CS395T: Learning Theory; Fall 2011

Lecture 3 and 4 ; Dates:08/31/2011 and 09/14/2011

Lecturer: Pradeep Ravikumar ; Scribe: Avhishek Chatterjee

Keywords: Graphical model, variable elimination, message passing, junction tree

Note: Scribed notes have only been lightly proofread.

1 Overview

These two lectures build the concept of inferencing over a graphical model. We shall first introduce the concept of marginalization using variable elimination. After that we shall show how variable elimination can be made efficient through message passing on a tree and general graphs. Then we shall introduce the concept of cluster graph for graphical models with larger (> 2) cliques. This leads to the development of the concept of junction tree and eventually to the junction tree algorithm. We end these lectures by pointing out some special features of the junction tree algorithm.

2 Variable Elimination

Unlike directed graphical models in an undirected graphical model the joint distribution (product of compatibility functions would be technically more correct term) does not normalize to one. Hence any inference problem over an undirected graphical model requires us to find the marginalization constant which is also called the *partition function*. Mathematically we can write it as, $Z = \sum_{\mathbf{X}} \prod_{C \in \mathcal{C}(G)} \psi_C(\mathbf{X}_C)$.

Please note that a brute force method for evaluating Z will require an exponential (in number of variables) number of addition operations. A natural and smarter alternative to brute force method would be *variable elimination*. In variable elimination the summation is done over one variable at a time. Let $\mathcal{C}_{i_1}$ be the set of cliques that contain X_{i_1} as a variable. Then given a chosen order of elimination say $\{i_1, i_2, \cdots i_n\}$, also called the *elimination schedule*, the variable elimination steps would be as the following.

$$\begin{aligned} Z &= \sum_{\mathbf{X}_{\setminus i_1}} \sum_{X_{i_1}} \prod_{C \in \mathcal{C}(G)} \psi_C(\mathbf{X}_C) \\ &= \sum_{\mathbf{X}_{\setminus i_1}} \prod_{C \in \mathcal{C}(G) \setminus \mathcal{C}_{i_1}} \psi_C(\mathbf{X}_C) \sum_{X_{i_1}} \prod_{C \in \mathcal{C}_{i_1}} \psi_C(\mathbf{X}_C) \\ &= \sum_{\mathbf{X}_{\setminus i_1}} M_{\mathcal{C}_{i_1} \setminus i_1}(\mathbf{X}_{(\cup_{C \in \mathcal{C}_{i_1}} C) \setminus i_1}) \prod_{C \in \mathcal{C}(G) \setminus \mathcal{C}_{i_1}} \psi_C(\mathbf{X}_C) \end{aligned}$$

Here $M_{\mathcal{C}_{i_1} \setminus i_1}(\mathbf{X}_{(\cup_{C \in \mathcal{C}_{i_1}} C) \setminus i_1})$ is the new compatibility function after eliminating X_{i_1} from the term $\prod_{C \in \mathcal{C}_{i_1}} \psi_C(\mathbf{X}_C)$. This term simply represent the compatibility function of new component formed after eliminating node i_1 and joining all the nodes that were connected to it.

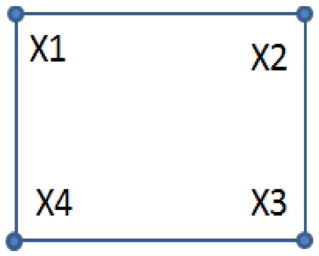

Figure 1: 4-node non-chordal cycle

Note that compared to brute force method variable elimination requires much less number of addition operations. This is due to the fact that in variable elimination while summing one has to only consider the variable nodes that are connected to the variable to be eliminated. This drastically reduces the number of operations. In fact the number of addition operations is polynomial in number of variables. This looks very promising and hence is worth of deeper study. Let us understand what variable elimination means over a graphical model by taking an example of 4-node non-chordal cycle as in Figure 1. We want to calculate

$$Z = \sum_{X_1,X_2,X_3,X_4} \psi_{12}(X_1,X_2)\psi_{23}(X_2,X_3)\psi_{34}(X_3,X_4)\psi_{14}(X_1,X_4)$$

Let us start with eliminating X_4. Then,

$$\begin{aligned} Z &= \sum_{X_1,X_2,X_3} \psi_{12}(X_1,X_2)\psi_{23}(X_2,X_3)\sum_{X_4}\psi_{34}(X_3,X_4)\psi_{14}(X_1,X_4) \\ &= \sum_{X_1,X_2,X_3} \psi_{12}(X_1,X_2)\psi_{23}(X_2,X_3)M_{13}(X_1,X_3) \\ &= \sum_{X_1,X_2} \psi_{12}(X_1,X_2)\sum_{X_3}\psi_{23}(X_2,X_3)M_{13}(X_1,X_3) \end{aligned}$$

where $M_{13} := \sum_{X_4} \psi_{34}(X_3,X_4)\psi_{14}(X_1,X_4)$. Observe that in each step of variable elimination while marginalizing over a variable we are also introducing a new function that will be used in next step. This new function will be a function of d variables where d is the degree of the eliminated variable node. As this new function will be used in future computations, we have to allocate additional storage for this function which is again exponential in d. So by doing variable elimination we might pay in memory (and computation) even though we eliminate one variable at a time.

Intuitively it may appear that an elimination schedule that marginalize over lower degree nodes first would save lot of memory. This is true in some graphs (optimal for tree) but not true in general.

Fact 1 (i) *Largest size (minus one) of the remnant factor over all elimination orders is called the* tree-width*. The worst case memory requirement for storing the intermediate factors $M_{i_1,i_2,\cdots i_k}$ is exponential in* tree-width*.*

(ii) *Finding the optimal elimination order is NP-hard.*

3 Message Passing on Trees

Message passing on trees is a way to do marginalization on graphical models that have a tree structure. We mentioned that using greedy method for variable elimination is an efficient way to compute marginals over a tree. Then the natural question that arises is what do we get using message passing. The answer is that the message passing technique that we will build in this section can also be applied efficiently to other graphical models while keeping the computation and memory requirements practical.

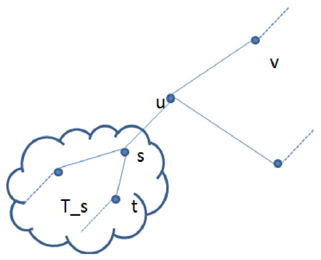

Figure 2: Message Passing Tree

Consider the tree in Figure 2. Let us call the tree $G = (V, E)$. As edges are the only cliques in a tree, we can write its joint distribution $\mathsf{P}(\mathbf{X})$ as,

$$\mathsf{P}(\mathbf{X}) \propto \prod_{s \in V} \psi_s(X_s) \prod_{(s,t) \in E} \psi_{st}(X_{st})$$

Here we consider the nodes to form singleton cliques and this representation is handy in some problems. Now note that if we want to find the marginal of X_u say $\mathsf{P}(X_u = x_u)$, then we must marginalize over all nodes other than u. Note that for any node s in neighborhood of u, say $N(u)$, we can think of a tree T_s rooted at s and by the definition of a tree $T_s \cap T_v = \phi$ if $s \neq v \in N(u)$. Hence we can marginalize over each sub-tree of neighbors of u separately. This is the key idea behind message passing on a tree. Also note that this marginalization can be done recursively. This will be clear when we follow the steps mathematically.

Note that we can not compute $\mathsf{P}(X_u = x_u)$ without computing the partition function. So we compute a quantity $\mu_u(x_u)$ which is proportional to the marginal distribution which can be efficiently used to calculate the marginal distribution. We define $\bar{\mathsf{P}}_T(X_T)$ is the restricted distribution of $\mathsf{P}()$ over the sub-tree T. $\bar{\mathsf{P}}_{T_s}(X_{T_s}) := \prod_{t \in T_s} \psi_t(X_t) \prod_{(t,r) \in E_{T_s}} \psi_{tr}(X_{tr})$.

$$\begin{aligned}
\mu_u(x_u) &= \sum_{\mathbf{X}: X_u = x_u} \prod_{s \in V} \psi_s(X_s) \prod_{(s,t) \in E} \psi_{st}(X_{st}) \\
&= \psi_u(x_u) \sum_{\mathbf{X}: X_u = x_u} \prod_{s \in N(u)} \left(\psi_{us}(x_u, X_s) \bar{\mathsf{P}}_{T_s}(X_{T_s})\right) && (1) \\
&= \psi_u(x_u) \prod_{s \in N(u)} \left(\sum_{X_{T_s}} \psi_{us}(x_u, X_s) \bar{\mathsf{P}}_{T_s}(X_{T_s}) \right) && (2)
\end{aligned}$$

Note that the equation 1 follows from the definition of the restricted distribution. Again the graph is a tree, so $T_s \cap T_v = \phi$ for $s, v \in N(u)$ and $s \neq v$. Hence the summation and the product can be interchanged in the equation 2.

Let us define $M_{ij}(X_j) := \sum_{X_{T_i}} \psi_{ji}(X_j, X_i) \bar{\mathsf{P}}_{T_i}$. With this new term defined, we can represent the variable elimination by the following simple update algorithm between the nodes of the tree.

$$\mu_u(X_u) = \psi_u(X_u) \prod_{s \in N(u)} M_{su}(X_u)$$

$$M_{su}(X_u) = \sum_{X_s} \left(\psi_{su}(X_{su}) \psi_s(X_s) \prod_{t \in N(s) \setminus u} M_{ts}(X_{ts}) \right)$$

Note that we have shown these update equations to be same as variable elimination for a tree. Hence it is implicit that the message passing on a tree that uses this update algorithm must converge to the true marginal.

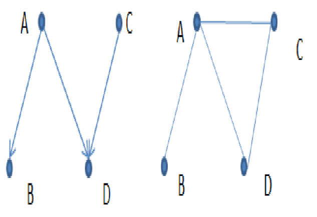

Figure 3: Moralization

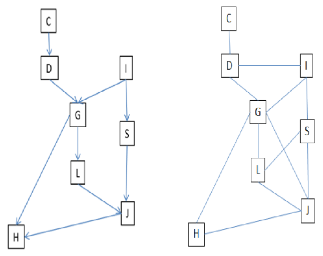

Figure 4: Extended Student Graph

Also note that the update equations make sense even for non-tree graph as the algorithm does not use any other graph structure than the concept of neighbouring nodes. For general graph this algorithm is called the *sum-product algorithm*. In fact sum-product algorithm also converges to the true marginal for some non-tree graphs. But note that sum-product algorithm as stated above only uses 2-cliques, but can be generalized to the case with higher order d-cliques where $d > 2$.

4 Moralization

Before directly delving into the *junction tree algorithm* which is a generalization of the message passing algorithm applied to general graphs, we shall take a necessary detour. Note that we have described the inference/marginalization problem for undirected graphical model so far. The question is now, how do we solve the same problems over a directed acyclic graph. It turns out that we can achieve that simply by performing an intermediate step called *moralization*. The idea behind moralization is simple and we shall explain it using the Figure 3. Distribution of the DAG in figure is given by $P(B)P(D)P(A|B)P(C|BD)$. This can be written as, $\psi(B)\psi(D)\psi(A,B)\psi(B,C,D)$ where $\psi(B) = P(B)$, $\psi(D) = P(D)$, $\psi(A,B) = P(A|B)$ and $\psi(B,C,D) = P(C|BD)$. Note that this factorization represents the undirected graph (right) in Figure 3. So it follows from this example that by connecting parents of a node and erasing the direction, a distributin on a DAG can be represented as a distribution over an undirected graphical model. This is called moralization of a DAG. So the steps for moralization are *(i)* making the edges un-directed and *(ii)* connecting the parents of a node to form a clique. After moralization of a graph marginal distribution can be found similarly like an undirected graph.

5 Junction Tree Algorithm

Let us start our study of junction tree or clique graph with the example of a graph from the extended student family. The directed and the undirected graphs in Figure 4 correspond to an extended student graph and its moralized version respectively. Suppose we want to find marginal over the variable J by carrying out variable elimination with a schedule $C - D - I - H - G - S$. In first step we do $(\sum_C \psi_{CD}(X_C, X_D))$. The summation results into a function of X_D. Note that D, I, G form a clique and the function gets absorbed as a component

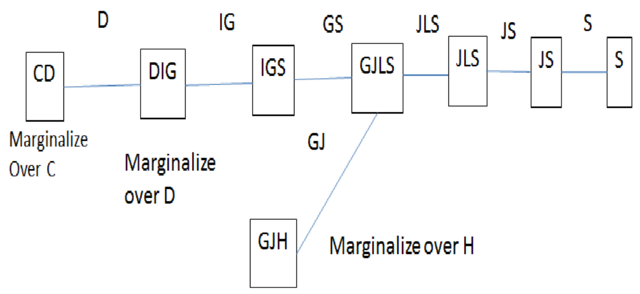

Figure 5: Junction Tree for Extended Student Graph

of the clique compatibility function. Then we marginalize over D and obtain a function of I and G. We absorb this function into the component IGS and marginalize out G and S from that. These steps can be graphically presented as in Figure 5. This graph is a special type of graph with multiple specific characterizations. *(i)* This is a tree and nodes that have edge between them must have common variables (reverse is not true), *(ii)* the variables that are carried from one node to another are the common variables and others are eliminated, and *(iii)* If a variable belongs to two nodes then it belongs to all nodes on the path connecting them (e.g. S belongs to IGS and JS, it also belongs to $GJLS$ and JLS. Indeed these properties are very important and a graph with these properties are called *clique or junction tree*.

Definition 1 *Cluster Graph: Given a set of factors Ψ (compatibility function) over V, a cluster graph has set of nodes $\mathcal{V}$ such that*
(i) any node $i \in V$ is associated with a cluster of nodes $C_i \in \mathcal{V}$.
(ii) two clusters C_i and C_j have an edge between them only if $S_{ij} = C_i \cap C_j \neq \phi$
(iii) Family Preserving Property: For any $\psi \in \Psi$ there is a unique $C_i \in \mathcal{V}$ such that scope of ψ is a subset of C_i. In other words a factor (compatibility function) can belong to only one cluster (This is to make sure the joint distribution over clique graph is consistent).

Note that the tree formed in the example with extended student graph is a cluster graph.

Proposition 5.1 *The cluster graph obtained from variable elimination is a tree.*

Proof: Because VE moves by eliminating variables, any node after say node $(stuv)$ that was reached by eliminating v will not have v. As the process gradually eliminates variables it can not pick up v which is already eliminated. Hence there is no way to get back to $(stuv)$. □

Definition 2 *Running Intersection Property: Let Y be a cluster tree. Let $X \in V$ and $X \in C_i$ and $X \in C_j$. Then $X \in C$ for all C on the path between C_i and C_j.*

Proposition 5.2 *Any cluster graph obtained through variable elimination will satisfy running intersection property.*

Proof: Let X be common to both C_i and C_j. Further, suppose X gets eliminated at C_X : this has to occur after C_i and C_j. It can be seen that all cliques between C_i and C_X (and similarly, between C_j and C_X) will contain X since it is eliminated only at C_X. □

Definition 3 *Clique or Junction Tree: A cluster graph that is tree and satisfies running intersection property is called a clique or junction tree.*

Note that this is true for the cluster graph formed by variable elimination on extended student graph. In general it is not always true for any cluster graph.

Let us have a look at how the variable elimination is done in Figure 5. Let us number the clusters CD, DIG, $IGS \cdots$ as C_1, C_2, $C_3 \cdots$. We define M_{ij} to be the message sent from C_i to C_j after eliminating variables $C_i \backslash C_j$. Also note that M_{ij} is a function of $S_{ij} = C_i \cap C_j$. In the example, $\psi_1(C_1)$ is the compatibility function for the component C_1 that have X_C and X_D. $M_{12}(C_1 \cap C_2) = \sum_{C_1 \backslash C_2} \psi_1(X_{C_1})$. Similarly, we can write M_{23} as, $\sum_{C_2 \backslash C_3} M_{12}\psi_2$ and so on. Note that the basic idea is that cluster i sends a message M_{ij} to the cluster j which is the marginalization of the product of the all the messages it received from other neighbors (other than j) and its own component function. Marginalization is done over all the variables of C_i that are not present in C_j. This basic idea leads to the following famous algorithm for junction trees, called the *junction tree algorithm*.

For some schedule over edges (i, j):
$M_{ij}(S_{ij}) = \sum_{C_i \backslash S_{ij}} \psi_i(C_i) \prod_{k \in N(i) \backslash j} M_{ki}(S_{ki})$.

Finally: Set $\mu_i(C_i) = \psi_i(C_i) \prod_{k \in N(i)} M_{ki}(S_{ki})$

5.1 Properties of Junction Tree Algorithm

Theorem 5.1 *When the Junction Tree algorithm converges, $\mu_i(C_i)$ as computed above satisfies: $\mu_i(C_i) \propto P(X_{C_i})$.*

If junction tree algorithm converges to a true marginal then the following must also be true

$$\sum_{C_i \backslash S_{ij}} \mu_i(C_i) = \sum_{C_j \backslash S_{ij}} \mu_j(C_j) \tag{3}$$

This is true because both give the marginal distribution over S_{ij}. Note that this equality holds after the JTA has converged to the true marginal, and need not necessarily hold at every iteration of the JTA.

When the equation (3) does holds for an edge (i, j) of the junction tree, then the edge (i, j) is said to be *calibrated*. A junction tree is called calibrated if all of its edges are calibrated. Also observe the following nice equality,

$$\begin{aligned} \mu_{ij}(S_{ij}) &= \sum_{C_i \backslash S_{ij}} \mu_i(C_i) \\ &= \sum_{C_i \backslash S_{ij}} \psi_i(C_i) \prod_{k \in N(i)} M_{ki}(S_{ki}) \\ &= M_{ji}(S_{ji}) \sum_{C_i \backslash S_{ij}} \psi_i(C_i) \prod_{k \in N(i) \backslash j} M_{ki}(S_{ki}) \\ &= M_{ji}(S_{ji}) M_{ij}(S_{ij}) \end{aligned} \tag{4}$$

In the equation 4 we have used the fact that $C_i \backslash S_{ij}$ and S_{ji} have no variables in common.