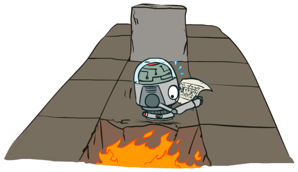
CS 188: Artificial Intelligence

Markov Decision Processes II



Instructor: Marco Alvarez

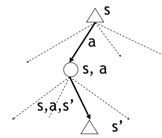
University of Rhode Island

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

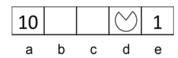
1

Recap: Defining MDPs

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

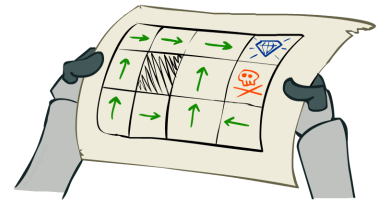


- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards



2

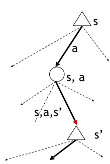
Solving MDPs



3

Optimal Quantities

- The value (utility) of a state s:
  $V^*(s)$ = expected utility starting in s and acting optimally
- The value (utility) of a q-state (s,a):
  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally
- The optimal policy:
  $\pi^*(s)$ = optimal action from state s



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

[Demo – gridworld values (L8D4)]

4

Snapshot of Demo – Gridworld V Values

Noise = 0
Discount = 1
Living reward = 0

5

Snapshot of Demo – Gridworld Q Values

Noise = 0
Discount = 1
Living reward = 0

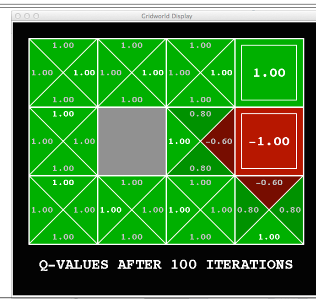
6

Snapshot of Demo – Gridworld V Values

Noise = 0.2
Discount = 1
Living reward = 0

7

Snapshot of Demo – Gridworld Q Values



Noise = 0.2
Discount = 1
Living reward = 0

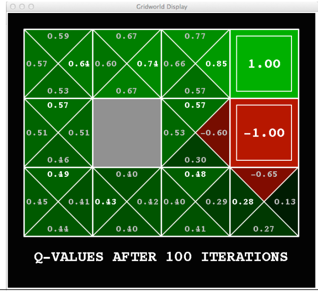
8

Snapshot of Demo – Gridworld V Values

Noise = 0.2
Discount = 0.9
Living reward = 0

9

Snapshot of Demo – Gridworld Q Values



Noise = 0.2
Discount = 0.9
Living reward = 0

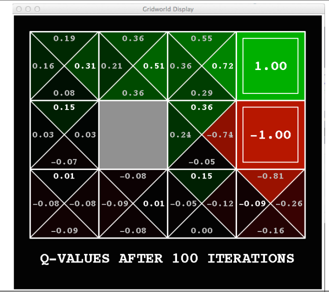
10

Snapshot of Demo – Gridworld V Values

Noise = 0.2
Discount = 0.9
Living reward = -0.1

11

Snapshot of Demo – Gridworld Q Values

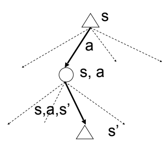


Noise = 0.2
Discount = 0.9
Living reward = -0.1

12

Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!
- Recursive definition of value:



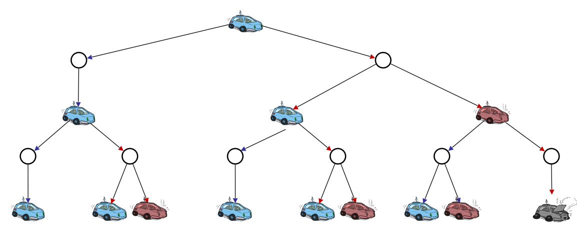
$$V^*(s) = \max_a Q^*(s,a)$$

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

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

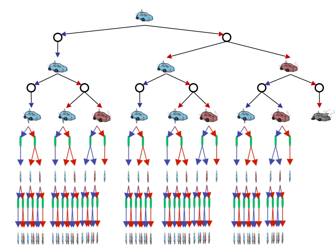
13

Racing Search Tree



14

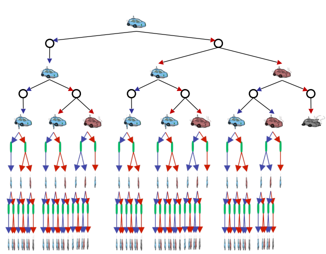
Racing Search Tree



15

Racing Search Tree

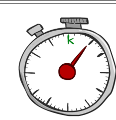
- We're doing way too much work with expectimax!
- Problem: States are repeated
  - Idea: Only compute needed quantities once
- Problem: Tree goes on forever
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don't matter if $\gamma < 1$

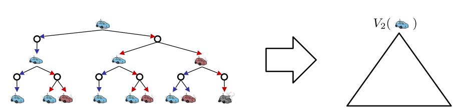


16

Time-Limited Values

- Key idea: time-limited values
- Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps
  - Equivalently, it's what a depth-k expectimax would give from s

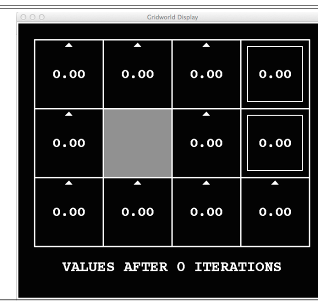




[Demo – time-limited values (L8D6)]

17

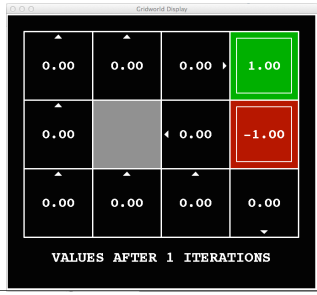
k=0



Noise = 0.2
Discount = 0.9
Living reward = 0

18

k=1



Noise = 0.2
Discount = 0.9
Living reward = 0

19

k=2

Noise = 0.2
Discount = 0.9
Living reward = 0

20

k=3

Noise = 0.2
Discount = 0.9
Living reward = 0

21

k=4

Noise = 0.2
Discount = 0.9
Living reward = 0

22

k=5

Noise = 0.2
Discount = 0.9
Living reward = 0

23

k=6

Noise = 0.2
Discount = 0.9
Living reward = 0

24

k=7

Noise = 0.2
Discount = 0.9
Living reward = 0

25

k=8

Noise = 0.2
Discount = 0.9
Living reward = 0

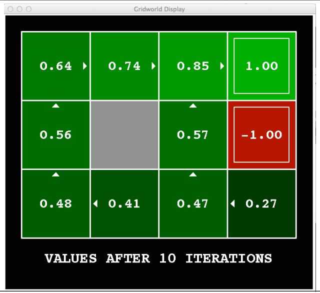
26

k=9

Noise = 0.2
Discount = 0.9
Living reward = 0

27

k=10



Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

Noise = 0.2
Discount = 0.9
Living reward = 0

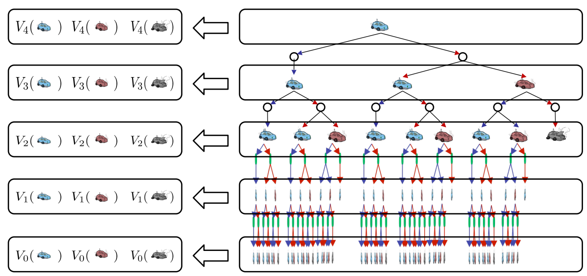
k=12

Noise = 0.2
Discount = 0.9
Living reward = 0

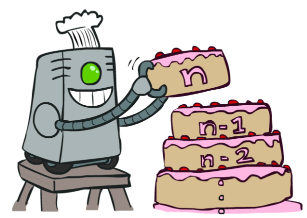
k=100

Noise = 0.2
Discount = 0.9
Living reward = 0

Computing Time-Limited Values



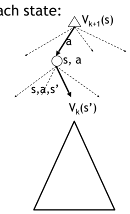
Value Iteration



Value Iteration

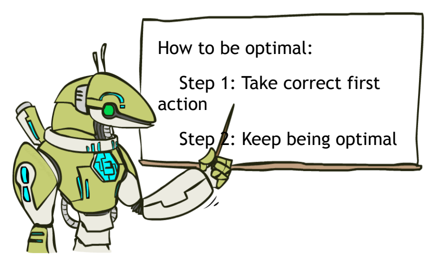
- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

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



- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

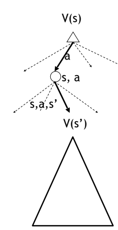
The Bellman Equations



Value Iteration

- Bellman equations characterize the optimal values:

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



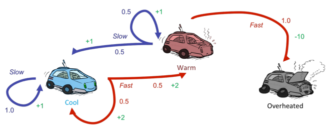
- Value iteration computes them:

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

- Value iteration is just a fixed point solution method
  - … though the V_k vectors are also interpretable as time-limited values

Example: Value Iteration

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



Assume no discount!

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

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