A Short Course on Graphical Models

3. The Junction Tree Algorithms

Mark Paskin
mark@paskin.org

Review: conditional independence

- Two random variables X and Y are **independent** (written $X \perp\!\!\!\perp Y$) iff

  $$p_X(\cdot) = p_{X|Y}(\cdot, y) \text{ for all } y$$

  If $X \perp\!\!\!\perp Y$ then Y gives us no information about X.

- X and Y are **conditionally independent given** Z (written $X \perp\!\!\!\perp Y \,|\, Z$) iff

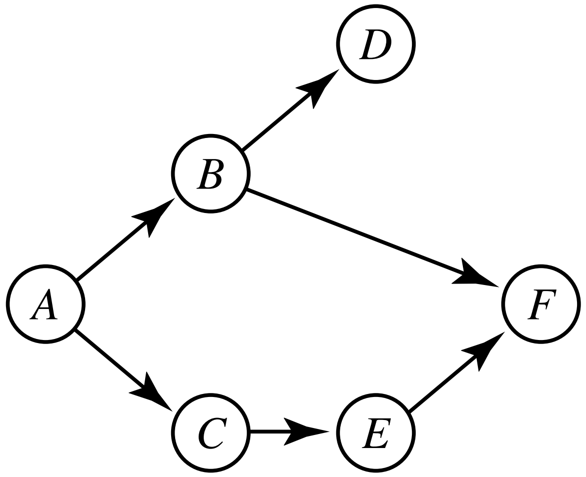
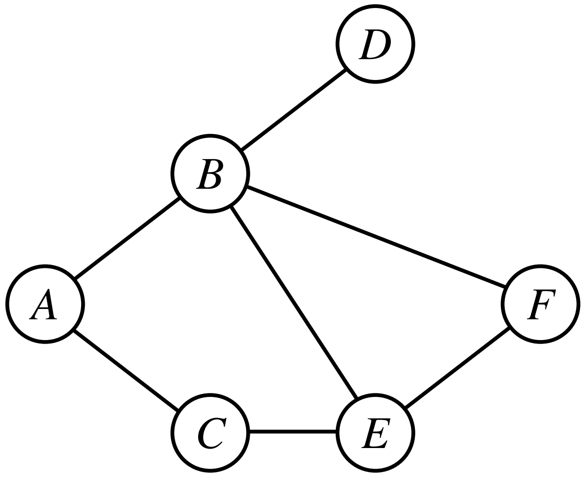
  $$p_{X|Z}(\cdot, z) = p_{X|YZ}(\cdot, y, z) \text{ for all } y \text{ and } z$$

  If $X \perp\!\!\!\perp Y \,|\, Z$ then Y gives us no new information about X once we know Z.

- We can obtain compact, factorized representations of densities by using the chain rule in combination with conditional independence assumptions.

- The Variable Elimination algorithm uses the distributivity of $\times$ over $+$ to perform inference efficiently in factorized densities.

Review: graphical models

Bayesian network	undirected graphical model
$p_A \cdot p_{B\|A} \cdot p_{C\|A} \cdot p_{D\|B} \cdot p_{E\|C} \cdot p_{F\|BE}$	$\frac{1}{Z} \cdot \psi_A \cdot \psi_{AB} \cdot \psi_{AC} \cdot \psi_{BD} \cdot \psi_{CE} \cdot \psi_{BEF}$
	
d-separation $\rightarrow$ cond. indep.	graph separation $\rightarrow$ cond. indep.

Moralization converts a Bayesian network into an undirected graphical model (but it does not preserve all of the conditional independence properties).

A notation for sets of random variables

It is helpful when working with large, complex models to have a good notation for sets of random variables.

- Let $X = (X_i : i \in V)$ be a **vector random variable** with density p.
- For each $A \subseteq V$, let $X_A \stackrel{\triangle}{=} (X_i : i \in A)$.
- For $A, B \subseteq V$, let $p_A \stackrel{\triangle}{=} p_{X_A}$ and $p_{A|B} \stackrel{\triangle}{=} p_{X_A|X_B}$.

Example. *If* $V = \{a, b, c\}$ *and* $A = \{a, c\}$ *then*

$$X = \begin{bmatrix} X_a \\ X_b \\ X_c \end{bmatrix} \text{ and } X_A = \begin{bmatrix} X_a \\ X_c \end{bmatrix}$$

where X_a*,* X_b*, and* X_c *are random variables.*

A notation for assignments

We also need a notation for dealing flexibly with functions of many arguments.

- An **assignment to** A is a set of index-value pairs $\mathbf{u} = \{(i, x_i) : i \in A\}$, one per index $i \in A$, where x_i is in the range of X_i.
- Let $\mathbb{X}_A$ be the set of assignments to X_A (with $\mathbb{X} \triangleq \mathbb{X}_V$).
- Building new assignments from given assignments:
  - Given assignments $\mathbf{u}$ and $\mathbf{v}$ to disjoint subsets A and B, respectively, their union $\mathbf{u} \cup \mathbf{v}$ is an assignment to $A \cup B$.
  - If $\mathbf{u}$ is an assignment to A then the **restriction of u to** $B \subseteq V$ is $\mathbf{u}_B \triangleq \{(i, x_i) \in \mathbf{u} : i \in B\}$, an assignment to $A \cap B$.
- If $\mathbf{u} = \{(i, x_i) : i \in A\}$ is an assignment and f is a function, then

$$f(\mathbf{u}) \triangleq f(x_i : i \in A)$$

Examples of the assignment notation

1. If p is the joint density of X then the marginal density of X_A is

$$p_A(\mathbf{v}) = \sum_{\mathbf{u} \in \mathbb{X}_{\overline{A}}} p(\mathbf{v} \cup \mathbf{u}), \qquad \mathbf{v} \in \mathbb{X}_A$$

   where $\overline{A} = V \setminus A$ is the complement of A.

2. If p takes the form of a normalized product of potentials, we can write it as

$$p(\mathbf{u}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{u}_C), \qquad \mathbf{u} \in \mathbb{X}$$

   where $\mathbf{C}$ is a set of subsets of V, and each ψ_C is a potential function that depends only upon X_C. The Markov graph of p has clique set $\mathbf{C}$.

Review: the inference problem

- Input:
  - a vector random variable $X = (X_i : i \in V)$;
  - a joint density for X of the form

$$p(\mathbf{u}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{u}_C)$$

  - an **evidence assignment w** to E; and
  - some **query variables** X_Q.
- Output: $p_{Q|E}(\cdot, \mathbf{w})$, the conditional density of X_Q given the evidence **w**.

Dealing with evidence

- From the definition of conditional probability, we have:

$$p_{\overline{E}|E}(\mathbf{u}, \mathbf{w}) = \frac{p(\mathbf{u} \cup \mathbf{w})}{p_E(\mathbf{w})} = \frac{\frac{1}{Z}\prod_{C \in \mathbf{C}} \psi_C(\mathbf{u}_C \cup \mathbf{w}_C)}{p_E(\mathbf{w})}$$

- For fixed evidence $\mathbf{w}$ on X_E, this is another normalized product of potentials:

$$p_{\overline{E}|\mathbf{w}}(\mathbf{u}) = \frac{1}{Z'} \prod_{C' \in \mathbf{C}'} \psi_{C'}(\mathbf{u}_{C'})$$

  where $Z' \triangleq Z \times p_E(\mathbf{w})$, $C' \triangleq C \backslash E$, and $\psi_{C'}(\mathbf{u}) \triangleq \psi_C(\mathbf{u} \cup \mathbf{w}_C)$.

- Thus, to deal with evidence, we simply instantiate it in all clique potentials.

The reformulated inference problem

Given a joint density for $X = (X_i : i \in V)$ of the form

$$p(\mathbf{u}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{u}_C)$$

compute the **marginal density** of X_Q:

$$\begin{aligned} p_Q(\mathbf{v}) &= \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q}}} p(\mathbf{v} \cup \mathbf{u}) \\ &= \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q}}} \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C) \end{aligned}$$

Review: Variable Elimination

- For each $i \in \overline{Q}$, push in the sum over X_i and compute it:

$$\begin{aligned}
p_Q(\mathbf{v}) &= \frac{1}{Z} \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q}}} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C) \\
&= \frac{1}{Z} \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q} \setminus \{i\}}} \sum_{\mathbf{w} \in \mathbb{X}_{\{i\}}} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C \cup \mathbf{w}_C) \\
&= \frac{1}{Z} \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q} \setminus \{i\}}} \prod_{\substack{C \in \mathbf{C} \\ i \notin C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C) \sum_{\mathbf{w} \in \mathbb{X}_{\{i\}}} \prod_{\substack{C \in \mathbf{C} \\ i \in C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C \cup \mathbf{w}) \\
&= \frac{1}{Z} \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q} \setminus \{i\}}} \prod_{\substack{C \in \mathbf{C} \\ i \notin C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C) \cdot \psi_{E_i}(\mathbf{v}_{E_i} \cup \mathbf{u}_{E_i})
\end{aligned}$$

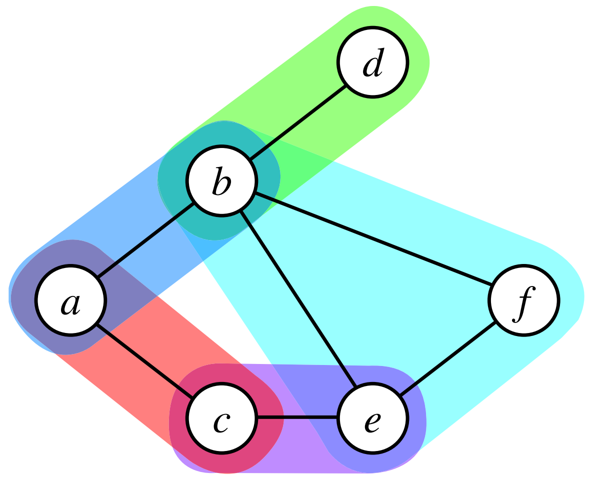
  This creates a new **elimination clique** $E_i = \bigcup_{\substack{C \in \mathbf{C} \\ i \in C}} C \setminus \{i\}$.

- At the end we have $p_Q = \frac{1}{Z} \psi_Q$ and we normalize to obtain p_Q (and Z).

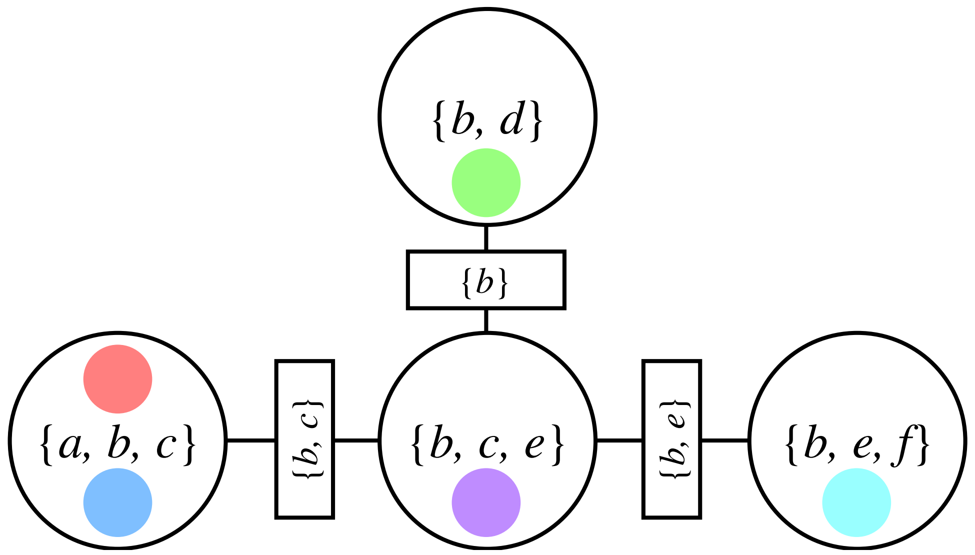
From Variable Elimination to the junction tree algorithms

- Variable Elimination is **query sensitive**: we must specify the query variables in advance. This means each time we run a different query, we must re-run the entire algorithm.
- The **junction tree algorithms** generalize Variable Elimination to avoid this; they **compile** the density into a data structure that supports the **simultaneous** execution of a large class of queries.

Junction trees



G



T

A cluster graph T is a **junction tree** for G if it has these three properties:

1. **singly connected**: there is exactly one path between each pair of clusters.
2. **covering**: for each clique A of G there is some cluster C such that $A \subseteq C$.
3. **running intersection**: for each pair of clusters B and C that contain i, each cluster on the unique path between B and C also contains i.

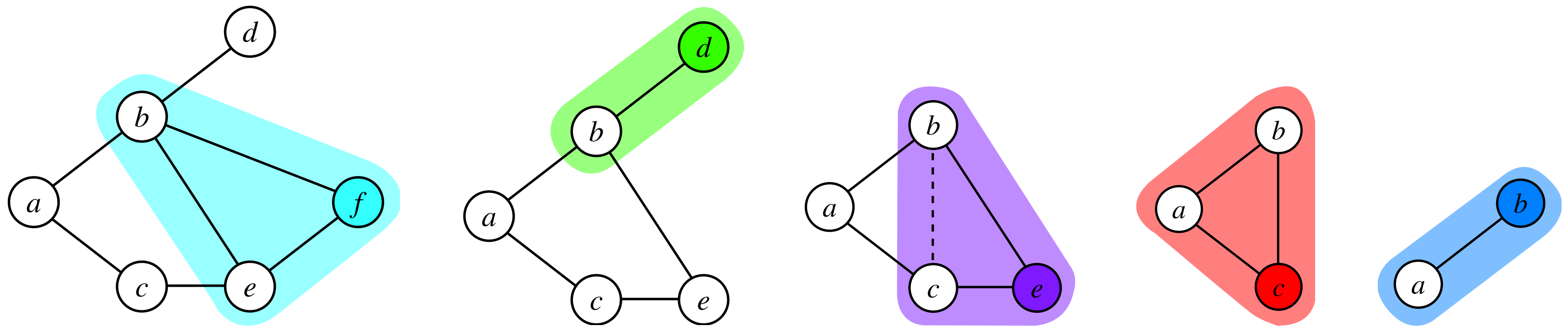
Building junction trees

- To build a junction tree:
  1. Choose an ordering of the nodes and use Node Elimination to obtain a set of elimination cliques.
  2. Build a **complete** cluster graph over the **maximal** elimination cliques.
  3. Weight each edge $\{B, C\}$ by $|B \cap C|$ and compute a maximum-weight spanning tree.

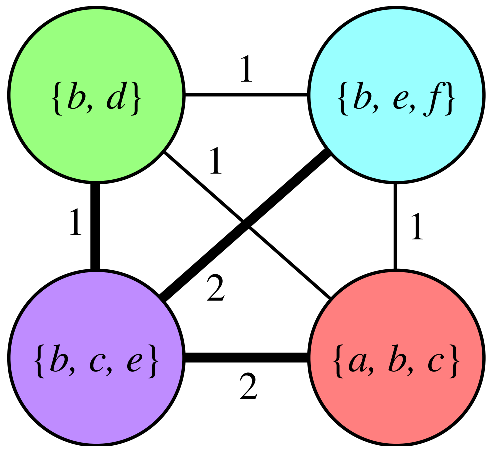
  This spanning tree is a junction tree for G (see Cowell *et al.*, 1999).
- Different junction trees are obtained with different elimination orders and different maximum-weight spanning trees.
- Finding the junction tree with the smallest clusters is an NP-hard problem.

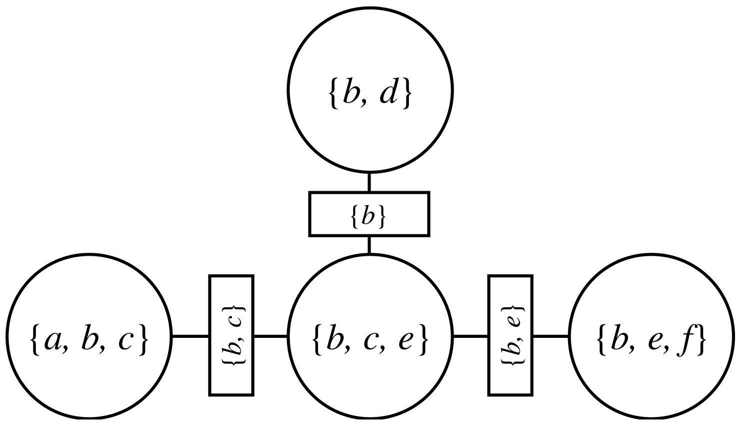
An example of building junction trees

1. Compute the elimination cliques (the order here is f, d, e, c, b, a).



2. Form the complete cluster graph over the maximal elimination cliques and find a maximum-weight spanning tree.





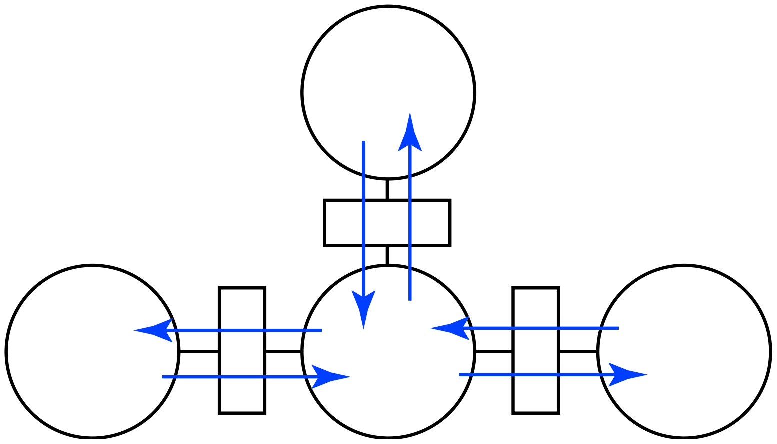
Decomposable densities

- A factorized density
$$p(\mathbf{u}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{u}_C)$$
is **decomposable** if there is a junction tree with cluster set $\mathbf{C}$.
- To convert a factorized density p to a decomposable density:
  1. Build a junction tree T for the Markov graph of p.
  2. Create a potential ψ_C for each cluster C of T and initialize it to unity.
  3. Multiply each potential ψ of p into the cluster potential of one cluster that covers its variables.
- Note: this is possible only because of the **covering** property.

The junction tree inference algorithms

The junction tree algorithms take as input a decomposable density and its junction tree. They have the same distributed structure:

- Each cluster starts out knowing only its local potential and its neighbors.
- Each cluster sends one message (potential function) to each neighbor.
- By combining its local potential with the messages it receives, each cluster is able to compute the marginal density of its variables.



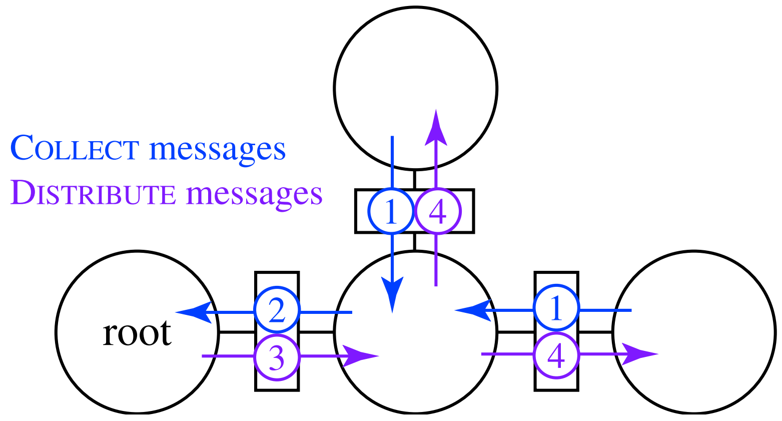
The message passing protocol

The junction tree algorithms obey the **message passing protocol**:

> Cluster B is allowed to send a message to a neighbor C only after it has received messages from all neighbors except C.

One admissible schedule is obtained by choosing one cluster R to be the root, so the junction tree is directed. Execute $\textsc{Collect}(R)$ and then $\textsc{Distribute}(R)$:

1. $\textsc{Collect}(C)$: For each child B of C, recursively call $\textsc{Collect}(B)$ and then pass a message from B to C.
2. $\textsc{Distribute}(C)$: For each child B of C, pass a message to B and then recursively call $\textsc{Distribute}(B)$.



The Shafer–Shenoy Algorithm

- The **message sent from** B **to** C is defined as

$$\mu_{BC}(\mathbf{u}) \triangleq \sum_{\mathbf{v} \in \mathbb{X}_{B \setminus C}} \psi_B(\mathbf{u} \cup \mathbf{v}) \prod_{\substack{(A,B) \in \mathbf{E} \\ A \neq C}} \mu_{AB}(\mathbf{u}_A \cup \mathbf{v}_A)$$

- Procedurally, cluster B computes the product of its local potential ψ_B and the messages from all clusters **except** C, marginalizes out all variables that are not in C, and then sends the result to C.

- Note: μ_{BC} is well-defined because the junction tree is **singly connected**.

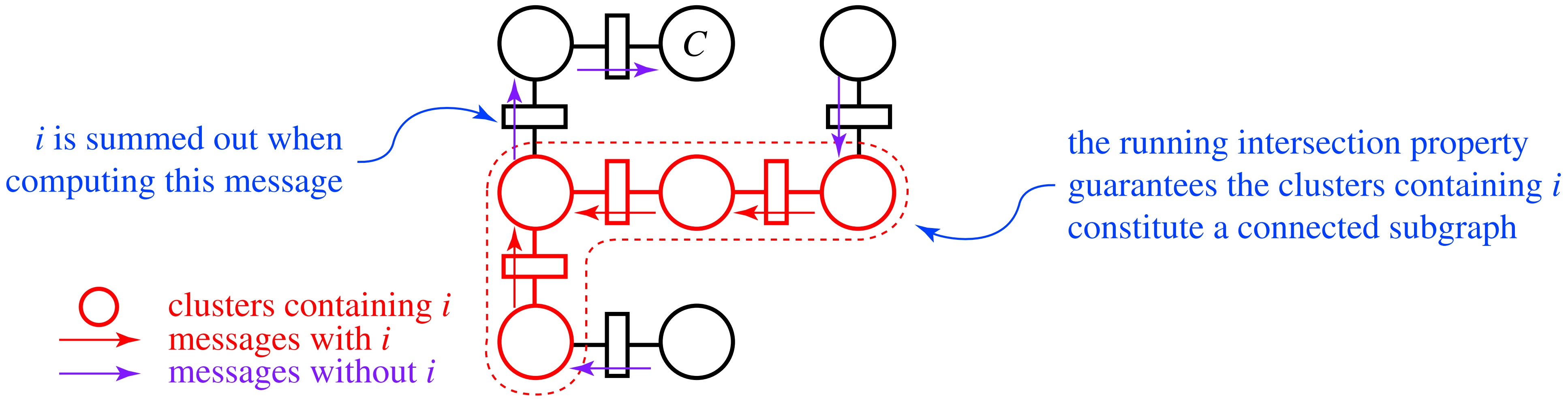
- The **cluster belief at** C is defined as

$$\beta_C(\mathbf{u}) \triangleq \psi_C(\mathbf{u}) \prod_{(B,C) \in \mathbf{E}} \mu_{BC}(\mathbf{u}_B)$$

  This is the product of the cluster's local potential and the messages received from **all** of its neighbors. We will show that $\beta_C \propto p_C$.

Correctness: Shafer–Shenoy is Variable Elimination in all directions at once

- The cluster belief β_C is computed by alternatingly multiplying cluster potentials together and summing out variables.
- This computation is of the same basic form as Variable Elimination.
- To prove that $\beta_C \propto p_C$, we must prove that no sum is "pushed in too far".
- This follows directly from the **running intersection property**:



C
i is summed out when computing this message
the running intersection property guarantees the clusters containing i constitute a connected subgraph
clusters containing i
messages with i
messages without i

The HUGIN Algorithm

- Give each cluster C and each separator S a potential function over its variables. Initialize:

$$\begin{aligned} \phi_C(\mathbf{u}) &= \psi_C(\mathbf{u}) \\ \phi_S(\mathbf{u}) &= 1 \end{aligned}$$

- To pass a message from B to C over separator S, update

$$\begin{aligned} \phi_S^*(\mathbf{u}) &= \sum_{\mathbf{v} \in \mathbb{X}_{B \setminus S}} \phi_B(\mathbf{u} \cup \mathbf{v}) \\ \phi_C^*(\mathbf{u}) &= \phi_C(\mathbf{u}) \frac{\phi_S^*(\mathbf{u}_S)}{\phi_S(\mathbf{u}_S)} \end{aligned}$$

- After all messages have been passed, $\phi_C \propto p_C$ for all clusters C.

Correctness: HUGIN is a time-efficient version of Shafer–Shenoy

- Each time the Shafer–Shenoy algorithm sends a message or computes its cluster belief, it multiplies together messages.
- To avoid performing these multiplications repeatedly, the HUGIN algorithm caches in ϕ_C the running product of ψ_C and the messages received so far.
- When B sends a message to C, it **divides out** the message C sent to B from this running product.

Summary: the junction tree algorithms

Compile time:

1. Build the junction tree T:
   (a) Obtain a set of maximal elimination cliques with Node Elimination.
   (b) Build a weighted, complete cluster graph over these cliques.
   (c) Choose T to be a maximum-weight spanning tree.
2. Make the density decomposable with respect to T.

Run time:

1. Instantiate evidence in the potentials of the density.
2. Pass messages according to the message passing protocol.
3. Normalize the cluster beliefs/potentials to obtain conditional densities.

Complexity of junction tree algorithms

- Junction tree algorithms represent, multiply, and marginalize potentials:

	tabular	Gaussian
storing ψ_C	$O(k^{\lvert C\rvert})$	$O(\lvert C\rvert^2)$
computing $\psi_{B\cup C} = \psi_B \times \psi_C$	$O(k^{\lvert B\cup C\rvert})$	$O(\lvert B\cup C\rvert^2)$
computing $\psi_{C\setminus B}(\mathbf{u}) = \sum_{\mathbf{v}\in\mathbb{X}_B} \psi_C(\mathbf{u}\cup\mathbf{v})$	$O(k^{\lvert C\rvert})$	$O(\lvert B\rvert^3\lvert C\rvert^2)$

- The number of clusters in a junction tree and therefore the number of messages computed is $O(|V|)$.

- Thus, the time and space complexity is dominated by the size of the largest cluster in the junction tree, or the **width** of the junction tree:
  - In tabular densities, the complexity is **exponential** in the width.
  - In Gaussian densities, the complexity is **cubic** in the width.

Generalized Distributive Law

- The general problem solved by the junction tree algorithms is the **sum-of-products** problem: compute

$$p_Q(\mathbf{v}) \propto \sum_{\mathbf{u} \in \mathbb{X}_{\overline{Q}}} \prod_{C \in \mathbf{C}} \psi_C(\mathbf{v}_C \cup \mathbf{u}_C)$$

- The property used by the junction tree algorithms is the distributivity of $\times$ over $+$; more generally, we need a **commutative semiring**:

$[0, \infty)$	$(+, 0)$	$(\times, 1)$	sum-product
$[0, \infty)$	$(\max, 0)$	$(\times, 1)$	max-product
$(-\infty, \infty]$	$(\min, \infty)$	$(+, 0)$	min-sum
$\{\mathrm{T}, \mathrm{F}\}$	$(\vee, \mathrm{F})$	$(\wedge, \mathrm{T})$	Boolean

- Many other problems are of this form, including **maximum a posteriori** inference, the Hadamard transform, and matrix chain multiplication.

Summary

- The junction tree algorithms generalize Variable Elimination to the efficient, simultaneous execution of a large class of queries.
- The algorithms take the form of message passing on a graph called a junction tree, whose nodes are clusters, or sets, of variables.
- Each cluster starts with one potential of the factorized density. By combining this potential with the potentials it receives from its neighbors, it can compute the marginal over its variables.
- Two junction tree algorithms are the Shafer–Shenoy algorithm and the HUGIN algorithm, which avoids repeated multiplications.
- The complexity of the algorithms scales with the width of the junction tree.
- The algorithms can be generalized to solve other problems by using other commutative semirings.