Exact Inference Algorithms:
Conditioning, Clique Trees

Lecture 7 – Apr 18, 2011
CSE 515, Statistical Methods, Spring 2011

Instructor: Su-In Lee
University of Washington, Seattle

Announcement

- Problem Set #2 is ready.
  - Check course website or pick it up.
  - 7 Questions. Hard. Please start working on it today.
  - Discussion OK! Check collaboration policy.
Variable Elimination Algorithm

- **Goal:** \( P(J) \rightarrow \) query variable(s) can be anything

\[
P(J) = \sum_{L,H,G,J,D,C} P(C, D, I, H, G, S, L)
\]

- Eliminate ordering: C, D, I, H, G, S, L

- Compute:

\[
P(J) = \sum_{L,H,G,J,D,C} \phi_p(J,L,S) \phi_p(L,G) \phi_p(S,I) \phi_p(G,J,D) \phi_p(H,G,J) \phi_p(I) \phi_p(C,D) \phi_p(C)
\]

- Computational complexity:

\[
O(n \max_i |\text{Val}(X_i)|), \text{ where } n \text{ is the number of variables}
\]

Part I

**EXACT INFERENCE:**

**CONDITIONING**
Inference By Conditioning

- **Goal:** compute \( P(J) \)
- **General idea:**
  - Enumerate the possible values \( i \) of a variable \( I \)
  - Apply Variable Elimination in a simplified network \( P(J, I = i) \)
  - Aggregate the results \( P(J) = \sum_{i \in \text{values}(I)} P(J, I = i) \)

**Cutset Conditioning**

- Select a subset of nodes \( X = U \)
- \( X \) is a **cutset** in \( G \) if \( G_{X \setminus X} \) is a polytree (no loop)
**Cutset Conditioning**

- Select a subset of nodes $X \subseteq U$
- $X$ is a cutset in $G$ if $G_{X=x}$ is a polytree

- Define the conditional Bayesian network $G_{X=x}$
  - $G_{X=x}$ has the same variables as $G$
  - $G_{X=x}$ has the same structure as $G$ except that all outgoing edges of nodes in $X$ are deleted, and CPDs of nodes in which edges were deleted are updated to
    
    $$P_{G_{X=x}}(Y | Pa(Y), X = x) = P_G(Y | Pa(Y), X = x)$$

- Compute original $P(Y)$ query by
  - Exponential in cutset
    
    $$P_c(Y) = \sum_{x \in Pa(X)} P_{G_{X=x}}(X = x, Y)$$

**Computational Complexity**

- Variable elimination
  
  $$P(J) = \sum_C \sum_D \sum_I \sum_S \sum_G \sum_L \sum_H P(C, D, I, S, G, L, H, J) \quad (*)$$
  
  $$= \sum_L \sum_I \sum_P L | S \sum_P L | G \sum_P H | G, J \sum_P I \sum_P S | I \sum_P G | D, J \sum_P D | C$$

- Conditioning ($U=\emptyset$)
  
  - Reordering the expression (*) slightly, we have that:
    
    $$P(J) \subseteq \sum_C \sum_D \sum_I \sum_S \sum_G \sum_L \sum_H P(C, D, I, S, G = g, L, H, J)$$

  - In general, both algorithms are performing the same set of basic operations (sums and products).

- Any advantages?
  - Memory gain
  - Forms the basis for a useful approximate inference algorithms (later)
Inference with Clique Trees

- Exploits factorization of the distribution for efficient inference, similar to variable elimination
- Uses global data structures (cluster graphs)
- Deals with a distribution given by (possibly un-normalized) measure

\[ P_F(U) = \prod_{\phi \in F} \phi \]

- For Bayesian networks, factors are CPDs
- For Markov networks, factors are clique potentials
Variable Elimination & Clique Trees

- **Variable elimination**
  - Each step creates a factor $\pi_i$ through multiplication
  - A variable is then eliminated in $\pi_i$ to generate new factor $\tau_j$
  - Process repeated until product contains only query variables

\[
P(J) = \sum_P P(J | L, S) \sum_P (L | G) \sum_P (H | G, J) \sum_P (I | J) \sum_P (G | D, I) \sum_P (C | P(D | C)
\]

- **Clique tree inference**
  - Another view of the above computation
  - **General idea**: $\pi_i$ is a computational data structure which takes "messages" $\tau_i$ generated by other factors $\pi_i$ and generates a message $\tau_j$ which is used by another factor $\pi_k$

Cluster Graph

- Data structure providing flowchart of the factor manipulation process

- A **cluster graph** $K$ for factors $F$ is an undirected graph
  - Nodes are associated with a subset of variables $C_i \subseteq U$
  - The graph is **family preserving**, each factor $\phi \in F$ is associated with one node $C_i$ such that $\text{Scope}([\phi]) = C_i$
  - Each edge $C_i - C_j$ is associated with a **sepset** $S_{ij} = C_i \cap C_j$

- **Key**: variable elimination defines a cluster graph
  - Cluster $C_i$ for each factor $\pi_i$ used in the computation
  - Draw edge $C_i - C_j$ if the factor generated from $\pi_i$ is used in the computation of $\pi_j$
Simple Example

Variable elimination

\[ P(X_1) = \sum_{X_1} \sum_{X_2} P(X_1, X_2, X_3) \]
\[ = \sum_{X_1} \sum_{X_2} P(X_2) P(X_1 | X_2) P(X_3 | X_2) \]
\[ = \sum_{X_1} P(X_1 | X_2) \sum_{X_2} P(X_2) P(X_3 | X_2) \]
\[ = \tau_2(X_2) \]

Cluster graph

- Cluster \( C_1 \) for each factor \( n_i \) used in the computation
- Draw edge \( C_i - C_j \) if the factor generated from \( n_i \) is used in the computation of \( n_j \)

A More Complex Example

- Goal: \( P(J) \), Eliminate: \( C,D,I,H,G,S,L \)

Cluster graph

- Each edge \( C_i - C_j \) is associated with a subset \( S_{i,j} = \{ X_i, X_j \} \)
- Node are associated with a subset \( C_i \)
- The graph is family preserving, each factor is associated with one node, each that Scored \( 1 \)
- Each edge \( C_i - C_j \) is associated with a subset \( S_{i,j} = \{ X_i, X_j \} \)

Key: variable elimination defines a cluster graph

- Cluster \( C_i \) for each factor \( n_i \) used in the computation
- Draw edge \( C_i - C_j \) if the factor generated from \( n_i \) is used in the computation of \( n_j \)
Properties of Cluster Graphs

- Cluster graphs are trees
  - In VE, each intermediate factor $\pi_i$ is used only once
  - Hence, each cluster “passes” an edge (message $\tau_i$) to exactly one other cluster
- Cluster graphs obey the running intersection property
  - If $X \in C_i$ and $X \in C_j$, then $X$ is in each cluster in the (unique) path between $C_i$ and $C_j$

Cluster graphs obey the running intersection property

Running Intersection Property

- **Theorem:** If $T$ is a cluster tree induced by VE over factors $F$, then $T$ obeys the running intersection property
- **Proof:**
  - Let $C$ and $C'$ be two clusters that contain $X$
  - Let $C_x$ be the cluster where $X$ is eliminated
  - $X$ must be present on each cluster on $C$ to $C_x$ path
    - Computation at $C_x$ must be after computation at $C$
    - $X$ is in $C$ by assumption and since $X$ is not eliminated in $C$, then $X$ is in the factor generated by $C$
    - By definition, $C$'s neighbor multiplies factor generated by $C$ and thus (multiplies $X$ and) has $X$ in its scope
    - By induction for all other nodes on the path
  - $X$ appears in all clusters between $C$ and $C_x$
Clique Tree

- A cluster graph over factors F that satisfies the running intersection property is called a *clique tree*.
- Clusters \( C_i \) in a clique tree are also called *cliques*.
- We saw, variable elimination \( \rightarrow \) clique tree
- Now we will see *clique tree* \( \rightarrow \) variable elimination

- Clique tree advantage: data structure for caching computations allowing multiple VE runs to be performed more efficiently than separate VE runs.

We begin with an example and then describe the general algorithm ...
Clique Tree Inference

- **Goal:** Compute $P(J)$

Verify:
- Tree and family preserving
- Running intersection property

**Diagram:**

1. Eliminate $C$, sending $\delta_1(D)$ to $C_2$
2. Eliminate $D$, sending $\delta_2(G,I)$ to $C_3$
3. Eliminate $I$, sending $\delta_3(G,S)$ to $C_5$
4. Eliminate $H$, sending $\delta_4(G,J)$ to $C_5$
5. Obtain $P(J)$ by summing out $G,S,L$ from $\pi(C_5)\delta_3\delta_4$.

**Equations:**

- $\pi(C)P(D|C)P(G|I,D)P(I)P(S|I)P(L|G)P(J|L,S)P(H|G,J)$
Clique Tree Inference

- **Goal:** Compute $P(J)$ – define root clique $C_r = C_4$
  - Set initial factors (CPDs) at each cluster as products $\pi_0$.
  - $C_1$: Eliminate $C$, sending a message $\delta_{C\rightarrow D}(D)$ to $C_2$.
  - $C_2$: Eliminate $D$, sending $\delta_{D\rightarrow I}(G,I)$ to $C_3$.
  - $C_3$: Eliminate $I$, sending $\delta_{I\rightarrow S}(G,S)$ to $C_5$.
  - $C_5$: Eliminate $S,L$, sending $\delta_{S\rightarrow J}(G,J)$ to $C_4$.
  - $C_4$: Obtain $P(J)$ by summing out $H,G$ from $\pi_0(C_4)\delta_{S\rightarrow J}(G,J)$.

Clique Tree Inference

- **C5 as the root**
  - $\delta_{C\rightarrow D}(D)$: $\sum \pi_0(C_1) \delta_{C\rightarrow D}(D)$.
  - $\delta_{D\rightarrow I}(G,I)$: $\sum \pi_0(C_2) \times \delta_{D\rightarrow I}(G,I)$.
  - $\delta_{I\rightarrow S}(G,S)$: $\sum \pi_0(C_3) \times \delta_{I\rightarrow S}(G,S)$.
  - $\delta_{S\rightarrow J}(G,J)$: $\sum \pi_0(C_4) \times \delta_{S\rightarrow J}(G,J)$.

- **C4 as the root**
  - $\delta_{C\rightarrow D}(D)$: $\sum \pi_0(C_1) \delta_{C\rightarrow D}(D)$.
  - $\delta_{D\rightarrow I}(G,I)$: $\sum \pi_0(C_2) \times \delta_{D\rightarrow I}(G,I)$.
  - $\delta_{I\rightarrow S}(G,S)$: $\sum \pi_0(C_3) \times \delta_{I\rightarrow S}(G,S)$.
  - $\delta_{S\rightarrow J}(G,J)$: $\sum \pi_0(C_4) \times \delta_{S\rightarrow J}(G,J)$.
Legal ordering

- The only constraint is that a clique gets all of its incoming messages from its downstream neighbors before it sends its outgoing message toward its upstream neighbor.
  - We say that \( C_i \) is ready to transmit to a neighbor \( C_j \) when \( C_i \) has messages from all of its neighbors except for \( C_j \).

- Example
  - Root \( C_6 \)
    - Legal ordering I: 1, 2, 3, 4, 5, 6
    - Legal ordering II: 2, 5, 1, 3, 4, 6
    - Illegal ordering: 3, 4, 1, 2, 5, 6

Here is the general algorithm ...
Clique Tree Message Passing

1. Let T be a clique tree and C₁,...,Cₖ its cliques
   - Multiply factors (CPDs) assigned to each clique, resulting in initial potentials as each factor is assigned to some clique α(φ):
     \[
     π_i^α(C_j) = \prod_{φ∈C_j} φ \quad \text{and} \quad \prod_{φ∈C_j} φ = \prod_{φ∈C_j} π_i^α(C_j)
     \]
   - Define Cᵣ as the root clique
     - If our goal is to compute P(J), any clique containing J can be Cᵣ
     - Use the clique-tree data structure to pass messages between neighboring cliques, sending all messages toward Cᵣ
     - Start from tree leaves and move inward
     - Let pᵣ(i) be the upstream neighbor of i (on the path to Cᵣ)
     - Each Cᵢ performs a computation that sends message δᵢ to Cᵢ₊₁
     - Multiply all incoming messages from downstream neighbors with the initial clique potential resulting in a factor whose scope is the clique.
     - Sum out all variables except those in the separator Cᵢ⁻¹_Cᵢ₊₁

2. When the root clique Cᵣ has received all messages, it multiplies them with its own initial potential, resulting in a factor called the belief
   - Summing out all variables except those in the separator
   - Resulting belief is P(J) and is computed as:
     \[
     P(J) = \sum_{π_j} \prod_{φ∈C_j} φ
     \]
Clique Tree Inference Correctness

- **Theorem**
  - Let \( C_r \) be the root clique in a clique tree.
  - If \( \pi_r \) is computed as above, then
    \[
    \pi_r[C_r] = \sum_{U \in \pi_r} P_r(U)
    \]

- **Algorithm applies to Bayesian and Markov networks**
  - For Bayesian network \( G \), if \( F \) consists of the CPDs reduced with some evidence \( e \) then \( \pi_r[C_r] = P_G(C_r, e) \)
    - Probability obtained by normalizing the factor over \( C_r \) to sum to 1.
  - For Markov network \( H \), if \( F \) consists of a set of clique potentials, then \( \pi_r[C_r] = P_H(C_r) \)
    - Probability obtained by normalizing the factor over \( C_r \) to sum to 1.
    - Partition function obtained by summing up all entries in \( \pi_r[C_r] \).

Clique Tree Calibration

- **Assume we want to compute marginal distributions over \( n \) variables:**
  \( P(X_1), \ldots, P(X_n) \)
  - With variable elimination, we perform \( n \) separate VE runs.
  - With clique trees, we can do this much more efficiently.
    - Idea 1: since marginal over a variable can be computed from any root clique that includes it, perform \( k \) clique tree runs (\( k = \# \text{ cliques} \)).
    - Idea 2: Can do much better! How?
Clique Tree Calibration

- Observation: a message from $C_i$ to $C_j$ is unique
  - Consider two neighboring cliques $C_i$ and $C_j$
  - If root $C_j$ is on $C_i$ side, $C_i$ sends $C_j$ a message
  - Message does not depend on specific $C_i$ (we only need $C_i$ to be on the $C_j$ side for $C_i$ to send a message to $C_j$)
    - Message from $C_i$ to $C_j$ will always be the same, regardless of what the query variables are.

- Each edge has two messages associated with it
  - One message for each direction of the edge
  - There are only $2(k-1)$ messages to compute
  - Can then readily compute the marginal probability over each variable

- Compute $2(k-1)$ messages by
  - Pick any node as the root
  - Upward pass: send messages to the root
    - Terminate when root received all messages
  - Downward pass: send messages to root children
    - Terminate when all leaves received messages
Clique Tree Calibration: Example

- Root: $C_5$ (first downward pass)

\[
P(C) = \Pi \delta_{j-2}\[C_j] = \Pi \delta_{j-2}[C_j]
\]

\[
P(D|C) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i]
\]

\[
P(G|I,D) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(I) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(S|I) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(L|G) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(J|L,S) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(H|G,J) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

Clique Tree Calibration: Example

- Root: $C_5$ (second downward pass)

\[
P(C) = \Pi \delta_{j-2}\[C_j] = \Pi \delta_{j-2}[C_j]
\]

\[
P(D|C) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i]
\]

\[
P(G|I,D) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(I) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(S|I) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(L|G) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(J|L,S) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]

\[
P(H|G,J) = \sum \delta_{j-2}[C_j] \times \delta_{i-2}[C_i] \times \delta_{k-2}[C_k]
\]
Clique Tree Calibration

- Theorem
  - "Belief" \( \pi_i \) is computed for each clique \( i \) as above:

\[
\pi_i(C_i) = \frac{\pi_i(C_i)^{\text{neighbor of } i}}{\sum_{\text{neighbor of } i} \pi_i(U)}
\]

- Important: avoid double-counting!
  - Each node \( i \) computes the message to its neighbor \( j \) using its initial potentials \( \pi_0^i \) and not its updated potential ("belief") \( \pi_i \), since \( \pi_i \) integrates information from \( C_j \) which will be counted twice.

\[
\sum_{\pi_i(C_i)^{\text{neighbor of } i}} \prod_{\text{neighbor of } i} \delta_{ij}
\]

Acknowledgement

- These lecture notes were generated based on the slides from Prof Eran Segal.