

- 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  $\varepsilon$ , the faster the search is, and the higher cost of the goal found.

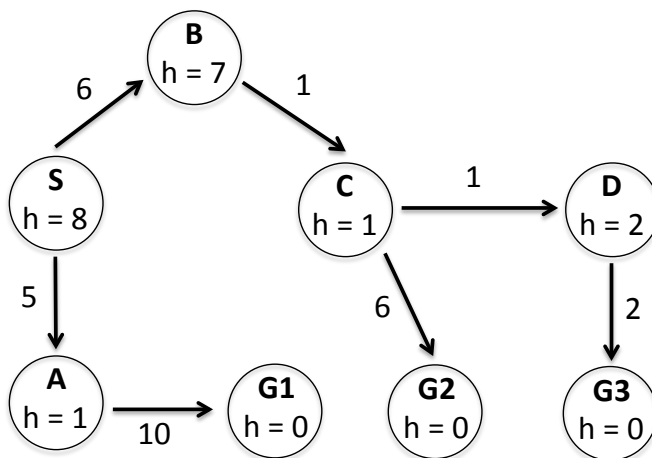
Pseudocode for weighted A\* tree search is given below. **NOTE:** The only differences from the A\* tree search pseudocode presented in the lectures are: (1) *fringe* is assumed to be initialized with the start node before this function is called (this will be important later), and (2) now INSERT takes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

```

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

```

(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.

<i>node</i>	Goal?	<i>fringe</i>
-	-	{S : <i>g</i> = 0, <i>f</i> = 16}
S	No	{S → A : <i>g</i> = 5, <i>f</i> = 7; S → B : <i>g</i> = 6, <i>f</i> = 20}
S → A	No	{S → A → G1 : <i>g</i> = 15, <i>f</i> = 15; S → B : <i>g</i> = 6, <i>f</i> = 20}
S → A → G1	Yes	-

- (b) [5 pts] After running weighted A\* with weight  $\epsilon \geq 1$  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.

- $g(G) \leq \epsilon C^*$         $g(G) \leq C^* + \epsilon$         $g(G) \leq C^* + 2\epsilon$         $g(G) \leq 2^\epsilon C^*$         $g(G) \leq \epsilon^2 C^*$

Proof: (Partial credit for reasonable proof sketches.)

When weighted A\* terminates, an ancestor  $n$  of the optimal goal  $G^*$  is on the fringe. Since  $G$  was expanded before  $n$ , we have  $f(G) \leq f(n)$ . As a result:

$$g(G) = f(G) \leq f(n) = g(n) + \epsilon h(n) \leq \epsilon(g(n) + h(n)) \leq \epsilon C^*$$

If you're confused about whether this all comes from, remember that  $f(n) = g(n) + \epsilon h(n)$  comes from the problem statement and the inequality  $g(n) + \epsilon h(n) \leq \epsilon(g(n) + h(n))$  is true by algebra.

Since we know that  $g(n)$  is non-negative, it must be true that  $g(n) + \epsilon h(n) \leq \epsilon(g(n) + h(n))$ . This is a common technique used when trying to prove/find a bound.

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

- (i) [1 pt]  $\epsilon = 1$ .

Algorithm: A\*

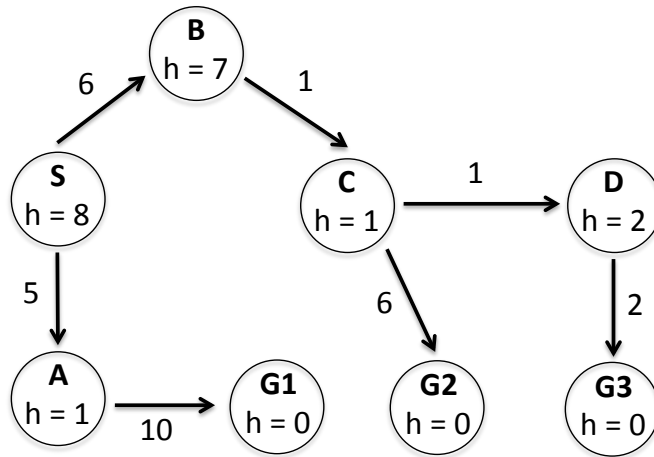
- (ii) [1 pt]  $\epsilon = 0$ .

Algorithm: UCS

- (iii) [1 pt]  $\epsilon \rightarrow \infty$  (i.e., as  $\epsilon$  becomes arbitrarily large).

Algorithm: Greedy search

(d) Here is the same graph again:



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

node	Goal?	fringe
-	-	$\{S : g = 0, f = 8\}$
S	No	$\{S \rightarrow A : g = 5, f = 6; S \rightarrow B : g = 6, f = 13\}$
S $\rightarrow$ A	No	$\{S \rightarrow B : g = 6, f = 13; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
S $\rightarrow$ B	No	$\{S \rightarrow B \rightarrow C : g = 7, f = 8; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
S $\rightarrow$ B $\rightarrow$ C	No	$\{S \rightarrow B \rightarrow C \rightarrow D : g = 8, f = 10; S \rightarrow B \rightarrow C \rightarrow G2 : g = 13, f = 13; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
S $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D	No	$\{S \rightarrow B \rightarrow C \rightarrow D \rightarrow G3 : g = 10, f = 10; S \rightarrow B \rightarrow C \rightarrow G2 : g = 13, f = 13; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
S $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D $\rightarrow$ G3	Yes	-

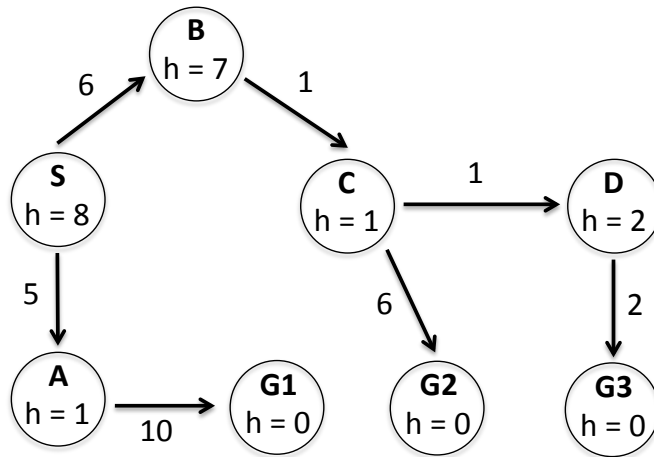
(ii) [4 pts] You'll notice that weighted A\* with  $\epsilon = 1$  repeats computations performed when run with  $\epsilon = 2$ . Is there a way to reuse the computations from the  $\epsilon = 2$  search by starting the  $\epsilon = 1$  search with a different fringe? Let  $F$  denote the set that consists of both (i) all nodes the fringe the  $\epsilon = 2$  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  $\epsilon$
- Use  $F$  with goal  $G$  removed as new starting fringe, updating the  $f$ -values to account for the new  $\epsilon$
- 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  $\epsilon$
- It is not possible to reuse computations, initialize the new starting fringe as usual

Justification:

We have to include  $G$  in the fringe as it might still be optimal (e.g. if it is the only goal). We don't have to update the  $g$ -values, but we do have to update the  $f$ -values to reflect the new value of  $\epsilon$ . With these modifications, it is valid to continue searching as the state of the fringe is as if A\* with the new  $\epsilon$  was run, but with some extraneous node expansions.

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.

<i>node</i>	Goal?	<i>fringe</i>
-	-	$\{S \rightarrow B : g = 6, f = 13; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
$S \rightarrow B$	No	$\{S \rightarrow B \rightarrow C : g = 7, f = 8; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
$S \rightarrow B \rightarrow C$	No	$\{S \rightarrow B \rightarrow C \rightarrow D : g = 8, f = 10; S \rightarrow B \rightarrow C \rightarrow G2 : g = 13, f = 13; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
$S \rightarrow B \rightarrow C \rightarrow D$	No	$\{S \rightarrow B \rightarrow C \rightarrow D \rightarrow G3 : g = 10, f = 10; S \rightarrow B \rightarrow C \rightarrow G2 : g = 13, f = 13; S \rightarrow A \rightarrow G1 : g = 15, f = 15\}$
$S \rightarrow B \rightarrow C \rightarrow D \rightarrow G3$	Yes	-

## 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  $\gamma$ . 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.

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

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

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

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

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

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

Justification:

Given a state-action pair  $(s, a)$ , the simulation will sample a next state according to  $T_{\text{sim}}(s, a, s')$ . However, that transition actually occurs as frequently as  $T_{\text{real}}(s, a, s')$ . To account for this bias during sampling, we will adjust the weight of the sample by dividing by the simulation probability. Specifically, each sample should be weighted by  $\frac{T_{\text{real}}(s, a, s')}{T_{\text{sim}}(s, a, s')}$ . In this way we use weights to adjust the Q-value function updates so that they are correct in expectation instead of sampling from the correct distribution directly.

(b) [2 pts] Now consider the case where we have  $n$  real robots with transition models  $T_{\text{real}}^1(s, a, s'), \dots, T_{\text{real}}^n(s, a, s')$  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)

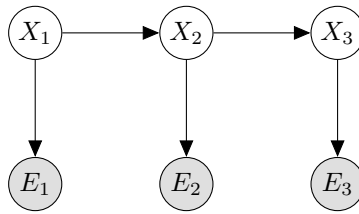
Yes       No

Justification:

Keep track of  $n$  Q-value functions. Given a new sample update each value function independently and according to the above update equation where the ratio incorporates the corresponding transition model.

### 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  $P(X_t|e_1, \dots, e_t)$ . The two steps of the algorithm are as follows:

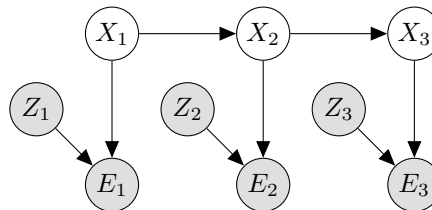
**Elapse Time**  $P(X_t|e_1, \dots, e_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_1, \dots, e_{t-1})$

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

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

$$P(X_t|e_1, \dots, e_{t-1}, z_1, \dots, z_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_{1:t-1}, z_{1:t-1})$$

(b) [3 pts] State the modified Observe update.

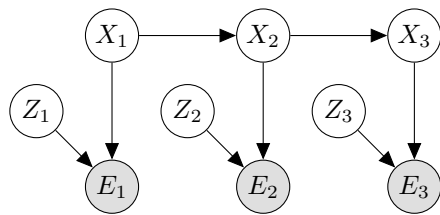
No Change

$$P(X_t|e_1, \dots, e_t, z_1, \dots, z_t) = \frac{P(X_t|e_{1:t-1}, z_{1:t-1})P(z_t)P(e_t|z_t, X_t)}{\sum_{x_t} P(x_t|e_{1:t-1}, z_{1:t-1})P(z_t)P(e_t|z_t, x_t)} = \frac{P(X_t|e_{1:t-1}, z_{1:t-1})P(e_t|z_t, X_t)}{\sum_{x_t} P(x_t|e_{1:t-1}, z_{1:t-1})P(e_t|z_t, x_t)}$$

Here we need to incorporate the new  $Z_i$  variables. Since they are observed, we can assume that variable  $Z_t$  has value  $z_t$ .



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



(c) [3 pts] State the modified Elapse Time update.

No Change

$$P(X_t|e_1, \dots, e_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_{1:t-1})$$

The  $Z_i$ 's don't effect the time update.

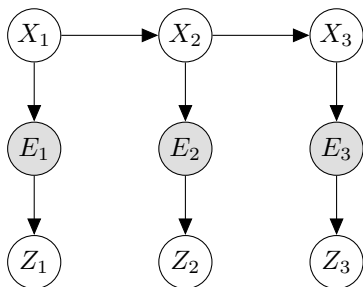
(d) [3 pts] State the modified Observe update.

No Change

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

Here we need to incorporate the new  $Z_i$  variables.

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

$$P(X_t|e_1, \dots, e_{t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_{1:t-1})$$

Nothing changes since  $Z_i$  are independent of  $X_i$  given  $E_i$ .

(f) [3 pts] State the modified Observe update.

No Change

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

Nothing changes since  $Z_i$  are independent of  $X_i$  given  $E_i$ .

## 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,  $x_1, x_2, \dots, x_n$  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  $\alpha_i$  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  $h_\alpha(z) \geq 0$ , and negative ( $-1$ ) otherwise.

Let the  $\alpha_i$  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  $j^{\text{th}}$  entry in the  $i^{\text{th}}$  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  $\alpha$ . ( $c_1$  has been completed for you.) Derive the inequalities by remembering the prediction rule  $h_\alpha(z)$  as the linear combinations of  $\alpha$  and  $K(x_i, x_j)$  then consulting the table of  $K(i, x_j)$  and the class labels. The inequalities are for  $\pm 1$  and not 0 to enforce a margin of 1.

$c_1$	$\alpha_1 - \alpha_4 \leq -1$	$c_3$	$-2\alpha_2 + \alpha_3 + \alpha_4 \geq 1$
$c_2$	$4\alpha_2 - 2\alpha_3 - 2\alpha_4 \leq -1$	$c_4$	$-\alpha_1 - 2\alpha_2 + \alpha_3 + 2\alpha_4 \geq 1$

- (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	3	$c_3$	Satisfied	Violated	0
$c_2$	Satisfied	Violated	-1	$c_4$	Satisfied	Violated	2

- (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  $\alpha'$  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 $c_i$	$\alpha'_1$	$\alpha'_2$	$\alpha'_3$	$\alpha'_4$	Updated CVM	Terminated?	
$c_1$	-1	-1	0	1	-1	Yes	No

First, you find which constraint has the highest CVM. That would be  $c_1$ , and you will perform min conflicts on the variables involved in  $c_1$ , which are  $a_1$  and  $a_4$ . That is, you pick new values for  $a_1$  and  $a_4$  as to minimize the CVM for constraint 1.

So right now, with  $a_1 = 1$  and  $a_4 = -1$ , we have  $a_1 a_4 = -1$ , so the CVM is 3. That one is the maximum, and so we assign  $a_1 = -1$  and  $a_4 = -1$ , then we have  $a_1 a_4 = 1$ , so the CVM is 0. We check if the algorithm

has terminated, and if you check the current alpha values against all the other weight vectors, you should find that all constraints are satisfied.

- (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	-4	-1
$z_2$	-2	1	2	-2	1	+1

# 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  $[x_1 \ x_2]^T$ )

**PBias** Linear perceptron with a bias term (features  $[1 \ x_1 \ x_2]^T$ )

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

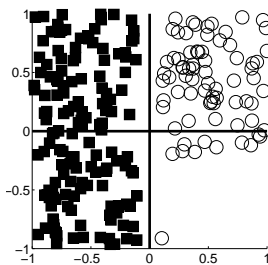
**PCutoff** Kernel perceptron with the kernel function  $K(x, z) = \max\{0, 0.01 - \|x - z\|_2\}$  ( $\|a - b\|_2$  is the Euclidean distance between  $a$  and  $b$ )

**1NN** 1-nearest neighbor classifier

**3NN** 3-nearest neighbor classifier

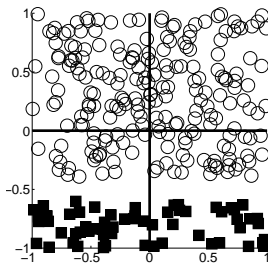
(a) [8 pts] In each of the plots below you are given points from two classes, shown as filled rectangles and open circles. For each plot, fill in the bubble next to each classifier that will be able to perfectly classify all of the training data (or, if none, mark “None of these will classify the data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

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



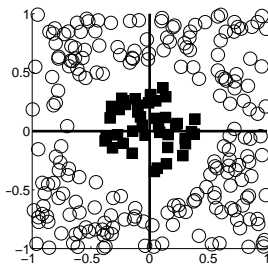
- PNoBias       PQuad       1NN
- PBias       PCutoff       3NN
- None of these will be able to classify the training data perfectly.

3NN misclassifies the bottom-most circle.



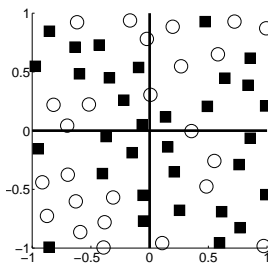
- PNoBias       PQuad       1NN
- PBias       PCutoff       3NN
- None of these will be able to classify the training data perfectly.

PNoBias is restricted to separators through the origin.



- PNoBias       PQuad       1NN
- PBias       PCutoff       3NN
- None of these will be able to classify the training data perfectly.

The data are not linearly separable.



- PNoBias       PQuad       1NN
- PBias       PCutoff       3NN
- None of these will be able to classify the training data perfectly.

The decision boundary is complicated and in particular neither linear, nor quadratic. 1NN and PCutoff classify locally.

- (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: **Either answer was accepted with justification. Add extra features – adding some really good features could better capture the structure in the data. Remove some features – the model may be using the noise in the abundant feature set to overfit to the training data rather than learning any meaningful underlying structure.**

- Collect more training data       Throw out some training data

**More data should yield a more representative sample of the true distribution of the data. Less data is more susceptible to overfitting.**

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)

**Increasing  $k$  reduces the impact of any one training instance to make the classifier less sensitive to overfitting of rare (= low count) patterns.**

Assuming your classifier is a Bayes' net:

- Add edges       Remove edges

**Removing edges reduces the class of distributions the Bayes' net can represent. Adding edges introduces more parameters so that the model could further overfit.**

- (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: **Under the current feature representation, we are unable to accurately model the training data for the purpose of the classification task we're interested in. The classifier may be able to deduce more information about the connections between data points and their classes from additional features, allowing it to better model the data for the classification task. For example, a linear perceptron could not accurately model two classes separated by a circle in a 2-dimensional feature space, but by using quadratic features in a kernel perceptron, we can find a perfect separating hyperplane.**

- Collect more training data       Throw out some training data

**More training data can only be a good thing. Marking neither of the bubbles was accepted, too, as given that train and hold-out validation already achieve the same performance, likely the underlying problem is not a lack of 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.

**Yes, this is a problem. The classifier is overfitting the training set. The fact that it had perfect accuracy with random labels suggests that it does not learn any real underlying structure in the data; it most likely essentially memorized each of the training cases.**

## 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:  $A \perp\!\!\!\perp D \mid C$  and  $B \perp\!\!\!\perp D \mid C$ . You must show work in order to receive credit.

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

There are many ways to solve this, here is one way.  $B \perp\!\!\!\perp D \mid C$  implies that:

$$P(+b \mid +d, +c) = P(+b \mid -d, +c)$$

Equivalently:

$$\frac{P(+b, +d, +c)}{P(+b, +d, +c) + P(-b, +d, +c)} = \frac{P(+b, -d, +c)}{P(+b, -d, +c) + P(-b, -d, +c)}$$

Computing those from the table above gives:

$$\frac{n + 18}{n + 18 + 3 + 9} = \frac{6 + 6}{6 + 6 + 1 + 3}$$

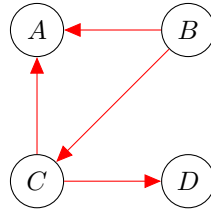
Equivalently:

$$(n + 18) * 15 = 12 * (n + 30)$$

Equivalently:

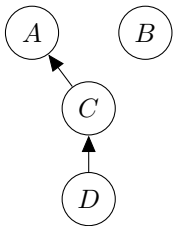
$n = 18$
----------

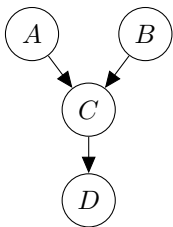
- (b) [3 pts] Draw a Bayes' net that makes exactly the 2 independence assumptions indicated by the set of samples shown above:  $A \perp\!\!\!\perp D | C$  and  $B \perp\!\!\!\perp D | C$ . Make sure it doesn't make any additional independence assumptions. There are multiple solutions. One example is below. Other solutions have the same connectivity but the direction of the edges may differ.

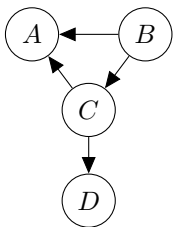


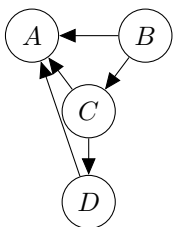
- (c) [6 pts] Now we run maximum likelihood learning of the parameters for each of the Bayes' nets below from the data on the previous page. The result is four learned distributions  $P^{(1)}(A, B, C, D)$ ,  $P^{(2)}(A, B, C, D)$ ,  $P^{(3)}(A, B, C, D)$ ,  $P^{(4)}(A, B, C, D)$ . We also define another 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  $A \perp\!\!\!\perp D | C$  and  $B \perp\!\!\!\perp D | C$ , but doesn't satisfy any other independence assumptions.

$P^{(1)}$     $P^{(0)}(B) = P^{(1)}(B)$    $P^{(0)}(D | C) = P^{(1)}(D | C)$    $P^{(0)}(A, B) = P^{(1)}(A, B)$

$P^{(2)}$     $P^{(0)}(B) = P^{(2)}(B)$    $P^{(0)}(D | C) = P^{(2)}(D | C)$    $P^{(0)}(A, B) = P^{(2)}(A, B)$

$P^{(3)}$     $P^{(0)}(B) = P^{(3)}(B)$    $P^{(0)}(D | C) = P^{(3)}(D | C)$    $P^{(0)}(A, B) = P^{(3)}(A, B)$

$P^{(4)}$     $P^{(0)}(B) = P^{(4)}(B)$    $P^{(0)}(D | C) = P^{(4)}(D | C)$    $P^{(0)}(A, B) = P^{(4)}(A, B)$

The key idea for this problem is to identify which independencies and conditional independencies are enforced by each of the candidate graphs. If the distribution we are considering ties in different dependencies, then they cannot be guaranteed to be the same when you maximize the likelihood of the data, because there are different constraints on the distribution, meaning that you are choosing from potentially a different set of distributions.



For  $P(B)$ , there are no conditional independence assumptions just involving  $B$  with no other variable, so we learn the same distribution from the data for all the Bayes' nets.

For  $P(D | C)$ , there are no conditional independence assumptions between  $C$  and  $D$  since there is a direct edge between the two. Hence, you learn the same distribution for all the Bayes' nets.

For  $P(A, B)$ , we see that in the first two BNs,  $A \perp\!\!\!\perp B$ , so when we compute  $P(A, B)$  for these two graphs, we compute  $P(A)P(B)$ . However, the other two BNs (and the given BN from part (b)) don't have that constraint, so we compute  $P(A, B) = P(A)P(B | A)$ . Because  $p^1$  and  $p^2$  make more assumptions than what we know the data reflects (which is what  $p^0$  represents), we can't guarantee that the results are equal. Hence, only the bottom two work for that one.

Here is an in-depth case study for  $P(B)$ :

Let's say we have the BN from the solution for part (b). We learn the parameters:

$P(+b)$ ,  $P(+c | +b)$ ,  $P(+c | -b)$ ,  $P(+a | +c, +b)$ ,  $P(+a | +c, -b)$ ,  $P(+a | -c, +b)$ ,  $P(+a | -c, -b)$ ,  $P(+d | +c)$ ,  $P(+d | -c)$ .

Note that we didn't have to learn the negative versions of the query variable since they are (1 - positive version)

Via the maximum likelihood equation:

$$\theta_{ML} = \arg \max_{\theta} \prod_i^m P_{\theta}(x_i)$$

.

The

$$P_{\theta}(x_i)$$

value we are considering is the joint of the BN, so for a data point say:  $+a, -b, +c, -d$ , the contribution it has on the likelihood is:

$$P(+a, -b, +c, -d) = (1 - P(+b)) * P(+c | -b) * P(+a | -b, +c) * (1 - P(+d | -c))$$

.

And we have these four terms for this samples. If we have  $m$  samples, we have  $4m$  terms in our likelihood product.

If we take the log of the product of the  $4m$  terms, we get the sum of the logs. Moreover, let's say we were interested in  $P(B)$  right now (which we are in the problem statement). Then, we want to differentiate the log likelihood in terms of  $P(+b)$ . It turns out that if you take the partial derivative with respect to  $P(+b)$ , all the non- $P(+b)$  involving terms disappear. Hence, we only have  $m$  terms remaining, which can be represented in the equation:

$$\log P(+b)^{n_{+b}} + \log(1 - P(+b))^{n_{-b}}$$

.

That is, the  $n_{+b}$  samples that had  $+b$  each contribute a  $\log P(+b)$  term, whereas the  $n_{-b}$  samples that had  $-b$  contribute a  $\log 1 - P(+b)$  term:

Hence:

$$n_{+b} \log P(+b) + n_{-b} \log(1 - P(+b))$$

.

Solving for the MLE by taking the derivative and setting this equal to 0, you get that:

$$P(+b) = \frac{n_{+b}}{n_{+b} + n_{-b}}$$

.

This should make sense. It's just the fraction of  $+b$  samples you had divided by all samples.

Now, if you had done this with, say,  $P^1$ , we would have learned:

$$P(+a \mid +c), P(+a \mid -c), P(+c \mid +d), P(+c \mid -d), P(+d), P(+b)$$

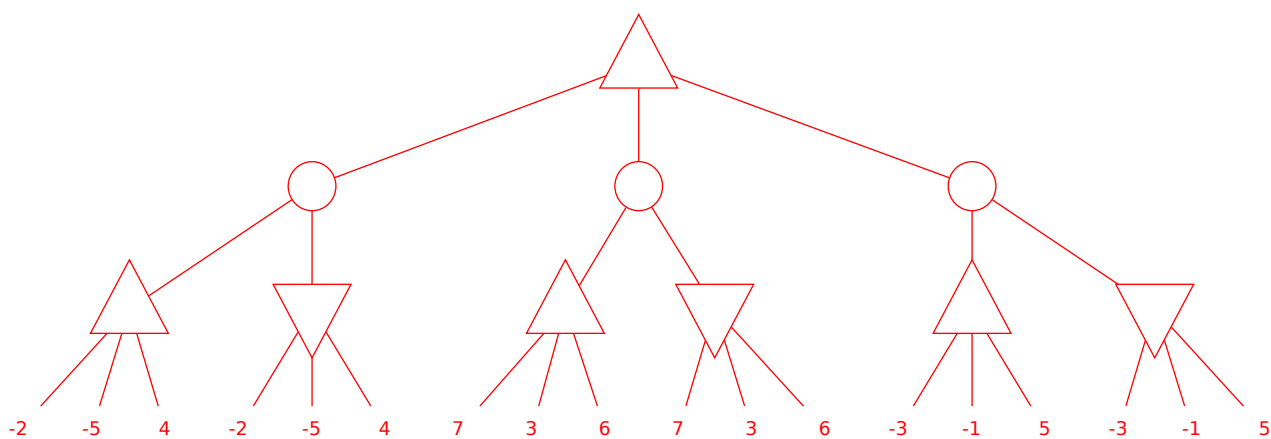
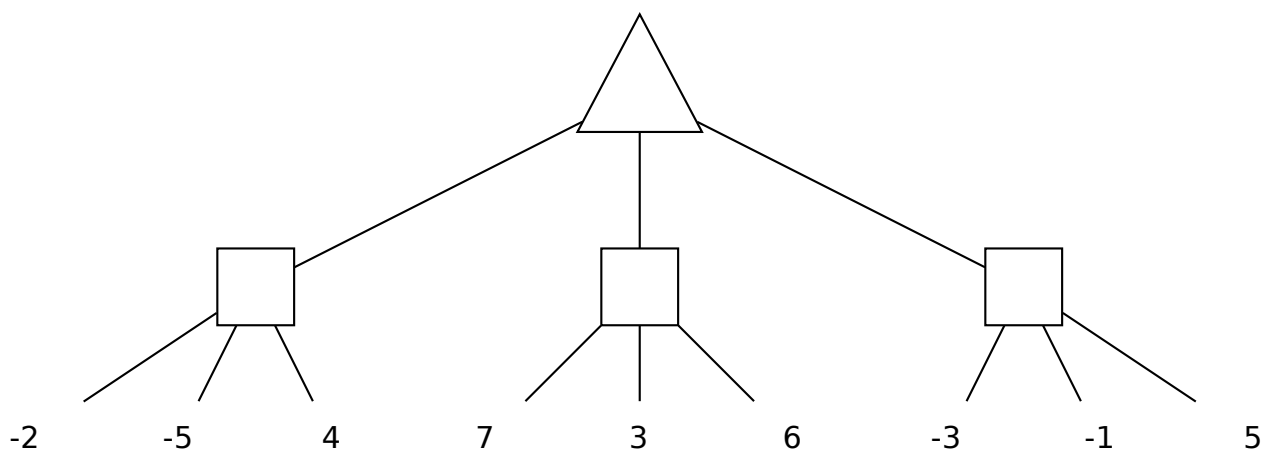
.  
If you follow the same process, you again have a product of  $4m$  terms for the likelihood. When you take the derivative with respect to  $P(+b)$  of the log likelihood, you end up with the same expression above (try it!). Hence, you learn the same distribution for  $P(B)$ .

This is the mathematical way of approaching it. The above explanation is the more intuitive way which just looks at dependencies that you consider in your query.

## 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,  $\alpha$ ,  $\beta$ )
  if node is leaf then
    return VALUE(node)
  (L, A, H)  $\leftarrow$  GET-LOWEST-AVG-HIGHEST(node)
  if (1)  then
    return H
  v  $\leftarrow$   $-\infty$ 
  for child  $\leftarrow$  CHILDREN(node) do
    v  $\leftarrow$  MAX(v, MIN-VALUE(child,  $\alpha$ ,  $\beta$ ))
    if v == H then
      return v
    if v  $\geq$   $\beta$  then
      return v
     $\alpha$   $\leftarrow$  MAX( $\alpha$ , 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,  $\alpha$ ,  $\beta$ )
  if node is leaf then
    return VALUE(node)
  (L, A, H)  $\leftarrow$  GET-LOWEST-AVG-HIGHEST(node)
  if (2)  then
    return L
  v  $\leftarrow$   $\infty$ 
  for child  $\leftarrow$  CHILDREN(node) do
    v  $\leftarrow$  MIN(v, MAX-VALUE(child,  $\alpha$ ,  $\beta$ ))
    if v == L then
      return v
    if v  $\leq$   $\alpha$  then
      return v
     $\beta$   $\leftarrow$  MIN( $\beta$ , v)
  return v

```

(2)

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

We can prune on the OR of the marked conditions. Marking just one of the conditions was sufficient for full credit. The average value  $A$  is unhelpful for pruning because the expectation does not bound the extreme values.

Here is a detailed explanation. Recall the definition of  $\alpha$  and  $\beta$  in the alpha-beta pruning algorithm.

$\alpha$  is the best value that MAX has seen so far somewhere along the path from the current node to the root of the tree.  $\beta$  is the best value that MIN has seen so far somewhere along the path from the current node to the root of the tree.

With that, we consider the problem, where Pacman has a magic power that lets him evaluate the lowest, average, and highest values of the leaves under that node prior to exploring that node. This could potentially allow us to stop exploring the tree early, but on what conditions can we prune?

For the follow examples, Pacman = MAX, Ghost = MIN.

Let's consider a small example for Pacman. Pacman is currently on one of his nodes. The magic powers awaken and tell Pacman that the  $(L,A,H)$  reading for the node is  $(2, 5, 10)$ . Let's imagine that the  $\alpha$  value is 12, which means that somewhere along the path to the root, Pacman can guarantee a score of 12. This means that Pacman would prefer that path with 12 over the current path (since at most, we can guaranteed  $H = 10$ ), so we don't need to explore this node any further. Generalizing this, the condition is that  $10 < 12$ , which corresponds to  $H < \alpha$ .

Let's consider another case of the same setup, where the Pacman is on his node and the  $(L, A, H)$  reading is  $(2, 5, 10)$ . Now, Pacman ponders about what the Ghost is up to. Let's assume that the  $\beta$  value is 0, which means that somewhere along path to the root, the ghost can guarantee a score of 0. This means that the ghost would prefer that path with 0 over the current path (since the ghost wants to minimize, so it doesn't like this current path where the best it can get is 2). Which means we don't need to explore this node any further. Generalizing this, the condition is that  $2 > 0$ , which corresponds to  $L > \beta$ .

We can prune on either of these conditions (and you can identically solve the mirror problem for MIN – it turns out that the same conditions holds for MIN). The average value does not help, because the average could be skewed.

THIS PAGE IS INTENTIONALLY LEFT BLANK