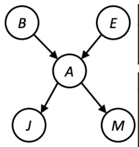
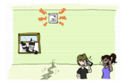
Bayes-net problems
Joint distribution expansion:

B	P(B)
+b	0.001
-b	0.999

E	P(E)
+e	0.002
-e	0.998





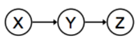
A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

$$P(+b,-e,+a,-j,+m) =$$
$$P(+b)P(-e)P(+a|+b,-e)P(-j|+a)P(+m|+a) =$$
$$0.001 \times 0.998 \times 0.94 \times 0.1 \times 0.7$$

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)



X: Low pr
Y: Rain
Z: Traffic

$$P(x,y,z) = P(x)P(y|x)P(z|y)$$

- Is X independent of Z given Y?

$$P(z|x,y) = \frac{P(x,y,z)}{P(x,y)} = \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)}$$

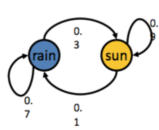
Cat problem: Y → X & Z

$$P(z|x,y) = \frac{P(x,y,z)}{P(x,y)} = \frac{P(y)P(x|y)P(z|y)}{P(y)P(x|y)}$$

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

Initial distribution: 1.0 sun

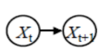


What is the probability distribution after one step?

$$P(X_2 = \text{sun}) = P(X_2 = \text{sun}|X_1 = \text{sun})P(X_1 = \text{sun}) + P(X_2 = \text{sun}|X_1 = \text{rain})P(X_1 = \text{rain})$$

$$0.9 \cdot 1.0 + 0.3 \cdot 0.0 = 0.9$$

HMM
Update beliefs based on observations



$$P(x_{t+1}) = \sum_{x_t} P(x_t, x_{t+1}) = \sum_{x_t} P(x_t)P(x_{t+1}|x_t)$$

$$P(x_{t+1}|e_{t+1}) = P(x_{t+1}, e_{t+1})/P(e_{t+1}) \propto P(x_{t+1}, e_{t+1}) = P(x_{t+1})P(e_{t+1}|x_{t+1})$$

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

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$
$$w_{y^*} = w_{y^*} + f(x)$$

MIRA

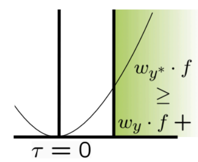
$$\min_w \frac{1}{2}\sum_y ||w_y - w'_y||^2$$
$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$\min_\tau ||\tau f||^2$$
$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$(w'_{y^*} + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1$$

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2 f \cdot f}$$

$$\boxed{\begin{aligned} w_y &= w'_y - \tau f(x) \\ w_{y^*} &= w'_{y^*} + \tau f(x) \end{aligned}}$$



min not τ=0, or would not have made an error, so min will be where equality holds

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 $\pi^*: S \rightarrow A$ – 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

Recursive definition of value:

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

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

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

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

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha) sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Algorithm: value iteration [Bellman, 1957]

Initialize $V^{(0)}_{\text{opt}}(s) \leftarrow 0$ for all states s.

For iteration $t = 1, \ldots, t_{\text{VI}}$:

For each state s:

$$V^{(t)}_{\text{opt}}(s) \leftarrow \max_{a \in \text{Actions}(s)} \underbrace{\sum_{s'} T(s,a,s')[\text{Reward}(s,a,s') + \gamma V^{(t-1)}_{\text{opt}}(s')]}_{Q^{(t-1)}_{\text{opt}}(s,a)}$$

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

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right (why?)
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right (why?)
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

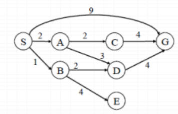
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

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

Problem 1: Search



Heuristic					
S	A	B	C	D	E
6	0	6	4	1	10

Consider the following search problem, represented as a graph. The start state is S ar G. Note that the following problems variously reference both tree search and graph se which require a heuristic, use the one given below.

(a) (2 pts) What path **would breadth-first graph search** (BFS) return for this search

S-G (BFS finds the path to the goal with the fewest edges.)

(b) (2 pt) What path would **uniform cost graph search** (UCS) return for this search p

S-B-D-G (UCS graph search always finds the optimal path.)

(c) (2 pt) What path would **greedy graph search** with *provided heuristic* return for th

S-G

Algorithm progression:

Path expanded	Fringe (ordered by heuristic alone)
S	S-A(0) S-G(0) S-B(6)
S-A	S-G(0) S-A-D(1) S-A-C(4) S-B(6)
S-G	S-A-D(1) S-A-C(4) S-B(6)

(d) (2 pt) What path would **A* graph search**, using the **provided** heuristic, return for

S-A-C-G

Algorithm progression:

Path expanded	Closed list	Fringe (ordered by path + heuristic cost)
S	S	S-A(2+0) S-B(1+6) S-G(9+0)
S-A	S A	S-A-D(5+1) S-B(1+6) S-A-C(4+4) S-G(9+0)
S-A-D	S A D	S-B(1+6) S-A-C(4+4) S-G(9+0) S-A-D-G(9+0)
S-B	S A D B	S-A-C(4+4) S-G(9+0) S-A-D-G(9+0) S-B-E(5+10)
S-A-C	S A D B C	S-A-C-G(8+0) S-G(9+0) S-A-D-G(9+0) S-B-E(5+10)
S-A-C-G	S A D B C G	S-G(9+0) S-A-D-G(9+0) S-B-E(5+10)

(a) **True False** An optimal solution path for a search problem with positive costs will never have repeated states.

True: will not expand closed (explored) states

(b) **True False** If two search heuristics h1(s) and h2(s) have the same average value, the heuristic h3(s) = max(h1(s), h2(s)) **could** give better A* efficiency than h1 or h2.

True: e.g., if h1 works better in the beginning, h2 works better in later stages

(c) **True False** If one search heuristic h1(s) is admissible, and another h2(s) is inadmissible, then a new heuristic h3(s) = min(h1(s), h2(s)) will be admissible.

True: by definition of admissibility

(d) **True False** In A* search, the first path to the goal which is added to the fringe will always be optimal.

False: the first goal removed from the fringe is optimal

(e) **True False** The minimax value of a state is always greater than or equal to the expectimax value of tha state.

False: the opposite: ExpectiMax chance nodes always predict at least as high (and usually higher) than Min nodes.

(f) **True False** Alpha-beta pruning can change the final minimax value of the root of a game search tree.

False: by construction, same result as MiniMax, but fewer nodes expanded

(g) **True False** When doing alpha-beta pruning on a game tree which is traversed from left to right, the leftmost branch can be pruned.

False: no previously seen Alpha, Beta values to prune against

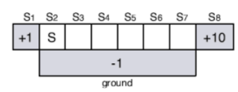
(h) **True False** Iterative deepening is a useful alternative to breadth first search because it uses less computation than BFS

False: uses more computation, but less space than BFS (only stores stack)

(i) **True False** In reinforcement learning, it is useful to sometimes act in a way which is believed to be suboptimal

True: need to explore (e.g., random action) occasionally to find higher reward states or can get trapped in local maximum

Consider the above MDP, representing a robot on a balance beam. Each grid square is a state and the available actions are right and left. The agent starts in state s2, and all states have reward 0 aside from the ends of the grid *s1* and *s8* and the *ground* state, which have the rewards shown. Moving left or right results in a move left or right (respectively) with probability *p*. With probability 1 – *p*, the robot falls off the beam (transitions to *ground*, and receives a reward of -1. Falling off, or reaching either endpoint, result in the end of the episode (i.e., they are terminal states). Terminal states do have instantaneous rewards, but have zero future rewards.



(a) (6 pts) Draw the Q-State transition diagram for this problem with the states, actions, rewards and transition probabilities clearly marked.

```
[S2] ---> [S2, L] --------> (p,+1) --------> [S1] (term)
 |            |
 |            ---> (1-p, -1) -->[ground]
 |
 --------->[S2,R]  ---------> (p,0)  --------> [S3] ------------------------> [S3,R] --->(p,0) ------------------------>[S4]
              |                                  |                              |
              ---> (1-p, -1) -->[ground]         ---> [S3,L] ----> [S2]         --------> (1-p,-1) -->[ground]
```

Consider an autonomous robot which can either move FAST or SLOW in any time step. Moving FAST generally gives a reward of +2, while moving SLOW gives a reward of only +1. However, the robot must also take into account its internal temperature, which can be either HOT or OK. Driving SLOW tends to lower the temperature, while driving FAST tends to raise it. If the robot is HOT, there is a danger if it overheating, at which point it must stop, cool down, and make repairs. The MDP transitions and rewards are specified as follows:

s	a	s'	$T(s,a,s')$	$R(s,a,s')$
OK	SLOW	OK	1.0	+1
OK	FAST	OK	0.5	+2
OK	FAST	HOT	0.5	+2
HOT	SLOW	OK	1.0	+1
HOT	FAST	HOT	0.5	+2
HOT	FAST	OK	0.5	-10

Note that while repairs are costly, the robot is OK afterwards (the last row in the table).

(1) (5 pts): Run two rounds of value iteration in the table below, using a discount of 0.8. You may skip the greyed-out square.

s	V_0	V_1	V_2
OK	0	2	3.2
HOT	0	1	

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

$$V_1(ok) \longleftarrow \max[(1(1+\gamma*0)), (0.5(2+\gamma*0) + 0.5(2+\gamma*0))] = max(1,2) = 2$$

$$V_1(hot) \longleftarrow \max[(1(1+\gamma*0)), (0.5(2+\gamma*0) + 0.5(-10+\gamma*0))] = max(1,-4) = 1$$

$$V_2(ok) \longleftarrow \max[(1(1+\gamma*2)), (0.5(2+\gamma*2) + 0.5(2+\gamma*1))] = max(2.6, 3.2) = 3.2$$

(1) (5 pts): Run Q-learning with a discount of 0.8 and a learning rate of 0.5, using the transition samples below. Do not copy over q-values which have not changed in a given step.

Assume the agent experiences the samples:

OK, FAST, HOT, reward +2, calculate Q_1
HOT, FAST OK, reward -10, calculate Q_2
OK, SLOW, OK, reward +1, calculate Q_3

s	a	Q_0	Q_1	Q_2	Q_3
OK	SLOW	0			0.9
OK	FAST	0	1.0		
HOT	SLOW	0			
HOT	FAST	0		-4.6	

$$Q(s,a) \longleftarrow Q(s,a) + 0.5[R(s,a,s') + 0.8\max_{a'} Q(s',a') - Q(s,a)]$$

$$Q_1(ok, fast) \longleftarrow 0.0 + 0.5[2.0 + 0.8\max_{a'} Q(hot,a') - 0.0] = 1$$

$$Q_2(hot, fast) \longleftarrow 0 + 0.5[-10 + 0.8\max_{a'} Q(ok,a') - 0] = -4.6$$

$$Q_3(ok, slow) \longleftarrow 0 + 0.5[1 + 0.8\max_{a'} Q(ok,a') \quad 0] = 0.9$$