expected total discounted rewards starting in s and following π



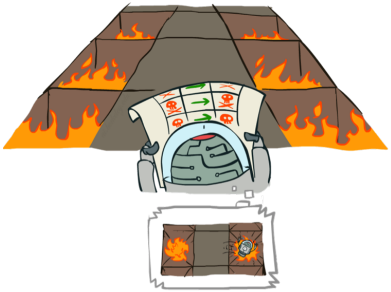
Utilities for a Fixed Policy

- Compute the utility of a state s under a fixed (generally non-optimal) policy
- Define the utility of a state s, under a fixed policy π: $V^{\pi}(s) =$expected total discounted rewards starting in s and following π
- Recursive relation (one-step look-ahead/Bellman equation): $V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s')\right]$



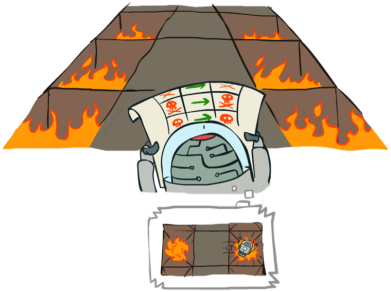
Example: Policy Evaluation

Always Go Right

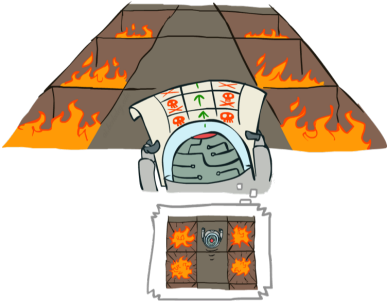


Example: Policy Evaluation

Always Go Right

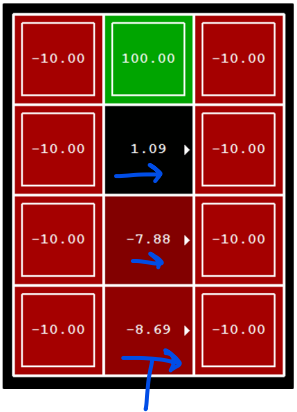


Always Go Forward



Example: Policy Evaluation

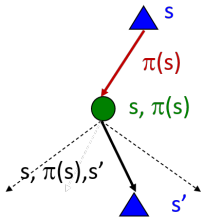
Always Go Right



Always Go Forward

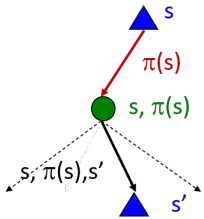
Policy Evaluation

- How do we calculate the V's for a fixed policy π?



Policy Evaluation

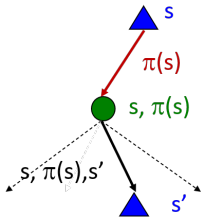
- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)



Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

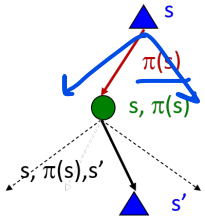
$$V_0^{\pi}(s) = 0$$



Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_k^\pi(s')\right]$$

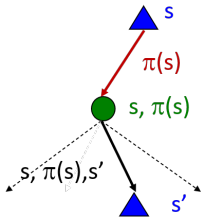


Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_k^\pi(s')\right]$$

  - Efficiency: $O(S^2)$ per iteration

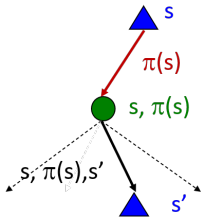


Policy Evaluation

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

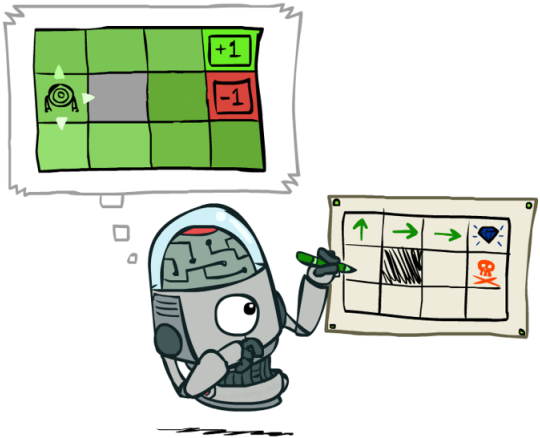
$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_k^\pi(s')\right]$$

  - Efficiency: $O(S^2)$ per iteration



- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Policy Extraction



Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?
  - It's not obvious!

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

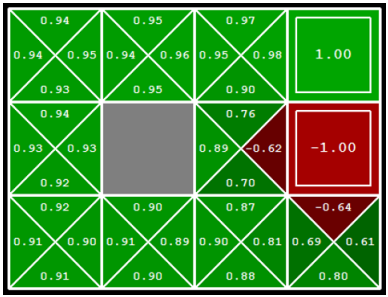
$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)
  $\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right]$
- This is called **policy extraction**, since it gets the policy implied by the values

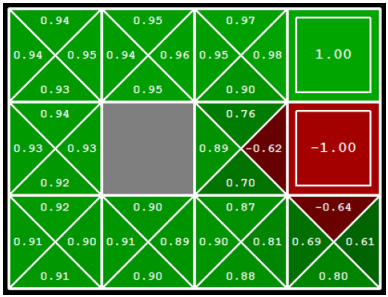
Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



Computing Actions from Q-Values

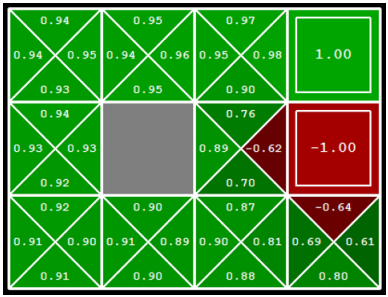
- Let's imagine we have the optimal q-values:
- How should we act?



Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a)$$

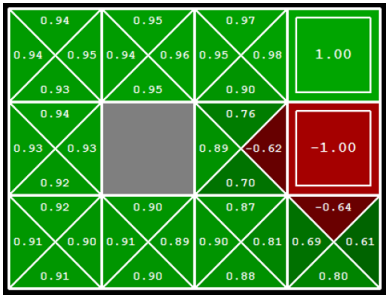


Computing Actions from Q-Values

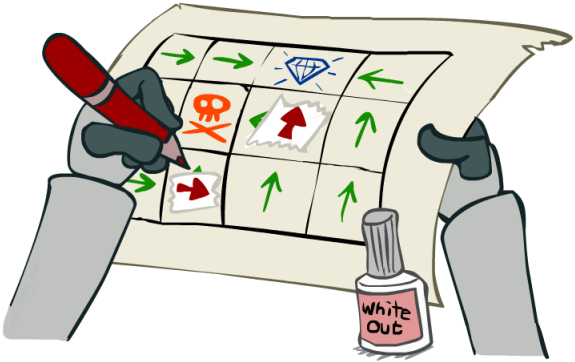
- Let's imagine we have the optimal q-values:
- How should we act?
  - Completely trivial to decide!

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a)$$

- Important: actions are easier to select from q-values than values!



Policy Iteration



Policy Iteration

- Alternative approach for optimal values:

Policy Iteration

- Alternative approach for optimal values:
  - **Step 1: Policy Evaluation** $\rightarrow$ calculate utilities for some fixed policy (not optimal utilities!) until convergence

Policy Iteration

- Alternative approach for optimal values:
  - **Step 1: Policy Evaluation** $\rightarrow$ calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy Improvement** $\rightarrow$ update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values

Policy Iteration

- Alternative approach for optimal values:
  - **Step 1: Policy Evaluation** $\rightarrow$ calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy Improvement** $\rightarrow$ update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

Policy Iteration

- Alternative approach for optimal values:
  - **Step 1: Policy Evaluation** $\rightarrow$ calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy Improvement** $\rightarrow$ update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges
- This is **policy iteration**
  - Optimal
  - Can converge (much) faster under some conditions

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:
    $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:
    $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$
- Improvement: For fixed values, get a better policy using policy extraction

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:
    $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$
- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:
    $\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it

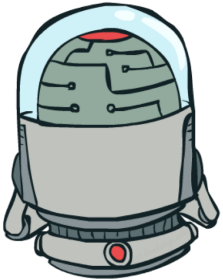
Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)

Comparison

- Both value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
  - The new policy will be better (or we're done)
- Both are dynamic programming approaches for solving MDPs

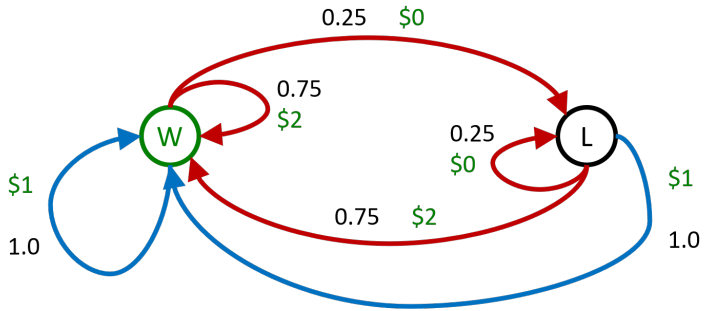
Double Bandits



Double Bandits

- Actions: *Blue*, *Red*
- States: Win, Lose

No discount
100 time steps
Both states have the same value

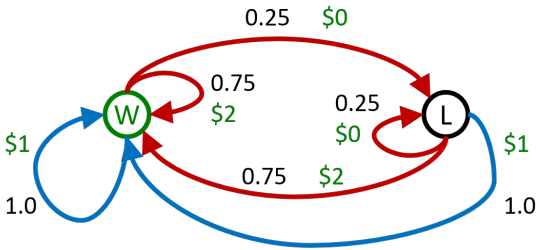


Offline Planning

- Solving MDPs is offline planning
  - You determine all quantities through computation
  - You need to know the details of the MDP
  - You do not actually play the game!

No discount
100 time steps
Both states have the same value

	Value
Play Red	150
Play Blue	100



Let's Play!

Let's Play!

$2

Let's Play!

$2 $2

Let's Play!

$2 $2 $0

Let's Play!

$2 $2 $0 $2

Let's Play!

$2 $2 $0 $2 $2

Let's Play!

$2 $2 $0 $2 $2
$2

Let's Play!

$2 $2 $0 $2 $2
$2 $2

Let's Play!

$2 $2 $0 $2 $2
$2 $2 $0

Let's Play!

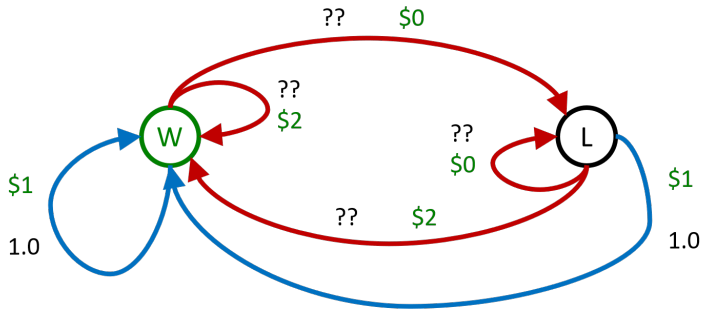
$2 $2 $0 $2 $2
$2 $2 $0 $0

Let's Play!

$$\begin{bmatrix} \$2 & \$2 & \$0 & \$2 & \$2 \\ \$2 & \$2 & \$0 & \$0 & \$0 \end{bmatrix}$$

Online Planning

- Rules changed! Red's win chance is different



Let's Play Again!

Let's Play Again!

$0

Let's Play Again!

$0 $0

Let's Play Again!

$0 $0 $0

Let's Play Again!

$0 $0 $0 $2

Let's Play Again!

$0 $0 $0 $2 $0

Let's Play Again!

$0 $0 $0 $2 $0
$2

Let's Play Again!

$0 $0 $0 $2 $0
$2 $0

Let's Play Again!

$0 $0 $0 $2 $0
$2 $0 $0

Let's Play Again!

$0 $0 $0 $2 $0
$2 $0 $0 $0

Let's Play Again!

$0 $0 $0 $2 $0
$2 $0 $0 $0 $0

What Just Happened?

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

What Just Happened?

- That wasn't planning, it was learning!
  - Specifically, reinforcement learning
  - There was an MDP, but you couldn't solve it with just computation
  - You needed to actually act to figure it out

- Important ideas in reinforcement learning that came up
  - Exploration: you have to try unknown actions to get information
  - Exploitation: eventually, you have to use what you know
  - Regret: even if you learn intelligently, you make mistakes
  - Sampling: because of chance, you have to try things repeatedly
  - Difficulty: learning can be much harder than solving a known MDP

Suggested Reading

- Russell & Norvig: Chapter 17.1-17.3
- Sutton and Barto: 3-4