CS 188 Spring 2013 Introduction to Artificial Intelligence Final

- You have approximately 2 hours and 50 minutes.
- The exam is closed book, closed notes except a three-page crib sheet.
- Please use non-programmable calculators only.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences AT MOST.

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.	Bounded suboptimal search: weighted A*	/19
Q2.	Generalizing Reinforcement Learning	/6
Q3.	Extending the Forward Algorithm	/18
Q4.	Dual Perceptron CSPs	/11
Q5.	Generalization	/18
Q6.	Fun with Probability	/17
Q7.	Games	/5
Q8.	Pruning	/6
	Total	/100

THIS PAGE IS INTENTIONALLY LEFT BLANK

Q1. [19 pts] Bounded suboptimal search: weighted A*

In this class you met A*, an algorithm for informed search guaranteed to return an optimal solution when given an admissible heuristic. Often in practical applications it is too expensive to find an optimal solution, so instead we search for good suboptimal solutions.

Weighted A* is a variant of A* commonly used for suboptimal search. Weighted A* is exactly the same as A* but where the f-value is computed differently:

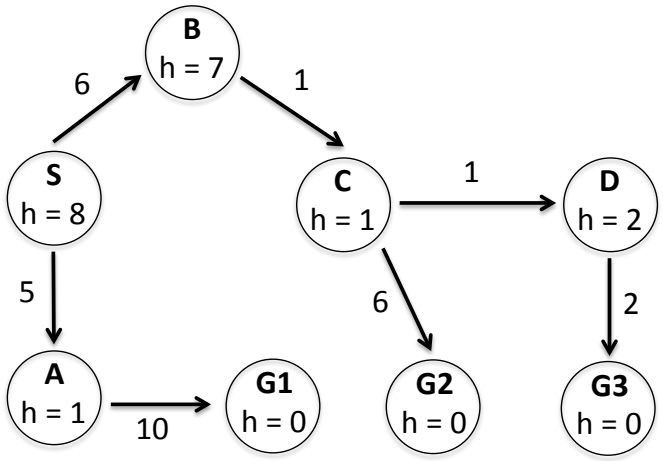
$$f(n) = g(n) + \varepsilon\, h(n)$$

where $\varepsilon \geq 1$ is a parameter given to the algorithm. In general, the larger the value of ε, the faster the search is, and the higher cost of the goal found.

Pseudocode for weighted A* tree search is given below. **NOTE:** The only differences from the A* tree search pseudocode presented in the lectures are: (1) *fringe* is assumed to be initialized with the start node before this function is called (this will be important later), and (2) now INSERT takes ε as a parameter so it can compute the correct f-value of the node.

```
1: function WEIGHTED-A*-TREE-SEARCH(problem, fringe, ε)
2:     loop do
3:         if fringe is empty then return failure
4:         node ← REMOVE-FRONT(fringe)
5:         if GOAL-TEST(problem, STATE[node]) then return node
6:         for child-node in child-nodes do
7:             fringe ← INSERT(child-node, fringe, ε)
```

(a) [2 pts] We'll first examine how weighted A* works on the following graph:



Execute weighted A* on the above graph with $\varepsilon = 2$, completing the following table. To save time, you can optionally just write the nodes added to the fringe, with their g and f values.

node	Goal?	*fringe*
-	-	$\{S : g = 0, f = 16\}$
S	No	$\{S \to A : g = 5, f = 7;\ S \to B : g = 6, f = 20\}$

(b) [5 pts] After running weighted A* with weight $\varepsilon \geq 1$ a goal node G is found, of cost $g(G)$. Let C^* be the optimal solution cost, and suppose the heuristic is admissible. Select the strongest bound below that holds, and provide a proof.

◯ $g(G) \leq \varepsilon\, C^*$ ◯ $g(G) \leq C^* + \varepsilon$ ◯ $g(G) \leq C^* + 2\varepsilon$ ◯ $g(G) \leq 2^{\varepsilon}\, C^*$ ◯ $g(G) \leq \varepsilon^2\, C^*$

Proof: (Partial credit for reasonable proof sketches.)

(c) Weighted A* includes a number of other algorithms as special cases. For each of the following, name the corresponding algorithm.

(i) [1 pt] $\varepsilon = 1$.

Algorithm:

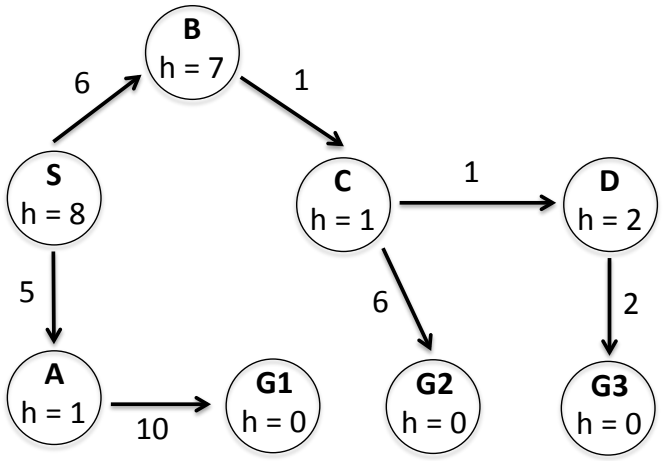
(ii) [1 pt] $\varepsilon = 0$.

Algorithm:

(iii) [1 pt] $\varepsilon \to \infty$ (i.e., as ε becomes arbitrarily large).

Algorithm:

(d) Here is the same graph again:



(i) [3 pts] Execute weighted A* on the above graph with $\varepsilon = 1$, completing the following table as in part (a):

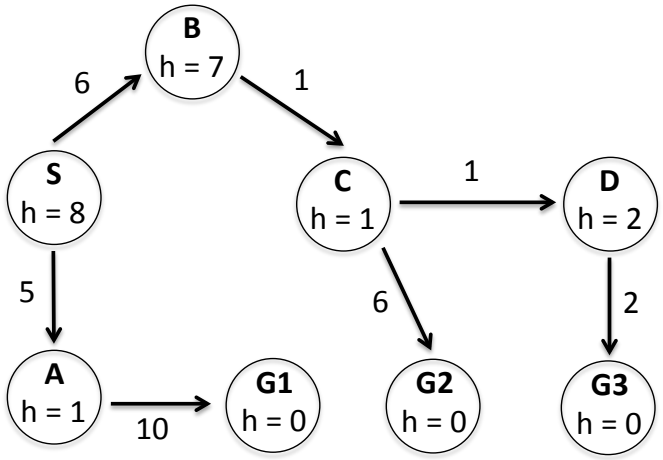
node	Goal?	*fringe*

(ii) [4 pts] You'll notice that weighted A* with $\varepsilon = 1$ repeats computations performed when run with $\varepsilon = 2$. Is there a way to reuse the computations from the $\varepsilon = 2$ search by starting the $\varepsilon = 1$ search with a different fringe? Let F denote the set that consists of both (i) all nodes the fringe the $\varepsilon = 2$ search ended with, and (ii) the goal node G it selected. Give a brief justification for your answer.

◯ Use F as new starting fringe
◯ Use F with goal G removed as new starting fringe
◯ Use F as new starting fringe, updating the f-values to account for the new ε
◯ Use F with goal G removed as new starting fringe, updating the f-values to account for the new ε
◯ Initialize the new starting fringe to all nodes visited in previous search
◯ Initialize the new starting fringe to all nodes visited in previous search, updating the f-values to account for the new ε
◯ It is not possible to reuse computations, initialize the new starting fringe as usual

Justification:

Here is the same graph again:



(iii) [2 pts] Now re-run the $\epsilon = 1$ search for the above graph using the fringe you selected in the previous question.

node	Goal?	*fringe*

Q2. [6 pts] Generalizing Reinforcement Learning

Assume we have an MDP with state space S, action space A, reward function $R(s, a, s')$, and discount γ. Our eventual goal is to learn a policy that can be used by a robot in the real world. However, we only have access to simulation software, not the robot directly. We know that the simulation software is built using the transition model $T_{\text{sim}}(s, a, s')$ which is unfortunately different than the transition model that governs our real robot, $T_{\text{real}}(s, a, s')$.

Without changing the simulation software, we want to use the samples drawn from the simulator to learn Q-values for our real robot.

Recall the Q-learning update rule. Given a sample (s, a, s', r), it performs the following update:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right]$$

(a) [4 pts] Assuming the samples are drawn from the simulator, which new update rule will learn the correct Q-value functions for the real world robot? Circle the correct update rule and provide an explanation for your choice in the box below.

- ◯ $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha T_{\text{sim}}(s, a, s') \left[r + \gamma \max_{a'} Q(s', a') \right]$
- ◯ $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha T_{\text{real}}(s, a, s') \left[r + \gamma \max_{a'} Q(s', a') \right]$
- ◯ $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \frac{1}{T_{\text{sim}}(s,a,s')} \left[r + \gamma \max_{a'} Q(s', a') \right]$
- ◯ $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \frac{1}{T_{\text{real}}(s,a,s')} \left[r + \gamma \max_{a'} Q(s', a') \right]$
- ◯ $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \frac{T_{\text{real}}(s,a,s')}{T_{\text{sim}}(s,a,s')} \left[r + \gamma \max_{a'} Q(s', a') \right]$
- ◯ $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \frac{T_{\text{sim}}(s,a,s')}{T_{\text{real}}(s,a,s')} \left[r + \gamma \max_{a'} Q(s', a') \right]$

Justification:

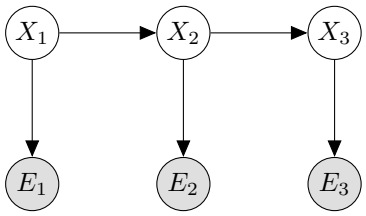
(b) [2 pts] Now consider the case where we have n real robots with transition models $T^1_{\text{real}}(s, a, s'), \ldots, T^n_{\text{real}}(s, a, s')$ and still only one simulator. Is there a way to learn policies for all n robots simultaneously by using the same samples from the simulator? If yes, explain how. If no, explain why not. (*1-2 sentences*)

◯Yes ◯No

Justification:

Q3. [18 pts] Extending the Forward Algorithm

Consider the HMM graph structure shown below.



Recall the Forward algorithm is a two step iterative algorithm used to approximate the probability distribution $P(X_t|e_1, \ldots, e_t)$. The two steps of the algorithm are as follows:

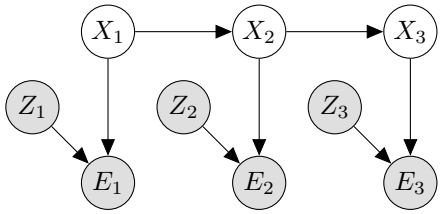
Elapse Time $P(X_t|e_1, \ldots, e_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_1, \ldots, e_{t-1})$

Observe $P(X_t|e_1, \ldots, e_t) = \frac{P(e_t|X_t)P(X_t|e_1,\ldots,e_{t-1})}{\sum_{x_t} P(e_t|x_t)P(x_t|e_1,\ldots,e_{t-1})}$

For this problem we will consider modifying the forward algorithm as the HMM graph structure changes. Our goal will continue to be to create an iterative algorithm which is able to compute the distribution of states, X_t, given all available evidence from time 0 to time t.

Note: If the probabilities required can be computed without *any* change to original update equations, mark the **no change** bubble. Otherwise write the new update equation inside the box.

Consider the graph below where new observed variables, Z_i, are introduced and influence the evidence.



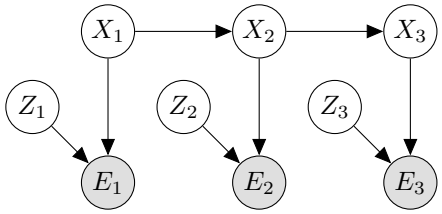
(a) [3 pts] State the modified Elapse Time update. ◯No Change

$P(X_t|e_1, \ldots, e_{t-1}, z_1, \ldots, z_{t-1}) =$

(b) [3 pts] State the modified Observe update. ◯No Change

$P(X_t|e_1, \ldots, e_t, z_1, \ldots, z_t) =$

Next, consider the graph below where the Z_i variables are unobserved.



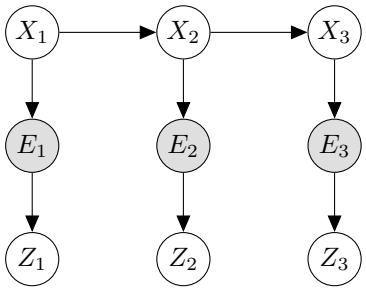
(c) [3 pts] State the modified Elapse Time update. ◯ No Change

$P(X_t|e_1, \ldots, e_{t-1}) =$

(d) [3 pts] State the modified Observe update. ◯No Change

$P(X_t|e_1, \ldots, e_t) =$

Finally, consider a graph where the newly introduced variables are unobserved and influenced by the evidence nodes.



(e) [3 pts] State the modified Elapse Time update. ◯ No Change

$P(X_t|e_1, \ldots, e_{t-1}) =$

(f) [3 pts] State the modified Observe update. ◯ No Change

$P(X_t|e_1, \ldots, e_t) =$

Q4. [11 pts] Dual Perceptron CSPs

In this question, we formulate the dual perceptron algorithm as a constraint satisfaction problem (CSP). We have a binary classification problem with classes $+1$ and -1 and a set of n training points, $x_1, x_2, ..., x_n$ with labels $y_i \in \{-1, +1\}$.

Recall that the dual perceptron algorithm takes as input a kernel function $K(x_i, x_j)$ defined on all pairs of training points x_i and x_j, estimates an α_i for each training point x_i, and predicts the class of a point z using $h_\alpha(z) = \sum_{i=1}^{n} \alpha_i K(x_i, z)$, classifying z as positive $(+1)$ if $h_\alpha(a) \geq 0$, and negative (-1) otherwise.

Let the α_i variables of the dual perceptron be the variables in a CSP, with domains restricted to $\{-1, 0, 1\}$. Each training point x_i induces a constraint c_i requiring that it is correctly classified with a margin of at least 1; i.e., $y_i h_\alpha(x_i) \geq 1$.

For this problem, we work with a predefined kernel function $K(x_i, x_j)$. The value of the kernel function (left) for the training points and their labels (right) are given in the tables below. In the kernel table, the j^{th} entry in the i^{th} row is the value of $K(x_i, x_j)$.

	x_1	x_2	x_3	x_4
x_1	1	0	0	-1
x_2	0	4	-2	-2
x_3	0	-2	1	1
x_4	-1	-2	1	2

i	1	2	3	4
y_i	-1	-1	+1	+1

(a) [2 pts] Write each constraint c_i as an inequality in terms of the variables α. (c_1 has been completed for you.)

c_1	$\alpha_1 - \alpha_4 \leq -1$	c_3	
c_2		c_4	

(b) We now randomly initialize to the full assignment $\alpha = (1, -1, 0, -1)$.

(i) [3 pts] For a constraint of the form $a \geq b$, define the *constraint violation margin* (CVM) as the difference $b - a$. For each of the above constraints, circle either *Satisfied* or *Violated* and compute the CVM.

	Satisfied?		CVM		Satisfied?		CVM
c_1	Satisfied	Violated		c_3	Satisfied	Violated	
c_2	Satisfied	Violated		c_4	Satisfied	Violated	

(ii) [4 pts] We decide to run a variation of the min-conflicts algorithm. Recall that min-conflicts begins with a full assignment of all variables and tries to get all constraints satisfied by iteratively modifying the assignment.

In our variant of the algorithm, a single iteration consists of selecting the currently most violated constraint—i.e., the constraint with the highest CVM—and then reassigning all variables that are part of the constraint to values such that the new CVM for the selected constraint is minimized.

Starting from the assignment above ($\alpha = (1, -1, 0, -1)$), run a single iteration of this algorithm. Indicate which constraint c_i is selected, then compute the updated assignment α' and the updated CVM for the selected constraint c_i. Finally, indicate whether or not after this single iteration all constraints have been satisfied (and the algorithm terminates).

Selected c_i	α'_1	α'_2	α'_3	α'_4	Updated CVM	Terminated?
						Yes No

(iii) [2 pts] Suppose we are given a solution to this CSP of $\alpha^* = (-1, -1, +1, +1)$. For each test point z_i whose kernel values with each training point are given in the table below, compute $h_{\alpha^*}(z_i)$ and the predicted classification for z_i.

	$K(x_1, z_i)$	$K(x_2, z_i)$	$K(x_3, z_i)$	$K(x_4, z_i)$	$h_{\alpha^*}(z_i)$	Class prediction?
z_1	3	0	-2	1		
z_2	-2	1	2	-2		

Q5. [18 pts] Generalization

We consider the following different classifiers for classification of samples in a 2-dimensional feature space.

PNoBias Linear perceptron *without* a bias term (features $\begin{bmatrix} x_1 & x_2 \end{bmatrix}^\intercal$)

PBias Linear perceptron with a bias term (features $\begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix}^\intercal$)

PQuad Kernel perceptron with the quadratic kernel function $K(x, z) = (1 + x \cdot z)^2$

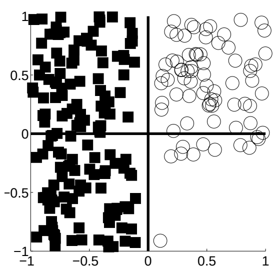
PCutoff Kernel perceptron with the kernel function $K(x, z) = \max\{0, 0.01 - ||x - z||_2\}$ ($||a - b||_2$ is the Euclidean distance between a and b)

1NN 1-nearest neighbor classifier

3NN 3-nearest neighbor classifier

(a) [8 pts] In each of the plots below you are given points from two classes, shown as filled rectangles and open circles. For each plot, fill in the bubble next to each classifier that will be able to perfectly classify all of the training data (or, if none, mark "None of these will classify the data perfectly").

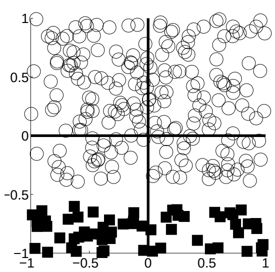
Note that when computing the nearest neighbors for a training data point, the training data point will be its own nearest neighbor.



○ PNoBias ○ PQuad ○ 1NN

○ PBias ○ PCutoff ○ 3NN

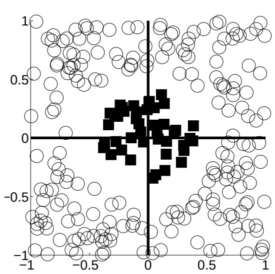
○ None of these will be able to classify the training data perfectly.



○ PNoBias ○ PQuad ○ 1NN

○ PBias ○ PCutoff ○ 3NN

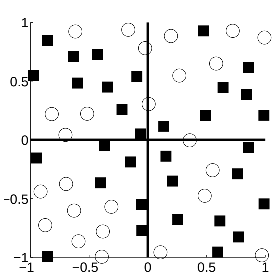
○ None of these will be able to classify the training data perfectly.



○ PNoBias ○ PQuad ○ 1NN

○ PBias ○ PCutoff ○ 3NN

○ None of these will be able to classify the training data perfectly.



○ PNoBias ○ PQuad ○ 1NN

○ PBias ○ PCutoff ○ 3NN

○ None of these will be able to classify the training data perfectly.

(b) **(i)** [5 pts] Suppose you train a classifier and test it on a held-out validation set. It gets 80% classification accuracy on the training set and 20% classification accuracy on the validation set.

From what problem is your model most likely suffering?

◯ Underfitting ◯ Overfitting

Fill in the bubble next to any measure of the following which could reasonably be expected to improve your classifier's performance on the validation set.

◯ Add extra features ◯ Remove some features

Briefly justify:

◯ Collect more training data ◯ Throw out some training data

Assuming features are outcome counts (k is the Laplace smoothing parameter controlling the number of extra times you "pretend" to have seen an outcome in the training data):

◯ Increase k ◯ Decrease k (assuming $k > 0$ currently)

Assuming your classifier is a Bayes' net:

◯ Add edges ◯ Remove edges

(ii) [3 pts] Suppose you train a classifier and test it on a held-out validation set. It gets 30% classification accuracy on the training set and 30% classification accuracy on the validation set.

From what problem is your model most likely suffering?

◯ Underfitting ◯ Overfitting

Fill in the bubble next to any measure of the following which could reasonably be expected to improve your classifier's performance on the validation set.

◯ Add extra features ◯ Remove some features

Briefly justify:

◯ Collect more training data ◯ Throw out some training data

(iii) [2 pts] Your boss provides you with an image dataset in which some of the images contain your company's logo, and others contain competitors' logos. You are tasked to code up a classifier to distinguish your company's logos from competitors' logos. You complete the assignment quickly and even send your boss your code for training the classifier, but your boss is furious. Your boss says that when running your code with images and a random label for each of the images as input, the classifier achieved perfect accuracy on the training set. And this happens for all of the many random labelings that were generated.

Do you agree that this is a problem? Justify your answer.

Q6. [17 pts] Fun with Probability

In this question you will be asked to complete a table specifying a count of samples drawn from a probability distribution, and subsequently answer questions about Bayes' nets constructed from the table using maximum likelihood techniques.

(a) [8 pts] The table below shows a count of samples drawn from a distribution over 4 variables: A, B, C and D. As each of the 4 variables is binary, there are 16 possible values a sample can take on. The counts for 15 of these have been recorded in the table below, but the remaining one is missing.

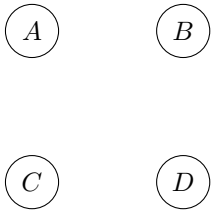
Calculate the remaining value such that a maximum likelihood estimate of the joint probability distribution over the 4 variables below will have the following properties: $A \perp\!\!\!\perp D \,|\, C$ and $B \perp\!\!\!\perp D \,|\, C$. You must show work in order to receive credit.

Hint: For this example just enforcing $B \perp\!\!\!\perp D \,|\, C$ is sufficient to find n. (I.e., the numbers in this examples are chosen such that after enforcing $B \perp\!\!\!\perp D \,|\, C$ the n you found will automatically also make $A \perp\!\!\!\perp D \,|\, C$ hold true.)

A	B	C	D	Sample Count
$+a$	$+b$	$+c$	$+d$	n
$-a$	$+b$	$+c$	$+d$	18
$+a$	$-b$	$+c$	$+d$	3
$-a$	$-b$	$+c$	$+d$	9
$+a$	$+b$	$-c$	$+d$	6
$-a$	$+b$	$-c$	$+d$	2
$+a$	$-b$	$-c$	$+d$	0
$-a$	$-b$	$-c$	$+d$	8
$+a$	$+b$	$+c$	$-d$	6
$-a$	$+b$	$+c$	$-d$	6
$+a$	$-b$	$+c$	$-d$	1
$-a$	$-b$	$+c$	$-d$	3
$+a$	$+b$	$-c$	$-d$	18
$-a$	$+b$	$-c$	$-d$	6
$+a$	$-b$	$-c$	$-d$	0
$-a$	$-b$	$-c$	$-d$	24

$n =$

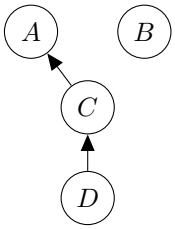
(b) [3 pts] Draw a Bayes' net that makes exactly the 2 independence assumptions indicated by the set of samples shown above: $A \perp\!\!\!\perp D \,|\, C$ and $B \perp\!\!\!\perp D \,|\, C$. Make sure it doesn't make any additional independence assumptions.



(c) [6 pts] Now we run maximum likelihood learning of the parameters for each of the Bayes' nets below from the data on the previous page. The result is four learned distributions $P^{(1)}(A, B, C, D)$, $P^{(2)}(A, B, C, D)$, $P^{(3)}(A, B, C, D)$, $P^{(4)}(A, B, C, D)$. We also define another distribution $P^{(0)}(A, B, C, D)$ which is obtained by renormalizing the table on the previous page. Fill in the bubble for every true statement below.

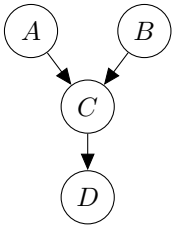
Hint: You shouldn't need to inspect the numbers in the table on the previous page. The key information about the table on the previous page is that the numbers are such that $P^{(0)}(A, B, C, D)$ satisfies $A \perp\!\!\!\perp D \mid C$ and $B \perp\!\!\!\perp D \mid C$, but doesn't satisfy any other independence assumptions.

$P^{(1)}$



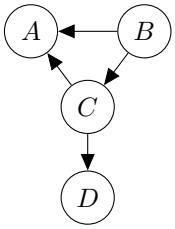
◯ $P^{(0)}(B) = P^{(1)}(B)$ ◯ $P^{(0)}(D \mid C) = P^{(1)}(D \mid C)$ ◯ $P^{(0)}(A, B) = P^{(1)}(A, B)$

$P^{(2)}$



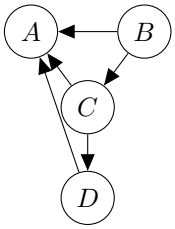
◯ $P^{(0)}(B) = P^{(2)}(B)$ ◯ $P^{(0)}(D \mid C) = P^{(2)}(D \mid C)$ ◯ $P^{(0)}(A, B) = P^{(2)}(A, B)$

$P^{(3)}$



◯ $P^{(0)}(B) = P^{(3)}(B)$ ◯ $P^{(0)}(D \mid C) = P^{(3)}(D \mid C)$ ◯ $P^{(0)}(A, B) = P^{(3)}(A, B)$

$P^{(4)}$

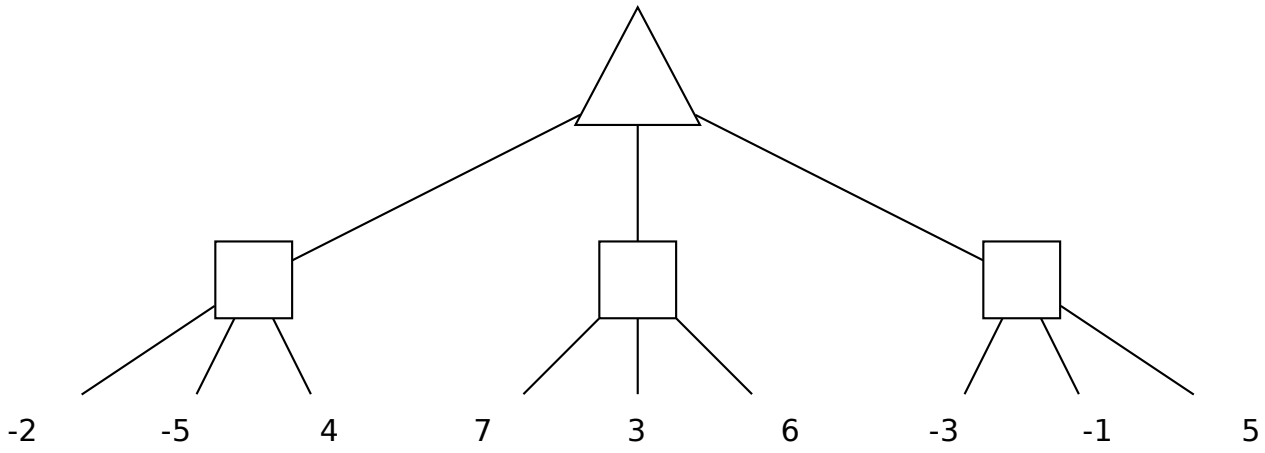


◯ $P^{(0)}(B) = P^{(4)}(B)$ ◯ $P^{(0)}(D \mid C) = P^{(4)}(D \mid C)$ ◯ $P^{(0)}(A, B) = P^{(4)}(A, B)$

Q7. [5 pts] Games

Consider a zero-sum game with two players, one maximizing agent and one minimizing agent, in which the ordering of moves is no longer deterministic. Each turn, a coin is flipped in order to determine which agent gets to make a move during that time step.

Consider the game tree below encoding the result of playing for two turns. It is currently the maximizer's move, so the top node is a max node, but we don't know which agent is going to play on the next turn, so we've replaced those nodes with boxes. Draw a new game tree that consists of only the traditional min, max, and expecti-nodes that models this situation.



Q8. [6 pts] Pruning

Pacman has a new kind of magic that allows him to look ahead during game search. Concretely, for a given node he can call a function GET-LOWEST-AVG-HIGHEST(node) which tells him the lowest, L, highest, H, and average, A, values of all leaves below that node in the tree.

Below is some modified code for performing alpha-beta pruning in this new situation. Select the choices that result in maximal pruning while still preserving that the correct value is found.

```
function MAX-VALUE(node, α, β)
    if node is leaf then
        return VALUE(node)
    (L, A, H) ← GET-LOWEST-AVG-HIGHEST(node)
    if [ (1) ] then
        return H
    v ← −∞
    for child ← CHILDREN(node) do
        v ← MAX(v, MIN-VALUE(child, α, β))
        if v == H then
            return v
        if v ≥ β then
            return v
        α ← MAX(α, v)
    return v
```

(1)

- ◯ $L < \alpha$
- ◯ $L < \beta$
- ◯ $L > \alpha$
- ◯ $L > \beta$
- ◯ $H < \alpha$
- ◯ $H < \beta$
- ◯ $H > \alpha$
- ◯ $H > \beta$
- ◯ $A < \alpha$
- ◯ $A < \beta$
- ◯ $A > \alpha$
- ◯ $A > \beta$

```
function MIN-VALUE(node, α, β)
    if node is leaf then
        return VALUE(node)
    (L, A, H) ← GET-LOWEST-AVG-HIGHEST(node)
    if [ (2) ] then
        return L
    v ← ∞
    for child ← CHILDREN(node) do
        v ← MIN(v, MAX-VALUE(child, α, β))
        if v == L then
            return v
        if v ≤ α then
            return v
        β ← MIN(β, v)
    return v
```

(2)

- ◯ $L < \alpha$
- ◯ $L < \beta$
- ◯ $L > \alpha$
- ◯ $L > \beta$
- ◯ $H < \alpha$
- ◯ $H < \beta$
- ◯ $H > \alpha$
- ◯ $H > \beta$
- ◯ $A < \alpha$
- ◯ $A < \beta$
- ◯ $A > \alpha$
- ◯ $A > \beta$

THIS PAGE IS INTENTIONALLY LEFT BLANK