CS 188: Artificial Intelligence

Markov Decision Processes



Instructors: Dan Klein and Pieter Abbeel

University of California, Berkeley

[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.]

Recap: Defining MDPs

- Markov decision processes:
 - Set of states S
 - Start state s₀
 - Set of actions A
 - Transitions P(s'|s,a) (or T(s,a,s'))
 - Rewards R(s,a,s') (and discount γ)



- Policy = Choice of action for each state
- Utility = sum of (discounted) rewards



Solving MDPs



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



Snapshot of Demo – Gridworld V Values

00	0	Gridworl	d Display	
	0.64)	0.74 →	0.85)	1.00
	•		• 0.57	-1.00
	• 0.49	∢ 0.43	• 0.48	∢ 0.28
	VALUES	SAFTER 1	LOO ITERA	ATIONS

Snapshot of Demo – Gridworld Q Values



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:

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



Racing Search Tree



Racing Search Tree



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 γ < 1



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





0.0	Gridworl	d Display	
	^	^	
0.00	0.00	0.00	0.00
		^	
0.00		0.00	0.00
	^	^	^
0.00	0.00	0.00	0.00
VALUES AFTER O TTERATIONS			

0	0	Gridworl	d Display	
	▲ 0.00	• 0.00	0.00)	1.00
	•		∢ 0.00	-1.00
	•	• 0.00	• 0.00	0.00
	VALUES AFTER 1 ITERATIONS			

k=2

0 0	Gridworl	d Display	al and "Restored al advances they "New as you manual way"
• 0.00	0.00 >	0.72 ▸	1.00
•		• 0.00	-1.00
• 0.00	• 0.00	• 0.00	0.00
VALUES AFTER 2 ITERATIONS			

k=3

0	0	Gridworl	d Display	
	0.00)	0.52 →	0.78)	1.00
	• 0.00		• 0.43	-1.00
	• 0.00	• 0.00	• 0.00	0.00
	VALUES AFTER 3 ITERATIONS			

k=4

0 0	Gridworl	d Display	
0.37 ▸	0.66)	0.83)	1.00
•		• 0.51	-1.00
• 0.00	0.00 →	• 0.31	∢ 0.00
VALUE	S AFTER	4 ITERA	FIONS

k=5

0.0	0	Gridworl	d Display	
	0.51)	0.72)	0.84)	1.00
	• 0.27		• 0.55	-1.00
	•	0.22 →	• 0.37	∢ 0.13
	VALUE	S AFTER	5 ITERA	TIONS

k=6

Gridworld Display					
	0.59)	0.73)	0.85)	1.00	
	• 0.41		• 0.57	-1.00	
	• 0.21	0.31 →	▲ 0.43	∢ 0.19	
	VALUES AFTER 6 ITERATIONS				

00	Gridworl	d Display	
0.62)	0.74 →	0.85 →	1.00
• 0.50		• 0.57	-1.00
▲ 0.34	0.36 →	▲ 0.45	◀ 0.24
VALU	ES AFTER	7 ITERA	FIONS

k=8

0 0	Gridworl	d Display	
0.63)	0.74 →	0.85)	1.00
• 0.53		• 0.57	-1.00
• 0.42	0.39)	• 0.46	∢ 0.26
VALUE	S AFTER	8 ITERA	LIONS

k=9

0 0	0	Gridworl	d Display	
ſ	0.64)	0.74 ▸	0.85)	1.00
	• 0.55		• 0.57	-1.00
	• 0.46	0.40 →	• 0.47	◀ 0.27
	VALUE	S AFTER	9 ITERA	FIONS

00		Gridworld	d Display	
	0.64 →	0.74 ›	0.85)	1.00
	• 0.56		• 0.57	-1.00
	• 0.48	∢ 0.41	• 0.47	◀ 0.27
	VALUE	S AFTER	10 ITERA	TIONS

0 0	C C Gridworld Display					
	0.64)	0.74 →	0.85 →	1.00		
	• 0.56		• 0.57	-1.00		
	•	◀ 0.42	• 0.47	∢ 0.27		
VALUES AFTER 11 ITERATIONS						

Gridworld Display						
0.64	▶ 0.74 ▶	0.85)	1.00			
• 0.57		• 0.57	-1.00			
• 0.49	◀ 0.42	0. 47	∢ 0.28			
VALUES AFTER 12 ITERATIONS						

O O Gridworld Display					
0.64)	0.74 →	0.85)	1.00		
•		•			
0.57		0.57	-1.00		
^		^			
0.49	∢ 0.43	0.48	∢ 0.28		
VALUES AFTER 100 ITERATIONS					

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(S²A)
- Theorem: will converge to unique optimal values
 - Basic idea: approximations get refined towards optimal values
 - Policy may converge long before values do



Example: Value Iteration



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 $\gamma^k \max |R|$ different
 - So as k increases, the values converge

