

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Markov Decision Processes

Many slides courtesy of Dan Klein, Stuart Russell, or Andrew Moore

CS 5300 / CS 6300
Artificial Intelligence
Spring 2009

Hal Daumé III
hal@cs.utah.edu

www.cs.utah.edu/~hal/courses/2009S_AI

Slide 1 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Announcements

- Please fill out early course survey:
 - <http://www.cs.utah.edu/~hal/tmp/ai09-early.html>
 - (was emailed to mailing list)
- HW3 is due this coming Tuesday
- Project 2 is due the following Tuesday

Slide 2 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

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**
 - **Change the rewards, change the learned behavior**
- Examples:
 - Playing a game, reward at the end for winning / losing
 - Vacuuming a house, reward for each piece of dirt picked up
 - Automated taxi, reward for each passenger delivered

Slide 3 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Human Reinforcement Learning

Slide 4 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

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

Slide 5 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Map 0: Would you go across the top?

- Start in top-right, +\$1 for top left, -\$1 for red squares
- Costs N cents per step
- For what value N would you risk the "high road"?
- Write something between 1 cent and 12 cents

Slide 6 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Map 1: Would you go across the top?

- Start in top-right, +\$1 for top left, -\$1 for red squares
- Costs N cents per step
- For what value N would you risk the "high road"?
 - Write something between 1 cent and 12 cents

Slide 7 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Solving MDPs

- In deterministic single-agent search problem, want an optimal **plan**, or sequence of actions, from start to a goal
- In an MDP, we want an optimal **policy** $\pi(s)$
 - A policy gives an action for each state
 - Optimal policy maximizes expected if followed
 - Defines a reflex agent

Optimal policy when $R(s, a, s') = -0.04$ for all non-terminals s

Slide 8 10: MDPs

Hal Daumé III (hal@cs.utah.edu)

Example Optimal Policies

$R(s) = -0.01$ $R(s) = -0.03$

$R(s) = -0.4$ $R(s) = -2.0$

Slide 9 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

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!

Slide 10 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

High-Low

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

Note: could choose actions with search. How?

Slide 11 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Example: High-Low

Slide 12 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

MDP Search Trees

- Each MDP state gives an expectimax-like search tree

Slide 13 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Utilities of Sequences

- In order to formalize optimality of a policy, need to understand utilities of sequences of rewards
- Typically consider **stationary preferences**:
 - $[r_0, r_1, r_2, \dots] \succ [r'_0, r'_1, r'_2, \dots]$
 - $\Leftrightarrow [r_0, r_1, r_2, \dots] \succ [r'_0, r'_1, r'_2, \dots]$
- 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:

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

Assuming that reward depends only on state for these slides!

Slide 14 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Infinite Utilities?!

- Problem: infinite sequences with infinite rewards
- Solutions:
 - Finite horizon:
 - Terminate episodes after a fixed T steps
 - Gives nonstationary policy (π depends on time left)
 - Absorbing state(s): guarantee that for every policy, agent will eventually "die" (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

Slide 15 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Discounting

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

Slide 16 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Optimal Utilities

- Fundamental operation: compute the optimal utilities of states s (all at once)
- Why? Optimal values define optimal policies!
- Define the utility of a state s :

$$V(s) = \text{expected return starting in } s \text{ and acting optimally}$$
- Define the utility of a q-state (s, a) :

$$Q(s, a) = \text{expected return starting in } s, \text{ taking action } a \text{ and thereafter acting optimally}$$
- Define the optimal policy:

$$\pi(s) = \text{optimal action from state } s$$

3	0.912	0.863	0.912	☑
2	0.782		0.690	☑
1	0.705	0.605	0.611	0.358
	1	2	3	4

Slide 17 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

The Bellman Equations

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

$$\text{Optimal rewards} = \text{maximize over first action and then follow optimal policy}$$
- Formally:

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

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

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

Slide 18 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Solving MDPs

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

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

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

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

Slide 19 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

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!

Slide 20 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

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

Slide 21 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Memoized Recursion?

- Recurrences:

$$V_0^*(s) = 0$$

$$V_i^*(s) = \max_a Q_i^*(s, a)$$

$$Q_i^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_{i-1}^*(s')]$$

$$\pi_i(s) = \arg \max_a Q_i^*(s, a)$$
- Cache all function call results so you never repeat work
- What happened to the evaluation function?

Slide 22 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

Value Iteration

- Problems with the recursive computation:
 - Have to keep all the $V_k(s)$ around all the time
 - Don't know which depth $\pi_k(s)$ to ask for when planning
- Solution: value iteration
 - Calculate values for all states, bottom-up
 - Keep increasing k until convergence

Slide 23 CS 5300: MDPs

UNIVERSITY OF UTAH

Hal Daumé III (hal@cs.utah.edu)

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

Slide 24 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Example: Bellman Updates

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

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

$$= 0.9 [0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0]$$

Slide 25 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Example: Value Iteration

$$V_2$$

$$V_3$$

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

Slide 26 [DEMO] CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Convergence*

> Define the max-norm: $\|U\| = \max_s |U(s)|$

> Theorem: For any two approximations U and V

$$\|U^{t+1} - V^{t+1}\| \leq \gamma \|U^t - V^t\|$$

> 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:

$$\|U^{t+1} - U^t\| < \epsilon, \Rightarrow \|U^{t+1} - U\| < 2\epsilon\gamma / (1 - \gamma)$$

> I.e. once the change in our approximation is small, it must also be close to correct

Slide 27 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Practice: Computing Actions

> Which action should we chose from state s:

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

> Given optimal values V?

> Given optimal q-values Q?

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

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

Slide 28 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

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 :

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

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

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

Slide 30 CS 5300: MDPs

Hal Daumé III (hal@cs.utah.edu)

Policy Evaluation

> How do we calculate the V's for a fixed policy?

> Idea one: turn recursive equations into 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)

Slide 31 CS 5300: MDPs

Policy Iteration

- Alternative approach:
 - **Step 1: Policy evaluation:** calculate utilities for a fixed policy (not optimal utilities!) until convergence
 - **Step 2: Policy improvement:** update policy using one-step lookahead with resulting converged (but not optimal!) utilities
 - Repeat steps until policy converges
- This is **policy iteration**
 - It's still optimal!
 - Can converge faster under some conditions

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

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

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