Readings: K&F 9.5, 10.1, 10.2, 10.3 (10.4)

Exact Inference Algorithms: Conditioning, Clique Trees

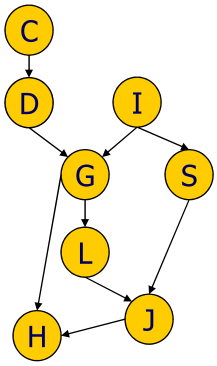
Lecture 7 – Apr 18, 2011
CSE 515, Statistical Methods, Spring 2011

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) → query variable(s) can be anything

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

$$= \sum_{L,S,G,H,I,D,C} \phi_J(J,L,S)\phi_L(L,G)\phi_S(S,I)\phi_G(G,I,D) \phi_H(H,G,J)\phi_I(I)\phi_D(C,D)\phi_C(C)$$

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

$$P(J) = \sum_{L,S}\phi_J(J,L,S)\sum_{G}\phi_L(L,G)\sum_{H}\phi_H(H,G,J)\sum_{I}\phi_I(I)\phi_S(S,I)\sum_{D}\phi_G(G,I,D)\sum_{C}\phi_D(C,D)\phi_C(C)$$

- Computational complexity:
  - $O(n \max_i |Val(\mathbf{X}_i)|)$, where n is the number of variables

3

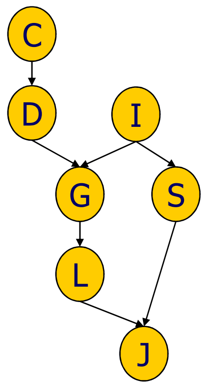
Part I

EXACT INFERENCE: CONDITIONING

4

Inference By Conditioning

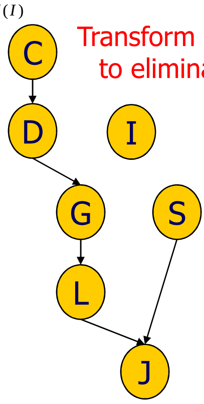
- Goal: compute $P(J)$
- General idea
  - Enumerate the possible values i of a variable I
  - Apply Variable Elimination in a simplified network $P(J, I = i)$
  - Aggregate the results $P(J) = \sum_{i \in Val(I)} P(J, I = i)$



(G⊥S | I)

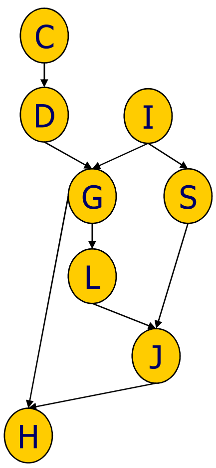
Observe I=i

Transform CPDs of G and S to eliminate I as parent

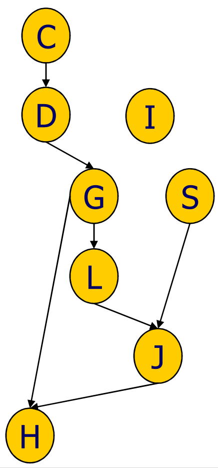


Cutset Conditioning

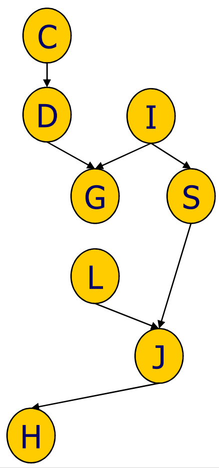
- Select a subset of nodes $\mathbf{X} \subset \mathbf{U}$
- $\mathbf{X}$ is a cutset in G if $G_{\mathbf{X}=\mathbf{x}}$ is a polytree (no loop)



I is not a cutset



G is a cutset



Cutset Conditioning

- Select a subset of nodes $\mathbf{X} \subset \mathbf{U}$
- $\mathbf{X}$ is a cutset in G if $G_{\mathbf{X=x}}$ is a polytree
- Define the conditional Bayesian network $G_{\mathbf{X=x}}$
  - $G_{\mathbf{X=x}}$ has the same variables as G
  - $G_{\mathbf{X=x}}$ has the same structure as G except that all outgoing edges of nodes in $\mathbf{X}$ are deleted, and CPDs of nodes in which edges were deleted are updated to

$$P_{G_{X=x}}(Y \mid Pa(Y) - \boldsymbol{X}) = P_G(Y \mid Pa(Y), \boldsymbol{X} = \boldsymbol{x})$$

- Compute original P(**Y**) query by
  - Exponential in cutset $\quad P_G(\boldsymbol{Y}) = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} P_{G_{X=x}}(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y})$

Computational Complexity

- Variable elimination

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

$$= \sum_L \sum_S P(J \mid L, S) \sum_G P(L \mid G) \sum_H P(H \mid G, J) \sum_I P(I) P(S \mid I) \sum_D P(G \mid D, I) \sum_C P(C) P(D \mid C)$$

- Conditioning (**U**=**u**)
  - Reordering the expression (*) slightly, we have that:

$$P(J) = \sum_g \left[\sum_C \sum_D \sum_I \sum_S \sum_L \sum_H P(C, D, I, S, G = g, L, H, J) \right]$$

  - 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)

Part II

EXACT INFERENCE: CLIQUE TREES

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

$$P_F(\boldsymbol{U}) = \prod_{\phi' \in F} \phi'$$

  - For Bayesian networks, factors are CPDs
  - For Markov networks, factors are clique potentials

Variable Elimination & Clique Trees

- Variable elimination
  - Each step creates a factor π_i through multiplication
  - A variable is then eliminated in π_i to generate new factor τ_i
  - Process repeated until product contains only query variables

$$P(J) = \sum_L \sum_S P(J \mid L,S) \sum_G P(L \mid G) \sum_H P(H \mid G,J) \sum_I P(I)P(S \mid I) \sum_D P(G \mid D,I) \sum_C P(C)P(D \mid C)$$

- Clique tree inference
  - Another view of the above computation
  - General idea: π_j is a computational data structure which takes "messages" τ_i generated by other factors π_i and generates a message τ_j which is used by another factor π_k

Cluster Graph

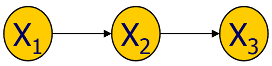
- Data structure providing flowchart of the factor manipulation process

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

- Key: variable elimination defines a cluster graph
  - Cluster $\mathbf{C_i}$ for each factor π_i used in the computation
  - Draw edge $\mathbf{C_i}$–$\mathbf{C_j}$ if the factor generated from π_i is used in the computation of π_j

Simple Example

- Key: variable elimination defines a cluster graph
  - Cluster $\mathbf{C_i}$ for each factor π_i used in the computation
  - Draw edge $\mathbf{C_i}$–$\mathbf{C_j}$ if the factor generated from π_i is used in the computation of π_j



Variable elimination

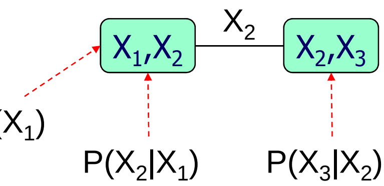
$$
\begin{aligned}
P(X_3) &= \sum_{X_1}\sum_{X_2} P(X_1, X_2, X_3) \\
&= \sum_{X_1}\sum_{X_2} P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_2) \\
&= \sum_{X_2} P(X_3 \mid X_2)\sum_{X_1} P(X_1)P(X_2 \mid X_1) \\
&= \sum_{X_2} P(X_3 \mid X_2)\tau_1(X_2) \\
&= \tau_2(X_3)
\end{aligned}
$$

Cluster graph

$C_1 = \{X_1,X_2\}$

$C_2 = \{X_2,X_3\}$

$S_{1,2} = \{X_2\}$



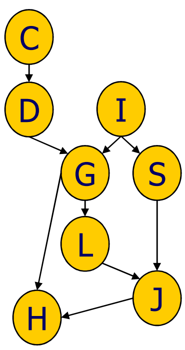
- A cluster graph K for factors F is an undirected graph
  - Nodes are associated with a subset of variables $\mathbf{C_i} \subseteq \mathbf{U}$
  - The graph is family preserving: each factor $\phi \in F$ is associated with one node $\mathbf{C_i}$ such that Scope$[\phi] \subseteq \mathbf{C_i}$
  - Each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

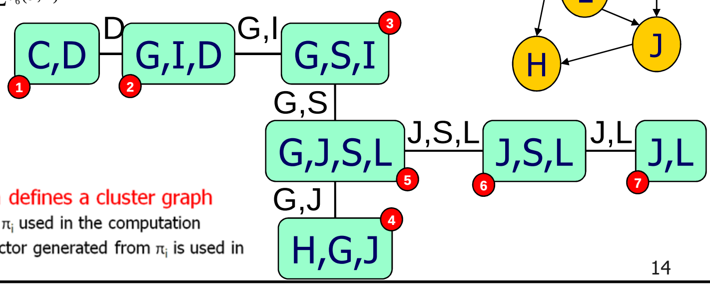
family preservation?

A More Complex Example

$$P(J) = \sum_{L,S}\phi_J(J,L,S)\sum_G \phi_L(L,G)\sum_H \phi_H(H,G,J)\sum_I \phi_I(I)\phi_S(S,I)\sum_D \phi_G(G,I,D)\sum_C \phi_D(C,D)\phi_C(C)$$

- Goal: P(J), Eliminate: C,D,I,H,G,S,L
  - C: $\tau_1(D) = \sum_C \phi_C(C)\phi_D(C,D)$
  - D: $\tau_2(G,I) = \sum_D \phi_G(G,I,D)\tau_1(D)$
  - I: $\tau_3(G,S) = \sum_I \phi_I(I)\phi_S(S,I)\tau_2(G,I)$
  - H: $\tau_4(G,J) = \sum_H \tau_H(H,G,J)$
  - G: $\tau_5(J,L,S) = \sum_G \phi_L(L,G)\tau_3(G,S)\tau_4(G,J)$
  - S: $\tau_6(J,L) = \sum_S \phi(J,L,S)\tau_5(J,L,S)$
  - L: $\tau_7(J) = \sum_L \tau_6(J,L)$

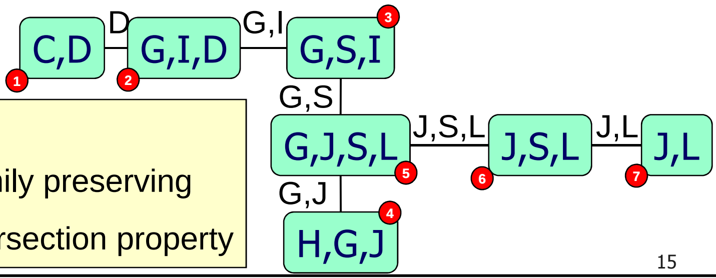




- Key: variable elimination defines a cluster graph
  - Cluster $\mathbf{C_i}$ for each factor π_i used in the computation
  - Draw edge $\mathbf{C_i}$–$\mathbf{C_j}$ if the factor generated from π_i is used in the computation of π_j

Properties of Cluster Graphs

- Cluster graphs are trees
  - In VE, each intermediate factor π_i is used only once
  - Hence, each cluster "passes" an edge (message τ_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

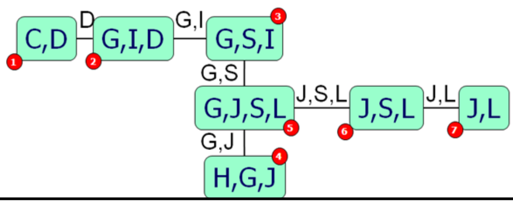


Verify:

- Tree and family preserving
- 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 → clique tree
- Now we will see clique tree → variable elimination
- Clique tree advantage: data structure for caching computations allowing multiple VE runs to be performed more efficiently than separate VE runs

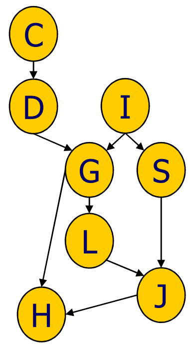
17

We begin with an example and then describe the general algorithm …

18

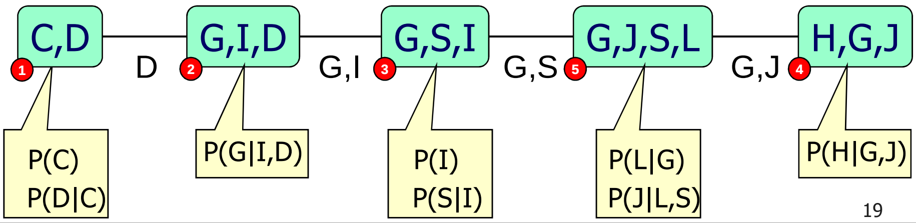
Clique Tree Inference

- Goal: Compute P(J)



Verify:

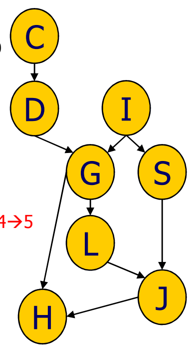
- Tree and family preserving
- Running intersection property

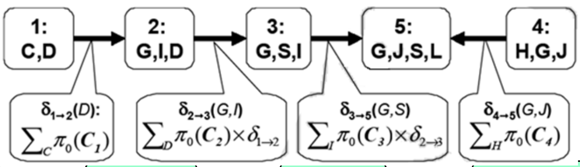


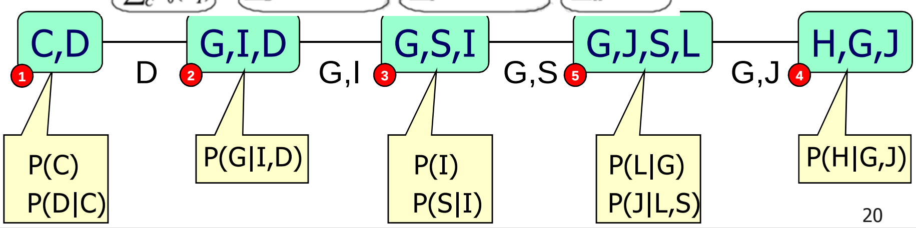
19

Clique Tree Inference

- Goal: Compute P(J) – define root clique $C_r=C_5$
  - Set initial factors (CPDs) at each cluster as products π_i^0
  - C_1: Eliminate C, sending a message $\delta_{1\to2}$(D) to C_2
  - C_2: Eliminate D, sending $\delta_{2\to3}$(G,I) to C_3
  - C_3: Eliminate I, sending $\delta_{3\to5}$(G,S) to C_5
  - C_4: Eliminate H, sending $\delta_{4\to5}$(G,J) to C_5
  - C_5: Obtain P(J) by summing out G,S,L from $\pi_0(C_5)\delta_{3\to5}\delta_{4\to5}$



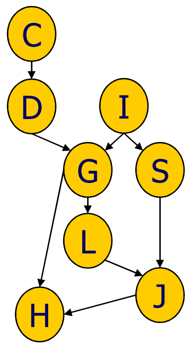


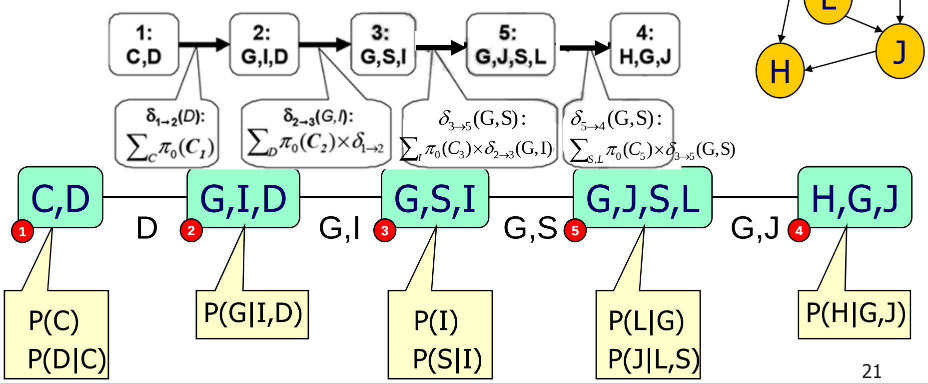


20

Clique Tree Inference

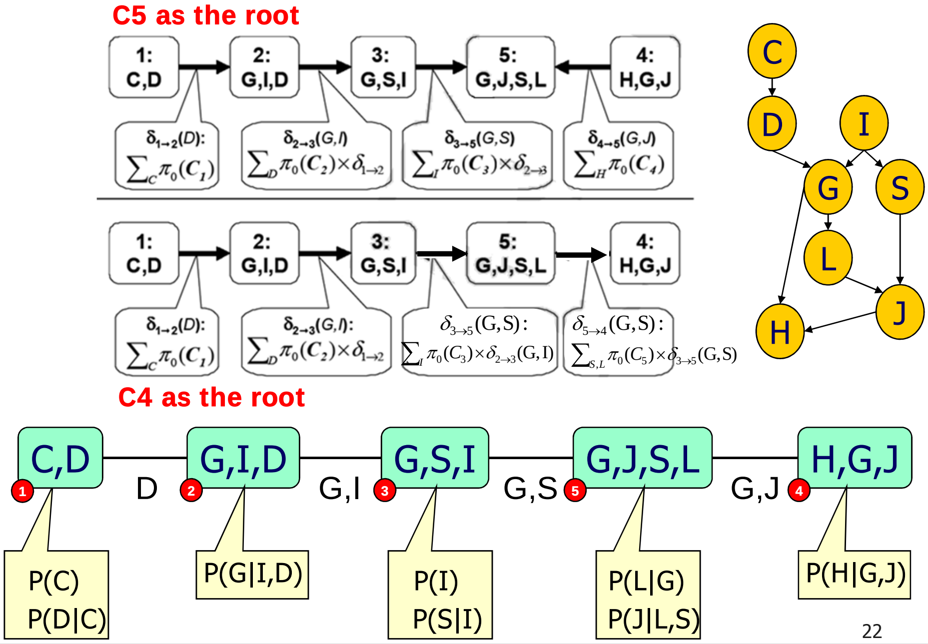
- Goal: Compute P(J) – define root clique $C_r = C_4$
  - Set initial factors (CPDs) at each cluster as products π_i^0
  - C_1: Eliminate C, sending a message $\delta_{1\to2}(D)$ to C_2
  - C_2: Eliminate D, sending $\delta_{2\to3}(G,I)$ to C_3
  - C_3: Eliminate I, sending $\delta_{3\to5}(G,S)$ to C_5
  - C_5: Eliminate S,L, sending $\delta_{5\to4}(G,J)$ to C_4
  - C_4: Obtain P(J) by summing out H,G from $\pi_0(C_4)\delta_{5\to4}$





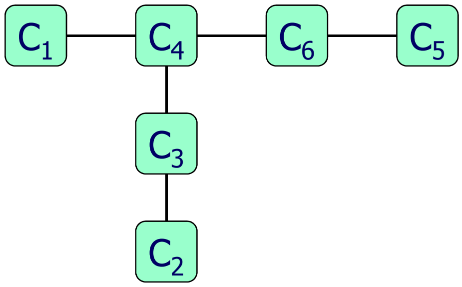
Clique Tree Inference

C5 as the root



Legal ordering

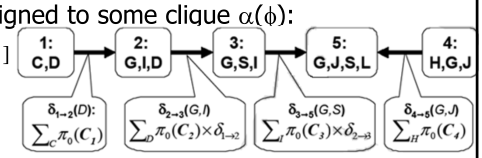
- The only constraint is that a clique gets all of its incoming messages from its downstream neighbors before it sends its outgoing message toward its upstream neighbor.
  - We say that C_i is ready to transmit to a neighbor C_j when C_i has messages from all of its neighbors except for C_j.
- Example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

- Let T be a clique tree and $C_1,\ldots C_k$ its cliques
  - Multiply factors (CPDs) assigned to each clique, resulting in initial potentials as each factor is assigned to some clique $\alpha(\phi)$:
    $\pi_j^0[C_j] = \prod_{\phi:\alpha(\phi)=j} \phi$ and $\prod_{\phi} \phi = \prod_{j=1}^{k} \pi_j^0[C_j]$

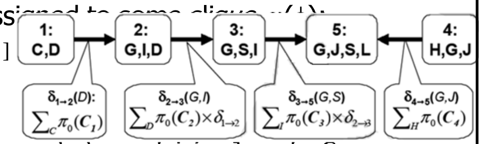


  - Define C_r as the root clique
    - If our goal is to compute P(J), any clique containing J can be C_r
  - Use the clique-tree data structure to pass messages between neighboring cliques, sending all messages toward C_r
    - Start from tree leaves and move inward
  - Let $p_r(i)$ be the upstream neighbor of i (on the path to C_r)
  - Each C_i performs a computation that sends message δ_i to $C_{p_r(i)}$
    - 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 sepset C_i—$C_{p_r(i)}$

$$\delta_{i\to j}(S_{i,j}) = \sum_{C_i - S_{i,j}} \pi_i^0[C_i] \prod_{k\in\{\text{neighbors of i except for j}\}} \delta_{k\to i}$$

Clique Tree Message Passing

- Let T be a clique tree and $C_1,\ldots C_k$ its cliques
  - Multiply factors (CPDs) assigned to each clique, resulting in initial potentials as each factor is ass
    $\pi_j^0[C_j] = \prod_{\phi:\alpha(\phi)=j} \phi$ and $\prod_{\phi} \phi = \prod_{j=1}^{k} \pi_j^0[C_j]$



  - Define C_r as the root clique
    - If our goal is to compute P(J), any cluster containing J can be C_r
  - Use the clique-tree data structure to pass messages between neighboring cliques, sending all messages toward C_r
    - Start from tree leaves and move inward
  - Let $p_r(i)$ be the upstream neighbor of i (on the path to C_r)
  - Each C_i performs a computation that sends message δ_i to $C_{p_r(i)}$

$$\delta_{i\to j}(S_{i,j}) = \sum_{C_i - S_{i,j}} \pi_i^0[C_i] \prod_{k\in\{\text{neighbors of i except for j}\}} \delta_{k\to i}$$

  - When the root clique C_r has received all messages, it multiplies them with its own initial potential, resulting in a factor called the belief
    - $\pi_r[C_r] = \pi_r^0[C_r] \prod_{i\in\{\text{neighbors of r}\}} \delta_{i\to r}$ representing $P(C_r) = \sum_{U-C_r} \prod_{\phi} \phi$

Clique Tree Inference Correctness

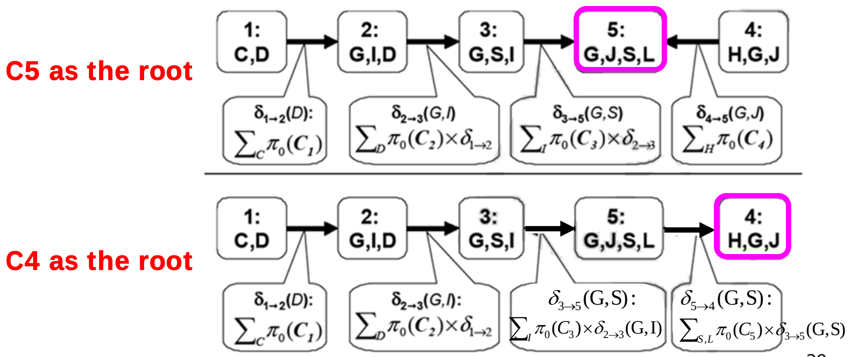
- Theorem
  - Let C_r be the root clique in a clique tree
  - If π_r is computed as above, then $\pi_r[C_r] = \sum_{U-C_r} P_F(\mathbf{U})$
- Algorithm applies to Bayesian and Markov networks
  - For Bayesian network G, if F consists of the CPDs reduced with some evidence **e** then $\pi_r[C_r] = P_G(C_r, \mathbf{e})$
    - 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=# 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_r is on C_j side, C_i sends C_j a message
  - Message does not depend on specific C_r (we only need C_r 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.



Clique Tree Calibration

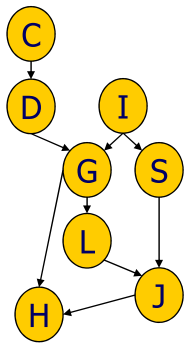
- Observation: a message from C_i to C_j is unique
  - Consider two neighboring cliques C_i and C_j
  - If root C_r is on C_j side, C_i sends C_j a message
  - Message does not depend on specific C_r (we only need C_r to be on the C_j side for C_i to send a message to C_j)
  - → Message from C_i to C_j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

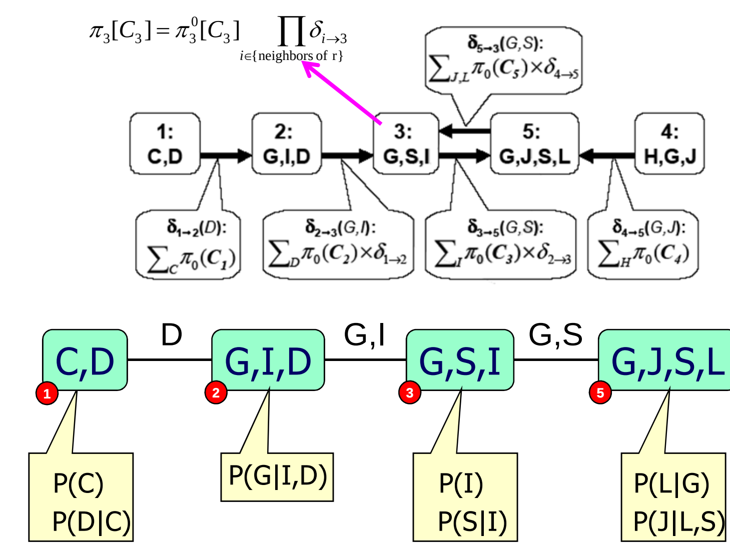
- Each edge has two messages associated with it
  - One message for each direction of the edge
  - There are only 2(k-1) messages to compute
  - Can then readily compute the marginal probability over each variable

- 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)

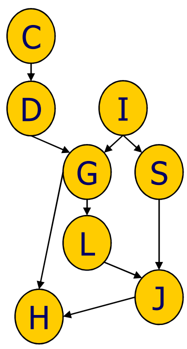


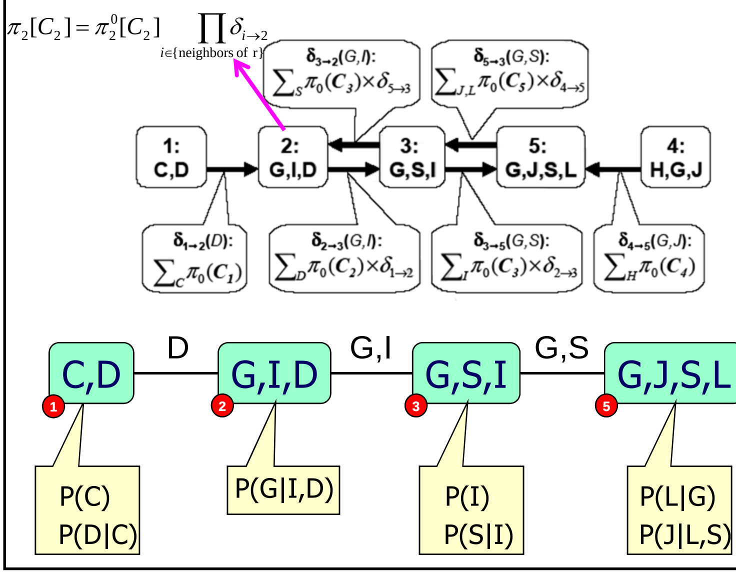


31

Clique Tree Calibration: Example

- Root: C_5 (second downward pass)





32

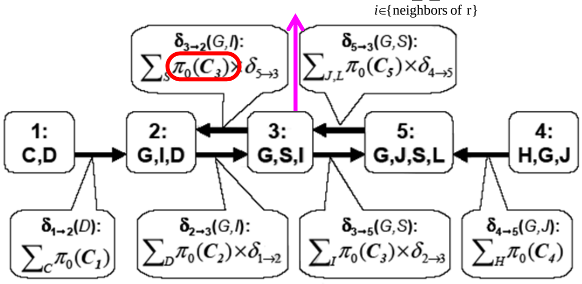
Clique Tree Calibration

- Theorem
  - "Belief" π_i is computed for each clique i as above:

$$\pi_i[C_i] = \pi_i^0[C_i] \prod_{j \in \{\text{neighbors of } i\}} \delta_{j \to i} = \sum_{U - C_i} P_F(\mathbf{U})$$

- Important: avoid double-counting!
  - Each node i computes the message to its neighbor j using its initial potentials π^0_i and not its updated potential ("belief") π_i, since π_i integrates information from C_j which will be counted twice

$$\pi_3[C_3] = \pi_3^0[C_3] \prod_{i \in \{\text{neighbors of } r\}} \delta_{i \to 3}$$



Acknowledgement

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