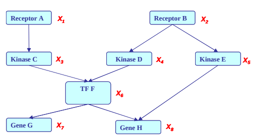
School of Computer Science
Carnegie Mellon

The Belief Propagation (Sum-Product) Algorithm

Probabilistic Graphical Models (10-708)

Lecture 5, October 1, 2007



Eric Xing

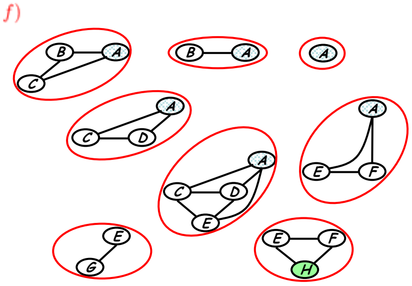
Reading: J-Chap 4

From Elimination to Belief Propagation

- Recall that Induced dependency during marginalization is captured in elimination cliques
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

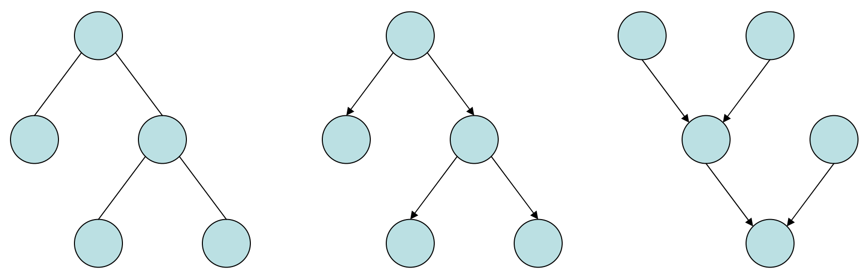
$P(A|H)$

$$
\begin{aligned}
& P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f) \\
\Rightarrow\; & P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)\phi_h(e,f) \\
\Rightarrow\; & P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)\phi_g(e)\phi_h(e,f) \\
\Rightarrow\; & P(a)P(b)P(c|b)P(d|a)P(e|c,d)\phi_f(a,e) \\
\Rightarrow\; & P(a)P(b)P(c|b)P(d|a)\phi_e(a,c,d) \\
\Rightarrow\; & P(a)P(b)P(c|b)\phi_d(a,c) \\
\Rightarrow\; & P(a)P(b)\phi_c(a,b) \\
\Rightarrow\; & P(a)\phi_b(a) \\
\Rightarrow\; & \phi(a)
\end{aligned}
$$



  - Can this lead to an generic inference algorithm?

Tree GMs



Undirected tree: a unique path between any pair of nodes

Directed tree: all nodes except the root have exactly one parent

Poly tree: can have multiple parents

We will come back to this later

Equivalence of directed and undirected trees

- Any undirected tree can be converted to a directed tree by choosing a root node and directing all edges away from it
- A directed tree and the corresponding undirected tree make the same conditional independence assertions
- Parameterizations are essentially the same.
  - Undirected tree: $$p(x) = \frac{1}{Z}\left(\prod_{i\in V}\psi(x_i)\prod_{(i,j)\in E}\psi(x_i,x_j)\right)$$
  - Directed tree: $$p(x) = p(x_r)\prod_{(i,j)\in E}p(x_j|x_i)$$
  - Equivalence: $$\psi(x_r) = p(x_r);\quad \psi(x_i,x_j) = p(x_j|x_i);$$ $$Z = 1,\quad \psi(x_i) = 1$$
  - Evidence:? $\delta(x_{\varepsilon}, \hat{x}_e)$

From elimination to message passing

- Recall ELIMINATION algorithm:
  - Choose an ordering $\mathcal{Z}$ in which query node f is the final node
  - Place all potentials on an active list
  - Eliminate node i by removing all potentials containing i, take sum/product over x_i.
  - Place the resultant factor back on the list
- For a TREE graph:
  - Choose query node f as the root of the tree
  - View tree as a directed tree with edges pointing towards from f
  - Elimination ordering based on depth-first traversal
  - Elimination of each node can be considered as message-passing (or Belief Propagation) directly along tree branches, rather than on some transformed graphs
  - → thus, we can use the tree itself as a data-structure to do general inference!!

The elimination algorithm

Procedure Initialize (G, Z)

1. Let $Z_1, \ldots, Z_k$ be an ordering of Z such that $Z_i \prec Z_j$ iff $i < j$
2. Initialize $\mathcal{F}$ with the full the set of factors

Procedure Evidence $(\mathbf{E})$

1. **for** each $i \in I_E$,
   $\mathcal{F} = \mathcal{F} \cup \delta(E_i, e_i)$

Procedure Sum-Product-Variable-Elimination $(\mathcal{F}, Z, \prec)$

1. **for** $i = 1, \ldots, k$
   $\mathcal{F} \leftarrow$ Sum-Product-Eliminate-Var$(\mathcal{F}, Z_i)$
2. $\phi^* \leftarrow \prod_{\phi \in \mathcal{F}} \phi$
3. **return** ϕ^*
4. Normalization (ϕ^*)

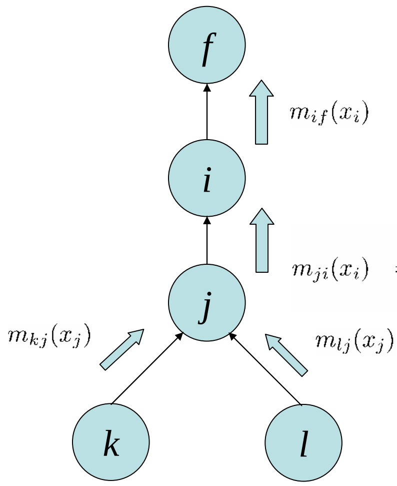
Procedure Normalization (ϕ^*)

1. $P(X|\mathbf{E}) = \phi^*(X) / \sum_x \phi^*(X)$

Procedure Sum-Product-Eliminate-Var (
$\mathcal{F}$, // Set of factors
Z // Variable to be eliminated
)

1. $\mathcal{F}' \leftarrow \{\phi \in \mathcal{F} : Z \in Scope[\phi]\}$
2. $\mathcal{F}'' \leftarrow \mathcal{F} - \mathcal{F}'$
3. $\psi \leftarrow \prod_{\phi \in \mathcal{F}'} \phi$
4. $\tau \leftarrow \sum_Z \psi$
5. **return** $\mathcal{F}'' \cup \{\tau\}$

Message passing for trees



Let $m_{ij}(x_i)$ denote the factor resulting from eliminating variables from bellow up to i, which is a function of x_i:

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

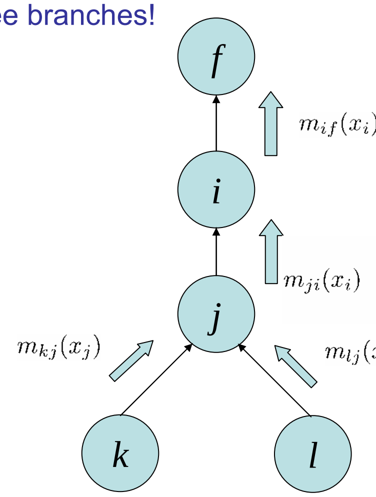
This is reminiscent of a ***message*** sent from j to i.

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

$$p(x_f) \propto \psi(x_f) \prod_{e \in N(f)} m_{ef}(x_f)$$

$m_{ij}(x_i)$ represents a "belief" of x_i from x_j!

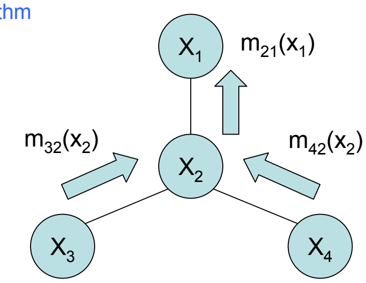
- Elimination on trees is equivalent to message passing along tree branches!



$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

The message passing protocol:

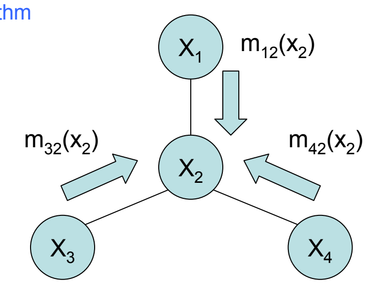
- A node can send a message to its neighbors when (and only when) it has received messages from all its ***other*** neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



Computing $P(X_1)$

The message passing protocol:

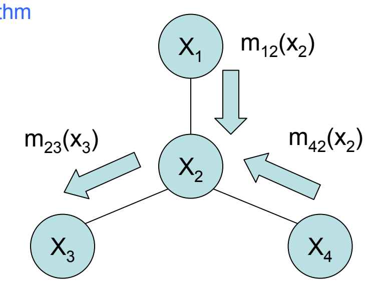
- A node can send a message to its neighbors when (and only when) it has received messages from all its ***other*** neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



Computing $P(X_2)$

The message passing protocol:

- A node can send a message to its neighbors when (and only when) it has received messages from all its ***other*** neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



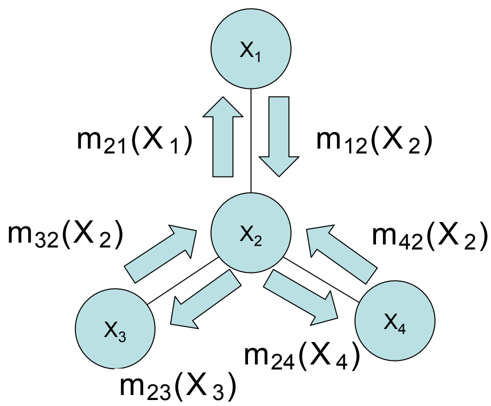
Computing $P(X_3)$

Computing node marginals

- Naïve approach:
  - Complexity: NC
    - N is the number of nodes
    - C is the complexity of a complete message passing
- Alternative dynamic programming approach
  - 2-Pass algorithm (next slide →)
  - Complexity: 2C!

The message passing protocol:

- A two-pass algorithm:



Belief Propagation (SP-algorithm): Sequential implementation

SUM-PRODUCT($\mathcal{T}$, E)
  EVIDENCE(E)
  f = CHOOSEROOT($\mathcal{V}$)
  for $e \in \mathcal{N}(f)$
    COLLECT(f, e)
  for $e \in \mathcal{N}(f)$
    DISTRIBUTE(f, e)
  for $i \in \mathcal{V}$
    COMPUTEMARGINAL(i)

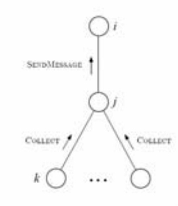
EVIDENCE(E)
  for $i \in E$
    $\psi^E(x_i) = \psi(x_i)\delta(x_i, \bar{x}_i)$
  for $i \notin E$
    $\psi^E(x_i) = \psi(x_i)$

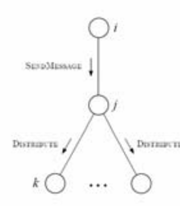
COLLECT(i, j)
  for $k \in \mathcal{N}(j)\backslash i$
    COLLECT(j, k)
  SENDMESSAGE(j, i)

DISTRIBUTE(i, j)
  SENDMESSAGE(i, j)
  for $k \in \mathcal{N}(j)\backslash i$
    DISTRIBUTE(j, k)

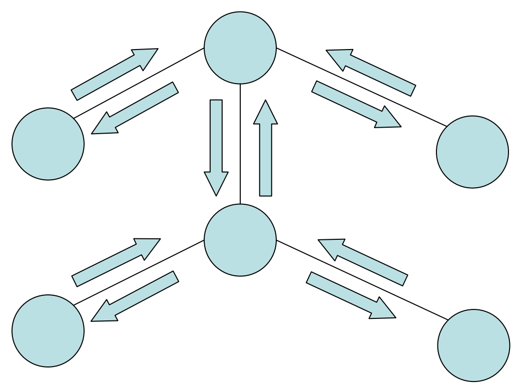
SENDMESSAGE(j, i)
  $m_{ji}(x_i) = \sum_{x_j}(\psi^E(x_j)\psi(x_i, x_j) \prod_{k \in \mathcal{N}(j)\backslash i} m_{kj}(x_j))$

COMPUTEMARGINAL(i)
  $p(x_i) \propto \psi^E(x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(x_i)$





Belief Propagation (SP-algorithm): Parallel synchronous implementation



- For a node of degree d, whenever messages have arrived on any subset of d-1 node, compute the message for the remaining edge and send!
  - A pair of messages have been computed for each edge, one for each direction
  - All incoming messages are eventually computed for each node

Correctness of BP on tree

- Collollary: the synchronous implementation is "non-blocking"
- Thm: The Message Passage Guarantees obtaining all marginals in the tree

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

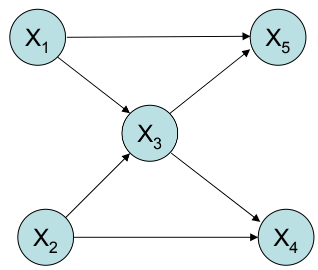
$\psi(x_j,$

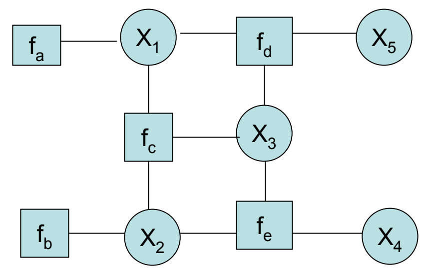
- What about non-tree?

$p(x_1) \propto \phi(x_1)\psi(x_1, x_4) m_{4\to1}$
$\psi(x_1, x_2) m_{2\to1}(x_1)$

Another view of SP: Factor Graph

- Example 1





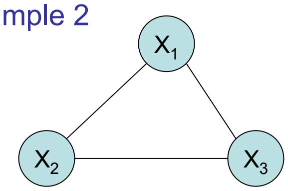
$$P(X_1)\ \ P(X_2)\ \ P(X_3|X_1,X_2)\ \ P(X_5|X_1,X_3)\ \ P(X_4|X_2,X_3)$$

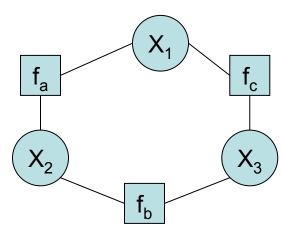
$$\Downarrow$$

$$f_a(X_1)\ f_b(X_2)\ \ f_c(X_3,X_1,X_2)\ \ f_d(X_5,X_1,X_3)\ \ f_e(X_4,X_2,X_3)$$

Factor Graphs

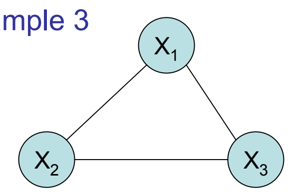
- Example 2

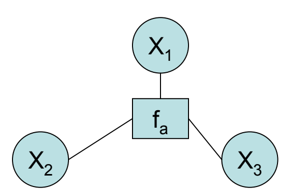




$\psi(x_1,x_2,x_3) = f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_3,x_1)$

- Example 3

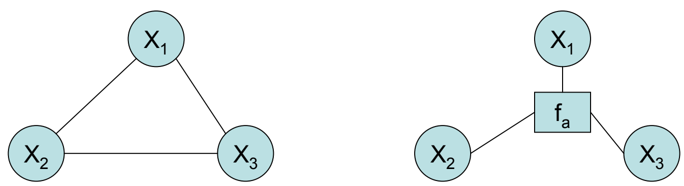




$\psi(x_1,x_2,x_3) = f_a(x_1,x_2,x_3)$

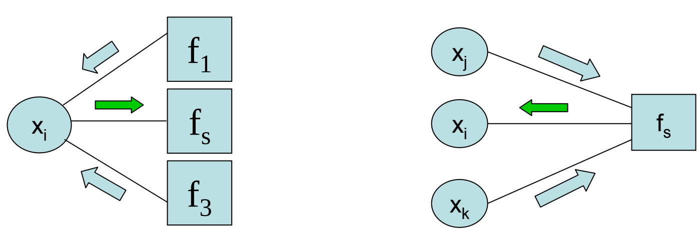
Factor Tree

- A Factor graph is a Factor Tree if the undirected graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree



Message Passing on a Factor Tree

- Two kinds of messages
  1. ν: from variables to factors
  2. μ: from factors to variables

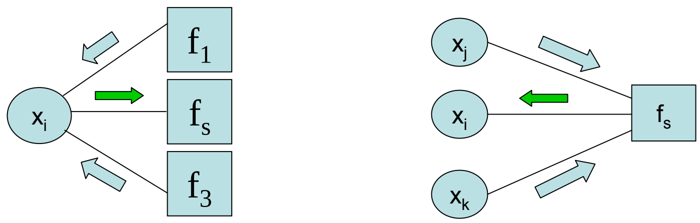


$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i)\backslash s} \mu_{ti}(x_i)$$

$$\mu_{si}(x_i) = \sum_{x_{\mathcal{N}(s)\backslash i}} \left(f_s(x_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s)\backslash i} \nu_{js}(x_j) \right)$$

Message Passing on a Factor Tree, con'd

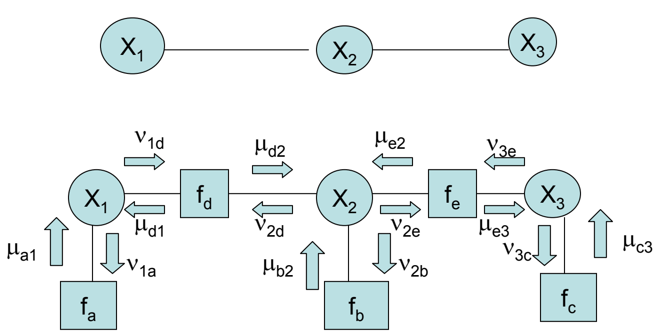
- Message passing protocol:
  - A node can send a message to a neighboring node only when it has received messages from all its ***other*** neighbors
- Marginal probability of nodes:



$P(x_i) \propto \prod_{s \in N(i)} \mu_{si}(x_i)$

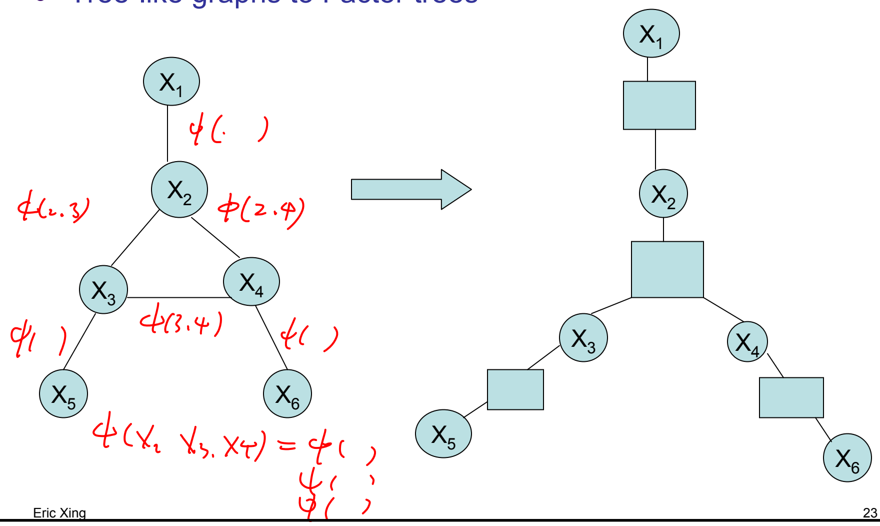
$\propto \nu_{is}(x_i)\mu_{si}(x_i)$

BP on a Factor Tree

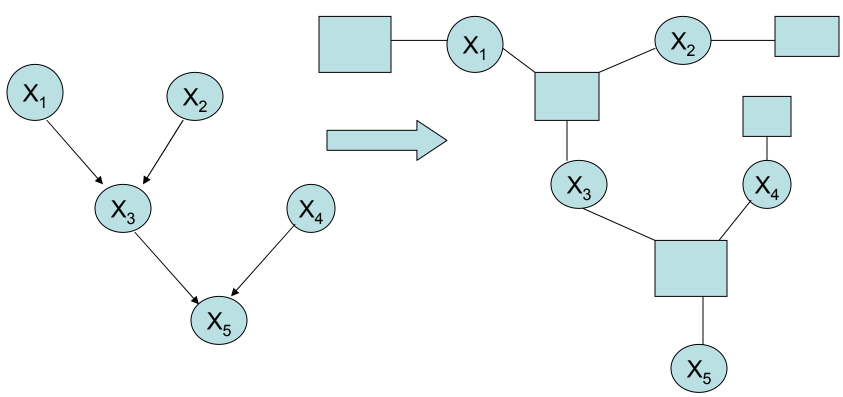


Why factor graph?

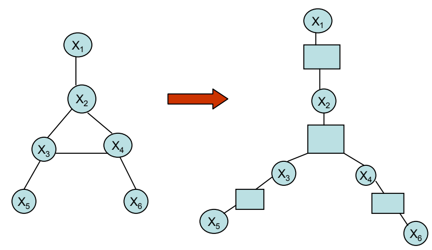
- Tree-like graphs to Factor trees

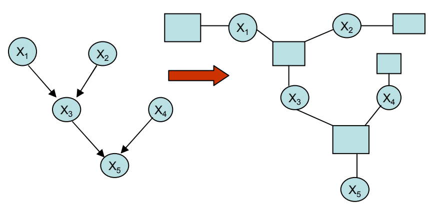


Poly-trees to Factor trees



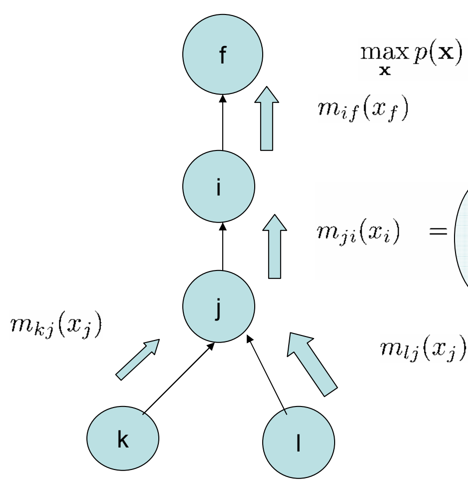
Why factor graph?





- Because FG turns tree-like graphs to factor trees,
- and trees are a data-structure that guarantees correctness of BP !

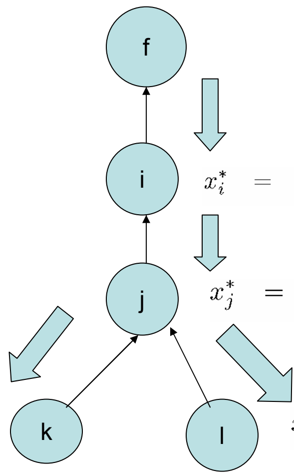
Max-product algorithm: computing MAP probabilities



$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_f}(\psi(x_f) m_{if}(x_f))$$

$$m_{ji}(x_i) = \max_{x_j}\left(\psi(x_j)\psi(x_i,x_j)\prod_{k\in N(j)-i} m_{kj}(x_j)\right)$$

Max-product algorithm:
computing MAP configurations using a final bookkeeping backward pass



$$x_f^* = \arg\max_{x_f} (\psi(x_f) m_{if}(x_f))$$

$$x_i^* = \arg\max_{x_i} \left(\psi(x_i)\psi(x_f^*, x_i) m_{ji}(x_i)\right)$$

$$x_j^* = \arg\max_{x_j} \left(\psi(x_j)\psi(x_i^*, x_j) m_{kj}(x_j) m_{lj}(x_j)\right)$$

$$x_l^* = \arg\max_{x_l} \left(\psi(x_l)\psi(x_l, x_j^*)\right)$$

$$x_k^* = \arg\max_{x_k} \left(\psi(x_k)\psi(x_k, x_j^*)\right)$$

Summary

- Sum-Product algorithm computes singleton marginal probabilities on:
  - Trees
  - Tree-like graphs
  - Poly-trees
- *Maximum a posteriori* configurations can be computed by replacing sum with max in the sum-product algorithm
  - Extra bookkeeping required

Inference on general GM

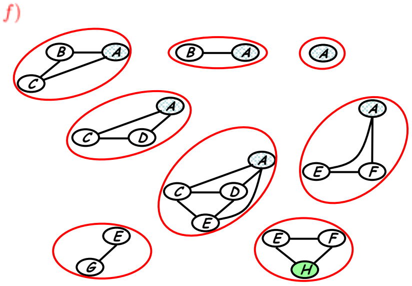
- Now, what if the GM is not a tree-like graph?
- Can we still directly run message
  message-passing protocol along its edges?
- For non-trees, we do not have the guarantee that message-passing will be consistent!
- Then what?
  - Construct a graph data-structure from P that has a tree structure, and run message-passing on it!

→ Junction tree algorithm

Elimination Clique

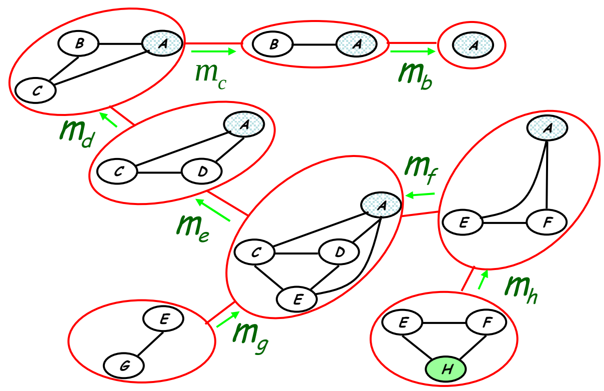
- Recall that Induced dependency during marginalization is captured in elimination cliques
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)\phi_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)\phi_g(e)\phi_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)\phi_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)\phi_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|b)\phi_d(a,c)$$
$$\Rightarrow P(a)P(b)\phi_c(a,b)$$
$$\Rightarrow P(a)\phi_b(a)$$
$$\Rightarrow \phi(a)$$



  - Can this lead to an generic inference algorithm?

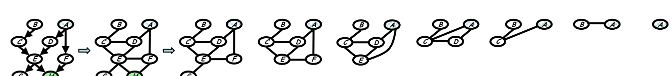
A Clique Tree



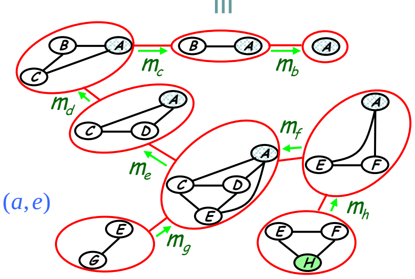
$$m_e(a,c,d) = \sum_e p(e \mid c,d) m_g(e) m_f(a,e)$$

From Elimination to Message Passing

- Elimination $\equiv$ message passing on a **clique tree**



|||

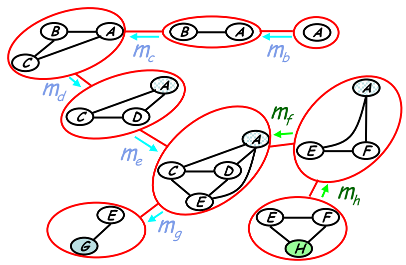


$$m_e(a,c,d) = \sum_e p(e \mid c,d) m_g(e) m_f(a,e)$$

- Messages can be reused

From Elimination to Message Passing

- Elimination ≡ message passing on a **clique tree**
  - **Another query ...**



- Messages m_f and m_h are reused, others need to be recomputed