CS 188 Spring 2012 Introduction to Artificial Intelligence Final

- You have approximately 3 hours.
- The exam is closed book, closed notes except a one-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	
Login	

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

For staff use only:

Q1. All Topics: Short Questions	/88
Q2. Variable Elimination Ordering	/28
Q3. Bidirectional A* Search	/14
Q4. 3-Player Games	/8
Q5. Learning a Bayes' Net Structure	/12
Total	/150

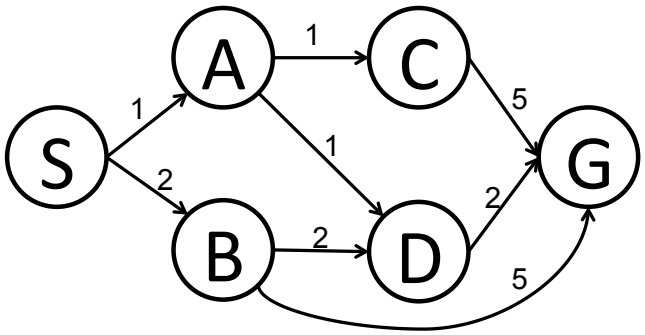
THIS PAGE IS INTENTIONALLY LEFT BLANK

Q1. [88 pts] All Topics: Short Questions

Each True/False question is worth 2 points. Leaving a question blank is worth 0 points. **Answering incorrectly is worth −2 points.**

For the questions that are not True/False, justify your answer concisely.

(a) Search.



Answer the following questions about the search problem shown above. S is the start-state, G is the (only) goal-state. Break any ties alphabetically. For the questions that ask for a path, please give your answers in the form '$S - A - D - G$.'

(i) [1 pt] What path would breadth-first graph search return for this search problem?

$S - B - G$

(ii) [1 pt] What path would uniform cost graph search return for this search problem?

$S - A - D - G$

(iii) [1 pt] What path would depth-first graph search return for this search problem?

$S - A - C - G$

(iv) [1 pt] What path would A* graph search, using a consistent heuristic, return for this search problem?

$S - A - D - G$

(b) CSPs.

(i) [4 pts] **CSP Formulation.**

PacStudent (S), PacBaby (B), PacMom (M), PacDad (D), GrandPac (P), and a friendly Ghost (G) are lining up next to each other. The positions are numbered 1, 2, 3, 4, 5, 6, where 1 neighbors 2, 2 neighbors 1 and 3, 3 neighbors 2 and 4, 4 neighbors 3 and 5, 5 neighbors 4 and 6, and 6 neighbors 5. Each one of them takes up exactly one spot. PacBaby (B) needs to be next to PacMom (M) one one side and PacDad (D) on the other side. GrandPac (P) needs to be next to the Ghost (G). PacStudent (S) needs to be at 1 or 2. Formulate this problem as a CSP: list the variables, their domains, and the constraints. Encode unary constraints as a constraint rather than pruning the domain. (No need to solve the problem, just provide variables, domains and implicit constraints.)

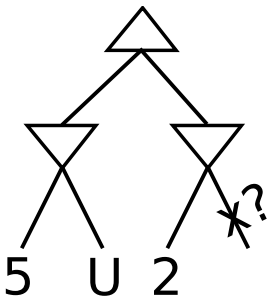
- Variables: S, B, M, D, P, G
- Domains: $\{1, 2, 3, 4, 5, 6\}$ for all variables
- Constraints: $\text{alldiff}(S, B, M, D, P, G), |B - M| = 1, |B - P| = 1, |P - G| = 1, S \in \{1, 2\}$.

(ii) [2 pts] Consider a CSP with variables X, Y with domains $\{1, 2, 3, 4, 5, 6\}$ for X and $\{2, 4, 6\}$ for Y, and constraints $X < Y$ and $X + Y > 8$. List the values that will remain in the domain of X after enforcing arc consistency for the arc $X \to Y$ (recall arc consistency for a specific arc only prunes the domain of the tail variable, in this case X).

The resulting domain of X is $\{3, 4, 5\}$.

(c) [2 pts] $\alpha - \beta$ **Pruning.**

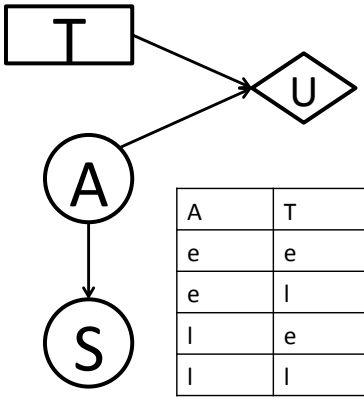
Consider the game tree shown below. For what range of U will the indicated pruning take place?



Two answers are ok: if pruning with equality then the answer is $U \geq 2$; if pruning with inequality then the answer is $U > 2$.

(d) [16 pts] **Probability and Decision Networks.**

A	P(A)
e	0.5
l	0.5



S	P(S)
e	0.6
l	0.4

A	S	P(S|A)
e	e	0.8
e	l	0.2
l	e	0.4
l	l	0.6

A	T	U(A,T)
e	e	600
e	l	0
l	e	300
l	l	600

S	A	P(A|S)
e	e	2/3
e	l	1/3
l	e	1/4
l	l	3/4

Your parents are visiting you for graduation. You are in charge of picking them up at the airport. Their arrival time (A) might be early (e) or late (l). You decide on a time (T) to go to the airport, also either early (e) or late (l). Your sister (S) is a noisy source of information about their arrival time. The probability values and utilities are shown in the tables above.

Compute $P(S), P(A|S)$ and compute the quantities below.

$EU(T=e) = P(A=e)U(A=e,T=e) + P(A=l)U(A=l,T=e) = 0.5*600 + 0.5*300 = 450$

$EU(T=l) = P(A=e)U(A=e,T=l) + P(A=l)U(A=l,T=l) = 0.5*0 + 0.5*600 = 300$

$MEU(\{\}) = 450$

Optimal action with no observations is $T = e$

Now we consider the case where you decide to ask your sister for input.

$EU(T=e|S=e) = P(A=e|S=e)U(A=e,T=e) + P(A=l|S=e)U(A=l,T=e) = \frac{2}{3}600 + \frac{1}{3}300 = 500$

$EU(T=l|S=e) = P(A=E|S=e)U(A=e,T=l) + P(A=l|S=e)U(A=l,T=l) = \frac{2}{3}0 + \frac{1}{3}600 = 200$

$MEU(\{S=e\}) = 500$

Optimal action with observation $\{S=e\}$ is $T = e$

$EU(T=e|S=l) = P(A=e|S=l)U(A=e,T=e) + P(A=l|S=e)U(A=l,T=e) = \frac{1}{4}600 + \frac{3}{4}300 = 375$

$EU(T=l|S=l) = P(A=e|S=l)U(A=e,T=l) + P(A=l|S=l)U(A=l,T=l) = \frac{1}{4}0 + \frac{3}{4}*600 = 450$

$MEU(\{S=l\}) = 450$

Optimal action with observation $S=l$ is $T = l$

$VPI(S) = P(S=e)MEU(\{S=e\}) + P(S=l)MEU(\{S=l\}) - MEU(\{\}) = 0.6*500 + 0.4*450 - 450 = 30$

(e) MDPs.

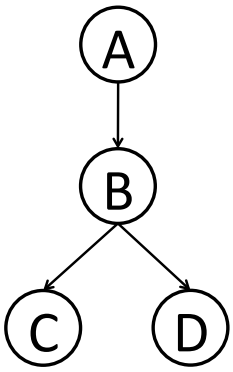
(i) [2 pts] [*true* or *false*] If the only difference between two MDPs is the value of the discount factor then they must have the same optimal policy.
A counterexample suffices to show the statement is false. Consider an MDP with two sink states. Transitioning into sink state A gives a reward of 1, transitioning into sink state B gives a reward of 10. All other transitions have zero rewards. Let A be one step North from the start state. Let B be two steps South from the start state. Assume actions always succeed. Then if the discount factor $\gamma < 0.1$ the optimal policy takes the agent one step North from the start state into A, if the discount factor $\gamma > 0.1$ the optimal policy takes the agent two steps South from the start state into B.

(ii) [2 pts] [*true* or *false*] When using features to represent the Q-function it is guaranteed that this feature-based Q-learning finds the same Q-function, Q^*, as would be found when using a tabular representation for the Q-function.
Whenever the optimal Q-function, Q^*, cannot be represented as a weighted combination of features, then the feature-based representation would not even have the expressiveness to find the optimal Q-function, Q^*.

(iii) [2 pts] [*true* or *false*] For an infinite horizon MDP with a finite number of states and actions and with a discount factor γ, with $0 < \gamma < 1$, value iteration is guaranteed to converge.
See lecture slides.

(iv) [2 pts] [*true* or *false*] When getting to act only for a finite number of steps in an MDP, the optimal policy is stationary. (A stationary policy is a policy that takes the same action in a given state, independent of at what time the agent is in that state.)
See lecture slides.

(f) [3 pts] **Bayes' Nets: Representation** Consider the joint distribution $P(A, B, C, D)$ defined by the Bayes' net below.



A	P(A)
+a	0.8
-a	0.2

A	B	P(B|A)
+a	+b	0.8
+a	-b	0.2
-a	+b	0.5
-a	-b	0.5

B	C	P(C|B)
+b	+c	0.8
+b	-c	0.2
-b	+c	0.5
-b	-c	0.5

B	D	P(D|B)
+b	+d	0.7
+b	-d	0.3
-b	+d	0.1
-b	-d	0.9

Compute the following quantities:

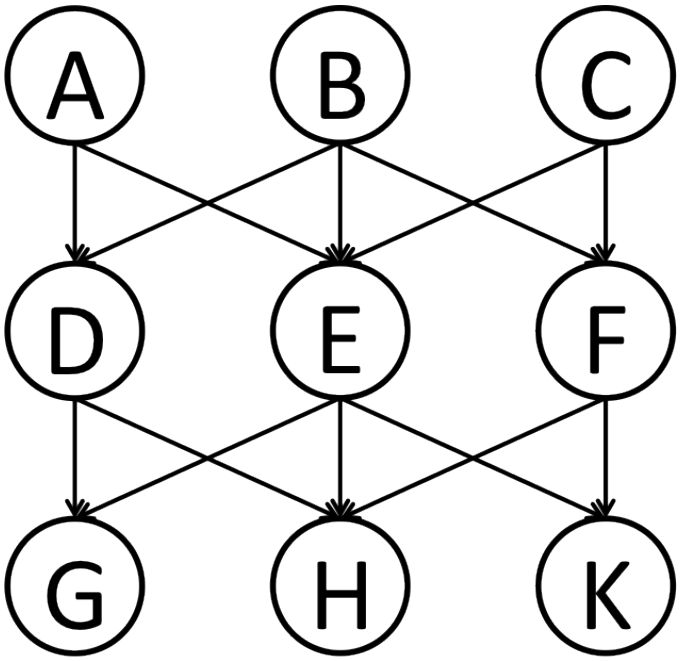
$P(A = +a) = 0.8$

$P(A = +a, B = -b, C = -c, D = +d) = 0.8 * 0.2 * 0.1 * 0.5 = 0.008$

$P(A = +a | B = -b, C = -c, D = +d) = \frac{0.8*0.2*0.1*0.5}{0.8*0.2*0.1*0.5+0.2*0.5*0.1*0.5} = 0.615$

(g) [8 pts] **Bayes' Nets: Conditional Independence**

Based only on the structure of the (new) Bayes' Net given below, circle whether the following conditional independence assertions are guaranteed to be true, guaranteed to be false, or cannot be determined by the structure alone.Note: The ordering of the three answer columns might have been switched relative to previous exams!



1	$A \perp\!\!\!\perp C$	Guaranteed false	Cannot be determined	Guaranteed true
2	$A \perp\!\!\!\perp C \mid E$	Guaranteed false	Cannot be determined	Guaranteed true
3	$A \perp\!\!\!\perp C \mid G$	Guaranteed false	Cannot be determined	Guaranteed true
4	$A \perp\!\!\!\perp K$	Guaranteed false	Cannot be determined	Guaranteed true
5	$A \perp\!\!\!\perp G \mid D, E, F$	Guaranteed false	Cannot be determined	Guaranteed true
6	$A \perp\!\!\!\perp B \mid D, E, F$	Guaranteed false	Cannot be determined	Guaranteed true
7	$A \perp\!\!\!\perp C \mid D, F, K$	Guaranteed false	Cannot be determined	Guaranteed true
8	$A \perp\!\!\!\perp G \mid D$	Guaranteed false	Cannot be determined	Guaranteed true

(h) Bayes' Nets: Elimination of a Single Variable

Assume we are running variable elimination, and we currently have the following three factors:

A	B	$f_1(A,B)$
$+a$	$+b$	0.1
$+a$	$-b$	0.5
$-a$	$+b$	0.2
$-a$	$-b$	0.5

A	C	D	$f_2(A,C,D)$
$+a$	$+c$	$+d$	0.2
$+a$	$+c$	$-d$	0.1
$+a$	$-c$	$+d$	0.5
$+a$	$-c$	$-d$	0.1
$-a$	$+c$	$+d$	0.5
$-a$	$+c$	$-d$	0.2
$-a$	$-c$	$+d$	0.5
$-a$	$-c$	$-d$	0.2

B	D	$f_3(B,D)$
$+b$	$+d$	0.2
$+b$	$-d$	0.2
$-b$	$+d$	0.5
$-b$	$-d$	0.1

The next step in the variable elimination is to eliminate B.

(i) [3 pts] Which factors will participate in the elimination process of B? f_1, f_3

(ii) [4 pts] Perform the join over the factors that participate in the elimination of B. Your answer should be a table similar to the tables above, it is your job to figure out which variables participate and what the numerical entries are.

A	B	D	$f_4'(A,B,D)$
$+a$	$+b$	$+d$	0.1*0.2 = 0.02
$+a$	$+b$	$-d$	0.1*0.2 = 0.02
$+a$	$-b$	$+d$	0.5*0.5 = 0.25
$+a$	$-b$	$-d$	0.5*0.1 = 0.05
$-a$	$+b$	$+d$	0.2*0.2 = 0.04
$-a$	$+b$	$-d$	0.2*0.2 = 0.04
$-a$	$-b$	$+d$	0.5*0.5 = 0.25
$-a$	$-b$	$-d$	0.5*0.1 = 0.05

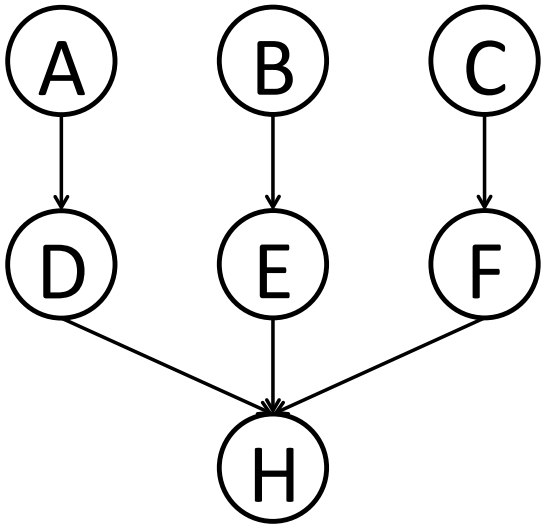
(iii) [4 pts] Perform the summation over B for the factor you obtained from the join. Your answer should be a table similar to the tables above, it is your job to figure out which variables participate and what the numerical entries are.

A	D	$f_4(A,D)$
$+a$	$+d$	0.02+ 0.25 = 0.27
$+a$	$-d$	0.02 + 0.05 = 0.07
$-a$	$+d$	0.04 + 0.25 = 0.29
$-a$	$-d$	0.04 + 0.05 = 0.09

(i) Elimination Sequence

For the Bayes' net shown below, consider the query $P(A|H = +h)$, and the variable elimination ordering B, E, C, F, D.

(i) [4 pts] In the table below fill in the factor generated at each step — we did the first row for you.



Variable Eliminated	Factor Generated	Current Factors
(no variable eliminated yet)	(no factor generated)	$P(A), P(B), P(C), P(D\|A), P(E\|B), P(F\|C), P(+h\|D, E, F)$
B	$f_1(E)$	$P(A), P(C), P(D\|A), P(F\|C), P(+h\|D, E, F), f_1(E)$
E	$f_2(+h, D, F)$	$P(A), P(C), P(D\|A), P(F\|C), f_2(+h, D, F)$
C	$f_3(F)$	$P(A), P(D\|A), f_2(+h, D, F), f_3(F)$
F	$f_4(+h, D)$	$P(A), P(D\|A), f_4(+h, D)$
D	$f_5(+h, A)$	$P(A), f_5(+h, A)$

(ii) [2 pts] Which is the largest factor generated? Assuming all variables have binary-valued domains, how many entries does the corresponding table have? $f_2(+h, D, F)$, its table has $2^2 = 4$ entries

(j) Sampling

(i) [2 pts] Consider the query $P(A|-b, -c)$. After rejection sampling we end up with the following four samples: $(+a, -b, -c, +d), (+a, -b, -c, -d), (+a, -b, -c, -d), (-a, -b, -c, -d)$. What is the resulting estimate of $P(+a|-b, -c)$?

$\frac{3}{4}$.

(ii) [2 pts] Consider again the query $P(A|-b, -c)$. After likelihood weighting sampling we end up with the following four samples: $(+a, -b, -c, -d), (+a, -b, -c, -d), (-a, -b, -c, -d), (-a, -b, -c, +d)$, and respective weights: $0.1, 0.1, 0.3, 0.3$. What is the resulting estimate of $P(+a|-b, -c)$?

$\frac{0.1+0.1}{0.1+0.1+0.3+0.3} = \frac{0.2}{0.8} = \frac{1}{4}$

(k) Maximum Likelihood

(i) [4 pts] **Geometric Distribution**

Consider the geometric distribution, which has $P(X = k) = (1-\theta)^{k-1}\theta$. Assume in our training data X took on the values 4, 2, 7, and 9.

(a) Write an expression for the log-likelihood of the data as a function of the parameter θ.

$$\begin{aligned} L(\theta) &= P(X=4)P(X=2)P(X=7)P(X=9) = (1-\theta)^3\theta(1-\theta)^1\theta(1-\theta)^6\theta(1-\theta)^8\theta = (1-\theta)^{18}\theta^4 \\ \log L(\theta) &= 18\log(1-\theta) + 4\log\theta \end{aligned}$$

(b) What is the value of θ that maximizes the log-likelihood, i.e., what is the maximum likelihood estimate for θ?

At the maximum we have: $\frac{\partial \log L(\theta)}{\partial \theta} = 18\frac{-1}{1-\theta} + 4\frac{1}{\theta} = 0$

After multiplying with $(1-\theta)\theta$, $-18\theta + 4(1-\theta) = 0$ and hence we have an extremum at $\theta = \frac{4}{22}$

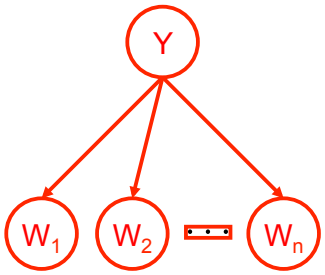
Also: $\frac{\partial^2 \log L(\theta)}{\partial \theta^2} = \frac{-18}{(1-\theta)^2} - \frac{4}{\theta^2} < 0$ hence extremum is indeed a maximum, and hence $\theta^{\text{ML}} = \frac{4}{22}$.

(ii) [6 pts] Consider the Bayes' net consisting of just two variables A, B, and structure $A \to B$. Find the maximum likelihood estimates and the $k = 2$ Laplace estimates for each of the table entries based on the following data: $(+a, -b), (+a, +b), (+a, -b), (-a, -b), (-a, -b)$.

A	$P^{\text{ML}}(A)$	$P^{\text{Laplace, k=2}}(A)$
$+a$	$\frac{3}{5}$	$\frac{5}{9}$
$-a$	$\frac{2}{5}$	$\frac{4}{9}$

A	B	$P^{\text{ML}}(\mathbf{B} \mid \mathbf{A})$	$P^{\text{Laplace, k=2}}(\mathbf{B} \mid \mathbf{A})$
$+a$	$+b$	$\frac{1}{3}$	$\frac{3}{7}$
$+a$	$-b$	$\frac{2}{3}$	$\frac{4}{7}$
$-a$	$+b$	0	$\frac{2}{6}$
$-a$	$-b$	1	$\frac{4}{6}$

(l) [5 pts] **Naive Bayes** Describe the naive Bayes bag-of-words model for document classification. Draw the Bayes net graph, annotate the class label node and the feature nodes, describe what the domain of the features is, describe any properties of the conditional probability tables that are specific to the bag-of-words (and not necessarily true in all naive Bayes models). For simplicity it is OK to assume that every document in consideration has exactly N words and that words come from a dictionary of D words.



The naive Bayes bag-of-words model is a naive Bayes model with a particular choice of features: there is a feature for each word in the document. Its value is the word at that position in the document. What we described so far this is just a naive Bayes model. The bag-of-words aspect lies in assuming that the conditional probability tables are the same for each feature—i.e., it assumes that where a word appears in a document is irrelevant. The Bayes net graph is shown in the figure above, Y is the class label, $W_1, \ldots, W_n$ are the features.

(m) Perceptron

Consider a multi-class perceptron with current weight vectors $w_A = (1, 2, 3), w_B = (-1, 0, 2), w_C = (0, -2, 1)$. A new training example is considered, which has feature vector $f(x) = (1, -3, 1)$ and label $y^* = B$.

(i) [2 pts]

Which class y would be predicted by the current weight vectors?

$w_A \cdot f(x) = -2,\ w_B \cdot f(x) = 1,\ w_C \cdot f(x) = 7$, hence the predicted class $y = C$.

(ii) [3 pts]

Would the perceptron update the weight vectors after having seen this training example? If yes, write the resulting weight vectors below:

$w_A = (1, 2, 3)$

$w_B = (-1, 0, 2) + f(x) = (-1, 0, 2) + (1, -3, 1) = (0, -3, 3)$

$w_C = (0, -2, 1) - f(x) = (0, -2, 1) - (1, -3, 1) = (-1, 1, 0)$

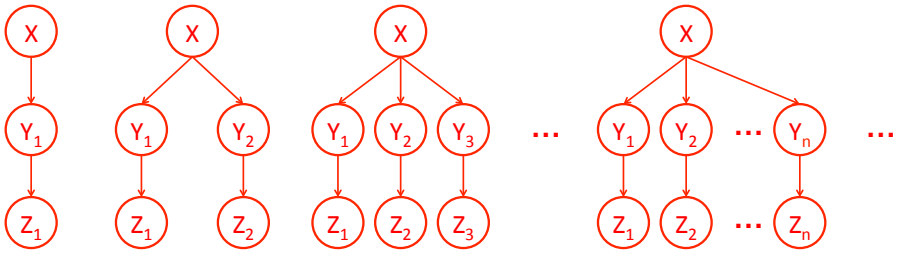
Q2. [28 pts] Variable Elimination Ordering

Assume all random variables are binary valued.

(a) [8 pts] **The Ordering Matters.** Consider the sequence of graphs below. For each, regardless of the elimination ordering, the largest factor produced in finding $p(X)$ will have a table with 2^2 entries.

Now draw a sequence of graphs such that, if you used the best elimination ordering for each graph, the largest factor table produced in variable elimination would have a constant number of entries, but if you used the worst elimination ordering for each graph, the number of entries in the largest factor table would grow exponentially as you move down the sequence. Provide (i) the sequence of graphs, (ii) the sequence of queries for which variable elimination is done, (iii) the best ordering, (iv) the worst ordering.

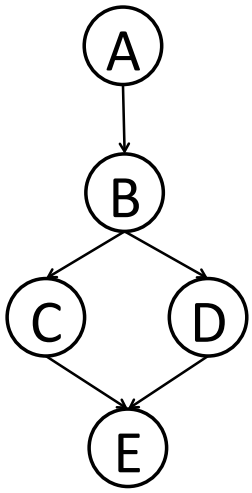
There are many solutions, here is one family of Bayes' net graphs that will do, with query $P(Y_n|Z_1,\ldots,Z_n)$, a best ordering: $Y_1, Y_2, \ldots, Y_{n-1}, X$, a worst ordering: $X, Y_1, Y_2, \ldots, Y_{n-1}$.

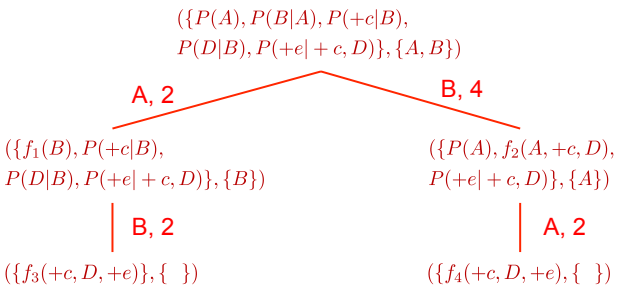


(b) Search Space Formulation for Finding an Ordering. Having established that ordering matters, let's investigate search methods that can find a good elimination ordering. The idea is to step through the process of variable elimination for various orderings of elimination of the hidden variables—and while doing so, only keep track of (i) which factors are present in each step, and (ii) for each factor which variables participate; but not actually compute and store the tables corresponding to each factor. (It is the join and the summation that are the expensive steps in variable elimination—computing which variables would participate in the new factor formed after the join and summation is relatively cheap.) We will use the following search-space formulation. We assume the hidden variables are called $H_1, H_2, \ldots, H_n$, and that all variables are binary.

- set of states S: a state s consists of the current set of factors, including the variables participating in each factor but not the corresponding tables, and any subset of $\{H_1, H_2, \ldots, H_n\}$ to track which variables yet have to be eliminated.
- successor function: choose any of the not yet eliminated variables, and update factors and list of not yet eliminated variables to account for the new elimination.
- cost function: the number of entries in the table representation of the new factor that is generated from the elimination of the current variable
- goal test: test whether the set of not yet eliminated hidden variables is empty.
- start state: set of conditional probability tables and set of all hidden variables.

(i) [4 pts] **Complete Search Tree.** Consider the query $P(D|+e,+c)$. Draw the complete search tree for this problem. Annotate nodes with states, and annotate costs and actions on the edges. Hint: the start state is $(\{P(A), P(B|A), P(+c|B), P(D|B), P(+e|+c, D)\}, \{A, B\})$.





(ii) [4 pts] **Solving the Search for the Example Problem.**

(a) Clearly mark all optimal plans in the search tree above.
There is one optimal plan: the left branch, which first eliminates A and then B.

(b) What is the cost of an optimal plan to a goal state?
The cost of the single optimal plan to the goal state equals $2 + 2 = 4$.

(iii) [12 pts] **Questions about this Search Formulation in General.**

For each of the following heuristics state whether they are admissible or not. Justify your answer. (No credit if there is no justification.) Notation: $\mathcal{H}$ is the set of hidden variables not yet eliminated. $\mathcal{Q}$ is the set of query variables. $\#\mathcal{H}$ is the number of hidden variables not yet eliminated. $\#\mathcal{Q}$ is the number of query variables. Again we assume that all variables are binary-valued.

(a) h_1: $\max_{H_i \in \mathcal{H}}\{$ size of factor generated when eliminating H_i next $\}$

Admissible **Not Admissible**
The example in part (a) shows that it is possible to have an optimal cost that grows linearly in the number of hidden variables, whereas the maximum size of factor generated (by choosing X first in our example for (a) is exponential in the number of hidden variables.

(b) h_2: $\min_{H_i \in \mathcal{H}}\{$ size of factor generated when eliminating H_i next $\}$

[Admissible] Not Admissible

You have to eliminate one of the hidden variables next, so the cost of the cheapest one to eliminate is a lower bound on the total cost still to be incurred.

(c) h_3: $2^{\#\mathcal{H}-1}$

Admissible [Not Admissible]

The example in part (a) shows that it is possible to have an optimal cost that grows linearly in the number of hidden variables, rather than exponential.

(d) h_4: if the current largest factor is of size 2^k and $k > \#Q$, then $2^{k-1} + 2^{k-2} + \ldots 2^{\#Q}$; otherwise, 0.

[Admissible] Not Admissible

When finished with elimination all factors left can have at most $\#Q$ variables in them, hence the elimination process will need to reduce a factor of size 2^k for $k > \#Q$ down to a factor of size $2^{\#Q}$, eliminating one variable at a time.

Q3. [14 pts] Bidirectional A* Search

If a search problem has only a single goal state, it is common to perform bidirectional search. In bidirectional search you build two search trees at the same time: the "forward" search tree is the one we have always worked with in CS188, the "backward" search tree is one that starts from the goal state, and calls a predecessor (rather than successor) function to work its way back to the start state. Both searches use the same cost function for transitioning between two states. There will now also be a backward heuristic, which for each state estimates the distance to the start state. Bidirectional search can result in significant computational advantages: the size of the search tree built grows exponentially with the depth of the search tree. If growing a tree from start and goal to each other, these two trees could meet in the middle, and one ends up with a computational complexity of just twice searching a tree of half the depth, which are very significant savings.

Recall the pseudo-code for a standard A* graph search

```
function Graph-Search(problem)

    forward-closed   <-- empty set
    forward-priority-queue  <-- Insert(Make-Node(Start-State(problem)), forward-priority-queue)

    LOOP DO
        IF forward-priority-queue is empty THEN return failure

        IF forward-priority-queue is not empty THEN
            node <-- pop(forward-priority-queue)
            IF (State(node) == Goal-State(problem) ) THEN return node

        IF State(node) is not in forward-closed THEN
            add State(node) to forward-closed
            forward-priority-queue <-- Insert-All(ExpandForward(node, problem), forward-priority-queue)

    END // LOOP
```

Now consider the following tentative pseudo-code for bidirectional A* search. We assume a consistent forward heuristic, and a consistent backward heuristic. Concatenation is a function that builds a plan that goes from start state to goal state by combining a forward partial plan and a backward partial plan that end in the same state.

```
function Bidirectional-Graph-Search(problem)

    forward-closed   <-- empty set
    backward-closed  <-- empty set
    forward-priority-queue  <-- Insert(Make-Node(Start-State(problem)), forward-priority-queue)
    backward-priority-queue <-- Insert(Make-Node(Goal-State(problem)), backward-priority-queue)

    LOOP DO
        IF forward-priority-queue is empty AND backward-priority-queue is empty THEN return failure

1       IF there exist a node n1 in forward-priority-queue and a node n2 in backward priority queue ...
1          such that State(n1) == State(n2) THEN
1               return Concatenation of n1 and n2

        IF forward-priority-queue is not empty THEN
            node <-- pop(forward-priority-queue)
            IF ( State(node) == Goal-State(problem) ) THEN return node

2           IF ( State(node) is in backward-priority-queue ) THEN
2               return Concatenation of node and matching node in backward-priority-queue

3           IF ( State(node) is in backward-closed ) THEN
3               return Concatenation of node and matching node in backward-closed

        IF State(node) is not in forward-closed THEN
            add State(node) to forward-closed
            forward-priority-queue <-- Insert-All(ExpandForward(node, problem), forward-priority-queue)

        IF backward-priority-queue is not empty THEN
            node <-- pop(backward-priority-queue)
            IF ( State(node) == Start-State(problem) ) THEN return node

4           IF ( State(node) is in forward-priority-queue ) THEN
4               return Concatenation of node and matching node in forward-priority-queue

5           IF ( State(node) is in forward-closed ) THEN
5               return Concatenation of node and matching node in forward-closed

        IF State(node) is not in backward-closed THEN
            add State(node) to backward-closed
            backward-priority-queue <-- Insert-All(ExpandBackward(node, problem), backward-priority-queue)

    END // LOOP
```

(a) The IF statements labeled 1, 2, 3, 4, 5 are modifications to try to connect both search trees.

(i) [2 pts] If cutting out all lines of code labeled 1, 2, 3, 4, or 5, will Bidirectional-Graph-Search return an optimal solution? Briefly justify your answer.

Yes. It will simply run two A^* searches in parallel, without any interaction. One from start to goal, and one from goal to start. Each one of them individually would return an optimal solution, and in case of running them in parallel it will be whichever one finishes first that will be returning an optimal solution.

(ii) [2 pts] If amongst the numbered lines of code we only retain 1, is Bidirectional-Graph-Search guaranteed to be optimal? Briefly justify your answer.

No. Consider the following example where it would not find the optimal solution: Have a state space graph with states S, A, B, C, G, with edges and respective costs: $S-A$: 1; $S-B$: 5, $A-C$: 1, $C-G$:1, $B-G$: 5. Assume the heuristic is zero everywhere. Enabling code section 1 will result in finding the path $S-B-G$ with cost 10, whereas the optimal path is $S-A-C-G$ with cost 2.

(iii) [2 pts] If amongst the numbered lines of code we only retain 2, is Bidirectional-Graph-Search guaranteed to be optimal? Briefly justify your answer.

No. Consider the same example as in the previous subquestion. Enabling code section 2 will result in finding the path $S-B-G$, which is suboptimal.

(iv) [2 pts] If amongst the numbered lines of code we only retain 3, is Bidirectional-Graph-Search guaranteed to be optimal? Briefly justify your answer.

No. Consider the following state space graph: states are S, A, G, edges and costs are $S-A$: 4, $A-G$: 4, $S-B$: 3, $B-C$: 3, $C-G$: 3. Assume the heuristics are: A to G: 4; A to S: 4, B to G: 3, B to S: 0, C to G: 0, C to S: 3. In the first iteration we will expand S and G on their respective priority queues. In the second iteration we will expand $S-B$ and $G-C$. In the third iteration we will expand $S-B-C$ and notice that C is on the backward closed list, hence declare success for $S-B-C-G$, which is not optimal. Intuitively: enabling code section 3 ensures that when success is declared in section 3 we have found the shortest path between S and G that goes through the current State(node), but we can only guarantee this plan to be optimal if none of the plans in the backward priority queue has an f-cost lower than this just found path's cost. The extension that would work would be to keep track of such found paths, but then only declare success once the f costs in their respective queues have gone above their path cost. This was a tricky question the way we set it up, and we decided to give full credit even if going wrong on code sections 3 and 5.

(v) [2 pts] If amongst the numbered lines of code we only retain 4, is Bidirectional-Graph-Search guaranteed to be optimal? Briefly justify your answer.

No. Consider the same example as in the question about code section 2. Enabling code section 2 will result in finding the path $S-B-G$, which is suboptimal.

(vi) [2 pts] If amongst the numbered lines of code we only retain 5, is Bidirectional-Graph-Search guaranteed to be optimal? Briefly justify your answer.

No. Consider the following state space graph: states are S, A, B, G, edges and costs are $S-A$: 3, $A-G$: 3, $S-B$: 4, $B-G$: 4. Assume the heuristics are: A to G: 3, A to S: 3, B to G: 1, B to S: 1, S to G: 4, G to S: 4. Then in the first iteration S and G will be expanded. In the second iteration $S-B$ and $G-B$ will be expanded, and in the expansion of $G-B$ the algorithm would declare success.

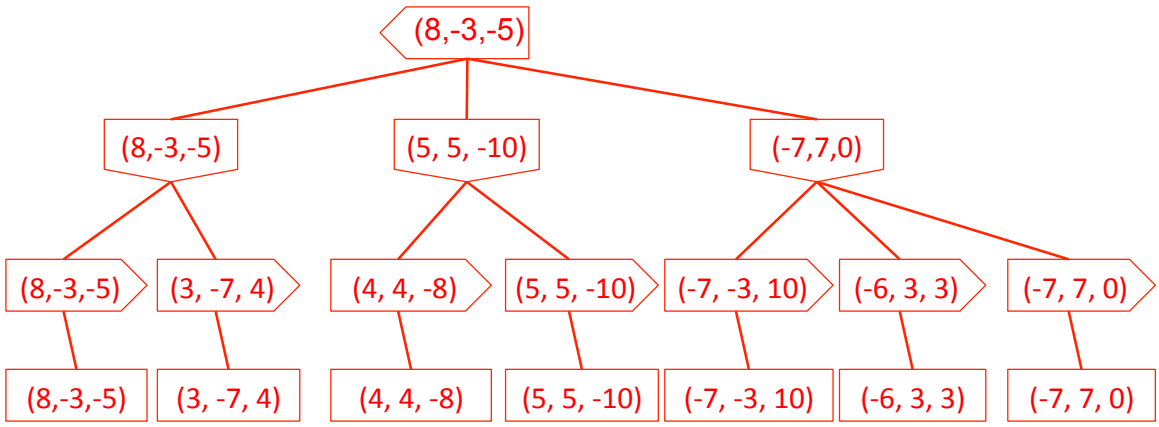
(vii) [2 pts] Which numbered code section(s) should be retained to maximally benefit from the bidirectional search and at the same time retain optimality guarantees?

None can be retained while maintaining optimality guarantees. See explanation about code section 3 for a hint at how you might be able to do better.

Q4. [8 pts] 3-Player Games

(a) [4 pts] **A 3-Player Game.**

Consider the 3-player game shown below. The player going first (top of the tree) is the Left player, the player going second is the Middle player, and the player going last is the Right player, optimizing the left, middle and right utility value respectively. Fill in the values at all nodes. Note all players maximize their respective utility value shown in the tree.



(b) [4 pts] **Pruning for Zero-Sum 3-Player Game.** The same game tree is shown again below.

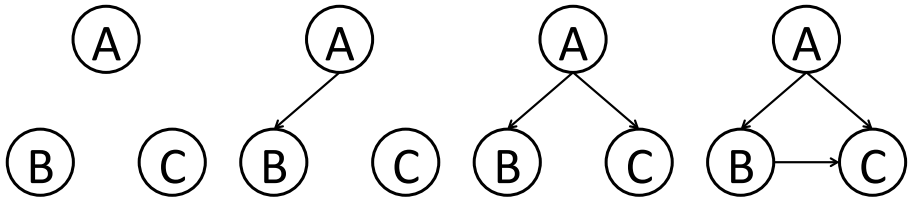
Now assume that we have the knowledge that the sum of the utilities of all 3 players is always zero. Under this assumption is any pruning possible similar to $\alpha - \beta$ pruning? If so mark the pruning on the tree above. If not, briefly explain why not below.

No. $\alpha - \beta$ pruning works based on the following mechanism: when considering options for the minimizer at a node n, you can prune if you found an option for the minimizer that is worse than something the maximizer could ensure by diverting higher-up in the three such that n would never be reached. Reason: no matter what the other options are at node n, diverting will be better for the maximizer.

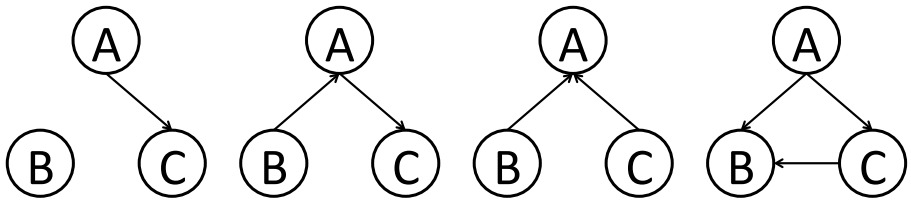
In the three player game we cannot perform such inference: if, let's say, the right player while considering options at a node n has an option that is not great for the left player, let's say giving the left player a pay-off of x, then it is still possible for the right player to prefer another option at node n which is better than x for the left-player.

Q5. [12 pts] Learning a Bayes' Net Structure

(a) You want to learn a Bayes' net over the random variables A, B, C. You decide you want to learn not only the Bayes' net parameters, but also the structure from the data. You are willing to consider the 8 structures shown below. First you use your training data to perform maximum likelihood estimation of the parameters of each of the Bayes' nets. Then for each of the learned Bayes' nets, you evaluate the likelihood of the training data (l^{train}), and the likelihood of your cross-validation data (l^{cross}). Both likelihoods are shown below each structure.



	(a)	(b)	(c)	(d)
l^{train}	0.0001	0.0005	0.0015	0.0100
l^{cross}	0.0001	0.0004	0.0011	0.0009



	(e)	(f)	(g)	(h)
l^{train}	0.0008	0.0015	0.0020	0.0100
l^{cross}	0.0006	0.0011	0.0010	0.0009

(i) [4 pts] Which Bayes' net structure will (on expectation) perform best on test-data? (If there is a tie, list all Bayes' nets that are tied for the top spot.) Justify your answer.

Bayes' nets (c) and (f) as they have the highest cross validation data likelihood.

(ii) [4 pts] Two pairs of the learned Bayes' nets have identical likelihoods. Explain why this is the case.

(c) and (f) have the same likelihoods, and (d) and (h) have the same likelihoods. When learning a Bayes' net with maximum likelihood, we end up selecting the distribution that maximizes the likelihood of the training data from the set of all distributions that can be represented by the Bayes' net structure. (c) and (f) have the same set of conditional independence assumptions, and hence can represent the same set of distributions. This means that they end up with the same distribution as the one that maximizes the training data likelihood, and therefore have identical training and cross validation likelihoods. Same holds true for (d) and (h).

(iii) [4 pts] For every two structures S_1 and S_2, where S_2 can be obtained from S_1 by adding one or more edges, l^{train} is higher for S_2 than for S_1. Explain why this is the case. When learning a Bayes' net with maximum likelihood, we end up selecting the distribution that maximizes the likelihood of the training data from the set of all distributions that can be represented by the Bayes' net structure. Adding an edge grows the set of distributions that can be represented by the Bayes' net, and can hence only increase the training data likelihood under the best distribution in this set.