

# CS 188: Artificial Intelligence

## Spring 2010

### Lecture 9: MDPs

2/16/2010

Pieter Abbeel – UC Berkeley  
Many slides adapted from Dan Klein

## Announcements

---

- **Assignments**
  - P2 due Thursday
  - We reserved Soda 271 on Wednesday Feb 17 from 4 to 6. One of the GSI's will periodically drop in to see if he can provide any clarifications/assistance. It's a great opportunity to meet other students who might still be looking for a partner.
- **Readings:**
  - For MDPs / reinforcement learning, we're using an online reading
  - Different treatment and notation than the R&N book, beware!
  - Lecture version is the standard for this class

## Example: Insurance

- Consider the lottery [0.5,\$1000; 0.5,\$0]
  - What is its **expected monetary value (EMV)**? (\$500)
  - What is its **certainty equivalent**?
    - Monetary value acceptable in lieu of lottery
    - \$400 for most people
  - Difference of \$100 is the **insurance premium**
    - There's an insurance industry because people will pay to reduce their risk
    - If everyone were risk-neutral, no insurance needed!

3

## Example: Insurance

- Because people ascribe different utilities to different amounts of money, insurance agreements can increase both parties' expected utility

You own a car. Your lottery:

$L_Y = [0.8, \$0 ; 0.2, -\$200]$

i.e., 20% chance of crashing

You do not want -\$200!

$$U_Y(L_Y) = 0.2 \cdot U_Y(-\$200) = -200$$

$$U_Y(-\$50) = -150$$

Amount	Your Utility $U_Y$
\$0	0
-\$50	-150
-\$200	-1000

## Example: Insurance

---

- Because people ascribe different utilities to different amounts of money, insurance agreements can increase both parties' expected utility

You own a car. Your lottery:

$$L_Y = [0.8, \$0 ; 0.2, -\$200]$$

i.e., 20% chance of crashing

You do not want -\$200!

$$U_Y(L_Y) = 0.2 * U_Y(-\$200) = -200$$

$$U_Y(-\$50) = -150$$

Insurance company buys risk:

$$L_I = [0.8, \$50 ; 0.2, -\$150]$$

i.e., \$50 revenue + your  $L_Y$

Insurer is risk-neutral:

$$U(L) = U(EMV(L))$$

$$\begin{aligned} U_I(L_I) &= U(0.8 * 50 + 0.2 * (-150)) \\ &= U(\$10) > U(\$0) \end{aligned}$$

## Example: Human Rationality?

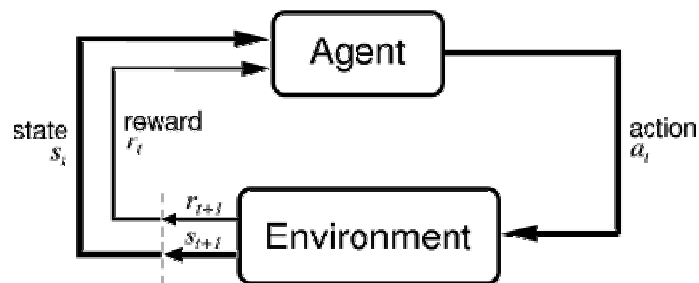
---

- Famous example of Allais (1953)
  - A: [0.8, \$4k; 0.2, \$0]
  - B: [1.0, \$3k; 0.0, \$0]
  
  - C: [0.2, \$4k; 0.8, \$0]
  - D: [0.25, \$3k; 0.75, \$0]
  
- Most people prefer  $B > A$ ,  $C > D$
- But if  $U(\$0) = 0$ , then
  - $B > A \Rightarrow U(\$3k) > 0.8 U(\$4k)$
  - $C > D \Rightarrow 0.2 U(\$4k) > 0.25 U(\$3k)$   
equivalently:  $0.8 U(\$4k) > U(\$3k)$

# Reinforcement Learning

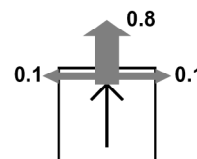
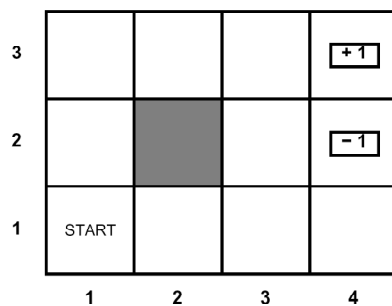
- Basic idea:

- 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**



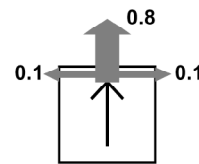
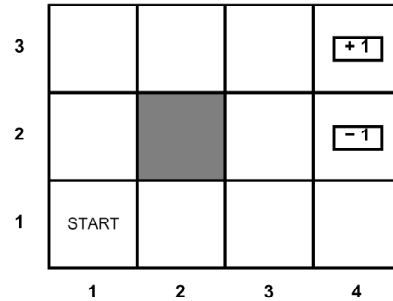
## Grid World

- 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



# 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



11

# 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:

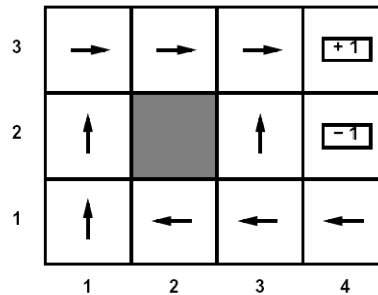


$$\begin{aligned}
 &P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots, S_0 = s_0) \\
 &= \\
 &P(S_{t+1} = s' | S_t = s_t, A_t = a_t)
 \end{aligned}$$

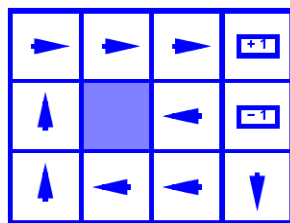
# Solving MDPs

- 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  $\pi$  gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

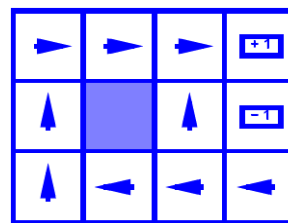
Optimal policy when  
 $R(s, a, s') = -0.03$  for all  
 non-terminals  $s$



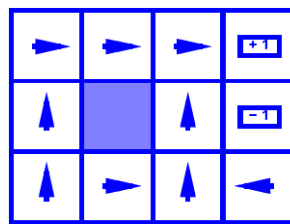
## Example Optimal Policies



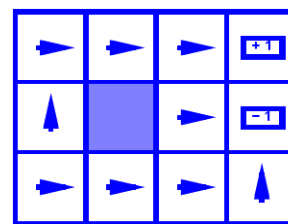
$R(s) = -0.01$



$R(s) = -0.03$



$R(s) = -0.4$

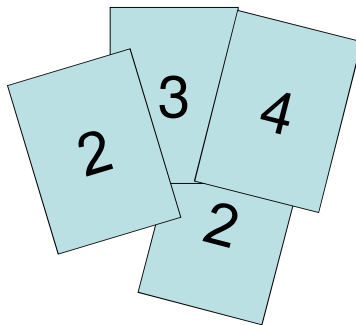


$R(s) = -2.0$

## 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!

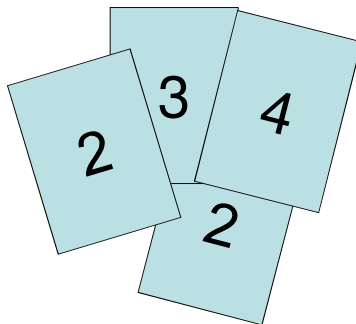


16

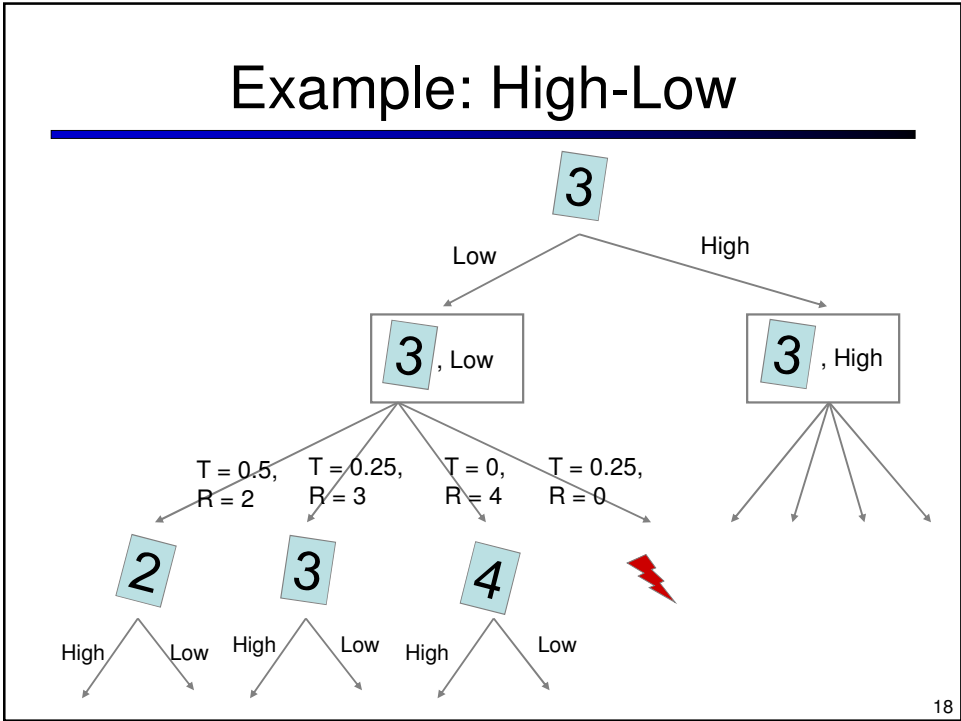
## High-Low as an MDP

---

- 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

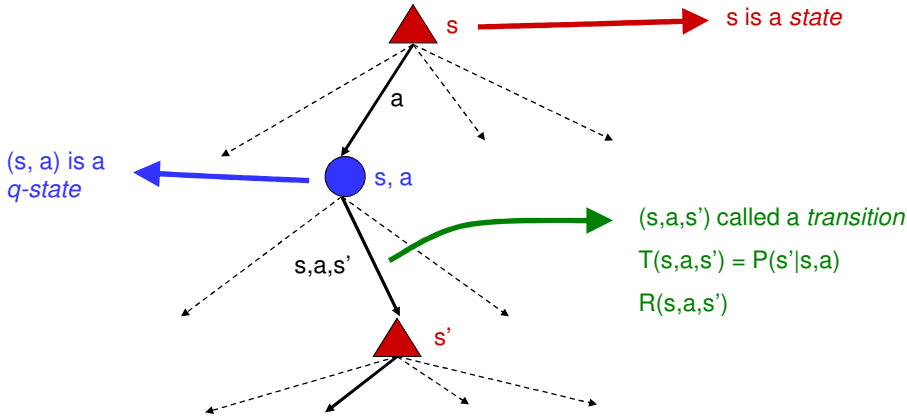


# Example: High-Low



# MDP Search Trees

- Each MDP state gives an expectimax-like search tree





# Utilities of Sequences

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

$$\begin{aligned}
 [r, r_0, r_1, r_2, \dots] &\succ [r', r'_0, r'_1, r'_2, \dots] \\
 &\Leftrightarrow \\
 [r_0, r_1, r_2, \dots] &\succ [r'_0, r'_1, r'_2, \dots]
 \end{aligned}$$

- 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:

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

20

# Infinite Utilities?!

- Problem:** infinite state sequences have infinite rewards

- Solutions:**

- Finite horizon:

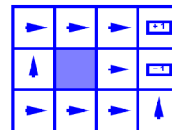
- Terminate episodes after a fixed T steps (e.g. life)
- Gives nonstationary policies ( $\pi$  depends on time left)

- Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like “done” for High-Low)

- Discounting: for  $0 < \gamma < 1$

$$U([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1 - \gamma)$$

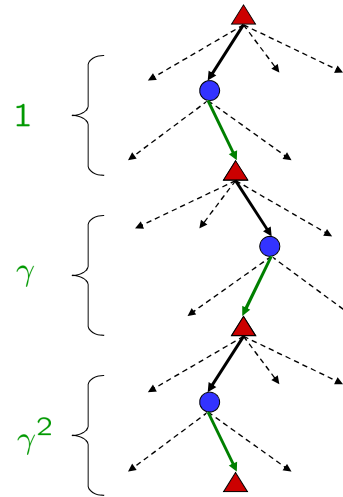
- Smaller  $\gamma$  means smaller “horizon” – shorter term focus



21

# Discounting

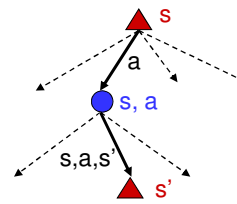
- Typically discount rewards by  $\gamma < 1$  each time step
  - Sooner rewards have higher utility than later rewards
  - Also helps the algorithms converge



22

# Recap: Defining MDPs

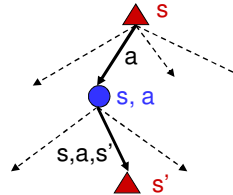
- Markov decision processes:
  - States  $S$
  - Start state  $s_0$
  - Actions  $A$
  - Transitions  $P(s'|s,a)$  (or  $T(s,a,s')$ )
  - Rewards  $R(s,a,s')$  (and discount  $\gamma$ )
- MDP quantities so far:
  - Policy = Choice of action for each state
  - Utility (or return) = sum of discounted rewards



23

# 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	0.812	0.868	0.912	+
2	0.762		0.660	-
1	0.705	0.655	0.611	0.388
	1	2	3	4

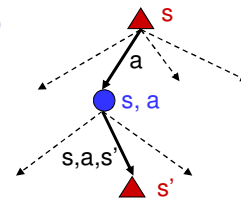
3	→	→	→	+
2	↑		↑	-
1	↑	←	←	←
	1	2	3	4

25

# 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:

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

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

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

26

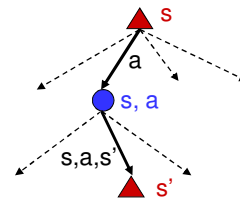
# Solving MDPs

- We want to find the **optimal policy**  $\pi^*$
- Proposal 1: modified expectimax search, starting from each state  $s$ :

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

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

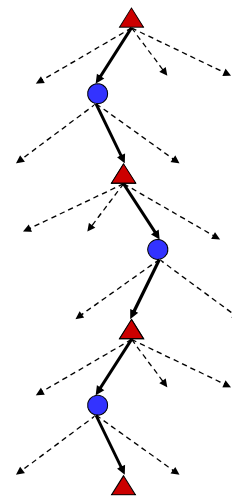
$$V^*(s) = \max_a Q^*(s, a)$$



27

# 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!



28

# Value Estimates

---

- Calculate estimates  $V_k^*(s)$ 
  - 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
  - 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

