Probabilistic Graphical Models

Lecture 7: Exact Inference: Belief Propagation

Matthew Walter

TTI-Chicago

April 28, 2020

Office Hours

- Wednesdays, 9:00am–10:00am CT
- We will use the same Zoom meeting as we do for lectures
  - **Meeting ID**: 558-526-524
  - **Password**: 31180
  - `https://uchicago.zoom.us/j/558526524`

Probabilistic Inference (Revisited)

- Consider the following conditional probability query

$$P(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e}) = \frac{P(\boldsymbol{Y}, \boldsymbol{e})}{P(\boldsymbol{e})}$$

  (e.g., probability of having the flu given you have a stomach ache)

- We can compute these joint distributions by marginalizing over $\boldsymbol{W} = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$

$$P(\boldsymbol{Y}, \boldsymbol{e}) = \sum_{\boldsymbol{w}} P(\boldsymbol{Y}, \boldsymbol{e}, \boldsymbol{w}) \qquad P(\boldsymbol{e}) = \sum_{\boldsymbol{y}} P(\boldsymbol{y}, \boldsymbol{e})$$

- Naive marginalization over unobserved variables requires an exponential number of computations
- Are there techniques for doing inference more efficiently?

Probabilistic Inference (Revisited)

- Worst-case analysis shows that marginal inference is NP-hard!
- Even approximate inference is NP-hard!
- In practice, we can perform inference in a tractable manner:
  1. Exploit the structure of the graphical model to identify subexpressions that depend only on a small number of variables
  2. Cache computations that are otherwise computed exponentially many times
- In other words, perform local operations on factors rather than generate the joint distribution

Sum-Product Variable Elimination (Revisited)

- Order the variables $\boldsymbol{Z}$ (known as the **elimination ordering**)
- Iteratively marginalize out each variable $Z_i \in \boldsymbol{Z}$
- For each $Z_i \in \boldsymbol{Z}$ according to $\prec$:
  1. Multiply all factors with Z_i in their scope, creating a new product factor
  2. Marginalize this product factor over Z_i, generating a smaller factor
  3. Remove the old factors from the set of all factors, add the new one
- Basic operation involves a manipulation of factors
- Efficiency depends on having a good elimination ordering
- Unfortunately, solving for best ordering is also NP-hard, but...
- We can often efficiently identify a good ordering (e.g., via heuristics)

Cluster Graphs

Definition (Cluster Graph)

A **cluster graph** for a set of factors Φ over $\mathcal{X}$ is an undirected graph where

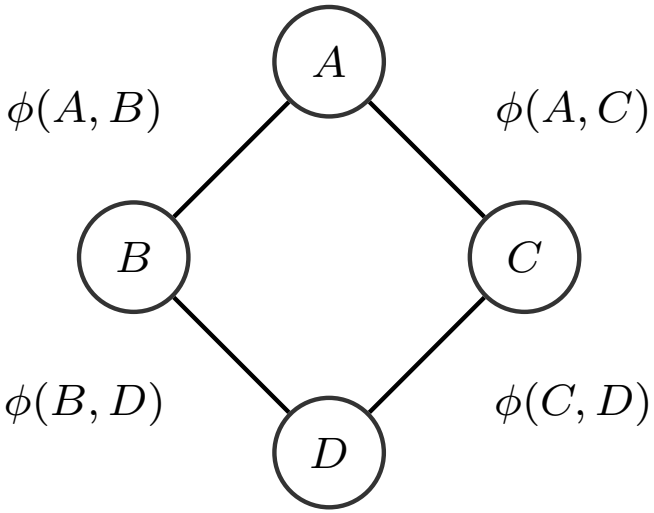
1. each node i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$
2. each factor $\phi \in \Phi$ is associated with a cluster $\boldsymbol{C}_i$ denoted as $\alpha(\phi)$ such that Scope$[\phi] \subseteq \boldsymbol{C}_i$ (the graph is **family preserving**)
3. an edge connects a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ if they share variables $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j \neq \emptyset$

Cluster Graphs

Definition (Cluster Graph)

A **cluster graph** for a set of factors Φ over $\mathcal{X}$ is an undirected graph where

1. each node i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$
2. each factor $\phi \in \Phi$ is associated with a cluster $\boldsymbol{C}_i$ denoted as $\alpha(\phi)$ such that Scope$[\phi] \subseteq \boldsymbol{C}_i$ (the graph is **family preserving**)
3. an edge connects a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ if they share variables $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j \neq \emptyset$

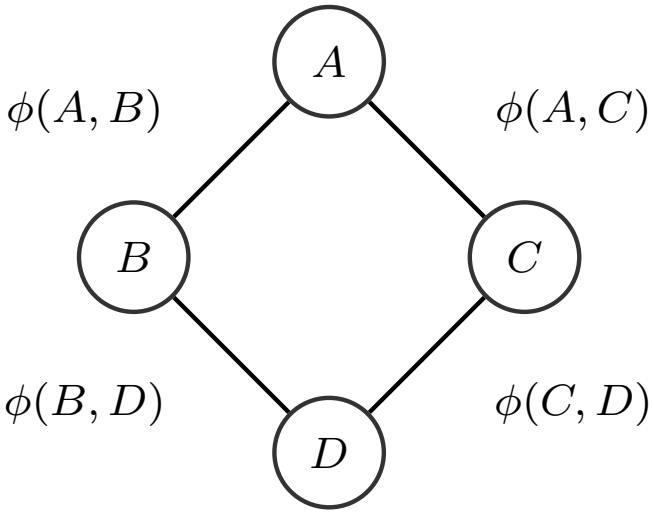


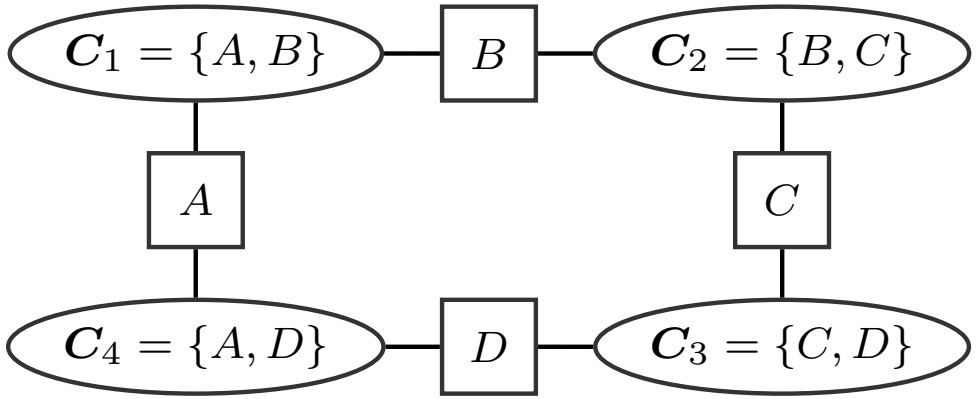
Cluster Graphs

Definition (Cluster Graph)

A **cluster graph** for a set of factors Φ over $\mathcal{X}$ is an undirected graph where

1. each node i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$
2. each factor $\phi \in \Phi$ is associated with a cluster $\boldsymbol{C}_i$ denoted as $\alpha(\phi)$ such that $\text{Scope}[\phi] \subseteq \boldsymbol{C}_i$ (the graph is **family preserving**)
3. an edge connects a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ if they share variables $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j \neq \emptyset$





Cluster Graphs

Definition (Cluster Graph)

A **cluster graph** for a set of factors Φ over $\mathcal{X}$ is an undirected graph where

1. each node i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$
2. each factor $\phi \in \Phi$ is associated with a cluster $\boldsymbol{C}_i$ denoted as $\alpha(\phi)$ such that $\text{Scope}[\phi] \subseteq \boldsymbol{C}_i$ (the graph is **family preserving**)
3. an edge connects a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ if they share variables $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j \neq \emptyset$

- Variable elimination induces a cluster graph, where $\boldsymbol{C}_i = \text{Scope}[\psi_i]$
- Variable elimination cluster graph is a tree ("cluster tree"). Why?

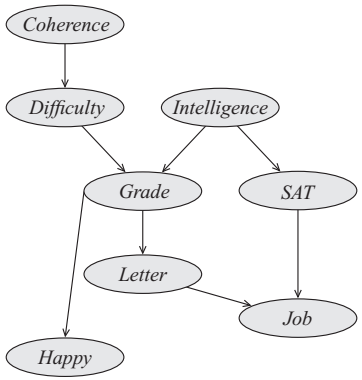
Cluster Graphs

Definition (Cluster Graph)

A **cluster graph** for a set of factors Φ over $\mathcal{X}$ is an undirected graph where

1. each node i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$
2. each factor $\phi \in \Phi$ is associated with a cluster $\boldsymbol{C}_i$ denoted as $\alpha(\phi)$ such that $\text{Scope}[\phi] \subseteq \boldsymbol{C}_i$ (the graph is **family preserving**)
3. an edge connects a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ if they share variables $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j \neq \emptyset$

- Variable elimination induces a cluster graph, where $\boldsymbol{C}_i = \text{Scope}[\psi_i]$
- Variable elimination cluster graph is a tree ("cluster tree"). Why?



Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G, I, D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G, S, I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H, G, J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G, J, L, S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J, L, S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J, L	$\tau_7(J)$

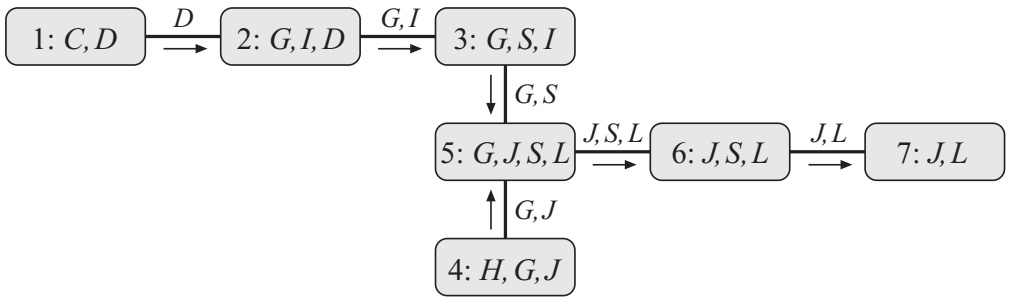
Cluster Graphs

Definition (Cluster Graph)

A **cluster graph** for a set of factors Φ over $\mathcal{X}$ is an undirected graph where

1. each node i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$
2. each factor $\phi \in \Phi$ is associated with a cluster $\boldsymbol{C}_i$ denoted as $\alpha(\phi)$ such that $\text{Scope}[\phi] \subseteq \boldsymbol{C}_i$ (the graph is **family preserving**)
3. an edge connects a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ if they share variables $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j \neq \emptyset$

- Variable elimination induces a cluster graph, where $\boldsymbol{C}_i = \text{Scope}[\psi_i]$
- Variable elimination cluster graph is a tree ("cluster tree"). Why?



Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G, I, D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G, S, I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H, G, J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G, J, L, S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J, L, S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J, L	$\tau_7(J)$

Running Intersection Property

Definition (Running Intersection Property)

A tree $\mathcal{T}$ satisfies the **running intersection property** if, whenever $X \in \boldsymbol{C}_i$ and $X \in \boldsymbol{C}_j$, then X is also in every cluster in the unique path between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$

Cluster Trees and Variable Elimination

Theorem

The cluster tree $\mathcal{T}$ induced by variable elimination satisfies the running intersection property

Proof.

1. Consider two clusters $\boldsymbol{C}$ and $\boldsymbol{C}'$ that both contain X, and suppose that X is eliminated at $\boldsymbol{C}'$ (i.e., last)
2. When we eliminate X from $\boldsymbol{C}'$, no other factors (or clusters) contain X in their domain
3. Summation on $\boldsymbol{C}$ does not eliminate X and the resulting message is multiplied into it's upstream neighbor, which then has X in its scope
4. This logic extends upstream till we reach $\boldsymbol{C}'$

Clique Trees

- Consider a tree-structured graph $\mathcal{T}$ where each node corresponds to a (maximal) clique $\boldsymbol{C}_i$ in an undirected graph H
- Let $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ be two cliques that are connected by an edge. We define $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$ to be the **sepset** between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$
- Let $\boldsymbol{W}_{<(i,j)}$ ($\boldsymbol{W}_{<(j,i)}$) be all of the variables that appear on the $\boldsymbol{C}_i$ ($\boldsymbol{C}_j$) side

Clique Trees

- Consider a tree-structured graph $\mathcal{T}$ where each node corresponds to a (maximal) clique $\boldsymbol{C}_i$ in an undirected graph H
- Let $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ be two cliques that are connected by an edge. We define $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$ to be the **sepset** between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$
- Let $\boldsymbol{W}_{<(i,j)}$ ($\boldsymbol{W}_{<(j,i)}$) be all of the variables that appear on the $\boldsymbol{C}_i$ ($\boldsymbol{C}_j$) side

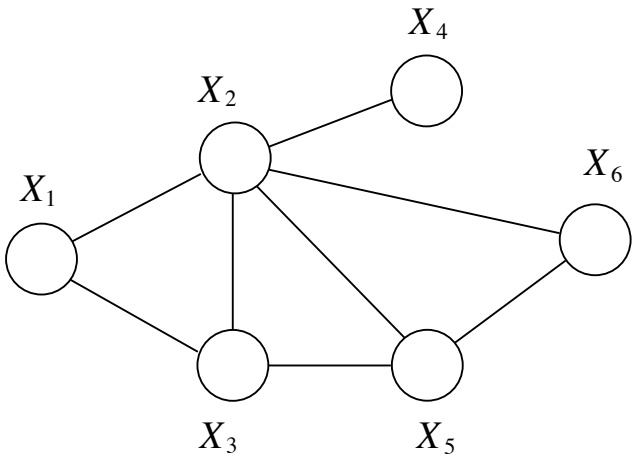
Definition (Clique Tree)

A tree $\mathcal{T}$ is a **clique tree** for an undirected graph H if:

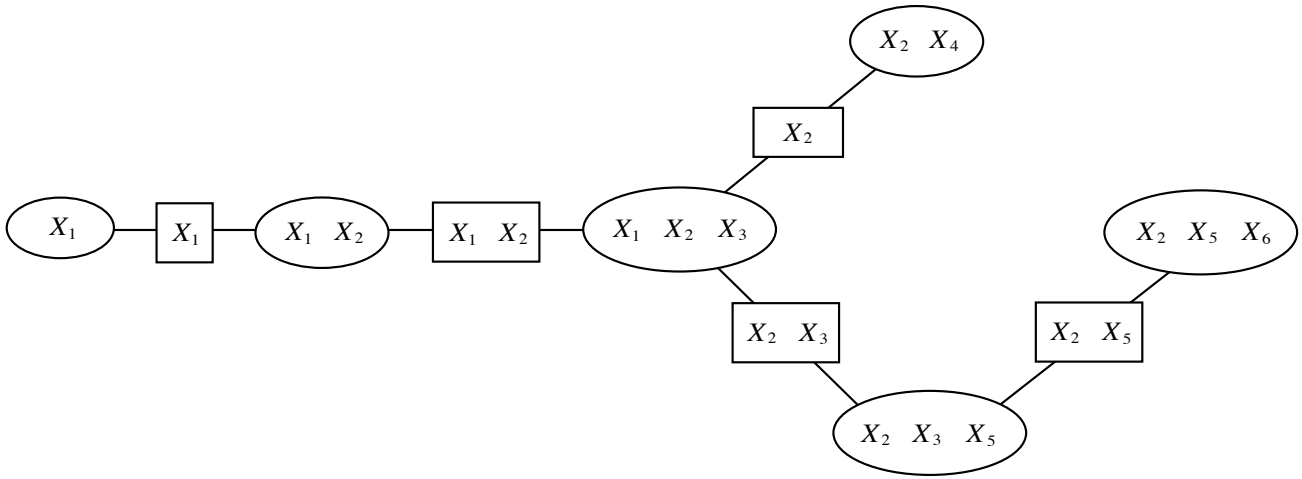
- Each node in $\mathcal{T}$ corresponds to a clique in H
- Each maximal clique in H is a node it $\mathcal{T}$
- Each sepset $S_{i,j}$ *separates* $\boldsymbol{W}_{<(i,j)}$ and $\boldsymbol{W}_{<(j,i)}$ in H

Also known as **junction trees** or **join trees**

Clique Trees: Example



Undirected graph H



Clique tree with ellipses for cliques and squares for sepsets

Clique Trees and Variable Elimination

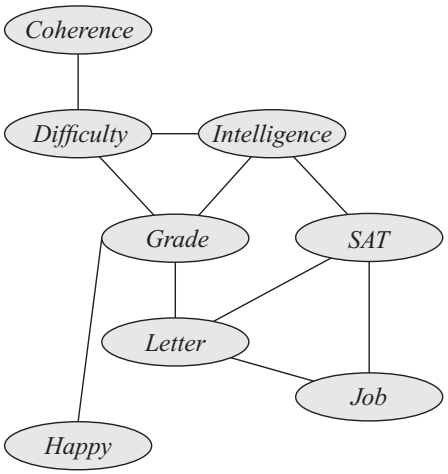
Theorem

A cluster tree $\mathcal{T}$ for a graph H satisfies the running intersection property iff for every sepset $\boldsymbol{S}_{i,j}$, $\boldsymbol{W}_{\prec(i,j)}$ and $\boldsymbol{W}_{\prec(j,i)}$ are separated in H given $\boldsymbol{S}_{i,j}$

- A cluster tree that satisfies the running intersection property is a clique tree
- This and the previous definition of a clique tree are equivalent
- Variable elimination induces a clique tree
- As we will see, there are other ways to produce a clique tree

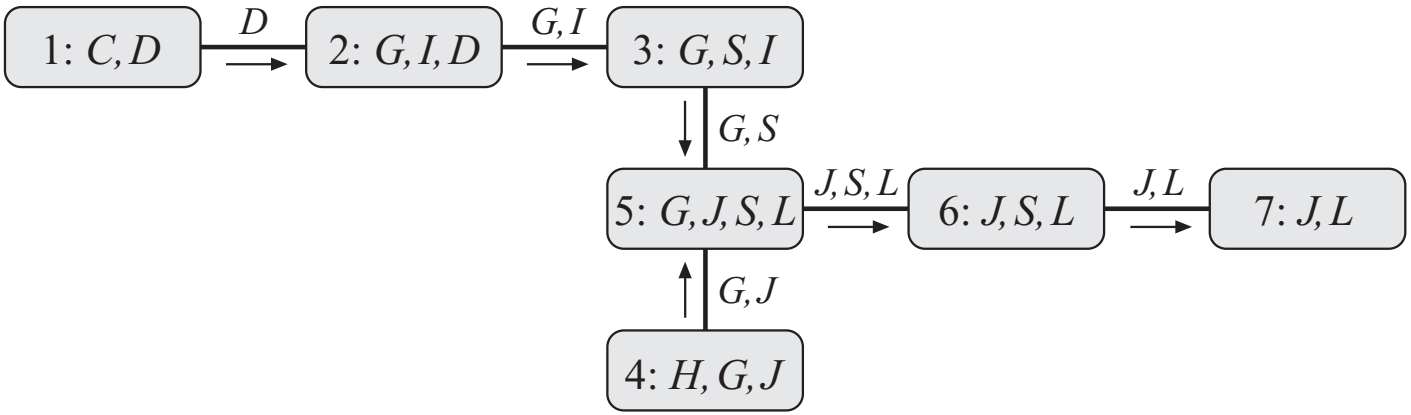
Clique Trees and Variable Elimination: Example

Recall the example from last lecture, where we are interested in $P(J)$ via the ordering C, D, I, H, G, S, L:



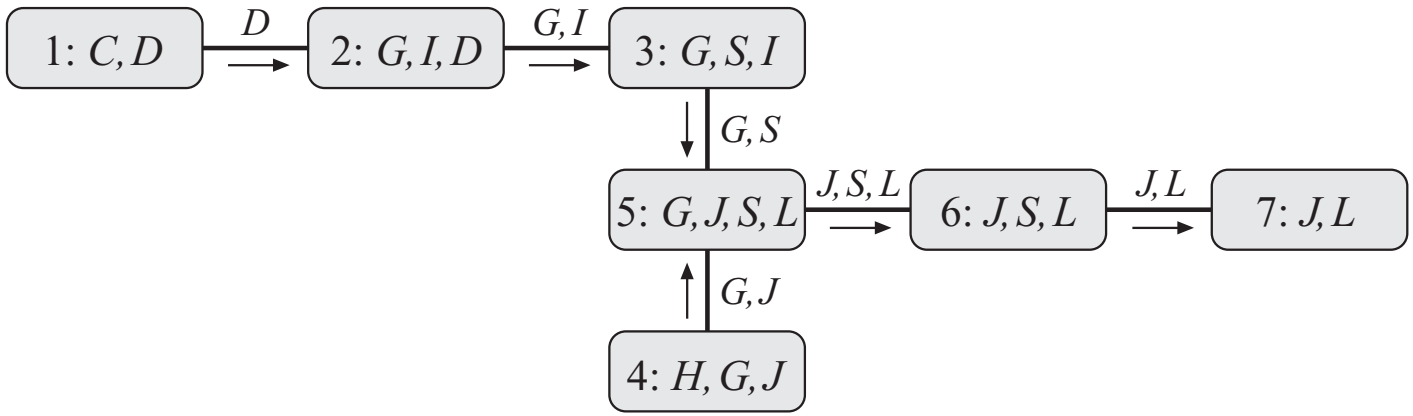
Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G, I, D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S(S,I)$, $\tau_2(G,I)$	G, S, I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H, G, J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$, $\tau_3(G,S)$, $\phi_L(L,G)$	G, J, L, S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$, $\phi_J(J,L,S)$	J, L, S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J, L	$\tau_7(J)$

This gives rise to the following clique tree:



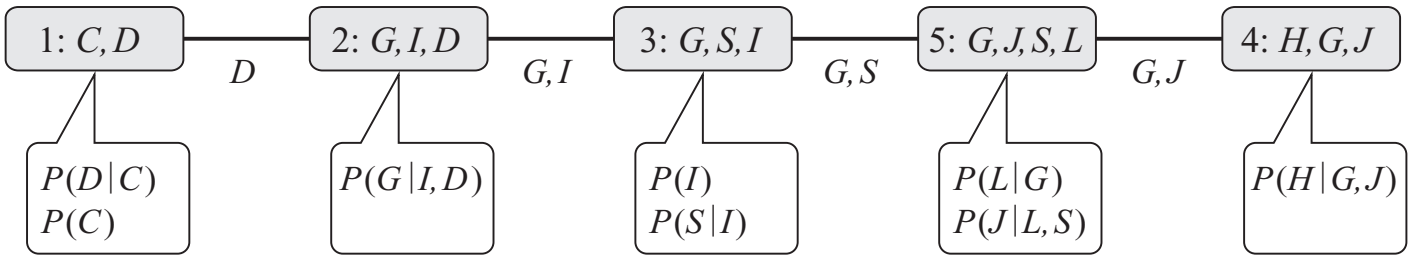
Clique Trees and Variable Elimination

- Clique trees are undirected, but variable elimination defines a direction for edges
- Results in a directed tree with all factors (messages) flowing towards a single cluster where the final computation is performed (the **root**)
- $\boldsymbol{C}_i$ is **upstream** from $\boldsymbol{C}_j$ if $\boldsymbol{C}_i$ is on the (unique) path to the root from $\boldsymbol{C}_j$
- Similarly, $\boldsymbol{C}_j$ is **downstream** from $\boldsymbol{C}_i$



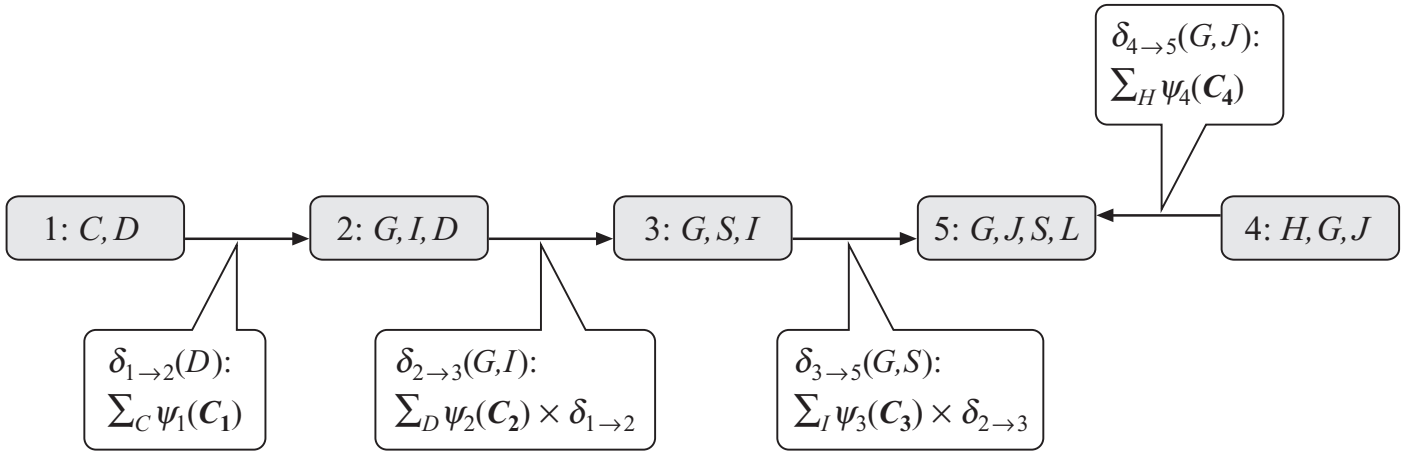
Clique Trees and Variable Elimination

- A clique tree is a useful computational data structure for performing variable elimination
- Each clique in the tree performs local computations:
  1. Takes incoming factors (messages) τ and multiplies them with its initial factor ψ
  2. Sums out (eliminates) one or more variables
  3. Sends outgoing messages to its neighbor cliques
- Structure dictates partial ordering over these operations: If $\boldsymbol{C}'$ is upstream of $\boldsymbol{C}$, then $\boldsymbol{C}'$ must wait for $\boldsymbol{C}$ to finish its computations



Simplified Clique Tree

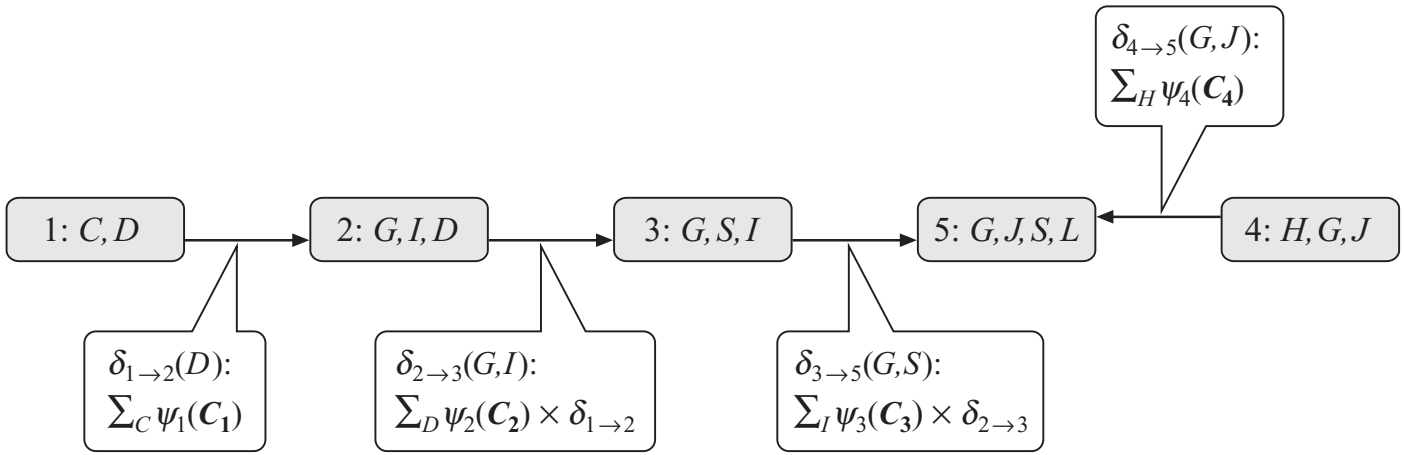
Clique Trees and Variable Elimination: Example



(Note, we use $m_{1\to2}(D)$ rather than $\delta_{1\to2}(D)$)

1. For each $\boldsymbol{C}_i$, set initial potential $\psi_i(\boldsymbol{C}_i)$ as the product of the clique's initial factors
2. Choose *any* clique with J to be the root (e.g., $\boldsymbol{C}_5$)

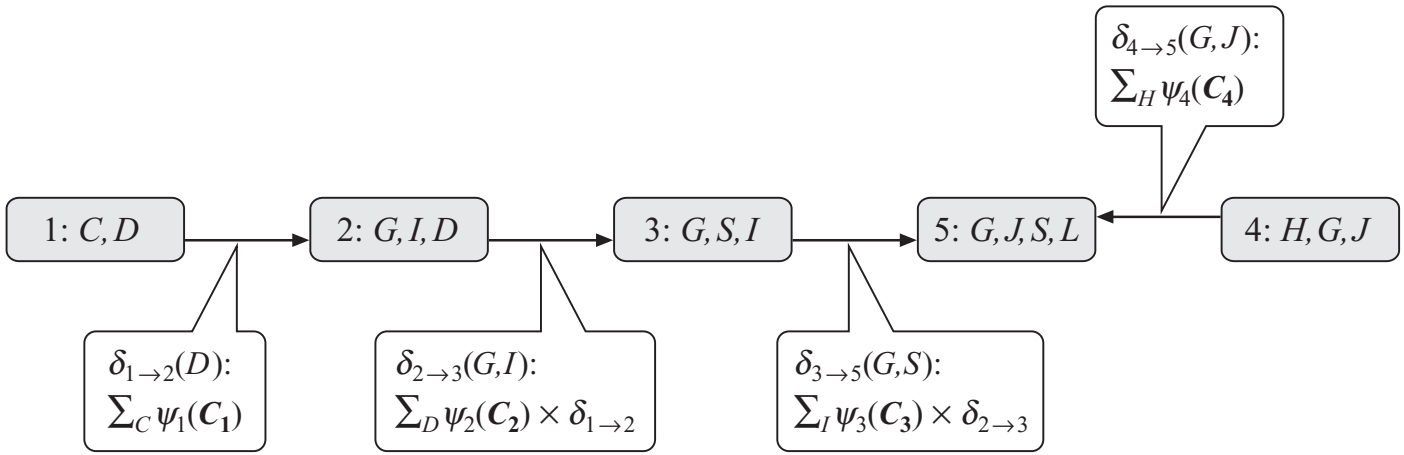
Clique Trees and Variable Elimination: Example



1. $\boldsymbol{C}_1$: Eliminate C as $m_{1\to2}(D) = \sum_C \psi_1(C, D)$[1] and send to $\boldsymbol{C}_2$

[1] We use $m_{1\to2}(D)$ to denote "messages" rather than $\delta_{1\to2}(D)$

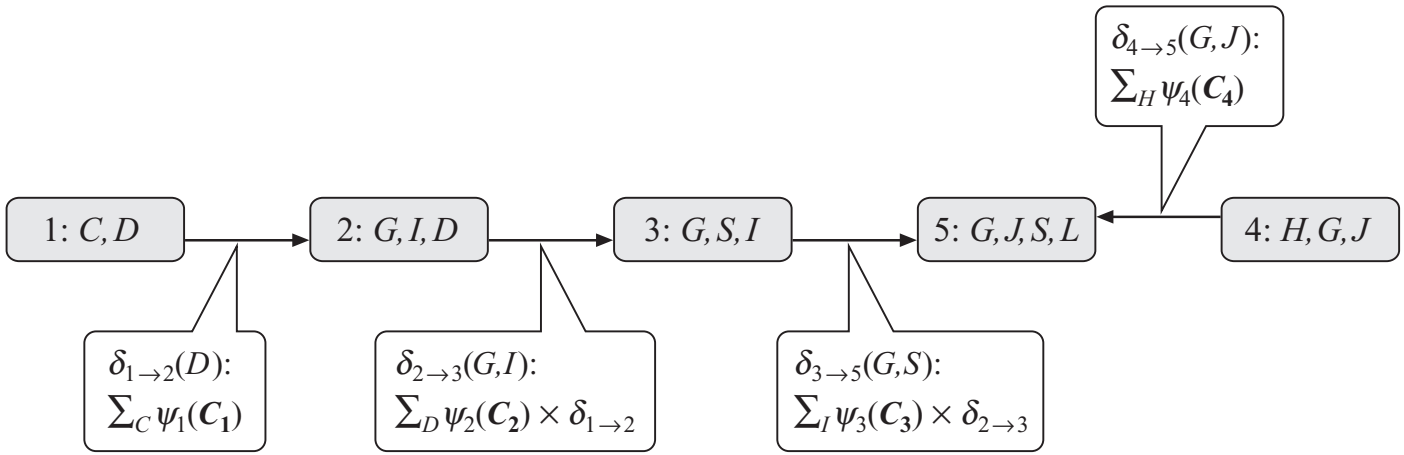
Clique Trees and Variable Elimination: Example



1. $\boldsymbol{C}_1$: Eliminate C as $m_{1\to2}(D) = \sum_C \psi_1(C, D)$[1] and send to $\boldsymbol{C}_2$
2. $\boldsymbol{C}_2$: Define new $\beta_2(G, I, D) = m_{1\to2}(D) \cdot \psi_2(G, I, D)$, eliminate D as $m_{2\to3}(G, I) = \sum_D \beta_2(G, I, D)$, and send to $\boldsymbol{C}_3$

[1] We use $m_{1\to2}(D)$ to denote "messages" rather than $\delta_{1\to2}(D)$

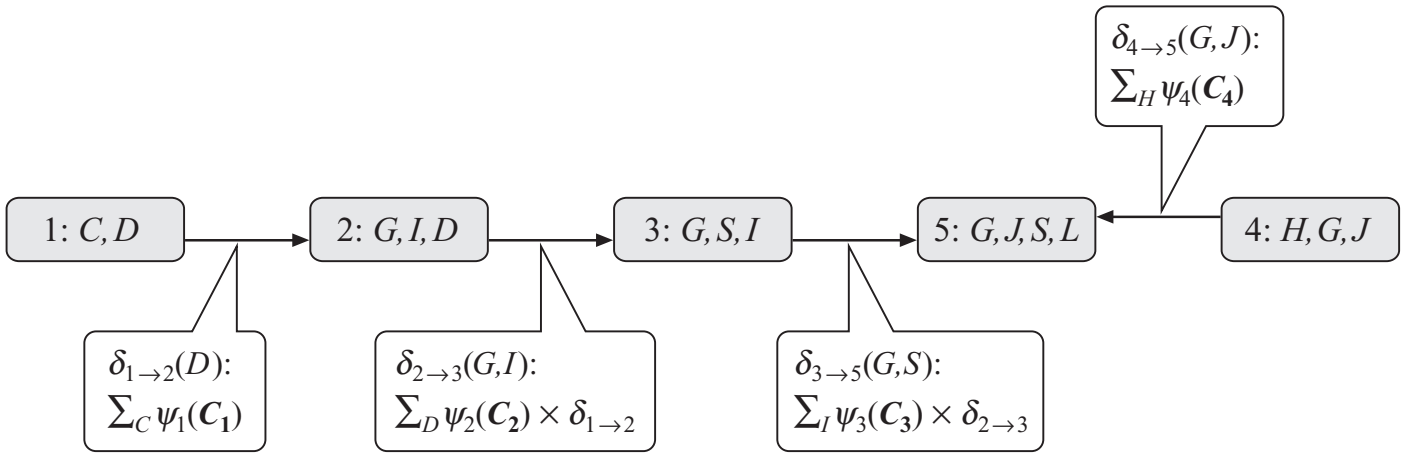
Clique Trees and Variable Elimination: Example



1. $\boldsymbol{C}_1$: Eliminate C as $m_{1\to2}(D) = \sum_C \psi_1(C, D)$[1] and send to $\boldsymbol{C}_2$
2. $\boldsymbol{C}_2$: Define new $\beta_2(G, I, D) = m_{1\to2}(D) \cdot \psi_2(G, I, D)$, eliminate D as $m_{2\to3}(G, I) = \sum_D \beta_2(G, I, D)$, and send to $\boldsymbol{C}_3$
3. $\boldsymbol{C}_3$: Define new $\beta_3(G, I, S) = m_{2\to3}(G, I) \cdot \psi_3(G, I, S)$, eliminate I as $m_{3\to5}(G, S) = \sum_I \beta_3(G, I, S)$, and send to $\boldsymbol{C}_5$

[1] We use $m_{1\to2}(D)$ to denote "messages" rather than $\delta_{1\to2}(D)$

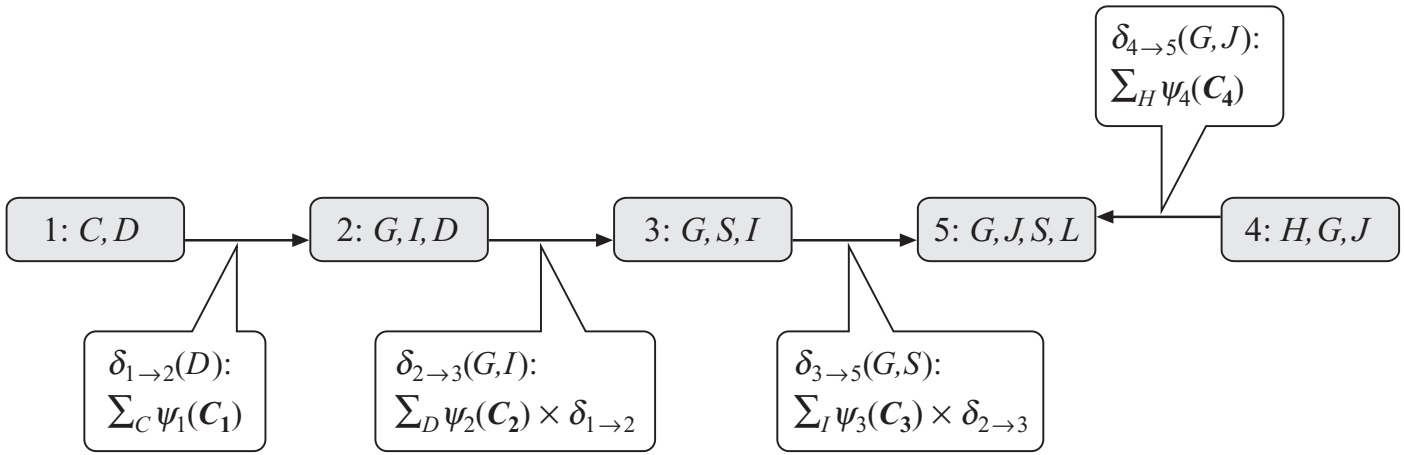
Clique Trees and Variable Elimination: Example



1. $\boldsymbol{C}_1$: Eliminate C as $m_{1\to 2}(D) = \sum_C \psi_1(C, D)$[1] and send to $\boldsymbol{C}_2$
2. $\boldsymbol{C}_2$: Define new $\beta_2(G, I, D) = m_{1\to 2}(D) \cdot \psi_2(G, I, D)$, eliminate D as $m_{2\to 3}(G, I) = \sum_D \beta_2(G, I, D)$, and send to $\boldsymbol{C}_3$
3. $\boldsymbol{C}_3$: Define new $\beta_3(G, I, S) = m_{2\to 3}(G, I) \cdot \psi_3(G, I, S)$, eliminate I as $m_{3\to 5}(G, S) = \sum_I \beta_3(G, I, S)$, and send to $\boldsymbol{C}_5$
4. $\boldsymbol{C}_4$: Eliminate H as $m_{4\to 5}(G, J) = \sum_H \psi(G, H, J)$, and send to $\boldsymbol{C}_5$

[1] We use $m_{1\to 2}(D)$ to denote "messages" rather than $\delta_{1\to 2}(D)$

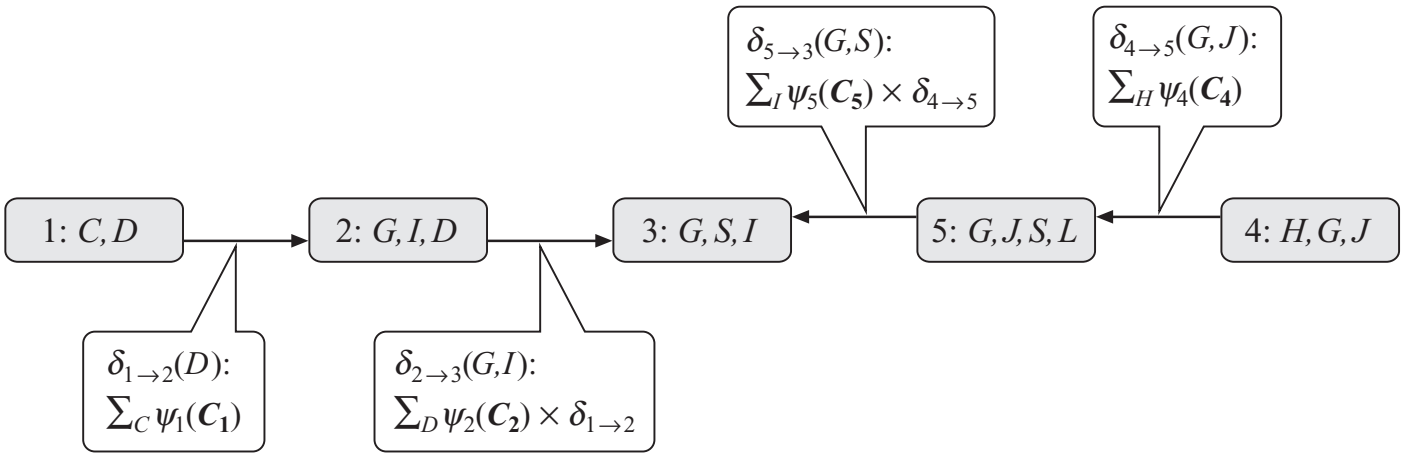
Clique Trees and Variable Elimination: Example



1. $\boldsymbol{C}_1$: Eliminate C as $m_{1\to 2}(D) = \sum_C \psi_1(C, D)$[1] and send to $\boldsymbol{C}_2$
2. $\boldsymbol{C}_2$: Define new $\beta_2(G, I, D) = m_{1\to 2}(D) \cdot \psi_2(G, I, D)$, eliminate D as $m_{2\to 3}(G, I) = \sum_D \beta_2(G, I, D)$, and send to $\boldsymbol{C}_3$
3. $\boldsymbol{C}_3$: Define new $\beta_3(G, I, S) = m_{2\to 3}(G, I) \cdot \psi_3(G, I, S)$, eliminate I as $m_{3\to 5}(G, S) = \sum_I \beta_3(G, I, S)$, and send to $\boldsymbol{C}_5$
4. $\boldsymbol{C}_4$: Eliminate H as $m_{4\to 5}(G, J) = \sum_H \psi(G, H, J)$, and send to $\boldsymbol{C}_5$
5. $\boldsymbol{C}_5$: Define $\beta_5(G, J, S, L) = m_{3\to 5}(G, S) \cdot m_{4\to 5}(G, J) \cdot \psi_5(G, J, S, L)$

[1] We use $m_{1\to 2}(D)$ to denote "messages" rather than $\delta_{1\to 2}(D)$

Clique Trees and Variable Elimination: Example



$\boldsymbol{C_3}$ as the root

- The choice of the root is flexible (e.g., we could have chosen $\boldsymbol{C_4}$)
- Computations are performed when the clique is *ready*
- We can employ the same process to compute other marginals
  Importantly, many of the computations would be repeated

Message Passing in Clique Trees

Now, let's consider the general form for variable elimination as **message passing** in a clique tree $\mathcal{T}$ with cliques $\boldsymbol{C}_1, \ldots \boldsymbol{C}_k$

- Each factor $\phi \in \Phi$ is assigned to some clique $\alpha(\phi)$
- Define the *initial potential* of each $\boldsymbol{C}_j$ to be

$$\psi_j(\boldsymbol{C}_j) = \prod_{\phi \,:\, \alpha(\phi) = \boldsymbol{C}_j} \phi$$

- Define Nb_i to be the cliques connected to $\boldsymbol{C}_i$
- The message from $\boldsymbol{C}_i$ to upstream $\boldsymbol{C}_j$ is computed via the **sum-product message passing** computation

$$m_{i \to j}(\boldsymbol{S}_{i,j}) = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \cdot \prod_{k \in (\text{Nb}_i - \{j\})} m_{k \to i}$$

- The resulting multiplied factor at the root is the **belief** $\beta_r(\boldsymbol{C}_r) = \tilde{P}(\boldsymbol{C}_r) = \sum_{\mathcal{X} - \boldsymbol{C}_r} \prod_\phi \phi$

Clique Tree Message Passing

- Often, we may want to calculate the marginals for several variables, e.g., $P(X_i)\ \forall i \in \{1, \ldots, n\}$

Clique Tree Message Passing

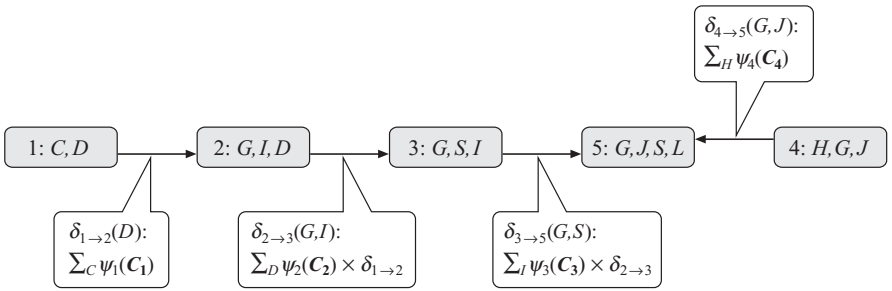
- Often, we may want to calculate the marginals for several variables, e.g., $P(X_i)\ \forall i \in \{1, \ldots, n\}$
- One approach would be to re-run variable elimination for each X_i
  $\Rightarrow$ Cost is $\mathcal{O}(nc)$, where c is the cost of each VE

Clique Tree Message Passing

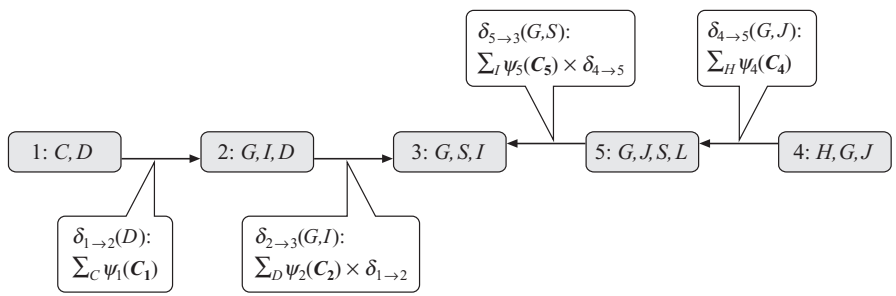
- Often, we may want to calculate the marginals for several variables, e.g., $P(X_i)\ \forall i \in \{1, \ldots, n\}$
- One approach would be to re-run variable elimination for each X_i
  $\Rightarrow$ Cost is $\mathcal{O}(nc)$, where c is the cost of each VE
- Alternatively, we can run clique tree message passing separately with each clique as the root
  $\Rightarrow$ Cost is $\mathcal{O}(Kc) < \mathcal{O}(nc)$, where K is the number of cliques

Clique Tree Message Passing

- Often, we may want to calculate the marginals for several variables, e.g., $P(X_i)\ \forall i \in \{1, \ldots, n\}$
- One approach would be to re-run variable elimination for each X_i
  $\Rightarrow$ Cost is $\mathcal{O}(nc)$, where c is the cost of each VE
- Alternatively, we can run clique tree message passing separately with each clique as the root
  $\Rightarrow$ Cost is $\mathcal{O}(Kc) < \mathcal{O}(nc)$, where K is the number of cliques
- However, there are repeated computations at cliques (e.g., computations at cliques $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are the same when $\boldsymbol{C}_3$, $\boldsymbol{C}_4$, and $\boldsymbol{C}_5$ are the root)



$P(J)$



$P(I)$

Clique Tree Message Passing

- Formally, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ takes the form

$$m_{i \to j}(\boldsymbol{S}_{i,j}) = \sum_{\mathcal{V}_{\prec(i \to j)}} \prod_{\phi \in \mathcal{F}_{\prec(i \to j)}} \phi$$

where $\mathcal{F}_{\prec(i \to j)}$ denotes the factors in the cliques on the $\boldsymbol{C}_i$-side and $\mathcal{V}_{\prec(i \to j)}$ denotes the variables on the $\boldsymbol{C}_i$-side (excluding sepset)

Clique Tree Message Passing

- Formally, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ takes the form

$$m_{i \to j}(\boldsymbol{S}_{i,j}) = \sum_{\mathcal{V}_{\prec(i \to j)}} \prod_{\phi \in \mathcal{F}_{\prec(i \to j)}} \phi$$

  where $\mathcal{F}_{\prec(i \to j)}$ denotes the factors in the cliques on the $\boldsymbol{C}_i$-side and $\mathcal{V}_{\prec(i \to j)}$ denotes the variables on the $\boldsymbol{C}_i$-side (excluding sepset)
- Thus, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ doesn't depend on root clique as long as the root is on the $\boldsymbol{C}_j$-side

Clique Tree Message Passing

- Formally, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ takes the form

$$m_{i \to j}(\boldsymbol{S}_{i,j}) = \sum_{\mathcal{V}_{\prec(i \to j)}} \prod_{\phi \in \mathcal{F}_{\prec(i \to j)}} \phi$$

  where $\mathcal{F}_{\prec(i \to j)}$ denotes the factors in the cliques on the $\boldsymbol{C}_i$-side and $\mathcal{V}_{\prec(i \to j)}$ denotes the variables on the $\boldsymbol{C}_i$-side (excluding sepset)
- Thus, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ doesn't depend on root clique as long as the root is on the $\boldsymbol{C}_j$-side
- Two messages $m_{i \to j}$ and $m_{j \to i}$ are associated with each edge, one for each direction

Clique Tree Message Passing

- Formally, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ takes the form

$$m_{i \to j}(\boldsymbol{S}_{i,j}) = \sum_{\mathcal{V}_{\prec(i \to j)}} \prod_{\phi \in \mathcal{F}_{\prec(i \to j)}} \phi$$

  where $\mathcal{F}_{\prec(i \to j)}$ denotes the factors in the cliques on the $\boldsymbol{C}_i$-side and $\mathcal{V}_{\prec(i \to j)}$ denotes the variables on the $\boldsymbol{C}_i$-side (excluding sepset)
- Thus, the message from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$ doesn't depend on root clique as long as the root is on the $\boldsymbol{C}_j$-side
- Two messages $m_{i \to j}$ and $m_{j \to i}$ are associated with each edge, one for each direction
- If there are c cliques, there are $2(c-1)$ messages to compute

Sum-Product Belief Propagation

- For a clique tree $\mathcal{T}$, we say that $\boldsymbol{C}_i$ is *ready* to transmit to neighbor $\boldsymbol{C}_j$ when $\boldsymbol{C}_i$ has received all messages from its neighbors except $\boldsymbol{C}_j$
- Results in an asynchronous algorithm in which each clique computes and sends its message as soon as it is *ready*
- Importantly, messages are computed using *initial potential* ψ_i not modified potential β_i (otherwise, we are double-counting)

Algorithm CTree-SP-Calibrate

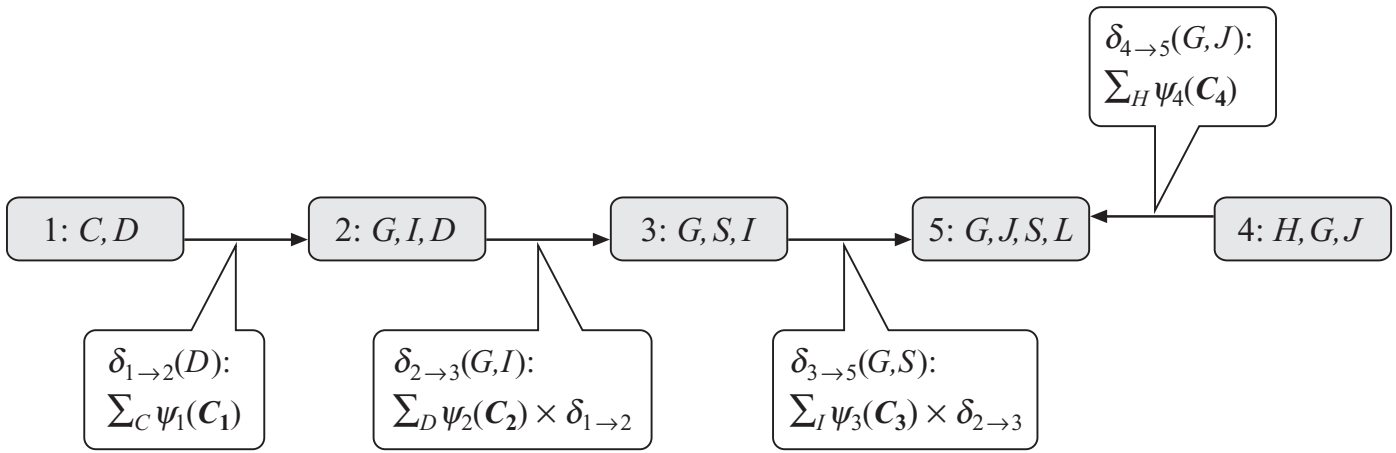
Require: Φ, $\mathcal{T}$

Initialize Cliques
while $\exists\ i, j$ such that i is ready to transmit to j **do**
  $\psi(\boldsymbol{C}_i) \leftarrow \psi_i \cdot \prod_{k \in (\text{Nb}_i - \{j\})} m_{k \to i}$
  $m_{i \to j}(\boldsymbol{S}_{i,j}) \leftarrow \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi(\boldsymbol{C}_i)$
end while
for each clique i **do**
  $\beta_i \leftarrow \psi_i \cdot \prod_{k \in \text{Nb}_i} m_{k \to i}$
end for

return $\{\beta_i\}$

Sum-Product Belief Propagation

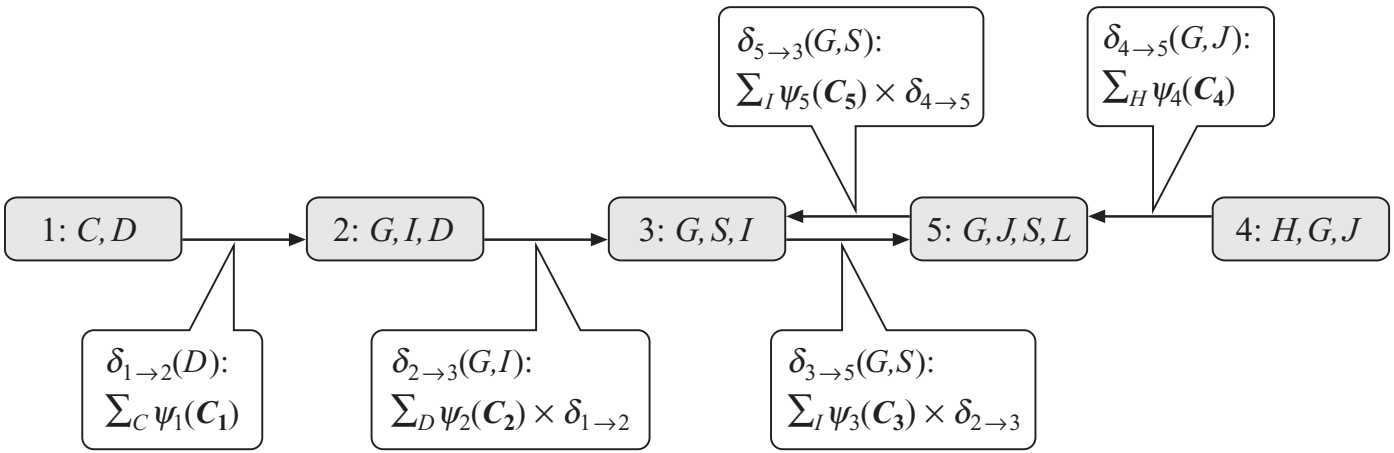
- Similar to a systematic process:
  1. **Upward pass**: Pick a root clique and send all messages *up* to the root
  2. **Downward pass**: Once the root has collected all messages, it sends messages *down* until the leaves are reached
- In sum-product, the root is the first node to receive all messages



Upward Pass

Sum-Product Belief Propagation

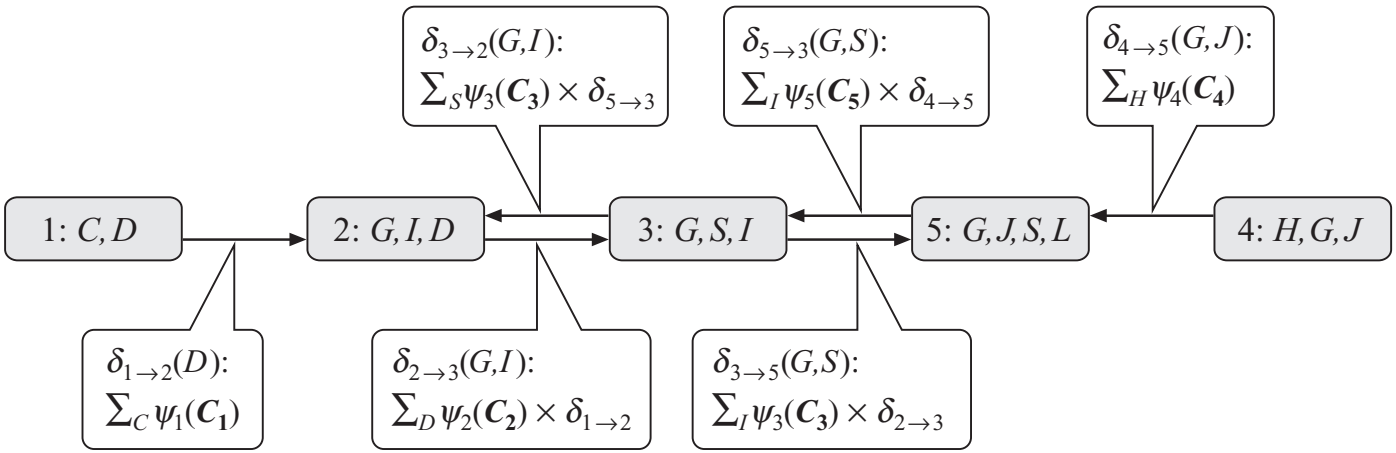
- Similar to a systematic process:
  1. **Upward pass**: Pick a root clique and send all messages *up* to the root
  2. **Downward pass**: Once the root has collected all messages, it sends messages *down* until the leaves are reached
- In sum-product, the root is the first node to receive all messages



Downward Pass

Sum-Product Belief Propagation

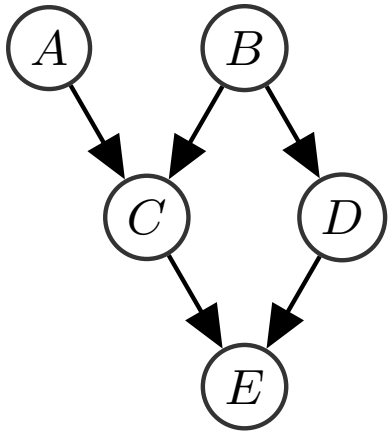
- Similar to a systematic process:
  1. **Upward pass**: Pick a root clique and send all messages *up* to the root
  2. **Downward pass**: Once the root has collected all messages, it sends messages *down* until the leaves are reached
- In sum-product, the root is the first node to receive all messages



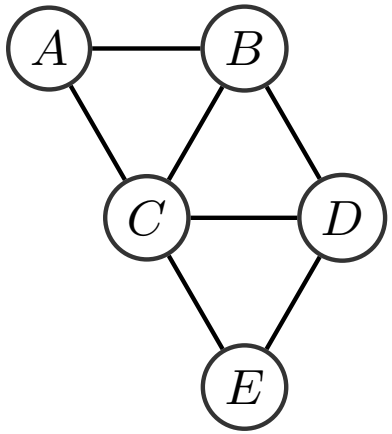
Downward Pass

Sum-Product Belief Propagation: Example

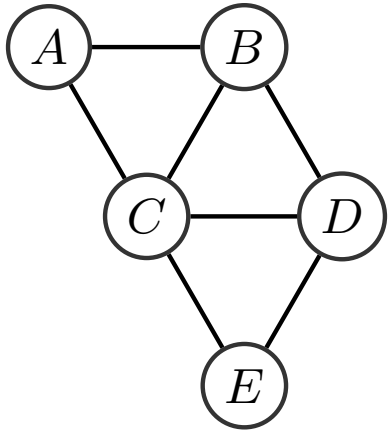
- Consider the following Bayesian network



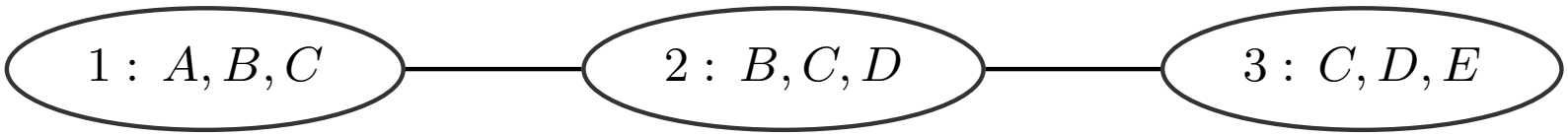
- We can compute the corresponding chordal graph via triangulation



Sum-Product Belief Propagation: Example



- Cliques are $\boldsymbol{C}_1 = \{A, B, C\}$, $\boldsymbol{C}_2 = \{B, C, D\}$, and $\boldsymbol{C}_3 = \{C, D, E\}$
- The corresponding clique tree is then



- Run through sum-product on board...

Sum-Product Belief Propagation

Belief Propagation (BP)

Input: A clique tree[2] with corresponding potentials Φ

1. Choose the root clique $\boldsymbol{C}_r$
2. Pass messages from leaves up to $\boldsymbol{C}_r$
3. Pass messages from $\boldsymbol{C}_r$ to leaves
4. Compute the *belief* β_i at each clique

- Computes posterior for all variables at once
- Running time is $2 \times$ cost of one variable elimination (upwards pass), which is $\mathcal{O}(mN_{\text{max}})$, where $N_{\text{max}} = k^{w_{K,\prec}}$ (see last lecture)

[2] We'll see where we get the clique tree later

Clique Tree Calibration

- Sum-product belief propagation results in a calibrated tree
- Belief propagation results in the marginal for each clique

$$\beta(\boldsymbol{C}_i) = \sum_{\mathcal{X}-\boldsymbol{C}_i} \tilde{P}(\mathcal{X})$$

Clique Tree Calibration

- Sum-product belief propagation results in a calibrated tree
- Belief propagation results in the marginal for each clique

$$\beta(\boldsymbol{C}_i) = \sum_{\mathcal{X}-\boldsymbol{C}_i} \tilde{P}(\mathcal{X})$$

- Two adjacent cliques $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ are **calibrated** if their beliefs agree:

$$\sum_{\boldsymbol{C}_i-\boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) = \sum_{\boldsymbol{C}_j-\boldsymbol{S}_{i,j}} \beta_j(\boldsymbol{C}_j)$$

- A clique tree is **calibrated** if all pairs of adjacent cliques are calibrated
- For a calibrated tree, we define the **clique beliefs** $\beta_i(\boldsymbol{C}_i)$ and the **sepset beliefs** as

$$\mu_{i,j}(\boldsymbol{S}_{i,j}) = \sum_{\boldsymbol{C}_i-\boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) = \sum_{\boldsymbol{C}_j-\boldsymbol{S}_{i,j}} \beta_j(\boldsymbol{C}_j)$$

Clique Tree Invariant

- For a calibrated tree

$$\beta_i = \psi_i \cdot \prod_{k \in \text{Nb}_i} m_{k \to i}$$

- By definition of the sepset marginals

$$\mu_{i,j}(S_{i,j}) = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i)$$

Clique Tree Invariant

- For a calibrated tree

$$\beta_i = \psi_i \cdot \prod_{k \in \mathsf{Nb}_i} m_{k \to i}$$

- By definition of the sepset marginals

$$\begin{aligned}
\mu_{i,j}(S_{i,j}) &= \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) \\
&= \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \cdot \prod_{k \in \mathsf{Nb}_i} m_{k \to i} \\
&= \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \cdot m_{j \to i} \cdot \prod_{k \in \mathsf{Nb}_i - \{j\}} m_{k \to i} \\
&= m_{j \to i} \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \cdot \prod_{k \in \mathsf{Nb}_i - \{j\}} m_{k \to i} \\
&= m_{j \to i} m_{i \to j}
\end{aligned}$$

Clique Tree Invariant

- At convergence, a calibrated clique tree yields

$$\tilde{P}_\Phi(\mathcal{X}) = \frac{\prod_{i \in \mathcal{V}_\mathcal{T}} \beta_i(\boldsymbol{C}_i)}{\prod_{(i-j) \in \mathcal{E}_\mathcal{T}} \mu_{i,j}(\boldsymbol{S}_{i,j})}$$

Clique Tree Invariant

- At convergence, a calibrated clique tree yields

$$\begin{aligned}
\tilde{P}_\Phi(\mathcal{X}) &= \frac{\prod_{i \in \mathcal{V}_\mathcal{T}} \beta_i(\boldsymbol{C}_i)}{\prod_{(i-j) \in \mathcal{E}_\mathcal{T}} \mu_{i,j}(\boldsymbol{S}_{i,j})} \\
&= \frac{\prod_{i \in \mathcal{V}_\mathcal{T}} \psi_i(\boldsymbol{C}_i) \prod_{k \in \mathsf{Nb}_i} m_{k \to i}}{\prod_{(i-j) \in \mathcal{E}_\mathcal{T}} m_{i \to j} m_{j \to i}} \\
&= \prod_{i \in \mathcal{V}_\mathcal{T}} \psi_i(\boldsymbol{C}_i) \\
&= \tilde{P}_\Phi(\mathcal{X})
\end{aligned}$$

Calibrated Clique Tree as a Distribution

- A calibrated tree $\mathcal{T}$ induces the following measure over $\mathcal{X}$, known as the **clique tree invariant**

$$Q_{\mathcal{T}} = \frac{\prod_{i \in \mathcal{V}_{\mathcal{T}}} \beta_i(\boldsymbol{C}_i)}{\prod_{(i-j) \in \mathcal{E}_{\mathcal{T}}} \mu_{i,j}(\boldsymbol{S}_{i,j})}$$

where

$$\mu_{i,j}(\boldsymbol{S}_{i,j}) = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) = \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j(\boldsymbol{C}_j)$$

Calibrated Clique Tree as a Distribution

- A calibrated tree $\mathcal{T}$ induces the following measure over $\mathcal{X}$, known as the **clique tree invariant**

$$Q_{\mathcal{T}} = \frac{\prod_{i \in \mathcal{V}_{\mathcal{T}}} \beta_i(\boldsymbol{C}_i)}{\prod_{(i-j) \in \mathcal{E}_{\mathcal{T}}} \mu_{i,j}(\boldsymbol{S}_{i,j})}$$

where

$$\mu_{i,j}(\boldsymbol{S}_{i,j}) = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) = \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j(\boldsymbol{C}_j)$$

Theorem

For a calibrated clique tree $\mathcal{T}$, $\tilde{P}_\Phi \propto Q_{\mathcal{T}}$ iff $\beta_i(\boldsymbol{C}_i) \propto \tilde{P}_\Phi(\boldsymbol{C}_i)$ for each $i \in \mathcal{V}_{\mathcal{T}}$

We can view the clique tree as an alternative representation of the joint measure that directly provides the clique marginals

Belief Update

- Consider a clique tree with an edge $(i-j)$ with $\boldsymbol{C}_i$ upstream of $\boldsymbol{C}_j$
- $\boldsymbol{C}_j$ first passes a message to $\boldsymbol{C}_i$ and having received all messages, $\boldsymbol{C}_i$ has its belief

$$\beta_i = \psi_i \cdot \prod_{k \in \text{Nb}_i} m_{k \to i}$$

- Later, when $\boldsymbol{C}_i$ passes its message back to $\boldsymbol{C}_j$, we don't pass back $m_{j \to i}$ (this would be double-counting)
- Instead, we were careful about how we generated the message

$$\begin{aligned} m_{i \to j} &= \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \prod_{k \in \{\text{Nb}_i\} - \{j\}} m_{k \to i} \\ &= \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \tilde{\beta}_i \end{aligned}$$

Belief Update

- Alternatively, we could multiply by all the messages and divide out $m_{j \to i}$ and others that would be double-counted
- This gives rise to a different expression for the messages

$$m_{i \to j} = \frac{\sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i}{m_{j \to i}}$$

  where we define $0/0 = 0$
- Requires that we keep track of sent messages

Belief Update

- Yields an alternative *sum-product-divide* belief propagation algorithm
  1. Each clique $\boldsymbol{C}_i$ maintains its fully updated beliefs β_i
  2. These beliefs are the product of the initial potentials and all the incoming messages
  3. Each sepset maintains the previous message along $(i - j)$ as $\mu_{i,j}$
  4. Whenever sending a new message along an edge, it is divided by the old message, removing doubly counted values
- Known as **belief update message passing** (alternatively, the *Lauritzen-Spiegelhalter algorithm*)

Belief Update

- Unlike sum-product, we are free to pass messages between any adjacent $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, *even if the sender is not ready*
- Related, the procedure is correct even if we send multiple messages
- This strategy is correct irregardless of which clique sends the message
- Equivalent to sum-product message passing
- At convergence, we have a calibrated tree

$$\mu_{i,j}(\boldsymbol{S}_{i,j}) = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) = \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j(\boldsymbol{C}_j)$$

Queries Outside a Clique

- So far, we have shown how to use belief propagation to compute clique marginals
- What about queries $P(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e})$ where $\boldsymbol{Y}$ spans multiple cliques?

Queries Outside a Clique

- So far, we have shown how to use belief propagation to compute clique marginals
- What about queries $P(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e})$ where $\boldsymbol{Y}$ spans multiple cliques?
- Naive approach: Create a new clique tree that includes all query variables in a single clique and then perform BP
  - Inefficient: May require separate trees (and BP) for different queries

Queries Outside a Clique

- So far, we have shown how to use belief propagation to compute clique marginals
- What about queries $P(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e})$ where $\boldsymbol{Y}$ spans multiple cliques?
- Naive approach: Create a new clique tree that includes all query variables in a single clique and then perform BP
  - Inefficient: May require separate trees (and BP) for different queries
- Better approach: Perform variable elimination over calibrated tree
  - We only need to consider the subtree that includes all query variables

Queries Outside a Clique

If a clique tree $\mathcal{T}$ is calibrated, so is any connected subtree $\mathcal{T}'$

1. Find a subtree $\mathcal{T}'$ of $\mathcal{T}$ that contains all query variables $\boldsymbol{Y}$
2. Pick a root node in $\mathcal{T}'$, $i \in \mathcal{V}_{\mathcal{T}'}$
3. Perform variable elimination of Scope$[\mathcal{T}'] \backslash \boldsymbol{Y}$ with factors β_r and

$$\phi_i = \frac{\beta_i}{\mu_{i,\text{upstream}(i)}}$$

for all $i \in \mathcal{V}_{\mathcal{T}'} - \{r\}$

Queries Outside a Clique

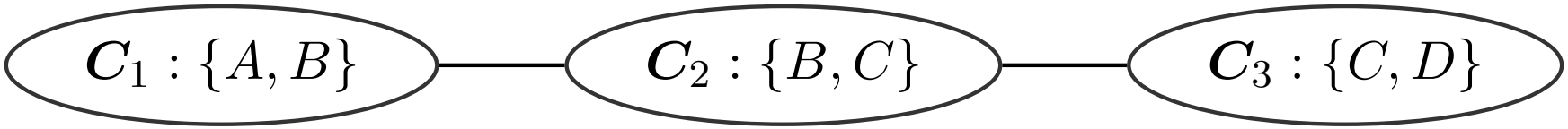
Algorithm 10.4 Out-of-clique inference in clique tree

Procedure CTree-Query (
  $\mathcal{T}$, // Clique tree over Φ
  $\{\beta_i\}, \{\mu_{i,j}\}$, // Calibrated clique and sepset beliefs for $\mathcal{T}$
  $\boldsymbol{Y}$ // A query
)

1 Let $\mathcal{T}'$ be a subtree of $\mathcal{T}$ such that $\boldsymbol{Y} \subseteq \mathit{Scope}[\mathcal{T}']$
2 Select a clique $r \in \mathcal{V}_{\mathcal{T}'}$ to be the root
3 $\Phi \leftarrow \beta_r$
4 **for** each $i \in \mathcal{V}_{\mathcal{T}'} - \{r\}$
5 $\phi \leftarrow \frac{\beta_i}{\mu_{i,p_r(i)}}$
6 $\Phi \leftarrow \Phi \cup \{\phi\}$
7 $\boldsymbol{Z} \leftarrow \mathit{Scope}[\mathcal{T}'] - \boldsymbol{Y}$
8 Let $\prec$ be some ordering over $\boldsymbol{Z}$
9 **return** Sum-Product-VE($\Phi, \boldsymbol{Z}, \prec$)

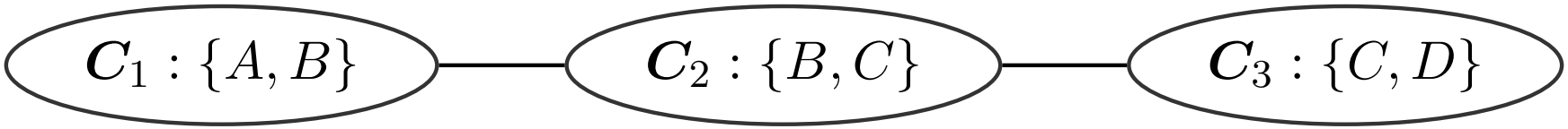
Queries Outside a Clique: Example

Consider the MRF $A - B - C - D$ with the corresponding clique tree $\mathcal{T}$



Queries Outside a Clique: Example

Consider the MRF $A - B - C - D$ with the corresponding clique tree $\mathcal{T}$



Algorithm 10.4 Out-of-clique inference in clique tree

Procedure CTree-Query (
  $\mathcal{T}$, // Clique tree over Φ
  $\{\beta_i\}, \{\mu_{i,j}\}$, // Calibrated clique and sepset beliefs for $\mathcal{T}$
  $\boldsymbol{Y}$ // A query
)

1 Let $\mathcal{T}'$ be a subtree of $\mathcal{T}$ such that $\boldsymbol{Y} \subseteq Scope[\mathcal{T}']$
2 Select a clique $r \in \mathcal{V}_{\mathcal{T}'}$ to be the root
3 $\Phi \leftarrow \beta_r$
4 **for** each $i \in \mathcal{V}_{\mathcal{T}'} - \{r\}$
5 $\phi \leftarrow \frac{\beta_i}{\mu_{i,p_r(i)}}$
6 $\Phi \leftarrow \Phi \cup \{\phi\}$
7 $\boldsymbol{Z} \leftarrow Scope[\mathcal{T}'] - \boldsymbol{Y}$
8 Let $\prec$ be some ordering over $\boldsymbol{Z}$
9 **return** Sum-Product-VE($\Phi, \boldsymbol{Z}, \prec$)

- $\mathcal{T}' = \boldsymbol{C}_2 - \boldsymbol{C}_3$
- Let $\boldsymbol{C}_3$ be the root
- $\phi_2 = \frac{\beta_2(B,C)}{\mu_{2,3}(C)}$
- $\Phi = \{\beta_3, \phi_2\}$
- $\boldsymbol{Z} = \{B, C, D\} - \{B, D\} = \{C\}$

$$\tilde{P}(B, D) = \sum_C \frac{\beta_2(B, C)\beta_3(C, D)}{\mu_{2,3}(C)}$$

Constructing Clique Trees

- Clique trees are a useful data structure for variable elimination and belief propagation, but how do we construct them?

Constructing Clique Trees

- Clique trees are a useful data structure for variable elimination and belief propagation, but how do we construct them?
- There are two basic approaches:
  1. Variable elimination (the cluster graph is a clique tree)
  2. Graph manipulation

Constructing Clique Trees

Theorem

For any clique tree $\mathcal{T}$, there exists a clique tree $\mathcal{T}'$ s.t.:

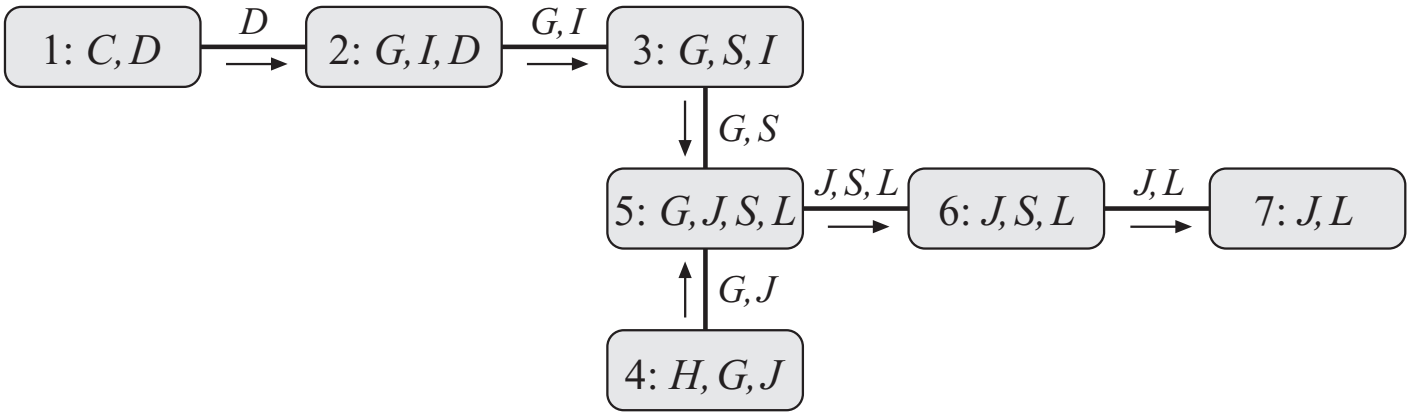
- *Each clique in $\mathcal{T}'$ is also a clique in $\mathcal{T}$*
- *There is no pair of cliques $\boldsymbol{C}_i, \boldsymbol{C}_j$ in $\mathcal{T}'$ such that $\boldsymbol{C}_i \subset \boldsymbol{C}_j$*

Constructing Clique Trees

Theorem

For any clique tree $\mathcal{T}$, there exists a clique tree $\mathcal{T}'$ s.t.:

- *Each clique in $\mathcal{T}'$ is also a clique in $\mathcal{T}$*
- *There is no pair of cliques $\boldsymbol{C}_i, \boldsymbol{C}_j$ in $\mathcal{T}'$ such that $\boldsymbol{C}_i \subset \boldsymbol{C}_j$*

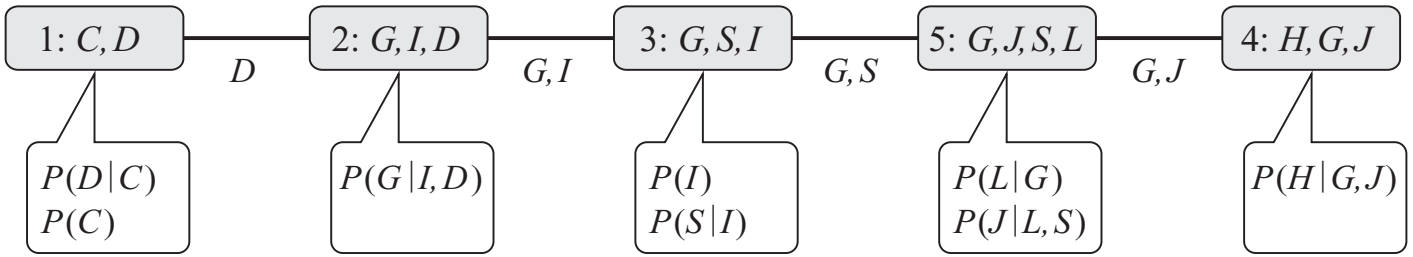


Constructing Clique Trees

Theorem

For any clique tree $\mathcal{T}$, there exists a clique tree $\mathcal{T}'$ s.t.:

- *Each clique in $\mathcal{T}'$ is also a clique in $\mathcal{T}$*
- *There is no pair of cliques $\boldsymbol{C}_i, \boldsymbol{C}_j$ in $\mathcal{T}'$ such that $\boldsymbol{C}_i \subset \boldsymbol{C}_j$*



Constructing Clique Trees via Variable Elimination

- Variable elimination gives rise to a cluster graph where each cluster $\boldsymbol{C}_i$ corresponds to a generated factor ψ_i
- The cluster graph is a tree and satisfies the running intersection property and is therefore a clique tree
- Recall from the variable elimination lecture that
  - Each factor in VE with ordering $\prec$ is a subset of a clique in $\mathcal{I}_{\Phi,\prec}$
  - Each maximal clique in $\mathcal{I}_{\Phi,\prec}$ is a factor in VE computation
- Each clique in $\mathcal{T}$ induced by VE is a clique in the induced graph $\mathcal{I}_{\Phi,\prec}$ and each (maximal) clique in $\mathcal{I}_{\Phi,\prec}$ is a clique in $\mathcal{T}$
- Thus, $\mathcal{T}$ is a clique tree for $\mathcal{I}_{\Phi,\prec}$
- Per last theorem, $\exists\ \mathcal{T}$ whose cliques are maximal cliques of $\mathcal{I}_{\Phi,\prec}$

Constructing Clique Trees from Chordal Graphs

- Recall that the induced graph $\mathcal{I}_{\Phi,\prec}$ is a chordal graph (no cycles > 3)
- **Theorem 4.12**: Every chordal graph has a clique tree
- We have seen that any clique tree can be used to perform inference

Constructing Clique Trees from Chordal Graphs

- Recall that the induced graph $\mathcal{I}_{\Phi,\prec}$ is a chordal graph (no cycles > 3)
- **Theorem 4.12**: Every chordal graph has a clique tree
- We have seen that any clique tree can be used to perform inference
- Construct a clique tree for H by generating its chordal graph H^*
  This process is known as **triangulation**

Constructing Clique Trees from Chordal Graphs

- Recall that the induced graph $\mathcal{I}_{\Phi,\prec}$ is a chordal graph (no cycles > 3)
- **Theorem 4.12**: Every chordal graph has a clique tree
- We have seen that any clique tree can be used to perform inference
- Construct a clique tree for H by generating its chordal graph H^*
  This process is known as **triangulation**
- Finding the minimum triangulation (where the largest clique is as small as possible) is NP-hard
- In most cases, one resorts to heuristic triangulation algorithms

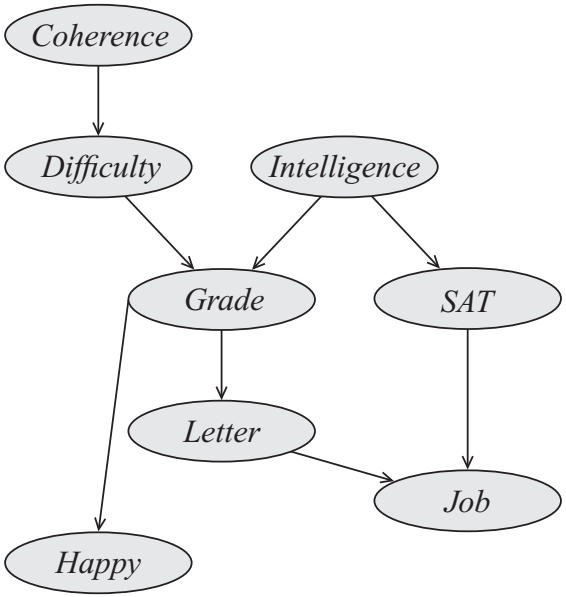
Constructing Clique Trees from Chordal Graphs

- Recall that the induced graph $\mathcal{I}_{\Phi,\prec}$ is a chordal graph (no cycles > 3)
- **Theorem 4.12**: Every chordal graph has a clique tree
- We have seen that any clique tree can be used to perform inference
- Construct a clique tree for H by generating its chordal graph H^*
  This process is known as **triangulation**
- Finding the minimum triangulation (where the largest clique is as small as possible) is NP-hard
- In most cases, one resorts to heuristic triangulation algorithms
- Having constructed the chordal graph H^*, we then find the maximal cliques (e.g., maximum cardinality search)

Constructing Clique Trees from Chordal Graphs

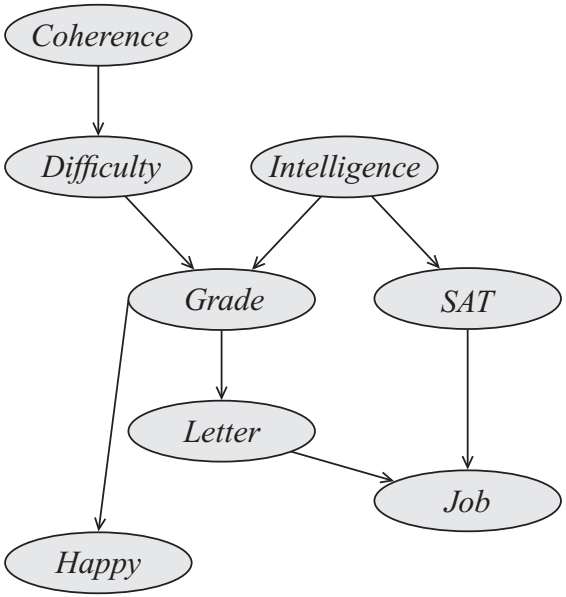
- Recall that the induced graph $\mathcal{I}_{\Phi,\prec}$ is a chordal graph (no cycles > 3)
- **Theorem 4.12**: Every chordal graph has a clique tree
- We have seen that any clique tree can be used to perform inference
- Construct a clique tree for H by generating its chordal graph H^*
  This process is known as **triangulation**
- Finding the minimum triangulation (where the largest clique is as small as possible) is NP-hard
- In most cases, one resorts to heuristic triangulation algorithms
- Having constructed the chordal graph H^*, we then find the maximal cliques (e.g., maximum cardinality search)
- Next, we need to determine the edges
  - Maximum cardinality search; or
  - Maximum spanning tree algorithm: Connect each pair $\boldsymbol{C}_i, \boldsymbol{C}_j$ by an edge with weight $|\boldsymbol{C}_i \cap \boldsymbol{C}_j|$ and find tree with maximal weight

Constructing Clique Trees from Chordal Graphs: Example

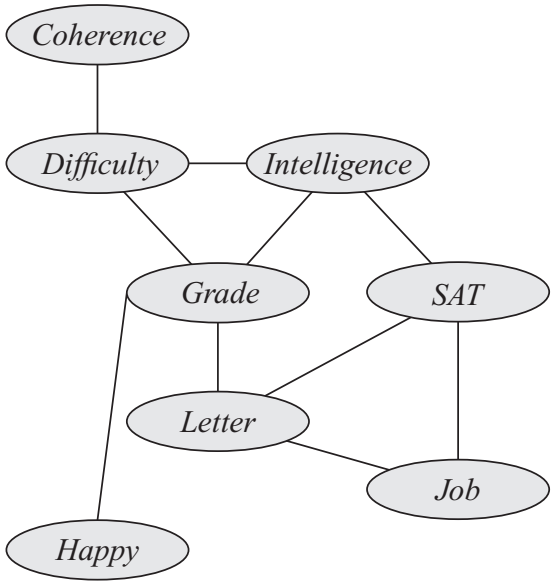


Original BN

Constructing Clique Trees from Chordal Graphs: Example

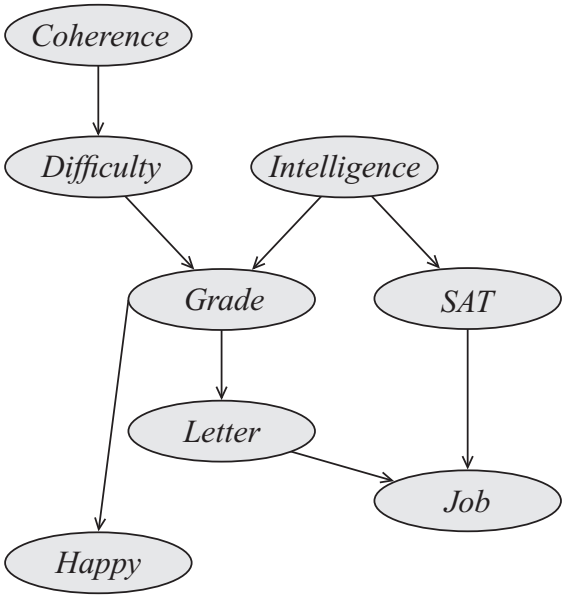


Original BN

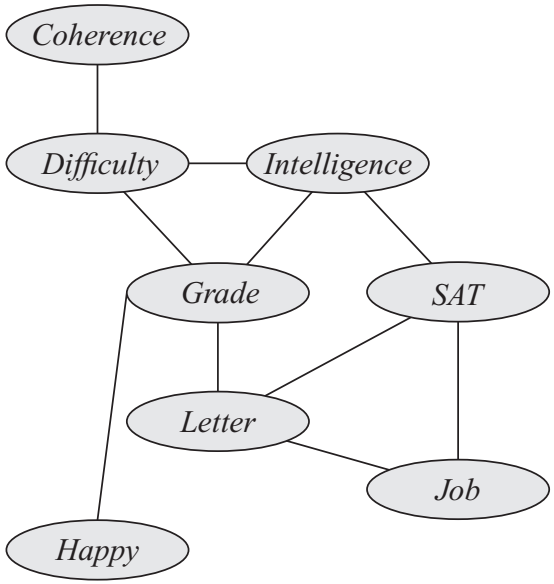


Moralized BN

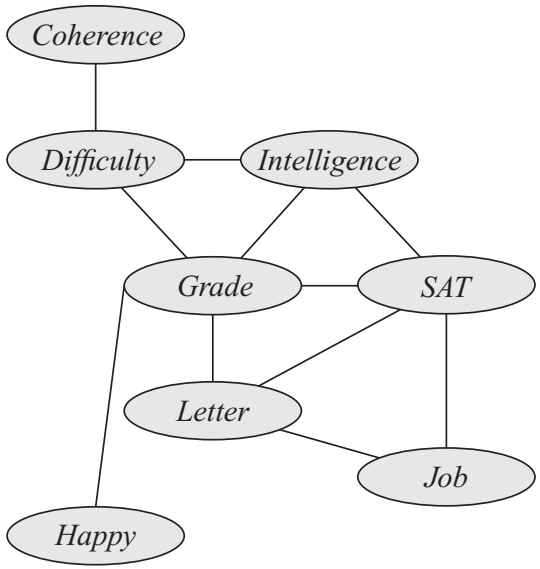
Constructing Clique Trees from Chordal Graphs: Example



Original BN

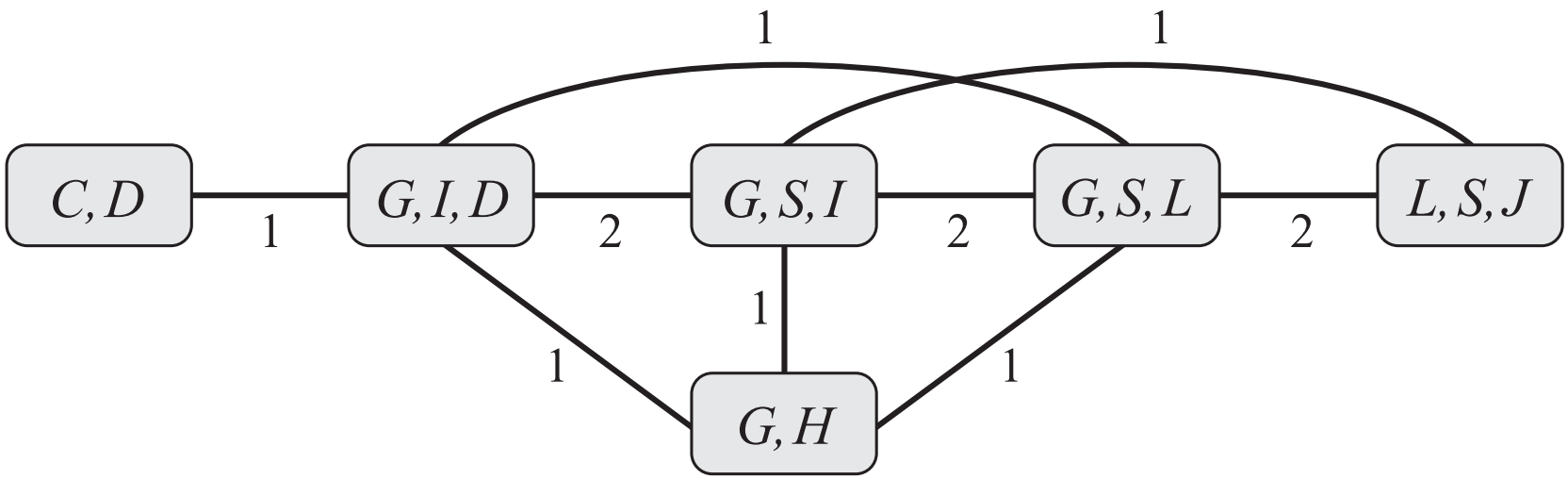


Moralized BN



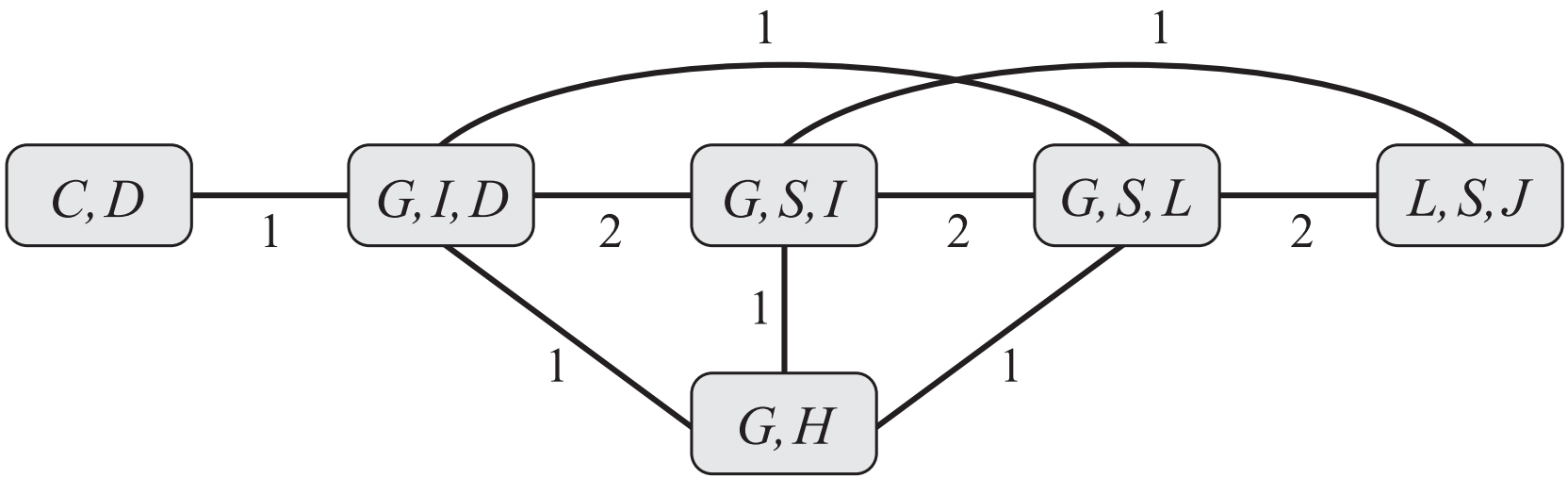
(Minimal) Triangulation

Constructing Clique Trees from Chordal Graphs: Example

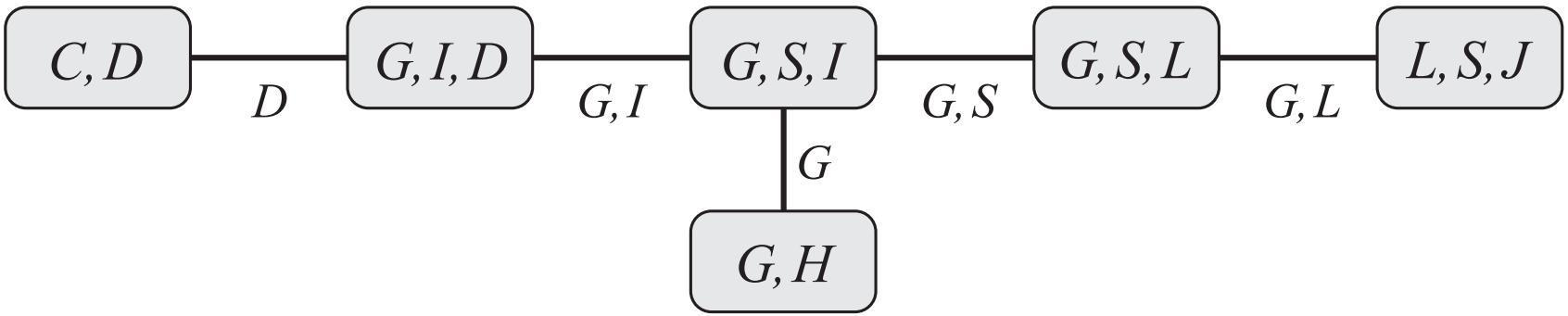


Cluster Graph

Constructing Clique Trees from Chordal Graphs: Example



Cluster Graph



Clique Tree