CS281A/Stat241A: Statistical Learning Theory

Junction Tree Algorithm

Lecturer: Michael I. Jordan *Scribes: Milch and Nadjakova*

So far, this class has presented a unified framework for some classical statistical models and algorithms. The junction tree algorithm performs inference on general graphical models, giving us a way to attack problems that the classical models don't handle.

1 The Inference Problem

Goal: Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, compute $p(x_F \mid x_E)$, where $\mathcal{V} = E \cup H$ and F is any clique or subset of a clique in H.

Today we will be assuming $\mathcal{G}$ is undirected; to handle a directed graph, we just need to moralize it to obtain an undirected graph. Why are we interested in computing the posterior over a clique? We can write:

$$p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

where $\mathcal{C}$ is the set of cliques in $\mathcal{G}$. Based on this equation, we can view $p(x)$ as an exponential family distribution where the sufficient statistics are functions over cliques. For example, in an HMM, the sufficient statistics include the $\xi_{t,t+1}$ variables defined over two-node cliques. So if we can compute posteriors over cliques, we can use the EM algorithm to do parameter estimation. Note that computing the posterior over a single node is just a special case.

With the junction tree algorithm, we now have three algorithms for computing posterior probabilities in graphical models:

1. **Elimination**: for arbitrary graphs; requires F to be a singleton set
2. **Sum-product**: for trees; computes posteriors for all singleton F's simultaneously
3. **Junction tree**: for arbitrary graphs; computes posteriors for all F's (cliques) simultaneously

2 Combining elimination with caching

We can view the junction tree algorithm as a combination of elimination and the sum-product algorithm. Consider how the sum-product algorithm works on the graph in Figure 1. To compute the posterior over X_1, we pass messages m_{32}, m_{42}, and then m_{21}. The key insight of the sum-product algorithm is that to compute the posterior over another node, say X_2, we can reuse some of the messages — in this case, we can reuse m_{32} and m_{42}, so we just have to compute m_{12}. Thus, we can compute all the single-node posteriors by passing just one message in each direction per edge.

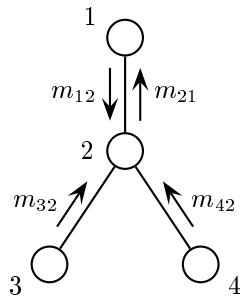


Figure 1: The sum-product algorithm on a small graph.

How can we combine this idea of reuse (or caching) with the elimination algorithm, to handle arbitrary graphs? Consider the general graph in Figure 2(a). To compute $p(x_1)$, we might begin by eliminating X_6: that is, combining the clique potentials $\psi(x_4, x_6)$ and $\psi(x_5, x_6)$ to create an intermediate term:

$$m_6(x_4, x_5) = \sum_{x_6} \psi(x_4, x_6)\psi(x_5, x_6)$$

The resulting potential is over $\{X_4, X_5\}$; we illustrate this by adding an edge between X_4 and X_5 in the graph. Then when we eliminate X_5, we get a term $m_5(x_2, x_4)$, so we add an edge (X_2, X_4). The resulting *triangulated graph* is shown in Figure 2(b). Note that our intermediate terms are over cliques in the triangulated graph. We should be able to reuse these terms — as long as we use the same elimination ordering to compute all posteriors.

This idea motivates drawing a tree whose nodes are the cliques of Figure 2(b). This *clique tree* is shown in Figure 3(a). We can view elimination as operating on this clique tree, with the intermediate terms being messages passed along the edges (as shown in Figure 3(a)). As we pass messages, we update potentials on the cliques.

Elimination just passes messages in one direction. But if we have already computed the posterior over X_1, we can compute the posterior over X_5 by passing more messages in the same tree. In general, we can get the posterior over any clique after passing just one message in each direction per edge. This is essentially the junction tree algorithm (although we have not yet specified exactly what the messages are).

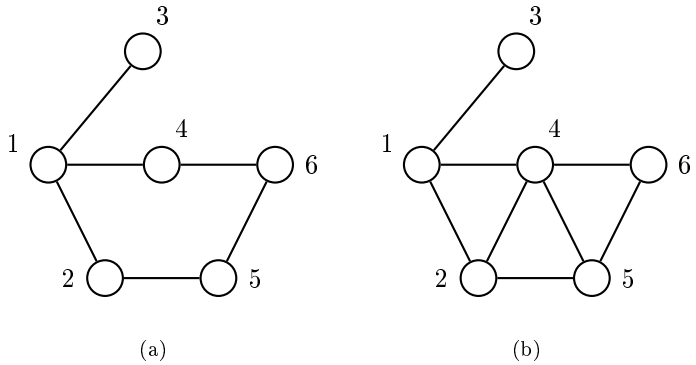


Figure 2: A general graph (a) and its triangulated version (b).

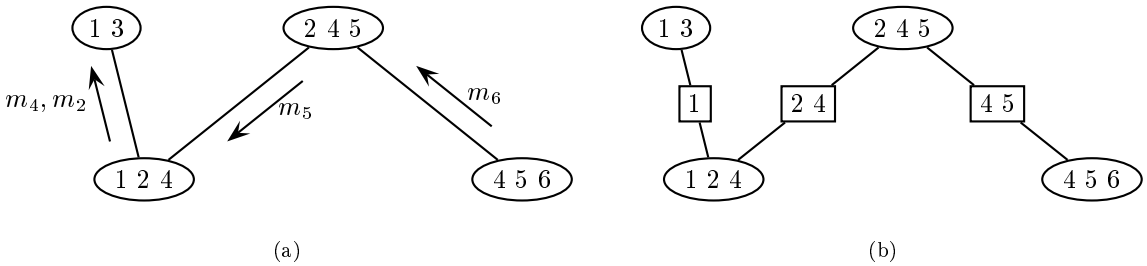


Figure 3: A clique tree for the graph of Figure 2(b). The messages generated in the elimination algorithm are shown in (a), and separator sets are shown as rectangular nodes in (b).

Note that in the clique tree of Figure 3(a), the subgraph picked out by any given node (e.g., 4) is connected. This means the tree is a *junction tree*. The junction tree algorithm will pass messages so as to achieve *local consistency*: the potentials on each pair of adjacent cliques will agree on the variables in their intersection. We will then show that if a clique tree is a junction tree and is locally consistent, then the clique potentials are in fact the posterior probabilities of the cliques.

3 Probability distributions on clique trees

For any pair of cliques V, W that are adjacent in a clique tree, their *separator set* is $S \triangleq V \cap W$. We will find it convenient to draw the separator sets explicitly in a clique tree, as in Figure 3(b). Then we define the probability distribution represented by this clique tree as:

$$p(x) = \frac{\prod_{C \in \mathcal{C}} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)}$$

where ϕ_S is the *separator potential* for S. By convention, $\mathcal{S}$ includes the empty set $\{\}$, and $\phi_{\{\}} = Z$ (the normalizing constant).

Clique trees are a new kind of graphical model, like directed graphs, undirected graphs, and factor graphs. Why are we introducing them? Consider the directed graph in Figure 4(a), which represents a distribution $p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)$. If we want to represent the same distribution with the undirected graph in Figure 4(b), we can let:

$$\begin{aligned} \psi(x_1, x_2) &= p(x_1)p(x_2 \mid x_1) \\ \psi(x_2, x_3) &= p(x_3 \mid x_2) \end{aligned}$$

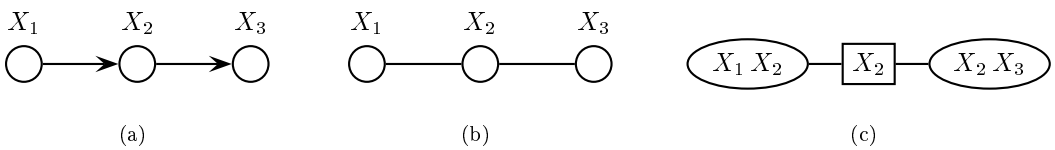


Figure 4: Three graphical models over the variables X_1, X_2, X_3.

Note that $\psi(x_1, x_2) = p(x_1, x_2)$, a marginal probability. But $\psi(x_2, x_3)$ is not a marginal probability. In general, we cannot write $p(x) = p(x_1, x_2)p(x_2, x_3)$. Thus, we cannot have $\psi(x_1, x_2)$ and $\psi(x_2, x_3)$ both be marginal probabilities simultaneously.

But if we use a clique tree, as in Figure 4(c), we can use the following potentials:

$$\begin{aligned} \psi(x_1, x_2) &= p(x_1, x_2) \\ \psi(x_2, x_3) &= p(x_2, x_3) \\ \phi(x_2) &= p(x_2) \end{aligned}$$

Thus:

$$p(x) = \frac{\psi(x_1, x_2)\psi(x_2, x_3)}{\phi(x_2)} = \frac{p(x_1, x_2)p(x_2, x_3)}{p(x_2)} = p(x_1, x_2)p(x_3 \mid x_2)$$

So we can represent the desired distribution with both $\psi(x_1, x_2)$ and $\psi(x_2, x_3)$ being marginal probabilities.

Note that we do not lose any generality by using a clique tree as opposed to an undirected graphical model: we can simply set $\phi_S = 1$ for $S \neq \{\}$. We do not gain anything either: since each S is a subset of some clique (in fact, at least two cliques), we can simply divide some clique potential by ϕ_S for each S, and recover an undirected graph representation. So the families of distributions representable by clique trees are the same as those representable by undirected graphical models (but the potentials in clique trees may be more convenient).

4 Message-Passing Algorithm

Here is an outline of the message-passing procedure that is the heart of the junction tree algorithm:

- Given a clique tree, initialize $\psi_C(x_C)$ to be the potentials from the original graph, and $\phi_S(x_S) = 1$
- Pass messages that leave $p(x)$ invariant
- If the clique tree is a junction tree (see below), the potentials ψ_C will become marginal probabilities:

$$\psi_C(x_C) = p(x_C)$$

It is remarkable that (given a junction tree) we can start with arbitrary clique potentials, which have no local probabilistic semantics, and end up with (local) marginal probabilities for the cliques.

More specifically, what messages do we pass between cliques? Recall that our first goal is to achieve local consistency, which is clearly a precondition for the clique potentials being marginal probabilities. Consider a pair of adjacent cliques V and W, as shown in Figure 5. We update the clique and separator potentials via the following equations:

$$\begin{aligned} \phi^*_S(x_S) &= \sum_{x_{V \setminus S}} \psi_V(x_V) \\ \psi^*_W(x_W) &= \psi_W(x_W)\frac{\phi^*_S(x_S)}{\phi_S(x_S)} \end{aligned}$$

We refer to this update as *sending a message from V to W*. ψ_V is not changed in this update, so we write $\psi^*_V(x_V) = \psi_V(x_V)$. Next, we do an analogous update in the other direction, using the same equations with V and W reversed.

For example, if $V = \{1, 2, 5\}$ and $W = \{2, 5, 6\}$, then we begin by summing over x_1 to get a 2-D table $\phi^*_S(x_2, x_5)$. Then we divide this table pointwise by $\phi_S(x_2, x_5)$. Finally, we multiply the resulting table into $\psi_W(x_W)$ to get $\psi^*_W(x_W)$.

5 Some Properties of the Algorithm

- **The joint is invariant**

  With each step of the algorithm, we are changing the values of some local potentials. We have to make sure that these changes don't affect the joint probability. Recall that we defined

  $$p(x) = \frac{\prod_{C \in \mathcal{C}} \psi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)}$$

  The only components of the above formula that change with each step are ψ_V, ψ_W and ϕ_S. So it is sufficient to show the following:

  $$\frac{\psi_V^*(x_V) \cdot \psi_W^*(x_W)}{\phi_S^*(x_S)} = \frac{\psi_V(x_V) \cdot \psi_W(x_W) \cdot \phi_S^*(x_S)}{\phi_S(x_S) \cdot \phi_S^*(x_S)} = \frac{\psi_V(x_V) \cdot \psi_W(x_W)}{\phi_S(x_S)}$$

  The first equality is obtained by replacing ψ_V^* and ψ_W^* with their update definitions. The second equality is obtained by cancelling out ϕ_S^*.

- **Local consistency**

  Our functions ψ and ϕ are going to represent marginal probabilities on their corresponding cliques. That means that they have to be consistent with each other: when two cliques overlap, they must give the same marginal for the nodes they have in common. Here we are going to show that the algorithm has the property of local consistency - two cliques that are *adjacent* in the tree are consistent. (What we really want is global consistency. Depending on how we construct our clique tree local consistency may or may not be sufficient. If we arrange the cliques in a special kind of tree - *a junction tree* - local consistency will turn out to be sufficient. More details in next lecture.)

  Let's first write out the update equations in the other direction - from W to V:

  $$\begin{aligned} \phi_S^{**}(x_S) &= \sum_{x_{W \setminus S}} \psi_W^*(x_W) \\ \psi_V^{**}(x_V) &= \psi_V^*(x_V) \frac{\phi_S^{**}(x_S)}{\phi_S^*(x_S)} \\ \psi_W^{**}(x_W) &= \psi_W^*(x_W) \end{aligned}$$

  We want our marginals to be the same, so we want to show that

  $$\sum_{x_{V \setminus S}} \psi_V^{**}(x_V) = \sum_{x_{W \setminus S}} \psi_W^{**}(x_W)$$

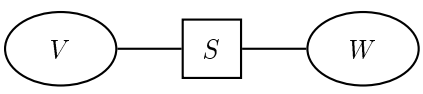


Figure 5: A pair of cliques and their separator.

Here is the derivation:

$$\begin{aligned}
\sum_{x_{V\setminus S}} \psi_V^{**}(x_V) &= \sum_{x_{V\setminus S}} \psi_V^{*}(x_V)\frac{\phi_S^{**}(x_S)}{\phi_S^{*}(x_S)} \\
&= \frac{\phi_S^{**}(x_S)}{\phi_S^{*}(x_S)} \cdot \sum_{x_{V\setminus S}} \psi_V^{*}(x_V) \\
&= \frac{\phi_S^{**}(x_S)}{\phi_S^{*}(x_S)} \cdot \sum_{x_{V\setminus S}} \psi_V(x_V) \\
&= \frac{\phi_S^{**}(x_S)}{\phi_S^{*}(x_S)} \cdot \phi_S^{*}(x_S) \\
&= \phi_S^{**}(x_S) \\
&= \sum_{x_{W\setminus S}} \psi_W^{*}(x_W) \\
&= \sum_{x_{W\setminus S}} \psi_W^{**}(x_W)
\end{aligned}$$

We obtained the second equation by pulling $\frac{\phi_S^{**}(x_S)}{\phi_S^{*}(x_S)}$ in front of the sum - we can do that since the set of variables x_S is disjoint from the set of variables $x_{V\setminus S}$ we are summing over. All the other equations are obtained just by replacing the definitions of ψ or ϕ.

6 Message-Passing Protocol

When we have a tree with more than two cliques in it, we have to take care not to ruin the local consistency of two cliques with updates of between them and their neighbors. For that we define a

Message-Passing Protocol: A clique can send a message to a neighboring clique only when it has received messages from all of its other neighbors.

The proof that this protocol yields local consistency is done in the chapter.

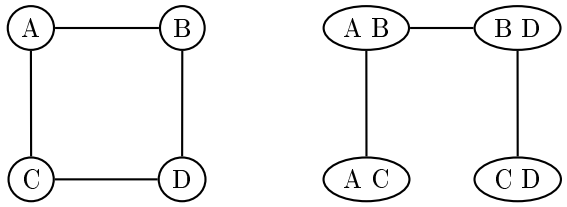
7 Junction Trees

There are two questions that we yet have to answer. One is how do we ensure global consistency? And the other is whether global consistency implies that our functions ψ and ϕ are actually marginals?

Lets start with the first question. Local consistency does not, in general, imply global consistency. Consider the graph in Figure 6(a). One possible clique tree is shown in Figure 6(b). After running our algorithm, we know that we have local consistency between the cliqes AC and AB since they are adjacent. Same for cliques AB and BD; BD and CD. But the cliques AC and CD also have a node in common - how do we know that those two would be consistent?

It turns out that there is a special kind of clique tree - the junction tree - for which local consistency is not only necessary but also sufficient for having global consistency. Here is the definition of a junction tree:

Junction Tree Property: A clique has the *junction tree property* if and only if for every pair of cliques V and W, all cliques on the unique path from V to W contain $V \cap W$.



(a) (b)

Figure 6: A graph and a corresponding clique tree

Unfortunately, not every graph can be directly converted to a junction tree. Consider again the graph in Figure 6(a): just going through all the cases of connecting the corresponding cliques should reveal that there is no way to connect those cliques so that the junction tree property would hold. The solution to this is to triangulate the graph - add an edge from B to C (or from A to D) and build the graph for the two cliques ABC and BCD. It turns out that this approach works in the general case:

Theorem: A graph has a junction tree if and only if it is triangulated.

One way to define what "triangulated" graph means is to say that it is the result of running the elimination algorithm on some graph. Another is to define it though *chordless cycles*:

Definition: A cycle in an undirected graph is *chordless* if there are no edges between nodes that are not neighbors in the cycle.

Definition: A graph is *triangulated* if it contains no chordless cycles.

For example the graph in Figure 7 is not triangulated - the cycle $B - F - H - D$ is not chordless. Adding an edge between B and H however yields a triangulated graph. You can check that by running elimination in the order A, C, G, I, D, F, B, E, H.

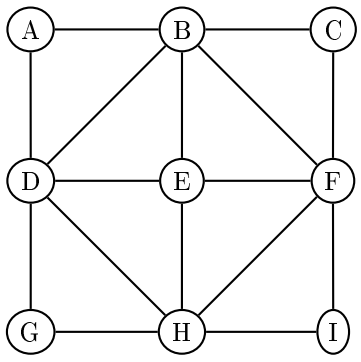


Figure 7: An example of non-triangulated graph