

CS 188: Artificial Intelligence Spring 2011

Lecture 9: MDPs 2/16/2011

Pieter Abbeel – UC Berkeley
Many slides over the course adapted from either Dan Klein,
Stuart Russell or Andrew Moore

Announcements

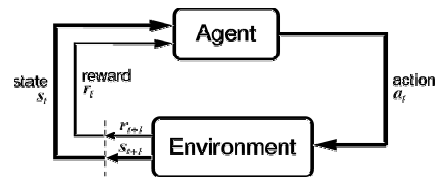
- Midterm: Tuesday March 15, 5-8pm
- P2: Due Friday 4:59pm
- W3: Minimax, expectimax and MDPs---out tonight, due Monday February 28.
- Online book: Sutton and Barto
<http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html>

Outline

- Markov Decision Processes (MDPs)
 - Formalism
 - Value iteration
- Expectimax Search vs. Value Iteration
 - Value Iteration:
 - No exponential blow-up with depth [cf. graph search vs. tree search]
 - Can handle infinite duration games
- Policy Evaluation and Policy Iteration

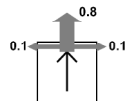
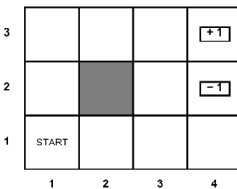
Reinforcement Learning

- Basic idea:
 - Receive feedback in the form of **rewards**
 - Agent's utility is defined by the reward function
 - Must learn to act so as to **maximize expected rewards**



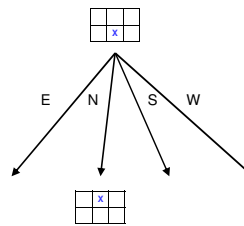
Grid World

- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
 - 80% of the time, the action North takes the agent North (if there is no wall there)
 - 10% of the time, North takes the agent West; 10% East
 - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards

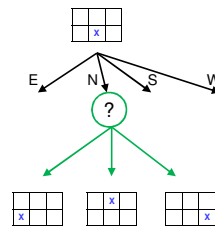


Grid Futures

Deterministic Grid World

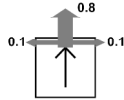
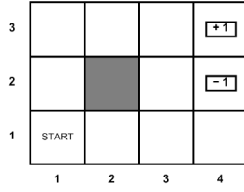


Stochastic Grid World



Markov Decision Processes

- An MDP is defined by:
 - A set of states $s \in S$
 - A set of actions $a \in A$
 - A transition function $T(s, a, s')$
 - Prob that a from s leads to s' i.e., $P(s' | s, a)$
 - Also called the model
 - A reward function $R(s, a, s')$
 - Sometimes just $R(s)$ or $R(s')$
 - A start state (or distribution)
 - Maybe a terminal state
- MDPs are a family of non-deterministic search problems
 - Reinforcement learning: MDPs where we don't know the transition or reward functions



7

What is Markov about MDPs?

- Andrey Markov (1856-1922)
- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means:

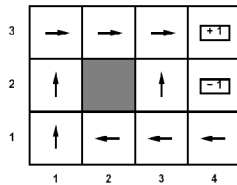


$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0) = P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Solving MDPs

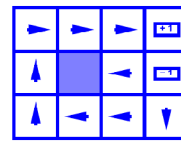
- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
 - A policy π gives an action for each state
 - An optimal policy maximizes expected utility if followed
 - Defines a reflex agent

Optimal policy when $R(s, a, s') = -0.03$ for all non-terminals s

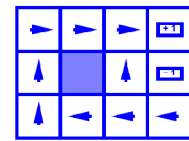


7

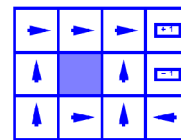
Example Optimal Policies



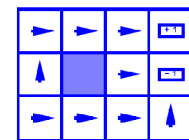
$R(s) = -0.01$



$R(s) = -0.03$



$R(s) = -0.4$

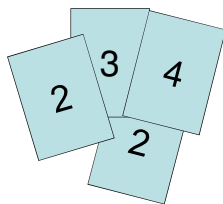


$R(s) = -2.0$

11

Example: High-Low

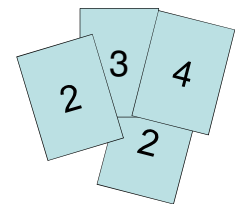
- Three card types: 2, 3, 4
- Infinite deck, twice as many 2's
- Start with 3 showing
- After each card, you say "high" or "low"
- New card is flipped
- If you're right, you win the points shown on the new card
- Ties are no-ops
- If you're wrong, game ends
- Differences from expectimax:
 - #1: get rewards as you go
 - #2: you might play forever!

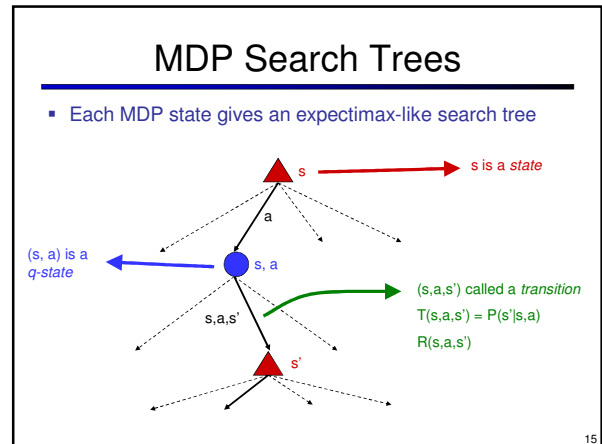
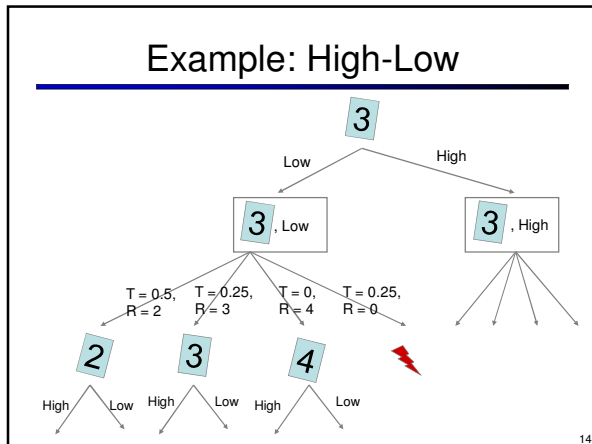


12

High-Low as an MDP

- States: 2, 3, 4, done
- Actions: High, Low
- Model: $T(s, a, s')$:
 - $P(s'=4 | 4, \text{Low}) = 1/4$
 - $P(s'=3 | 4, \text{Low}) = 1/4$
 - $P(s'=2 | 4, \text{Low}) = 1/2$
 - $P(s'=\text{done} | 4, \text{Low}) = 0$
 - $P(s'=4 | 4, \text{High}) = 1/4$
 - $P(s'=3 | 4, \text{High}) = 0$
 - $P(s'=2 | 4, \text{High}) = 0$
 - $P(s'=\text{done} | 4, \text{High}) = 3/4$
 - ...
- Rewards: $R(s, a, s')$:
 - Number shown on s' if $s \neq s'$ and a is "correct"
 - 0 otherwise
- Start: 3





Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider **stationary preferences**:

$$[r_0, r_1, r_2, \dots] > [r'_0, r'_1, r'_2, \dots]$$

$$\Leftrightarrow [r_0, r_1, r_2, \dots] > [\gamma^0 r'_0, \gamma^1 r'_1, \gamma^2 r'_2, \dots]$$
- Theorem: only two ways to define stationary utilities**
 - Additive utility:

$$U([r_0, r_1, r_2, \dots]) = r_0 + r_1 + r_2 + \dots$$
 - Discounted utility:

$$U([r_0, r_1, r_2, \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

Infinite Utilities?!

- Problem: infinite state sequences have infinite rewards
- Solutions:
 - Finite horizon:
 - Terminate episodes after a fixed T steps (e.g. life)
 - Gives nonstationary policies (π depends on time left)
 - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "done" for High-Low)
 - Discounting: for $0 < \gamma < 1$

$$U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max} / (1 - \gamma)$$
 - Smaller γ means smaller "horizon" – shorter term focus

Discounting

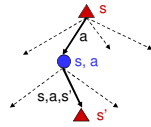
- Typically discount rewards by $\gamma < 1$ each time step
 - Sooner rewards have higher utility than later rewards
 - Also helps the algorithms converge

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

- Fundamental operation: compute the values (optimal expectimax utilities) of states s
- Why? Optimal values define optimal policies!
- 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

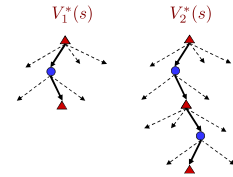


3	0.912	0.858	0.912	←
2	0.742	0.660	0.660	↑
1	0.705	0.655	0.611	←
	1	2	3	4

21

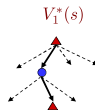
Value Estimates

- Calculate estimates $V_k^i(s)$
 - Not the optimal value of $s!$
 - The optimal value considering only next k time steps (k rewards)
 - As $k \rightarrow \infty$, it approaches the optimal value
- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming



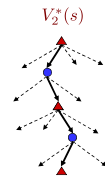
22

Value Iteration: V_1^*



23

Value Iteration: V_2^*



24

Value Iteration V_{i+1}^*

25

Value Iteration

- Idea:
 - $V_i^*(s)$: the expected discounted sum of rewards accumulated when starting from state s and acting optimally for a horizon of i time steps.
 - Start with $V_0^*(s) = 0$, which we know is right (why?)
 - Given V_i^* , calculate the values for all states for horizon $i+1$:
- $$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_i^*(s')]$$
- 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

26

Example: $\gamma=0.9$, living reward=0, noise=0.2

Example: Bellman Updates

3	0	0	0	1
2	0	0	0	-1
1	0	0	0	0
	1	2	3	4

3	0	0	0.72	1
2	0	0	0	-1
1	0	0	0	0
	1	2	3	4

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

$V_2((3, 3)) = \sum_{s'} T((3, 3), \text{right}, s') [R((3, 3)) + 0.9 V_1(s')]$

max happens for a=right, other actions not shown
 $= 0.9 [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$

27

Convergence*

- Define the max-norm: $\|U\| = \max_s |U(s)|$
- Theorem: For any two approximations U and V

$$\|U_{i+1} - V_{i+1}\| \leq \gamma \|U_i - V_i\|$$
 - I.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true U and value iteration converges to a unique, stable, optimal solution
- Theorem:

$$\|U_{i+1} - U_i\| < \epsilon \Rightarrow \|U_{i+1} - U\| < 2\epsilon\gamma/(1 - \gamma)$$
 - I.e. once the change in our approximation is small, it must also be close to correct

29

At Convergence

- At convergence, we have found the optimal value function V^* for the discounted infinite horizon problem, which satisfies the Bellman equations:

$$\forall s \in S: V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

30

The Bellman Equations

- Definition of "optimal utility" leads to a simple one-step lookahead relationship amongst optimal utility values:

Optimal rewards = maximize over first action and then follow optimal policy

- Formally:

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

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

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

31

Practice: Computing Actions

- Which action should we choose from state s:
 - Given optimal values V?

$$\arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$
 - Given optimal q-values Q?

$$\arg \max_a Q^*(s, a)$$
- Lesson: actions are easier to select from Q's!

32

Complete Procedure

- 1. Run value iteration (off-line)
 - Returns V, which (assuming sufficiently many iterations is a good approximation of V^*)
- 2. Agent acts. At time t the agent is in state s_t and takes the action a_t :

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

33

Complete Procedure

34

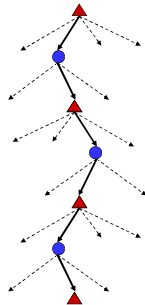
Outline

- Markov Decision Processes (MDPs)
 - Formalism
 - Value iteration
- Expectimax Search vs. Value Iteration
 - Value Iteration:
 - No exponential blow-up with depth [cf. graph search vs. tree search]
 - Can handle infinite duration games
- Policy Evaluation and Policy Iteration

38

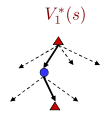
Why Not Search Trees?

- 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!



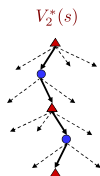
40

Expectimax vs. Value Iteration: V_1^*



41

Expectimax vs. Value Iteration: V_2^*



42

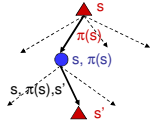
Outline

- Markov Decision Processes (MDPs)
 - Formalism
 - Value iteration
- Expectimax Search vs. Value Iteration
 - Value Iteration:
 - No exponential blow-up with depth [cf. graph search vs. tree search]
 - Can handle infinite duration games
- Policy Evaluation and Policy Iteration

45

Utilities for Fixed Policies

- Another basic operation: compute the utility of a state s under a fixed (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):



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

46

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)

47

Policy Iteration

- Alternative approach:
 - Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
 - Step 2: Policy improvement:** 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**
 - It's still optimal!
 - Can converge faster under some conditions

48

Policy Iteration

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

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

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

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

51

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
 - Occasional passes to update policies
- 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

53

Asynchronous Value Iteration*

- In value iteration, we update every state in each iteration
- Actually, *any* sequences of Bellman updates will converge if every state is visited infinitely often
- In fact, we can update the policy as seldom or often as we like, and we will still converge
- Idea: Update states whose value we expect to change: If $|V_{i+1}(s) - V_i(s)|$ is large then update predecessors of s

MDPs recap

- Markov decision processes:
 - States S
 - Actions A
 - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
 - Rewards $R(s,a,s')$ (and discount γ)
 - Start state s_0
- Solution methods:
 - Value iteration (VI)
 - Policy iteration (PI)
 - Asynchronous value iteration
- Current limitations:
 - Relatively small state spaces
 - Assumes T and R are known

55