14 The Junction Tree Algorithm

In our development thus far, we have looked at exact inference on general undirected graphical models using the elimination algorithm, and then trees over discrete random variables using sum-product and max-product. Specialized to hidden Markov models, we related sum-product to the forward-backward algorithm, max-product to the Viterbi algorithm.

We now venture back to general undirected graphs potentially with loops and again ask how to do exact inference. Our focus will be on computing marginal distributions. To this end, we could just run the elimination algorithm to obtain the marginal at a single node and then repeat this for all other nodes in the graph to find all the marginals. But as in the sum-product algorithm, it seems that we should be able to recycle some intermediate calculations. Today, we provide a method to do this bookkeeping with an algorithm that does exact inference on general undirected graphs, referred to as the *junction tree algorithm*.

At a high-level, the basic idea of the junction tree algorithm is to convert the input graph into a tree and then apply sum-product. While this may seem too good to be true, alas, there's no free lunch and the fineprint is that the nodes in the new tree may have alphabet sizes substantially larger than those of the original graph! In particular, we require the trees to be what are called *junction trees*.

14.1 Junction Trees

The idea behind junction trees is that certain probability distributions corresponding to possibly loopy undirected graphs can be reparameterized as trees, enabling us to run sum-product on this tree reparameterization and rest assured that we extract exact marginals.

We begin with the following definitions.

Definition 1 (Clique Graph)**.** *Given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a* clique graph *of $\mathcal{G}$ is a graph where the set of nodes is precisely the set of maximal cliques in $\mathcal{G}$.*

Definition 2 (Clique Tree)**.** *Any clique graph that is a tree is a* clique tree.

For example, Fig. 1(b) and Fig. 1(c) show two clique trees for the graph in Fig. 1(a). Note that if each x_i takes on a value in $\mathcal{X}$, then each clique node in the example clique trees corresponds to a random variable that takes on $\mathcal{X}^3$ values since all the maximal clique sizes are 3. In other words, clique nodes are supernodes of multiple random variables in the original graph.

We aim to give an alternative description of the distribution on some arbitrary undirected graphical model by specifying a distribution on a clique tree. Intuitively,

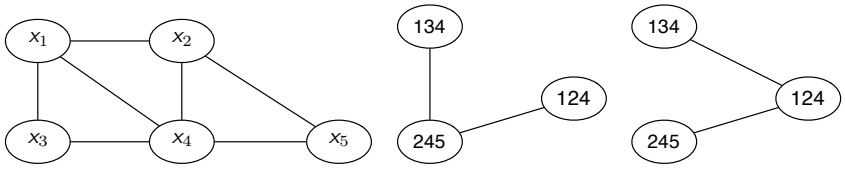

Figure 1: (a) Original graph. (b)-(c) Clique trees for the original graph in (a).

the clique tree in Fig. 1(b) seems inadequate for describing the underlying distribution because maximal cliques $\{1, 3, 4\}$ and $\{1, 2, 4\}$ both share node 1 but there is no way to encode a constraint that says that the x_1 value that these two clique nodes take on must be the same; we can only put constraints as edge potentials for edges $(\{1, 3, 4\}, \{2, 4, 5\})$ and $(\{2, 4, 5\}, \{1, 2, 4\})$. Indeed, what we need is that the maximal cliques that node 1 participates in must form a *connected* subtree, meaning that if we delete all other maximal cliques that do not involve node 1, then what we are left with should be a connected tree.

With this guiding intuition, we make the following definition.

Definition 3 (Junction Tree Property)**.** *Let $\mathcal{C}_v$ denote the set of all maximal cliques in the original graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that contain node v. Then we say that a clique tree for $\mathcal{G}$ satisfies the* **junction tree property** *if for each $v \in \mathcal{V}$, the set of nodes $\mathcal{C}_v$ in the clique tree are still connected when all the other nodes (and the edges incident to them) are removed. Finally, we define a* **junction tree** *to be a clique tree that satisfies the junction tree property.*

For example, the clique tree in Fig. 1(c) satisfies the junction tree property and is therefore a junction tree.

With our previous intuition for why we need the junction tree property, we can now specify constraints on shared nodes via pairwise potentials. We will first work off our example before discussing the general case.

The original graph in Fig. 1(a) has corresponding factorization

$$p_{x_1,x_2,x_3,x_4,x_5}(x_1, x_2, x_3, x_4, x_5) \propto \psi_{134}(x_1, x_3, x_4)\psi_{124}(x_1, x_2, x_4)\psi_{245}(x_2, x_4, x_5), \quad (1)$$

whereas the junction tree in Fig. 1(c) has corresponding factorization

$$\begin{aligned} &p_{y_{134},y_{124},y_{245}}(y_{134}, y_{124}, y_{245}) \\ &\quad \propto \phi_{134}(y_{134})\phi_{124}(y_{124})\phi_{245}(y_{245})\psi_{134,124}(y_{134}, y_{124})\psi_{124,245}(y_{124}, y_{245}). \end{aligned} \quad (2)$$

For notation, we'll denote $[y_V]_S$ to refer to the value that y_V assigns to nodes in S. For example, $[y_{134}]_3$ refers to the value that $y_{134} \in \mathcal{X}^3$ assigns to node 3. Then note

that we can equate distributions (1) and (2) by assigning the following potentials:

$$\begin{aligned}
\phi_{134}(y_{134}) &= \psi_{134}([y_{134}]_1, [y_{134}]_3, [y_{134}]_4) \\
\phi_{124}(y_{124}) &= \psi_{124}([y_{124}]_1, [y_{124}]_2, [y_{124}]_4) \\
\phi_{245}(y_{245}) &= \psi_{245}([y_{245}]_2, [y_{245}]_4, [y_{245}]_5) \\
\psi_{134,124}(y_{134}, y_{124}) &= \mathbb{1}_{[y_{134}]_1=[y_{124}]_1} \mathbb{1}_{[y_{134}]_4=[y_{124}]_4} \\
\psi_{124,245}(y_{124}, y_{245}) &= \mathbb{1}_{[y_{124}]_2=[y_{245}]_2} \mathbb{1}_{[y_{124}]_4=[y_{245}]_4}
\end{aligned}$$

The edge potentials constrain shared nodes to take on the same value, and once this consistency is ensured, the rest of the potentials are just the original clique potentials.

In the general case, consider a distribution for graph $\mathcal{G}$ given by

$$p_{\mathbf{x}}(\mathbf{x}) \propto \prod_{C \in \mathcal{C}} \phi_C(x_C), \tag{3}$$

where $\mathcal{C}$ is the set of maximal cliques in $\mathcal{G}$. Then a junction tree $\mathcal{T} = (\mathcal{C}, \mathcal{F})$ for $\mathcal{G}$ has factorization

$$p_{\mathbf{y}}(\{y_C : C \in \mathcal{C}\}) \propto \prod_{C \in \mathcal{C}} \phi_C(y_C) \prod_{U,W \in \mathcal{F}} \psi_{V,W}(y_V, y_W), \tag{4}$$

where edge potential $\psi_{V,W}$ is given by

$$\psi_{V,W}(y_V, y_W) = \mathbb{1}_{[y_V]_S=[y_W]_S} \qquad \text{for } S = V \cap W.$$

Comparing (3) with (1) and (4) with (2), note that importantly, y_C's are values over cliques. In particular, for $V, W \in \mathcal{C}$ with $S = V \cap W \neq \varnothing$, we can plug into the joint distribution (4) values for y_V and y_W that disagree over shared nodes in S; of course, the probability of such an assignment will just be 0.

With the singleton and edge potentials defined, applying sum-product directly yields marginals for all the maximal cliques. However, we wanted marginals for nodes in the original graph, not marginals over maximal cliques! However, we can extract node marginals from maximal-clique marginals by simply summing out all but one of the variables in a maximal clique's marginal distribution. Of course, such a summation is expensive if the clique under consideration is large.

So we see that junction trees are useful as we can apply sum-product on them along with some additional marginalization to compute all the node marginals. But a few key questions arise that demand answers:

1. Which undirected graphical models have junction trees?
2. For a graph that has a junction tree, even if we know a junction tree exists for it, how do we actually find it?
3. For a graph that does not have a junction tree, can we modify it so that it does have a junction tree?

We'll answer these in turn, in the process obtaining the full junction tree algorithm.

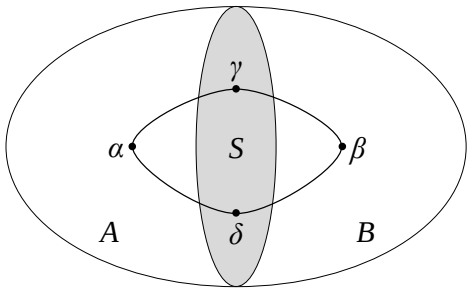

Figure 2: Diagram to help explain the proof of Lemma 1.

14.2 Existence of Junction Trees

It turns out that there is a simple characterization of graphs with junction trees.

Theorem 1. *If a graph is chordal, then it has a junction tree.*

In fact, the converse is also true: If a graph has a junction tree, then it must be chordal. We won't need this direction though and will only prove the forward direction. First, we collect a couple of lemmas.

Lemma 1. *If graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is chordal, has at least three nodes, and is not fully connected, then $\mathcal{V} = \mathcal{A} \cup \mathcal{B} \cup \mathcal{S}$ where: (i) sets $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{S}$ are disjoint, (ii) $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$, and (iii) $\mathcal{S}$ is a clique (i.e., $\mathcal{S}$ is fully connected).*

Proof. Refer to Fig. 2. Nodes $\alpha, \beta \in \mathcal{V}$ are chosen to be nonadjacent (requiring the graph to have at least three nodes and not be fully connected), and $\mathcal{S}$ is chosen to be a minimal set of nodes such that any path from α to β passes through $\mathcal{S}$ (i.e., removing any node from $\mathcal{S}$ makes this no longer true). Define $\mathcal{A}$ to be the set of nodes reachable from α with $\mathcal{S}$ removed, and define $\mathcal{B}$ to be the set of nodes reachable from β with $\mathcal{S}$ removed. By construction, $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ and $\mathcal{V} = \mathcal{A} \cup \mathcal{B} \cup \mathcal{S}$ where $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{S}$ are disjoint. It remains to show (*iii*).

Let $\gamma, \delta \in \mathcal{S}$. By minimality of $\mathcal{S}$ these nodes cannot be removed and hence there must be a path from α to γ and also a path from α to δ, implying existence of a path from γ to δ in $\mathcal{A} \cup \mathcal{S}$. Similarly, there must be a path from γ to δ in $\mathcal{B} \cup \mathcal{S}$.

Suppose for the sake of deriving a contradiction that there is no edge from γ to δ. Consider a shortest path from γ to δ in $\mathcal{A} \cup \mathcal{S}$ that contains at least one node from $\mathcal{A}$; similarly consider a shortest path from γ to δ in $\mathcal{B} \cup \mathcal{S}$ that contains at least one node from $\mathcal{B}$. String these two shortest paths together to form a cycle of at least four nodes. Since we chose shortest paths, this cycle has no chord, contradicting our assumption; hence there must be an edge from γ to δ. Since $\gamma, \delta \in \mathcal{S}$ were arbitrary, we conclude that $\mathcal{S}$ is fully connected. □

Lemma 2. *If graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is chordal and has at least two nodes, then $\mathcal{G}$ has at least two nodes each of which has all its neighbors connected. Furthermore, if $\mathcal{G}$ is not a clique, then there are two such nodes that are nonadjacent.*

Proof. Note that the statement is trivial if $\mathcal{G}$ is a clique, since then any two nodes are connected to all other nodes, so we may assume that $\mathcal{G}$ is not a clique.

We use strong induction on the number of nodes in the graph. The base case of two nodes is trivial as each node has only one neighbor. Our inductive hypothesis is that any chordal graph with number of nodes between 2 and N, which is not a clique, has two nonadjacent nodes each with all its neighbors connected.

We want to show that the same holds for any non-clique chordal graph $\mathcal{G}'$ over $N+1$ nodes. We first apply Lemma 1 to decompose the graph into disjoint nonempty sets $\mathcal{A}, \mathcal{B}, \mathcal{S}$, where $\mathcal{S}$ is a clique separating $\mathcal{A}$ and $\mathcal{B}$. Note that $\mathcal{A} \cup \mathcal{S}$ and $\mathcal{B} \cup \mathcal{S}$ are each chordal with at least two nodes and at most N nodes. We will show that we can pick one node from each of $\mathcal{A}$ and $\mathcal{B}$ that has all its neighbors connected, which will complete the proof. By symmetry of the arguments for $\mathcal{A}$ and $\mathcal{B}$, we just need to show how to find one node in $\mathcal{A}$ that has all its neighbors connected.

Now, by the inductive hypothesis, $\mathcal{A} \cup \mathcal{S}$ has at least two nodes each with all its neighbors connected. In particular, if $\mathcal{A} \cup \mathcal{S}$ is fully connected, then we can just choose any point in $\mathcal{A}$ and its neighbors will all be connected and it certainly won't have any neighbors in $\mathcal{B}$ since $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$. On the other hand, if $\mathcal{A} \cup \mathcal{S}$ is not fully connected, by the inductive hypothesis there exist two nonadjacent nodes in $\mathcal{A} \cup \mathcal{S}$ each with all its neighbors connected, for which certainly one will be $\mathcal{A}$ because if both are in $\mathcal{S}$, they would actually be adjacent (since $\mathcal{S}$ is fully connected). □

We now prove our main result, that chordal graphs have junction trees.

Proof of Theorem 1. We use induction on the number of nodes in the graph. For the base case, we note that if a chordal graph has only one or two nodes, then trivially the graph has a junction tree. For the inductive hypothesis, suppose that any graph of up to N nodes that is chordal has a junction tree.

We want to show that a chordal graph $\mathcal{G}$ with $N+1$ nodes also has a junction tree. By Lemma 2, $\mathcal{G}$ has a node α with all its neighbors connected. Removing node α from $\mathcal{G}$ will result in a graph $\mathcal{G}'$ which is also chordal (check this!). By the inductive hypothesis, $\mathcal{G}'$, which has N nodes, has a junction tree $\mathcal{T}'$. We next show that we can modify $\mathcal{T}'$ to obtain a new junction tree $\mathcal{T}$ for our $(N+1)$-node chordal graph $\mathcal{G}$.

Let C be the maximal clique that node α participates in, which consists of α and its neighbors (which are all connected). If $C \backslash \alpha$ is a maximal-clique node in $\mathcal{T}'$, then we can just add α to this clique node to obtain a junction tree $\mathcal{T}$ for $\mathcal{G}$. (Check that the junction tree property holds for $\mathcal{T}$.)

Otherwise, if $C \backslash \alpha$ is not a maximal-clique node in $\mathcal{T}'$, then $C \backslash \alpha$, which we know to be a clique, must be a subset of a a maximal-clique node D in $\mathcal{T}'$. Then we add C as a new maximal-clique node in $\mathcal{T}'$ which we connect to D to obtain a junction tree $\mathcal{T}$ for $\mathcal{G}$; justifying why $\mathcal{T}$ is in fact a junction tree just involves checking the junction tree property and noting that α participates only in maximal clique C. □

14.3 Construction of Junction Trees

We now know that chordal graphs have junction trees. However, the proof given for Theorem 1 is existential and fails to deliver an efficient algorithm. Turning to the second question raised above, we now show how to construct a junction tree given a chordal graph.

Surprisingly, finding a junction tree for a chordal graph just boils down to finding a maximum-weight spanning tree, which can be efficiently solved using Kruskal's algorithm or Prim's algorithm. More precisely:

Theorem 2. *Consider the weighted graph $\mathcal{H}$, which is a clique graph for some underlying chordal graph $\mathcal{G}$, where we have an edge between two maximal-clique nodes V, W with weight $|V \cap W|$ whenever this weight is positive.*

Then a clique tree for $\mathcal{G}$ is a junction tree if and only if it is a maximum-weight spanning tree of $\mathcal{H}$.

Before we prove this, we state a key inequality: Suppose $\mathcal{T} = (\mathcal{C}, \mathcal{F})$ is a clique tree for underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with maximal cliques $\mathcal{C}$. Denote S_e as the separator set for edge $e \in \mathcal{F}$, i.e., S_e is the nodes in common between the two maximal cliques connected by e. Then for any $v \in \mathcal{V}$ we have the inequality

$$\sum_{e \in \mathcal{F}} \mathbb{1}_{v \in S_e} \leq \sum_{C \in \mathcal{C}} \mathbb{1}_{v \in C} - 1. \tag{5}$$

What this is saying is that the number of separator sets v participates in is bounded above by the number of maximal cliques v participates in minus 1. The minus 1 appears because the left-hand side is counting edges in $\mathcal{T}$ and the right-hand side is counting nodes in $\mathcal{T}$ (remember that a tree with N nodes has $N - 1$ edges). As a simple example, if v participates in two maximal cliques, then v will participate in at most one separator set, which is precisely the intersection of the two maximal cliques.

Lemma 3. *Eq.* (5) *is an equality for all $v \in \mathcal{V}$ if and only if the clique tree $\mathcal{T}$ is a junction tree.*

Proof. To see the $\Leftarrow$ direction, we have that if $\mathcal{T}$ is a junction tree then the set of nodes $\mathcal{C}_v$ is connected once we remove all cliques not containing v. Note that the RHS of (5) is equal to $|\mathcal{C}_v| - 1$. Now, connectivity implies that we're left with a tree on $\mathcal{C}_v$, hence there are $|\mathcal{C}_v| - 1$ edges, and each of these edges e has v in both endpoints hence also in S_e.

For the $\Rightarrow$ direction, note that $\mathcal{T}$ is a clique tree by assumption, so we only need to check the junction tree property. Equality in (5) says that the set of maximal clique nodes associated with v have number of edges one less than the number of such nodes, and this implies connectedness of $\mathcal{C}_v$ (there are no cycles because $\mathcal{T}$ is a tree). $\square$

Proof of Theorem 2. Let $\mathcal{T} = (\mathcal{C}, \mathcal{F})$ be a clique tree for underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Note that $\mathcal{T}$ is a subgraph of the weighted graph $\mathcal{H}$. Let $w(\mathcal{T})$ be the sum of all the weights of edges of $\mathcal{T}$ using the edge weights from $\mathcal{H}$. We obtain an upper-bound for $w(\mathcal{T})$ via inequality (5):

$$\begin{aligned} w(\mathcal{T}) = \sum_{e \in \mathcal{F}} |S_e| = \sum_{e \in \mathcal{F}} \sum_{v \in \mathcal{V}} \mathbb{1}_{v \in S_e} = \sum_{v \in \mathcal{V}} \sum_{e \in \mathcal{F}} \mathbb{1}_{v \in S_e} &\leq \sum_{v \in \mathcal{V}} \left(\sum_{C \in \mathcal{C}} \mathbb{1}_{v \in C} - 1 \right) \\ &= \sum_{v \in \mathcal{V}} \sum_{C \in \mathcal{C}} \mathbb{1}_{v \in C} - |V| \\ &= \sum_{C \in \mathcal{C}} \sum_{v \in \mathcal{V}} \mathbb{1}_{v \in C} - |V| \\ &= \sum_{C \in \mathcal{C}} |C| - |V|. \end{aligned}$$

This means that any maximum-weight spanning (clique) tree $\mathcal{T}$ for $\mathcal{H}$ has weight bounded above by $\sum_{C \in \mathcal{C}} |C| - |V|$, where equality is attained if and only if inequality (5) is an equality for all $v \in \mathcal{V}$, i.e., when $\mathcal{T}$ is a junction tree (which we know exists since G is chordal). □

14.4 Inference via Junction Trees

The last question asked was: if a graph doesn't have a junction tree, how can we modify it so that it does have a junction tree? Theorem 1 instructs us to modify the graph to make it chordal. However, we learned how to make a graph chordal when we discussed the elimination algorithm. In particular, when running the Elimination Algorithm, the reconstituted graph is always chordal. So it suffices to run the elimination algorithm using any elimination ordering to obtain the (chordal) reconstituted graph. As we saw there, the elimination ordering can have a big effect on the size of the resulting cliques, but any ordering will yield a chordal graph.

With this last question answered, we outline the key steps for the junction tree algorithm. As input, we have an undirected graph and an elimination ordering, and the output is comprised of all the node marginals.

1. Chordalize the input graph using the elimination algorithm.
2. Find all the maximal cliques in the chordal graph.
3. Determine all the separator set sizes for the maximal cliques to build a (possibly loopy) weighted clique graph.
4. Find a maximum-weight spanning tree in the weighted clique graph from the previous step to obtain a junction tree.
5. Assign singleton and edge potentials to the junction tree.

6. Run sum-product on the junction tree.
7. Do additional marginalization to get node marginals for each node in the original input graph.

As with the elimination algorithm, we incorporate observations by fixing observed variables to take on specific values as a preprocessing step. Also, we note that we've intentionally brushed aside implementation details, which give rise to different variants of the junction tree algorithm that have been named after different people. For example, by accounting for the structure of the edge potentials, we can simplify the sum-product message passing equation, resulting in the Shafer-Shenoy algorithm. Additional clever bookkeeping to avoid having to multiply many messages repeatedly yields the Hugin algorithm.

We end by discussing the computational complexity of the junction tree algorithm. The key quantity of interest is the treewidth of the original graph, defined in a previous lecture as the largest maximal clique size for an optimal elimination ordering. Note that using an optimal elimination ordering will result in the largest maximal-clique node in the junction tree having alphabet size exponential in the treewidth, so the junction tree algorithm will have running time exponential in the treewidth.

Yet, the fact that the junction tree algorithm may take exponential time should not be surprising: we can encode NP-complete problems such as 3-coloring as an inference problem over an undirected graph, so if the junction tree algorithm could efficiently solve all inference problems over undirected graphs, then the P vs. NP question would have long ago been settled.