CS 188 Spring 2015 — Introduction to Artificial Intelligence

Final V1

- You have approximately 2 hours and 50 minutes.
- The exam is closed book, closed calculator, and closed notes except your three crib sheets.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation or show your work.
- For multiple choice questions,
  - ☐ means mark **all options** that apply
  - ○ means mark a **single choice**
- There are multiple versions of the exam. For fairness, this does not impact the questions asked, only the ordering of options within a given question.

First name	
Last name	
SID	
edX username	

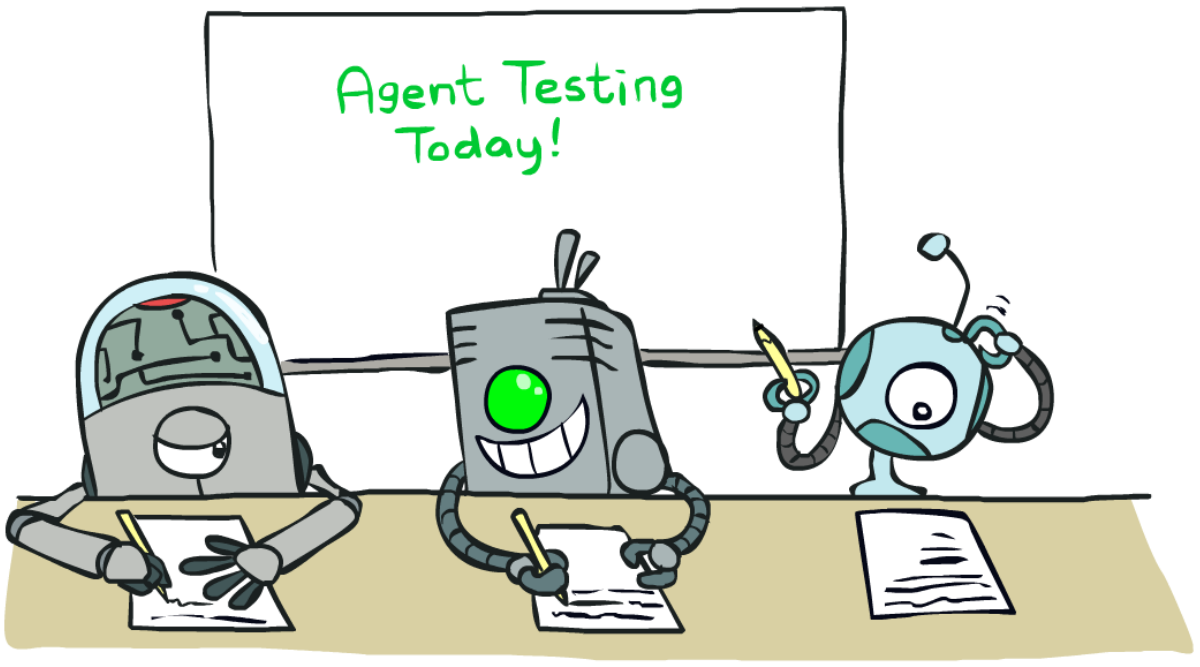
First and last name of student to your left	
First and last name of student to your right	

For staff use only:

Q1.	Agent Testing Today!	/1
Q2.	Power Pellets	/6
Q3.	Satisfying Search	/15
Q4.	Worst-Case Backtracking	/6
Q5.	Best-Case Pruning	/7
Q6.	Ghostbusters	/10
Q7.	MDPs	/6
Q8.	RL	/8
Q9.	Bayes Net and Decision Networks	/8
Q10.	DNA Sequencing	/7
Q11.	Dynamic Bayes' Nets	/11
Q12.	Decision Trees and Other Classifiers	/15
	Total	/100

Q1. [1 pt] Agent Testing Today!

It's testing time! Not only for you, but for our CS188 robots as well! Circle your favorite robot below.



Any answer was acceptable.

Q2. [6 pts] Power Pellets

Consider a Pacman game where Pacman can eat 3 types of pellets:

- Normal pellets (n-pellets), which are worth one point.
- Decaying pellets (d-pellets), which are worth $max(0, 5 - t)$ points, where t is time.
- Growing pellets (g-pellets), which are worth t points, where t is time.

The pellet's point value stops changing once eaten. For example, if Pacman eats one g-pellet at $t = 1$ and one d-pellet at $t = 2$, Pacman will have won $1 + 3 = 4$ points.

Pacman needs to find a path to win at least 10 points but he wants to minimize distance travelled. The cost between states is equal to distance travelled.

(a) [2 pts] Which of the following must be including for a minimum, sufficient state space?

☐ Location and type of each pellet
[✓ Total points Pacman has won]
☐ How far Pacman has travelled
[✓ Current time]
☐ How many pellets Pacman has eaten and the point value of each eaten pellet
[✓ Pacman's location]
[✓ Which pellets Pacman has eaten]

A state space should include which pellets are left on the board, the current value of pellets, Pacman's location, and the total points collected so far. With this in mind:
(1) The starting location and type of each pellet are not included in the state space as this is something that does not change during the search. This is analogous to how the walls of a Pacman board are not included in the state space.
(2) How far Pacman has travelled does not need to be explicitly tracked by the state, since this will be reflected in the cost of a path.
(3) Pacman does need the current time to determine the value of pellets on the board.
(4) The number of pellets Pacman has eaten is extraneous.
(5) Pacman must track the total number of points won for the goal test.
(6) Pacman must know which pellets remain on the board, which is the complement of the pellets he has eaten.

(b) [2 pts] Which of the following are admissible heuristics? Let x be the number of points won so far.

[✓ Distance to closest pellet, except if in the goal state, in which case the heuristic value is 0.]
☐ Distance needed to win $10 - x$ points, determining the value of all pellets as if they were n-pellets.
⊛ Distance needed to win $10 - x$ points, determining the value of all pellets as if they were g-pellets (i.e. all pellet values will be t.)
☐ Distance needed to win $10 - x$ points, determining the value of all pellets as if they were d-pellets (i.e. all pellet values will be $max(0, 5 - t)$.
☐ Distance needed to win $10 - x$ points assuming all pellets maintain current point value (g-pellets stop increasing in value and d-pellets stop decreasing in value)
☐ None of the above

(1) Admissible; to get 10 points Pacman will always have to travel at least as far as the distance to the closest pellet, so this will always be an underestimate.
(2) Not admissible; if all the pellets are actually g-pellets, assuming they are n-pellets will lead to Pacman collecting more pellets in more locations, and thus travel further.
(3) Ambiguous; if pellets are n-pellets or d-pellets, Pacman will generally have to go further, except at the beginning of the game when d-pellets are worth more, in which case this heuristic will over-estimate the cost to the goal. However, if Pacman is allowed to stay in place with no cost, then this heuristic is admissable because the heuristic

will instead calculate all pellet values as 10. This option was ignored in scoring.
(4) Not admissible; if pellets are n-pellets or g-pellets, Pacman would have an overestimate.
(5) Not admissible; if pellets are g-pellets, then using the current pellet value might lead Pacman to collect more locations, and thus travel further than necesarry.

(c) [2 pts] Instead of finding a path which minimizes distance, Pacman would like to find a path which minimizes the following:

$$C_{new} = a * t + b * d$$

where t is the amount of time elapsed, d is the distance travelled, and a and b are non-negative constants such that $a + b = 1$. Pacman knows an admissible heuristic when he is trying to minimize time (i.e. when $a = 1, b = 0$), h_t, and when he is trying to minimize distance, h_d (i.e. when $a = 0, b = 1$).
Which of the following heuristics is guaranteed to be admissible when minimizing C_{new}?

☐ $max(h_t, h_d)$ [✓ $min(h_t, h_d)$] ☐ $mean(h_t, h_d)$ [✓ $a * h_t + b * h_d$]
☐ None of the above

For this question, think about the inequality $C_{new} = a*t+b*d \geq a*h_t+b*h_d$. We can guarantee a heuristic h_{new} is admissible if $h_{new} \leq a * h_t + b * h_d$
(1) If $a = b$, $0.5 * h_t + 0.5 * h_d$ is not guaranteed to be less than $a * h_t + b * h_d$, so this will not be admissible.
(2) $min(h_t, h_d) = a * min(h_t, h_d) + b * min(h_t, h_d) \leq a * h_t + b * h_d$
(3) $max(h_t, h_d)$ will be greater than $a * h_t + b * h_d$ unless $h_t = h_d$, wo this will not be admissible.
(4) Admissible.

Q3. [15 pts] Satisfying Search

Consider a search problem $(S, A, Succ, s_0, G)$, where all actions have cost 1. S is the set of states, $A(s)$ is the set of legal actions from a state s, $Succ(s, a)$ is the state reached after taking action a in state s, s_0 is the start state, and $G(s)$ is true if and only if s is a goal state.

Suppose we have a search problem where we know that the solution cost is exactly k, but we do not know the actual solution. The search problems has $|S|$ states and a branching factor of b.

(a) **(i)** [1 pt] Since the costs are all 1, we decide to run breadth-first tree search. Give the tightest bound on the worst-case running time of breadth-first tree search in terms of $|S|$, b, and k.

The running time is $O($ b^k $)$ This is the normal running time for BFS.

(ii) [1 pt] Unfortunately, we get an out of memory error when we try to use breadth first search. Which of the following algorithms is the best one to use instead?

○ Depth First Search

[✓ Depth First Search limited to depth k]

○ Iterative Deepening

○ Uniform Cost Search

Firstly, notice that depth first search limited to depth k will find the solution, since we know that the solution is k moves long. Depth First search would explore paths of length larger than k, which is useless computation. Iterative Deepening would first explore paths of length 1, then 2, and so on, which is useless computation. Uniform Cost Search is equivalent to BFS when the costs are 1, and so will probably also have an out of memory error.

Instead of running a search algorithm to find the solution, we can phrase this as a CSP:

Variables: $X_0, X_1, X_2, \cdots X_k$

Domain of each variable: S, the set of all possible states

Constraints:

1. X_0 is the start state, that is, $X_0 = s_0$.
2. X_k must be a goal state, that is, $G(X_k)$ has to be true.
3. For every $0 \le i < k$, (X_i, X_{i+1}) is an edge in the search graph, that is, there exists an action $a \in A(X_i)$ such that $X_{i+1} = Succ(X_i, a)$.

With these constraints, when we get a solution $(X_0 = s_0, X_1 = s_1, \cdots X_k = s_k)$, the solution to our original search problem is the path $s_0 \to s_1 \to \cdots \to s_k$.

(b) [2 pts] This is a tree-structured CSP. Illustrate this by drawing the constraint graph for $k = 3$ and providing a linearization order. (For $k = 3$, the states should be named X_0, X_1, X_2, and X_3.)

$$\begin{tikzcd} X_0 \arrow[r, dash] & X_1 \arrow[r, dash] & X_2 \arrow[r, dash] & X_3 \end{tikzcd}$$

Linearization Order: X_0, X_1, X_2, X_3 or X_3, X_2, X_1, X_0, although others are possible.

(c) We can solve this CSP using the tree-structured CSP algorithm. You can make the following assumptions:

1. For any state s, computing $G(s)$ takes $O(1)$ time.
2. Checking consistency of *a single arc* $F \to G$ takes $O(fg)$ time, where f is the number of remaining values that F can take on and g is the number of remaining values that G can take on.

Remember that the search problem has a solution cost of exactly k, $|S|$ states, and a branching factor of b.

(i) [1 pt] Give the tightest bound on the time taken to enforce unary constraints, in terms of $|S|$, b, and k.

The running time to enforce unary constraints is $O($ ___$|S|$___ $)$

For the first constraint $X_0 = s_0$, we just need to change the domain of X_0 to $\{s_0\}$ For the constraint that X_k is a goal state, we need to compute $G(s)$ for all states s, which takes $O(|S|)$ time.

(ii) [1 pt] Give the tightest bound on the time taken to run the backward pass, in terms of $|S|$, b, and k.

The running time for the backward pass is $O($ ___$k|S|^2$___ $)$

The backward pass simply enforces the consistency of k arcs, each of which takes $O(|S|^2)$ time, for a total of $O(k|S|^2)$ time.

(iii) [1 pt] Give the tightest bound on the time taken to run the forward pass, in terms of $|S|$, b, and k.

The running time for the forward pass is $O($ ___$k|S|$___ $)$

The forward pass assigns values to variables in order, and then enforces consistency of an arc so that the values for the next variable are all legal. When enforcing the consistency of $X_{i+1} \to X_i$, X_i has already been assigned, so the time taken is $O(|S| \cdot 1) = O(|S|)$. This is done for each of the k arcs, for a total of $O(k|S|)$ time.

We would also accept $O(kb)$, on the basis that after assigning a variable to X_i, you only need to restrict X_{i+1} to the b possible values that follow X_i.

(d) [2 pts] Suppose $s_0 \to s_1 \to \cdots \to s_k$ is a solution to the search problem. Mark all of the following options that are *guaranteed* to be true after enforcing unary constraints and running arc consistency.

- ☐ The remaining values of X_i will be s_i and nothing else.
- ■ The remaining values of X_i will be s_i and possibly other values.
- ☐ A solution can be found by setting each X_i to any of the remaining states in its domain.
- ■ A solution can be found by executing the forward pass of the tree-structured CSP algorithm.
- ☐ None of the above

After enforcing unary constraints and running arc consistency, since this is a tree-structured CSP, we are guaranteed that all remaining values are part of *some* solution, but not necessarily *all* solutions. In addition, since arc consistency only eliminates impossible values, all the s_i values will still be present (since they are part of the solution $s_0 \to s_1 \to \cdots \to s_k$).

Thus, X_i will have s_i and other values (corresponding to other solutions). However, we cannot set each X_i to any of the remaining states in its domain, if we set X_1 to a state s' and X_2 to s'', while we know there is some solution where $X_1 = s'$, and some (possibly different) solution where $X_2 = s''$, we are not guaranteed that there is a solution where $X_1 = s'$ *and* $X_2 = s''$.

The backward pass of the tree-structured CSP algorithm simply enforces consistency of some arcs. So, running full arc consistency will eliminate at least all the values that the backward pass would have eliminated. So, we can run the forward pass to find a solution.

(e) [4 pts] Suppose you have a heuristic $h(s)$. You decide to add more constraints to your CSP (with the hope that it speeds up the solver by eliminating many states quickly). Mark all of the following options that are valid constraints that can be added to the CSP, under the assumption that $h(s)$ is (a) any function (b) admissible and (c) consistent. *Recall that the cost of every action is* 1.

	Any $h(s)$	$h(s)$ is admissible	$h(s)$ is consistent
For every $0 \leq i \leq k$, $h(X_i) \leq i$	☐	☐	☐
For every $0 \leq i \leq k$, $h(X_i) \leq k - i$	☐	🟥	🟥
For every $0 \leq i < k$, $h(X_{i+1}) \leq h(X_i) - 1$	☐	☐	☐
For every $0 \leq i < k$, $h(X_{i+1}) \geq h(X_i) - 1$	☐	☐	🟥

☐ None of the above

If we know nothing about the heuristic function $h(s)$, none of the constraints are valid. (Indeed, it would be very strange if we were able to write down constraints based on a function that could be anything.)

An admissible heuristic means that the value of the heuristic is an underestimate of the true cost to the goal. At variable X_i, we know that the cost to the goal must be $k - i$, and so we can infer that $h(X_i) \leq k - i$. As a consequence, it is valid to add the constraints $h(X_i) \leq k - i$.

Any consistent heuristic is also admissible, so a consistent heuristic also means that $h(X_i) \leq k - i$ is valid. In addition, a consistent heuristic means that the heuristic value drops by at most 1 (the cost) across an edge. Comparing X_{i+1} and X_i, the drop in heuristic value is $h(X_i) - h(X_{i+1})$. Thus, we have $h(X_i) - h(X_{i+1}) \leq 1$, or $h(X_{i+1}) \geq h(X_i) - 1$. Thus it is valid to add that constraint to the CSP as well.

(f) [2 pts] Now suppose we only know that the solution will have $\leq k$ moves. We do not need to find the optimal solution - we only need to find some solution of cost $\leq k$. Mark all of the following options such that if you make single change described in that line it will correctly modify the CSP to find some solution of cost $\leq k$. *Remember, the CSP can only have unary and binary constraints.*

☐ Remove the constraints "(X_i, X_{i+1}) is an edge in the search graph". Instead, add the constraints "(X_i, X_{i+1}) is an edge in the search graph, AND $X_i = X_{i+1}$".

🟥 Remove the constraints "(X_i, X_{i+1}) is an edge in the search graph". Instead, add the constraints "(X_i, X_{i+1}) is an edge in the search graph, OR $X_i = X_{i+1}$".

⊛ Remove the constraint "X_k is a goal state." Instead, add the constraint "There is some i, $0 \leq i \leq k$, such that X_i is a goal state".

☐ Remove the constraint "X_k is a goal state." Instead, add the constraint "For every $0 \leq i \leq k$, X_i is a goal state".

☐ None of the above

If we say (X_i, X_{i+1}) is an edge AND $X_i = X_{i+1}$, we are forcing all of the variables to be the same state s_0 (since they all have to be equal to each other), so that cannot be right.

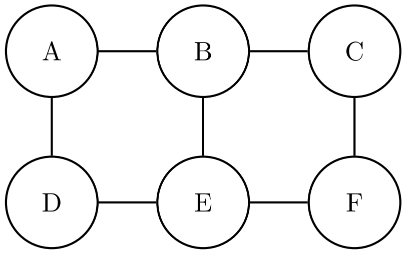
If we say (X_i, X_{i+1}) is an edge OR $X_i = X_{i+1}$, then basically the constraint says that transitions have to be made, but we can also choose to not make a transition from $i \to i + 1$ (in which case we would have $X_i = X_{i+1}$). This is exactly what we want - this will allow us to make only as many transitions as necessary, and then all the remaining "extra" states can be set to the previous state to satisfy the constraints.

It does not make sense to say that every state is a goal state - for example, the start state $X_0 = s_0$ is usually not a goal state.

Saying that there is some i such that X_i is a goal state would work. It is not a unary or binary constraint (it is a constraint on *all* of the variables). Because of the ambiguity in question statement regarding acceptable constraints, this option was ignored. (Either filled in or blank was accepted.)

Q4. [6 pts] Worst-Case Backtracking

Consider solving the following CSP with standard backtracking search where we enforce arc consistency of all arcs before every variable assignment. Assume every variable in the CSP has a domain size $d > 1$.



(a) For each of the variable orderings, mark the variables for which backtracking search (with arc consistency checking) could end up considering more than one different value during the search.

(i) [1 pt] Ordering: A, B, C, D, E, F

[✓ A] [✓ B] ☐ C ☐ D ☐ E ☐ F

(ii) [1 pt] Ordering: B, D, F, E, C, A

☐ A [✓ B] ☐ C ☐ D ☐ E ☐ F

Since we are enforcing arc consistency before every value assignment, we will only be guaranteed that we won't need to backtrack when our remaining variables left to be assign form a tree structure (or a forest of trees). For the first ordering, after we assign A and B, the nodes C, D, E, F, form a tree. For the second ordering, after we assign B, the nodes A, C, D, E, F form a tree.

(b) Now assume that an adversary gets to observe which variable ordering you are using, and after doing so, gets to choose to add one additional binary constraint between any pair of variables in the CSP in order to maximize the number of variables that backtracking could occur in the worst case. For each of the following variable orderings, select which additional binary constraint should the adversary add. Then, mark the variables for which backtracking search (with arc consistency checking) could end up considering more than one different value during the search when solving the modified CSP.

(i) [2 pts] Ordering: A, B, C, D, E, F

The adversary should add the additional binary constraint:

○ AC ○ AE ○ AF ○ BD
○ BF ○ CD ○ CE [✓ DF]

When solving the modified CSP with this ordering, backtracking might occur at the following variable(s):

[✓ A] [✓ B] [✓ C] [✓ D] ☐ E ☐ F

By adding the edge DF, now only after we assign A, B, C, D, the remaining nodes E, F form a tree.

(ii) [2 pts] Ordering: B, D, F, E, C, A

The adversary should add the additional binary constraint:

○ AC ○ AE ○ AF ○ BD
○ BF ○ CD [✓ CE] ○ DF

When solving the modified CSP with this ordering, backtracking might occur at the following variable(s):

☐ A 🟥 B ☐ C 🟥 D ☐ E 🟥 F

By adding the edge CE, now only after we assign B, D, F, the remaining nodes A, C, E form a tree.

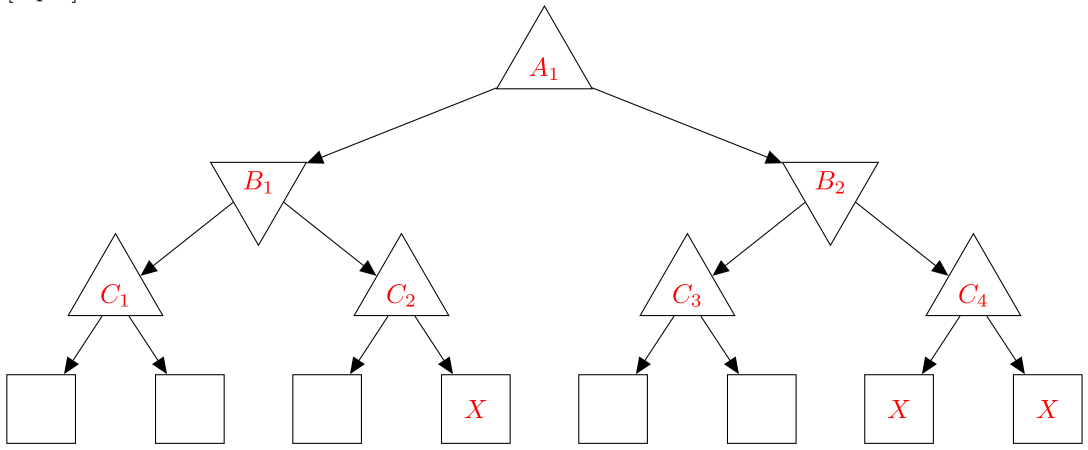
Q5. [7 pts] Best-Case Pruning

For the following zero-sum game trees, the upward pointing triangles represent maximizer nodes, and the downward pointing triangles represent minimizer nodes. Assume that we expand the children of each node in the game tree from left to right. For each tree, cross out the maximal set of **leaf** utility nodes (represented by squares) that can possibly be pruned with a single assignment of the utility nodes, in order to determine the correct minimax value of the root of the game tree. You do *not* need to provide such assignment of the utility nodes.

For a zero-sum game tree with max and min nodes, the pruning condition for a max node is $v > \beta$. This pruning condition can only be satisfied when (1) v is finite (which means we know the value down one branch of that node), and (2) β is finite, which means that at some min node on path from the current max node to the root of the game tree, we know the best value that the minimizer can guarantee by taking some branch of that node. Similarly, the pruning condition for a min node is $v < \alpha$. This pruning condition can only be satisfied when (1) v is finite (which means we know the value down one branch of that node), and (2) α is finite, which means that at some max node on path from the current min node to the root of the game tree, we know the best value that the maximizer can guarantee by taking some branch of that node.

For the solutions in the following parts, we have labeled the intermediate nodes in the graph.

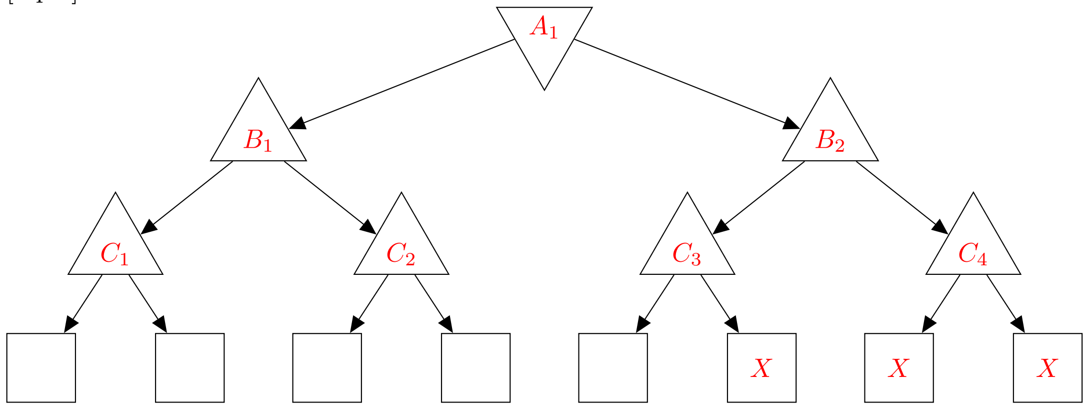
(a) [2 pts]



- It is possible to prune C_2's right branch. After C_2 explores its left branch, the value v at C_2 will be finite. β would also be finite since we have explored B_1's left branch and know the best value the minimizer can guarantee at B_1 so far. So we cross out the 4th utility leaf node from the left.

- It is possible to prune B_2's entire right branch. After B_2 explores its left branch, the value v at B_2 will be finite. α would also be finite since we have explored A_1's left branch and know the best value the maximizer can guarantee at A_1 so far. So we cross out the last two utility leaf nodes.

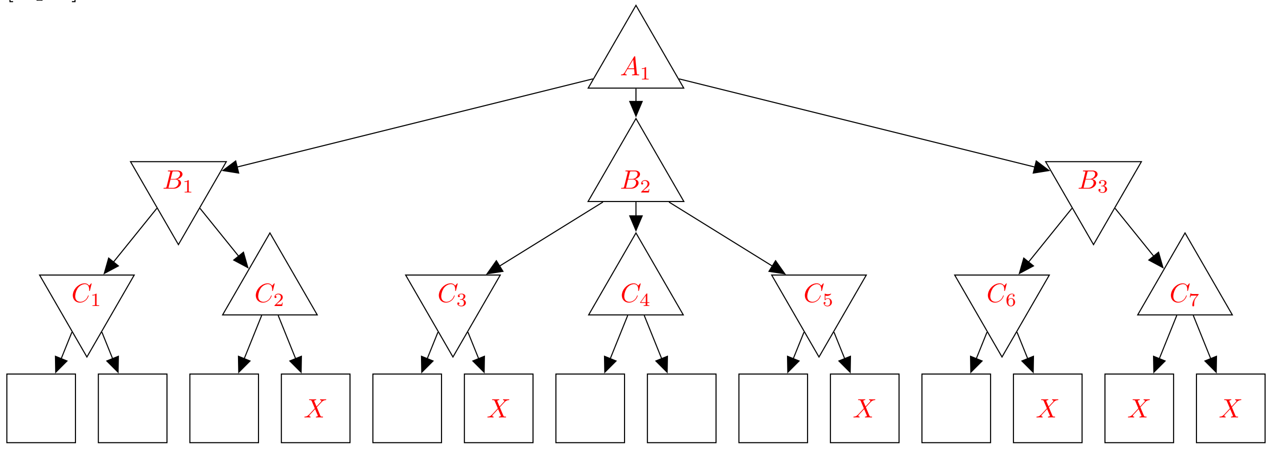
(b) [2 pts]



- It is possible to prune C_3's right branch. After C_3 explores its left branch, the value v at C_3 will be finite. β would also be finite since we have explored A_1's left branch and know the best value the minimizer can guarantee at A_1 so far. So we cross out the 6th utility leaf node from the left.

- It is possible to prune B_2's entire right branch. After B_2 explores its left branch, the value v at B_2 will be finite. β would also be finite since we have explored A_1's left branch and know the best value the minimizer can guarantee at A_1 so far. So we cross out the last two utility leaf nodes.

(c) [3 pts]



- It is possible to prune C_2's right branch. After C_2 explores its left branch, the value v at C_2 will be finite. β would also be finite since we have explored B_1's left branch and know the best value the minimizer can guarantee at B_1 so far. So we cross out the 4th utility leaf node from the left.

- It is possible to prune C_3's right branch. After C_3 explores its left branch, the value v at C_3 will be finite. α would also be finite since we have explored A_1's left branch and know the best value the maximizer can guarantee at A_1 so far. So we cross out the 6th utility leaf node from the left.

- It is possible to prune C_5's right branch. After C_5 explores its left branch, the value v at C_5 will be finite. α would also be finite since we have explored A_1's left branch and know the best value the maximizer can guarantee at A_1 so far. So we cross out the 10th utility leaf node from the left.

- It is possible to prune C_6's right branch. After C_6 explores its left branch, the value v at C_6 will be finite. α would also be finite since we have explored A_1's left and middle branches and know the best value the maximizer can guarantee at A_1 so far. So we cross out the 12th utility leaf node from the left.

- It is possible to prune B_3's entire right branch. After B_3 explores its left branch, the value v at B_2 will be finite. β would also be finite since we have explored A_1's left and middle branches and know the best value the maximizer can guarantee at A_1 so far. So we cross out the last two utility leaf nodes.

Q6. [10 pts] Ghostbusters

Suppose Pacman gets a noisy observation of a ghost's location for T moves, and then may guess where the ghost is at timestep T to eat it. To model the problem, you use an HMM, where the ith hidden state is the location of the ghost at timestep i and the ith evidence variable is the noisy observation of the ghost's location at time step i. *Assume Pacman always acts rationally.*

(a) [2 pts] If Pacman guesses correctly, he gets to eat the ghost resulting in a utility of 20. Otherwise he gets a utility of 0. If he does not make any guess, he gets a utility of 0.

Which of the following algorithms could Pacman use to determine the ghost's most likely location at time T? (Don't worry about runtime.)

- [✓] Variable elimination on the Bayes Net representing the HMM
- [✓] Particle filtering with a lot of particles
- [ ] Viterbi
- [✓] Forward algorithm for HMMs
- [ ] None of the above, Pacman should use ________

We want to find the ghost location X_T that maximizes $P(X_T|e_{1:T})$. This can be done by calculating $P(X_T|e_{1:T})$ using the forward algorithm or variable elimination, and can be estimated using particle filtering. However, it cannot be calculated using Viterbi (since that maximizes $P(X_1, \cdots X_T|e_{1:T})$).

(b) [2 pts] In the previous part, there was no penalty for guessing. Now, Pacman has to *pay* 10 *utility* in order to try to eat the ghost. Once he pays, he still gets 20 utility for correctly guessing and eating the ghost, and 0 utility for an incorrect guess. Pacman determines that the most likely ghost location at time T is (x, y), and the probability of that location is p.

What is the expected utility of guessing that the ghost is at (x, y), as a function of p? ___$20p - 10$___

With probability p, Pacman is right and gets utility 20, and with probability $1 - p$ he is wrong and gets utility 0. He always pays 10 utility. So the expected utility becomes $20p + 0(1 - p) - 10$.

When should Pacman guess that the ghost is at (x, y)?

- ○ Never (he should not guess)
- ○ If $p <$ ________.
- [✓ ●] If $p >$ ___0.5___.
- ○ Always

Not guessing has a utility of 0, so Pacman should guess when the expected utility of guessing is > 0, which is when $p > 0.5$.

(c) [2 pts] Now, in addition to the -10 utility for trying to eat the ghost, Pacman can also pay 5 utility to learn the exact location of the ghost. (So, if Pacman pays the 5 utility and eats the ghost, he pays 15 utility and gains 20 utility for a total of 5 utility.)

When should Pacman pay the 5 utility to find the exact ghost location?

- ○ Never
- [✓ ●] If $p <$ ___0.75___.
- ○ If $p >$ ________.
- ○ Always

Paying 5 utility means that Pacman is guaranteed to eat the ghost, getting $20 - 10 - 5 = 5$ utility in total. He should choose this option when it is better than the other two options (not guessing, or guessing without the info). This happens when $5 > 0$ and $5 > 20p - 10$, and thus it would be when $p < 0.75$.

(d) Now, Pacman can try to eat one out of Blinky (B), Inky (I) and Clyde (C) (three of the ghosts). He has some preferences about which one to eat, but he's afraid that his preferences are not rational. Help him out by showing him a utility function that matches his listed preferences, or mark "Not possible" if no rational utility function will work. You may choose any real number for each utility value. **If "Not possible" is marked, we will ignore any written utility function.**

(i) [2 pts] The preferences are $B \prec I$ and $I \prec C$ and $[0.5, B; 0.5, C] \prec I$

$U(B)$	$U(I)$	$U(C)$
1	4	5

○ Not possible

(ii) [2 pts] The preferences are $I \prec B$ and $[0.5, B; 0.5, C] \prec C$ and $[0.5, B; 0.5, C] \prec [0.5, B; 0.5, I]$

$U(B)$	$U(I)$	$U(C)$

[✓ Not possible]

The second preference implies $B \prec C$, the third implies $C \prec I$, and so we have $I \prec B \prec C \prec I$, which is irrational and no utility function would work.

Q7. [6 pts] MDPs

Pacman is in the 3x3 gridworld shown below. In each grid cell, Pacman has 5 actions available: [$\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$, $\circ$]. Taking the $\circ$ action moves Pacman to a special **Done** state and ends the game. All actions are deterministic. *Pacman is not allowed to take an action into the wall.* Otherwise, *all actions* (including $\circ$) are available from *all grid cells.*

For each policy, mark the reward function/discount factor pairs for which the policy is optimal.

1. $R_1(s,a,s') = \begin{cases} 1 & s=(0,0), a=\circ, s'=\textbf{Done} \\ 0 & else \end{cases}$

2. $R_2(s,a,s') = \begin{cases} 1 & s=(0,0) \\ 0 & else \end{cases}$

3. $R_3(s,a,s') = \begin{cases} 2 & s'=\textbf{Done} \\ 1 & else \end{cases}$

4. $R_4(s,a,s') = \begin{cases} -3 & a=\circ \\ -1 & else \end{cases}$

Hint: for any $x \in \mathbb{R}$, $|x| < 1$, we have $1 + x + x^2 + x^3 + x^4 + \cdots = 1/(1-x)$.

We should first think about what each reward function/discount factor wants us to do on an intuitive level:

1. $R_1, \gamma = 0.5$: Take action $\circ$ at $s = (0,0)$ as soon as possible.
2. $R_1, \gamma = 0.9$: Take action $\circ$ at state $(0,0)$ as soon as possible.
3. $R_2, \gamma = 0.5$: Take as many actions as possible from state $(0,0)$.
4. $R_2, \gamma = 0.9$: Take as many actions as possible from state $(0,0)$.
5. $R_3, \gamma = 0.5$: This one is a little tricky. For taking action $\circ$, our discounted sum of rewards is 2. For never taking action $\circ$, our discounted sum of rewards is also 2. Therefore, any policy is optimal for this reward function/discount factor.
6. $R_3, \gamma = 0.9$: With a higher discount factor, the optimal policy here is to never take action $\circ$.
7. $R_4, \gamma = 0.5$: For taking action $\circ$, our discount sum of rewards is -3. For never taking action $\circ$, our discounted sum of rewards is -2. Therefore, the optimal policy is to never take action $\circ$.
8. $R_4, \gamma = 0.9$: Again, taking action $\circ$ will give us a discounted sum of rewards of -3. Now, never taking action $\circ$ will give us a discounted sum of rewards of $-1 + 0.9(-1) + 0.9^2(-1) + ... = -1/(1-0.9) = -9$. Therefore, the optimal policy is to take action $\circ$ at every grid cell.

(a) [2 pts]

	0	1	2
0	$\circ$	$\leftarrow$	$\leftarrow$
1	$\uparrow$	$\uparrow$	$\uparrow$
2	$\uparrow$	$\uparrow$	$\uparrow$

[✓ R_1, $\gamma = 0.5$] [✓ R_1, $\gamma = 0.9$]
☐ R_2, $\gamma = 0.5$ ☐ R_2, $\gamma = 0.9$
[✓ R_3, $\gamma = 0.5$] ☐ R_3, $\gamma = 0.9$
☐ R_4, $\gamma = 0.5$ ☐ R_4, $\gamma = 0.9$
☐ None of the provided options

(b) [2 pts]

	0	1	2
0	$\rightarrow$	$\leftarrow$	$\leftarrow$
1	$\uparrow$	$\uparrow$	$\uparrow$
2	$\uparrow$	$\uparrow$	$\uparrow$

☐ R_1, $\gamma = 0.5$	☐ R_1, $\gamma = 0.9$
[✓ R_2, $\gamma = 0.5$]	[✓ R_2, $\gamma = 0.9$]
[✓ R_3, $\gamma = 0.5$]	[✓ R_3, $\gamma = 0.9$]
[✓ R_4, $\gamma = 0.5$]	☐ R_4, $\gamma = 0.9$
☐ None of the provided options	

(c) [2 pts]

	0	1	2
0	$\rightarrow$	$\rightarrow$	$\downarrow$
1	$\uparrow$	$\leftarrow$	$\downarrow$
2	$\uparrow$	$\leftarrow$	$\leftarrow$

☐ R_1, $\gamma = 0.5$	☐ R_1, $\gamma = 0.9$
☐ R_2, $\gamma = 0.5$	☐ R_2, $\gamma = 0.9$
[✓ R_3, $\gamma = 0.5$]	[✓ R_3, $\gamma = 0.9$]
[✓ R_4, $\gamma = 0.5$]	☐ R_4, $\gamma = 0.9$
☐ None of the provided options	

Q8. [8 pts] RL

Pacman is in an unknown MDP where there are three states [A, B, C] and two actions [Stop, Go]. We are given the following samples generated from taking actions in the unknown MDP. For the following problems, assume $\gamma = 1$ and $\alpha = 0.5$.

(a) We run Q-learning on the following samples:

s	**a**	**s'**	**r**
A	Go	B	2
C	Stop	A	0
B	Stop	A	-2
B	Go	C	-6
C	Go	A	2
A	Go	A	-2

What are the estimates for the following Q-values as obtained by Q-learning? All Q-values are initialized to 0.

(i) [2 pts] $Q(C, Stop) =$ 0.5

(ii) [2 pts] $Q(C, Go) =$ 1.5

For this, we only need to consider the following three samples.

$$Q(A, Go) \leftarrow (1-\alpha)Q(A, Go) + \alpha(r + \gamma \max_a Q(B, a)) = 0.5(0) + 0.5(2) = 1$$

$$Q(C, Stop) \leftarrow (1-\alpha)Q(C, Stop) + \alpha(r + \gamma \max_a Q(A, a)) = 0.5(0) + 0.5(1) = 0.5$$

$$Q(C, Go) \leftarrow (1-\alpha)Q(C, Go) + \alpha(r + \gamma \max_a Q(A, a)) = 0.5(0) + 0.5(3) = 1.5$$

(b) For this next part, we will switch to a feature based representation. We will use two features:

- $f_1(s, a) = 1$
- $f_2(s, a) = \begin{cases} 1 & a = \text{Go} \\ -1 & a = \text{Stop} \end{cases}$

Starting from initial weights of 0, compute the updated weights after observing the following samples:

s	**a**	**s'**	**r**
A	Go	B	4
B	Stop	A	0

What are the weights after the first update? (using the first sample)

(i) [1 pt] $w_1 =$ 2

(ii) [1 pt] $w_2 =$ 2

$$Q(A, Go) = w_1 f_1(A, Go) + w_2 f_2(A, Go) = 0$$

$$difference = [r + max_a Q(B, a)] - Q(A, Go) = 4$$

$$w_1 = w_1 + \alpha(difference) f_1 = 2$$

$$w_2 = w_2 + \alpha(difference) f_2 = 2$$

What are the weights after the second update? (using the second sample)

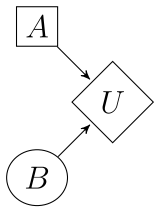
(iii) [1 pt] $w_1 =$ ___4___

(iv) [1 pt] $w_2 =$ ___0___

$$\begin{aligned} Q(B, Stop) &= w_1 f_1(B, Stop) + w_2 f_2(B, Stop) = 2(1) + 2(-1) = 0 \\ Q(A, Go) &= w_1 f_1(A, Go) + w_2 f_2(A, Go) = 2(1) + 2(1) = 4 \\ difference &= [r + max_a Q(A, a)] - Q(B, Stop) = [0 + 4] - 0 = 4 \\ w_1 &= w_1 + \alpha(difference) f_1 = 4 \\ w_2 &= w_2 + \alpha(difference) f_2 = 0 \end{aligned}$$

Q9. [8 pts] Bayes Net and Decision Networks

(a) [2 pts] We have the following decision network with the conditional probability and utility tables:



B	$P(B)$
0	x
1	$1-x$

B	A	$U(B,A)$
0	$+a$	10
0	$-a$	4
1	$+a$	0
1	$-a$	8

Suppose that we also know that $MEU(B) = 8.5$. What is the value of x?

$x =$ 0.25

First, from the definition of MEU, we have:

$$MEU(B) = P(B=0)MEU(B=0) + P(B=1)MEU(B=1)$$

When $B = 0$, we can either choose $A = +a$ to receive a utility of $U(0,+a) = 10$, or we can choose $A = -a$ to receive a utility of $U(0,-a) = 4$. So we choose $A = +a$ and thus $MEU(B=0) = \max(U(0,+a), U(0,-a)) = 10$.

When $B = 1$, we can either choose $A = +a$ to receive a utility of $U(1,+a) = 0$, or we can choose $A = -a$ to receive a utility of $U(1,-a) = 8$. So we choose $A = -a$ and thus $MEU(B=1) = \max(U(1,+a), U(1,-a)) = 8$.

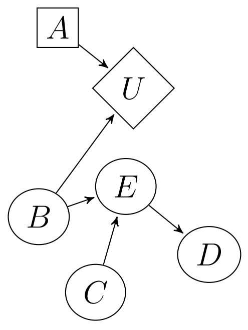
Substituting these into the above equation, using the parameterization of $P(B)$ above, and using our knowledge of $MEU(B)$, we have:

$$8.5 = MEU(B) = x * 10 + (1-x) * 8 = 8 + 2x$$

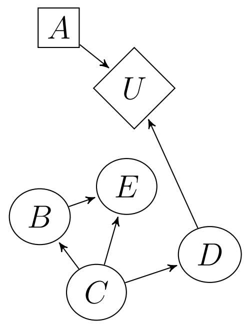
and so $x = 0.25$.

(b) Which of the following decision networks can simultaneously satisfy all of the given VPI and conditional independence constraints for some setting of conditional probability and utility tables? Mark the box below each decision network that can satisfy the constraints, or mark None of the above if none of the decision networks can satisfy the constraints.

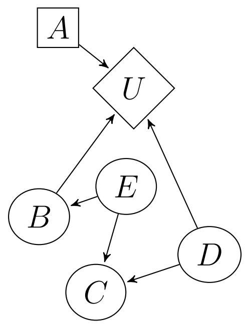
(i) [3 pts] $VPI(E) > 0, \quad E \perp\!\!\!\perp C$



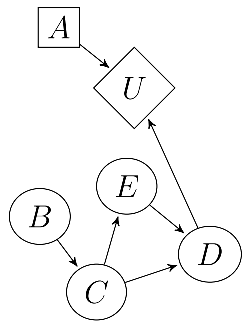
■



□



■



■

□ None of the above

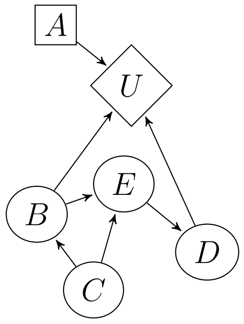
The first decision network can satisfy the constraints $VPI(E) > 0$ by making U's table dependent on the values of B and A, and making E dependent on B. The constraint $E \perp\!\!\!\perp C$ can be satisfied by setting E's table to not vary for different values of C (making it independent of C).

In order for the second decision network to satisfy the $E \perp\!\!\!\perp C$ constraint, either E must be independent of B and C, or it must be independent of C and E must be independent of C as well. In either case, E is now independent of D, which means that $VPI(E) = 0$. So this network cannot satisfy both constraints.

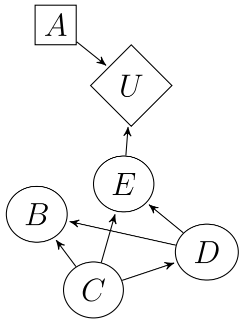
The third decision network can satisfy the constraints $VPI(E) > 0$ by making U's table dependent on the values of B and A, and making E dependent on B. The constraint $E \perp\!\!\!\perp C$ can be satisfied by setting E's table to not vary for different values of C (making it independent of C).

The fourth decision network can satisfy the constraints $VPI(E) > 0$ by making U's table dependent on the values of D and A, and making E dependent on D. The constraint $E \perp\!\!\!\perp C$ can be satisfied by setting E's table to not vary for different values of C (making it independent of C).

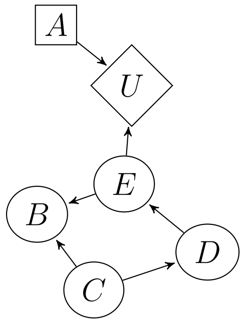
(ii) [3 pts] $VPI(C) > 0, \quad VPI(D|E) = 0, \quad C \perp\!\!\!\perp D, \quad C \perp\!\!\!\perp E|B$



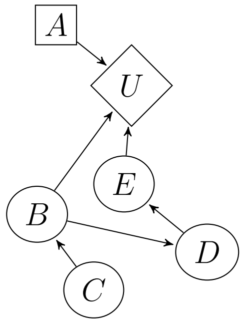
■



□



□



■

□ None of the above

The first decision network can satisfy the constraints $VPI(C) > 0$ by making U's table dependent on the values of B and A, and making B dependent on C. The constraints $C \perp\!\!\!\perp E|B$ and $C \perp\!\!\!\perp D$ can both be satisfied by making E's CPT not dependent on B and C. This allows the network to satisfy the constraint $VPI(D|E) = 0$ by making U not vary for different values of D, since D is independent of C and B.

In order for the second decision network to satisfy the $C \perp\!\!\!\perp D$ constraint, D's CPT must not depend on C. In order for the decision network to satisfy the $C \perp\!\!\!\perp E|B$ constraint, E's CPT must not depend on C and either B must not depend on C, or B must not depend on D, or E must not depend on D (to make the path from C to E through B and D inactive). In either case, C is now independent of E, which means that $VPI(C) = 0$. So this network cannot satisfy all constraints.

In order for the third decision network to satisfy the $C \perp\!\!\!\perp D$ constraint, D's CPT must not depend on C. In order for the decision network to satisfy the $C \perp\!\!\!\perp E|B$ constraint, either B must not depend on C, or B must not depend on E (to make the path from C to E through B inactive). In either case, C is now independent of E, which means that $VPI(C) = 0$. So this network cannot satisfy all constraints.

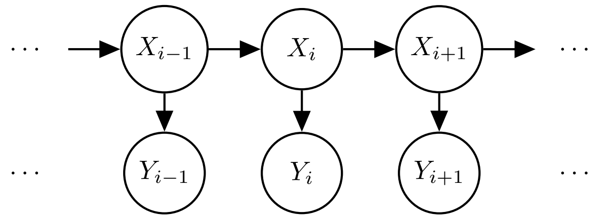
The fourth decision network can satisfy the constraints $VPI(C) > 0$ by making U's table dependent on the values of B and A, and making B dependent on C. The constraints $C \perp\!\!\!\perp E|B$ and $C \perp\!\!\!\perp D$ can both be satisfied by making D's CPT not dependent on B. This makes the network satisfy the constraint $VPI(D|E) = 0$ since $D \perp\!\!\!\perp B|E$ so the only path to U from D is through E.

Q10. [7 pts] DNA Sequencing

Suppose you want to model the problem of DNA sequencing using the following set-up:

- $X_i, Y_i \in \{A, T, C, G\}$
- X_i : ith base of an individual
- Y_i : ith base output by DNA sequencer

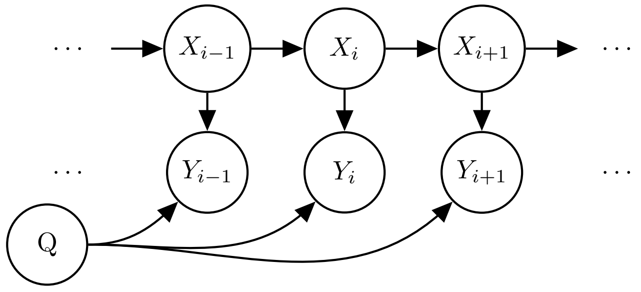
(a) First, you start by using a standard HMM model, shown below.



(i) [1 pt] Which of the following assumptions are made by the above HMM model

🟥 $X_i \perp\!\!\!\perp Y_{i+1} \mid X_{i+1} \quad \forall\, i$	🟥 $X_{i-1} \perp\!\!\!\perp X_{i+1} \mid X_i \quad \forall\, i$
☐ $X_i \perp\!\!\!\perp X_j \quad \forall\, i \neq j$	☐ $X_i \perp\!\!\!\perp Y_j \quad \forall\, i \neq j$
☐ $Y_i \perp\!\!\!\perp Y_j \quad \forall\, i \neq j$	☐ None of the provided options.

(b) Now you want to model the quality of your sequencer with a random variable Q, and decide to use the following modified HMM:



(i) [2 pts] Which of the following assumptions are made by the above modified HMM model?

🟥 $X_{i-1} \perp\!\!\!\perp X_{i+1} \mid X_i \quad \forall\, i$	☐ $Q \perp\!\!\!\perp X_i \mid Y_i \quad \forall\, i$
☐ $X_i \perp\!\!\!\perp X_j \quad \forall\, i \neq j$	☐ $Q \perp\!\!\!\perp X_i \mid Y_1, ...Y_N \quad \forall\, i$
☐ $X_i \perp\!\!\!\perp Y_j \quad \forall\, i \neq j$	🟥 $Q \perp\!\!\!\perp X_i \quad \forall\, i$
☐ $Y_i \perp\!\!\!\perp Y_j \quad \forall\, i \neq j$	☐ None of the provided options.
🟥 $X_i \perp\!\!\!\perp Y_{i+1} \mid X_{i+1} \quad \forall\, i$	

(ii) [2 pts] You observe the sequencer output $y_1, \ldots, y_N$ and want to estimate probability distribution of the particular sequence of length c starting at base k: $P(X_k \ldots X_{k+c-1} \mid y_1, \ldots y_N)$.
Select all elimination orderings which are maximally efficient with respect to the sum of the generated factors' sizes.

☐ $X_1, \ldots, X_{k-1}, X_{k+c}, \ldots, X_N, Q$	☐ $Q, X_1, \ldots, X_{k-1}, X_{k+c}, \ldots, X_N$
☐ $Q, X_1, \ldots, X_{k-1}, X_N, \ldots, X_{k+c}$	☐ $X_1, \ldots, X_{k-1}, Q, X_N, \ldots, X_{k+c}$
☐ $X_1, \ldots, X_{k-1}, Q, X_{k+c}, \ldots, X_N$	☐ None of the provided options: _________
🟥 $X_1, \ldots, X_{k-1}, X_N, \ldots, X_{k+c}, Q$	

The most efficient ordering above eliminates Q last because eliminating Q generates a factor that includes all of the remaining variables. It is more efficient to eliminate from the outside, in because they generate factors with only 1 variable rather than 2.

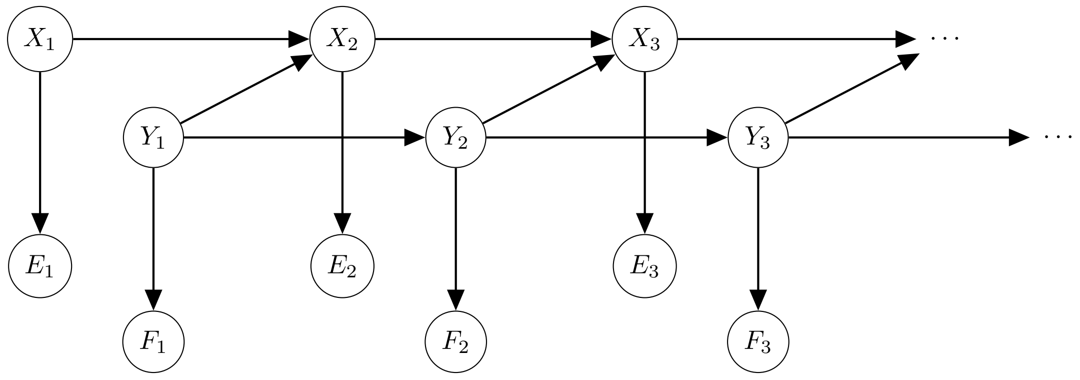
(iii) [2 pts] How many entries are in the final conditional probability table $P(X_k, \ldots, X_{k+c-1} \mid y_1, \ldots, y_N)$? The answer takes the form a^b – what are a and b?

$a =$ 4 $\qquad\qquad b =$ c

Each X_i can take on 4 values and there are c variables (the Y's are fixed). Therefore, there are 4^c entries in the CPT.

Q11. [11 pts] Dynamic Bayes' Nets

Suppose you have the following Dynamic Bayes Net model, with the associated conditional probability tables (CPTs).



X_1	$\mathbf{P}(X_1)$
$+x_1$	0.5
$-x_1$	0.5

Y_1	$\mathbf{P}(Y_1)$
$+y_1$	0.5
$-y_1$	0.5

X_t	E_t	$\mathbf{P}(E_t \mid X_t)$
$+x_t$	$+e_t$	0.2
$+x_t$	$-e_t$	0.8
$-x_t$	$+e_t$	0.5
$-x_t$	$-e_t$	0.5

Y_t	F_t	$\mathbf{P}(F_t \mid Y_t)$
$+y_t$	$+f_t$	0.4
$+y_t$	$-f_t$	0.6
$-y_t$	$+f_t$	0.8
$-y_t$	$-f_t$	0.2

X_t	Y_t	X_{t+1}	$\mathbf{P}(\ X_{t+1} \mid X_t, Y_t)$
$+x_t$	$+y_t$	$+x_{t+1}$	0.8
$+x_t$	$+y_t$	$-x_{t+1}$	0.2
$+x_t$	$-y_t$	$+x_{t+1}$	0.5
$+x_t$	$-y_t$	$-x_{t+1}$	0.5
$-x_t$	$+y_t$	$+x_{t+1}$	0.6
$-x_t$	$+y_t$	$-x_{t+1}$	0.4
$-x_t$	$-y_t$	$+x_{t+1}$	0.8
$-x_t$	$-y_t$	$-x_{t+1}$	0.2

Y_t	Y_{t+1}	$\mathbf{P}(\ Y_{t+1} \mid Y_t)$
$+y_t$	$+y_{t+1}$	0.6
$+y_t$	$-y_{t+1}$	0.4
$-y_t$	$+y_{t+1}$	0.2
$-y_t$	$-y_{t+1}$	0.8

You observe the evidence up to $t = 2$ as $(+e_1, -f_1, -e_2, +f_2)$ and want to infer $P(X_2, Y_2|E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$.

Throughout this problem, you may answer as either numeric expressions (e.g. 0.03+0.1∗0.5) or numeric values (e.g. 0.08), or **None** if you think no result can be obtained based on given information.

(a) [1 pt] *Prior Sampling.* The following five samples were generated from prior sampling. What is the sample based estimate of $P(Y_2 = +y_2 \mid E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$?

$-x_1$	$-y_1$	$+x_2$	$+y_2$	$+e_1$	$-f_1$	$-e_2$	$+f_2$
$-x_1$	$+y_1$	$+x_2$	$-y_2$	$+e_1$	$-f_1$	$-e_2$	$+f_2$
$+x_1$	$-y_1$	$+x_2$	$+y_2$	$+e_1$	$+f_1$	$-e_2$	$+f_2$
$+x_1$	$+y_1$	$-x_2$	$+y_2$	$+e_1$	$-f_1$	$-e_2$	$+f_2$
$+x_1$	$-y_1$	$-x_2$	$-y_2$	$+e_1$	$-f_1$	$-e_2$	$-f_2$

Answer: 2/3

Based on rejection sampling, we know the 3rd and 5th sample is not consistent with evidence, thus will be rejected. Out of three accepted samples, we have two samples with $Y_2 = +y_2$, so the estimation of $P(Y_2 = +y_2 \mid E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$ is 2/3.

(b) [1 pt] *Rejection Sampling.* You generate samples of $(X_1, Y_1, X_2, Y_2, E_1, F_1, E_2, F_2)$, variable by variable. In the first sample, you have already sampled the first four variables as $-x_1, -y_1, +x_2, +y_2$ and have not yet sampled (E_1, F_1, E_2, F_2). What is the probability of this sample being rejected?

Answer: $1 - 0.5 * 0.2 * 0.8 * 0.4 = 0.968$

The sample will be accepted if and only if the assignment of (E_1, F_1, E_2, F_2) is consistent with evidence. Therefore, the probability of being accepted is $P(E_1 = +e_1|X_1 = -x_1)P(F_1 = -f_1|Y_1 = -y_1)P(E_2 = -e_2|X_2 = +x_2)P(F_2 = +f_2|Y_2 = +y_2) = 0.5 * 0.2 * 0.8 * 0.4 = 0.032$. And the probability of being rejected is 1 minus the probability of being accepted.

(c) *Likelihood Weighting.* The following two samples were generated with likelihood weighting.

$$\begin{array}{cccccccc} -x_1 & -y_1 & +x_2 & +y_2 & +e_1 & -f_1 & -e_2 & +f_2 \\ -x_1 & -y_1 & +x_2 & -y_2 & +e_1 & -f_1 & -e_2 & +f_2 \end{array}$$

(i) [1 pt] What is the weight of the first sample?

Answer: $0.5 * 0.2 * 0.8 * 0.4 = 0.032$

(ii) [1 pt] What is the weight of the second sample?

Answer: $0.5 * 0.2 * 0.8 * 0.8 = 0.064$

(iii) [1 pt] What is the sample-based estimate of $P(Y_2 = +y_2 \mid E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$?

Answer: $0.4/(0.4 + 0.8) = 1/3$

Based on likelihood weighting sampling, we know weights for both samples are $P(E_1|X_1)P(F_1|Y_1)P(E_2|X_2)P(F_2|Y_2)$. Observe two sample are only different in the value of Y_2, we know the estimation of $P(Y_2 = +y_2 \mid E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$ is $w_1/(w_1 + w_2) = P(F_2 = +f_2|Y_2 = +y_2)/[P(F_2 = +f_2|Y_2 = +y_2) + P(F_2 = +f_2|Y_2 = -y_2)] = 0.4/(0.4 + 0.8) = 1/3$

(d) [2 pts] *Gibbs Sampling.* You want to use Gibbs Sampling to estimate $P(Y_2 = +y_2 \mid E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$, choosing to ignore evidence at $t = 3$ and onward.
The current sample is

$$\begin{array}{cccccccc} -x_1 & -y_1 & +x_2 & +y_2 & +e_1 & -f_1 & -e_2 & +f_2 \end{array}$$

and the next step is to resample X_1, what is the probability that the new assignment to X_1 is $+x_1$?

Answer: $0.2/(0.2 + 0.8) = 0.2$

Gibbs sampling sample X_1 according to distribution $P(X_1|Y_1, X_2, Y_2, E_1, F_1, E_2, F_2)$ $\propto P(X_1, Y_1, X_2, Y_2, E_1, F_1, E_2, F_2) \propto P(X_2|X_1, Y_1)P(E_1|X_1)P(X_1)$. Given $X_2 = +x_2, Y_1 = -y_1, E_1 = +e_1$, we know $P(X_2 = +x_2|X_1 = +x_1, Y_1 = -y_1)P(E_1 = +e_1|X_1 = +x_1)P(X_1 = +x_1) = 0.5 * 0.2 * 0.5$ and $P(X_2 = +x_2|X_1 = -x_1, Y_1 = -y_1)P(E_1 = +e_1|X_1 = -x_1)P(X_1 = -x_1) = 0.8 * 0.5 * 0.5$. The answer follows by normalizing.

(e) *Particle Filtering.* You want to estimate $P(X_2, Y_2|E_1 = +e_1, F_1 = -f_1, E_2 = -e_2, F_2 = +f_2)$ using particle filtering.

(i) [1 pt] At $t = 1$, you have a single particle $(X_1 = -x_1, Y_1 = -y_1)$. After passing it through the transition model, what is the probability of this particle becoming $(X_2 = +x_2, Y_2 = +y_2)$?

Answer: $0.8 * 0.2 = 0.16$

After passing through the transition model, the probability of new particle is $P(X_2, Y_2|X_1, Y_1) = P(X_2|X_1, Y_1)P(Y_2|Y_1)$. Thus, $P(X_2 = +x_2|X_1 = -x_1, Y_1 = -y_1)P(Y_2 = +y_2|Y_1 = -y_1) = 0.8 * 0.2 = 0.16$

(ii) [1 pt] Suppose after passing the sample through the transition model, you have the particle: $(X_2 = +x_2, Y_2 = +y_2)$. What is the weight of this particle after the observe update?

Answer: $0.8 * 0.4 = 0.32$

The weights of new particle is calculated by $P(E_2, F_2|X_2, Y_2) = P(E_2|X_2)P(F_2|Y_2)$. which is $0.8 * 0.4$.

(f) [2 pts] You now want to estimate $P(X_t, Y_t|e_{1:t}, f_{1:t})$ for some large t. You have limited computational resources and can either get N samples before rejection for rejection sampling, get N samples with likelihood weighting, or track of N particles using particle filtering.
Rank these three algorithms, indicating "1" for the algorithm which will give the most accurate sample-based approximation for $P(X_t, Y_t|e_{1:t}, f_{1:t})$, and "3" for the least accurate.

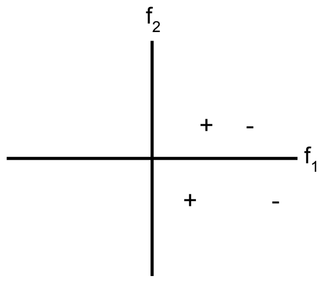
Rejection Sampling: 3 Likelihood Weighting: 2 Particle Filtering: 1

As t becomes large, rejection sampling will reject samples with high probability, only very limited amount of samples will be accepted to give a poor estimation. Likelihood weighting sampling is a bit better since at least every sample is used (never rejected). However, as t goes large, most samples will have extremely small weights and only a few samples will have reasonably large weights. The sample-based estimation of this algorithm is effectively only use the samples with large weights, (the contribution of extremely small weight samples is very little). The resampling mechanism of particle filtering effectively avoid this situation as in each time step, particles with small weight is resampled with small probability, and the particles remained tend to always be the one with large weight. Therefore, particle filtering gives the best approximation among the three algorithms.

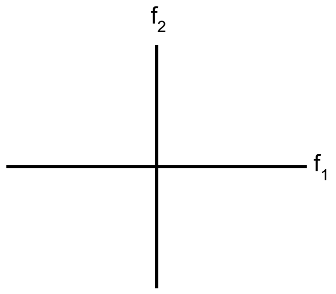
Q12. [15 pts] Decision Trees and Other Classifiers

(a) Suppose you have a small training data set of four points in *distinct* locations, two from the "+" class and two from the "–" class. For each of the following conditions, draw a particular training data set (**of exactly four points**: +, +, −, and −) that satisfy the conditions. If this is impossible, mark "Not possible". *If "Not possible" is marked, we will ignore any data points.*

For example, if the conditions were "A depth-1 decision tree can perfectly classify the training data points," an acceptable answer would be the data points to the right.



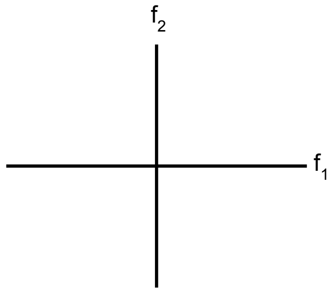
(i) [2 pts] A linear perceptron with a bias term can perfectly classify the training data points, but a linear perceptron without a bias term cannot.



○ Not possible

Any four points that are linearly separable, with the separating line clearly not passing through the origin

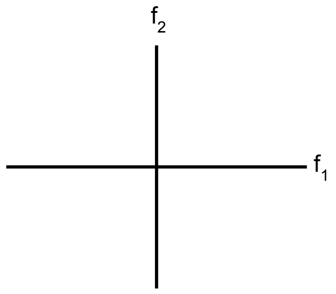
(ii) [2 pts] A dual perceptron with a quadratic kernel function $K(x, z) = (1 + x \cdot z)^2$ can perfectly classify the training data points, but a linear perceptron with a bias term cannot.



○ Not possible

Any four points where one class is "surrounding" the other class – e.g., "− + + −" in a line.

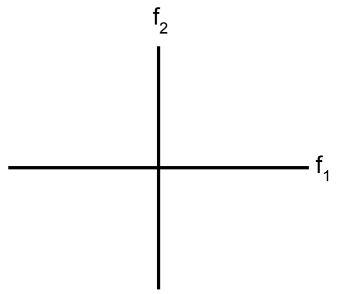
(iii) [2 pts] A depth-2 decision tree can classify the training data points perfectly, but a dual perceptron with a quadratic kernel function $K(x, z) = (1 + x \cdot z)^2$ cannot.



○ Not possible

Any configuration of four data points that are alternating, e.g. "− + − +"

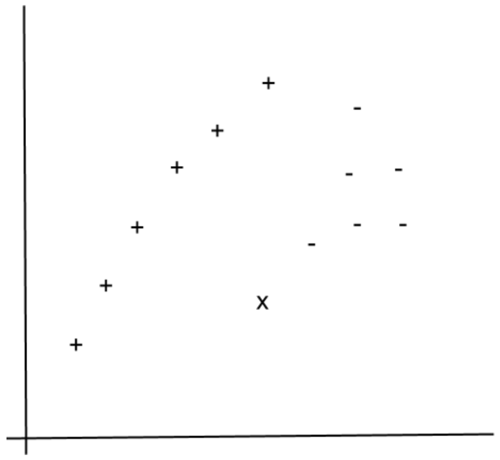
(iv) [2 pts] A depth-2 decision tree cannot classify the training data perfectly



● Not possible

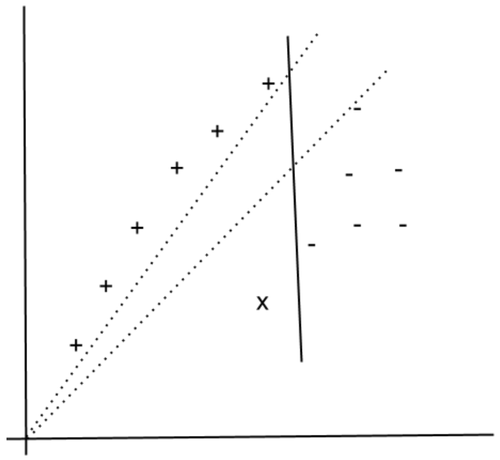
Not possible, since the points must be in distinct locations.

(b) [2 pts] The plot below shows training instances for two classes ("+" and "-"). You use various methods to train a classifier between "+" and "-". You are then given a new point (marked by the "x") and use the previously trained systems to classify the new point.



Which of these methods are guaranteed to classify the new point as a "-"?

■ Linear SVM (without using a bias term)
■ Linear SVM (using a bias term)
■ Dual perceptron with $K(x, z) = x \cdot z$
☐ Dual perceptron with $K(x, z) = x \cdot z + 1$
☐ None of the above



In the above figure, the dotted lines represent the range of linear classifiers that pass through origin while correctly classifying the data. A linear SVM without bias and dual perceptron classifier with $K(x, z) = x \cdot z$ will always learn a decision boundary passing through origin and therefore classify the "x" as "-".

A dual perceptron with $K(x,z) = x \cdot z + 1$ can learn any linear decision boundary, including the solid line shown, and is therefore not guaranteed to classify the "x" as "-".
For a linear SVM with a bias term, since it chooses a boundary that maximizes the margin, a decision boundary like the solid boundary, which correspond to a small margin, will not be learned. Therefore, it also will classify the "x" as "-".

(c) [2 pts] Let $\{x_i, y_i | i = 1 \ldots N\}$ be training examples and their class labels s.t. $y_i \in \{-1, 1\}$. Assume that the training data is separable when using a linear SVM with an additional bias feature.
For which of these kernel functions is the training data guaranteed to be separable using a dual perceptron?

[✓ $K(x,z) = x \cdot z + 1$] ☐ $K(x,z) = x \cdot z$
[✓ $K(x,z) = (x \cdot z + 1)^2$] ☐ None of the options

From the observation that a linear SVM with a bias term separates the data, we know that the data is linearly separable if we include an additional constant feature in the original feature descriptors.
Dual perceptron with $K(x,z) = x \cdot z$ corresponds to a linear perceptron in original feature space but without a constant feature - hence it is not guaranteed to perfectly separate the training data.
Dual perceptron with $K(x,z) = x \cdot z + 1$ corresponds to a linear perceptron in original feature space with a constant feature - we already know the training data is separable in this space.
Dual perceptron with $K(x,z) = (x \cdot z + 1)^2$ corresponds to a linear perceptron in the quadratic space which contains within itself the original features, a constant feature (as well as additional terms) i.e. it is more general than a feature space where training data is known to be linearly separable. Hence, the training data is also linearly separable in this space.

(d) You are still trying to classify between "+" and "-", but your two features now can take on only three possible values, $\{-1, 0, 1\}$. You would like to use a Naive Bayes model with the following CPTs:

X	$P(X)$
-	0.4
+	0.6

X	F_1	$P(F_1 \mid X)$
-	-1	0.4
-	0	0.5
-	1	0.1
+	-1	0.7
+	0	0.1
+	1	0.2

X	F_2	$P(F_2 \mid X)$
-	-1	0.1
-	0	0.1
-	1	0.8
+	-1	0.6
+	0	0.1
+	1	0.3

(i) [1 pt] If you observe that $F_1 = -1$ and $F_2 = -1$, how will you classify X using Naive Bayes?
○ $X = -$ [✓ $X = +$]

$P(F_1 = -1, F_2 = -1, X = +) = 0.7 * 0.6 * 0.6 > 0.4 * 0.1 * 0.4 = P(F_1 = -1, F_2 = -1, X = -)$

(ii) [1 pt] If you observe that $F_1 = 0$ and $F_2 = 0$, how will you classify X using Naive Bayes?
[✓ $X = -$] ○ $X = +$

$P(F_1 = 0, F_2 = 0, X = +) = 0.1 * 0.1 * 0.6 < 0.5 * 0.1 * 0.4 = P(F_1 = 0, F_2 = 0, X = -)$

(iii) [1 pt] If you observe that $F_1 = 1$ and $F_2 = 1$, how will you classify X using Naive Bayes?
○ $X = -$ [✓ $X = +$]

$P(F_1 = 1, F_2 = 1, X = +) = 0.2 * 0.3 * 0.6 > 0.8 * 0.1 * 0.4 = P(F_1 = 1, F_2 = 1, X = -)$

THIS PAGE IS INTENTIONALLY LEFT BLANK