Graphical Models

Lecture 11:
Clique Trees

Andrew McCallum
mcallum@cs.umass.edu

Thanks to Noah Smith and Carlos Guestrin for some slide materials.
Inference

• Last two lectures: variable elimination
  – Sum out variables one at a time.
• Today: alternative algorithms.
  – Also based on factors.
  – Central data structure: *clique tree*. 
Formalisms

Lecture 5

Markov Network

triangulate

clique tree
Every maximal clique becomes a vertex.

Connect vertices with overlapping variables

Tree structure? then “Clique Tree”

(If graph was triangulated, always get a tree structure.)
Clique Tree

“Sep-set” = \{A, D\}

For each edge, intersection of r.v.s separates the rest in \(\mathcal{H}\).
“Cluster Graph”

• A cluster graph for a set of factors Φ over X is an undirected graph, each of whose nodes i is associated with a subset Ci ⊆ X.

• A cluster graph must be “family preserving”: each factor φ ∈ Φ must be associated with a cluster Ci (denoted α(φ)), such that Scope[φ] ⊆ Ci.

• Each edge between a pair of clusters Ci and Cj is associated with a sepset Si,j ⊆ Ci ∩ Cj.

• “Cluster Tree” a cluster graph that is a tree.
“Running Intersection” Property

- Cluster tree $\mathcal{T}$ (with vertices $C_i...$) has the running intersection property if, whenever there is a variable $X$ such as $X \in C_i$ and $X \in C_j$, then $X$ is also in every cluster in the (unique) path in $\mathcal{T}$ between $C_i$ and $C_j$. 

![Diagram of cluster tree]

ABC $\rightarrow$ BCD $\rightarrow$ CDE
“Clique Tree” Definition
aka “Junction Tree,” or “Join Tree”

• Given an undirected graph $\mathcal{H}$, a tree $\mathcal{T}$ is a clique tree for $\mathcal{H}$ if:
  – Each node in $\mathcal{T}$ corresponds to a clique in $\mathcal{H}$
  – Each maximal clique in $\mathcal{H}$ is a node in $\mathcal{T}$
  – Each sepset $S_{i,j}$ separates the variables strictly on one side of the edge from the variables on the other side.

$S_{i,j} = C_i \cap C_j$

\textit{In other words}

\textit{Clique Tree} = cluster tree that satisfies running intersection property.
How to Obtain a Clique Tree

Option 1

• Start with factor graph or undirected graph.
• Do variable elimination. On iteration $i$:
  – Let $\psi_i$ be the intermediate factor
  – Let $\tau_i$ be the marginalized factor
  – Let $C_i$ be the set of variables involved in $\psi_i$
• One node per $C_i$, edge $C_i$-$C_j$ when $\tau_i$ gets used in computing $\psi_j$. 
Example

• Eliminate S.
Example

- Eliminate S.
- $\psi_1 = \phi_{\text{FAS}} \cdot \phi_{\text{SR}} \cdot \phi_{\text{SH}}$
Example

- Eliminate S.

- $\psi_1 = \phi_{\text{FAS}} \cdot \phi_{\text{SR}} \cdot \phi_{\text{SH}}$
Example

• Eliminate S.

• $\psi_1 = \phi_{\text{FAS}} \cdot \phi_{\text{SR}} \cdot \phi_{\text{SH}}$

• $\tau_1 = \sum_S \psi_1$
Example

• Eliminate S.

• $\psi_1 = \Phi_{FAS} \cdot \Phi_{SR} \cdot \Phi_{SH}$

• $\tau_1 = \sum_S \psi_1$
Example

- Eliminate $R$.
- $\psi_2 = \tau_1$
- $\tau_2 = \sum_R \psi_2$
Example

• Eliminate R.

• $\psi_2 = \tau_1$

• $\tau_2 = \sum_R \psi_2$
Example

- Eliminate H.
- $\psi_3 = \tau_2$
- $\tau_3 = \sum_H \psi_3$
Example

• Eliminate H.

• $\psi_3 = \tau_2$

• $\tau_3 = \sum_H \psi_3$
Example

- Eliminate A.
- $\psi_4 = \tau_3 \phi_A$
- $\tau_4 = \sum_A \psi_3$
Example

• Eliminate A.

• $\psi_4 = \tau_3 \phi_A$

• $\tau_4 = \sum_A \psi_3$
Example

- Eliminate $F$.
- $\psi_5 = \tau_4 \phi_F$
- $\tau_5 = \sum_F \psi_5$
Example

- Eliminate $F$.
- $\psi_5 = \tau_4 \phi_F$
- $\tau_5 = \sum_F \psi_5$
Alternative Ordering

• \{H, R, S, A, F\}
How to Obtain a Clique Tree

**Option 1**

- Start with factor graph or undirected graph.
- Do variable elimination.
- One node per $C_i$, edge $C_i-C_j$ when $\tau_i$ gets used in computing $\psi_j$.
- Result is always a clique tree!
  - Running intersection property.
  - Different orderings will give different clique trees.
Reducing a Clique Tree

• Given a clique tree $T$ for a set of factors $\Phi$, there is always a reduced clique tree $T'$ such that
  – all clique-vertices in $T'$ are in $T$
  – no clique vertex in $T'$ is a subset of another.

• Construction exploits the running intersection property.
Reduced Clique Tree

SR

HS

SAF

FA

F
Reduced Clique Tree

SR

HS

SAF

FA

F

SR

HS

SAF
Reduced Clique Tree

FASRH  -->  FARH  -->  FAR  -->  FA  -->  F
Reduced Clique Tree

FASRH → FARH → FAR → FA → F

FASRH
How to Obtain a Clique Tree

Option 2

1. Given factors, construct the undirected graph.
2. Triangulate to get a chordal graph.
3. Find maximal cliques in the chordal graph.
4. Construct a tree from the “cluster graph” of maximal cliques.

Actually the method we discussed in Lecture 5
Step 2: Triangulate

- NP-hard in general to get the smallest one.
- Use heuristics.
Step 3: Find Maximal Cliques

- Maximum cardinality search: gives an induced graph with no fill edges (assuming chordal graph as input, which we have).
- We proved last time that every maximal clique in the induced graph equates to the scope of an intermediate factor from VE.
- Other approaches exist.
Maximum Cardinality Search for VE Ordering

• Start with undirected graph on $X$, all nodes unmarked.

• For $i = |X|$ to 1:
  – Let $Y$ be the unmarked variable in $X$ with the largest number of marked neighbors
  – $\pi(Y) = i$
  – Mark $Y$.

• Eliminate using permutation $\pi$. 
Example

- First node is arbitrary.
Example

• First node is arbitrary.
Example

- 1: A, S
- 0: R, H
Example

- 2: A
- 1: R, H
Example

- 1: R, H
Example

- 1: H
Example

- Order: \{H, R, A, S, F\}
How to Obtain a Clique Tree

Option 2

1. Given factors, construct the undirected graph.
2. Triangulate to get a chordal graph.
3. Find maximal cliques in the chordal graph.
4. Construct a tree from the “cluster graph” of maximal cliques.
   – Results from VE with the ordering from maximum cardinality search; or use maximum spanning tree (edge weights = sepset sizes).
Example

• Order: \{H, R, A, S, F\}
Example (Reduce)

- Order: \{H, R, A, S, F\}

Diagram:
- HS
- RS
- ASF
- Flu
- S.I.
- R.N.
- All.
- H.

Connections:
- HS to RS
- RS to ASF
- Flu to S.I.
- S.I. to R.N.
- S.I. to All.
- All. to H.
Now You Know ...

• How to construct a clique tree.

• Also called a “junction tree” or a “join tree.”
Running Intersection Property

- For connected clique-vertices $C_i$ and $C_j$, the sepset $S_{i,j}$ is $C_i \cap C_j$, and it separates the variables on the $C_i$ side from those on the $C_j$ side.
Running Intersection Property

For each edge, intersection of r.v.s separates the rest in \( \mathcal{H} \).

Lecture 5
Family Preservation

- Every factor in the original graph is associated with one clique-vertex in the clique tree.
Family Preservation Example
Inference

• Last two lectures: variable elimination
  – Sum out variables one at a time.
• Today: alternative algorithms.
  – Also based on factors.
  – Central data structure: clique tree.

• What are these algorithms?
Clique Trees and Variable Elimination

- Each product-factor $\psi$ belongs to a clique-vertex.
- Each $\tau$ is a “message” from one clique-vertex to another.
Example

- \{H, R, S, A, F\}
Example

- \{H, R, S, A, F\}

Just happens to be that in this graph with this ordering there is a one-to-one correspondence between original \(\phi\)s and product factors \(\psi\), because there was only one factor touching each variable-to-eliminate.
Messages

• Notice that the variable elimination ordering implies a direction to the edges.

• Messages go “upstream.”

• This leads to a clique-tree-centric view of VE, in which the same calculations happen as in VE.
Example

- \{H, R, S, A, F\}
Example

- \{H, R, S, A, F\}
- Reduced version
Example

- \{H, R, S, A, F\}
- Reduced version
- Initial potentials

\[ \nu_{\text{FAS}} = \phi_{FAS} \phi_A \phi_F \]

Go through math that gets done for VE here.
Message Passing in the Clique Tree

- Calculate **initial potentials** for each clique-vertex.
  \[ \nu_j(C_j) = \prod_{\phi \in \Phi_j} \phi \]

- Choose an arbitrary **root** clique \( C_r \). This imposes a directionality on the graph (think of the root as a *sink*).

- Pass messages “upstream”:
  \[ \delta_{i \rightarrow j} = \sum_{C_i \setminus S_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \rightarrow i} \]

Ordering can be different from the VE ordering that might have created the clique tree.
Example

- \{H, R, S, A, F\}
Message Passing in the Clique Tree

\[ \delta_{i \rightarrow j} = \sum_{C_i \setminus S_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \rightarrow i} \]

- Sum out variables that are not involved in \( C_j \)
- Original factors involving the family of \( C_i \)
- Incoming messages from other neighbors of \( i \)
Message Passing in the Clique Tree

\[ \delta_{i \rightarrow j} = \sum_{S_{i,j} \in C_i \setminus S_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \rightarrow i} \]
Message Passing in the Clique Tree

• Notice that we have a recursive structure here: message-sum of products of message-sums of products of message-sums of ...

$$\delta_{i \rightarrow j} = \sum_{C_i \setminus S_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \rightarrow i}$$
Message Passing in the Clique Tree

- Notice that you have to work “upward.”
- Root will be last.

\[
\delta_{i \rightarrow j} = \sum_{C_i \setminus S_{i,j}} \nu_i \prod_{k \in \text{Neighbors}_i \setminus \{j\}} \delta_{k \rightarrow i}
\]
Message Passing in the Clique Tree

• Notice that you have to work “upward.”
• Root $r$ will be last.

$$\beta_r = \nu_r \prod_{k \in \text{Neighbors}_r} \delta_{k \rightarrow r}$$

$$= \sum \prod_{X \setminus C_i} \prod_{\phi \in \Phi} \phi$$

$$= Z \cdot P(C_r)$$
The Algorithm

• Input: clique tree $\mathcal{T}$, factors $\Phi$, root $C_r$

• For each clique $C_i$, calculate $v_i$

• While $C_r$ is still waiting on incoming messages:
  – Choose a $C_i$ that has received all of its incoming messages.
  – Calculate and send the message from $C_i$ to $C_{\text{upstream-neighbors(i)}}$

• Return $\beta_r$
Obtaining Marginals

• Given a set of query variables $Q$ that is contained in some clique $C$:
  – Make that clique the root.
  – Run the upward pass of clique tree $VE$, resulting in factor $\beta$.
  – Return $\sum_{C \setminus Q} \beta$.

• Later: How to get marginals for $Q$ that are not contained in one clique.
Going Farther

• We can modify this algorithm to give us the marginal probability of every random variable in the network.
  – Naively: run clique tree VE with each clique as the root.
\[ \delta_{SR \rightarrow SAF} \]

\[ \delta_{HS \rightarrow SAF} \]

root
Same message as needed for previous root!
root

$\delta_{\text{SAF} \rightarrow \text{SR}}$

$\delta_{\text{HS} \rightarrow \text{SAF}}$
Going Farther

• We can modify this algorithm to give us the marginal probability of every random variable in the network.
  – Naively: run clique tree VE with each clique as the root.
  – Better: notice that the same messages get used on different runs; only two messages per clique tree edge.
Sum-Product Message Passing

• Each clique tree vertex $C_i$ passes messages to each of its neighbors once it’s ready to do so.
  – This is asynchronous; might want to be careful about scheduling.

• At the end, for all $C_i$:

$$\beta_i = \nu_i \prod_{k \in \text{Neighbors}_i} \delta_{k \rightarrow i}$$

  – This is the unnormalized marginal for $C_i$. 
\[\beta_{SR} \quad \delta_{SR\rightarrow SAF} \quad \delta_{SAF\rightarrow SR} \quad \beta_{HS} \quad \delta_{HS\rightarrow SAF} \quad \delta_{SAF\rightarrow HS} \quad \beta_{SAF} \quad \text{root}\]
Calibration

- Two adjacent cliques $C_i$ and $C_j$ are calibrated when:

$$\sum_{C_i \setminus S_{i,j}} \beta_i = \sum_{C_j \setminus S_{i,j}} \beta_j$$
Calibration

• Two adjacent cliques $C_i$ and $C_j$ are calibrated when:

$$\sum_{C_i \setminus S_{i,j}} \beta_i = \sum_{C_j \setminus S_{i,j}} \beta_j = \mu_{i,j}(S_{i,j})$$
Sum-Product Message Passing

• Computes the marginal probability of all variables using only twice the computation of the upward pass.
• Results in a calibrated clique tree.
• Attractive if we expect different kinds of queries.
Calibrated Clique Tree as a Graphical Model

- Original (unnormalized) factor model and calibrated clique tree represent the same (unnormalized) measure:

\[
\prod_{\phi \in \Phi} \phi = \frac{\prod_{C \in \text{Vertices}(T)} \beta_C}{\prod_{S \in \text{Edges}(T)} \mu_S}
\]
Calibrated Clique Tree as a Graphical Model

\[
\prod_{C \in \text{Vertices}(T)} \beta_C \prod_{S \in \text{Edges}(T)} \mu_S = \prod_{C \in \text{Vertices}(T)} \nu_C \prod_{C' \in \text{Neighbors}(C)} \delta_{C' \rightarrow C} \prod_{S = \in \text{Edges}(T): S = C \cap C'} \mu_{C \cap C'} \\
\prod_{C \in \text{Vertices}(T)} \nu_C \prod_{C' \in \text{Neighbors}(C)} \delta_{C' \rightarrow C} = \prod_{S = \in \text{Edges}(T): S = C \cap C'} \sum \beta_C \prod_{C \in \text{Vertices}(T)} \nu_C \prod_{C' \in \text{Neighbors}(C)} \delta_{C' \rightarrow C} \\
= \prod_{S = \in \text{Edges}(T): S = C \cap C'} \sum \nu_C \prod_{C \in \text{Vertices}(T)} \nu_C \prod_{C' \in \text{Neighbors}(C)} \delta_{C' \rightarrow C} \prod_{S = \in \text{Edges}(T): S = C \cap C'} \sum \nu_C \prod_{C' \in \text{Neighbors}(C')} \prod_{C'' \in \text{Neighbors}(C)} \delta_{C'' \rightarrow C}
\]
Calibrated Clique Tree as a Graphical Model

\[
\prod_{C \in \text{Vertices}(T)} \beta_C \prod_{S \in \text{Edges}(T)} \mu_S = \ldots
\]

\[
= \frac{\prod_{C \in \text{Vertices}(T)} \nu_C \prod_{C' \in \text{Neighbors}(C)} \delta_{C' \rightarrow C}}{\prod_{S = \in \text{Edges}(T) : S = C \cap C'} \delta_{C' \rightarrow C} \sum_{C \setminus S} \nu_C \prod_{C'' \in \text{Neighbors}(C) \setminus \{C'\}} \delta_{C'' \rightarrow C}}
\]

\[
= \frac{\prod_{C \in \text{Vertices}(T)} \nu_C \prod_{C' \in \text{Neighbors}(C)} \delta_{C' \rightarrow C}}{\prod_{S = \in \text{Edges}(T) : S = C \cap C'} \delta_{C' \rightarrow C} \delta_{C \rightarrow C'}}
\]

\[
= \prod_{C \in \text{Vertices}(T)} \nu_C
\]

\[
= \prod_{\phi \in \Phi} \phi
\]

\[
\mu_{i,j} = \delta_{C_i \rightarrow C_j} \delta_{C_j \rightarrow C_i}
\]
What You Now Know

• Can reinterpret variable elimination as message passing in a clique tree.
• Can share computation to get lots of marginals with only double the cost of VE.
  – Sum-product belief propagation
• Calibrated clique tree is an alternative representation of the original Gibbs distribution.