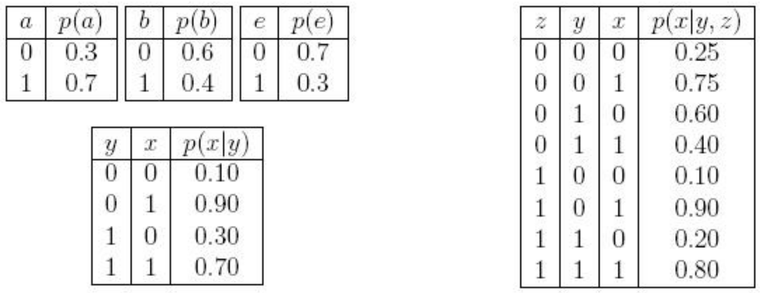
Homework 3: Statistical methods in AI/ML

Instructor: Vibhav Gogate
Vibhav.Gogate@utdallas.edu

Problem 1: [20 points] **



a	$p(a)$
0	0.3
1	0.7

b	$p(b)$
0	0.6
1	0.4

e	$p(e)$
0	0.7
1	0.3

y	x	$p(x\|y)$
0	0	0.10
0	1	0.90
1	0	0.30
1	1	0.70

z	y	x	$p(x\|y,z)$
0	0	0	0.25
0	0	1	0.75
0	1	0	0.60
0	1	1	0.40
1	0	0	0.10
1	0	1	0.90
1	1	0	0.20
1	1	1	0.80

Figure 3: Conditional probability tables

The question investigates the AND/OR search space of the network given in Figure 1 assuming that each variable is binary. The CPTs are given in Figure 3. The CPTs for G, H and D are identical to the 3-dimensional CPT in Figure 3 and the CPTs for H and F are identical to the 2-dimensional CPT in the same figure

- Find and present a pseudo tree of the network whose depth is minimal. Call it T_1. Do the best you can.
- Generate an AND/OR search tree driven by T_1 assuming that each variable has at most two values.
- Annotate the arcs with appropriate weights
- What is the computational cost of computing the probability of evidence $G = 0$ and $H = 1$ in such a network if you use depth-first search over the AND/OR search tree. Demonstrate your computation.
- Can the AND/OR search tree be reduced to a smaller AND/OR search graph.

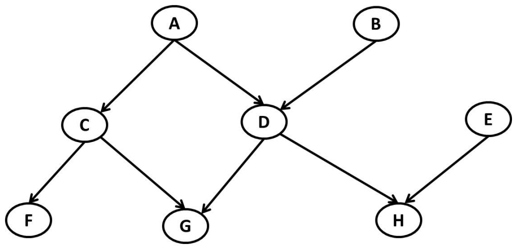


Figure 1:

- Assume that the CPT $P(x|y,z)$ is changed by making some entries deterministic as follows: the first two probabilities are changed to 1 and 0 respectively. Similarly, the last two probabilities are changed to 1 and 0 respectively. Show what would be the changes in the AND/OR search tree as a result.

Part 2: Inference [10 points] Consider a chain Markov network $X_1 - X_2 - X_3 - \ldots - X_n$. Provide an optimal algorithm which calculates $\Pr(X_i, X_j)$ for **all pairs** $i \neq j$. Prove its optimality.

Part 3: [**20 points**] Exercise 12.9 from Koller and Friedman.

Part 4: [**10 points**] Exercise 12.13 from Koller and Friedman.

Part 5: [**10 points**] In class, we saw that the Metropolis-Hastings algorithm converges to a stationary distribution π if it satisfies the following detailed balance condition:

$$\pi(\mathbf{x})\mathcal{T}(\mathbf{x} \to \mathbf{x}')A(\mathbf{x} \to \mathbf{x}') = \pi(\mathbf{x}')\mathcal{T}(\mathbf{x}' \to \mathbf{x})A(\mathbf{x}' \to \mathbf{x})$$

Show that if we use

$$A(\mathbf{x} \to \mathbf{x}') = \frac{\pi(\mathbf{x}')T(\mathbf{x}' \to \mathbf{x})}{\pi(\mathbf{x})\mathcal{T}(\mathbf{x} \to \mathbf{x}') + \pi(\mathbf{x}')\mathcal{T}(\mathbf{x}' \to \mathbf{x})}$$

then the detailed balance condition is satisfied.

What to turn in for Parts 1-5?

- A PDF containing your answers.

Part 6: Programming [60 points]

In this mini coding project, you will implement a "sampling-based version" of the variable elimination and conditioning algorithm (see Algorithm 1). The key idea in the algorithm is to compute the probability of evidence or the partition function by combining sampling with variable elimination (exact inference).

```
Algorithm 1: Sampling-based Variable Elimination and Conditioning
Input: An evidence instantiated Markov or a Bayesian network denoted by $\mathcal{G}$, Integers $w$ and $N$
Output: Estimate of the probability of evidence or the partition function
begin
    $Z = 0$;
    /* Heuristically remove variables from the PGM until the treewidth of the PGM is
       bounded by $w$.  Let $\mathbf{X}$ be the removed variables.                    */
    $\mathbf{X} = wCutset(\mathcal{G})$ // See Algorithm 2
    Let $Q$ be a uniform distribution over $\mathbf{X}$;
    for $i = 1$ to $N$ do
        /* Generate a sample (namely generate a value assignment to all variables in
           $\mathbf{X}$) from $Q$.  Let the sampled assignment be $\mathbf{X} = \mathbf{x}$   */
        $GenerateSample(Q)$;
        Set $\mathbf{X} = \mathbf{x}$ as evidence in the PGM;
        /* Run the variable elimination algorithm to compute the probability of
           evidence (or the partition function).                                      */
        $w = \frac{VE(\mathcal{G}|\mathbf{X}=\mathbf{x})}{Q(\mathbf{X}=\mathbf{x})}$ // Weight of the sample
        $Z = Z + w$;
    end
    return $\frac{Z}{N}$
end
```

Assuming that you have already implemented the variable elimination algorithm, the hard part here is computing the w-cutset. You can use the following algorithm to compute the w-cutset.

What to do and turn in for Part 6?

1. Write a program that implements Algorithm 1 and 2. The program should take the following inputs:
   - A Bayesian or Markov network in the UAI format
   - Evidence in the UAI format
   - An integer w, which denotes the w-cutset bound
   - An integer N which denotes the number of samples

   It should output an estimate of the partition function or the probability of evidence.

2. Replace the proposal distribution Q by the following adaptive proposal distribution. Let the functional form of Q be $Q = \prod_{X \in \mathbf{X}} Q(X)$. Namely, we assume that Q is an empty Bayesian network. Initialize $Q(X)$ to the uniform distribution and then after every 100 samples update each $Q(X)$ using the Monte Carlo estimate that we discussed in class. Formally, let $\mathbf{x}^1, \ldots, \mathbf{x}^T$ be all the samples generated upto time T

$$Q(X = x) = \frac{\sum_{i=1}^{T} \delta_{\mathbf{x}^t}(X = x) w(\mathbf{x}^t)}{\sum_{i=1}^{T} w(\mathbf{x}^t)}$$

Algorithm 2: wCutset

```
Input: An evidence instantiated Markov or a Bayesian network denoted by $\mathcal{G}$, Integer $w$
Output: A set of variables which form a w-cutset
Assumptions:
Let $(X_1, \ldots, X_n)$ denote the min-degree ordering of variables of $\mathcal{G}$;
Let $C(X_i)$ denote the set of variables in the bucket of $X_i$ when you run bucket elimination along the
 order $(X_1, \ldots, X_n)$. $C(X_i)$ is thus a cluster in a tree decomposition;
begin
    $\mathbf{X}=\emptyset$;
    repeat
        Find a variable that appears in the most clusters. Ties broken randomly. Let us call the
         variable $X$;
        Remove $X$ from all the clusters;
        $\mathbf{X} = \mathbf{X} \cup X$;
    until no cluster has more than $w+1$ variables;
    return $\mathbf{X}$;
end
```

where $\delta_{\mathbf{x}^t}(X = x)$ is the dirac-delta function which is 1 when $X = x$ is present in $\mathbf{x}^t$ and 0 otherwise and

$$w(\mathbf{x}^t) = \frac{VE(\mathcal{G}|\mathbf{x}^t)}{Q(\mathbf{x}^t)}$$

Add an option to your program so that it will use the adaptive proposal distribution instead of the uniform distribution.

For parts 1 and 2, turn in your source code and a readme file that describes how to compile and use your software.

3. Run the two programs on the six PGMs given on the class web page. Try the following values for $N = \{100, 1000, 10000, 20000\}$ and $w = \{1, 2, 3, 4, 5\}$ and run each algorithm at least 10 times using different random seeds. For each run, compute the log-relative error between the exact partition function and the approximate one computed by your algorithm. Namely, compute

$$\frac{\log(Z) - \log(\widehat{Z})}{\log(Z)}$$

where Z is the exact answer and $\widehat{Z}$ is the approximate answer.

For each PGM, report your results in a table such as the one given below and describe your findings in a few sentences (e.g., which method is better and why?; how N and w impact the accuracy, time-complexity and variance; etc.).

Problem-name			**Uniform Proposal**				**Adaptive Proposal**			
		N →	100	1K	10K	20K	100	1K	10K	20K
Grids 14	w=1	Time	t± t_s							
	w=1	Error	e ± e_s							
Grids 15	w=2	Time								
	w=2	Error								
Grids 16	w=3	Time								
	w=3	Error								
Grids 17	w=4	Time								
	w=4	Error								
Grids 18	w=5	Time								
	w=5	Error								

In the table, the time is in seconds. The quantity before ± is the average (e.g., e and t) and the quantity after ± is the standard deviation (e.g., t_s and e_s) over 10 runs (recall that you will run each algorithm 10 times using 10 different random seeds).

For part 3, turn in a PDF or a Word file.

A useful tip. Use a python script to set up your experiments. If you run each experiment individually by hand, you will take forever.