Consider computing $P(X_i|y_1:N)$ for each $i$ using variable elimination. This would take $O(N^2)$ time.

However, there is a lot of repeated computation.

$P(X_1|e_{1:3}) \propto P(X_1)p(e_1|X_1) \sum_{X_2} P(X_2|X_1)p(e_2|X_2) \sum_{X_3} P(X_3|X_2)p(e_3)$

$P(X_2|e_{1:3}) \propto \sum_{X_1} P(X_1)p(e_1|X_1)P(X_2|X_1)p(e_2|X_2) \sum_{X_3} P(X_3|X_2)p(e_3)$

$P(X_3|e_{1:3}) \propto \sum_{X_1} P(X_1)p(e_1|X_1) \sum_{X_2} P(X_2|X_1)p(e_2|X_2)P(X_3|X_2)p(e_3)$

We will show how to use caching to compute all $N$ marginals in $O(N)$ time.
A cluster graph is called a junction tree if it is a tree and if for every \( X \in C_i \cap C_j \), then \( X \) occurs in every cluster in the (unique) path between \( C_i \) and \( C_j \). (The book incorrectly calls this the running intersection property.)

- Thm 8.1.5: Variable elimination produces a junction tree.
- Pf: once a variable is encountered in the ordering, it occurs in all factors that mention it until it is summed out. Once it has been removed, it cannot be used again.

**Constructing an elimination tree**

- The clusters (nodes) produced by variable elimination using order \( \prec \) applied to \( G \) are (non-maximal) cliques in the induced graph \( I_G, \prec \).
- These clusters \( C_i \) are called elimination sets.
- We can connect the esets into a tree that satisfies the jtree property in 2 steps:
  1. Run the variable elimination algorithm. Let \( v_i \) be the variable eliminated at the \( i \)th step, and \( C_i \) be the set of variables in \( v_i \)'s bucket at that time (so \( \tau_i = \sum \psi_i(C_i) \)).
  2. Connect \( C_i - C_j \) if \( \tau_i \) goes into \( j \)'s bucket, i.e., \( j \) is the largest index of a vertex in \( C_i \setminus \{v_i\} \).
- The etree has the property that residuals \( R_i = C_i \setminus S_{ij} \) are singleton sets, where \( S_{ij} = C_i \cap C_j \) is the separator between \( S_i \) and \( S_j \).

**Example of etree construction**

\[
P(i) = \sum_u \sum_j \sum_l p(u, l, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i)
\]

\[
= \sum_u \sum_j \sum_l p(u, l, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i)
\]

\[
= \sum_u \sum_j \sum_l p(u, l, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i)
\]

\[
= \sum_u \sum_j \sum_l p(u, l, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i)
\]

\[
= \sum_u \sum_j \sum_l p(u, l, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i) \sum_u p(u, G, i)
\]
From etree to jtree of maximal cliques

- Thm 8.4.1: We can remove non-maximal cliques and preserve the jtree property as follows.
- Let $C_j, C_i$ be a pair of cliques s.t. $C_j \subset C_i$. By the jtree property, $C_j$ is a subset of all cliques on the path from $C_j$ to $C_i$.
- Let $C_l$ be a neighbor of $C_j$ s.t. $C_j \subseteq C_l$. We remove $C_j$ and connect all of its neighbors to $C_l$.

\[
\begin{align*}
\text{HG} & \quad \text{JLSLSGSIGCDDIGIGG} \\
\text{Cj} & \quad \text{HG JLSLSGSIGCDDIGIG} \\
\text{Cl} & \quad \text{HG JLSLSGSIGCDDIGIGG}
\end{align*}
\]

From chordal graph to jtree of maximal cliques

- Thm 8.4.1 shows that there is a jtree for $F$ whose cliques are the maximal cliques in $I_{F,\prec}$.
- Suppose we are given the chordal graph $I_{F,\prec}$; how can we find the jtree directly?
- Step 1: find the maximal cliques of the chordal graph.
  - Finding maximal cliques is in general NP-hard.
  - But for chordal graphs, we can just run max cardinality search (or some other elimination algorithm) and save the maximal cliques.
- Step 2: connect the cliques so as to satisfy the jtree property.

Junction tree property

- Not every clique tree derived from a triangulated graph has the junction tree property.
- Defn: the weight of a clique tree is
  \[
  W(T) = \sum_{j=1}^{M-1} |S_j|
  \]
  where $M$ is the number of cliques and $S_j$ are separators.
- So the left graph (that does not have the jtree property) has weight $|\{C, D\}| + |\{D\}| = 3$, whereas the right graph (that does have the jtree property) has weight $|\{C, D\}| + |\{B, D\}| = 4$.

Jtree iff MWST

- Thm: a clique tree is a junction tree iff it is a maximal weight spanning tree.
- Proof. For a tree, the number of times $X_k$ appears in all separators is one less than the number of times $X_k$ appears in all cliques:
  \[
  \sum_{j=1}^{M-1} 1(X_k \in S_j) \leq \sum_{i=1}^{M} 1(X_k \in C_i) - 1
  \]
  which becomes an inequality if the subgraph induced by $X_k$ is a tree (i.e., $T$ is a jtree).
**Jtree iff MWST**

\[ w(T) = \sum_{j=1}^{M-1} |S_j| \]

\[ = \sum_{j=1}^{M-1} \sum_{k=1}^{N} 1(X_k \in S_j) \]

\[ = \sum_{k=1}^{N} \sum_{j=1}^{M-1} 1(X_k \in S_j) \]

\[ \leq \sum_{k=1}^{N} \sum_{j=1}^{M-1} 1(X_k \in C_j) - 1 \]

\[ = \sum_{j=1}^{M-1} \sum_{k=1}^{N} 1(X_k \in C_j) - N \]

\[ = \sum_{j=1}^{M} |C_j| - N \]

- This is an equality iff \( T \) is a jtree.

- To make a jtree from a set of cliques of a chordal graph
  - Build a junction graph, where weight on edge \( C_i \rightarrow C_j \) is \(|S_{ij}|\).
  - Find MWST using Prim’s or Kruskal’s algorithm.

**Initializing clique trees**

- The potential for clique \( C \) is initialized to the product of all assigned factors from the model:

\[ \pi_j(C_j) = \prod_{\phi: \alpha(\phi) = j} \phi \]

**From Bayes net to jtree**

- To compute \( P(J) \), we find some clique that contains \( J \) (eg. \( C_5 \)) and call it the root.

- We then send messages from the leaves up to the root.

- A node \( C_i \) can send to \( C_j \) (closer to the root) once it has received messages from all its other neighbors \( C_k \).

- The order to send the messages is called a schedule.

**Message passing in clique trees**
**General procedure for upwards pass**

$$
\psi_r^1 \overset{\text{def}}{=} \text{function Ctree-VE-up} (\{ \phi \}, T, \alpha, r) \\

\text{for } \quad \quad j := \text{pa}(DT, i) \\
\quad \delta_i \rightarrow j := \text{VE-msg}(\{ \delta_k \rightarrow i : k \in \text{ch}(DT, i) \}, \psi_i^0) \\
\text{end} \\
\psi_r^1 := \psi_r^0 \prod_{k \in \text{ch}(DT, r)} \delta_k \rightarrow r
$$

**Sub-functions**

$$
\{ \psi_i^0 \} \overset{\text{def}}{=} \text{function } \text{initialize Cliques}(\phi, \alpha) \\
\text{for } \quad i := 1 : C \\
\quad \psi_i^0 (C_i) = \prod_{\phi: \alpha(\phi) = i} \phi \\

\delta_i \rightarrow j \overset{\text{def}}{=} \text{function } \text{VE-msg}(\{ \delta_k \rightarrow i \}, \psi_i^0) \\
\psi_i^1 (C_i) := \psi_i^0 (C_i) \prod_k \delta_k \rightarrow i \\
\delta_i \rightarrow (S_i, j) := \sum_{C_j \setminus S_i} \psi_i^1 (C_i)
$$
Tree traversal orders

preorder = \([n, \text{pre}(T1), \text{pre}(T2)]\) (parents then children)
inorder = \([\text{in}(T1), n, \text{in}(T2)]\)
postorder = \([\text{post}(T1), \text{post}(T2), n]\) (children then parents)

Depth first search of a graph

- See e.g., “Introduction to algorithms”, Cormen, Leiserson, Rivest
- Initialize all nodes white; when first discovered, paint gray; when finished (all neighbors explored), paint black.
- \(d(u) = \) discovery time, \(f(u) = \) finish time, \(\pi(u) = \) predecessor in the dfs ordering

\((d, f, \pi) = \text{function } \text{dfs}(G)\)

for each vertex \(u\)
  
  \(\text{color}(u) := \text{white}\)
  \(\pi(u) := []\)
  
  time := 0
  
  for each \(u\)
    
    if \(\text{color}(u) = \text{white}\)
      then \(\text{dfs-visit}(u)\)
    endif
  
  \(\text{color}(u) := \text{black}\)
  \(f(u) := (\text{time} := \text{time} + 1)\)

Function dfs-visit(u)

\(\text{color}(u) := \text{gray}\)
\(d(u) := (\text{time} := \text{time} + 1)\)

for each \(v\) in neighbors(u)
  
  if \(\text{color}(v) = \text{white}\)
    then \(\text{pi}(v) := u;\)
      \(\text{dfs-visit}(v)\)
  elseif \(\text{color}(v) = \text{gray}\)
    then cycle detected
  endif

\(\text{color}(u) := \text{black}\)
\(f(u) := (\text{time} := \text{time} + 1)\)

Depth first search of a graph

Nodes labeled as \(d/f\)
Uses of DFS

- For message passing on an undirected tree:
  - We can root a tree at $R$ and make all arcs point away from $R$ by starting the DFS at $R$ and connecting $\pi(i)\rightarrow i$.
  - preorder (parents then children) = nodes sorted by discovery time
  - postorder (children then parents) = nodes sorted by finish time
- For visiting nodes in a DAG in a topological order (parents before children)
  - Topological order = nodes sorted by reverse finish time
- For checking if a DAG has cycles
  - Run DFS, see if you ever encounter a back-edge to a gray node
- For finding strongly connected components

Correctness of upwards pass

- Consider edge $C_i \rightarrow C_j$ in the clique tree. Let $F_{< (i \rightarrow j)}$ be all factors on the $C_i$ side, and $V_{< (i \rightarrow j)}$ be all variables on the $C_i$ side that are not in $S_{ij}$.
- Thm 8.2.3: the message from $i$ to $j$ summarizes everything to the left of the edge (since $S_{ij}$ separates the left from the right):
  $$\delta_{i \rightarrow j}(S_{ij}) = \sum_{V_{< (i \rightarrow j)}} \prod_{\phi \in F_{< (i \rightarrow j)}} \phi$$
- Corollary 8.2.4: for the root clique,
  $$\pi_r(C_r) = \sum_{X \setminus C_r} P'(X)$$

Meaning of the messages

- For edge $C_3 \rightarrow C_5$,
  $$F_{< (3 \rightarrow 5)} = \{P(D|C), P(C), P(G|I,D), P(I), P(S|I)\}$$
  $$V_{< (3 \rightarrow 5)} = \{C, D, I\}$$
  $$\delta_{3 \rightarrow 5}(G, S) = \sum_{C,D,I} P(D|C)P(C)P(G|I,D)P(I)P(S|I)$$

- e.g., for edge $C_3 \rightarrow C_5$,
  $$F_{< (3 \rightarrow 5)} = \{P(D|C), P(C), P(G|I,D), P(I), P(S|I)\}$$
  $$V_{< (3 \rightarrow 5)} = \{C, D, I\}$$
  $$\delta_{3 \rightarrow 5}(G, S) = \sum_{C,D,I} P(D|C)P(C)P(G|I,D)P(I)P(S|I)$$

Partial messages may not be probability distributions unless the ordering is topologically consistent with a Bayes net.

- Causal order
  $$\delta_{1 \rightarrow 2}(X_2) = \sum_{X_1} P(X_1)p(y_1|X_1)P(X_2|X_1)p(y_2|X_2) \propto P(X_2|y_{1:2})$$
  $$\delta_{2 \rightarrow 3}(X_3) = \sum_{X_2} \delta_{1 \rightarrow 2}(X_2)P(X_3|X_2)p(y_3|X_3) \propto P(X_3|y_{1:3})$$

- Anti-causal order
  $$\delta_{3 \rightarrow 2}(X_3) = \sum_{X_4} P(X_4|X_3)p(y_4|X_4) = p(y_4|X_3)$$
  $$\delta_{2 \rightarrow 1}(X_2) = \sum_{X_3} \delta_{3 \rightarrow 2}(X_2)P(X_3|X_2)p(y_3|X_3) = p(y_{3:4}|X_3)$$
Computing messages for each edge

- If we collect to $C_5$ (to compute $P(J)$)
  \[ \text{Message for } C_5 \]

- If we collect to $C_3$ (to compute $P(G)$)
  \[ \text{Message for } C_3 \]

- The messages $\delta_{1 \rightarrow 2}$, $\delta_{2 \rightarrow 3}$, $\delta_{4 \rightarrow 5}$ are the same in both cases.
- In general, if the root $R$ is on the $C_j$ side, the message from $C_i \rightarrow C_j$ is independent of $R$. If the root is on the $C_i$ side, the message from $C_j \rightarrow C_i$ is independent of $R$.
- Hence we can send an edge along each edge in both directions and thereby compute all marginals in $O(C)$ time.

Shafer-Shenoy algorithm

\[
\{ \psi_i^1 \} \overset{\text{def}}{=} \text{function Ctree-VE-calibrate}(\{ \phi \}, T, \alpha)
\]

\[ R := \text{pickRoot}(T) \]
\[ DT := \text{mkRootedTree}(T, R) \]
\[ \{ \psi_i^0 \} := \text{initializeClique}(\phi, \alpha) \]

(* Upwards pass *)

for $i \in \text{postorder}(DT)$
  \[ j := \text{pa}(DT, i) \]
  \[ \delta_i \rightarrow j := \text{VE-msg}(\{ \delta_k \rightarrow i : k \in \text{ch}(DT, i) \}, \psi_i^0) \]

(* Downwards pass *)

for $i \in \text{preorder}(DT)$
  for $j \in \text{ch}(DT, i)$
    \[ \delta_i \rightarrow j := \text{VE-msg}(\{ \delta_k \rightarrow i : k \in N_i \setminus j \}, \psi_i^0) \]

(* Combine *)

for $i := 1 : C$
  \[ \psi_i^1 := \psi_i^0 \prod_{k \in N_i \setminus i} \delta_k \rightarrow i \]

Correctness of Shafer Shenoy

- Thm 8.2.7: After running the algorithm,
  \[ \psi_i^1(C_i) = \sum_{X \setminus C_i} P'(X, e) \]

- Pf: the incoming messages $\delta_k \rightarrow i$ are exactly the same as those computed by making $C_i$ be the root; so correctness follows from the correctness of collect-to-root (upwards pass).
- The posterior of any set of nodes contained in a clique can be computed using
  \[ P(C_i | e) = \psi_i^1(C_i) / p(e) \]
  where the likelihood of the evidence can be computed from any clique
  \[ p(e) = \sum_{c_i} \psi_i^1(c_i) \]
Shafer Shenoy for HMMs

\[ \psi_t^0(X_t, X_{t+1}) = P(X_{t+1} \mid X_t) p(y_{t+1} \mid X_{t+1}) \]
\[ \delta_{t \to t+1}(X_{t+1}) = \sum_{X_t} \delta_{t \to t}(X_t) \psi_t^0(X_t, X_{t+1}) \]
\[ \delta_{t \to t-1}(X_t) = \sum_{X_{t+1}} \delta_{t+1 \to t}(X_{t+1}) \psi_t^1(X_t, X_{t+1}) \]
\[ \psi_t^1(X_t, X_{t+1}) = \delta_{t \to t}(X_t) \delta_{t+1 \to t}(X_{t+1}) \psi_t^0(X_t, X_{t+1}) \]

Forward-backwards algorithm for HMMs

\[ \alpha_t(i) \overset{\text{def}}{=} \delta_{t-1 \to t}(i) = P(X_t = i, y_{1:t}) \]
\[ \beta_t(i) \overset{\text{def}}{=} \delta_{t \to t-1}(i) = p(y_{t+1:T} \mid X_t = i) \]
\[ \xi_t(i, j) \overset{\text{def}}{=} \psi_t^1(X_t = i, X_{t+1} = j) = P(X_t = i, X_{t+1} = j, y_{1:T}) \]
\[ P(X_{t+1} = j \mid X_t = i) \overset{\text{def}}{=} A(i, j) \]
\[ p(y_t \mid X_t = i) \overset{\text{def}}{=} B_t(i) \]
\[ \alpha_t(j) = \sum_i \alpha_{t-1}(i) A(i, j) B_t(j) \]
\[ \beta_t(i) = \sum_j \beta_{t+1}(j) A(i, j) B_{t+1}(j) \]
\[ \xi_t(i, j) = \alpha_t(i) \beta_{t+1}(j) A(i, j) B_{t+1}(j) \]
\[ \gamma_t(i) \overset{\text{def}}{=} P(X_t = i \mid y_{1:T}) \propto \alpha_t(i) \beta_t(j) \propto \sum_j \xi_t(i, j) \]

Forward-backwards algorithm, matrix-vector form

\[ \alpha_t(j) = \sum_i \alpha_{t-1}(i) A(i, j) B_t(j) \]
\[ \alpha_t = (A^T \alpha_{t-1}) \cdot B_t \]
\[ \beta_t = A(\beta_{t+1} \cdot B_{t+1}) \]
\[ \xi_t(i, j) = \alpha_t(i) \beta_{t+1}(j) A(i, j) B_{t+1}(j) \]
\[ \xi_t = \left( \alpha_t(\beta_{t+1} \cdot B_{t+1})^T \right) \cdot A \]
\[ \gamma_t(i) \propto \alpha_t(i) \beta_t(j) \]
\[ \gamma_t \propto \alpha_t \cdot \beta_t \]
Forwards algorithm uses dynamic programming to efficiently sum over all possible paths that state $i$ at time $t$.

\[
\alpha(i) \overset{\text{def}}{=} P(X_t = i, y_{1:t}) = \sum_{X_{i-1}} \sum_{X_{t-1}} P(X_{i-1}, \ldots, X_1, y_{1:t-1}) P(X_t | X_{t-1}) p(y_t | X_t)
\]

\[
= \sum_{X_{i-1}} P(X_t = 1, y_{1:t-1}) P(X_t | X_{t-1}) p(y_t | X_t)
\]

\[
= \sum_{X_{i-1}} \alpha_{i-1}(X_{i-1}) P(X_t | X_{t-1}) p(y_t | X_t)
\]