CSE 473: Artificial Intelligence

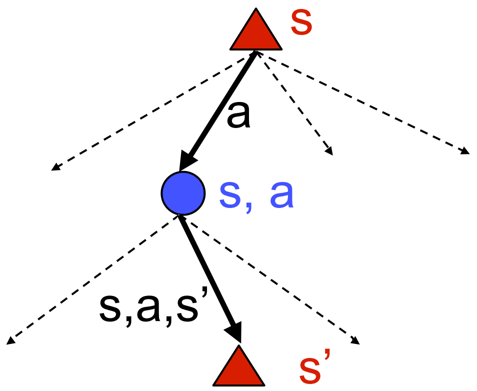
Markov Decision Processes (MDPs)

Hanna Hajishirzi

Many slides over the course adapted from Luke Zettlemoyer, Dan Klein, Pieter Abbeel, Stuart Russell or Andrew Moore

Recap: Defining MDPs

- Markov decision processes:
  - States S
  - Start state s_0
  - Actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)



- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards

Optimal Utilities

- Define the value of a state s:

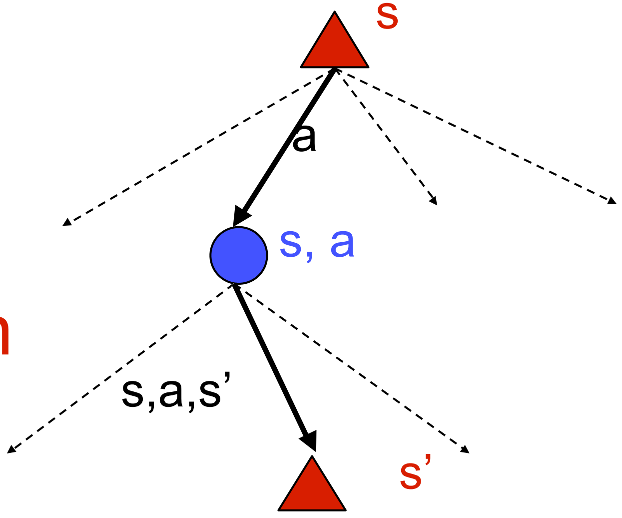
  $V^*(s)$ = expected utility starting in s and acting optimally

- Define the value of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting in s, taking action a and thereafter acting optimally

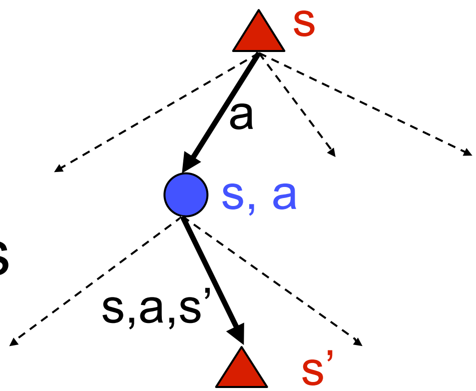
- Define the optimal policy:

  $\pi^*(s)$ = optimal action from state s



The Bellman Equations

- Definition of "optimal utility" leads to a simple one-step lookahead relationship amongst optimal utility values:
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax does



- Formally:

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

Solving MDPs

- Find V*(s) for all the states in S
  - |S| non-linear equations with |S| unknown

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

- Our proposal:
  - Dynamic programming
  - Define V*i(s) as the optimal value of s if game ends in i steps
  - V*0(s)=0 for all the states

$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

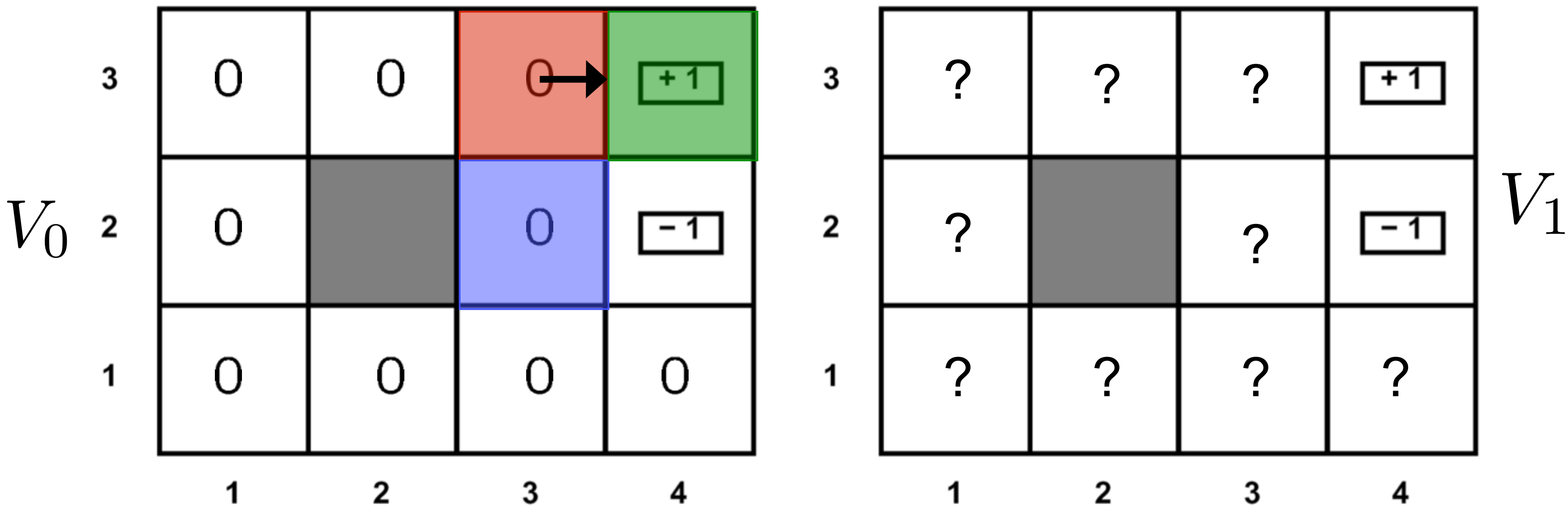
Example: γ=0.9, living reward=0, noise=0.2


0.00
0.00
0.00
1.00
0.00
0.00
-1.00
0.00
0.00
0.00
0.00
VALUES AFTER 1 ITERATIONS

VALUES AFTER 2 ITERATIONS

Example: γ=0.9, living reward=0, noise=0.2

Example: Bellman Updates



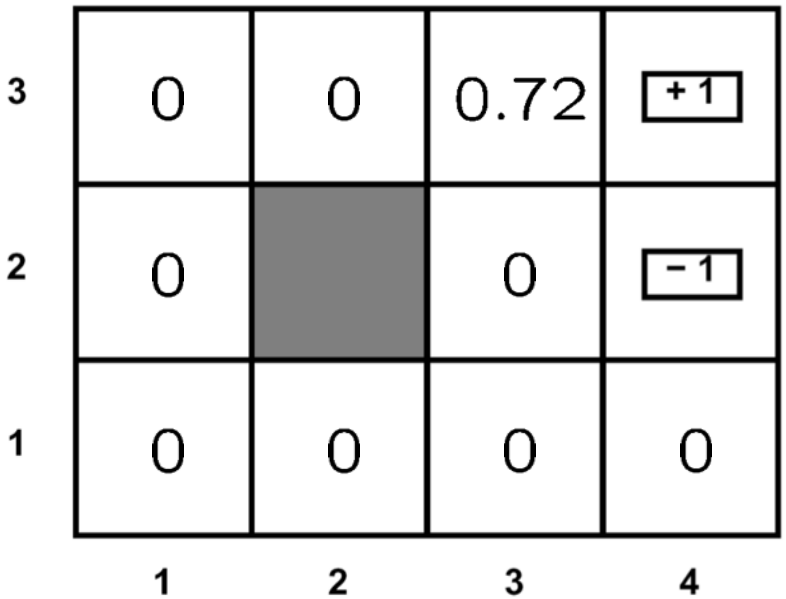
$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_i(s')\right] = \max_a Q_{i+1}(s,a)$$

$$Q_1(\langle 3,3\rangle, \text{right}) = \sum_{s'} T(\langle 3,3\rangle, \text{right}, s')\left[R(\langle 3,3\rangle, \text{right}, s') + \gamma V_i(s')\right]$$

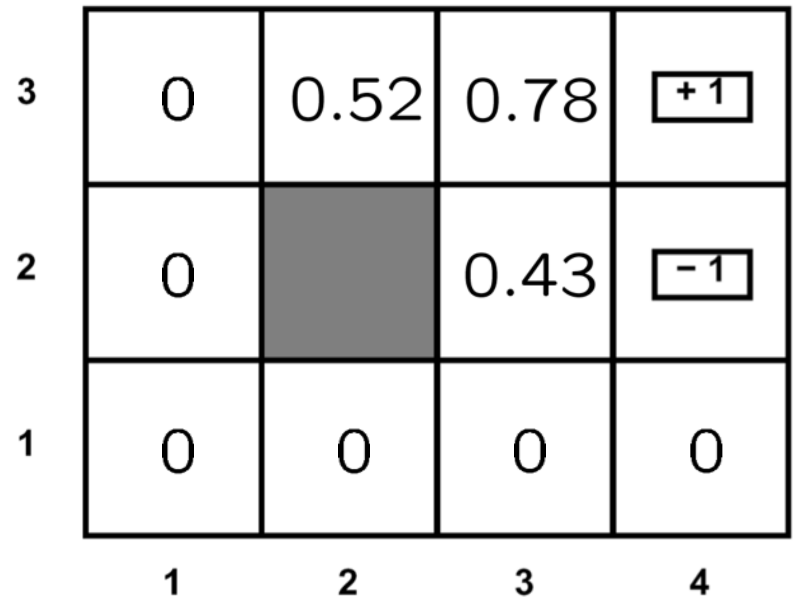
$$= 0.8 * [0.0 + 0.9 * 1.0] + 0.1 * [0.0 + 0.9 * 0.0] + 0.1 * [0.0 + 0.9 * 0.0]$$

Example: Value Iteration

V_1



V_2



- Information propagates outward from terminal states and eventually all states have correct value estimates

VALUES AFTER 3 ITERATIONS

VALUES AFTER 4 ITERATIONS

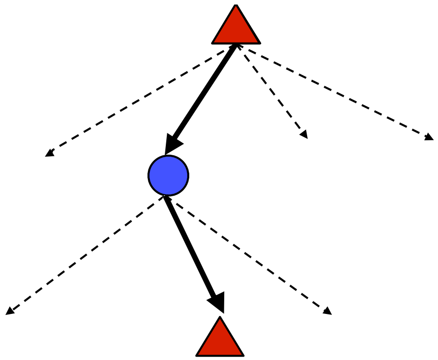

0.51
0.72
0.84
1.00
0.27
0.55
-1.00
0.00
0.22
0.37
0.13
VALUES AFTER 5 ITERATIONS

VALUES AFTER 6 ITERATIONS


0.62
0.74
0.85
1.00
0.50
0.57
-1.00
0.34
0.36
0.45
0.24
VALUES AFTER 7 ITERATIONS

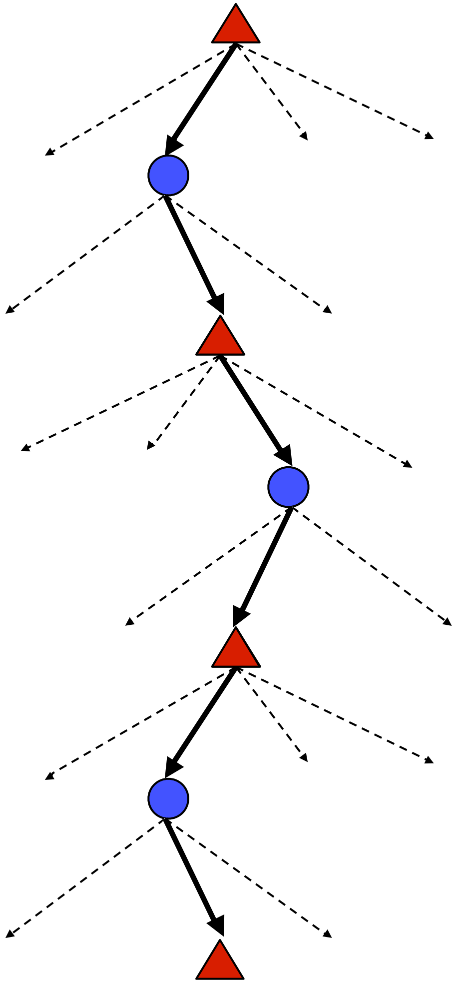
Value Estimates

- Calculate estimates $V_k^*(s)$
  - The optimal value considering only next k time steps (k rewards)
  - As $k \rightarrow \infty$, it approaches the optimal value
- Why:
  - If discounting, distant rewards become negligible
  - If terminal states reachable from everywhere, fraction of episodes not ending becomes negligible
  - Otherwise, can get infinite expected utility and then this approach actually won't work

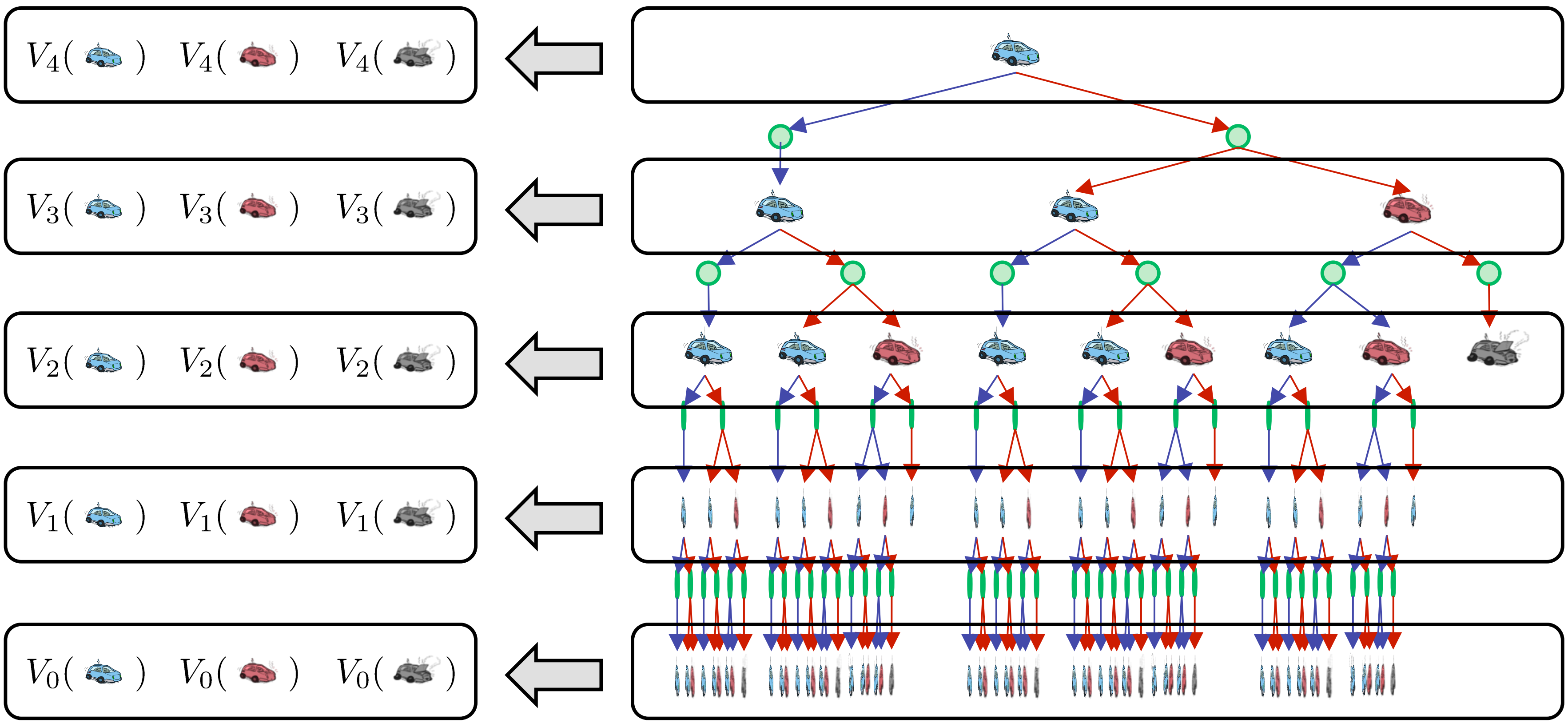


Why Not Search Trees?

- Why not solve with expectimax?
- Problems:
  - This tree is usually infinite (why?)
  - Same states appear over and over (why?)
  - We would search once per state (why?)
- Idea: Value iteration
  - Compute optimal values for all states all at once using successive approximations
  - Will be a bottom-up dynamic program similar in cost to memoization
  - Do all planning offline, no replanning needed!

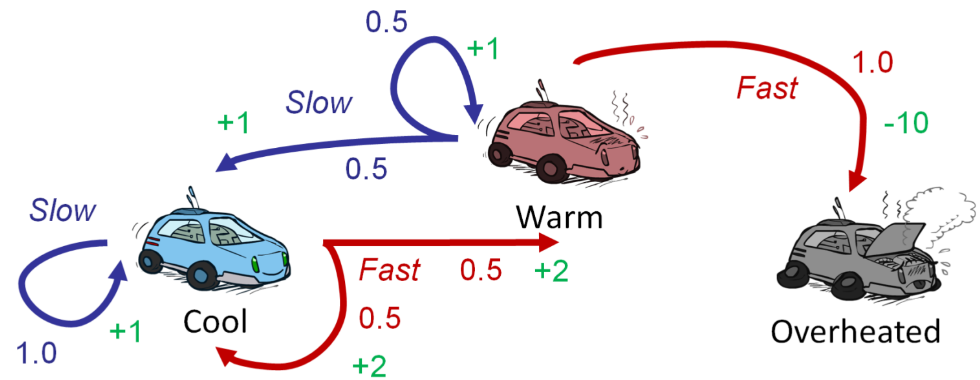


Computing time limited values



Example of Value iteration

V_2	3.5	2.5	0
V_1	2	1	0
V_0	0	0	0



Assume no discount!

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

Value Iteration

- Idea:
  - Start with $V_0^*(s) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth i+1:

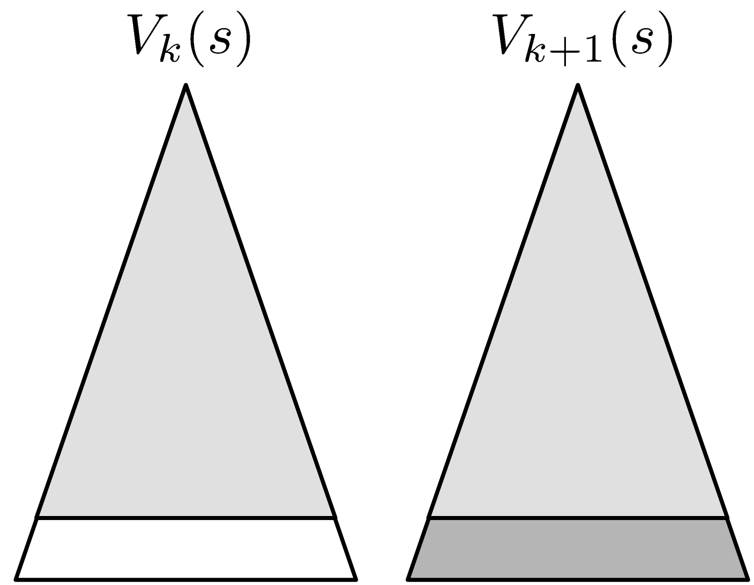
$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

  - This is called a value update or Bellman update
  - Repeat until convergence

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

Convergence

- How do we know the V_k vectors are going to converge?
- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: For any state V_k and V_{k+1} can be viewed as depth k +1 expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
  - That last layer is at best all R_{MAX}
  - It is at worst R_{MIN}
  - But everything is discounted by γ^k that far out
  - So V_k and V_{k+1} are at most γ^k max|R| different
  - So as k increases, the values converge



Value Iteration Complexity

- Problem size:
  - $|A|$ actions and $|S|$ states
- Each Iteration
  - Computation: $O(|A| \cdot |S|^2)$
  - Space: $O(|S|)$
- Num of iterations
  - Can be exponential in the discount factor γ

Computing Actions from Values

Computing Actions from Values

Computing Actions from Values

- Which action should we chose from state s:
  - Given optimal values Q?

    $$\arg\max_a Q^*(s,a)$$

  - Given optimal values V?

    $$\arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

  - Lesson: actions are easier to select from Q's!

Aside: Q-Value Iteration

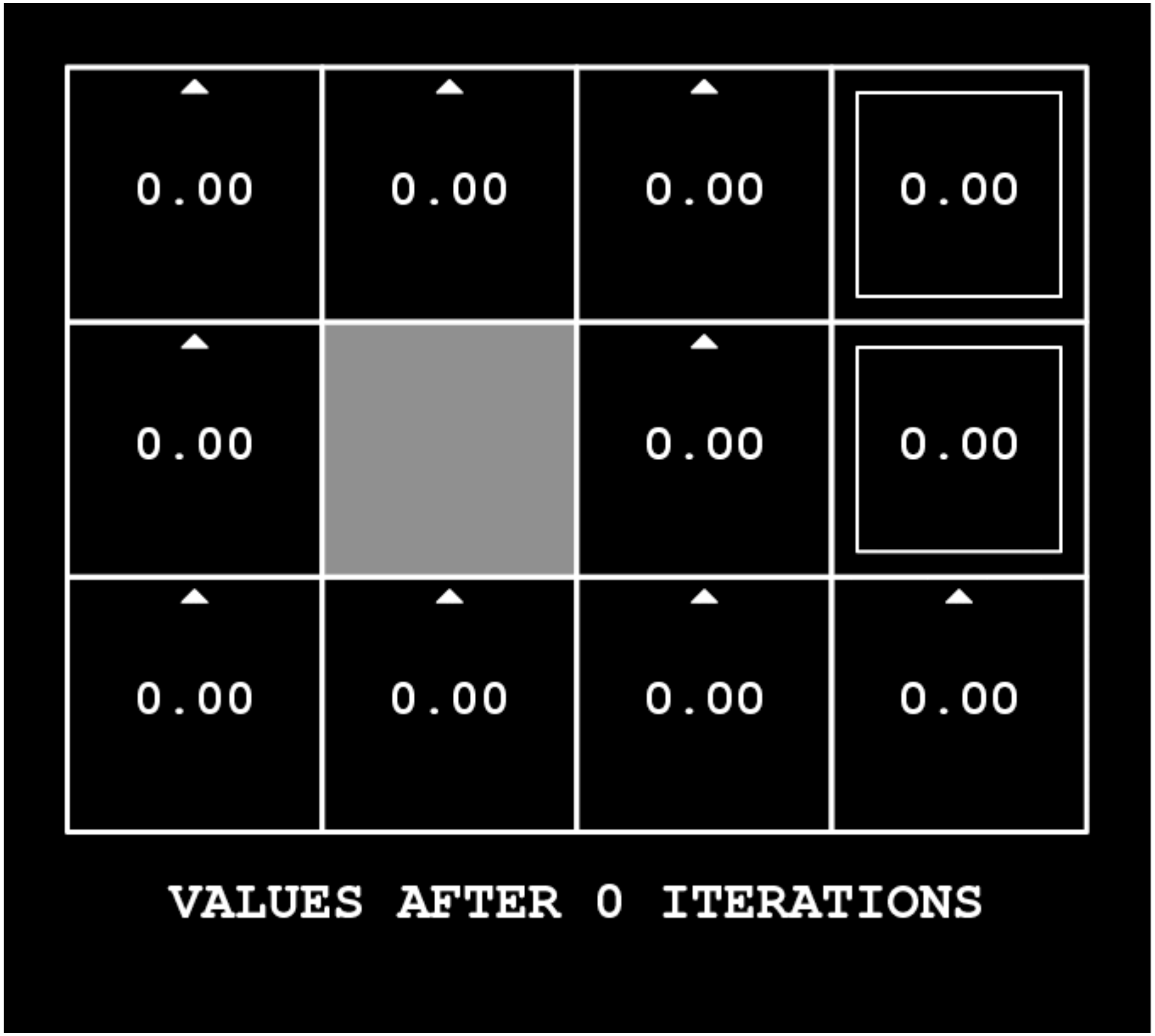
- Value iteration: find successive approx optimal values
  - Start with $V_0^*(s) = 0$
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s')\right]$$

- But Q-values are more useful!
  - Start with $Q_0^*(s,a) = 0$
  - Given Q_i^*, calculate the q-values for all q-states for depth i+1:

$$Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_i(s',a')\right]$$

Example: Value Iteration



Outline

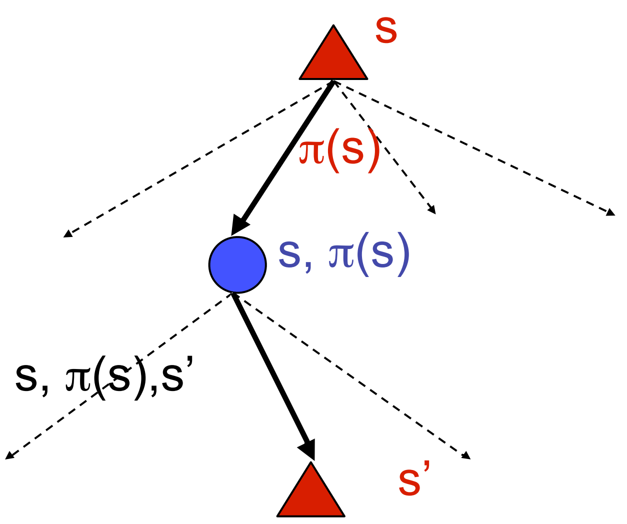
- Markov Decision Processes (MDPs)
  - MDP formalism
  - Value Iteration
  - Policy Iteration
- Reinforcement Learning (RL)
  - Relationship to MDPs
  - Several learning algorithms

Utilities for Fixed Policies

- Another basic operation: compute the utility of a state s under a fix (general non-optimal) policy
- Define the utility of a state s, under a fixed policy π:

  $V^{\pi}(s)$ = expected total discounted rewards (return) starting in s and following π

- Recursive relation (one-step look-ahead / Bellman equation):



$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

Policy Evaluation

Always Go Right

Always Go Forward

Policy Evaluation

- How do we calculate the V's for a fixed policy?
- Idea one: modify Bellman updates

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

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

- Idea two: it's just a linear system, solve with Matlab (or whatever)

Policy Iteration

- Problem with value iteration:
  - Considering all actions each iteration is slow: takes |A| times longer than policy evaluation
  - But policy doesn't change each iteration, time wasted
- Alternative to value iteration:
  - Step 1: Policy evaluation: calculate utilities for a fixed policy (not optimal utilities!) until convergence (fast)
  - Step 2: Policy improvement: update policy using one-step lookahead with resulting converged (but not optimal!) utilities (slow but infrequent)
  - Repeat steps until policy converges

Policy Iteration

- Policy evaluation: with fixed current policy π, find values with simplified Bellman updates
  - Iterate until values converge

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s,\pi_k(s),s') \left[R(s,\pi_k(s),s') + \gamma\, V_i^{\pi_k}(s')\right]$$

  - Note: could also solve value equations with other techniques

- Policy improvement: with fixed utilities, get a better policy
  - find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi_k}(s')\right]$$

Policy Iteration Complexity

- Problem size:
  - $|A|$ actions and $|S|$ states
- Each Iteration
  - Computation: $O(|S|^3 + |A| \cdot |S|^2)$
  - Space: $O(|S|)$
- Num of iterations
  - Unknown, but can be faster in practice
  - Convergence is guaranteed

Comparison

- In value iteration:
  - Every pass (or “backup”) updates both utilities (explicitly, based on current utilities) and policy (possibly implicitly, based on current policy)
- In policy iteration:
  - Several passes to update utilities with frozen policy
  - After a policy is evaluated, a new policy is chosen
  - The new policy is better (or we are done)
- Hybrid approaches (asynchronous policy iteration):
  - Any sequences of partial updates to either policy entries or utilities will converge if every state is visited infinitely often

Summary: MDP Algorithms

So you want to ....

- Compute opimal values: use value iteration or policy iteration
- Compute values for a particular policy: use policy evaluation
- Turn your values into a policy: use policy extraction (one-step lookahead)

- These all look the same!

  They basically are – they are all variations of Bellman updates

  They all use one-step lookahead expectimax fragments

  They differ only in whether we plug in a fixed policy or max over actions