CS 67800, Spring 2017/18

Problem Set 3: Exact and Approximate Inference

Submission Date : Sunday 10/6/18, 23:59

---

1. **Elimination Ordering:** In this problem we will consider the computational implications of various variable elimination ordering strategies.

   Definition: *The Induced Graph*
   Let Φ be a set of factors over $\mathcal{X} = X_1 \dots X_n$ and $\prec$ be an elimination ordering for some subset $\mathbf{X} \in \mathcal{X}$. The induced graph $\mathcal{I}_{\Phi,\prec}$ is an undirected graph over $\mathbf{X}$, where X_i and X_j are connected by an edge if they both appear in some intermediate factor generated by the VE algorithm using $\prec$ as an elimination ordering.

   Note: The *moralized graph* is the first step of the *induced graph* since a child and its parents will always be together in some term.

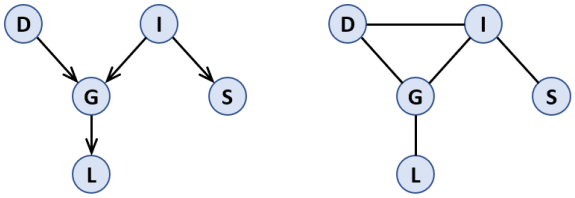

Figure 1: G and $M[G]$

   (a) Consider the *student* BN $\mathcal{G}$ above (Similar to PGM book, Fig. 3.3) and its undirected moralized graph $\mathcal{M}[\mathcal{G}]$. For the elimination ordering $\prec$ of G, D, I, L, and S, show the resulting induced graph $\mathcal{I}_{\mathcal{G},\prec}$.

   Answers: Eliminating G adds edges $D - L$ and $I - L$. Eliminating D then adds no edges because I and L are already connected. Eliminating I adds an edge $S - L$ (NOT an edge $S - D$ since D has already been eliminated). Then eliminating L and S add no extra edges.

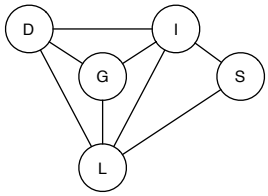

Figure 2: $I_{\mathcal{G},\prec}$

   (b) Consider the following greedy algorithm for producing a good elimination ordering:

   Greedy VE Ordering
   - Initialize $\mathcal{G}' = \mathcal{G}$, set all nodes in $\mathcal{G}'$ as unmarked
   - for $k = 1 \dots n$

  - Choose the unmarked node in $\mathcal{G}'$ that minimizes some greedy function f
  - Assign it to be X_k (number k in the elimination ordering) and mark it in $\mathcal{G}'$
  - Add edges in $\mathcal{G}'$ between all neighbors of X_k **in** $\mathcal{G}$ that are unmarked in $\mathcal{G}'$
- Output $X_1 \dots X_n$ (the chosen elimination order in $\mathcal{G}$)

Consider three greedy functions f for the above algorithm:

- $f_A(X_i)$ = Number of unmarked neighbors of X_i in $\mathcal{G}'$
- $f_B(X_i)$ = Number of added edges caused by marking X_i
- $f_C(X_i)$ = Size (number of entries in the table) of the intermediate factor produced by eliminating X_i at this stage

Show that none of these functions *dominate* the others. That is, show that no function results in an algorithm that, in all cases, produces an ordering that is better than all other orderings with respect to the computational cost of full variable elimination (e.g. to compute the normalization constant for a MN). For instance, you could show an undirected graph $\mathcal{G}$ where an ordering produced by f_C results in less computations in variable elimination than an ordering produced by f_B, another example where f_C performs better than f_A, and another example where f_B performs better than f_C.

For each case, define the undirected graph, the factors over it, and the number of values each variable can take. From your examples, argue that none of the above heuristic functions is optimal.

Answers: I will show an example where f_C is better than both f_A and f_B and an example where f_A is better than f_C.

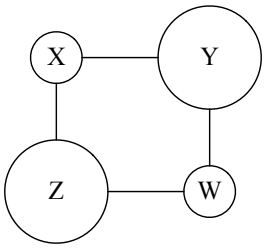

Figure 3: f_C better than f_A and f_B

Consider Figure (3). Suppose pairwise factors $\phi(X, Y)$, $\phi(Y, W)$, $\phi(X, Z)$, and $\phi(Z, W)$. Suppose $|Val(X)| = |Val(W)| = d$, $|Val(Y)| = |Val(Z)| = D$, and $D >> d$.

f_C could choose the ordering Y, Z, X, W (it's ensured to pick one of Z or Y first to avoid creating a large factor over both Z and Y). The cost of variable elimination under this ordering is Dd^2 multiplications and $(D-1)d^2$ additions to eliminate Y, Dd^2 multiplications and $(D-1)d^2$ additions to eliminate Z, d^2 multiplications and $(d-1)d$ additions to eliminate X, and $(d-1)$ additions to eliminate W.

f_A and f_B could each choose the ordering X, Y, Z, W (any possible ordering is equally attractive to these heuristics because they don't consider information about domain sizes). The cost of variable elimination under this ordering is D^2d multiplications and $D^2(d-1)$ additions to eliminate X, D^2d multiplications and $(D-1)Dd$ additions to eliminate Y, Dd multiplications and $(D-1)d$ additions to eliminate Z, and $(d-1)$ additions to eliminate W.

Since $D \gg d$ the $D^2(d-1)$ terms in the cost of variable elimination under an ordering produced by f_B or f_A outweigh the cost of variable elimination under an ordering produced by f_C. So neither f_A nor f_B dominates. The intuition in this example is that f_A and f_B can create unnecessarily large factors because they don't consider the domain sizes of variables.

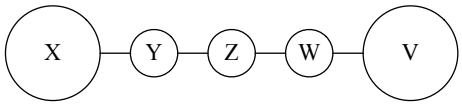

Figure 4: f_A better than f_C

Consider Figure (4). Suppose pairwise factors $\phi(X,Y)$, $\phi(Y,Z)$, $\phi(Z,W)$, and $\phi(W,V)$. Suppose $|Val(Y)| = |Val(Z)| = |Val(W)| = d$, $|Val(X)| = |Val(V)| = D$, $D = 13$, and $d = 2$.

f_A could choose the ordering X, Y, Z, W, V (it's ensured to only pick from the ends of chain). The cost of variable elimination under this ordering is $(D-1)d$ additions to eliminate X, d^2 multiplications and $(d-1)d$ additions to eliminate Y, d^2 multiplications and $(d-1)d$ additions to eliminate Z, Dd multiplications and $(d-1)D$ additions to eliminate W, and $(D-1)$ additions to eliminate V. This is 34 multiplications and 53 additions.

f_C could choose the ordering Z, X, Y, W, V (it's ensured to pick Z first since the size of a factor over X, Z, W is 8, which is less than that for eliminating X or V (26) and less than that for eliminating Y or W (52)). The cost of variable elimination under this ordering is d^3 multiplications and $(d-1)d^2$ additions to eliminate Z, Dd multiplications and $(D-1)d$ additions to eliminate X, d^2 multiplications and $(d-1)d$ additions to eliminate Y, Dd multiplications and $(d-1)D$ additions to eliminate W, and $(D-1)$ additions to eliminate V. This is 64 multiplications and 55 additions.

So f_C doesn't dominate. The intuition in this example is that f_C can avoid dealing with large intermediate factors that it will eventually have to deal with anyways, and in the meantime create a bigger mess for itself.

2. **Sampling on a Tree:** Suppose we have a distribution $P(\boldsymbol{X}, \boldsymbol{E})$ over two sets of variables $\boldsymbol{X}$ and $\boldsymbol{E}$. Our distribution is represented by a nasty BN with very dense connectivity, and our sets of variables $\boldsymbol{X}$ and $\boldsymbol{E}$ are spread arbitrarily throughout the network. In this problem our goal is to use the sampling methods we learned in class to estimate the *posterior* probability $P(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e})$. More specifically, we will use a tree-structured BN as the proposal distribution for use in the Importance Sampling algorithm.

   (a) For a particular value of $\boldsymbol{x}$ and $\boldsymbol{e}$: Can we compute $P(\boldsymbol{x} \mid \boldsymbol{e})$ exactly, in a tractable way? Can we sample directly from the distribution $P(\boldsymbol{X} \mid \boldsymbol{e})$? Can we compute $\tilde{P}(\boldsymbol{x} \mid \boldsymbol{e}) = P(\boldsymbol{x}, \boldsymbol{e})$ exactly, in a tractable way? For each question, provide a Yes/No answer and a single sentence explanation or description.

   Answers: No, No, Yes

   (b) Now, suppose your friendly TAs have given you a tree network, where X_1 is the root and each X_i for $i \neq 1$ has exactly one parent $X_{PA(i)}$. They tell you that the distribution $Q(\boldsymbol{X}, \boldsymbol{E})$ defined by this network is "close" to the distribution $P(\boldsymbol{X}, \boldsymbol{E})$. You now want to use **the posterior** in Q as your proposal distribution for importance sampling.

i. Show how to sample from the posterior in Q. More specifically, provide an explicit construction for a clique tree over this network, and show how to use it to compute the posterior $Q(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$. Describe how to sample from this posterior, once it has been computed.

ii. Now you must re-weight the samples according to the rules of importance sampling. You want your weighted samples to accurately represent the actual posterior in the original network $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$. Show precisely how you determine the weights $w[m]$ for the samples.

iii. Show the form of the final estimator $\hat{P}(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e})$ for $P(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e})$, in terms of the samples from part i, and the weights from part ii.

Answer: Create a clique tree where each clique is a pair of variables (child and parent). Multiply in the indicator functions for the evidence $\boldsymbol{E} = \boldsymbol{e}$. The distribution across the tree now represents $Q(\boldsymbol{X}, \boldsymbol{E} \mid \boldsymbol{E} = \boldsymbol{e})$. Calibrate the tree. Now, the belief at a clique over $(X, \mathbf{Pa}_X)$ is proportional to $Q(X, \mathbf{Pa}_X \mid \boldsymbol{E} = \boldsymbol{e})$. From this belief, we can easily compute $Q(X \mid \mathbf{Pa}_X, \boldsymbol{E} = \boldsymbol{e})$ (use Bayes' Rule). Using these CPDs, we can now forward sample directly from the posterior in Q (sample the first variable, then instantiate it in neighboring cliques, repeating to forward sample).

To get weights:

$$w[m] = \frac{P'(\boldsymbol{x}[m], \boldsymbol{e}[m])}{Q(\boldsymbol{x}[m], \boldsymbol{e}[m] \mid \boldsymbol{e})}$$

The estimator:

$$\hat{P}(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{E} = \boldsymbol{e}) = \frac{\sum_{m=1}^{M} w[m] 1\{\boldsymbol{x}[m] = \boldsymbol{x}\}}{\sum_{m=1}^{M} w[m]}$$

Additional Notes:

- Sampling directly from the BN tree results in sampling from the joint distribution. We could try VE on the tree BN to calculate Q(E=e), but since the evidence is spread all over, it is not trivial.
- The CT is a tool for us to factor-in the evidence. If we try to zero-out entries directly in the BN, we will not have a valid representation.
- A CT can start with an unnormalized set of factors, for example a Gibbs distribution without $1/Z$ or the joint distribution in a BN with some evidence $P(\mathbf{X}, \mathbf{E} = e)$. VE can be performed on the unnormalized CT.
- Zeroing in all factors the entries that are not consistent with the evidence is a way to factor-in the evidence in a clique tree.

3. **Gibbs sampling:** Consider a BN: $X \rightarrow Z \leftarrow Y$, where the variables are all binary, X and Y are both uniformly distributed, and Z is the deterministic exclusive or of X and Y. In class we have seen that Gibbs sampling on this structure with the evidence $Z = 1$ will estimate $P(X = 1 \mid Z = 1)$ either as 1 or as 0.

   (a) What happens if we make Z a slightly noisy exclusive or of its parents? (i.e., in the CPD of $P(Z|X, Y)$, 0 is replaced with some small probability q and 1 with $1 - q$.) Calculate the expected number of iterations until state transition as a function of q. What can you conclude about problems that Gibbs sampling might encounter in its attempt to do a random walk?

   Answer: The probability to move from the current state is proportional to the noise parameter q. Each Gibbs step can be seen as a Bernoulli trial, so the first

success (transition) is distributed geometric with $O(\frac{1}{q})$ expected number of steps until transition.

(b) Alternatively, we can think of a variant of Gibbs sampling where larger steps are taken. Specifically, larger sets of variables are sampled simultaneously where the rest are fixed. Show that sampling two variables given the third (rest) overcomes the problem of Gibbs sampling in the XOR network (without noise).

(c) To use this last sampler we need to calculate $P(X_i, X_j | \mathcal{X} - \{X_i, X_j\})$. Write down the formula for this calculation in a general Markov Network.

Answer:

$$
\begin{aligned}
P(x_i, x_j | x_{-ij}) =& \frac{P(x_i, x_j, x_{-ij})}{P(x_{-ij})} = \frac{P(x_i, x_j, x_{-ij})}{\sum_{x'_i, x'_j} P(x'_i, x'_j, x_{-ij})} \\
=& \frac{\frac{1}{Z} \prod_c \varphi_c((x_i, x_j, x_{-ij})_c)}{\sum_{x'_i, x'_j} \frac{1}{Z} \prod_c \varphi_c((x'_i, x'_j, x_{-ij})_c)} \\
=& \frac{\prod_{c: i \in c \text{ or } j \in c} \varphi_c((x_i, x_j, x_{-ij})_c) \prod_{c': i \notin c' \text{ and } j \notin c'} \varphi_{c'}((x_{-ij})_{c'})}{\sum_{x'_i, x'_j} \prod_{c: i \in c \text{ or } j \in c} \varphi_c((x'_i, x'_j, x_{-ij})_c) \prod_{c': i \notin c' \text{ and } j \notin c'} \varphi_{c'}((x_{-ij})_{c'})} \\
=& \frac{\prod_{c: i \in c \text{ or } j \in c} \varphi_c((x_i, x_j, x_{-ij})_c)}{\sum_{x'_i, x'_j} \prod_{c: i \in c \text{ or } j \in c} \varphi_c((x'_i, x'_j, x_{-ij})_c)}
\end{aligned}
$$

So this is a local calculation involving only the factors that contain X_i or X_j.