Stat 521A
Lecture 7

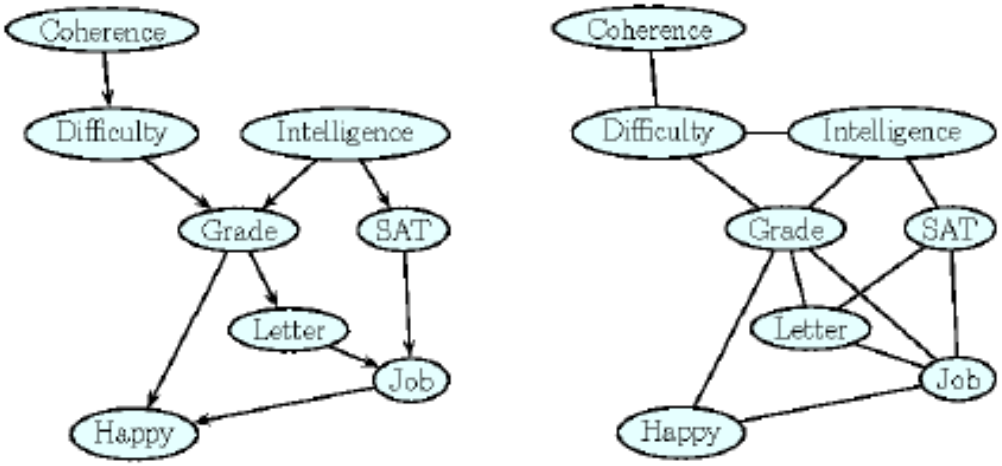
Outline

- Variable elimination (9.2-9.3)
- Complexity of VE (9.4)
- Conditioning (9.5)
- From VE to clique trees (10.1)
- Message passing on clique trees (10.2-10.3)
- Creating clique trees (10.4)

Inference

- Consider the following distribution

$$P(C, D, I, G, S, L, J, H)$$
$$= P(C)P(D|C)P(I)P(G|I, D)P(S|I)P(L|G)P(J|L, S)P(H|G, .$$
$$P(C, D, I, G, S, L, J, H)$$
$$= \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D)\psi_S(S, I)$$
$$\psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J)$$



Brute force enumeration

- Compute marginal probability someone has a job

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

Variable elimination 1

- Push sums inside products (distributive law)

$$
\begin{aligned}
P(J) &= \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C P(C, D, I, G, S, L, J, H) \\
&= \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D)\psi_S(S, I) \\
&\quad \psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J) \\
&= \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I)\psi_I(I) \\
&\quad \sum_D \psi_G(G, I, D) \sum_C \psi_C(C)\psi_D(D, C)
\end{aligned}
$$

VE 2: work right to left

$$
\begin{aligned}
P(J) &= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \psi_G(G,I,D) \underbrace{\sum_C \psi_C(C)\psi_D(D,C)}_{\tau_1(D)} \\
&= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_G(G,I,D)\tau_1(D)}_{\tau_2(G,I)} \\
&= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \underbrace{\sum_I \psi_S(S,I)\psi_I(I)\tau_2(G,I)}_{\tau_3(G,S)} \\
&= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \underbrace{\sum_H \psi_H(H,G,J)}_{\tau_4(G,J)} \tau_3(G,S) \\
&= \sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_G \psi_L(L,G)\tau_4(G,J)\tau_3(G,S)}_{\tau_5(J,L,S)} \\
&= \sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_5(J,L,S)}_{\tau_6(J,L)} \\
&= \underbrace{\sum_L \tau_6(J,L)}_{\tau_7(J)}
\end{aligned}
$$

Variable elimination
Bucket elimination
Peeling
Non-serial dynamic programming

Pseudocode

Algorithm 9.1 Sum-Product Variable Elimination algorithm

```
    Procedure Sum-Product-Variable-Elimination (
      Φ,    // Set of factors
      Z,    // Set of variables to be eliminated
      ≺     // Ordering on Z
    )
1     Let Z_1, ..., Z_k be an ordering of Z such that
2       Z_i ≺ Z_j iff i < j
3     for i = 1, ..., k
4       Φ ← Sum-Product-Eliminate-Var(Φ, Z_i)
5     φ* ← ∏_{φ∈Φ} φ
6     return φ*

    Procedure Sum-Product-Eliminate-Var (
      Φ,    // Set of factors
      Z     // Variable to be eliminated
    )
1     Φ' ← {φ ∈ Φ : Z ∈ Scope[φ]}
2     Φ'' ← Φ − Φ'
3     ψ ← ∏_{φ∈Φ'} φ
4     τ ← ∑_Z ψ
5     return Φ'' ∪ {τ}
```

Dealing with evidence

- Conditional prob is ratio of uncond prob

$$P(J|I=1,H=0) = \frac{P(J,I=1,H=0)}{P(I=1,H=0)}$$

- Soft/ virtual evidence: $\phi_i(X_i) = p(y_i|X_i)$

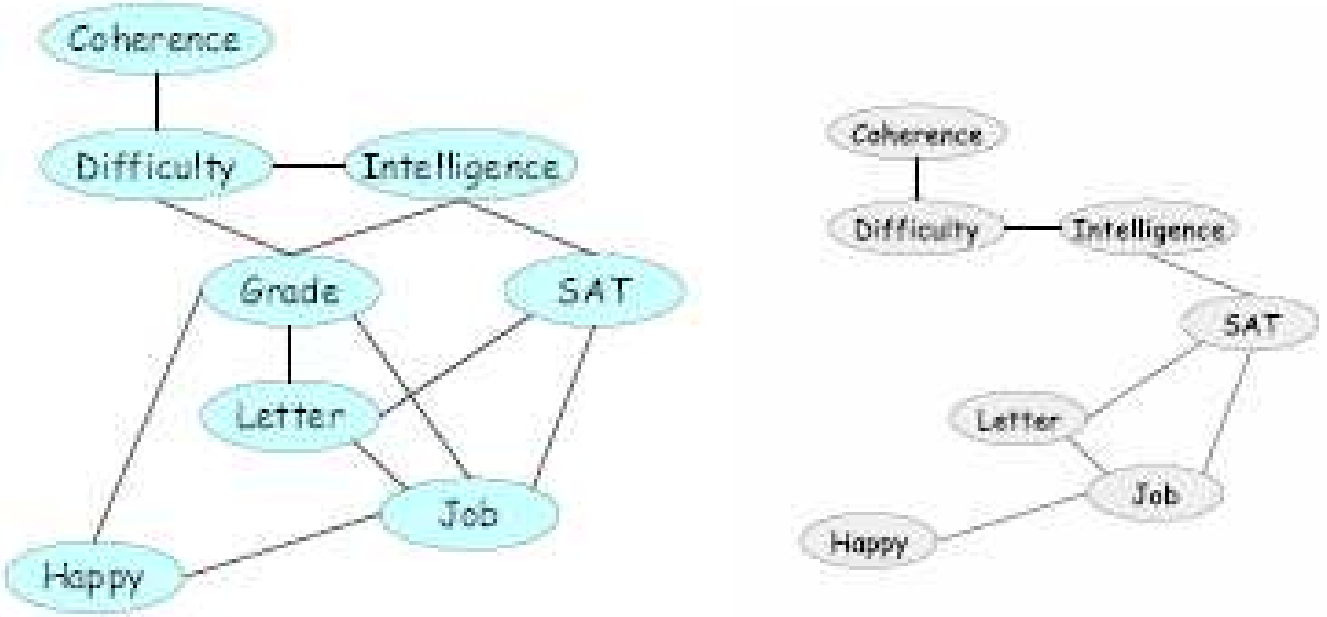
$$P(J,I=1,H=0) =$$
$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J)\phi_{\mathbf{H}}(\mathbf{H}) \sum_I \psi_S(S,I)\psi_I(I)\phi_{\mathbf{I}}(\mathbf{I})$$
$$\sum_D \psi_G(G,I,D) \sum_C \psi_C(C)\psi_D(D,C)$$

- Hard evidence: $\phi_i(X_i) = I(X_i=x_i^*)$

$$P(J,I=1,H=0) =$$
$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G)\psi_H(H=0,G,J)\psi_S(S,I=1)\psi_I(I=1)$$
$$\sum_D \psi_G(G,I,D) \sum_C \psi_C(C)\psi_D(D,C)$$

Reduced graph

- If nodes are instantiated (fully observed), we can remove them and their edges and absorb their effect by updating all the other factors that reference them
- Eg if G is observed



VE with hard evidence

Procedure Cond Prob VE (
  $\mathcal{K}$, // A network over $\mathcal{X}$
  $\boldsymbol{Y}$, // Set of query variables
  $\boldsymbol{E} = \boldsymbol{e}$ // Evidence
)

1 $\Phi \leftarrow$ Factors parameterizing $\mathcal{K}$
2 Replace each $\phi \in \Phi$ by $\phi[\boldsymbol{E} = \boldsymbol{e}]$
3 Select an elimination ordering $\prec$
4 $\boldsymbol{Z} \leftarrow = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$
5 $\phi^* \leftarrow$ Sum Product Variable Elimination$(\Phi, \prec, \boldsymbol{Z})$
6 $\alpha \leftarrow \sum_{\boldsymbol{y} \in Val(\boldsymbol{Y})} \phi^*(\boldsymbol{y})$
7 **return** α, ϕ^*

Complexity analysis of VE

- At step i, we multiply all factors involving x_i into a large factor, then sum out x_i to get τ_i.
- Let N_i be number of entries in factor ψ_i.
- The total number of factors is m+n, where m = original number of factors in model ($m \geq n$), and n=num. vars.
- Each factor gets multiplied into something bigger once. Hence #mult is at most

$$(n+m)N_i \leq (n+m)N_{max} = O(mN_{max})$$

- When we sum out a node from a factor, we touch each entry once, so #adds is at most

$$nN_{max}$$

Complexity analysis of VE

- If each variable has v values, and factor ψ_i involves k_i variables, then $N_i \leq v^k$
- So complexity is exponential in the size of the largest factor.

Different elimination ordering

$$
\begin{aligned}
P(J) &= \sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \sum_I \psi_I(I)\psi_S(S,I) \underbrace{\sum_G \psi_G(G,I,D)\psi_L(L,)\psi_H(H,G,J)}_{\tau_1(I,D,L,J,H)} \\
&= \sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_I \psi_I(I)\psi_S(S,I)\tau_1(I,D,L,J,H)}_{\tau_2(D,L,S,J,H)} \\
&= \sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_2(D,L,S,J,H)}_{\tau_3(D,L,J,H)} \\
&= \sum_D \sum_C \psi_D(D,C) \sum_H \underbrace{\sum_L \tau_3(D,L,J,H)}_{\tau_4(D,J,H)} \\
&= \sum_D \sum_C \psi_D(D,C) \underbrace{\sum_H \tau_4(D,J,H)}_{\tau_5(D,J)} \\
&= \sum_D \underbrace{\sum_C \psi_D(D,C)\tau_5(D,J)}_{\tau_6(D,J)} \\
&= \underbrace{\sum_D \tau_6(D,J)}_{\tau_7(J)}
\end{aligned}
$$

Effect of ordering

- A bad ordering can create larger intermediate factors, and therefore is slower

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$. $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$. $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$. $\phi_S(S,I)$. $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	G	$\tau_4(G,J)$. $\tau_3(G,S)$. $\phi_L(L,G)$	G,J,L,S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S)$. $\phi_J(J,L,S)$	J,L,S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J,L	$\tau_7(J)$

4

Table 9.1 A run of variable elimination for the query $P(J)$.

Step	Variable eliminated	Factors used	Variables involved	New factor
1	G	$\phi_G(G,I,D)$. $\phi_L(L,G)$. $\phi_H(H,G,J)$	G,I,D,L,J,H	$\tau_1(I,D,L,J,H)$
2	I	$\phi_I(I)$. $\phi_S(S,I)$. $\tau_1(I,D,L,S,J,H)$	S,I,D,L,J,H	$\tau_2(D,L,S,J,H)$
3	S	$\phi_J(J,L,S)$. $\tau_2(D,L,S,J,H)$	D,L,S,J,H	$\tau_3(D,L,J,H)$
4	L	$\tau_3(D,L,J,H)$	D,L,J,H	$\tau_4(D,J,H)$
5	H	$\tau_4(D,J,H)$	D,J,H	$\tau_5(D,J)$
6	C	$\tau_5(D,J)$. $\phi_D(D,C)$	D,J,C	$\tau_6(D,J)$
7	D	$\tau_6(D,J)$	D,J	$\tau_7(J)$

6
6

Table 9.2 A different run of variable elimination for the query $P(J)$.

Graph theoretic analysis

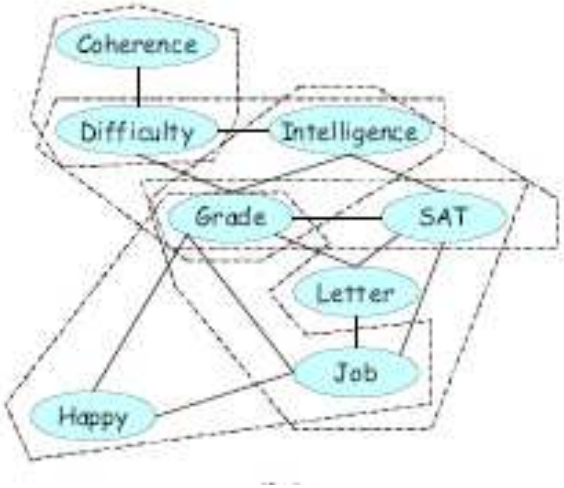
- Every time we eliminate a node, we build a new factor which combines variables that may have previously been in separate factors. Let us add an edge (fill-in edge) between such nodes to create the induced graph

When we eliminate I, we add a fill-in between G and S

$$\tau_3(G,S) = \sum_I \psi_S(S,I)\psi_I(I)\tau_2(G,I)$$

Induced graph

- Def 9.4.3. Let I(G,<) represent the graph induced by applying VE with order < to graph G.
- Thm 9.4.4.Every factor generated by VE is a clique in I(G,<). Also, every maximal clique in I(G,<) corresponds to some intermediate factor.



Variables involved
C, D
G, I, D
G, S, I
H, G, J
G, J, L, S
J, L, S
J, L

$$\{C, D\}, \{D, I, G\}, \{G, L, S, J\}, \{G, J, H\}, \{G, I, S\}$$

Treewidth

- Def 9.4.5. The width of an induced graph is the number of nodes in the largest clique minus 1. The minimal induced width of a graph, aka the treewidth, is defined as

$$W_G = \min_{\prec} \max_i |\tau_i| - 1$$

- The treewidth of a tree is 1, since the max clique (edge) in the original graph has size 2, and the optimal elimination order (eliminate all the leaves, then the root) adds no fill-in edges.

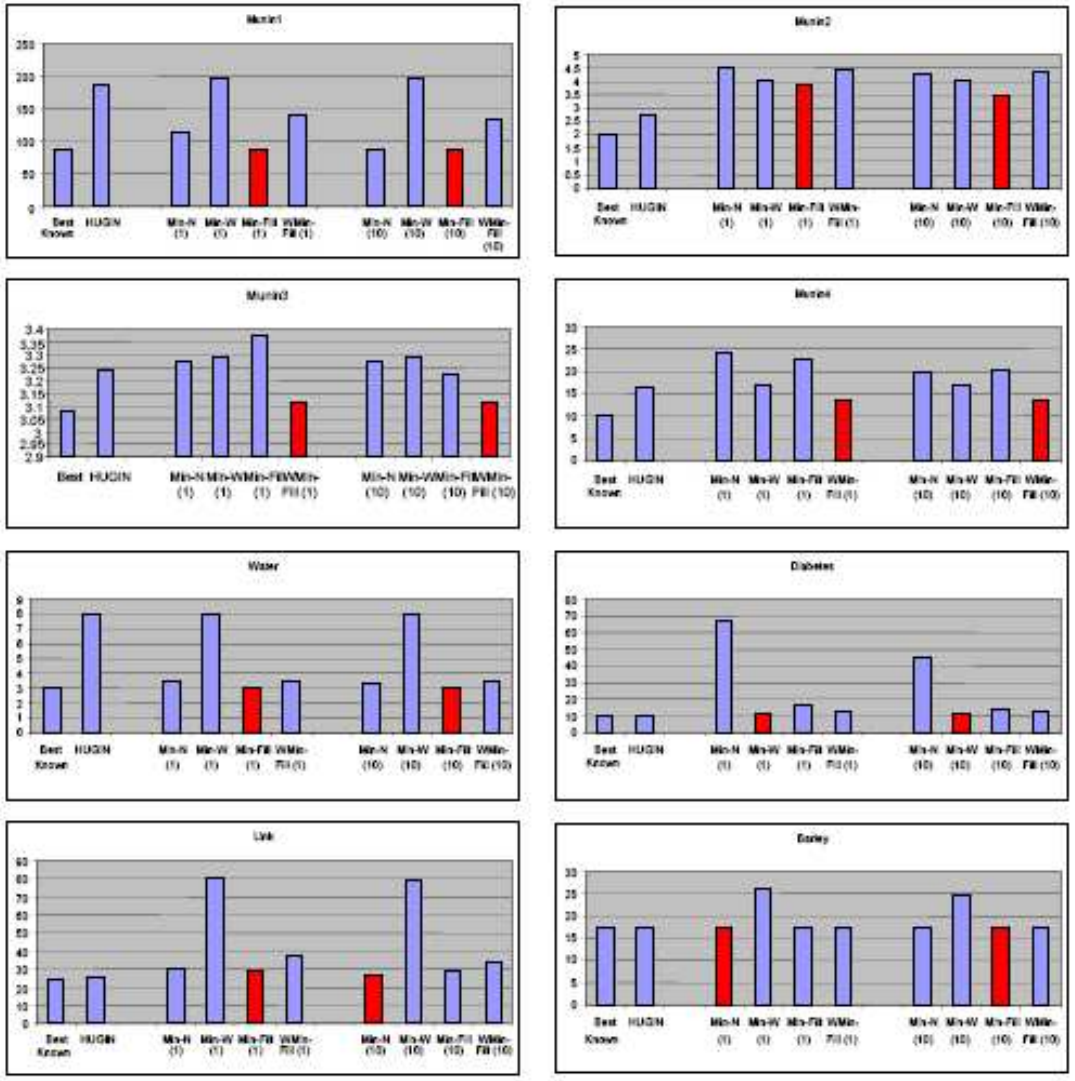
$$1,2,3 \quad : \quad \sum_{x_3} \sum_{x_2} \phi(x_3, x_2) \sum_{x_1} \phi(x_3, x_1)$$

$$3,2,1 \quad : \quad \sum_{x_1} \sum_{x_2} \sum_{x_3} \phi(x_3, x_1) \phi(x_3, x_2)$$

Finding an elim order

- Thm 9.4.6. Finding the optimal elimination order (which minimizes induced width) is NP-hard.
- Typical approach: greedy search, where at each step, we eliminate the node that minimizes some cost function
- Min-fill heuristic: the cost of a node is the number of fill-in edges that would be added.
- Min-weight heuristic: the cost of a node is the number of states in the factor that would be created (product of cardinalities).

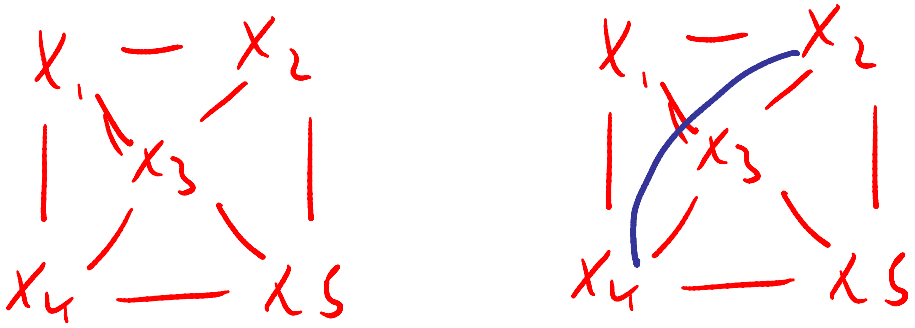
Empirical comparison of heuristics



Min-fill is often close to best known ordering (computed with simAnneal)

Chordal graphs

- Def 2.2.15. Let $X_1 – X_2$ - … $X_k – X_1$ be a loop in a graph. A chord is an edge connecting X_i and X_j for two nonconsecutive nodes. An undirected graph is chordal (triangulated) if every loop of length k >= 4 has a chord.

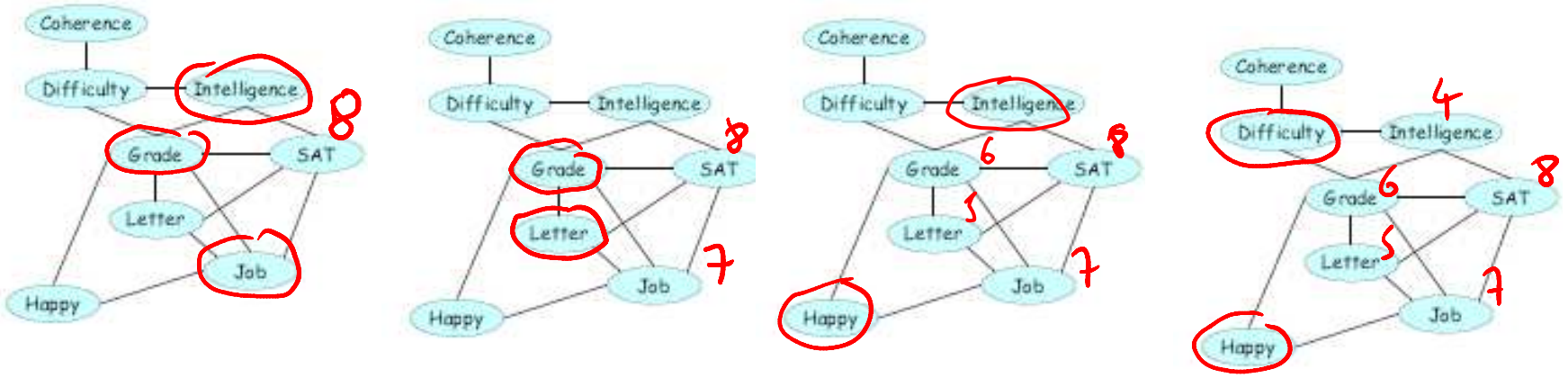


- Thm 9.4.7. Every induced graph is chordal.
- Thm 9.4.8. Any chordal graph admits a perfect elimination order which does not introduce any fill-in edges.

Finding perfect elim order

- The max cardinality search algorithm will find a perfect elimination ordering for a chordal graph.

```
Procedure Max-Cardinality (
    H    // An undirected graph over X
)
1    Initialize all nodes in X as unmarked
2    for k = |X| ... 1
3        X ← unmarked variable in X with largest number of marked neighbors
4        π(X) ← k
5        Mark X
6    return π
```



For non-chordal graphs, the MCS ordering often results in large induced width

Conditioning

- We can condition on a variable to break the graph into smaller pieces, run VE on each piece, and then add up the results. We also need the probability of each conditioning case.

$$\tilde{P}(\mathbf{Y}) = \sum_{\mathbf{u}} \tilde{P}(\mathbf{Y}, \mathbf{u}) \qquad Z = \sum_{\mathbf{u}} Z(\mathbf{u})$$

Evidence G=g

Condition on S

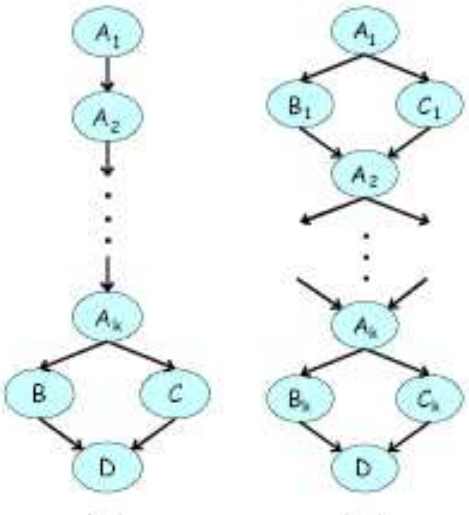
Conditioning + VE

Procedure Sum-Product-Conditioning (
  Φ, // Set of factors, possibly reduced by evidence
  $\boldsymbol{Y}$, // Set of query variables
  $\boldsymbol{U}$ // Set of variables on which to condition
)

1 **for** each $\boldsymbol{u} \in Val(\boldsymbol{U})$
2 $\quad \Phi_{\boldsymbol{u}} \leftarrow \{\phi[\boldsymbol{U} = \boldsymbol{u}] : \phi \in \Phi\}$
3 $\quad$ Construct $\mathcal{H}_{\Phi_{\boldsymbol{u}}}$
4 $\quad (\alpha_{\boldsymbol{u}}, \phi_{\boldsymbol{u}}(\boldsymbol{Y})) \leftarrow$ Cond-Prob-VE$(\mathcal{H}_{\Phi_{\boldsymbol{u}}}, \boldsymbol{Y}, \emptyset)$
5 $\phi^*(\boldsymbol{Y}) \leftarrow \frac{\sum_{\boldsymbol{u}} \phi_{\boldsymbol{u}}(\boldsymbol{Y})}{\sum_{\boldsymbol{u}} \alpha_{\boldsymbol{u}}}$
6 Return $\phi^*(\boldsymbol{Y})$

Cutset conditioning

- If we instantiate a set of nodes such that the resulting network is a tree, we can apply a simple message passing algorithm on the tree (see later).
- This is called cutset conditioning.
- Thm 9.5.2. Conditioning + VE is never more efficient than VE.

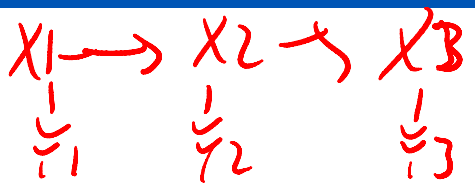


Left: condition on A_k. Repeatedly re-eliminate $A_1 \ldots A_{k-1}$ instead of reusing computation (as in DP/VE).

Right: condition on A_k, k odd. Exponential in k. But induced width is only 2.

Space-time tradeoff.

VE on chain = forwards algorithm



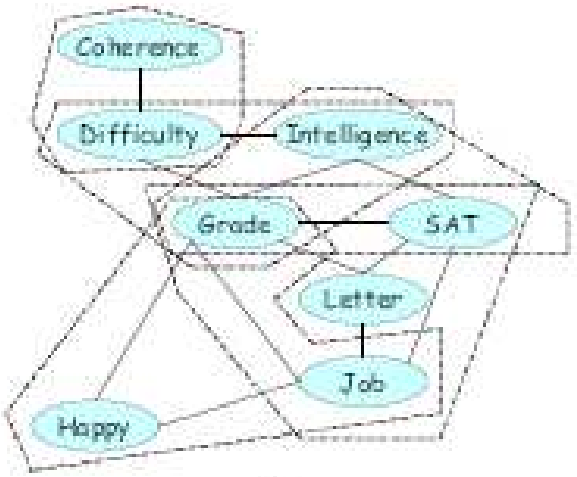
$$
\begin{aligned}
p(x_1, x_2, x_3 | y_1, y_2, y_3) &\propto \phi_1(x_1)\psi(x_1, x_2)\phi_2(x_2)\psi(x_2, x_3)\phi_3(x_3) \\
\phi_1(x_1) &= \pi_1(x_1)p(y_1|x_1) \\
\phi_t(x_t) &= p(y_t|x_t), t > 1 \\
\psi(x_{t-1}, x_t) &= p(x_t|x_{t-1}) \\
p(x_3|y_{1:3}) &\propto \phi_3(x_3) \sum_{x_2} \phi_2(x_2)\psi(x_2, x_3) \sum_{x_1} \phi_1(x_1)\psi(x_1, x_2) \\
\alpha_1(x_1) &\propto \phi_1(x_1) \\
\alpha_2(x_2) &\propto \phi_2(x_2) \sum_{x_1} \alpha_1(x_1)\psi(x_1, x_2) \\
\alpha_3(x_3) &\propto \phi_3(x_3) \sum_{x_2} \alpha_2(x_2)\psi(x_2, x_3)
\end{aligned}
$$

What's wrong with VE?

- Consider a chain X1 – X2 - .. – XT, where the local evidence has been absorbed into the node factors.
- If we use VE to compute p(XT|y(1:T)), it is equivalent to the forwards algorithm for HMMs, and takes $O(T K^2)$ time, where K = #states.
- Suppose we also want to compute p(X(T-1)|y(1:T)). We could rerun the algorithm for an additional $O(T K^2)$ time.
- We now discuss how to reuse most of the computation we have already done in eliminating X(1:T-2). We can then compute all marginals in $O(2 K^2 T)$ time (FB algorithm).

Cluster graphs

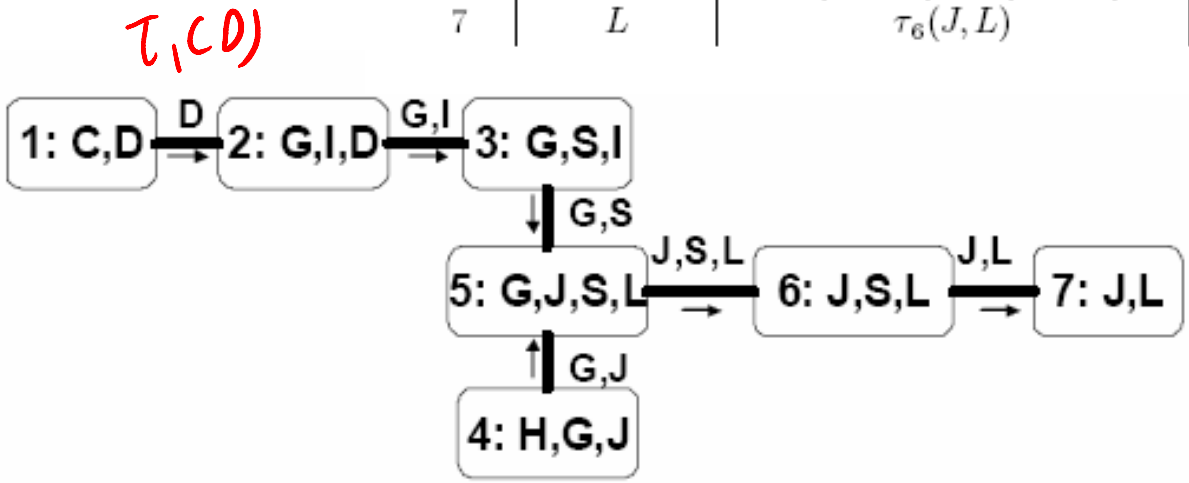
- Def 10.1.1. A cluster graph for a set of factors on X is an undirected graph, each of whose nodes I is associated with a set $C_i \subseteq X$. Each factor is contained in precisely one cluster. Each edge between a pair of clusters C_i, C_j is associated with a sepset (separating set) S_{ij}. $S_{ij} = C_i \cap C_j$



Cluster graph from VE

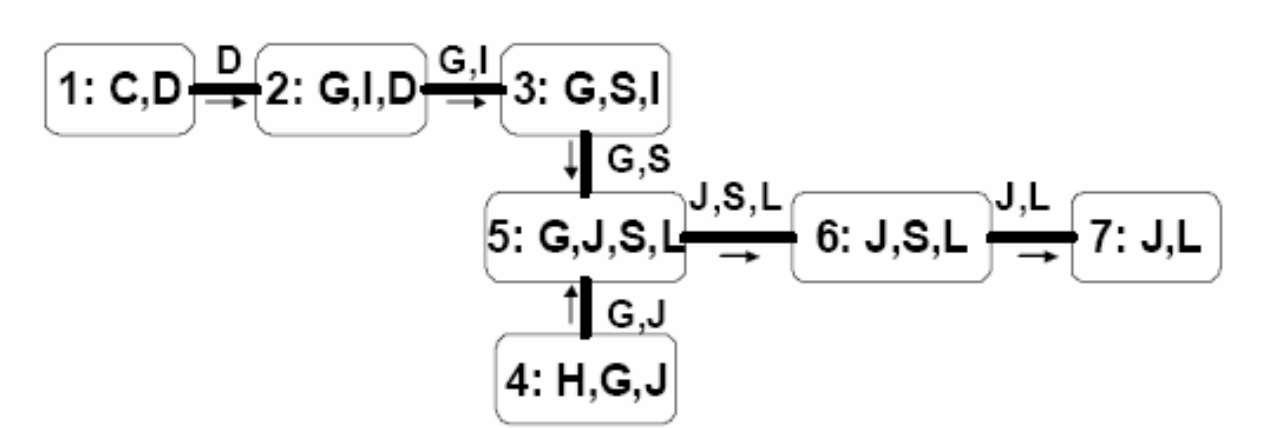
- We can create a cluster graph to represent the process of VE. Before we marginalize out x_i, we create factor ψ_i (its bucket potential); make this a cluster. When we marginalize out x_i, we create factor τ_i which is stored in bucket j; think of this as a message from i to j. Draw an edge $C_i - C_j$.

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C), \phi_D(D,C)$	C, D	$\tau_1(D)$
2	D	$\phi_G(G,I,D), \tau_1(D)$	G, I, D	$\tau_2(G,I)$
3	I	$\phi_I(I), \phi_S(S,I), \tau_2(G,I)$	G, S, I	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	H, G, J	$\tau_4(G,J)$
5	G	$\tau_4(G,J), \tau_3(G,S), \phi_L(L,G)$	G, J, L, S	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S), \phi_J(J,L,S)$	J, L, S	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	J, L	$\tau_7(J)$



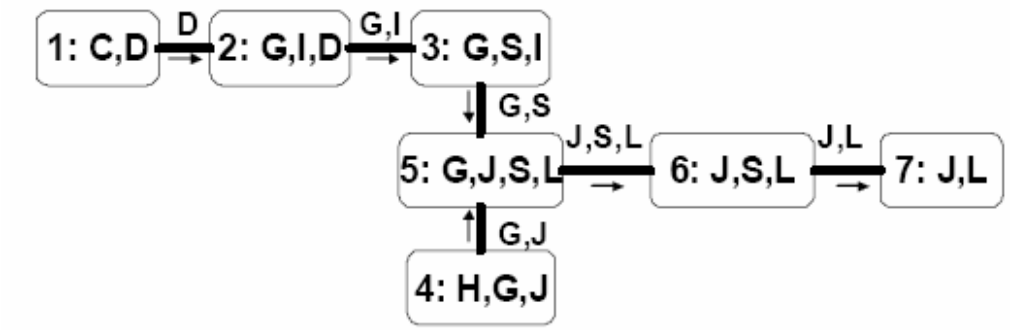
Properties of VE cluster graph

- The VE cluster graph is a tree, since each message gets sent to a single bucket (so each cluster connects to at most one other cluster)
- Def 10.1.3. Let T be a cluster tree. T has the **running intersection property** if , whenever X in Ci and X in Cj, then X is also in every cluster on the unique path from Ci to Cj.
- Thm 10.1.5. The VE CG has RIP.
- Pf (sketch). A variable appears in every factor from the moment it is introduced to when it is summed out.



Messages

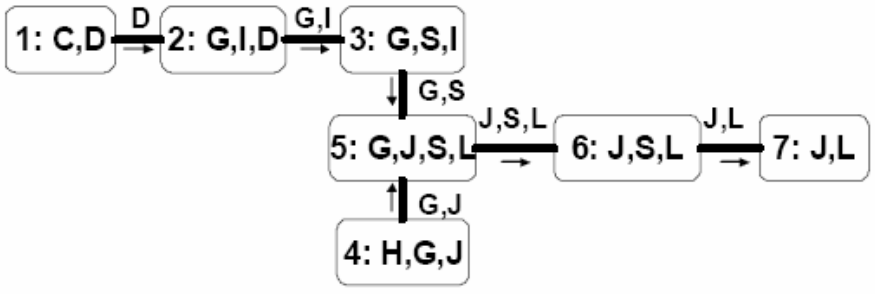
- Thm 10.1.6. The scope of the msg τ_i from C_i to C_i is $S_{i,j}$.

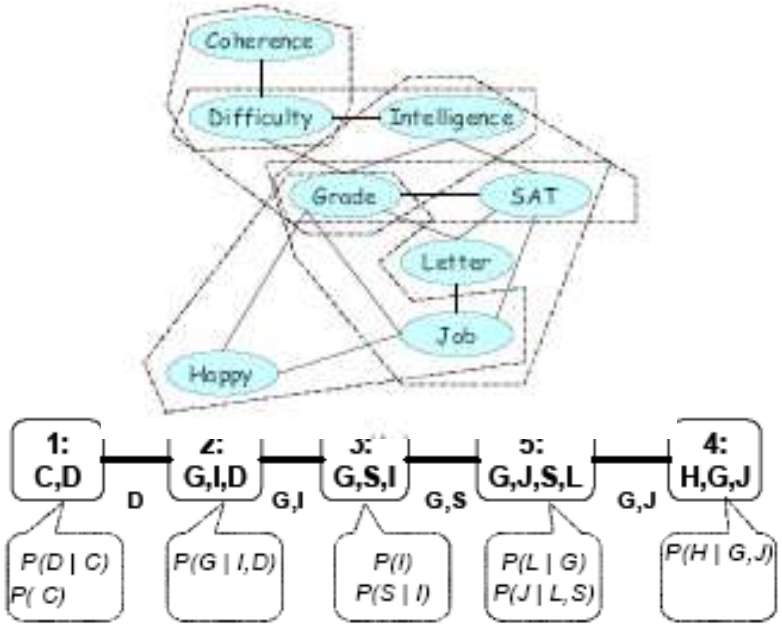


- Def. For any sepset S_{ij}, let $W_{<ij}$ be the variables in the scope of the clusters on the C_i side, and $W_{<ji}$ be the vars on the C_j side.
- Thm 10.1.8. T satisfies RIP iff for every S_{ij}, $W_{<ij} \perp W_{<ji} \mid S_{ij}$.
- Hence msg from C_i to C_j is sufficient statistic for all info to left of C_i – C_j.
- RIP ensures local communication => global consistency.

Clique trees

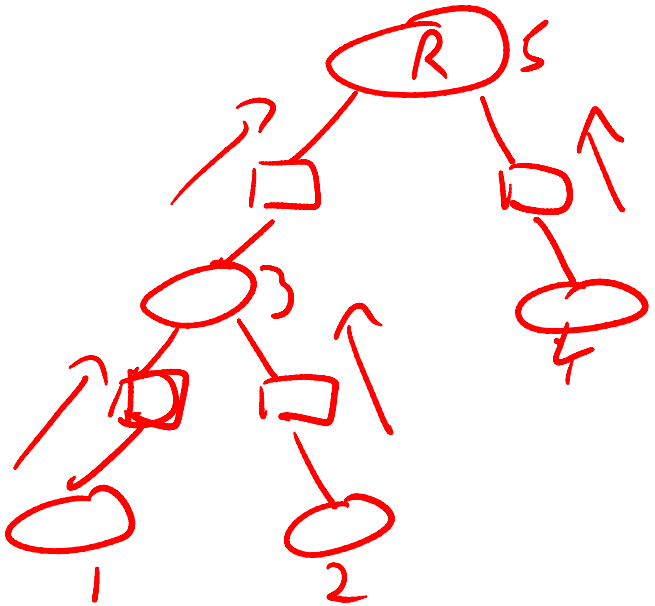
- Def 10.1.7. A cluster tree that satisfies RIP is called a clique tree or join tree or junction tree.
- Thm 4.5.15. A graph has a Jtree (where the clusters are the maxcliques) iff it is chordal.
- Thm 10.4.1. We can always remove non maximal cliques from a Jtree without violating RIP.





Message passing on a clique tree

- To compute p(X_i), find a clique that contains X_i, make it the root, and send messages to it from all other nodes.
- A clique cannot send a node to its parent until it is ready, ie. Has received msgs from all its children.
- Hence we send from leaves to root.

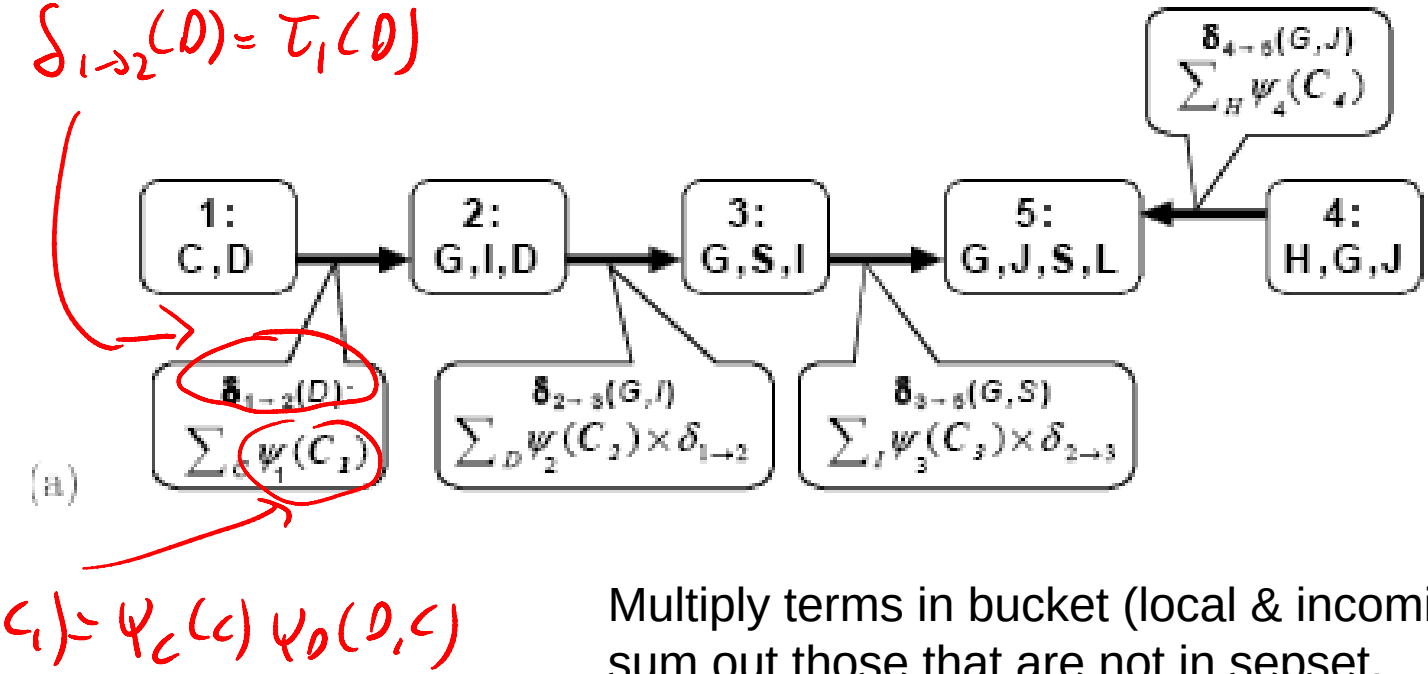


Message passing on a clique tree

- .

$$P(J) = \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \psi_G(G,I,D) \underbrace{\sum_C \psi_C(C)\psi_D(D,C)}_{\tau_1(D)}$$

$$= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_G(G,I,D)\tau_1(D)}_{\tau_2(G,I)}$$



Multiply terms in bucket (local & incoming), sum out those that are not in sepset, send to nbr upstream

Upwards pass (collect to root)

Procedure CTree Sum Product Up (
  Φ, // Set of factors
  $\mathcal{T}$, // Clique tree over Φ
  α, // Initial assignment of factors to cliques
  C_r // Some selected root clique
)

1 Initialize Cliques
2 **while** C_r is not ready
3 Let C_i be a ready clique
4 $\delta_{i \to p_r(i)}(S_{i,p_r(i)}) \leftarrow$ SP Message$(i, p_r(i))$
5 $\beta_r \leftarrow \psi_r \cdot \prod_{k \in \text{Nb}_{C_r}} \delta_{k \to r}$
6 **return** β_r

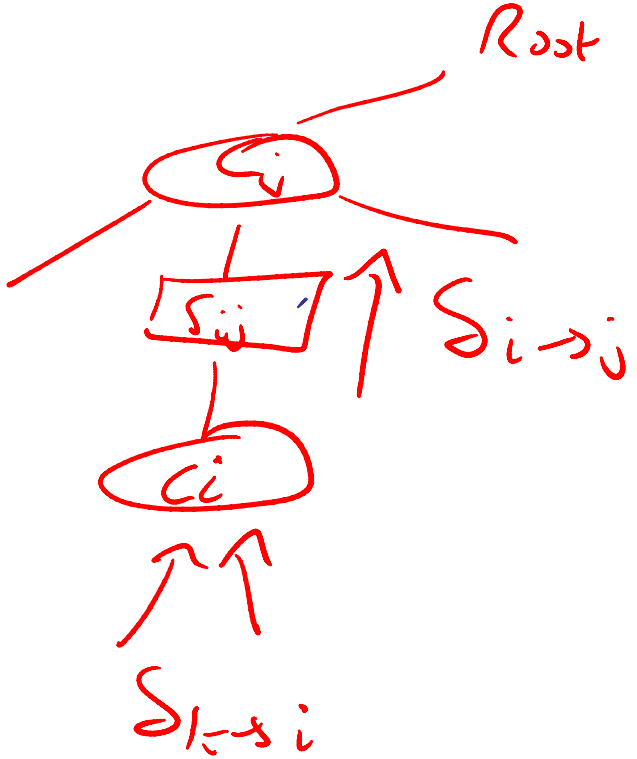
Procedure Initialize Cliques (

)

1 **for** each clique C_i
2 $\psi_i[C_i] \leftarrow \prod_{\phi_j : \alpha(\phi_j)=i} \phi$
3

Procedure SP Message (
  i, // sending clique
  j // receiving clique
)

1 $\psi(C_i) \leftarrow \psi_i \cdot \prod_{k \in (\text{Nb}_i - \{j\})} \delta_{k \to i}$
2 $\tau(S_{i,j}) \leftarrow \sum_{C_i - S_{i,j}} \psi(C_i)$
3 **return** $\tau(S_{i,j})$

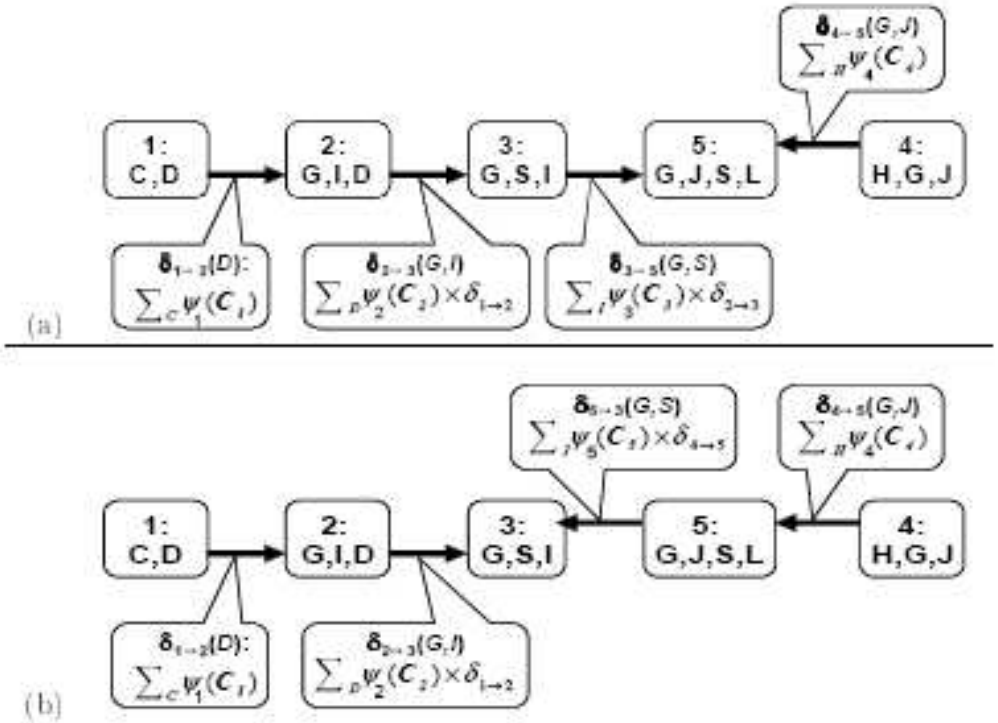


$$\beta_i(C_i) = \phi_i(C_i) \prod_{k \in n_i, k \neq j} \delta_{k \to i}(S_{k,i})$$

$$\delta_{i \to j}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \beta_i(C_i)$$

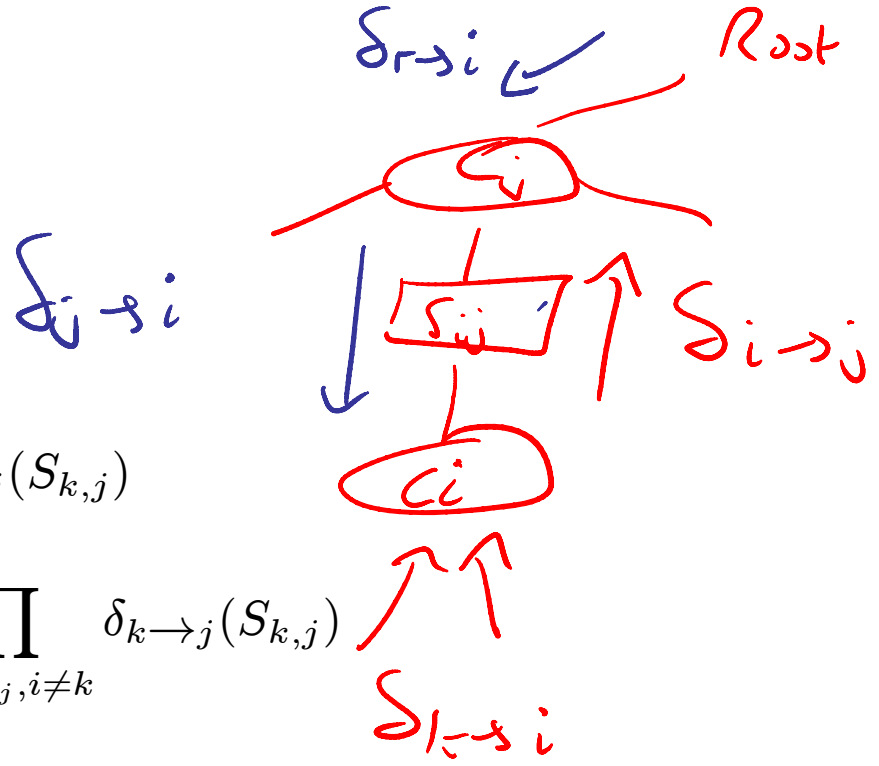
Message passing to a different root

- If we send messages to a different root, many of them will be the same
- Hence if we send messages to all the cliques, we can reuse the messages- dynamic programming!



Downwards pass (distribute from root)

- At the end of the upwards pass, the root has seen all the evidence.
- We send back down from root to leaves.



$$\beta_j(C_j) = \phi_j(C_j) \prod_{k \in n_j} \delta_{k \to j}(S_{k,j})$$

$$\delta_{j \to i}(S_{ij}) = \sum_{C_j \setminus S_{ij}} \phi_j(C_j) \prod_{k \in n_j, i \neq k} \delta_{k \to j}(S_{k,j})$$

$$= \sum_{C_j \setminus S_{ij}} \frac{\beta_j(C_j)}{\delta_{i \to j}(S_{ij})}$$

Use division operator to avoid double counting

Beliefs

- Thm 10.2.7. After collect/distribute, each clique potential represents a marginal probability (conditioned on the evidence)

$$\beta_i(C_i) = \sum_{\mathbf{x}\, C_i} \tilde{P}(\mathbf{x})$$

- If we get new evidence on X_i, we can multiply it in to any clique containing i, and then distribute messages outwards from that clique to restore consistency.

MAP configuration

- We can generalize the Viterbi algorithm to find a MAP configuration as follows.
- On the upwards pass, replace sum with max.
- At the root, find the most probable joint setting and send this as evidence to the root's children.
- Each child finds its most probable setting and sends this to its children.
- The jtree property ensures that when the state of a variable is fixed in one clique, that variable assumes the same state in all other cliques.

Samples

- We can generalize forwards-filtering backwards-sampling to draw exact samples from the joint as follows.
- Do a collect pass to the root as usual.
- Sample xR from the root marginal, and then enter it as evidence in all the children.
- Each child then samples itself from its updated local distribution and sends this to its children.

Calibrated clique tree

- Def 102.8. A clique tree is calibrated if, for all pairs of neighboring cliques, we have

$$\sum_{C_i \setminus S_{i,j}} \beta_i(C_i) = \sum_{C_j \setminus S_{i,j}} \beta_j(C_j) = \mu_{i,j}(S_{i,j})$$

- Eg. A-B-C clq tree AB – [B] – BC. We require

$$\sum_a \beta_{ab}(a,b) = \sum_c \beta_{bc}(b,c)$$

- Thm. After collect/distribute, all cliques are calibrated.
- Thm 10.2.12. A calibrated tree defines a joint distribution as follows $p(x) = \frac{\prod_i \beta_i(C_i)}{\prod_{<ij>} \mu_{i,j}(S_{ij})}$

eg $p(A,B,C) = \frac{p(A,B)p(B,C)}{p(C)} = p(A,B)p(C|B) = p(A|B)p(B,C)$

Clique tree invariant

- Suppose at every step, clique i sends a msg to clique j, and stores it in $\mu_{i,j}$:

Procedure Send-BU-Msg (
  i, // sending clique
  j // receiving clique
)

1 $\sigma_{i \to j} \leftarrow \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i$
2 // marginalize the clique over the sepset
3 $\beta_j \leftarrow \beta_j \cdot \frac{\sigma_{i \to j}}{\mu_{i,j}}$
4 $\mu_{i,j} \leftarrow \sigma_{i \to j}$

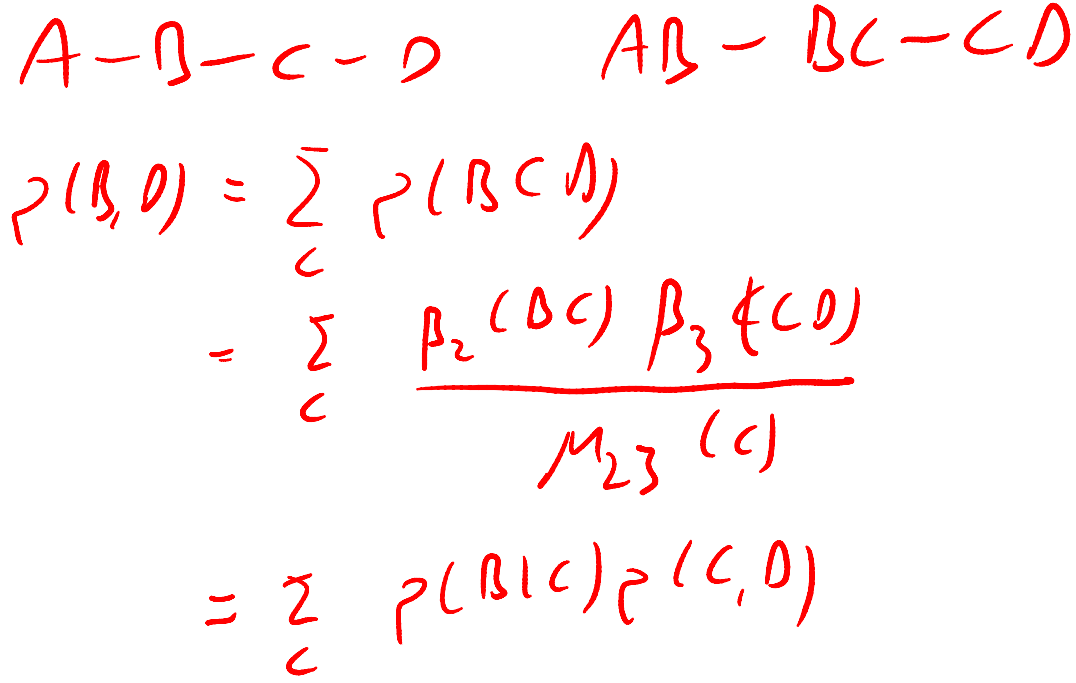
- Initially $\mu_{i,j}=1$ and $\beta_i = \prod_{f:\ f \text{ ass to } i} \phi_f$. Hence the following holds.

$$p(x) = \frac{\prod_i \beta_i(C_i)}{\prod_{<ij>} \mu_{i,j}(S_{ij})}$$

- Thm 10.3.4. This property holds after every belief updating operation.

Out of clique queries

- We can compute the distribution on any set of variables inside a clique. But suppose we want the joint on variables in different cliques. We can run VE on the calibrated subtree
- eg

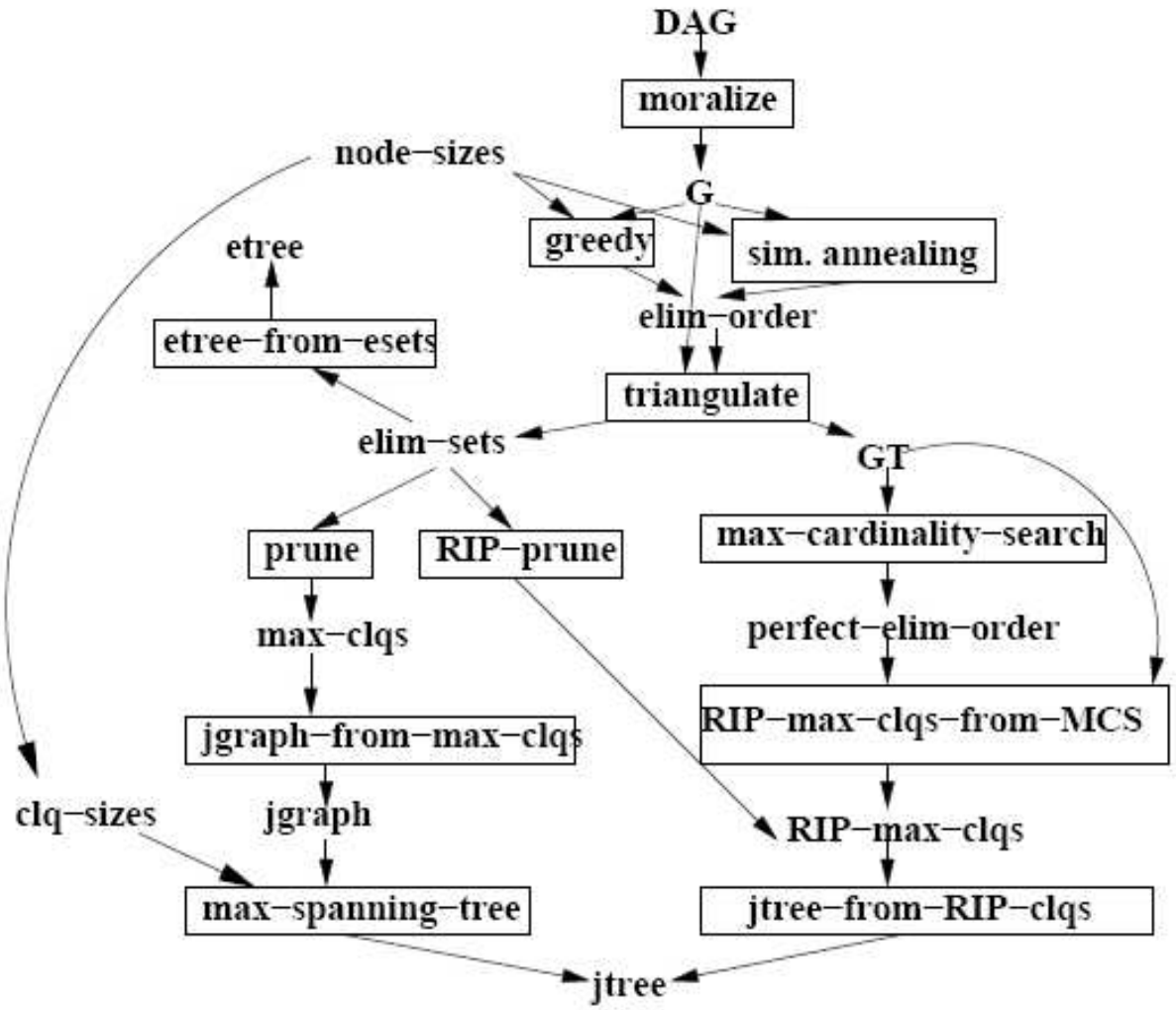


Out of clique inference

Procedure CTree-Query (
  $\mathcal{T}$, // Clique tree over Φ
  $\{\beta_i\}, \{\mu_{i,j}\}$, // Calibrated clique and sepset beliefs for $\mathcal{T}$
  $\mathbf{Y}$ // A query
)

Let $\mathcal{T}'$ be a subtree of $\mathcal{T}$ such that $\mathbf{Y} \subseteq Scope[\mathcal{T}']$
Select a clique $r \in \mathcal{V}_{\mathcal{T}'}$ to be the root
$\Phi \leftarrow \beta_r$
for each $i \in \mathcal{V}_{\mathcal{T}}$
  $\phi \leftarrow \frac{\beta_i}{\mu_{i,Pr(i)}}$
  $\Phi \leftarrow \Phi \cup \{\phi\}$
$\mathbf{Z} \leftarrow Scope[\mathcal{T}'] - \mathbf{Y}$
Let $\prec$ be some ordering over $\mathbf{Z}$
return Sum-Product-Variable-Elimination$(\Phi, \mathbf{Z}, \prec)$

Creating a Jtree



Murphy PhD thesis (2002) p140

Max cliques from a chordal graph

- Triangulate the graph according to some ordering.
  - Start with all vertices unnumbered, set counter $i := N$.
  - While there are still some unnumbered vertices:
    - Let $v_i = \pi(i)$.
    - Form the set C_i consisting of v_i and its (unnumbered/ uneliminated) neighbors.
    - Fill in edges between all pairs of vertices in C_i.
    - Eliminate v_i and decrement i by 1.
- At each step, keep track of the clique that is created; if it is a subset of any previously created clique, discard it (since non maximal).

Cliques to Jtree

- Build a weighted graph where
  $W_{ij} = |C_i \text{ intersect } C_j|$
- Find max weight spanning tree. This is a jtree.