# Probabilistic graphical models
# CPSC 532c (Topics in AI)
# Stat 521a (Topics in multivariate analysis)

## Lecture 9

Kevin Murphy

Monday 18 October 2004

## Administrivia

- HW4 due today

## Review

- Variable elimination can be used to answer a single query, $P(X_q|e)$.
- VarElim requires an elimination ordering; you can use `elimOrderGreedy` to find this.
- VarElim implicitly creates an elimination tree (a junction tree with non-maximal cliques).
- You can create a jtree of maximal cliques by triangulating and using max weight spanning tree.
- Given a jtree, we can compute $P(X_c|e)$ for all cliques $c$ using belief propagation (BP).

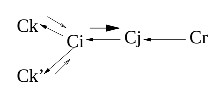
## Belief propagation

- There are 2 variants of BP, which we will cover today:
- Shafer-Shenoy, that multiplies by all-but-one incoming message:

$$\delta_{i \longrightarrow j} = f\left( \prod_{k \in N_i \setminus \{j\}} \delta_{k \longrightarrow i} \right)$$

- Lauritzen-Spiegelhalter, that multiplies by all incoming messages and then divides out by one

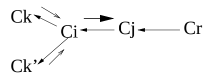
$$\delta_{i \longrightarrow j} = f\left( \frac{\prod_{k \in N_i} \delta_{k \longrightarrow i}}{\delta_{j \longrightarrow i}} \right)$$



## Shafer-Shenoy algorithm

$\{\psi_i^1\} \stackrel{\text{def}}{=}$ function Ctree-VE-calibrate$(\{\phi\}, T, \alpha)$

$R :=$ pickRoot$(T)$
$DT :=$ mkRootedTree$(T, R)$
$\{\psi_i^0\} :=$ initializeCliques$(\phi, \alpha)$
(* Upwards pass *)
for $\quad i \in$ postorder$(DT)$
$\quad j := pa(DT, i)$
$\quad \delta_{i \to j} :=$ VE-msg$(\{\delta_{k \to i} : k \in ch(DT, i)\}, \psi_i^0)$




## Sub-functions

$\{\psi_i^0\} \stackrel{\text{def}}{=}$ function initializeCliques$(\phi, \alpha)$

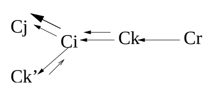
for $\quad i := 1 : C$
$\quad \psi_i^0(C_i) = \prod_{\phi:\alpha(\phi)=i} \phi$

$\delta_{i \to j} \stackrel{\text{def}}{=}$ function VE-msg$(\{\delta_{k \to i}\}, \psi_i^0)$

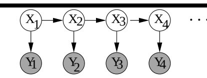
$\psi_i^1(C_i) := \psi_i^0(C_i) \prod_k \delta_{k \to i}$
$\delta_{i \to j}(S_{i,j}) := \sum_{C_i \setminus S_{ij}} \psi_i^1(C_i)$

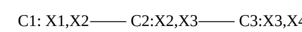
## Shafer-Shenoy algorithm

(* Downwards pass *)
for $i \in$ preorder$(DT)$
$\quad$ for $j \in ch(DT, i)$
$\quad\quad \delta_{i \to j} =$ VE-msg$(\{\delta_{k \to i} : k \in N_i \setminus j\}, \psi_i^0)$
(* Combine *)
for $i := 1 : C$
$\quad \psi_i^1 := \psi_i^0 \prod_{k \in N_i} \delta_{k \to i}$




## Shafer Shenoy for HMMs



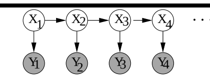




$$
\begin{aligned}
\psi_t^0(X_t, X_{t+1}) &= P(X_{t+1}|X_t)p(y_{t+1}|X_{t+1}) \\
\delta_{t \to t+1}(X_{t+1}) &= \sum_{X_t} \delta_{t-1 \to t}(X_t)\psi_t^0(X_t, X_{t+1}) \\
\delta_{t \to t-1}(X_t) &= \sum_{X_{t+1}} \delta_{t+1 \to t}(X_{t+1})\psi_t^0(X_t, X_{t+1}) \\
\psi_t^1(X_t, X_{t+1}) &= \delta_{t-1 \to t}(X_t)\delta_{t+1 \to t}(X_{t+1})\psi_t^0(X_t, X_{t+1})
\end{aligned}
$$

## Forwards-backwards algorithm for HMMs

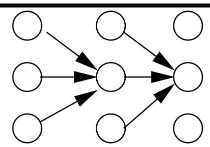
$$
\begin{aligned}
\alpha_t(i) &\stackrel{\text{def}}{=} \delta_{t-1\to t}(i) = P(X_t = i, y_{1:t}) \\
\beta_t(i) &\stackrel{\text{def}}{=} \delta_{t\to t-1}(i) = p(y_{t+1:T}|X_t = i) \\
\xi_t(i,j) &\stackrel{\text{def}}{=} \psi_t^1(X_t = i, X_{t+1} = j) = P(X_t = i, X_{t+1} = j, y_{1:T} \\
P(X_{t+1} = j|X_t = i) &\stackrel{\text{def}}{=} A(i,j) \\
p(y_t|X_t = i) &\stackrel{\text{def}}{=} B_t(i) \\
\alpha_t(j) &= \sum_i \alpha_{t-1}(i) A(i,j) B_t(j) \\
\beta_t(i) &= \sum_j \beta_{t+1}(j) A(i,j) B_{t+1}(j) \\
\xi_t(i,j) &= \alpha_t(i)\beta_{t+1}(j) A(i,j) B_{t+1}(j) \\
\gamma_t(i) &\stackrel{\text{def}}{=} P(X_t = i|y_{1:T}) \propto \alpha_t(i)\beta_t(j) \propto \sum_j \xi_t(i,j)
\end{aligned}
$$

## Forwards-backwards algorithm, matrix-vector form




$$
\begin{aligned}
\alpha_t(j) &= \sum_i \alpha_{t-1}(i) A(i,j) B_t(j) \\
\alpha_t &= (A^T \alpha_{t-1}) .* B_t \\
\beta_t(i) &= \sum_j \beta_{t+1}(j) A(i,j) B_{t+1}(j) \\
\beta_t &= A(\beta_{t+1} .* B_{t+1}) \\
\xi_t(i,j) &= \alpha_t(i)\beta_{t+1}(j) A(i,j) B_{t+1}(j) \\
\xi_t &= \left(\alpha_t(\beta_{t+1} .* B_{t+1})^T\right) .* A \\
\gamma_t(i) &\propto \alpha_t(i)\beta_t(j) \\
\gamma_t &\propto \alpha_t .* \beta_t
\end{aligned}
$$

## HMM trellis



- Forwards algorithm uses dynamic programming to efficiently sum over all possible paths that state $i$ at time $t$.

$$
\begin{aligned}
\alpha_t(i) &\stackrel{\text{def}}{=} P(X_t = i, y_{1:t}) \\
&= \left[\sum_{X_1} \cdots \sum_{X_{t-1}} P(X_1, \ldots, X_t - 1, y_{1:t-1}) P(X_t|X_{t-1})\right] p(y_t|X_t) \\
&= \left[\sum_{X_{t-1}} P(X_t - 1, y_{1:t-1}) P(X_t|X_{t-1})\right] p(y_t|X_t) \\
&= \left[\sum_{X_{t-1}} \alpha_{t-1}(X_{t-1}) P(X_t|X_{t-1})\right] p(y_t|X_t)
\end{aligned}
$$

## Avoiding numerical underflow in HMMs

- $\alpha_t(j) \stackrel{\text{def}}{=} P(X_t = j, y_{1:t})$ is a tiny number
- Hence in practice we use

$$
\begin{aligned}
\hat{\alpha}_t(j) &\stackrel{\text{def}}{=} P(X_t = j|y_{1:t}) = \frac{P(X_t, y_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \\
&= \frac{\sum_i P(X_{t-1} = i|y_{1:t-1}) P(X_t = j|X_{t-1} = i) p(y_t|X_t = j)}{p(y_t|y_{1:t-1})} \\
&= \frac{1}{c_t} \sum_i \hat{\alpha}_{t-1}(i) A(i,j) B_t(j)
\end{aligned}
$$

where

$$
c_t \stackrel{\text{def}}{=} P(y_t|y_{1:t-1}) = \sum_j \sum_i \hat{\alpha}_{t-1}(i) A(i,j) B_t(j)
$$

$$
\log p(y_{1:T}) = \log p(y_1) p(y_2|y_1) p(y_3|y_{1:2}) \ldots = \log \prod_{t=1}^T c_t = \sum_{t=1}^T \log c_t
$$

## Avoiding numerical underflow in Shafer Shenoy

- We always normalize all the messages

$$\hat{\delta}_{i \to j} = \frac{1}{z_i} \text{VE-msg}(\hat{\delta}_{k \to i}, \psi_i^0)$$

- By keeping track of the normalization constants during the collect-to-root, we can compute the log-likelihood

$$\log p(e) = \sum_i \log z_i$$

## Shafer-Shenoy for pairwise MRFs

- Consider an MRF with one potential per edge

$$P(X) = \frac{1}{Z} \prod_{<ij>} \psi_{ij}(X_i, X_j) \prod_i \phi_i(X_i)$$

- We can generalize the forwards-backwards algorithm as follows:

$$m_{ij}(x_j) = \sum_{x_i} \phi_i(x_i) \psi_{ij}(x_i, x_j) \prod_{k \in N_i \setminus \{j\}} m_{ji}(x_i)$$

$$b_i(x_i) \propto \phi_i(x_i) \prod_{j \in N_i} m_{ji}(x_i)$$

- In matrix-vector form, this becomes

$$m_{ij} = \phi_i .* \psi_{ij}^T \prod_k m_{ki}$$

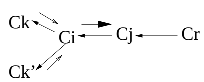
$$b_i \propto \phi_i .* \prod_{j \in N_i} m_{ji}$$

## Message passing with division

- The posterior is the product of all incoming messages

$$\pi_i(C_i) = \pi_i^0(C_i) \prod_{k \in N_i} \delta_{k \to i}(S_{ik})$$

- The message from $i$ to $j$ is the product of all incoming messages excluding $\delta_{j \to i}$:




$$\delta_{i \to j}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \pi_i^0(C_i) \prod_{k \in N_i \setminus \{j\}} \delta_{k \to i}(S_{ik})$$

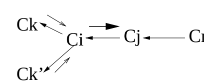
$$= \sum_{C_i \setminus S_{ij}} \pi_i^0(C_i) \frac{\prod_{k \in N_i} \delta_{k \to i}(S_{ik})}{\delta_{j \to i}(S_{ij})}$$

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

## Lauritzen-Spiegelhalter algorithm

$\{\psi_i\} \stackrel{\text{def}}{=}$ function Ctree-BP-two-pass$(\{\phi\}, T, \alpha)$

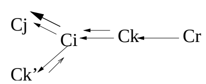
$R :=$ pickRoot$(T)$
$DT :=$ mkRootedTree$(T, R)$
$\{\psi_i\} :=$ initializeCliques$(\phi, \alpha)$
$\mu_{i,j} := 1$ (* initialize messages for each edge *)
(* Upwards pass *)
for $i \in$ postorder$(DT)$
    $j := pa(DT, i)$
    $[\psi_j, \mu_{i,j}] :=$ BP-msg$(\psi_i, \psi_j, \mu_{i,j})$



## Lauritzen-Spiegelhalter algorithm

(* Downwards pass *)
for $i \in$ preorder$(DT)$
    for $j \in ch(DT, i)$
    $[\psi_j, \mu_{ij}] :=$ BP-msg$(\psi_i, \psi_j, \mu_{i,j})$

$[\psi_j, \mu_{i,j}] \stackrel{\text{def}}{=}$ function BP-msg$(\psi_i, \psi_j, \mu_{i,j})$
$\delta_{ij} := \sum_{C_i \setminus S_{ij}} \psi_i$
$\psi_j := \psi_j * \frac{\delta_{i \rightarrow j}}{\mu_{i,j}}$
$\mu_{i,j} := \delta_{i \rightarrow j}$




## Properties of BP

- $\mu_{i,j}$ stores the most recent message sent alone edge $C_i - C_j$, in either direction.
- We can send messages in any order, including multiple times, because the recipient divides out by the old $\mu_{i,j}$, to avoid overcounting.
- Hence the algorithm can be run in a parallel, distributed fashion.
- $\psi_i \propto P(C_i|e')$ contains the product of all received messages so far (summarizing evidence $e'$); it is our best partial guess (belief) about $P(C_i|e)$.

## Parallel BP

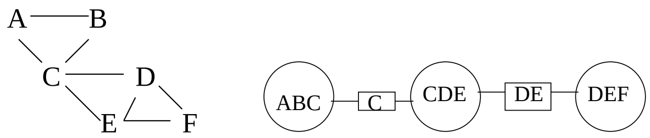
(* send *)
for $i = 1 : C$
    for $j \in N_i$
        $\delta^{old}_{i \rightarrow j} = \delta_{i \rightarrow j}$
        $\delta_{i \rightarrow j} = \sum_{C_i \setminus S_{ij}} \psi_i$
    end
end
(* receive *)
for $i = 1 : C$
    for $j \in N_i$
        $\psi_i := \psi_i * \frac{\delta_{j \rightarrow i}}{\delta^{old}_{i \rightarrow j}}$
    end
end

## Using a clique tree to answer queries

- We can enter evidence about $X_i$ by multiplying a local evidence factor into any potential that contains $X_i$ in its scope.
- After the tree is calibrated, we can compute $P(X_q|e)$ for any $q$ contained in a clique (e.g., a node and its parents).
- If new evidence arrives about $X_i$, we pick a clique $C_r$ that contains $X_i$ and distribute the evidence (downwards pass from $C_r$).

## Separator sets

- Define the separator sets on each edge to be $S_{ij} = C_i \cap C_J$.
- Thm 8.1.8: Let $X_i$ be all the nodes to the "left" of $S_{ij}$ and $X_j$ be all the nodes to the "right". Then $X_i \perp X_j | S_{ij}$.
- $ABCDE \perp DEF | DE$, i.e., $ABC \perp F | DE$.




## Clique tree as a distribution

- Consider Markov net $A - B - C$ with clique tree

$$C1 : A, B - C2 : B, C$$

- After BP has converged, we have

$$\psi_1(A,B) = P_F(A,B), \psi_2(B,C) = P_F(B,C)$$

- In addition, neighboring cliques agree on their intersection, e.g.

$$\sum_A \psi_1(A,B) = \sum_C \psi_2(B,C) = P_F(B)$$

- Hence the joint is

$$\begin{aligned} P(A,B,C) &= P(A,B)P(C|B) = P(A,B)\frac{P(B,C)}{P(B)} \\ &= \psi_1(A,B)\frac{\psi_2(B,C)}{\sum_c \psi_2(B,c)} = \psi_1(A,B)\frac{\psi_2(B,C)}{\sum_a \psi_1(a,c)} \\ &= \psi_1(A,B)\frac{\psi_2(B,C)}{\mu_{1,2}(B)} \end{aligned}$$

## Clique tree as a distribution

- Defn 8.9: The clique tree invariant for $T$ is

$$\pi_T = \prod \phi = \frac{\prod_{i \in T} \psi_i(C_i)}{\prod_{<ij \in T>} \mu_{i,j}(S_{i,j})}$$

- Initially, the clique tree over all factors satisfies the invariant since $\mu_{i,j} = 1$ and all the factors $\phi$ are assigned to cliques.
- Thm 8.3.6: Each step of BP maintains the clique invariant.

## Message passing maintains clique invariant

- Proof. Suppose $C_i$ sends to $C_j$ resulting in new message $\mu_{i,j}^{new}$ and new potential

$$\psi_j^{new} = \psi_j \frac{\mu_{ij}^{new}}{\mu_{ij}}$$

Then

$$\begin{aligned} \pi_T &= \frac{\prod_{i'} \psi_{i'}^{new}}{\prod_{<ij>'} \mu_{i',j'}^{new}} \\ &= \frac{\psi_j^{new} \prod_{i' \neq j} \psi_{i'}}{\mu_{ij}^{new} \prod_{<ij>' \neq (i,j)} \mu_{i',j'}} \\ &= \frac{\psi_j \mu_{ij}^{new}}{\mu_{ij}} \frac{\prod_{i' \neq j} \psi_{i'}}{\mu_{ij}^{new} \prod_{<ij>' \neq (i,j)} \mu_{i',j'}} \\ &= \frac{\prod_{i'} \psi_{i'}}{\prod_{<ij>'} \mu_{i',j'}} \end{aligned}$$

## Proof of correctness of BP

- Message passing does not change the invariant, so the clique tree always represents the distribution as a whole.
- However, we want to show that when the algorithm has converged, the clique potentials represent correct marginals.
- Defn 8.3.7. $C_i$ is **ready to transmit** to $C_j$ when $C_i$ has received informed messages from all its neighbors except from $C_j$; a message from $C_i$ to $C_j$ is **informed** if it is sent when $C_i$ is ready to transmit to $C_j$.
- e.g., leaf nodes are always ready to transmit.
- Defn 8.3.8: A connected subtree $T'$ is fully informed if, for each $C_i \in T'$ and each $C_j \notin T'$, we have that $C_j$ has sent an informed message to $C_i$.
- Thm 8.3.9: After running BP, then $\pi_{T'} = P_F(Scope(T'))$ for any fully informed connected subtree $T'$.

## Proof of correctness of BP

- Corollary 8.3.10: If all nodes in $T$ are fully informed, then $\pi_T = P_F(Scope(T))$. Hence $\pi_i = P_F(C_i)$.
- Claim: There is a scheduling such that all nodes can become fully informed (namely postorder/ preorder).
- Defn 8.3.11. A clique tree is said to be **calibrated** if for each edge $C_i - C_j$, they agree on their intersection

$$\sum_{C_i \backslash S_{ij}} \psi_i(C_i) = \sum_{C_j \backslash S_{ij}} \psi_j(C_j)$$

- Claim: if all nodes are fully informed, the clique tree is calibrated. Hence any further message passing will have no effect.

## Out-of-clique queries

- To compute $P(X_q|e)$ where $q$ is not contained with a clique, we look at the smallest subtree that contains $q$, and perform variable elimination on those factors.
- e.g. Consider Markov net $A - B - C - D$ with clique tree

$$C1 : A, B - C2 : B, C - C3 : C, D$$

- We can compute $P(B, D)$ as follows

$$\begin{aligned}
P(B, D) &= \sum_C P(B, C, D) \\
&= \sum_C \frac{\pi_2(B, C)\pi_3(C, D)}{\mu_{2,3}(C)} \\
&= \sum_C P(B|C)P(C, D) \\
&= \text{VarElim}(\{\pi_2, \frac{\pi_3}{\mu_{2,3}}\}, \{B, D\})
\end{aligned}$$

## Viterbi decoding (finding the MPE)

- Let $x^*_{1:n} = \arg\max_{x_{1:N}} P(x_{1:N})$ be (one of the) most probable assignments.
- We can compute $p^* = P(x^*_{1:N})$ using the max product algorithm.
- e.g., $A \rightarrow B$.

$$\begin{aligned}
P(a^*, b^*) &= \max_a \max_b P(a)P(b|a) \\
&= \max_a \max_b \phi_A(a)\phi_B(b, a) \\
&= \max_a \phi_A(a) \underbrace{\max_b \phi_B(b, a)}_{\tau_B(a)} \\
&= \underbrace{\max_a \phi_A(a)\tau_B(a)}_{\tau_A(\emptyset)}
\end{aligned}$$

- We can push max inside products.
- $(\max, \prod)$ and $(\sum, \prod)$ are both commutative semi-rings.

## Viterbi decoding (finding the MPE)

- Max-product gives us $p^* = \max_{x_{1:N}} P(x_{1:N})$, but not $x^*_{1:N} = \arg\max_{x_{1:N}} P(x_{1:N})$.
- To compute the most probable assignment, we need to do max-product followed by a traceback procedure.
- e.g., $A \rightarrow B$.
- We cannot find the most probable value for $A$ unless we know what $B$ we would choose in response.
- So when we compute $\tau_B(a) = \max_b \phi_B(b, a)$, also store

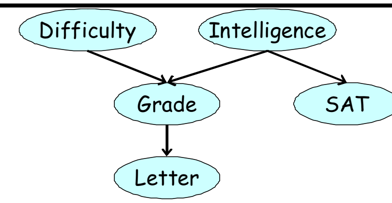
$$\lambda_B(a) = \arg\max_b \phi_B(b, a)$$

- When we compute $\tau_A(\emptyset) = \max_a \phi_A(a)\tau_A(a)$, we also compute

$$a^* = \arg\max_a \phi_A(a)\tau_A(a)$$

- Then traceback: $b^* = \lambda_B(a^*)$.

## More complex example




$$p^* = \max_G \max_L \phi_L(L,G) \max_D \phi_D(D) \max_I \phi_I(I)\phi_G(G,I,D) \max_S \phi_S(I,S)$$

## More complex example

$$\begin{aligned}
p^* &= \max_G \max_L \phi_L(L,G) \max_D \phi_D(D) \max_I \phi_I(I)\phi_G(G,I,D) \underbrace{\max_S \phi_S(I,S)}_{\tau_1(I)} \\
&= \max_G \max_L \phi_L(L,G) \max_D \phi_D(D) \underbrace{\max_I \phi_I(I)\phi_G(G,I,D)\tau_1(I)}_{\tau_2(G,D)} \\
&= \max_G \max_L \phi_L(L,G) \underbrace{\max_D \phi_D(D)\tau_2(G,D)}_{\tau_3(G)} \\
&= \max_G \underbrace{\max_L \phi_L(L,G)\tau_3(G)}_{\tau_4(G)} \\
&= \underbrace{\max_G \tau_4(G)}_{\tau_5(\emptyset)} \\
&= 0.184
\end{aligned}$$

## Traceback

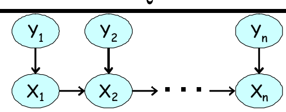

$$\begin{aligned}
p^* &= \max_G \max_L \phi_L(L,G) \max_D \phi_D(D) \max_I \phi_I(I)\phi_G(G,I,D) \underbrace{\max_S \phi_S(I,S)}_{\tau_1(I)} \\
&= \max_G \max_L \phi_L(L,G) \max_D \phi_D(D) \underbrace{\max_I \phi_I(I)\phi_G(G,I,D)\tau_1(I)}_{\tau_2(G,D)} \\
&= \max_G \max_L \phi_L(L,G) \underbrace{\max_D \phi_D(D)\tau_2(G,D)}_{\tau_3(G)} \\
&= \max_G \underbrace{\max_L \phi_L(L,G)\tau_3(G)}_{\tau_4(G)} \\
&= \underbrace{\max_G \tau_4(G)}_{\tau_5(\emptyset)}
\end{aligned}$$


$$\begin{aligned}
\lambda_5(\emptyset) &= \arg\max_g \tau_4(g) = g^* \\
\lambda_4(g) &= \arg\max_l \phi_L(L,G)\tau_3(g), l^* = \lambda_4(g^*) \\
\lambda_3(g) &= \arg\max_d \phi_D(d)\tau_2(G,d), d^* = \lambda_3(g^*) \\
\lambda_2(g,d) &= \arg\max_i \phi_I(i)\phi_G(G,i,D)\tau_1(i), i^* = \lambda_2(g^*, d^*) \\
\lambda_1(i) &= \arg\max_s \phi_S(I,s), s^* = \lambda_1(i^*)
\end{aligned}$$

## Finding K-most probable assignments

- There may be several ($m_1$) assignments with the same highest probability, call them $x_{1:n}^{(1,1)}, \ldots, x_{1:n}^{(1,m_1)}$.
- These can be found by breaking ties in the argmax.
- The second most probable assignment(s) after these, $x_{1:n}^{(2,1)}, \ldots, x_{1:n}^{(2,m_2)}$, must differ in at least one assignment,.
- Hence we assert evidence that the next assignment must be distinct from all $m_1$ MPEs, and re-run Viterbi.
- Project idea: implement this and compare to the loopy belief propagation version to be discussed later.
- This is often used to produce the "N-best list" in speech recognition; these hypotheses are then re-ranked using more sophisticated (discriminative) models.

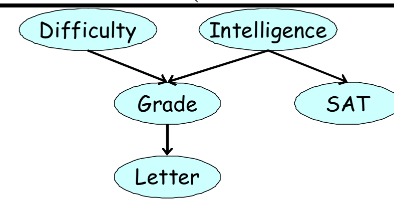
## Constrained elimination orderings may induce large cliques




$$p^* = \max_{Y_1,\ldots,Y_n} \sum_{X_1,\ldots,X_n} P(Y_{1:n}, X_{1:n})$$

- We must eliminate all the $X_i$'s first, which induces a huge clique over all the $Y_i$'s!
- Thm: exact max-marginal inference is NP-hard even in tree-structured graphical models.
- An identical problem arises with decision diagrams, where we must sum out random variables before maxing out action variables.
- An identical problem arises with "hybrid networks", where we must sum out discrete random variables before integrating out Gaussian random variables (ch 11).

## Marginal MAP (max-sum-product)




$$p^* = \max_L \max_S \sum_G \phi_L(L,G) \sum_I \phi_I(I)\phi_S(S,I) \sum_D \phi_G(G,I,D)$$

- We can easily modify the previous algorithms to cope with examples such as this.
- However, max and sum do not commute!

$$\max_X \sum_Y \phi(X,Y) \neq \sum_Y \max_X \phi(X,Y)$$

- Hence we must use a constrained elimination ordering, in which we sum out first, then max out.