Homework 3

CS 6347: Statistical methods in AI/ML

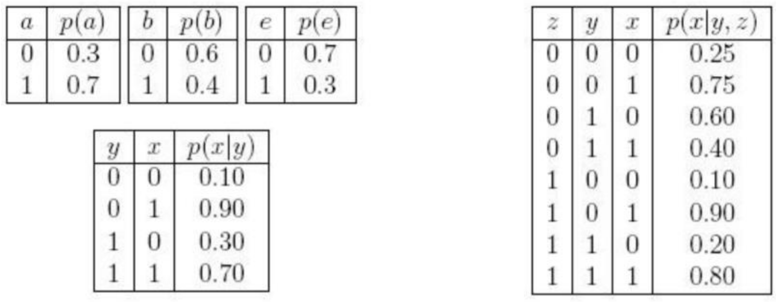
Instructor: Vibhav Gogate

Vibhav.Gogate@utdallas.edu

Student: Xiaodi Li

Net ID: XXL170011

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 C 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.

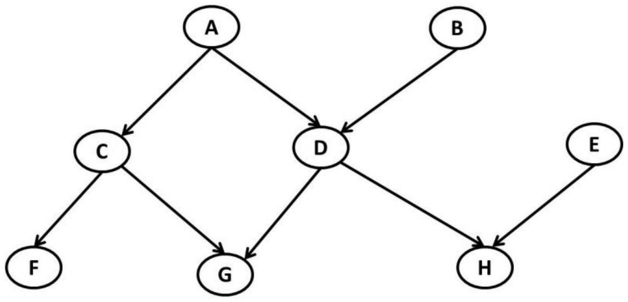
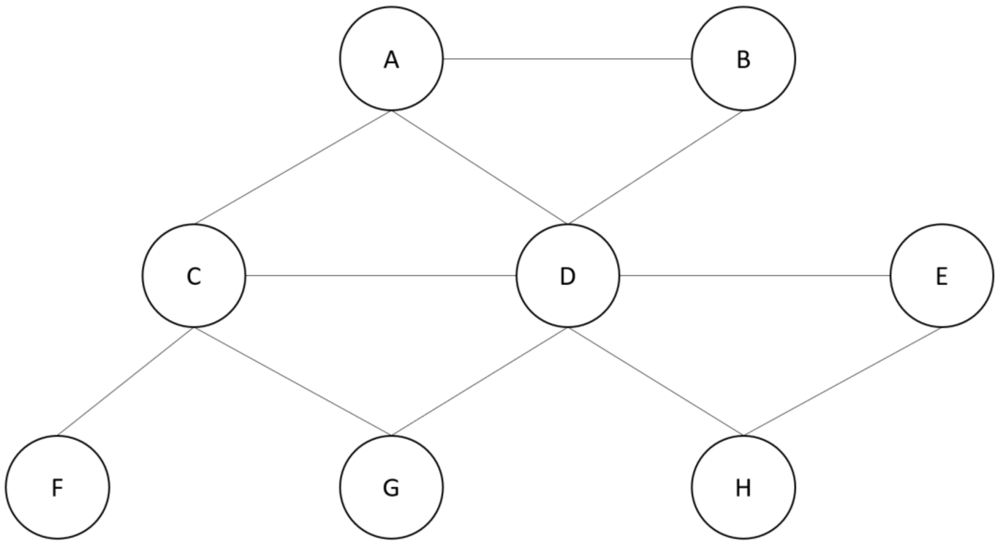


Figure 1:

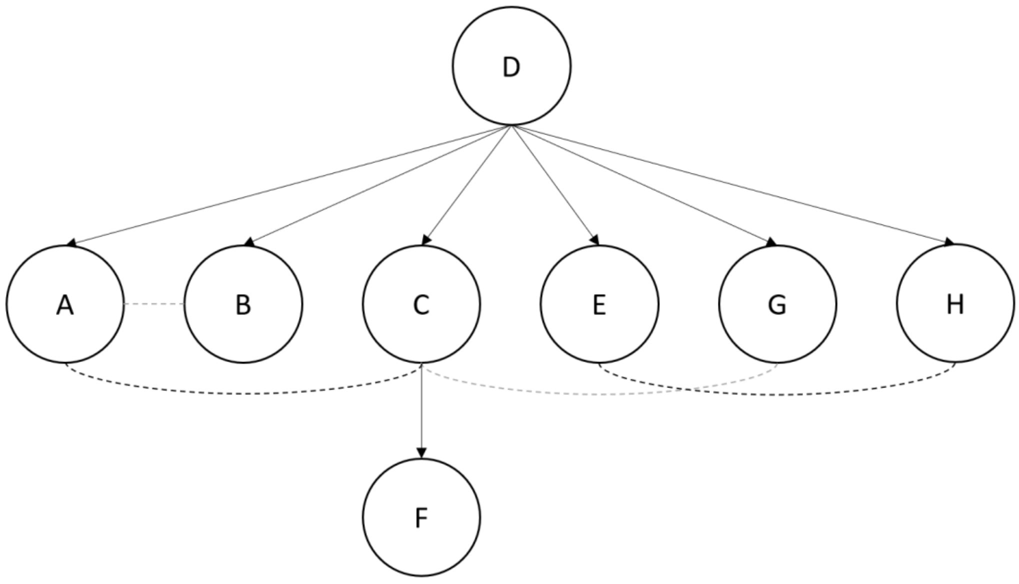
- Assume that the CPT P(x|y, z) is changed by making some entries deterministic as follows: the first two probabilities are changed to 1 and 0 respectively. Similarly, the last two probabilities are changed to 1 and 0 respectively. Show what would be the changes in the AND/OR search tree as a result.

Answer:

- First, we transfer the Bayesian network into a Markov network as follows:

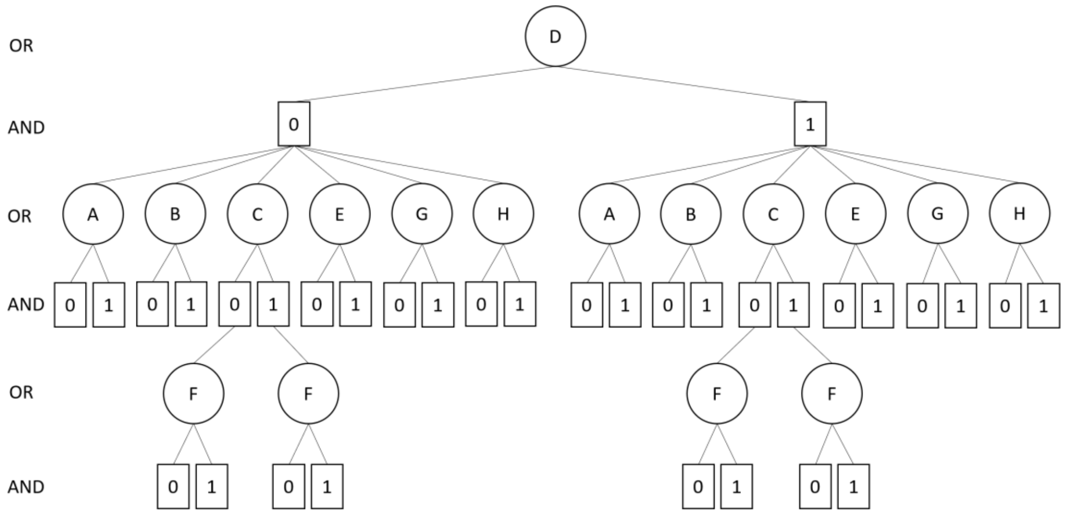


Then, using the Max degree heuristic method, we can construct the following pseudo tree T_1 (rooted at D):

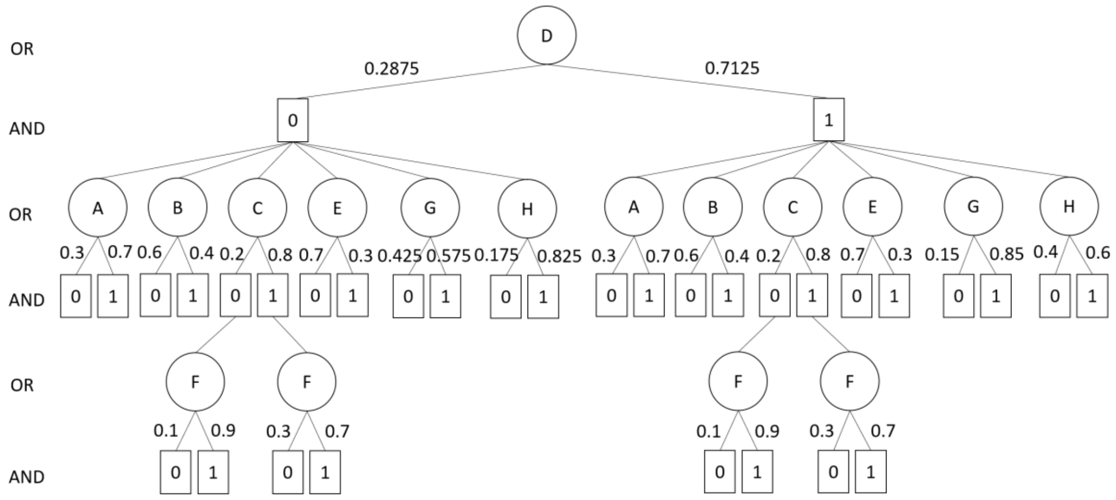


Thus, the depth of T_1 is 2.

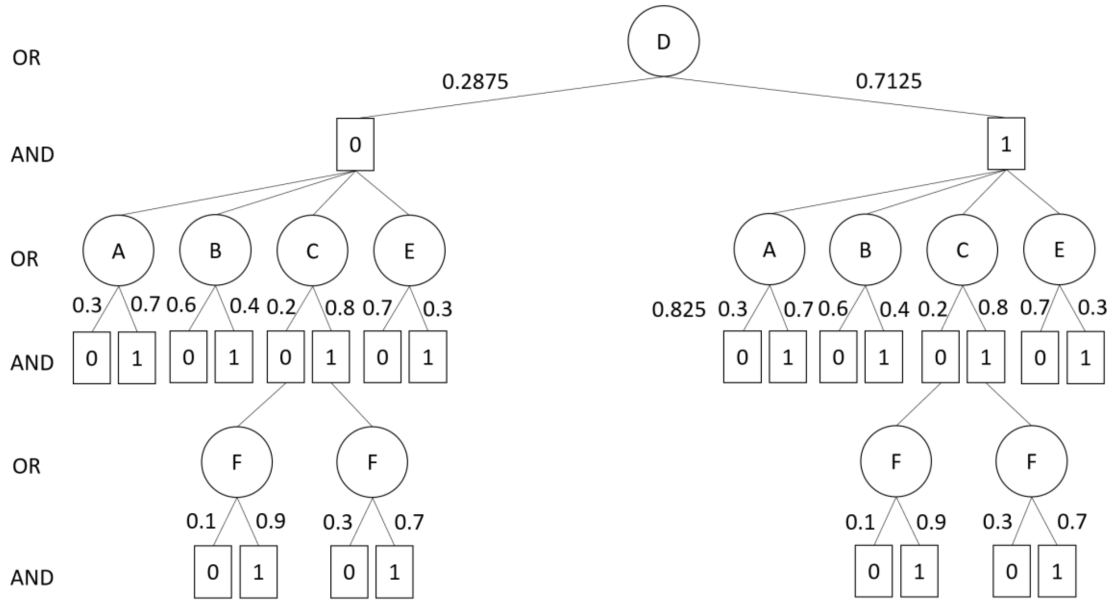
- The AND/OR search tree is as follows:



- The weights of arcs are as follows:



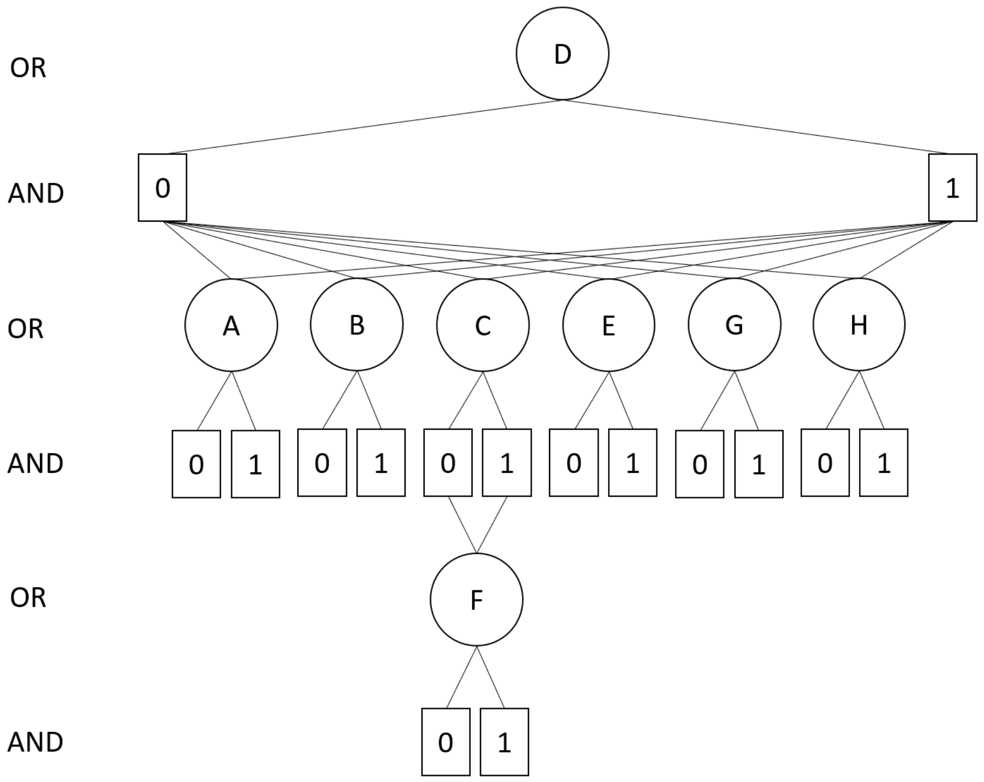
- First, after eliminating node G and node H, our AND/OR search tree will become:



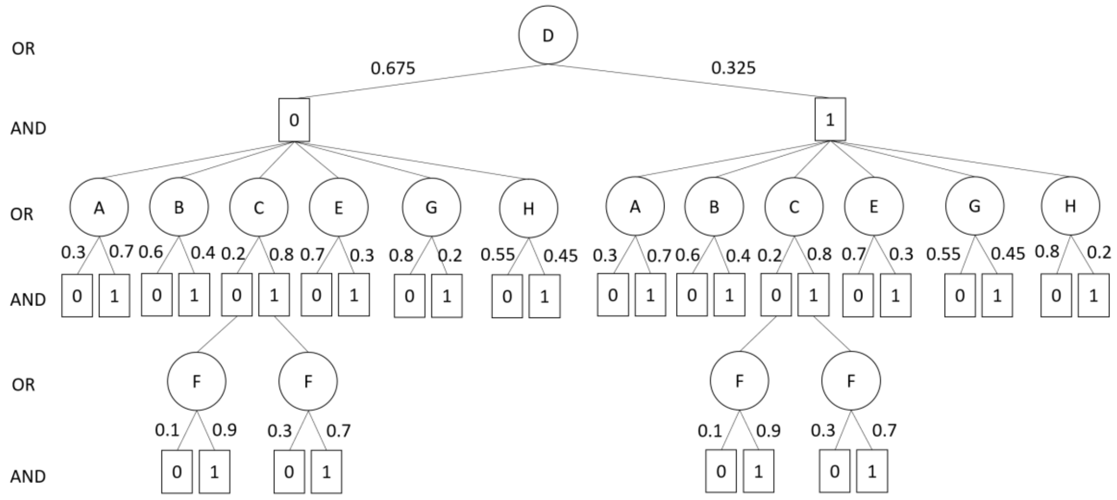
Second, we apply DFS on this AND/OR search tree. The visiting order will be $D, 0, A, 0, 1, B, 0, 1, C, 0, F, 0, 1, 1, F, 0, 1, E, 0, 1, 1, A, 0, 1, B, 0, 1, C, 0, F, 0, 1, 1, F, 0, 1, E, 0, 1$

. Let us denote k as the domain size, n as the number of variables, and d as the induced-depth of the pseudo tree T_1. There are $n-2$ variables in the network. The DFS algorithm will visit each node and arc once. Thus, the computational cost is $O(|V|+|E|) = O(2(n-2)k^d - 1)$. In this case, the computational cost is $O(2*39-1) = O(77)$.

- Yes, it can. We can merge those nodes that root identical subtree as shown below:



- The weights of arcs will change as shown below:



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.

Answer:

According to the conditional probability theory, we know that $Pr(X_i, X_j) = Pr(X_i|X_j)Pr(X_j)$. Where $Pr(X_i|X_j) = T(X_j \to X_i)$, $Pr(X_j) = \sum_{X_k} P(X_k)T(X_k \to X_j)$. Thus, we can calculate $Pr(X_i, X_j)$ by traversing from X_1 to X_n using the chain dynamics in one pass. As we can calculate the probabilities in one pass, this algorithm is optimal.

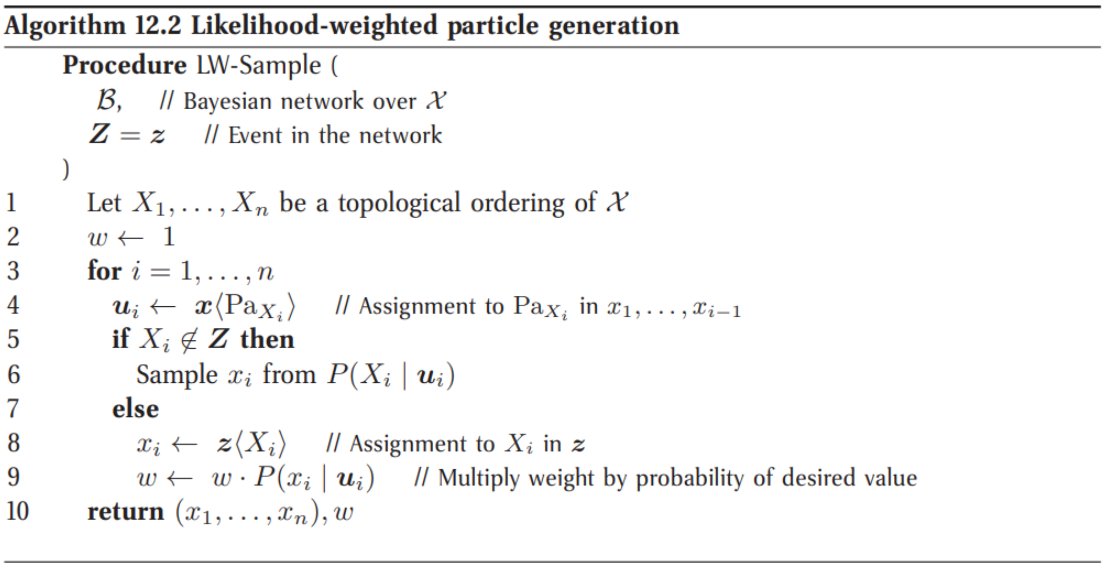
Part 3: [20 points] Exercise 12.9 from Koller and Friedman.

In this question, we consider the application of importance sampling to Markov networks.

a. Explain intuitively why we cannot simply apply likelihood weighting to Markov networks.
b. Show how likelihood weighting can be applied to chordal Markov networks. Is this approach interesting? Explain.
c. Provide a technique by which the more general framework of importance sampling can be applied to Markov networks. Be sure to define both a reasonable proposal distribution and an algorithmic technique for computing the weights.

Answer:

a. On one hand, because likelihood is a function P(A|B) which requires the dependence of data (e.g., A is dependent on B) while Markov networks are undirected and thus do not have such dependence. Thus, we cannot derive weights of samples from likelihood of evidence accumulated during sampling process. On the other hand, in Algorithm 12.2, line 4, we do not have parent-child relationship in Markov networks. Thus, we cannot simply apply likelihood weighting to Markov networks.



Algorithm 12.2 Likelihood-weighted particle generation

```
    Procedure LW-Sample (
       $\mathcal{B}$,      // Bayesian network over $\mathcal{X}$
       $\boldsymbol{Z} = \boldsymbol{z}$      // Event in the network
    )
1     Let $X_1, \ldots, X_n$ be a topological ordering of $\mathcal{X}$
2     $w \leftarrow 1$
3     for $i = 1, \ldots, n$
4       $\boldsymbol{u}_i \leftarrow \boldsymbol{x}\langle \mathrm{Pa}_{X_i} \rangle$      // Assignment to $\mathrm{Pa}_{X_i}$ in $x_1, \ldots, x_{i-1}$
5       if $X_i \notin \boldsymbol{Z}$ then
6         Sample $x_i$ from $P(X_i \mid \boldsymbol{u}_i)$
7       else
8         $x_i \leftarrow \boldsymbol{z}\langle X_i \rangle$      // Assignment to $X_i$ in $\boldsymbol{z}$
9         $w \leftarrow w \cdot P(x_i \mid \boldsymbol{u}_i)$      // Multiply weight by probability of desired value
10    return $(x_1, \ldots, x_n), w$
```

b. We can first convert chordal Markov networks into Bayesian networks and then apply likelihood weighting on the generalized Bayesian networks. Since for chordal graphs, a set of independences can be perfectly represented by both the Bayesian networks and the Markov networks, we can guarantee that the likelihood weighting on the Bayesian networks can have the same effect on the chordal Markov networks. Thus, this approach is interesting to implement.

c. In this problem, we provide with a normalized importance sampling on Markov networks. Let us denote $\tilde{P}$ and unnormalized version of P. We want to sample from Q instead of P and we do not know P. $P(X) = k \prod_i f_i(X)$ while $\tilde{P}(X) = \prod_i f_i(X)$. We will generate samples x from Q and assign weights $\tilde{P}(X)/Q(x)$, and normalize the estimator.

Algorithm:

```
For $i = 1$ to $n$ do (sample n particles):
    Sample $x^{(i)}$ from $Q$
    Assign weight: $w_i \leftarrow \tilde{P}(x^{(i)})/Q(x^{(i)})$
```

Approximation: $E_P[f(x)] \approx \frac{\sum_{i=1}^{n} w_i f(x^{(i)})}{\sum_{i=1}^{n} w_i}$. This is a biased estimator for finite n (unbiased for $n = \infty$). Variance of estimator decreases linearly with sample size.

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

Show directly from equation (12.21) (without using the detailed balance equation) that the posterior distribution $P(X|e)$ is a stationary distribution of the Gibbs chain (equation (12.22)).

Answer:

$$\pi(X = x') = \sum_{x \in Val(X)} \pi(X = x) T(x \rightarrow x'). \text{ (12.21)}$$

$$T_i\big((x_{-i}, x_i) \rightarrow (x_{-i}, x_i')\big) = P(x_i' | x_{-i}) \text{ (12.22)}$$

The distribution at time t can be represented as:

$$P^t(x') \approx P^{t+1}(x') = \sum_{x \in Val(X)} P^t(x) T(x \rightarrow x').$$

Equation (12.21) refers to the global transition model, which in the case of the Gibbs Chain corresponds to the probability of choosing a variable to flip and then applying the local transition model. The probabilities of flipping of all variables are the same. The variable x_i is sampled from the distribution $P(x_i|u_i)$ where u_i is the assignment to all variables other than x_i.

Notice in the case of the Gibbs chain, only one value is changed at a time.

$$\frac{1}{|X|} \sum_{k \in |X|} \sum_{x_k \in Val(X_k)} P(x_k, u_k | e) P(x_k' | u_k, e)$$

$$\frac{1}{|X|}\sum_{k\in|X|}\sum_{x_k\in Val(X_k)} P(x_k|u_k)P(x_k', u_k|e)$$

$$P(X'|e)\frac{1}{|X|}\sum_{k\in|X|}\sum_{x_k\in Val(X_k)} P(x_k|u_k)$$

$$P(X'|e)$$

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:

$$\pi(x)T(x \to x')A(x \to x') = \pi(x')T(x' \to x)A(x' \to x)$$

Show that if we use

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

then the detailed balance condition is satisfied.

Answer:

If we use $A(x \to x') = \frac{\pi(x')T(x'\to x)}{\pi(x)T(x\to x')+\pi(x')T(x'\to x)}$, we will have $A(x' \to x) = \frac{\pi(x)T(x\to x')}{\pi(x')T(x'\to x)+\pi(x)T(x\to x')}$.

Then,

$$\begin{aligned}\pi(x)T(x \to x')A(x \to x') &= \pi(x)T(x \to x')\frac{\pi(x')T(x' \to x)}{\pi(x)T(x \to x') + \pi(x')T(x' \to x)}\\ &= \pi(x')T(x' \to x)\frac{\pi(x)T(x \to x')}{\pi(x')T(x' \to x) + \pi(x)T(x \to x')}\\ &= \pi(x')T(x' \to x)A(x' \to x)\end{aligned}$$

Thus, 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.

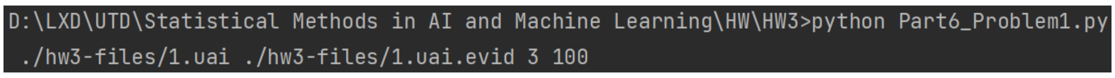
What to do and turn in for Part 6?

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

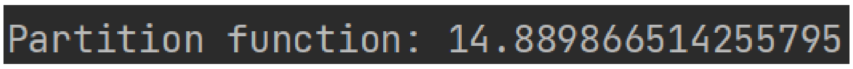
    It should output an estimate of the partition function or the probability of evidence.

Answer: See Part6_Problem1.py and README_Problem1.txt

1.uai

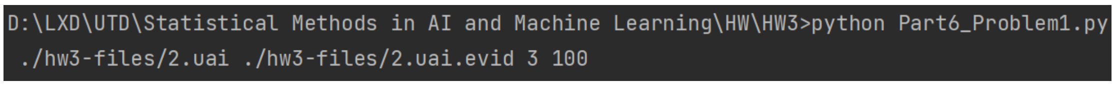


```
D:\LXD\UTD\Statistical Methods in AI and Machine Learning\HW\HW3>python Part6_Problem1.py
 ./hw3-files/1.uai ./hw3-files/1.uai.evid 3 100
```

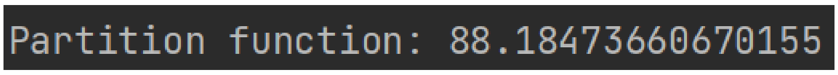


```
Partition function: 14.889866514255795
```

2.uai

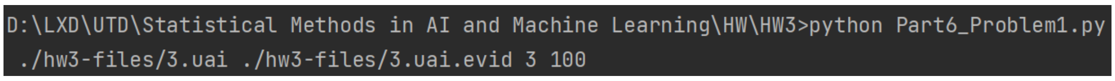


```
D:\LXD\UTD\Statistical Methods in AI and Machine Learning\HW\HW3>python Part6_Problem1.py
 ./hw3-files/2.uai ./hw3-files/2.uai.evid 3 100
```

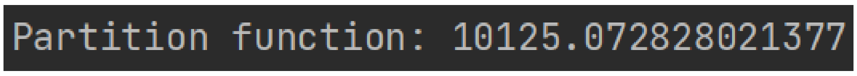


```
Partition function: 88.18473660670155
```

3.uai



```
D:\LXD\UTD\Statistical Methods in AI and Machine Learning\HW\HW3>python Part6_Problem1.py
 ./hw3-files/3.uai ./hw3-files/3.uai.evid 3 100
```



```
Partition function: 10125.072828021377
```

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 $x^1, \ldots, x^T$ be all the samples generated upto time T

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

```
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
```

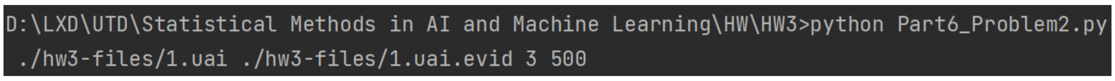
where $\delta_{x^t}(X = x)$ is the dirac-delta function which is 1 when $X = x$ is present in x^t and 0 otherwise and

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

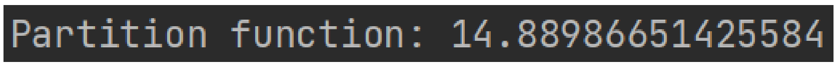
Add an option to your program so that it will use the adaptive proposal distribution instead of the uniform distribution.

Answer: See Part6_Problem2.py and README_Problem2.txt

1.uai

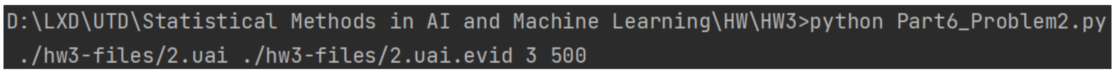


```
D:\LXD\UTD\Statistical Methods in AI and Machine Learning\HW\HW3>python Part6_Problem2.py
 ./hw3-files/1.uai ./hw3-files/1.uai.evid 3 500
```

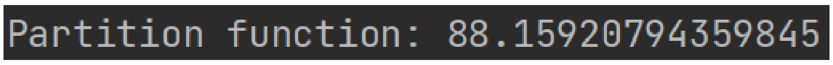


```
Partition function: 14.88986651425584
```

2.uai

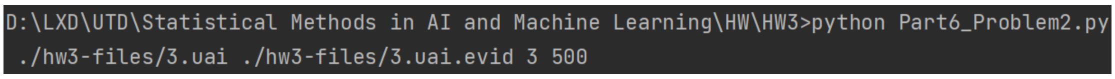


```
D:\LXD\UTD\Statistical Methods in AI and Machine Learning\HW\HW3>python Part6_Problem2.py
 ./hw3-files/2.uai ./hw3-files/2.uai.evid 3 500
```

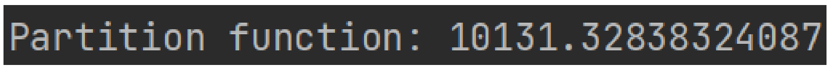


```
Partition function: 88.15920794359845
```

3.uai



```
D:\LXD\UTD\Statistical Methods in AI and Machine Learning\HW\HW3>python Part6_Problem2.py
 ./hw3-files/3.uai ./hw3-files/3.uai.evid 3 500
```



```
Partition function: 10131.32838324087
```

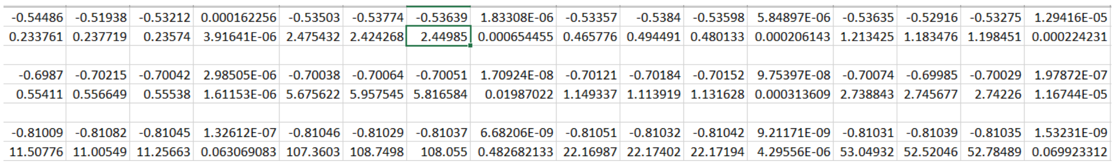
For parts 1 and 2, turn in your source code and a readme file that describes how to compile and use your software.

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
$$\frac{log(Z) - log(\hat{Z})}{log(Z)}$$
where Z is the exact answer and $\hat{Z}$ 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.).

Answer: I cannot run my program on the given datasets because of the long running time and the overflow of the output. Thus, I run the experiments on previous datasets with 2 runs each experiment instead.

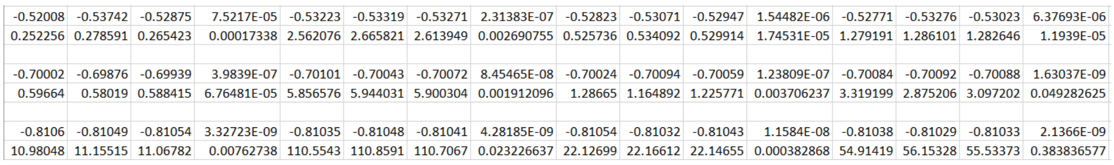
Problem-name			Uniform Proposal				Adaptive Proposal			
		N →	100	1K	200	500	100	1K	200	500
1	w=1	Time	$t \pm t_s$							
	w=1	Error	$e \pm e_s$							
2	w=2	Time								
	w=2	Error								
3	w=3	Time								
	w=3	Error								

See the table below:
Order: Error, Time
Uniform Proposal:



-0.54486	-0.51938	-0.53212	0.000162256	-0.53503	-0.53774	-0.53639	1.83308E-06	-0.53357	-0.5384	-0.53598	5.84897E-06	-0.53635	-0.52916	-0.53275	1.29416E-05
0.233761	0.237719	0.23574	3.91641E-06	2.475432	2.424268	2.44985	0.000654455	0.465776	0.494491	0.480133	0.000206143	1.213425	1.183476	1.198451	0.000224231
-0.6987	-0.70215	-0.70042	2.98505E-06	-0.70038	-0.70064	-0.70051	1.70924E-08	-0.70121	-0.70184	-0.70152	9.75397E-08	-0.70074	-0.69985	-0.70029	1.97872E-07
0.55411	0.556649	0.55538	1.61153E-06	5.675622	5.957545	5.816584	0.01987022	1.149337	1.113919	1.131628	0.000313609	2.738843	2.745677	2.74226	1.16744E-05
-0.81009	-0.81082	-0.81045	1.32612E-07	-0.81046	-0.81029	-0.81037	6.68206E-09	-0.81051	-0.81032	-0.81042	9.21171E-09	-0.81031	-0.81039	-0.81035	1.53231E-09
11.50776	11.00549	11.25663	0.063069083	107.3603	108.7498	108.055	0.482682133	22.16987	22.17402	22.17194	4.29556E-06	53.04932	52.52046	52.78489	0.069923312

Adaptive Proposal:



-0.52008	-0.53742	-0.52875	7.5217E-05	-0.53223	-0.53319	-0.53271	2.31383E-07	-0.52823	-0.53071	-0.52947	1.54482E-06	-0.52771	-0.53276	-0.53023	6.37693E-06
0.252256	0.278591	0.265423	0.00017338	2.562076	2.665821	2.613949	0.002690755	0.525736	0.534092	0.529914	1.74531E-05	1.279191	1.286101	1.282646	1.1939E-05
-0.70002	-0.69876	-0.69939	3.9839E-07	-0.70101	-0.70043	-0.70072	8.45465E-08	-0.70024	-0.70094	-0.70059	1.23809E-07	-0.70084	-0.70092	-0.70088	1.63037E-09
0.59664	0.58019	0.588415	6.76481E-05	5.856576	5.944031	5.900304	0.001912096	1.28665	1.164892	1.225771	0.003706237	3.319199	2.875206	3.097202	0.049282625
-0.8106	-0.81049	-0.81054	3.32723E-09	-0.81035	-0.81048	-0.81041	4.28185E-09	-0.81054	-0.81032	-0.81043	1.1584E-08	-0.81038	-0.81029	-0.81033	2.1366E-09
10.98048	11.15515	11.06782	0.00762738	110.5543	110.8591	110.7067	0.023226637	22.12699	22.16612	22.14655	0.000382868	54.91419	56.15328	55.53373	0.383836577

In the table, the time is in seconds. The quantity before $\pm$ is the average (e.g., e and t) and the quantity after $\pm$ 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.