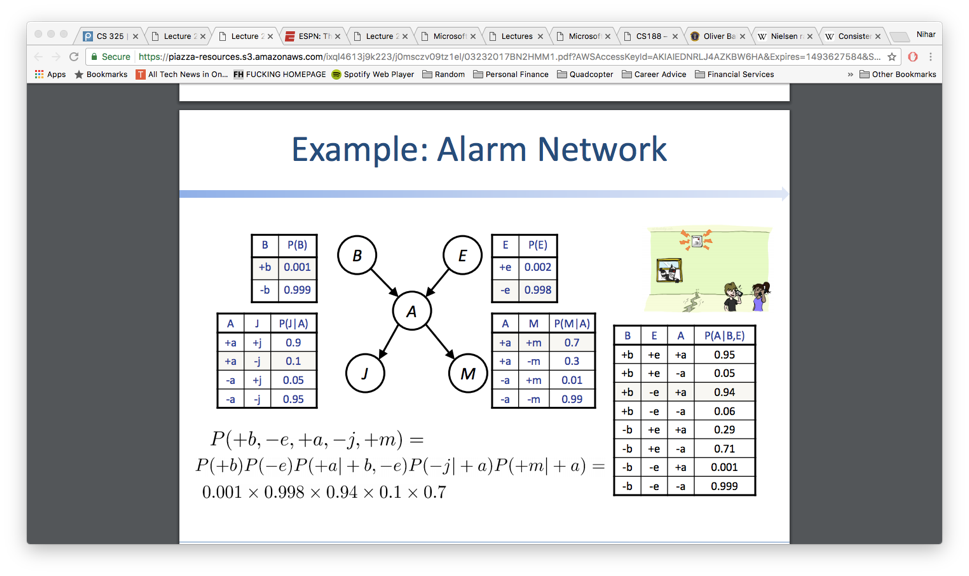
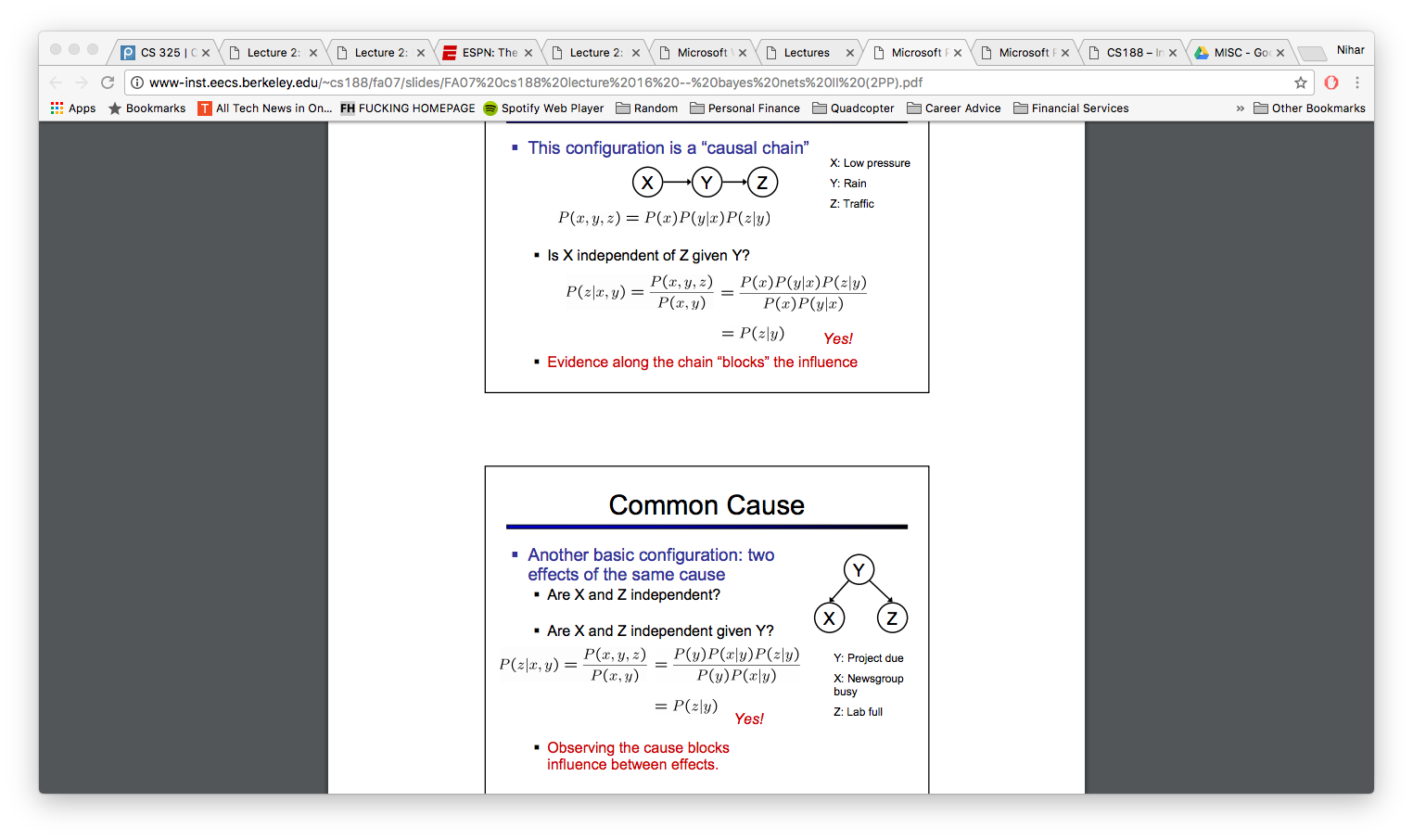
Bayes-net problems

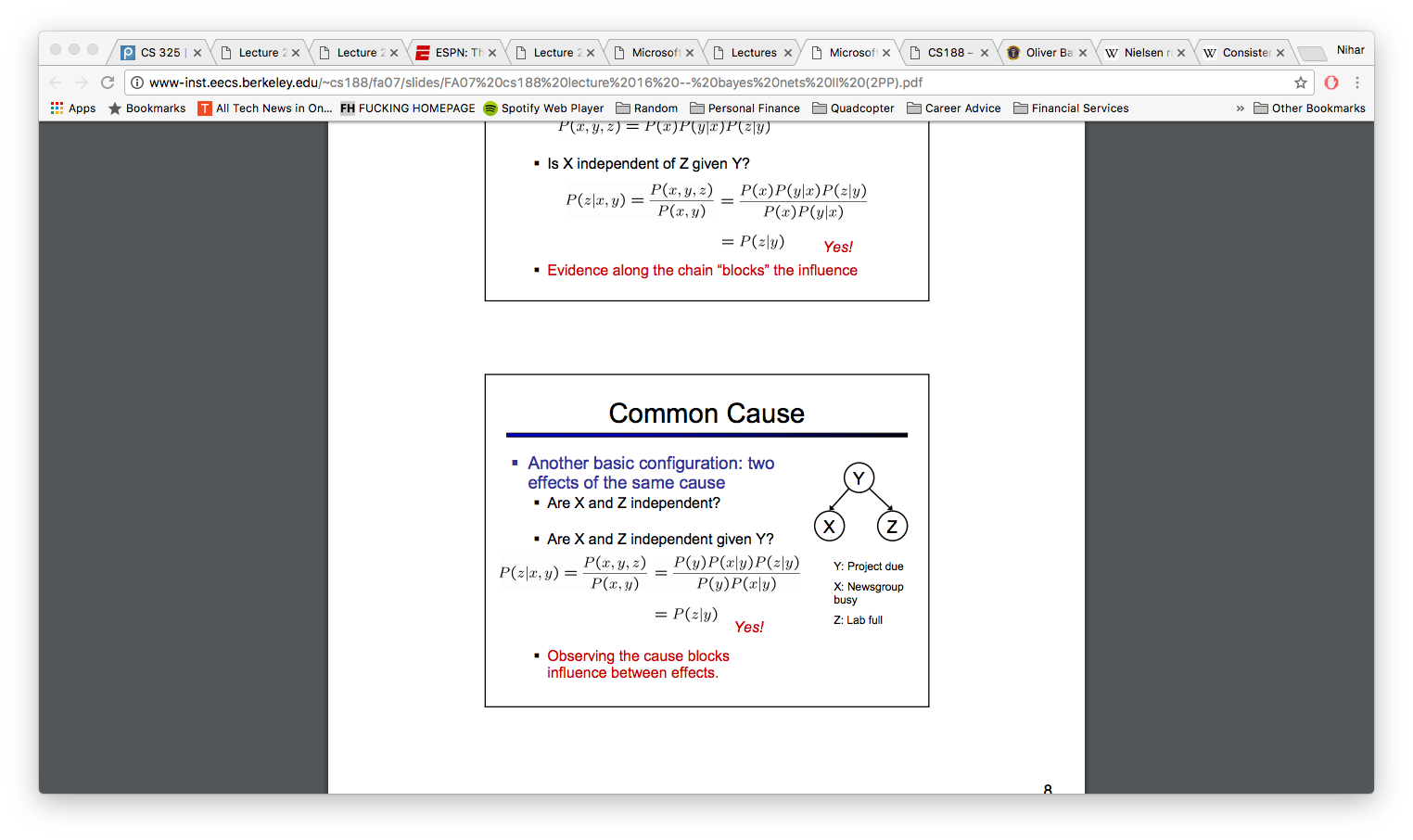
Joint distribution expansion:



P(j|a,b,e) = p(a,b,e,j) / p(a,b,e) = p(b)p(e)p(a|b,e)p(j|a) / p(b)p(e)p(a|b,e) = p(j|a)



Cat problem: Y 🡪 X & Z

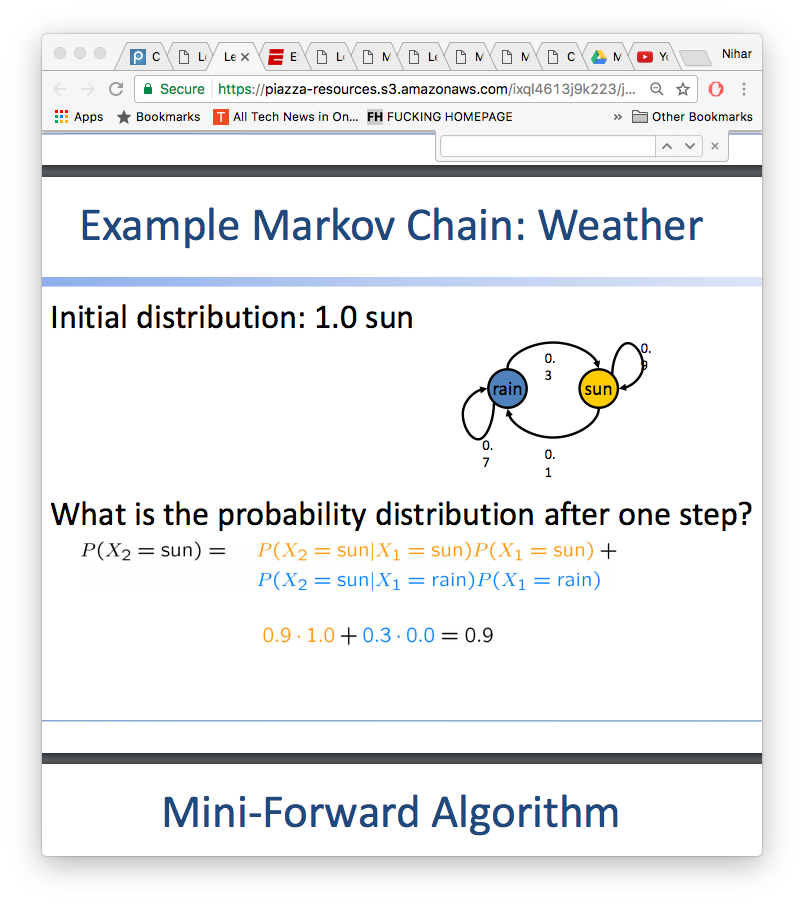


Variable Elimination:

1. Join factors – P(r,t) = P(r) \* P(t|r)
2. Sum r to get P(t)
3. Keep doing that, also only use observations that matter

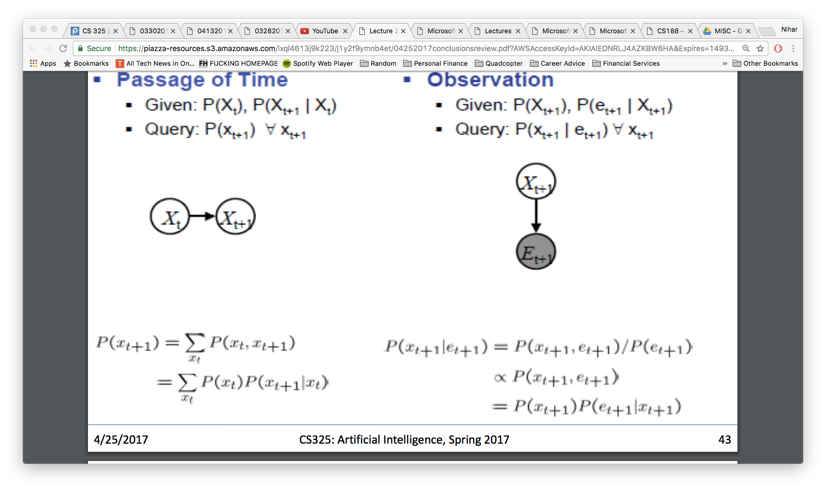
Markov Models

Have implied conditional independency



HMM

Update beliefs based on observations



Smoothing

In practice, Laplace often performs poorly for P(X|Y):

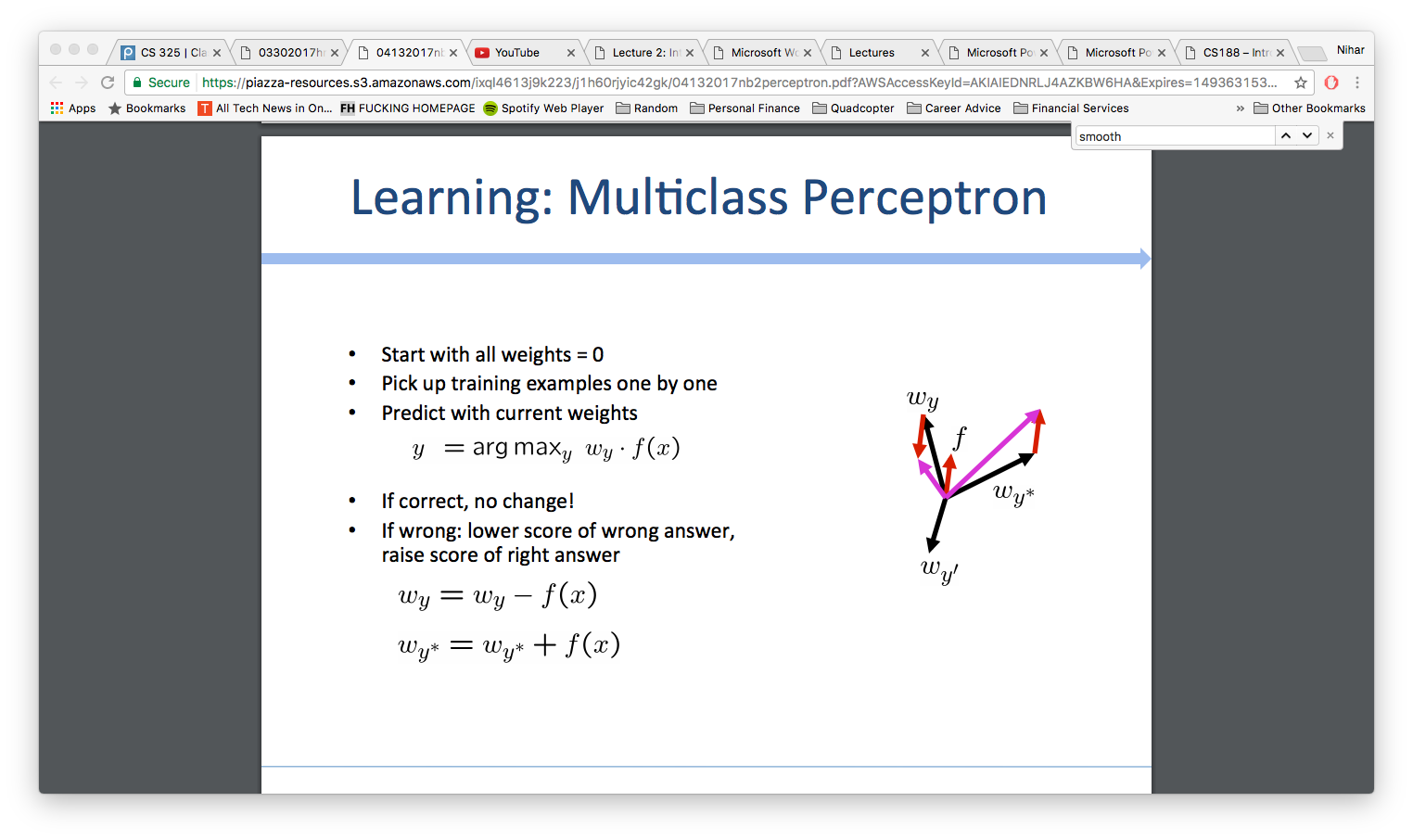
– When |X| is very large

– When |Y| is very large

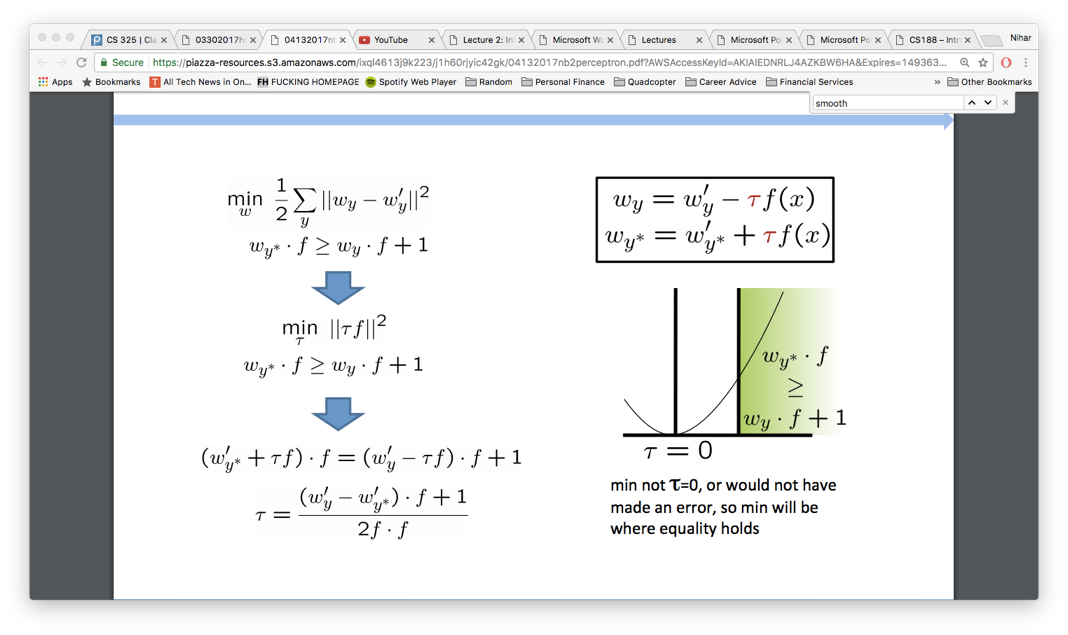
Particle filtering – less accurate but higher speed

Properties of Perceptrons

Separability: true if some parameters get the training set perfectly correct • Convergence: if the training is separable, perceptron will eventually converge (binary case) • Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability



MIRA



Search: DFS – O(b^m), O(bm) BFS – O(b^s), O(b^s) ID – O(b^d), O(bd)

UCS - O(b^C\*/ε ), O(b^C\*/ε), If that solution costs C\* and arcs cost at least ε, optimal

A\* - heuristic is admissible if its less than true cost to goal, A\* Review: f(n) = UCS + Heuristic

* As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself
* The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
* Max of admissible heuristics is admissible, Optimal if heuristic is admissible / consistent
* Consistent is subset of admissable
* O(b^d), O(b^d)
* Greedy prioritizes only heuristic, not true cost to goal

CSPs: Backtracking search is the basic uninformed algorithm for solving CSPs - Idea 1: One variable at a time Idea 2: Check constraints as you go

Filtering: Keep track of domains for unassigned variables and cross off bad options • Forward checking: Cross off values that violate a constraint when added to the existing assignment

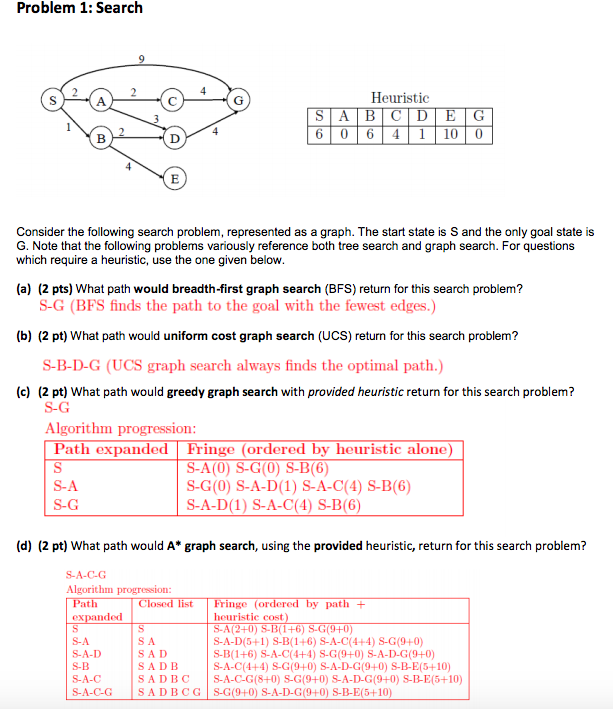
Arc: A simple form of propagation makes sure all arcs are consistent:

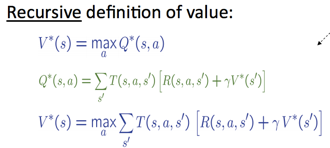
Variable Ordering: Minimum remaining values (MRV): – Choose the variable with the fewest legal left values in its domain

Value Ordering: Least Constraining Value – Given a choice of variable, choose the least constraining value – I.e., the one that rules out the fewest values in the remaining variables

How efficient is minimax? – Just like (exhaustive) DFS – Time: O(b^m) – Space: O(bm)

Pruning: With perfect ordering, time complexity is O(b^d/2) , space is O(bm)

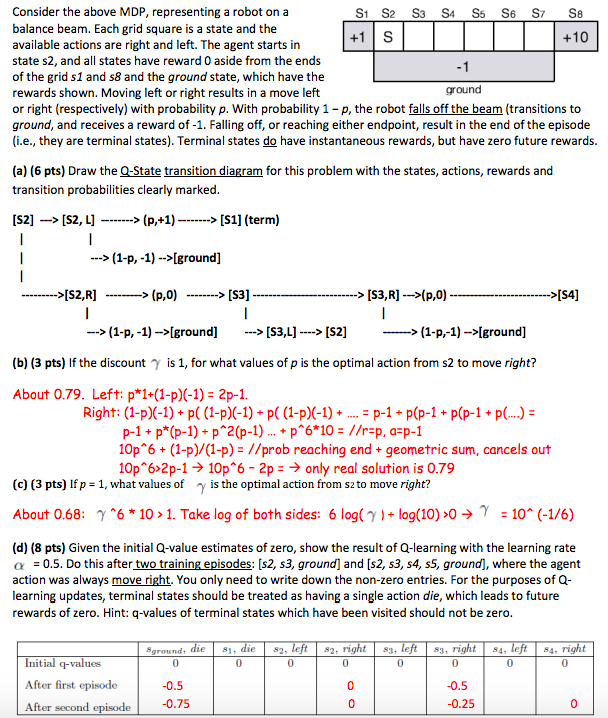
Alpha-beta pruning = go from left to right, if first leaf of branch is less than previous branch’s min, you can prune remaining leaves, as you are seeking the max in the next round



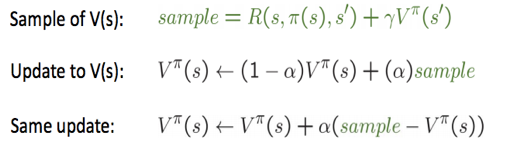
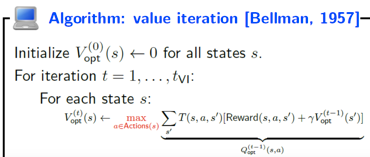
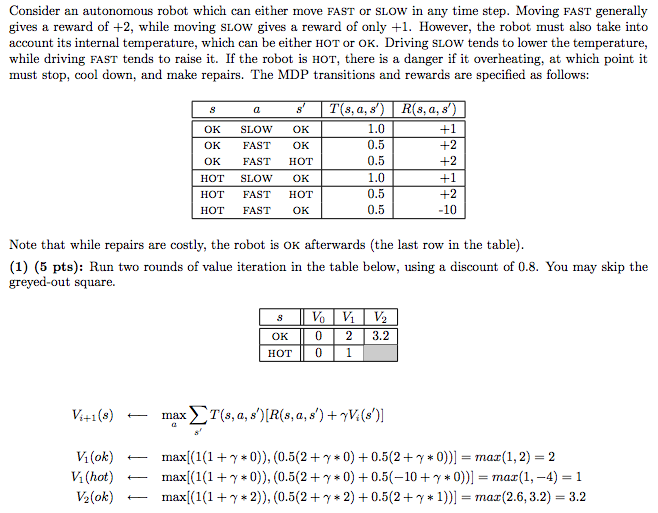
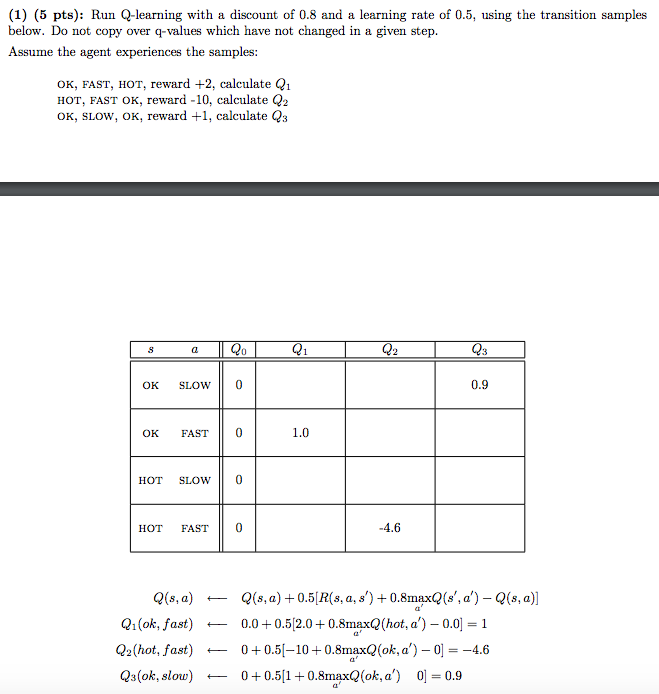
MDPs: For MDPs, we want an optimal policy π\*: S → A – A policy π gives an action for each state – An optimal policy is one that maximizes expected utility if followed – An explicit policy defines a reflex agent

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)

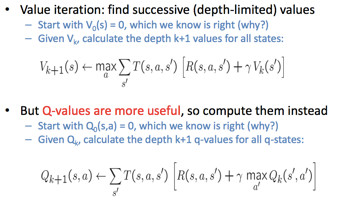
Value Iteration: O(A\*S^2) per iteration

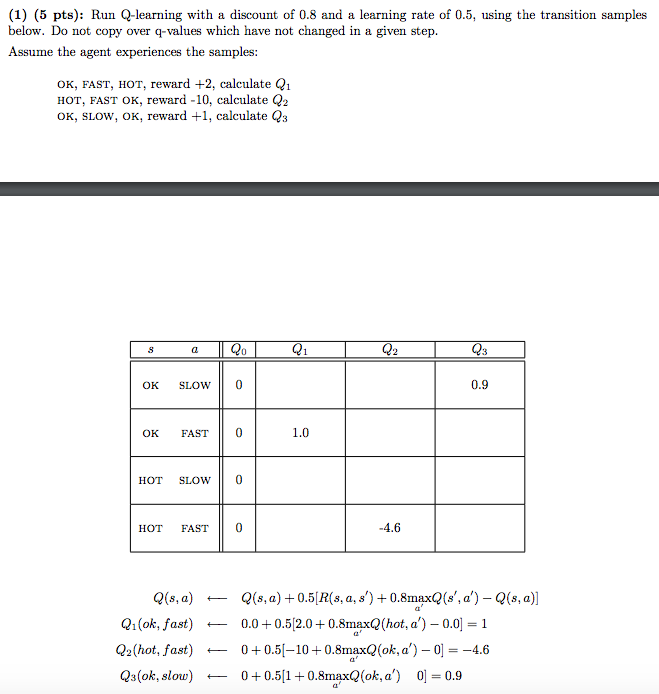
Both value interaction and policy interaction compute the same thing (all optimal values) • In value interaction: – Every interaction updates both the values and (implicitly) the policy – We don’t track the policy, but taking the max over actions implicitly recomputes it • In policy interaction: – We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them) – After the policy is evaluated, a new policy is chosen (slow like a value interaction pass) – The new policy will be better (or we’re done)

So you want to…. – Compute optimal values: use value iteration or policy iteration – Compute values for a par.cular policy: use policy evalua.on – Turn your values into a policy: use policy extraction (one-step lookahead) • These all look the same! – They basically are – they are all varia.ons of Bellman updates – They all use one-step lookahead expectimax fragments – They differ only in whether we plug in a fixed policy or max over ac.ons

Convergence: Case 1: If the tree has maximum depth M, then VM holds the actual untruncated values • Case 2: If the discount is less than 1 – Sketch: For any state Vk and Vk+1 can be viewed as depth k+1 expectimax results in nearly identical search trees – The difference is that on the bottom layer, Vk+1 has actual rewards while Vk has zeros – That last layer is at best all RMAX – It is at worst RMIN – But everything is discounted by γk that far out – So Vk and Vk+1 are at most γk max|R| different – So as k increases, the values converge

Temporal: Temporal difference learning of values – Policy still fixed, still doing evaluation! – Move values toward value of whatever successor occurs: running average

Better is q-value as gives policy



a. For what values of p is the optimal action from s2 to move right if the discount γ is 1? E[left] = (+1)(p) +(-1)(1-p) = 2p-1 E[always go right] >= 10p^6 – 1(1-p^6) > 2p-1 => (11/2)p^5 > 1 => p > 0.71

b. For what values of γ is the optimal action from s2 to move right if p = 1? 10 \* γ^6 > γ => γ >=0.63

