

# Homework 3.5

## Statistical methods in AI/ML

Instructor: Vibhav Gogate  
Vibhav.Gogate@utdallas.edu

Due date: March 31 via E-learning

**Description** 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?

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

---

**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  $(X_1, \dots, X_n)$  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  $(X_1, \dots, X_n)$ .  $C(X_i)$  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**

---

- 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  $\mathbf{x}^1, \dots, \mathbf{x}^T$  be all the samples generated upto time  $T$

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

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

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

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  $w = \{2, 4, 6, 8, 10\}$  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.).

	N $\rightarrow$	Uniform Proposal				Adaptive Proposal			
		100	1K	10K	20K	100	1K	10K	20K
w=2	Time	2.534 $\pm$ 3							
w=2	Error	7 $\pm$ 2.2							
w=4	Time								
w=4	Error								
w=6	Time								
w=6	Error								
w=8	Time								
w=8	Error								
w=10	Time								
w=10	Error								

The time is in seconds. In the table, the quantity before  $\pm$  is the average and the quantity after  $\pm$  is the standard deviation over 10 runs (recall that you will run each algorithm 10 times using 10 different random seeds). When the standard deviation is small, your algorithm is robust.

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

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