Cluster trees and message propagation

CS3710 Advanced AI
Tomas Singliar

Outline

- Simple graphs: trees and polytrees
- Cluster graphs and clique trees
  - running intersection, sepsetsMessage propagation ( = VE )
- Message passing VE in detail
- Caching, out-of-clique queries, DP
- Incremental updating
- Constructing clique trees
  - variable elimination
- VE and BP: Pros & cons and tradeoffs
Trees and polytrees

**Tree**: one directed path from the root to each node

**PolyTree**: one undirected trail from the root to each node

Undirected representation: a tree graph, treewidth=1

---

Clique trees

- VE works on *factors*
- Make factor a data structure
  - Sends and receives messages
- Cluster graph for set of factors, each node $i$ is associated with a subset (cluster) $C_i$ of .
  - Family-preserving: each factor’s variables are completely embedded in a cluster
Clique tree properties

- **Sepset**  \( S_{ij} = C_i \cap C_j \)
  - **separation** set: Variables \( X \) on one side of sepset are separated from the variables \( Y \) on the other side in the factor graph given variables in \( S \)

- **Running intersection**
  - if \( C_i \) and \( C_j \) both contain \( X \), then all cliques on the unique path between them do

---

Clique trees

Running intersection:
Clique trees involving \( S \) form a connected subtree.

**Initial potentials** \( \pi^0_i \):
Assign factors to cliques and multiply them. Have respect for families!

(Tree need not be minimal)
Message Passing VE

**Query for P(J)**

- Eliminate C: \( \tau_1(D) = \sum_C \pi_1[C, D] \)

Message sent from [C,D] to [G,I,D]

\[ \pi_1[G,I,D] \]


\[ \pi_1[G,I,D] = \tau_1(D) \times \pi_1[G,I,D] \]

---

Message Passing VE

**Query for P(J)**

- Eliminate D: \( \tau_2(G,I) = \sum_D \pi_2[G,I,D] \)


\[ \pi_2[G,S,I] \]


\[ \pi_2[G,S,I] = \tau_2(G,I) \times \pi_2[G,S,I] \]
Message Passing VE

- Query for \( P(J) \)
  - Eliminate \( I \): \( \tau_i(G,S) = \sum \pi_i[G,S,I] \)

Message sent from \([G,S,I]\) to \([G,J,S,L]\)

\[ \pi_4[G,J,S,L] = \tau_4(G,S) \times \pi_4^0[G,J,S,L] \]

[G,J,S,L] is not ready!

Message sent from \([H,G,J]\) to \([G,J,S,L]\)

All messages received at \([G,J,S,L]\)

\[ \pi_4[G,J,S,L] = \tau_4(G,J) \times \pi_4^0[G,J,S,L] \]

And so on...
Message Passing VE

- Chose \([J,L]\) as the root clique
- Could we have chosen otherwise?

\[
\begin{align*}
\text{C,D} & \rightarrow \text{G,I,D} \rightarrow \text{G,S,I} \\
\text{D} & \rightarrow \text{G,I} \downarrow \text{G,S} \\
\text{G,J,S,L} & \rightarrow \text{J,S,L} \rightarrow \text{J,L} \\
\text{G,J} & \uparrow \text{J,S,L} \rightarrow \text{J,L} \\
\text{H,G,J} &
\end{align*}
\]

Observation:
Some messages did not change!
\([D], [G,I], …\)

Notation:
number the cliques and denote the messages
\(\delta_{i \rightarrow j}\)
Correctness of VE on clique trees

- Message summarizes information in the part of tree it separates
  \[ \delta_{i \to j}(S_y) = \sum_{V < (i \to j)} \prod_{\phi \in F_{i \to j}} \phi \]

- Proof is by induction from leaves

- Base case – leaf clique \( C_i \)
  \[ \delta_{i \to j}(C_i \cap C_j) = \sum_{C_i-S_y} \pi_i^0(C_i) = \sum_{C_i-S_y \phi \in F_i} \prod \phi \]

---

Correctness of VE on clique trees

- Induction case: non-leaf clique \( C_j \) sending to \( C_i \) with children \( C_{i_1}, ..., C_{i_k} \)
  \[ \sum_{V_{< (i \to j)}} \prod \phi = \sum_{V_{< (i \to j - 1)}} \prod_{\phi \in F_{i \to j - 1}} \phi \times \left( \prod_{\phi \in F_{i_1 \to j - 1}} \phi \right) \times \left( \prod_{\phi \in F_{i_2 \to j - 1}} \phi \right) \times \left( \prod_{\phi \in F_{i_k \to j - 1}} \phi \right) \]

- by intersection property, the unions are disjoint
  \[ V_{< (i \to j)} = \bigcup_{w=1}^{k} V_{< (i_w \to j)} \quad F_{< (i \to j)} = \bigcup_{w=1}^{k} F_{< (i_w \to j)} \]

- Then
  \[ \sum_{V_{< (i \to j)}} \prod \phi = \sum_{V_{< (i \to j)}} \left( \prod_{\phi \in F_{i \to j}} \phi \right) \times \sum_{V_{< (i_1 \to j)}} \left( \prod_{\phi \in F_{i_1 \to j}} \phi \right) \times \sum_{V_{< (i_2 \to j)}} \left( \prod_{\phi \in F_{i_2 \to j}} \phi \right) \times \sum_{V_{< (i_k \to j)}} \left( \prod_{\phi \in F_{i_k \to j}} \phi \right) \]
Correctness of VE on clique trees

By induction hypothesis

\[ \delta_{i \rightarrow j}(S_y) = \sum_{V \in \{(i \rightarrow j) \phi \in \mathcal{F}_{i \rightarrow j}\}} \prod \phi \]

\[ \sum_{V \in \{(i \rightarrow j) \phi \in \mathcal{F}_{i \rightarrow j}\}} \prod \phi = \sum_{V \in \{(i \rightarrow j) \phi \in \mathcal{F}_{i \rightarrow j}\}} \prod_{i} \pi^0_i \times \prod_{m=1}^{v} \delta_{m \rightarrow i} = \delta_{i \rightarrow j} \]

Then the root clique has the correct marginal:

\[ \pi_r(C_r) = \sum_{X-C_r} \sum_{i \in \text{Children}(r)} \prod \delta_{i \rightarrow r} = \sum_{X-C_r} \sum_{i \in \text{Children}(r)} \prod_{V \in \{(i \rightarrow r) \phi \in \mathcal{F}_{i \rightarrow r}\}} \]

\[ \sum_{X-C_r} \sum_{i \in \text{Children}(r)} \prod \pi^0_i(C_i) \prod_{j \in \text{Children}(i)} \delta_{j \rightarrow i} = \ldots = \sum_{X-C_r} \sum_{k \in V \backslash C_r} \prod \pi^0_k(C_k) \]

Message passing VE

- Message order is only partial
- Computes marginals for any node Y
  - Results in a calibrated clique tree
- Often, many marginals desired
  - Inefficient to re-run inference
  - One distinct message per edge & direction
- Recap: three kinds of factor objects
  - initial, final potentials and messages
Message Passing VE

- Shafer-Shenoy algorithm
  - asynchronous implementation of two passes: upward and downward

- Asynchronously do:
  - node $i$ ready to send $m$ to node $j$ when it has received a message from all other nodes
  - Send message $\delta_{i \rightarrow j} = \sum_{C_i \cap S_j} \pi_i \times \prod_{k \in N(i) \setminus j} \delta_{k \rightarrow i}$

- Marginalize root clique’s ancillary vars

Message Passing: BP

- Graphical model of a **distribution**
  - More edges = larger expressive power
  - Clique tree also a model of distribution
  - Message passing preserves model but changes parameterization

- Different but equivalent algorithm
### Factor division

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Inverse of factor product

Message Passing: BP

- Each node: multiply all the messages and divide by the one coming from node we send to
  - Clearly the same as VE

$$
\delta_{i \rightarrow j} = \frac{\sum_{C_i \sim \tilde{S}_i} \pi_j}{\delta_{j \rightarrow i}} = \frac{\sum_{C_i \sim \tilde{S}_i} \prod_{k \in N(i)} \delta_{k \rightarrow i}}{\delta_{j \rightarrow i}} = \sum_{C_i \sim \tilde{S}_i} \prod_{k \in N(i) \setminus j} \delta_{k \rightarrow i}
$$
Message Passing: BP

\[ \pi_i^2(A, B) \quad \pi_3^2(B, C) \quad \pi_4^2(C, D) \]

Store the last message on the edge and divide each passing message by the last stored.

\[ \delta_{2\rightarrow 3} = \sum_B \pi_3^0(B, C) \]
\[ \pi_3(C, D) = \pi_4^0(C, D) \sum_B \pi_3^0(B, C) \]
\[ \delta_{3\rightarrow 2}(C) = \sum_D \pi_3(C, D) \]
\[ \pi_3(B, C) = \frac{\pi_3^0(B, C) \times \delta_{3\rightarrow 2}(C)}{\delta_{2\rightarrow 3}(C)} = \frac{\pi_3^0(B, C) \times \sum_D \pi_3^0(C, D) \times \delta_{3\rightarrow 2}(C)}{\delta_{2\rightarrow 3}(C)} = \pi_3^0(B, C) \times \sum_D \pi_3^0(C, D) \]

Message Propagation: BP

- Lauritzen-Spiegelhalter algorithm
- Two kinds of objects
  - Initial potentials not kept
- Improved “stability” of asynchronous algorithm (repeated messages cancel out)
- Distribution representation – clique tree invariant

\[ \pi_T = \prod_{C_i \in T} \pi_i(C_i) \prod_{(C_i, C_j) \in T} \mu_{ij}(S_{ij}) = P_F(X) \]

11
Multiple queries

- Much caching possible over a clique tree
- Example: compute $P(X,Y)$, for each $X, Y \in \mathcal{X}$
- Dynamic programming
  - Base case, $X$ and $Y$ are in neighbor cliques
    \[
    P(C_i | C_j) = \frac{\pi_i(C_i)}{\mu_y(C_i \cap C_j)} \quad P(C_i) \propto \pi_i
    \]
  - Take advantage of conditional independence:
    \[
    \exists l : C_i \perp C_j | C_l \quad P(C_i, C_j) = \sum_{C_j - C_l} P(C_i, C_j) P(C_i | C_j)
    \]

Incremental updates

- Fully-informed: all neighbors have sent their messages
- Calibrated -- messages and cliques agree on marginals:
  \[
  \sum_{C_i - s_y} \pi_i = \sum_{C_j - s_y} \pi_j = \mu_y
  \]
  - fixed point of MP
- Evidence available in pieces
  - Re-running inference inefficient
- Express evidence in indicator vector
  - multiply into some clique $C_i$
  - run one pass away from $C_i$ to inform the rest
  - works for soft evidence as well
Out-of-clique queries

- I want $P(B, D)$, no clique with both $B$ and $D$!
  - Build a new clique tree – expensive, or
  - Do variable elimination over calibrated tree

$$P(B, D) = \sum_c P(B, C, D)$$

$$= \sum_c \frac{\pi_2(B, C)\pi_3(C, D)}{\mu_{23}(C)}$$

This is back to VE, we save if variables of interest are close in the clique tree.

Defining clique trees

- VE defines cliques
  - Each factor is subset of a clique of
  - Every max clique in is a factor
  - Each clique in is a subclique in
  - Each clique in is a clique in
  - Non-maximal cliques can be eliminated

- Chordal graphs
  - Maximal cliques of any c.g. that is a superset of can be arranged into a clique tree for triangulation
Clique trees generated by VE

Graph induced by ordering: C,D,I,H,G,S,L

VE constructing a clique tree
VE constructing a clique tree

C,D → G,I,D

D → I

H → J
VE constructing a clique tree

C,D → G,I,D → G,S,I

VE constructing a clique tree


H, J, L, G, S, I, D

I

G

S

L

J

H
VE constructing a clique tree


VE constructing a clique tree


H,G,J → J,S,L → J, S, L

J
Summary

- Clique trees
  - factors assigned to cliques many-to-one
  - running intersection
- Message passing on clique trees
  - Variable Elimination
  - Belief propagation
  - different views, algebraically the same
- VE defines cliques
- Time and space tradeoff spectrum
Thank you

- Questions solicited