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

- $X$ and $Y$ are conditionally independent given $Z$ (written $X \perp Y | Z$) iff
  \[ p_{X|Z}(\cdot, z) = p_{X|YZ}(\cdot, y, z) \text{ for all } y \text{ and } z \]

  If $X \perp Y | Z$ 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

<table>
<thead>
<tr>
<th>Bayesian network</th>
<th>undirected graphical model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_A \cdot p_{B</td>
<td>A} \cdot p_{C</td>
</tr>
</tbody>
</table>

$d$-separation $\rightarrow$ cond. indep. $\quad$ 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 $X_A \triangleq (X_i : i \in A)$.
- For $A, B \subseteq V$, let $p_A \triangleq p_{X_A}$ and $p_{A|B} \triangleq p_{X_A|X_B}$.

**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 $u = \{(i, x_i) : i \in A\}$, one per index $i \in A$, where $x_i$ is in the range of $X_i$.

• Let $X_A$ be the set of assignments to $X_A$ (with $X \triangleq X_V$).

• Building new assignments from given assignments:
  - Given assignments $u$ and $v$ to disjoint subsets $A$ and $B$, respectively, their union $u \cup v$ is an assignment to $A \cup B$.
  - If $u$ is an assignment to $A$ then the **restriction of $u$ to** $B \subseteq V$ is $u_B \triangleq \{(i, x_i) \in u : i \in B\}$, an assignment to $A \cap B$.

• If $u = \{(i, x_i) : i \in A\}$ is an assignment and $f$ is a function, then
  $$f(u) \triangleq f(x_i : i \in A)$$
Examples of the assignment notation

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

$$p_A(v) = \sum_{u \in X_{\bar{A}}} p(v \cup u), \quad v \in X_A$$

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

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

$$p(u) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi_C(u_C), \quad u \in X$$

where $\mathbf{C}$ is a set of subsets of $V$, and each $\psi_C$ 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
    \[
    p(u) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(u_C)
    \]
  - an evidence assignment \( w \) to \( E \); and
  - some query variables \( X_Q \).
- Output: \( p_{Q|E}(\cdot, w) \), the conditional density of \( X_Q \) given the evidence \( w \).
Dealing with evidence

- From the definition of conditional probability, we have:

\[
p_{E|E}(u, w) = \frac{p(u \cup w)}{p_E(w)} = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(u_C \cup w_C)
\]

- For fixed evidence \( w \) on \( X_E \), this is another normalized product of potentials:

\[
p_{E|w}(u) = \frac{1}{Z'} \prod_{C' \in \mathcal{C}'} \psi_{C'}(u_{C'})
\]

where \( Z' \triangleq Z \times p_E(w) \), \( \mathcal{C}' \triangleq \mathcal{C} \setminus E \), and \( \psi_{C'}(u) \triangleq \psi_C(u \cup w_C) \).

- 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

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

compute the **marginal density** of $X_Q$:

$$p_Q(v) = \sum_{u \in X_Q} p(v \cup u)$$

$$= \sum_{u \in X_Q} \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(v_C \cup u_C)$$
Review: Variable Elimination

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

$$p_Q(v) = \frac{1}{Z} \sum_{u \in X_{\overline{Q}\{i\}}} \prod_{C \in C} \psi_C(v_C \cup u_C)\psi_E(v_E \cup u_E)$$

$$= \frac{1}{Z} \sum_{u \in X_{\overline{Q}\{i\}}} \prod_{C \in C} \psi_C(v_C \cup u_C) \sum_{w \in X_{\{i\}}} \prod_{C \in C} \psi_C(v_C \cup u_C \cup w)$$

$$= \frac{1}{Z} \sum_{u \in X_{\overline{Q}\{i\}}} \prod_{C \in C} \psi_C(v_C \cup u_C) \cdot \psi_{E_i}(v_{E_i} \cup u_{E_i})$$

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

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

\[ p(u) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(u_C) \]

is decomposable if there is a junction tree with cluster set \( \mathcal{C} \).

• 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 \( \psi_C \) for each cluster \( C \) of \( T \) and initialize it to unity.
  3. Multiply each potential \( \psi \) 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 COLLECT$(R)$ and then DISTRIBUTE$(R)$:

1. COLLECT$(C)$: For each child $B$ of $C$, recursively call COLLECT$(B)$ and then pass a message from $B$ to $C$.

2. DISTRIBUTE$(C)$: For each child $B$ of $C$, pass a message to $B$ and then recursively call DISTRIBUTE$(B)$.
The Shafer–Shenoy Algorithm

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

$$
\mu_{BC}(u) \triangleq \sum_{v \in X_B \setminus C} \psi_B(u \cup v) \prod_{(A,B) \in E} \mu_{AB}(u_A \cup v_A)
$$

- Procedurally, cluster $B$ computes the product of its local potential $\psi_B$ 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: $\mu_{BC}$ is well-defined because the junction tree is singly connected.

- The cluster belief at $C$ is defined as

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

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 $\beta_C$ 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:

```
\begin{itemize}
  \item $i$ is summed out when computing this message
  \item Messages with $i$ will be summed out
  \item Messages without $i$ will not be summed out
  \item The running intersection property guarantees the clusters containing $i$ constitute a connected subgraph
\end{itemize}
```
The HUGIN Algorithm

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

\[
\begin{align*}
\phi_C(u) &= \psi_C(u) \\
\phi_S(u) &= 1
\end{align*}
\]

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

\[
\begin{align*}
\phi^*_S(u) &= \sum_{v \in \mathcal{X}_{B \setminus S}} \phi_B(u \cup v) \\
\phi^*_C(u) &= \phi_C(u) \frac{\phi^*_S(u_S)}{\phi_S(u_S)}
\end{align*}
\]

• 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 $\phi_C$ the running product of $\psi_C$ 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:
  
  storing $\psi_C$ : \( O(k^{|C|}) \) \( O(|C|^2) \)
  
  computing $\psi_{B \cup C} = \psi_B \times \psi_C$ : \( O(k^{|B \cup C|}) \) \( O(|B \cup C|^2) \)
  
  computing $\psi_{C \setminus B}(u) = \sum_{v \in X_B} \psi_C(u \cup v)$ : \( O(k^{|C|}) \) \( O(|B|^3|C|^2) \)

- 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

\[ p_Q(v) \propto \sum_{u \in \mathcal{X} - Q} \prod_{C \in \mathcal{C}} \psi_C(v_C \cup u_C) \]

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

<table>
<thead>
<tr>
<th>[0, \infty))</th>
<th>(+, 0)</th>
<th>((\times), 1)</th>
<th>sum-product</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, \infty))</td>
<td>((\max), 0)</td>
<td>((\times), 1)</td>
<td>max-product</td>
</tr>
<tr>
<td>((-\infty, \infty])</td>
<td>((\min), (\infty))</td>
<td>(+, 0)</td>
<td>min-sum</td>
</tr>
<tr>
<td>{T, F}</td>
<td>((\lor), F)</td>
<td>((\land), T)</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

- 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.