PETUUM

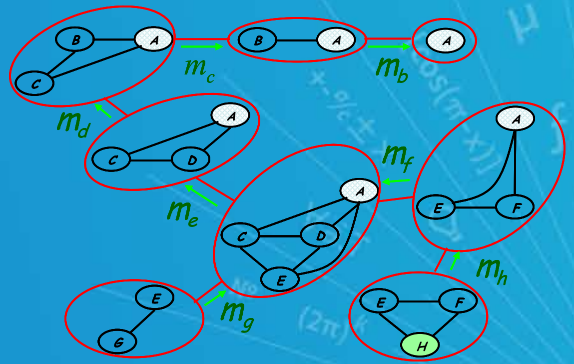
Carnegie Mellon University
School of Computer Science

Probabilistic Graphical Models

Exact Inference

Eric Xing

Lecture 4, January 27, 2020



Reading: see class homepage

Probabilistic Inference and Learning

- We now have compact representations of probability distributions: Graphical Models
- A GM M describes a unique probability distribution P
- Typical tasks:
  - Task 1: How do we answer queries about P_M, *e.g.*, $P_M(X|Y)$?
    - We use inference as a name for the process of computing answers to such queries
  - Task 2: How do we estimate a plausible model M from data D?
    1. We use learning as a name for the process of obtaining point estimate of M.
    2. But for *Bayesian*, they seek $p(M|D)$, which is actually an inference problem.
    3. When not all variables are observable, even computing point estimate of M need to do inference to impute the *missing data*.

Query 1: Likelihood

- Most of the queries one may ask involve evidence
  - Evidence $\mathbf{e}$ is an assignment of values to a set $\mathbf{E}$ variables in the domain
  - Without loss of generality $\mathbf{E} = \{ X_{k+1}, \ldots, X_n \}$
- Simplest query: compute probability of evidence

$$P(\mathbf{e}) = \sum_{x_1} \cdots \sum_{x_k} P(x_1, \ldots, x_k, \mathbf{e})$$

  - this is often referred to as computing the likelihood of $\mathbf{e}$

Query 2: Conditional Probability

- Often we are interested in the conditional probability distribution of a variable given the evidence

$$P(X \mid \mathbf{e}) = \frac{P(X,\mathbf{e})}{P(\mathbf{e})} = \frac{P(X,\mathbf{e})}{\sum_x P(X=x,\mathbf{e})}$$

  - this is the *a posteriori* belief in X, given evidence $\mathbf{e}$

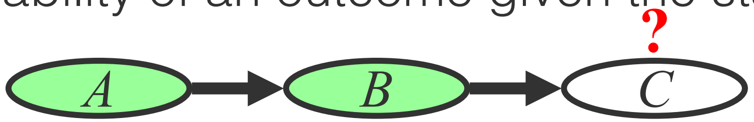
- We usually query a subset $\mathbf{Y}$ of all domain variables $\mathbf{X}=\{\mathbf{Y},\mathbf{Z}\}$ and "don't care" about the remaining, $\mathbf{Z}$:

$$P(\mathbf{Y} \mid e) = \sum_{\mathbf{z}} P(\mathbf{Y},\mathbf{Z}=\mathbf{z} \mid \mathbf{e})$$

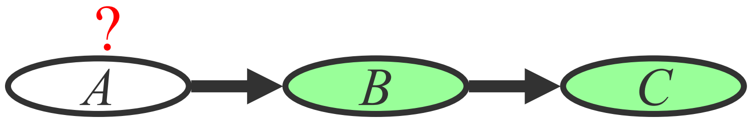
  - the process of summing out the "don't care" variables z is called marginalization, and the resulting $P(y|e)$ is called a marginal prob.

Applications of a posteriori Belief

- **Prediction**: what is the probability of an outcome given the starting condition



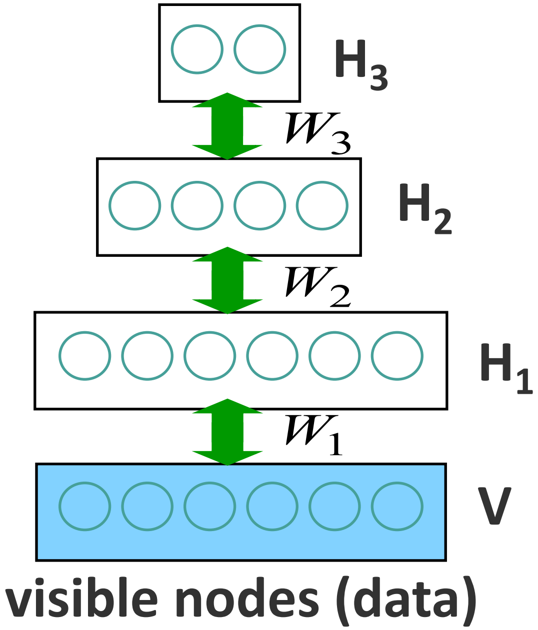
  - the query node is a descendent of the evidence
- **Diagnosis**: what is the probability of disease/fault given symptoms

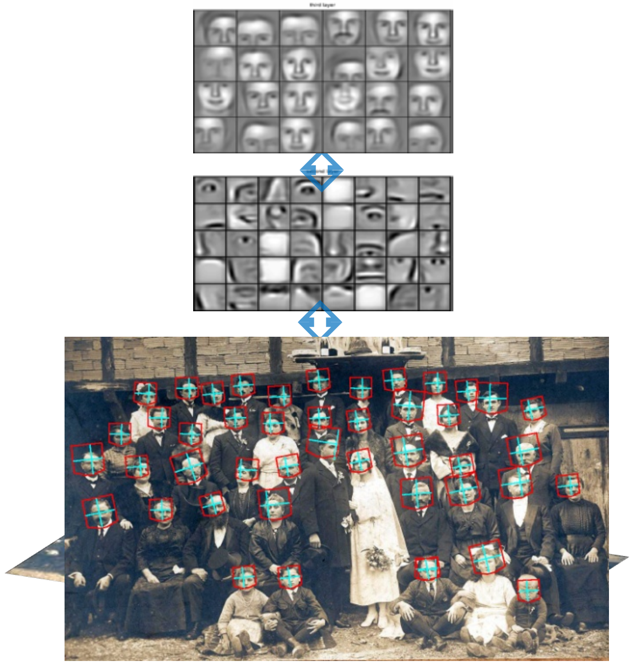


  - the query node an ancestor of the evidence
- **Learning** under partial observation
  - fill in the unobserved values under an "EM" setting (more later)
- The directionality of information flow between variables is not restricted by the directionality of the edges in a GM
  - probabilistic inference can combine evidence form all parts of the network

Example: Deep Belief Network

- Deep Belief Network (DBN) [Hinton et al., 2006]
  - Generative model or RBM with multiple hidden layers
  - Successful applications
    - Recognizing handwritten digits
    - Learning motion capture data
    - Collaborative filtering





Query 3: Most Probable Assignment

- In this query we want to find the most probable joint assignment (MPA) for *some* variables of interest

- Such reasoning is usually performed under some given evidence $\mathbf{e}$, and ignoring (the values of) other variables z :

$$\mathrm{MPA}(\mathbf{Y} \mid \mathbf{e}) = \arg\max_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y} \mid \mathbf{e}) = \arg\max_{\mathbf{y}\in\mathcal{Y}} \sum_{\mathbf{z}} P(\mathbf{y}, \mathbf{z} \mid \mathbf{e})$$

  - this is the maximum *a posteriori* configuration of y.

Applications of MPA

- Classification
  - find most likely label, given the evidence
- Explanation
  - what is the most likely scenario, given the evidence

Cautionary note:

- The MPA of a variable depends on its "context"---the set of variables been jointly queried
- Example:
  - MPA of Y_1?
  - MPA of (Y_1, Y_2)?

y_1	y_2	$P(y_1,y_2)$
0	0	0.35
0	1	0.05
1	0	0.3
1	1	0.3

Complexity of Inference

- Thm:

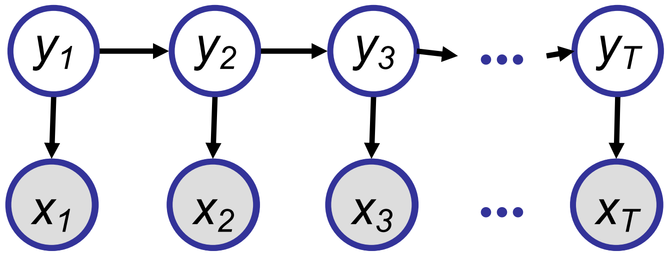
  Computing $P(\mathrm{X} = \mathrm{x} \mid \mathrm{e})$ in a GM is NP-hard

- Hardness does not mean we cannot solve inference
  - It implies that we cannot find a general procedure that works efficiently for arbitrary GMs
  - For particular families of GMs, we can have provably efficient procedures

Approaches to inference

- Exact inference algorithms
  - The elimination algorithm
  - Message-passing algorithm (sum-product, belief propagation)
  - The junction tree algorithms
- Approximate inference techniques
  - Stochastic simulation / sampling methods
  - Markov chain Monte Carlo methods
  - Variational algorithms

Variable Elimination on Hidden Markov Model

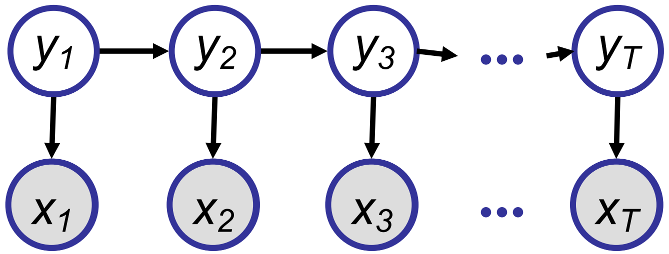


$$p(\mathbf{x}, \mathbf{y}) = p(x_1 \ldots\ldots x_T, y_1, \ldots\ldots, y_T)$$
$$= p(y_1)\, p(x_1 \mid y_1)\, p(y_2 \mid y_1)\, p(x_2 \mid y_2) \ldots p(y_T \mid y_{T-1})\, p(x_T \mid y_T)$$

Conditional probability:

$$p(y_i|x_1,\ldots,x_T) = \sum_{y_1}\cdots\sum_{y_{i-1}}\sum_{y_{i+1}}\cdots\sum_{y_T} p(y_i,\ldots,y_T,x_1,\ldots,x_T)$$
$$= \sum_{y_1}\cdots\sum_{y_{i-1}}\sum_{y_{i+1}}\cdots\sum_{y_T} p(y_1)p(x_1|y_1)\ldots p(y_T|y_{T-1})p(x_T|y_T)$$

Variable Elimination on Hidden Markov Model



Conditional probability:

$$
\begin{aligned}
p(y_i|x_1,\ldots,x_T) &= \sum_{y_1}\cdots\sum_{y_{i-1}}\sum_{y_{i+1}}\cdots\sum_{y_T} p(y_i,\ldots,y_T,x_1,\ldots,x_T) \\
&= \sum_{y_1}\cdots\sum_{y_{i-1}}\sum_{y_{i+1}}\cdots\sum_{y_T} p(y_1)p(x_1|y_1)\ldots p(y_T|y_{T-1})p(x_T|y_T)
\end{aligned}
$$

The Sum-Product Operation

- In general, we can view the task at hand as that of computing the value of an expression of the form:

$$\sum_{\mathbf{z}} \prod_{\phi \in \mathcal{F}} \phi$$

where F is a set of factors

- We call this task the *sum-product* inference task.

Inference on General GM via Variable Elimination

- General idea:
- Write query in the form

$$P(X_1, \mathbf{e}) = \sum_{x_n} \cdots \sum_{x_3} \sum_{x_2} \prod_i P(x_i \mid pa_i)$$

  - this suggests an "elimination order" of latent variables to be marginalized
- Iteratively
  - Move all irrelevant terms outside of innermost sum
  - Perform innermost sum, getting a new term
  - Insert the new term into the product
- wrap-up

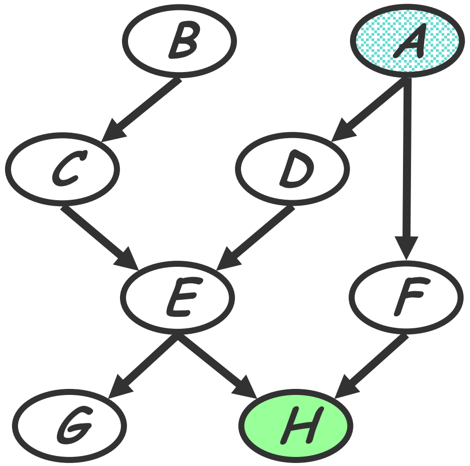
$$P(X_1 \mid \mathbf{e}) = \frac{\phi(X_1, \mathbf{e})}{\sum_{x_1} \phi(X_1, \mathbf{e})}$$

Variable Elimination on a general BN

- Query: *P(A | h)*
  - Need to eliminate: *B,C,D,E,F,G,H*
- Initial factors:

$$P(a)P(b)P(c\,|\,b)P(d\,|\,a)P(e\,|\,c,d)P(f\,|\,a)P(g\,|\,e)P(h\,|\,e,f)$$

- Choose an elimination order: *H,G,F,E,D,C,B*

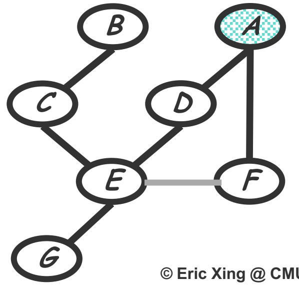


- Step 1:
  - **Conditioning** (fix the evidence node (i.e., *h*) on its observed value (i.e., $\tilde{h}$)):

$$m_h(e,f) = p(h = \tilde{h}\,|\,e,f)$$

  - This step is isomorphic to a marginalization step:

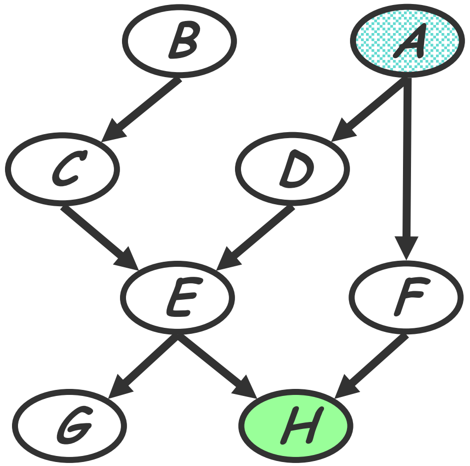
$$m_h(e,f) = \sum_h p(h\,|\,e,f)\,\delta(h = \tilde{h})$$



Variable Elimination on a general BN

- Query: $P(B \mid h)$
  - Need to eliminate: B,C,D,E,F,G
- Initial factors:

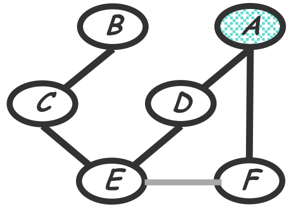
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)\underline{P(g|e)}m_h(e,f)$$



- Step 2: Eliminate G
  - compute

$$m_g(e) = \sum_g p(g|e) = 1$$

$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_g(e)m_h(e,f)$$
$$= P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)\underline{m_h(e,f)}$$



Variable Elimination on a general BN

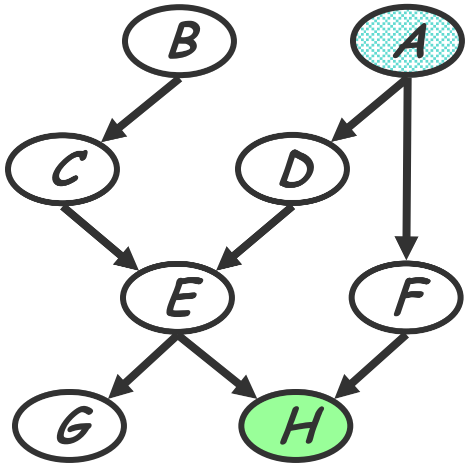
- Query: $P(B \mid h)$
  - Need to eliminate: B,C,D,E,F

- Initial factors:

$$P(a)P(b)P(c \mid b)P(d \mid a)P(e \mid c,d)P(f \mid a)P(g \mid e)P(h \mid e,f)$$

$$\Rightarrow P(a)P(b)P(c \mid b)P(d \mid a)P(e \mid c,d)P(f \mid a)P(g \mid e)m_h(e,f)$$

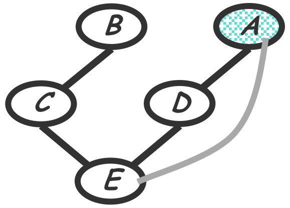
$$\Rightarrow P(a)P(b)P(c \mid b)P(d \mid a)P(e \mid c,d)\underline{P(f \mid a)m_h(e,f)}$$



- Step 3: Eliminate F
  - compute

$$m_f(e,a) = \sum_f p(f \mid a)m_h(e,f)$$

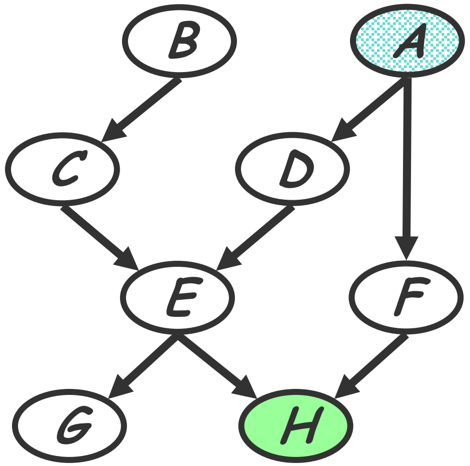
$$\Rightarrow P(a)P(b)P(c \mid b)P(d \mid a)P(e \mid c,d)\underline{m_f(a,e)}$$



Variable Elimination on a general BN

- Query: $P(B \mid h)$
  - Need to eliminate: B,C,D,E
- Initial factors:

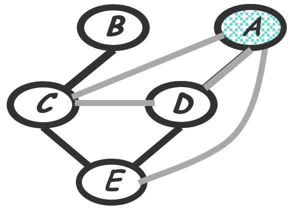
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)\underline{P(e|c,d)m_f(a,e)}$$



- Step 4: Eliminate E
  - compute

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

$$\Rightarrow P(a)P(b)P(c|b)P(d|a)\underline{m_e(a,c,d)}$$

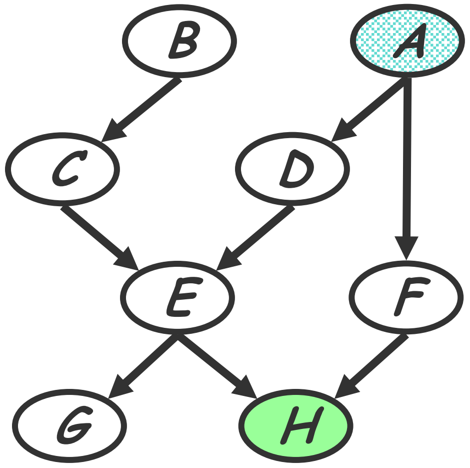


Variable Elimination on a general BN

- Query: $P(B\,|\,h)$
  - Need to eliminate: B,C,D

- Initial factors:

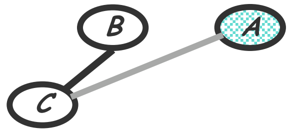
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|b)\underline{P(d|a)m_e(a,c,d)}$$



- Step 5: Eliminate D
  - compute

$$m_d(a,c)=\sum_d p(d|a)m_e(a,c,d)$$

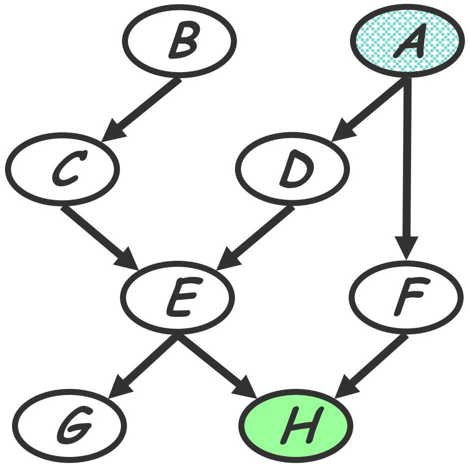
$$\Rightarrow P(a)P(b)P(c|d)\underline{m_d(a,c)}$$



Variable Elimination on a general BN

- Query: $P(B \mid h)$
  - Need to eliminate: B, C
- Initial factors:

$$P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)P(f \mid a)P(g \mid e)P(h \mid e,f)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)P(f \mid a)P(g \mid e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)P(f \mid a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)\underline{P(c \mid d)m_d(a,c)}$$



- Step 6: Eliminate C
  - compute

$$m_c(a,b) = \sum_c p(c \mid b)m_d(a,c)$$

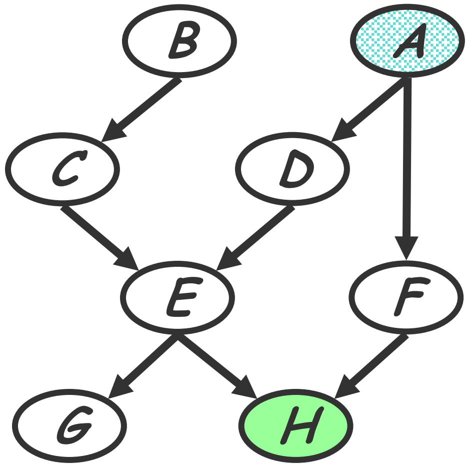
$$\Rightarrow P(a)P(b)\underline{P(c \mid d)m_d(a,c)}$$

Variable Elimination on a general BN

- Query: $P(B \mid h)$
  - Need to eliminate: B

- Initial factors:

$$P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)P(f \mid a)P(g \mid e)P(h \mid e,f)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)P(f \mid a)P(g \mid e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)P(f \mid a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)P(e \mid c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c \mid d)P(d \mid a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c \mid d)m_d(a,c)$$
$$\Rightarrow P(a)\underline{P(b)m_c(a,b)}$$



- Step 7: Eliminate B
  - compute

$$m_b(a) = \sum_b p(b)m_c(a,b)$$

$$\Rightarrow P(a)\underline{m_b(a)}$$

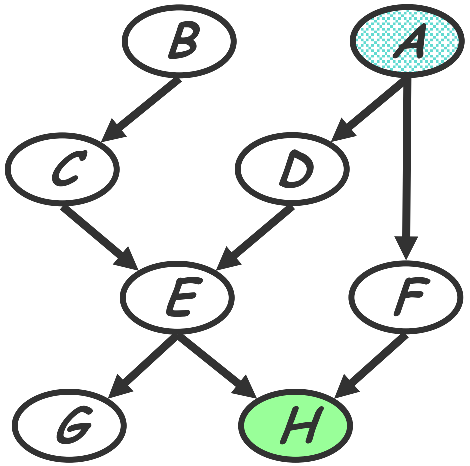
Variable Elimination on a general BN

- Query: $P(B\,|\,h)$
  - Need to eliminate: B
- Initial factors:

$$P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$
$$\Rightarrow P(a)P(b)m_c(a,b)$$
$$\Rightarrow P(a)m_b(a)$$

- Step 8: Wrap-up

$$p(a,\tilde{h}) = p(a)m_b(a), \quad p(\tilde{h}) = \sum_a p(a)m_b(a)$$
$$\Rightarrow P(a|\tilde{h}) = \frac{p(a)m_b(a)}{\sum_a p(a)m_b(a)}$$



Outcome of elimination

- Let $\mathbf{X}$ be some set of variables,
  let $\mathcal{F}$ be a set of factors such that for each $\phi \in \mathcal{F}$, $Scope[\phi] \in \mathbf{X}$,
  let $\mathbf{Y} \subset \mathbf{X}$ be a set of query variables,
  and let $Z = \mathbf{X} - \mathbf{Y}$ be the variable to be eliminated

- The result of eliminating the variable Z is a factor

$$\tau(\mathbf{Y}) = \sum_{\mathbf{z}} \prod_{\phi \in \mathcal{F}} \phi$$

  - This factor does not necessarily correspond to any probability or conditional probability in this network. (example forthcoming)

Dealing with evidence

- Conditioning as a Sum-Product Operation
  - The evidence potential:

$$\delta(E_i, \bar{e}_i) = \begin{cases} 1 & \text{if } E_i \equiv \bar{e}_i \\ 0 & \text{if } E_i \neq \bar{e}_i \end{cases}$$

  - Total evidence potential:

$$\delta(\mathbf{E}, \bar{\mathbf{e}}) = \prod_{i \in I_{\mathbf{E}}} \delta(E_i, \bar{e}_i)$$

  - Introducing evidence --- restricted factors:

$$\tau(\mathbf{Y}, \bar{\mathbf{e}}) = \sum_{\mathbf{z}, \mathbf{e}} \prod_{\phi \in \mathcal{F}} \phi \times \delta(\mathbf{E}, \bar{\mathbf{e}})$$

The elimination algorithm

Procedure Elimination (

G, // the GM

$\mathbf{E}$, // evidence

Z, // Set of variables to be eliminated

X, // query variable(s)

)

1. Initialize (G)
2. Evidence ($\mathbf{E}$)
3. Sum-Product-Elimination (F, Z, $\prec$)
4. Normalization (F)

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 F with the full the set of factors

Procedure Evidence $(\mathbf{E})$

1. for each $i \in I_E$,

   $F = F \cup \delta(E_i, e_i)$

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

1. for $i = 1, \ldots, k$

   $F \leftarrow$ Sum-Product-Eliminate-Var(F, Z_i)
2. $\phi^* \leftarrow \prod_{\phi \in F} \phi$
3. return ϕ^*
4. Normalization (ϕ^*)

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 F with the full the set of factors

Procedure Evidence $(\mathbf{E})$

1. for each $i \in I_E$,

   $F = F \cup \delta(E_i, e_i)$

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

1. for $i = 1, \ldots, k$

   $F \leftarrow$ Sum-Product-Eliminate-Var(F, Z_i)
2. $\phi^* \leftarrow \prod_{\phi \in F} \phi$
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 (

F, // Set of factors

Z // Variable to be eliminated

)

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

Complexity of variable elimination

- Suppose in one elimination step we compute

$$m_x(y_1, \ldots, y_k) = \sum_x m'_x(x, y_1, \ldots, y_k)$$

$$m'_x(x, y_1, \ldots, y_k) = \prod_{i=1}^{k} m_i(x, \mathbf{y}_{c_i})$$

This requires

$$k \bullet |\mathrm{Val}(X)| \bullet \prod_i |\mathrm{Val}(\mathbf{Y}_{C_i})| \quad \text{multiplications}$$

- For each value for $x, y_1, \ldots, y_k$, we do k multiplications

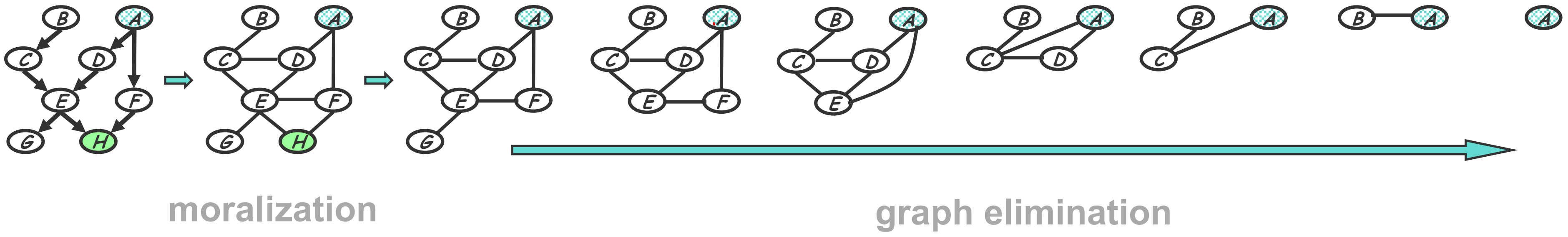
$$|\mathrm{Val}(X)| \bullet \prod_i |\mathrm{Val}(\mathbf{Y}_{C_i})| \quad \text{additions}$$

- For each value of $y_1, \ldots, y_k$, we do $|Val(X)|$ additions

Complexity is exponential in number of variables
in the intermediate factor

Understanding Variable Elimination

- A graph elimination algorithm

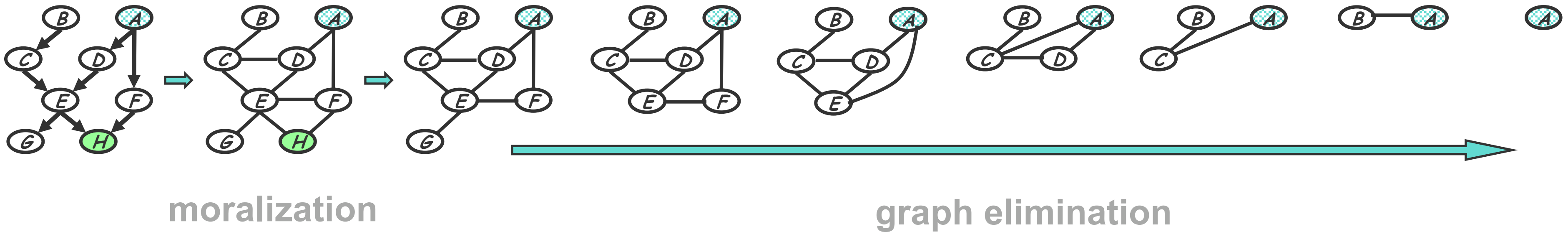


Graph elimination

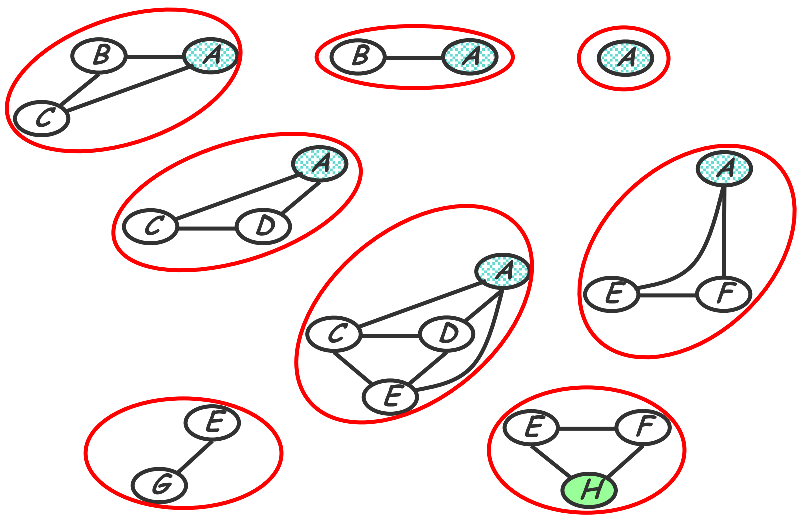
- Begin with the undirected GM or moralized BN
- Graph $G(V, E)$ and elimination ordering I
- Eliminate next node in the ordering I
  - Removing the node from the graph
  - Connecting the remaining neighbors of the nodes
- The reconstituted graph $G'(V, E')$
  - Retain the edges that were created during the elimination procedure
  - The graph-theoretic property: the factors resulted during variable elimination are captured by recording the elimination clique

Understanding Variable Elimination

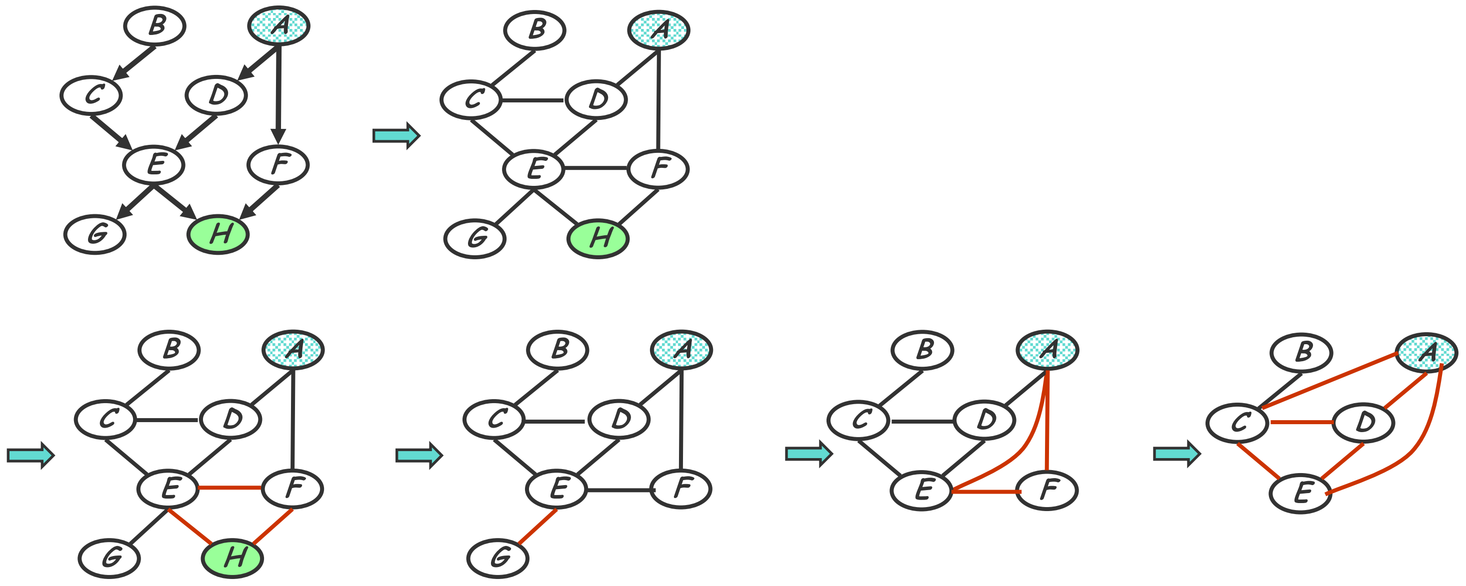
- A graph elimination algorithm



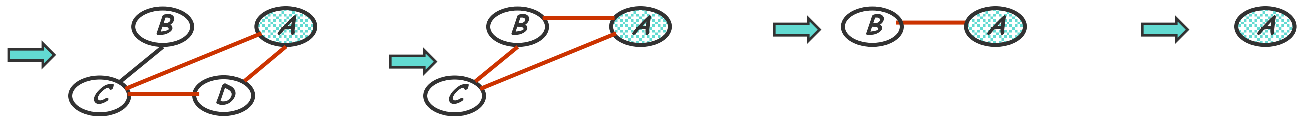
- Intermediate terms correspond to the cliques resulted from elimination



Elimination Cliques



$m_h(e,f)$ $m_g(e)$ $m_f(e,a)$ $m_e(a,c,d)$

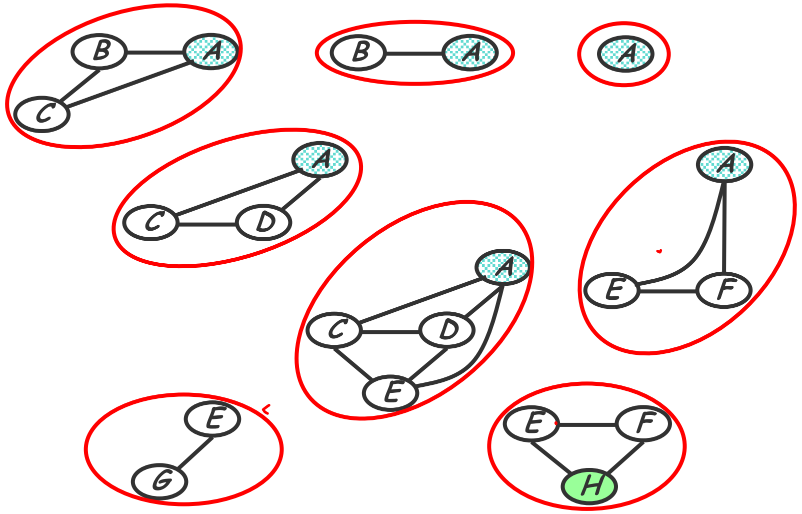


$m_d(a,c)$ $m_c(a,b)$ $m_b(a)$

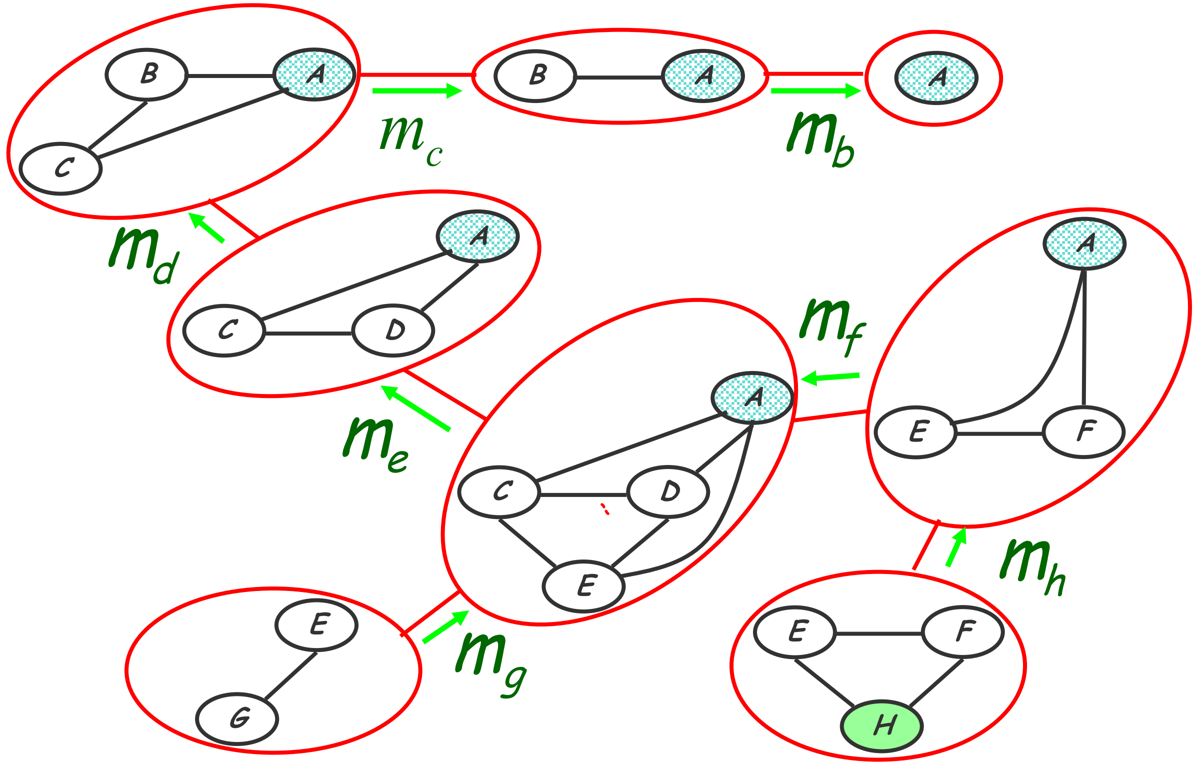
Graph elimination and marginalization

- Induced dependency during marginalization vs. elimination clique
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

$$P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)P(e|c,d)m_f(a,e)$$
$$\Rightarrow P(a)P(b)P(c|d)P(d|a)m_e(a,c,d)$$
$$\Rightarrow P(a)P(b)P(c|d)m_d(a,c)$$
$$\Rightarrow P(a)P(b)m_c(a,b)$$
$$\Rightarrow P(a)m_b(a)$$



A clique tree



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

Complexity

- The overall complexity is determined by the number of the largest elimination clique
  - What is the largest elimination clique? – a pure graph theoretic question
  - Tree-width k: one less than the smallest achievable value of the cardinality of the largest elimination clique, ranging over all possible elimination ordering
  - "good" elimination orderings lead to small cliques and hence reduce complexity (what will happen if we eliminate "e" first in the above graph?)
  - Find the best elimination ordering of a graph --- NP-hard

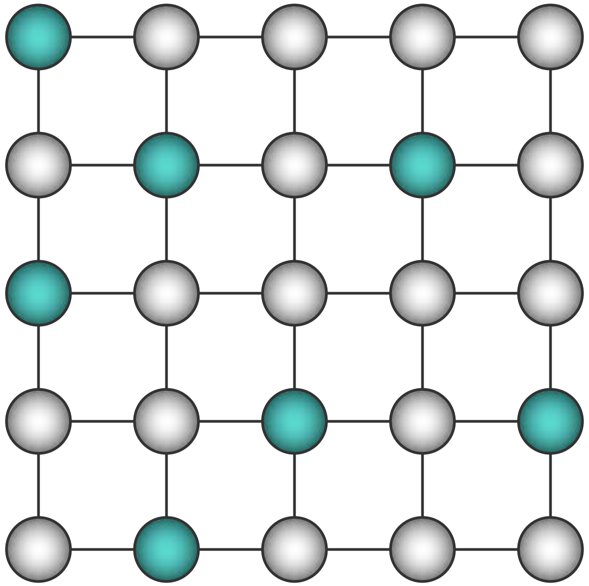
  → Inference is NP-hard
  - But there often exist "obvious" optimal or near-opt elimination ordering

Examples

- Star

- Tree

More example: Ising model



Summary

- The simple Eliminate algorithm captures the key algorithmic Operation underlying probabilistic inference:

  --- That of taking a sum over product of potential functions

- What can we say about the overall computational complexity of the algorithm? In particular, how can we control the "size" of the summands that appear in the sequence of summation operation.

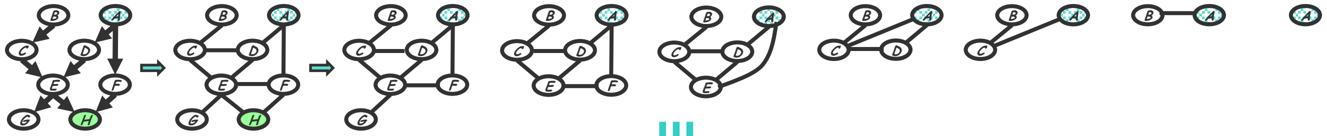
- The computational complexity of the Eliminate algorithm can be reduced to purely graph-theoretic considerations.

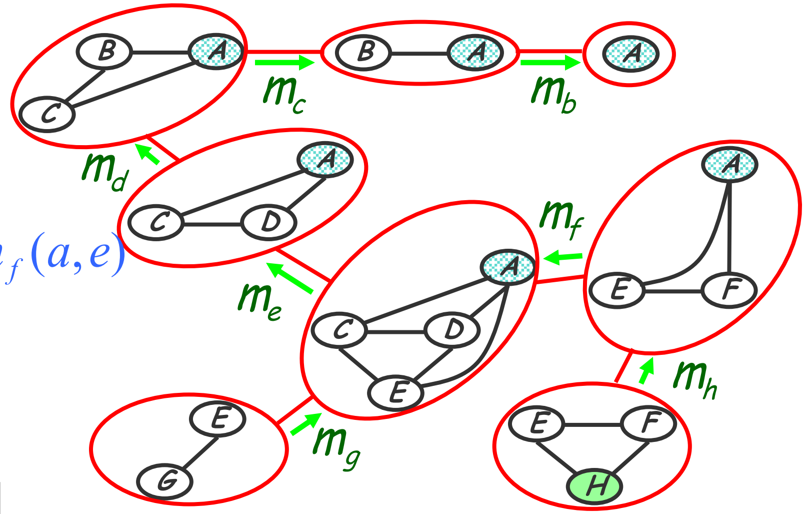
- This graph interpretation will also provide hints about how to design improved inference algorithm that overcome the limitation of Eliminate.

From Elimination to Message Passing

- Our algorithm so far answers only one query (e.g., on one node), do we need to do a complete elimination for every such query?

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



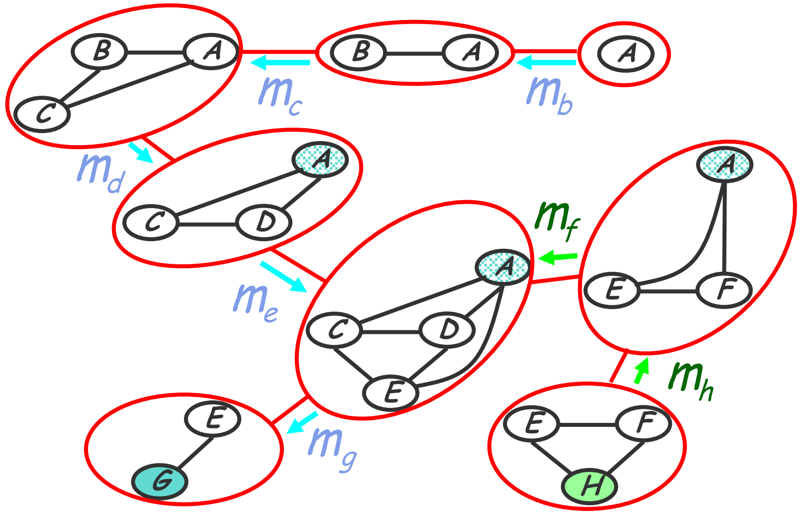


$$m_e(a,c,d) = \sum_e p(e|c,d) m_g(e) m_f(a,e)$$

- Messages can be reused

From Elimination to Message Passing

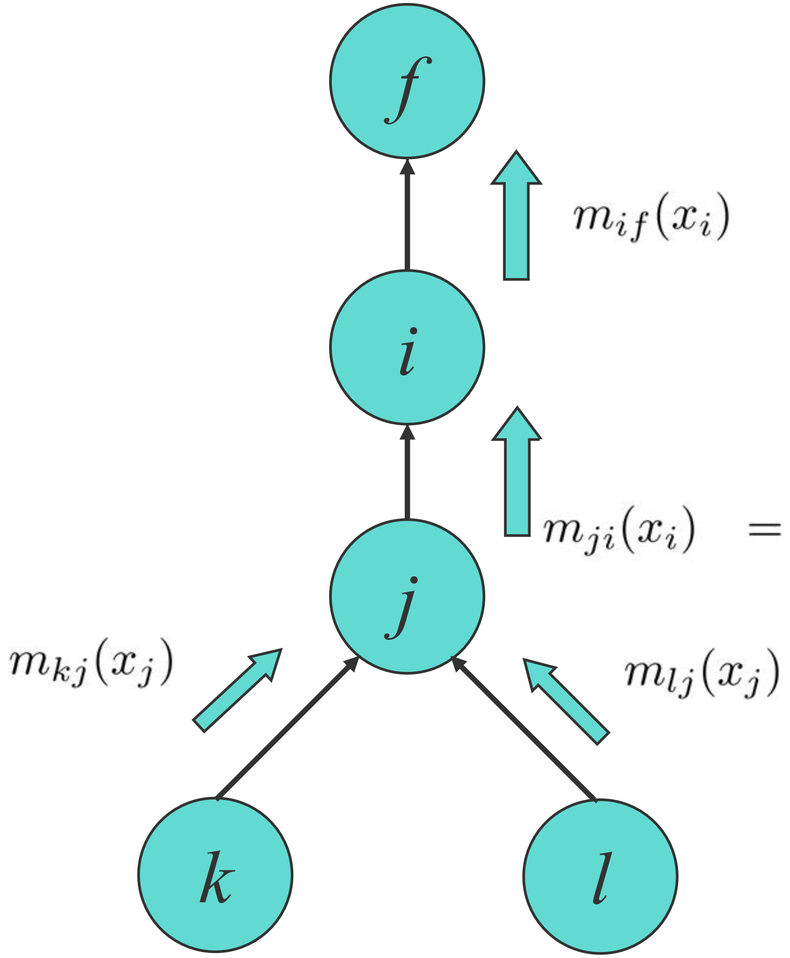
- Our algorithm so far answers only one query (e.g., on one node), do we need to do a complete elimination for every such query?
- Elimination ≡ message passing on a clique tree
  - Another query ...



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

Message passing on a tree

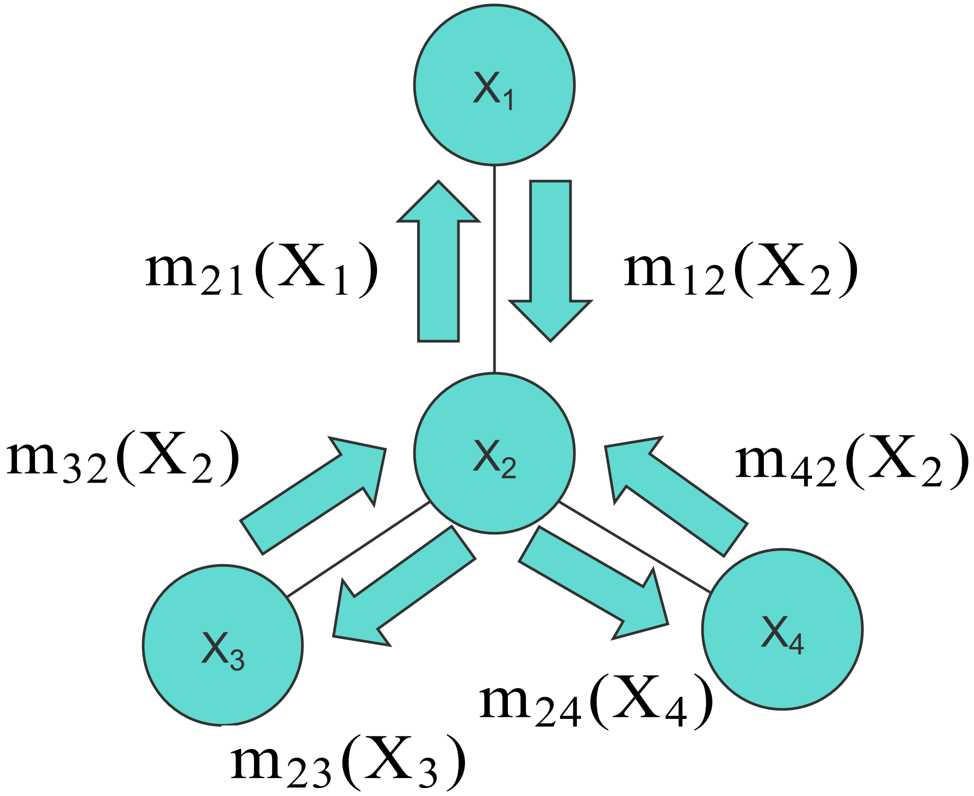
- Elimination on trees is equivalent to message passing along tree branches!



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

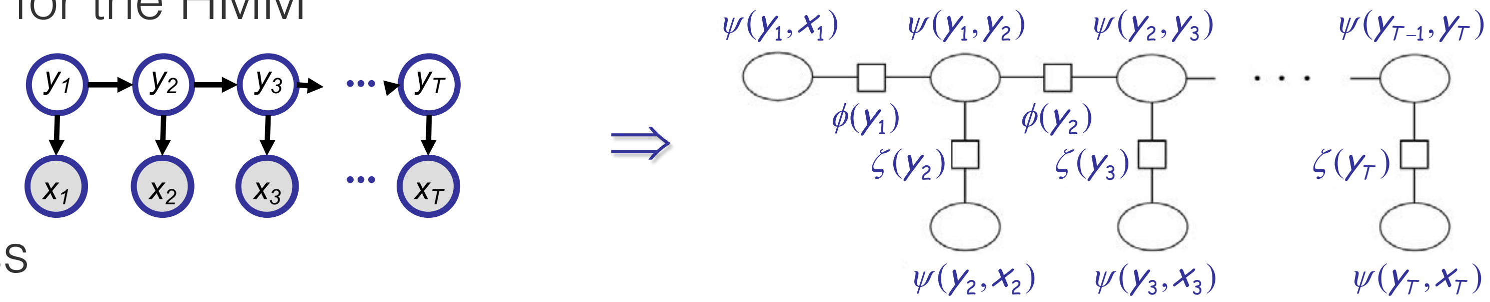
The message passing protocol:

- A two-pass algorithm:



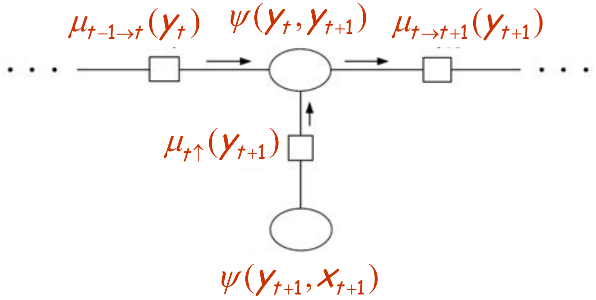
Message Passing for HMMs (cont.)

- A junction tree for the HMM



- Rightward pass

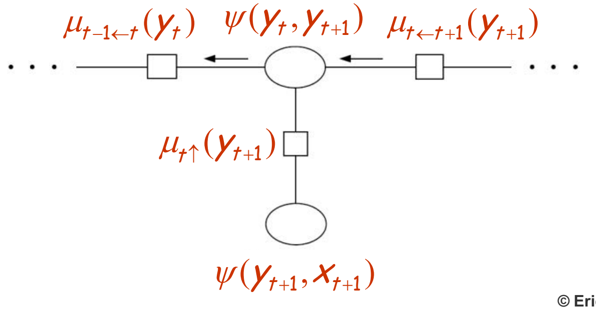
$$\mu_{t\to t+1}(y_{t+1}) = \sum_{y_t} \psi(y_t, y_{t+1}) \mu_{t-1\to t}(y_t) \mu_{t\uparrow}(y_{t+1})$$
$$= \sum_{y_t} p(y_{t+1} \mid y_t) \mu_{t-1\to t}(y_t) p(x_{t+1} \mid y_{t+1})$$
$$= p(x_{t+1} \mid y_{t+1}) \sum a_{y_t, y_{t+1}} \mu_{t-1\to t}(y_t)$$



  - This is exactly the *forward algorithm*!

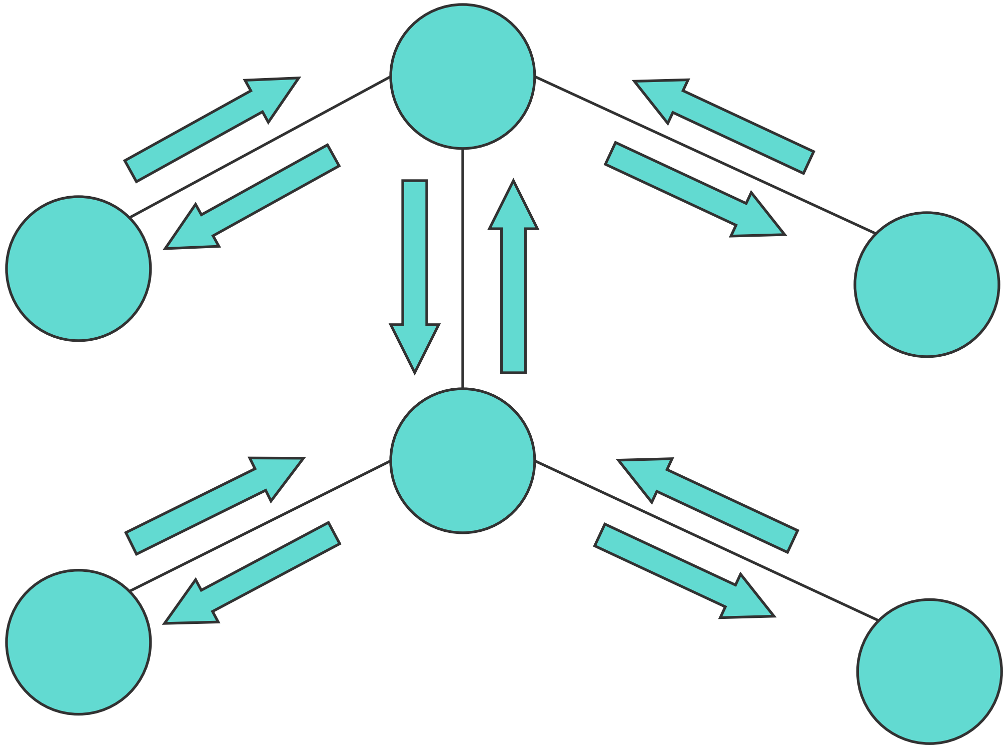
- Leftward pass …

$$\mu_{t-1\leftarrow t}(y_t) = \sum_{y_{t+1}} \psi(y_t, y_{t+1}) \mu_{t\leftarrow t+1}(y_{t+1}) \mu_{t\uparrow}(y_{t+1})$$
$$= \sum_{y_{t+1}} p(y_{t+1} \mid y_t) \mu_{t\leftarrow t+1}(y_{t+1}) p(x_{t+1} \mid y_{t+1})$$



  - This is exactly the *backward algorithm*!

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

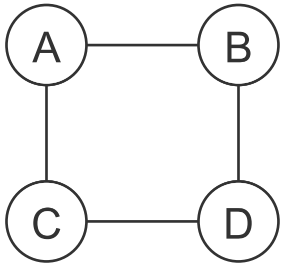
Correctness of BP on tree

- Collollary: the synchronous implementation is "non-blocking"

- Thm: The Message Passage Guarantees obtaining all marginals in the tree

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

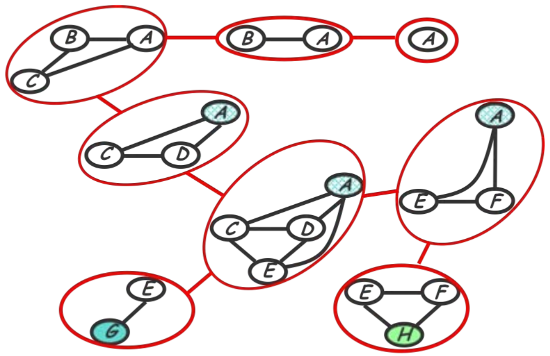
- What about non-tree? (a home work problem)
  - Please do message passing from B->D->C->A
  - And from C->D->B->A
  - Compare the marginals of A



Inference on general GM

- Now, what if the GM is not a tree-like graph?
- Can we still directly run
  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
  → Messaging passing on a JT

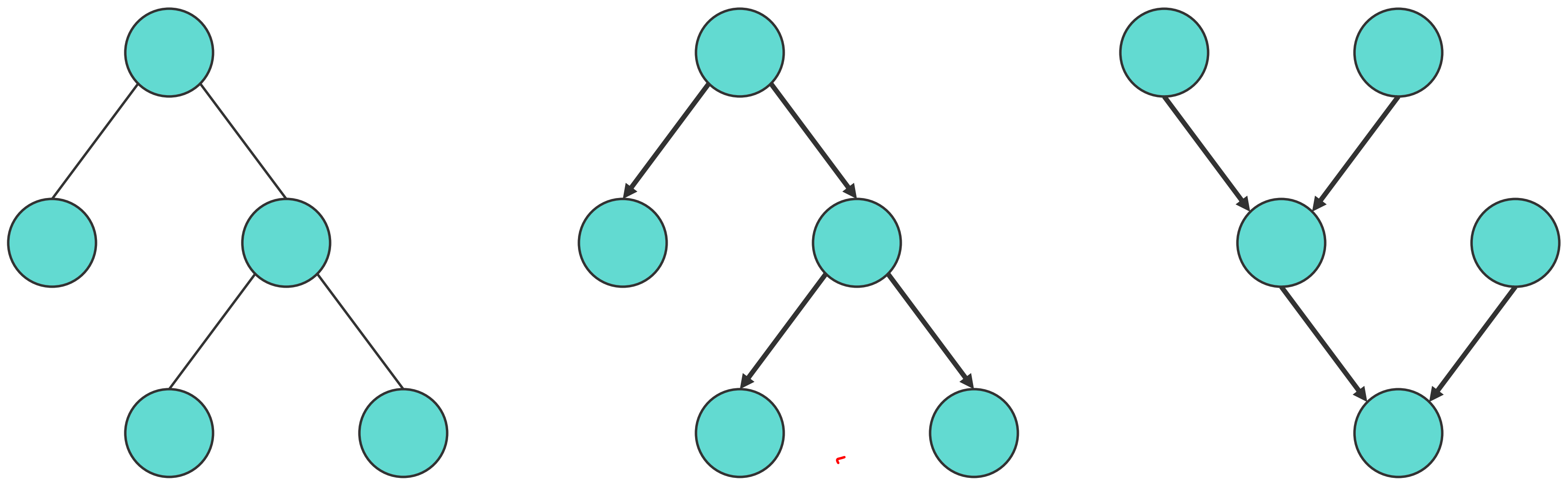


Supplementary materials

Examples of VE on chain GMs

Message Passing

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

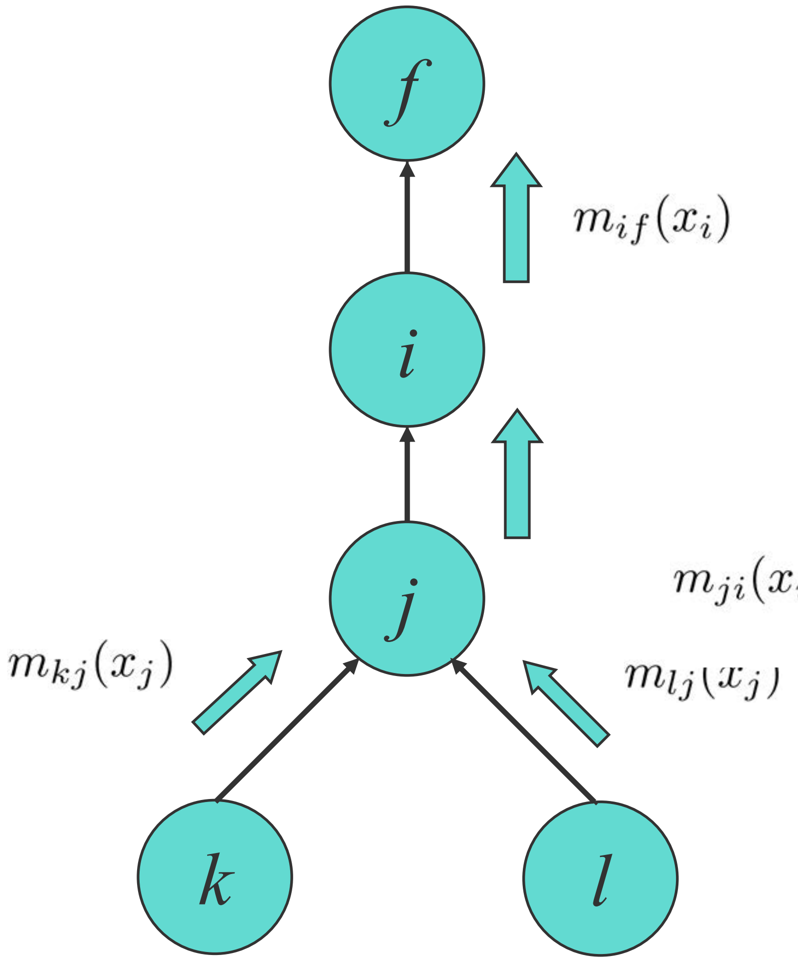
Equivalence of directed and undirected trees

- 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:?

From elimination to message passing

- Recall ELIMINATION algorithm:
  - Choose an ordering 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

Elimination on a tree



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

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

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

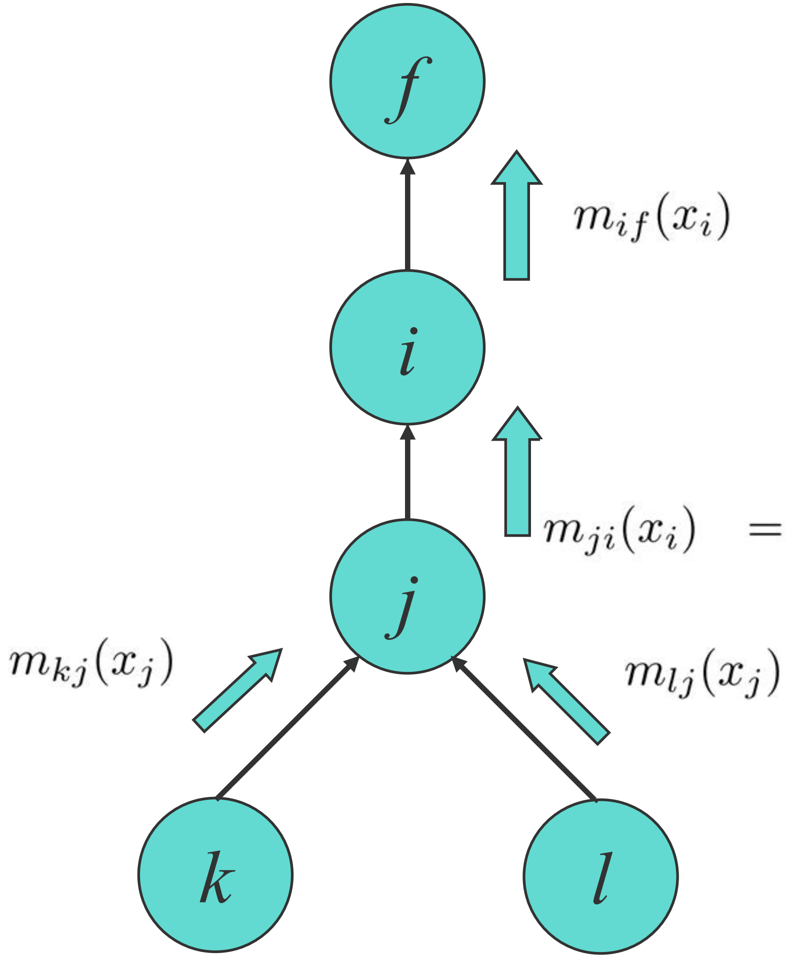
$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j)\backslash i} m_{kj}(x_j) \right)$$

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

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

Message passing on a tree

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



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

From elimination to message passing

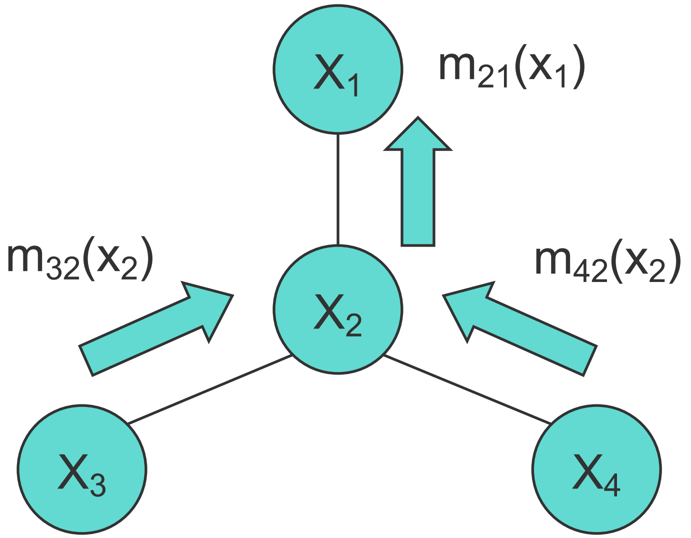
- Recall ELIMINATION algorithm:
  - Choose an ordering Z in which query node f is the final node
  - Place all potentials on an active list
  - Eliminate node i by removing all potentials containing i, take sum/product over x_i.
  - Place the resultant factor back on the list

- For a TREE graph:
  - Choose query node f as the root of the tree
  - View tree as a directed tree with edges pointing towards leaves 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 message passing protocol:

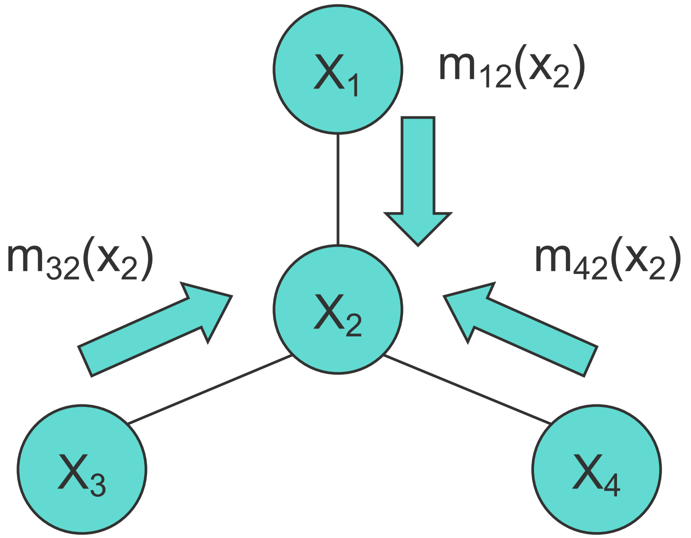
- A node can send a message to its neighbors when (and only when) it has received messages from all its *other* neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



Computing $P(X_1)$

The message passing protocol:

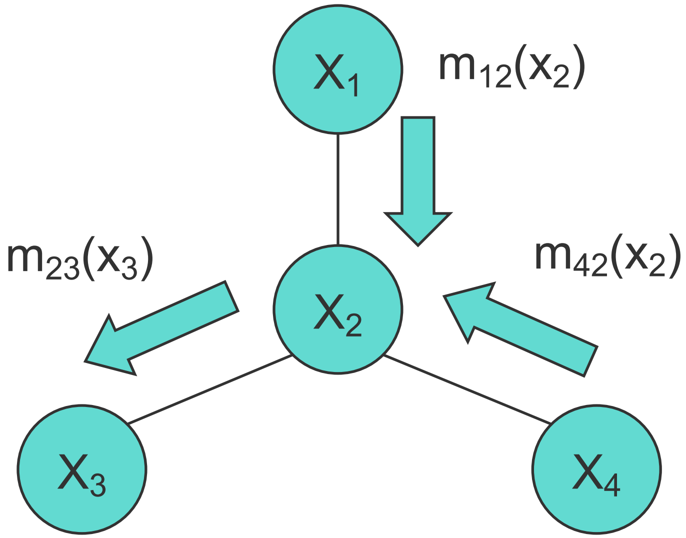
- A node can send a message to its neighbors when (and only when) it has received messages from all its *other* neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



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 $P(X_3)$

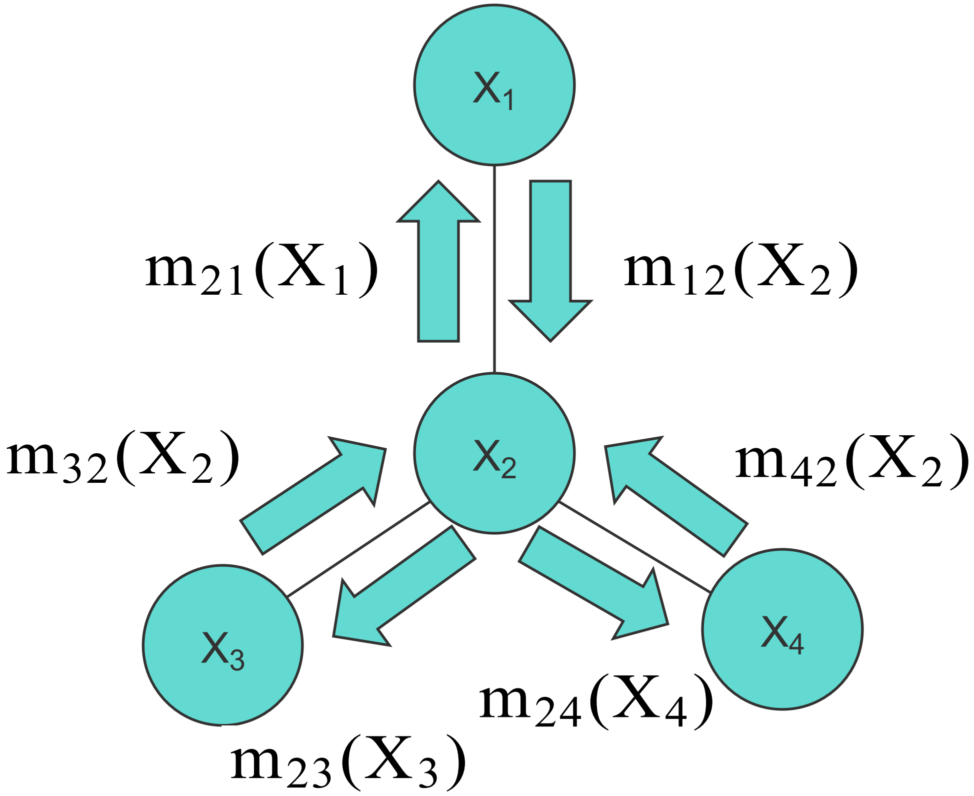
Computing node marginals

- 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)

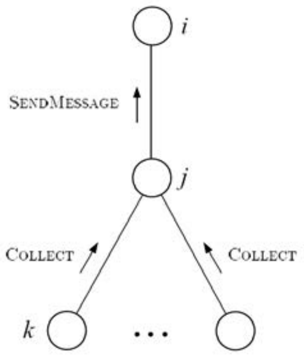
DISTRIBUTE(i, j)
    SENDMESSAGE(i, j)
    for $k \in \mathcal{N}(j) \backslash i$
        DISTRIBUTE(j, k)

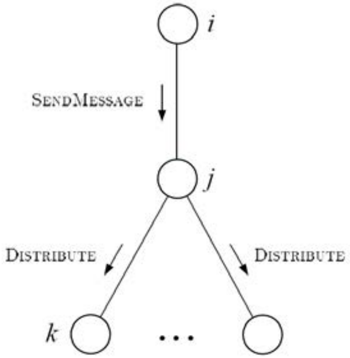
SENDMESSAGE(j, i)

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

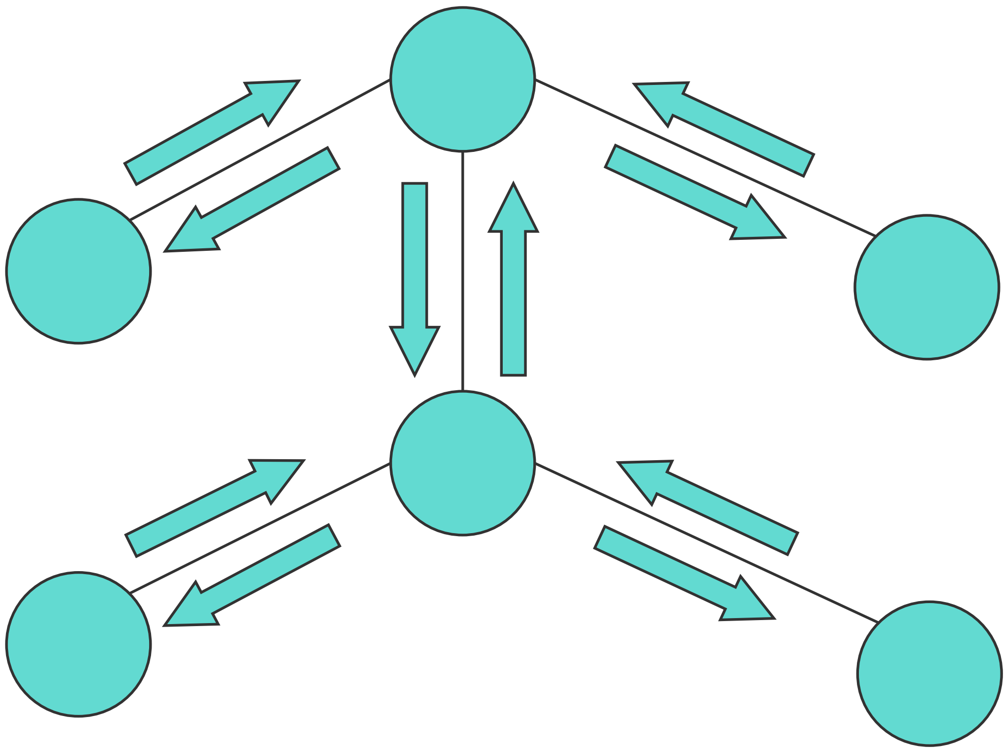
COMPUTEMARGINAL(i)

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





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"

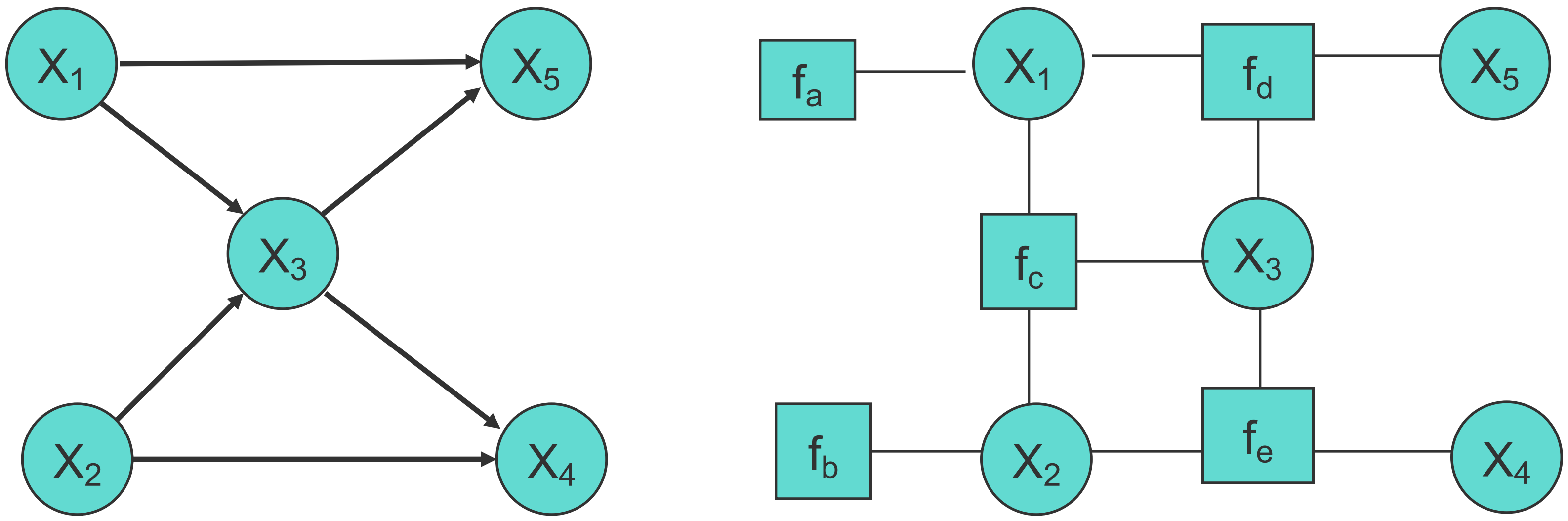
- Thm: The Message Passage Guarantees obtaining all marginals in the tree

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

- What about non-tree?

Another view of SP: Factor Graph

- Example 1

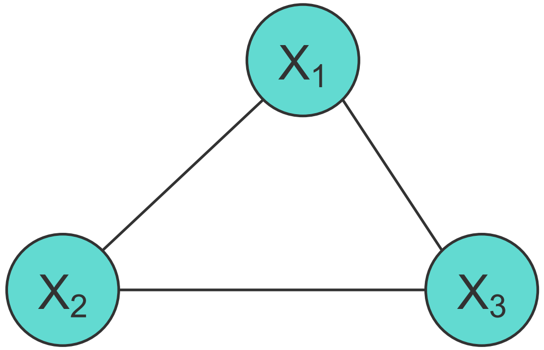


$P(X_1)$ $P(X_2)$ $P(X_3|X_1,X_2)$ $P(X_5|X_1,X_3)$ $P(X_4|X_2,X_3)$

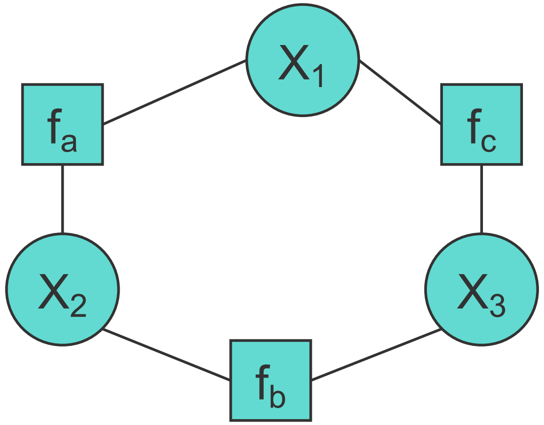
$f_a(X_1)$ $f_b(X_2)$ $f_c(X_3,X_1,X_2)$ $f_d(X_5,X_1,X_3)$ $f_e(X_4,X_2,X_3)$

Factor Graphs

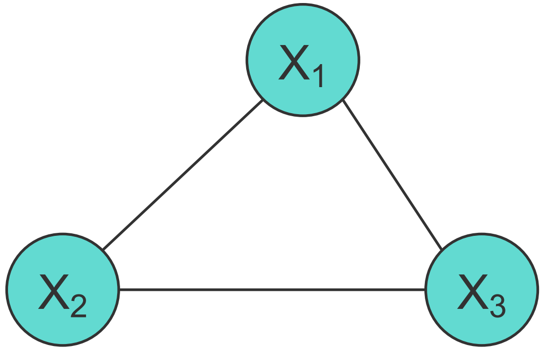
- Example 2



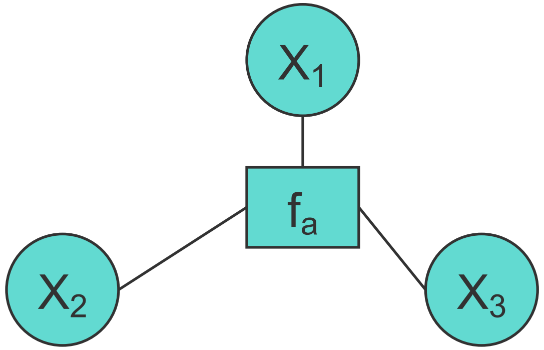
$\psi(x_1,x_2,x_3) = f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_3,x_1)$



- Example 3

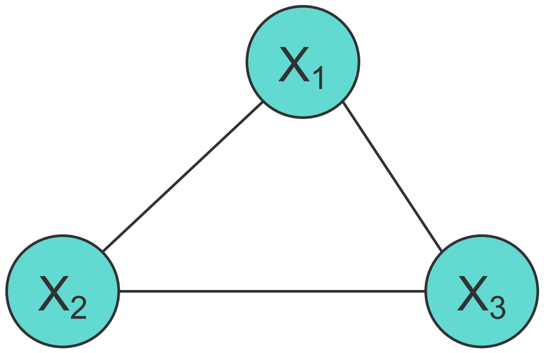


$\psi(x_1,x_2,x_3) = f_a(x_1,x_2,x_3)$

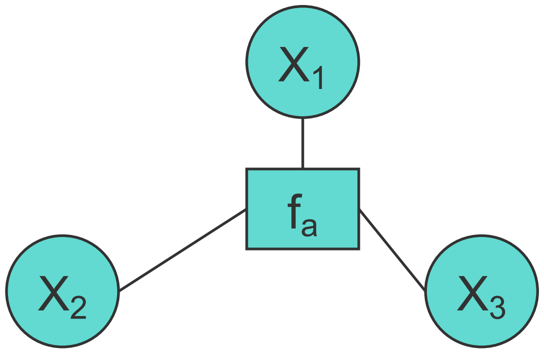


Factor Tree

- 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

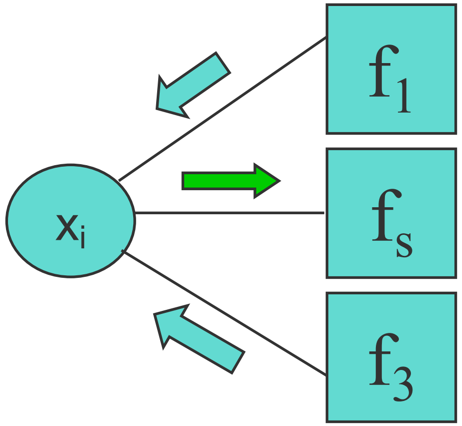


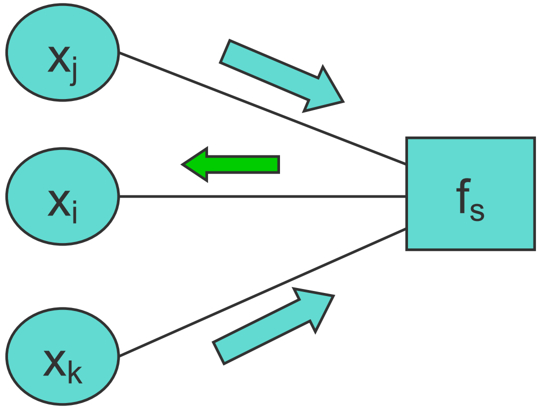
$\psi(x_1,x_2,x_3) = f_a(x_1,x_2,x_3)$



Message Passing on a Factor Tree

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



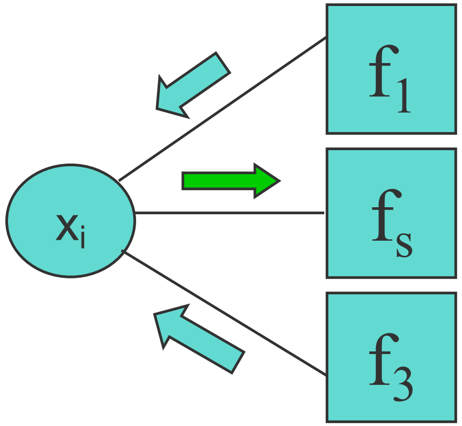


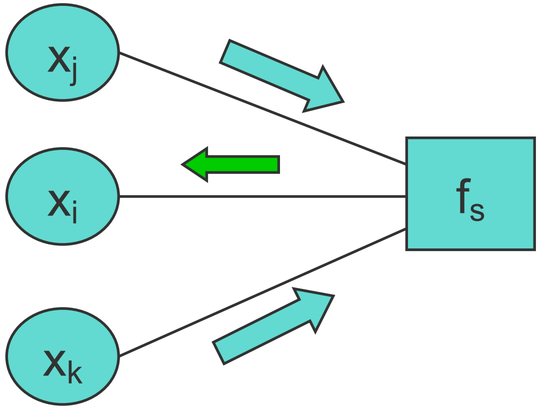
$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \backslash s} \mu_{ti}(x_i)$$

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

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:

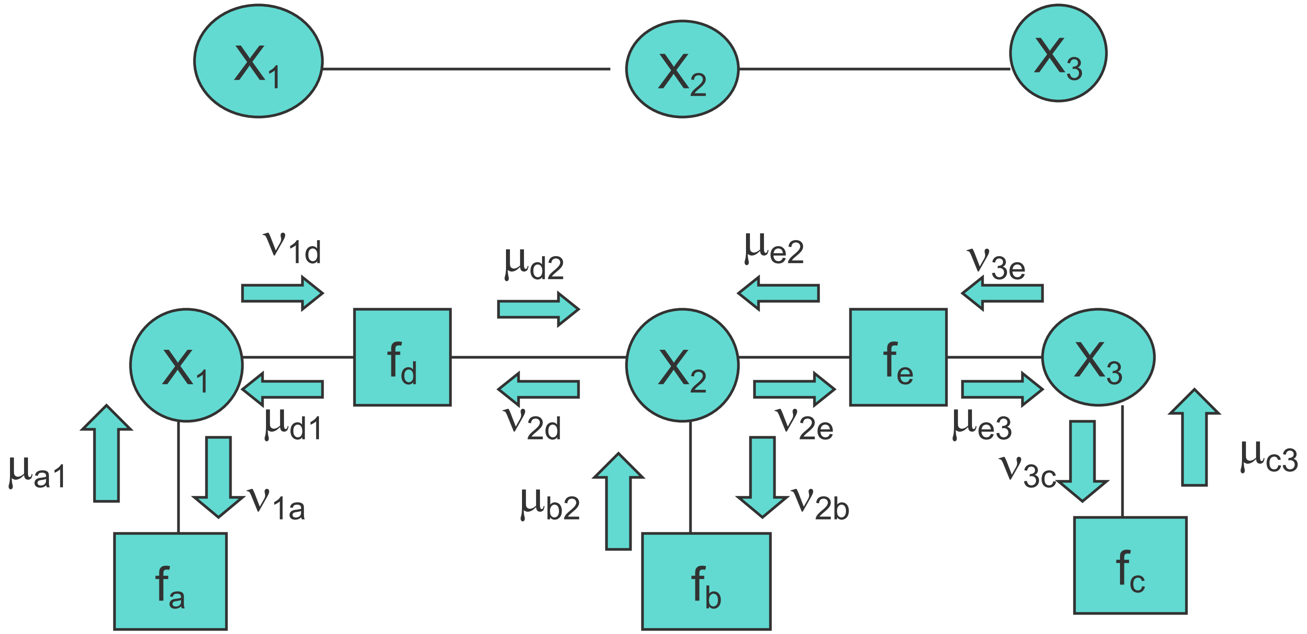




$$P(x_i) \propto \prod_{s \in N(i)} \mu_{si}(x_i)$$

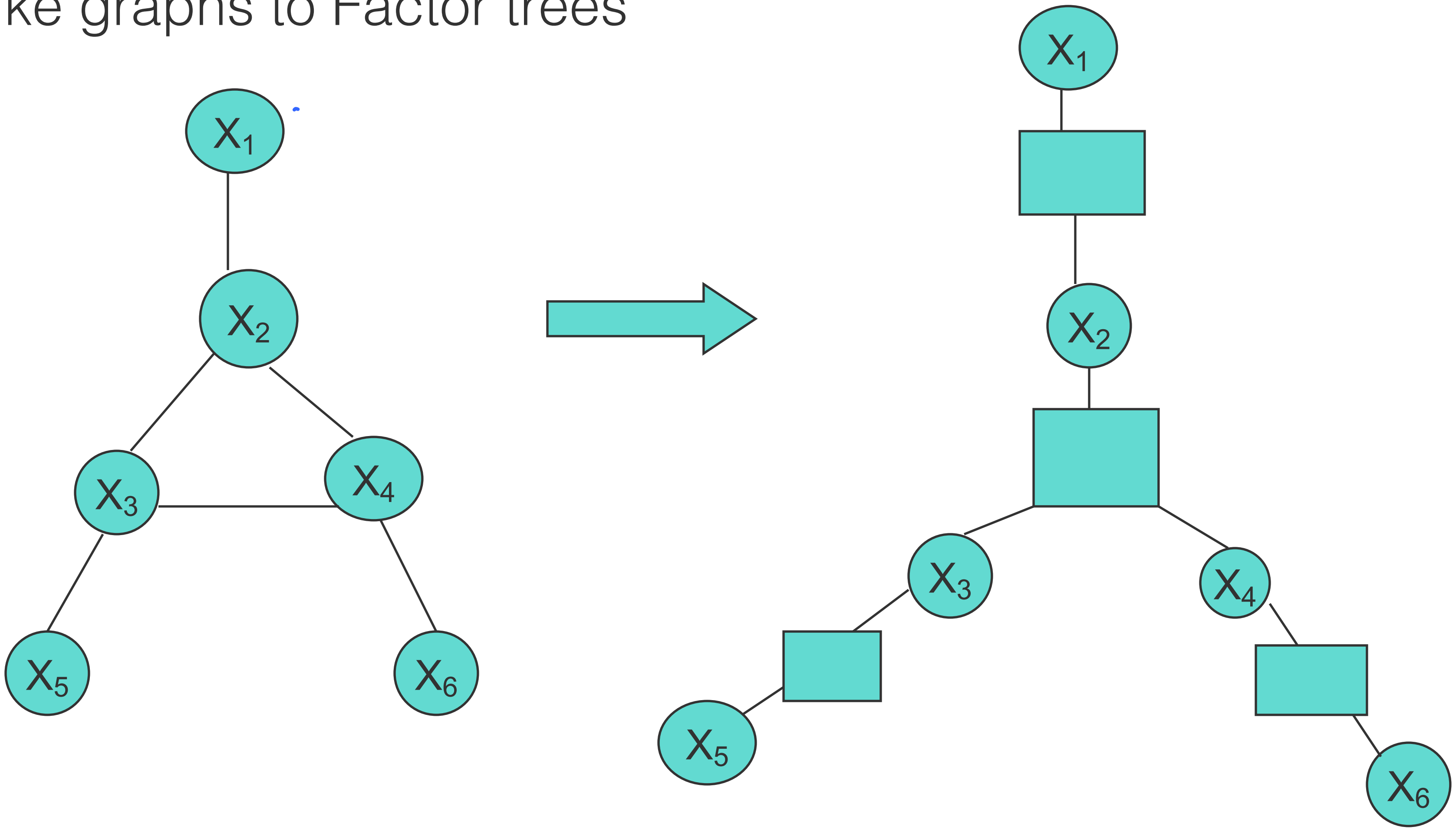
$$\propto \nu_{is}(x_i)\mu_{si}(x_i)$$

BP on a Factor Tree

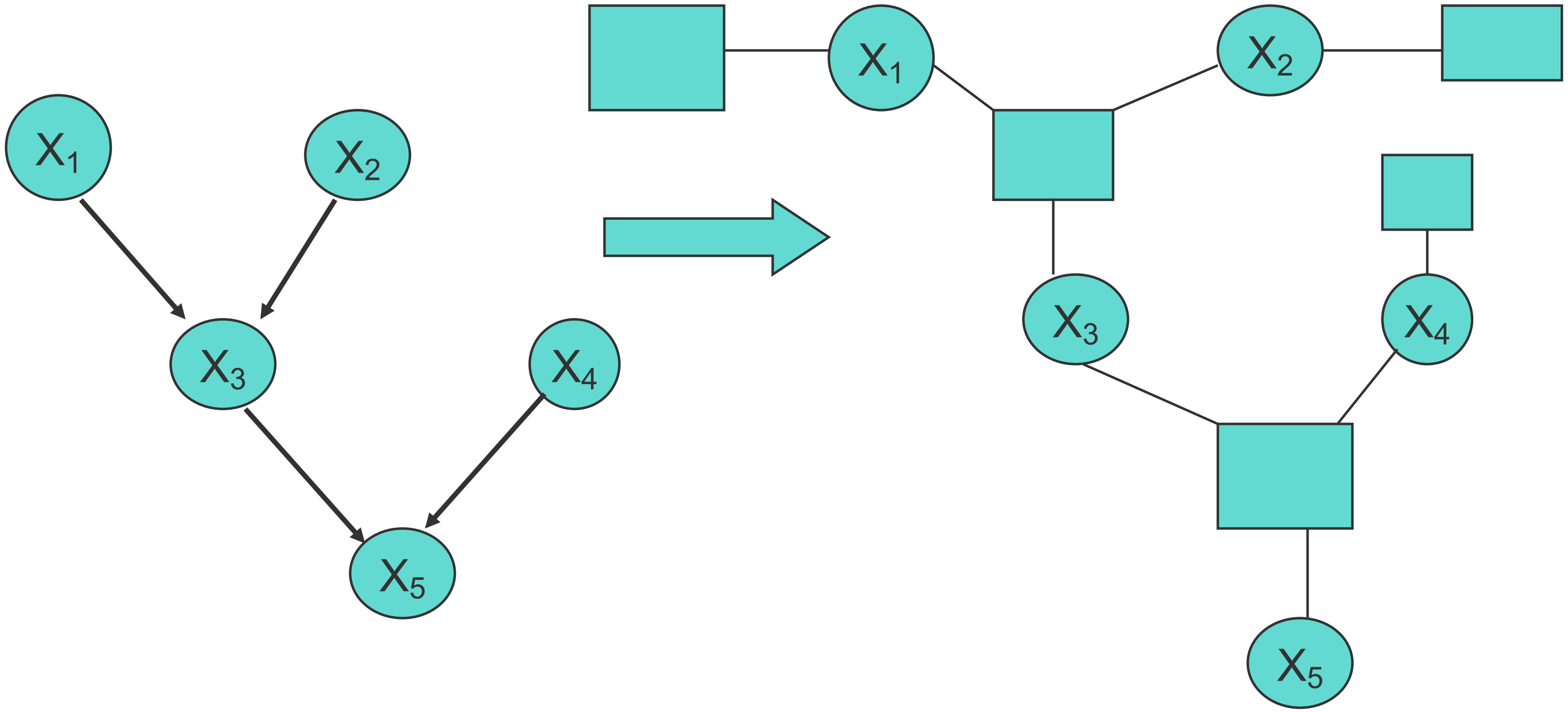


Why factor graph?

- Tree-like graphs to Factor trees

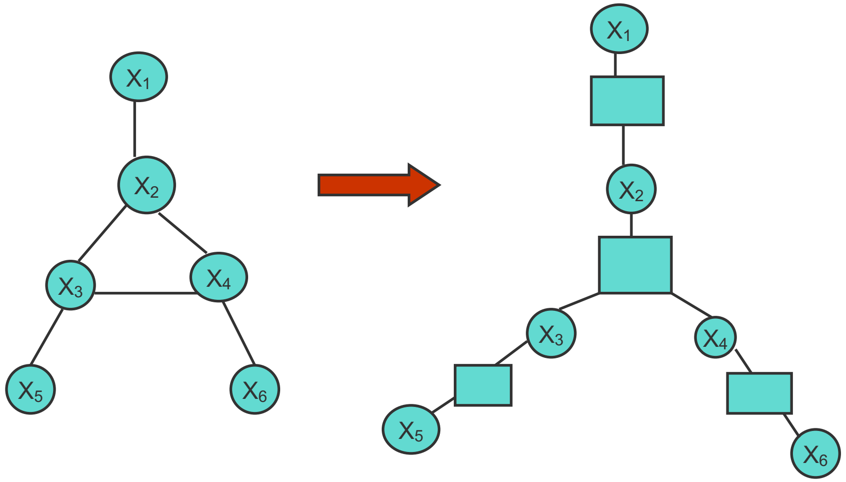


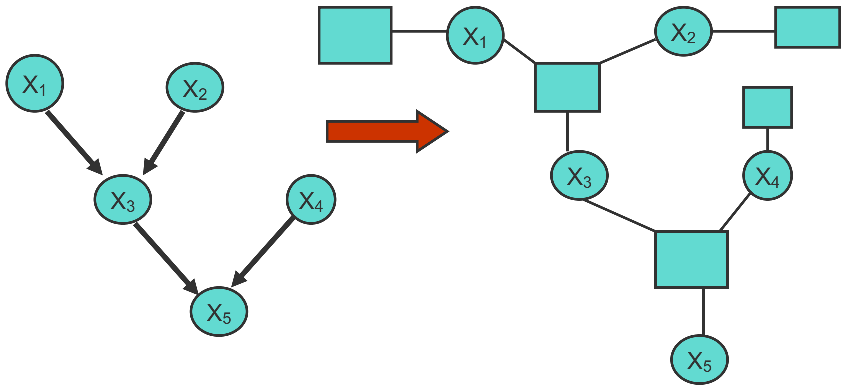
Poly-trees to Factor trees



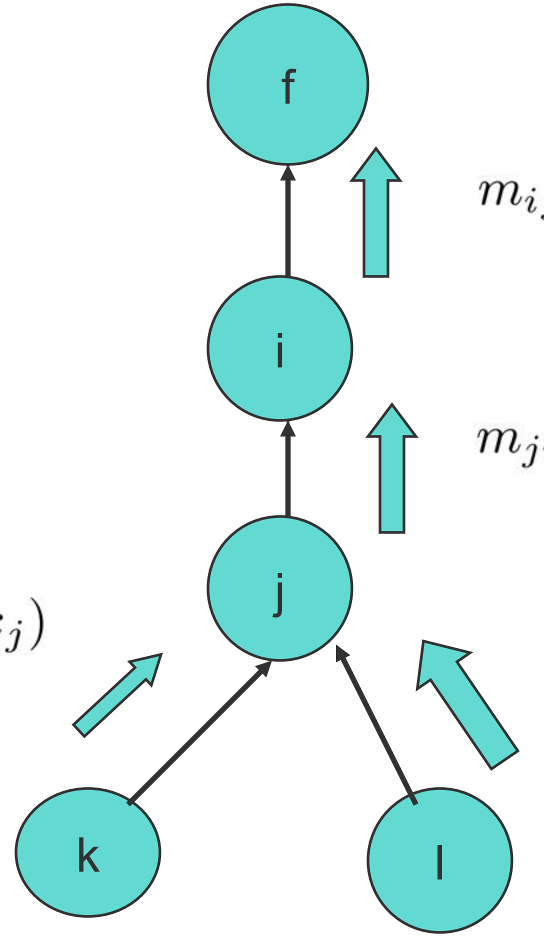
Why factor graph?

- 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



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

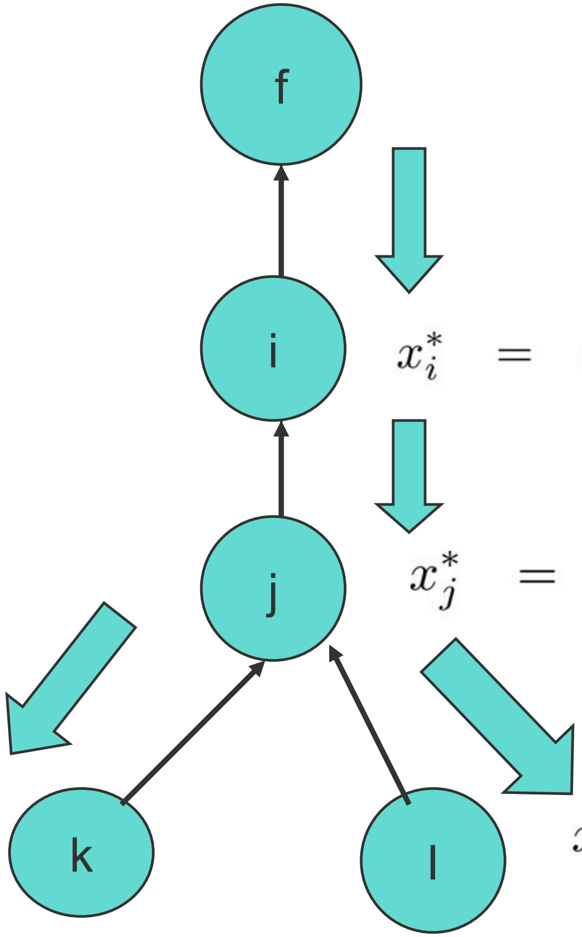
$m_{if}(x_f)$

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

$m_{kj}(x_j)$

$m_{lj}(x_j)$

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-passing protocol along its edges?

- For non-trees, we do not have the guarantee that message-passing will be consistent!

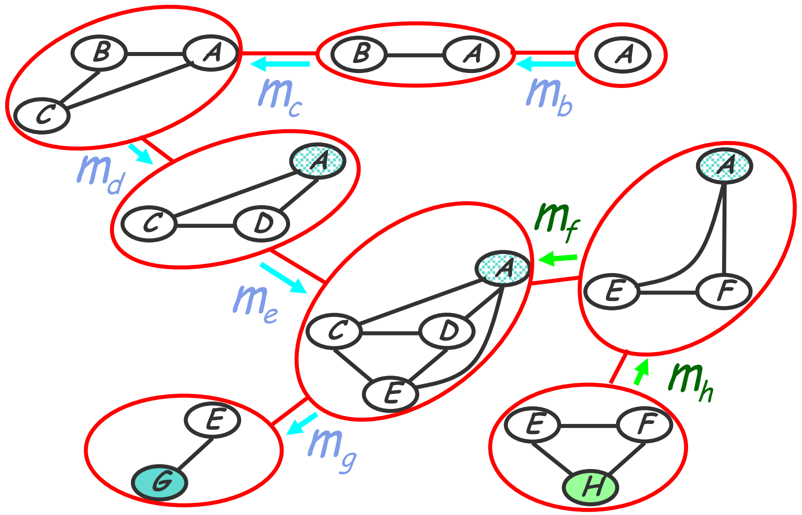
- Then what?
  - Construct a graph data-structure from P that has a tree structure, and run message-passing on it!

→ Junction tree algorithm

Elimination Clique

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

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



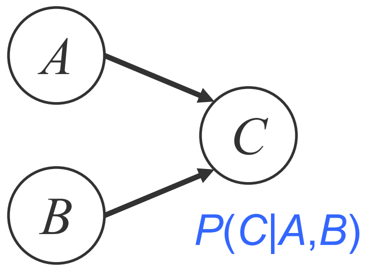
- Can this lead to an generic inference algorithm?

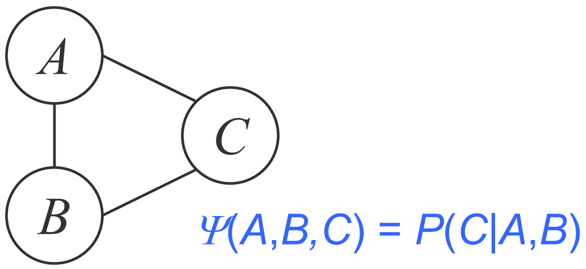
Moral Graph

- Note that for both directed GMs and undirected GMs, the joint probability is in a product form:

BN: $P(\mathbf{X}) = \prod_{i=1:d} P(X_i \mid \mathbf{X}_{\pi_i})$ **MRF:** $P(\mathbf{X}) = \frac{1}{Z} \prod_{c \in C} \psi_c(\mathbf{X}_c)$

- So let's convert local conditional probabilities into potentials; then the second expression will be generic, but how does this operation affect the directed graph?
  - We can think of a conditional probability, e.g,. $P(C|A,B)$ as a function of the three variables A, B, and C (we get a real number of each configuration):

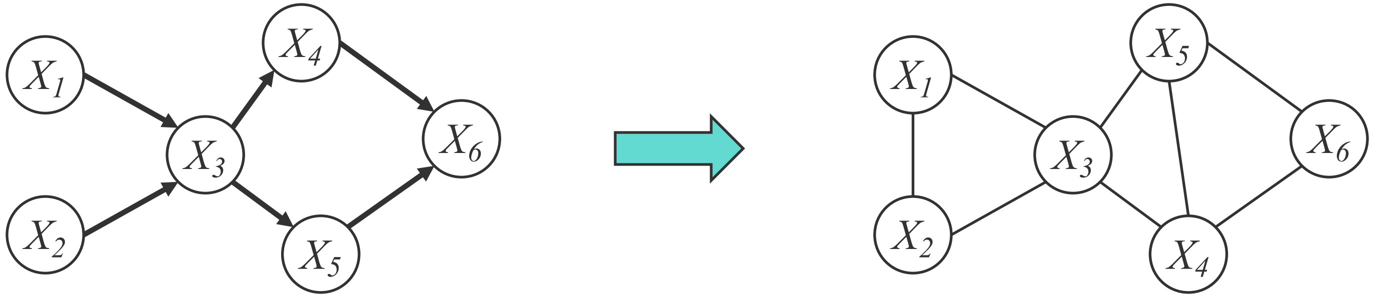




  - Problem: But a node and its parent are not generally in the same clique in a BN
  - Solution: Marry the parents to obtain the "moral graph"

Moral Graph (cont.)

- Define the potential on a clique as the product over all conditional probabilities contained within the clique
- Now the product of potentials gives the right answer:



$$
\begin{aligned}
& P(X_1, X_2, X_3, X_4, X_5, X_6) \\
&= P(X_1)P(X_2)P(X_3 \mid X_1, X_2)P(X_4 \mid X_3)P(X_5 \mid X_3)P(X_6 \mid X_4, X_5) \\
&= \psi(X_1, X_2, X_3)\psi(X_3, X_4, X_5)\psi(X_4, X_5, X_6)
\end{aligned}
$$

where

$$\psi(X_1, X_2, X_3) = P(X_1)P(X_2)P(X_3 \mid X_1, X_2)$$

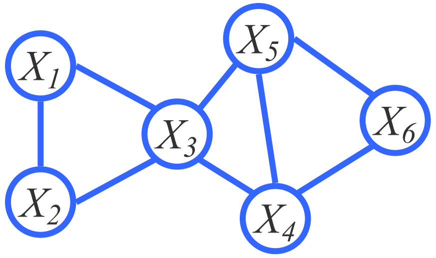
$$\psi(X_3, X_4, X_5) = P(X_4 \mid X_3)P(X_5 \mid X_3)$$

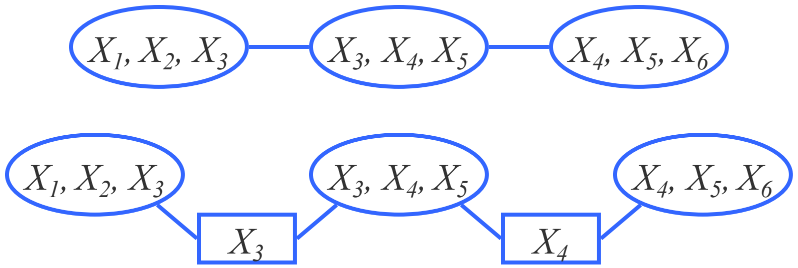
$$\psi(X_4, X_5, X_6) = P(X_6 \mid X_4, X_5)$$

Note that here the interpretation of potential is ambivalent:
it can be either *marginals* or *conditionals*

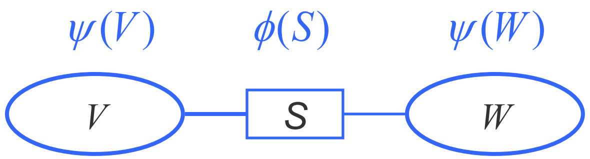
Clique trees

- A clique tree is an (undirected) tree of cliques





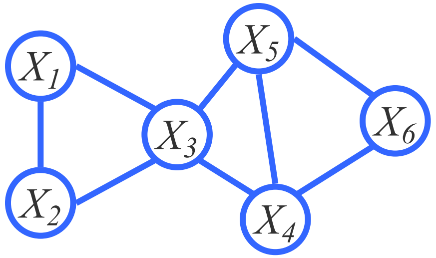
- Consider cases in which two neighboring cliques V and W have an overlap S (e.g., (X_1, X_2, X_3) overlaps with (X_3, X_4, X_5)),

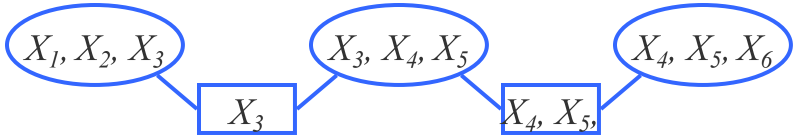


- Now we have an alternative representation of the joint in terms of the potentials:

Clique trees

- A clique tree is an (undirected) tree of cliques





- The alternative representation of the joint in terms of the potentials:

$$
\begin{aligned}
& P(X_1, X_2, X_3, X_4, X_5, X_6) \\
&= P(X_1)P(X_2)P(X_3 \mid X_1, X_2)P(X_4 \mid X_3)P(X_5 \mid X_3)P(X_6 \mid X_4, X_5) \\
&= P(X_1, X_2, X_3)\frac{P(X_3, X_4, X_5)}{P(X_3)}\frac{P(X_4, X_5, X_6)}{P(X_4, X_5)} \\
&= \psi(X_1, X_2, X_3)\frac{\psi(X_3, X_4, X_5)}{\phi(X_3)}\frac{\psi(X_4, X_5, X_6)}{\phi(X_4, X_5)}
\end{aligned}
$$

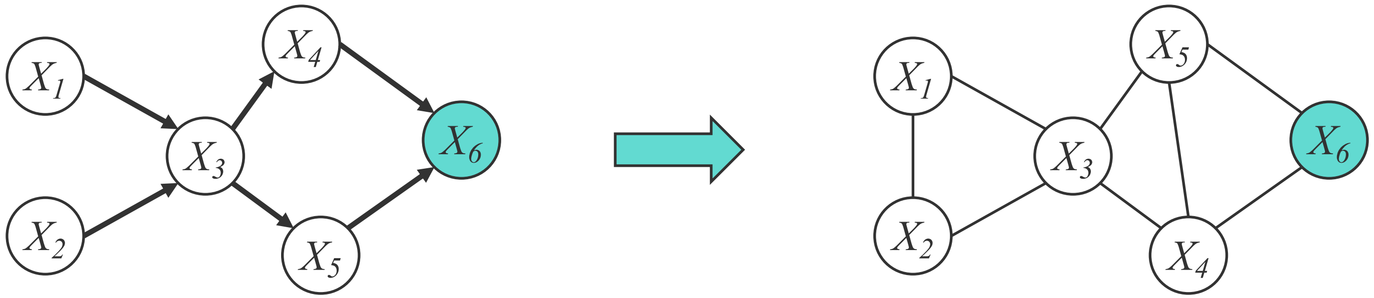
- Generally:

$$
\boxed{P(\mathbf{X}) = \frac{\prod_C \psi_C(\mathbf{X}_C)}{\prod_S \phi_S(\mathbf{X}_S)}}
$$

Now each potential is isomorphic to the *cluster marginal* of the attendant set of variables

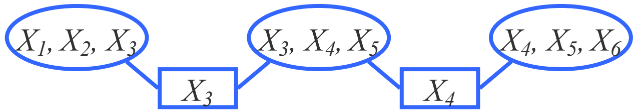
Why this is useful?

- Propagation of probabilities
  - Now suppose that some evidence has been "absorbed" (i.e., certain values of some nodes have been observed). How do we propagate this effect to the rest of the graph?



  - What do we mean by propagate?
    Can we adjust all the potentials $\{\psi\}$, $\{\phi\}$ so that they still represent the correct cluster marginals (or unnormalized equivalents) of their respective attendant variables?

  - Utility?



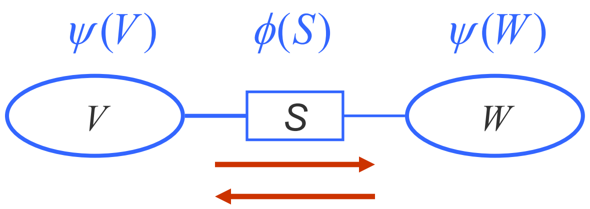
$$P(X_1 \mid X_6 = x_6) = \sum_{X_2, X_3} \psi(X_1, X_2, X_3)$$

$$P(X_3 \mid X_6 = x_6) = \phi(X_3)$$

$$P(x_6) = \sum_{X_4, X_5} \psi(X_4, X_5, x_6)$$

Local operations!

Local Consistency



- We have two ways of obtaining $p(S)$

$$P(S) = \sum_{V \setminus S} \psi(V) \qquad P(S) = \sum_{W \setminus S} \psi(W)$$

and they must be the same

- The following update-rule ensures this:
  - Forward update:

$$\phi_S^* = \sum_{V \setminus S} \psi^*{}_V \qquad \psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W$$

  - Backward update

$$\phi_S^{**} = \sum_{W \setminus S} \psi_W^* \qquad \psi_V^{**} = \frac{\phi_S^{**}}{\phi_S^*} \psi_V^*$$

- Two important identities can be proven

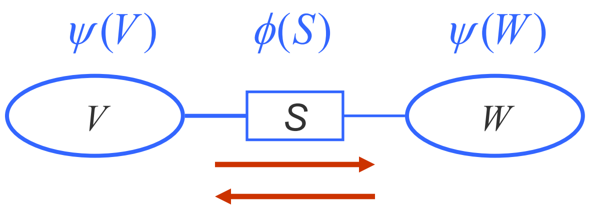
$$\sum_{V \setminus S} \psi_V^{**} = \sum_{W \setminus S} \psi_W^* = \phi_S^{**}$$

Local Consistency

$$\frac{\psi_V^* \psi_W^*}{\phi_S^*} = \frac{\psi_V^{**} \psi_W^{**}}{\phi_S^{**}} = \frac{\psi_V \psi_W}{\phi_S}$$

Invariant Joint

Message Passing Algorithm



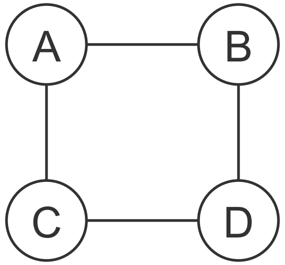
$$\phi_S^{*} = \sum_{V \setminus S} \psi_V^{*} \qquad \psi_W^{*} = \frac{\phi_S^{*}}{\phi_S} \psi_W$$

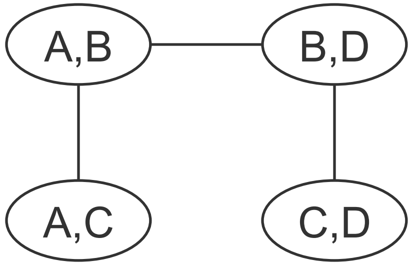
$$\phi_S^{**} = \sum_{W \setminus S} \psi_W^{*} \qquad \psi_V^{**} = \frac{\phi_S^{**}}{\phi_S^{*}} \psi_V^{*}$$

- This simple local message-passing algorithm on a clique tree defines the general probability propagation algorithm for directed graphs!
  - Many interesting algorithms are special cases:
    - Forward-backward algorithm for hidden Markov models,
    - Kalman filter updates
    - Pealing algorithms for probabilistic trees
  - The algorithm seems reasonable. Is it correct?

A problem

- Consider the following graph and a corresponding clique tree

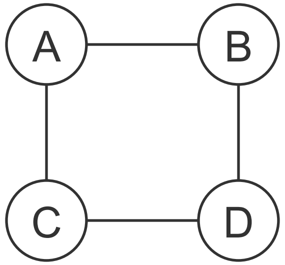




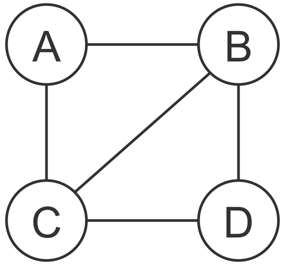
  - Note that C appears in two non-neighboring cliques
- *Question*: with the previous message passage, can we ensure that the probability associated with C in these two (non-neighboring) cliques consistent?
- Answer: No. It is not true that in general local consistency implies global consistency
- What else do we need to get such a guarantee?

Triangulation

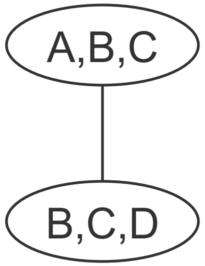
- A triangulated graph is one in which *no cycles* with four or more nodes exist in which there is no *chord*



- We triangulate a graph by adding chords:

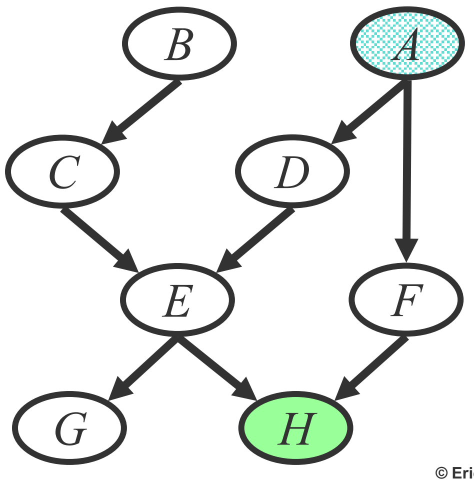


- Now we no longer have our global inconsistency problem.
  - A clique tree for a triangulated graph has the *running intersection property*: If a node appears in two cliques, it appears everywhere on the path between the cliques
  - Thus local consistency implies global consistency



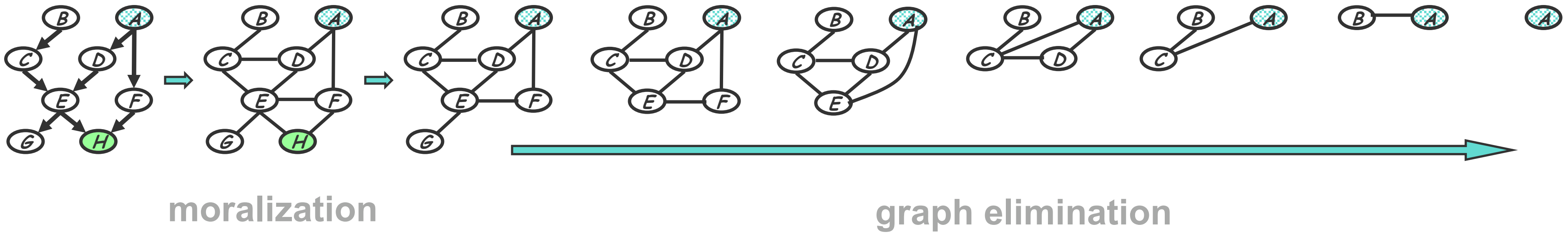
Junction trees

- A clique tree for a triangulated graph is referred to as a *junction tree*
- In junction trees, local consistency implies global consistency. Thus the local message-passing algorithms is (provably) correct
- It is also possible to show that *only* triangulated graphs have the property that their clique trees are junction trees. Thus if we want local algorithms, we *must* triangulate
- Are we now all set?
  - How to triangulate?
  - The complexity of building a JT depends on how we triangulate!!
  - Consider this network: it turns out that we will need to pay an $O(2^4)$ or $O(2^6)$ cost depending on how we triangulate!



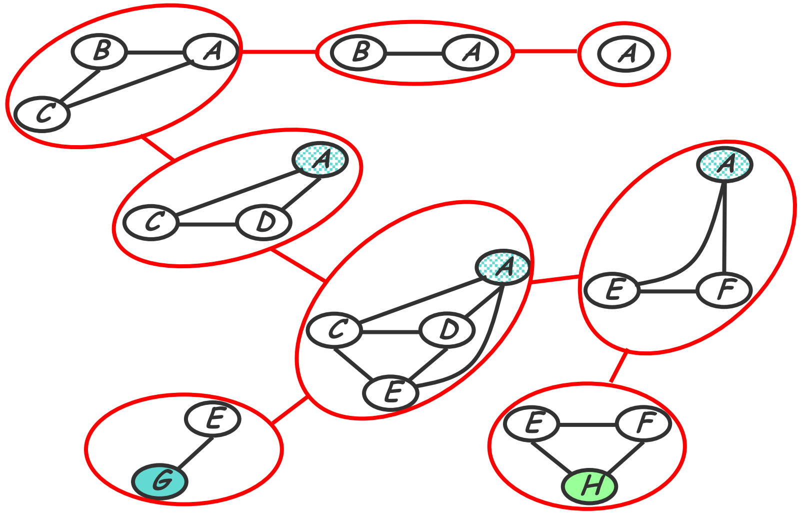
How to triangulate

- A graph elimination algorithm

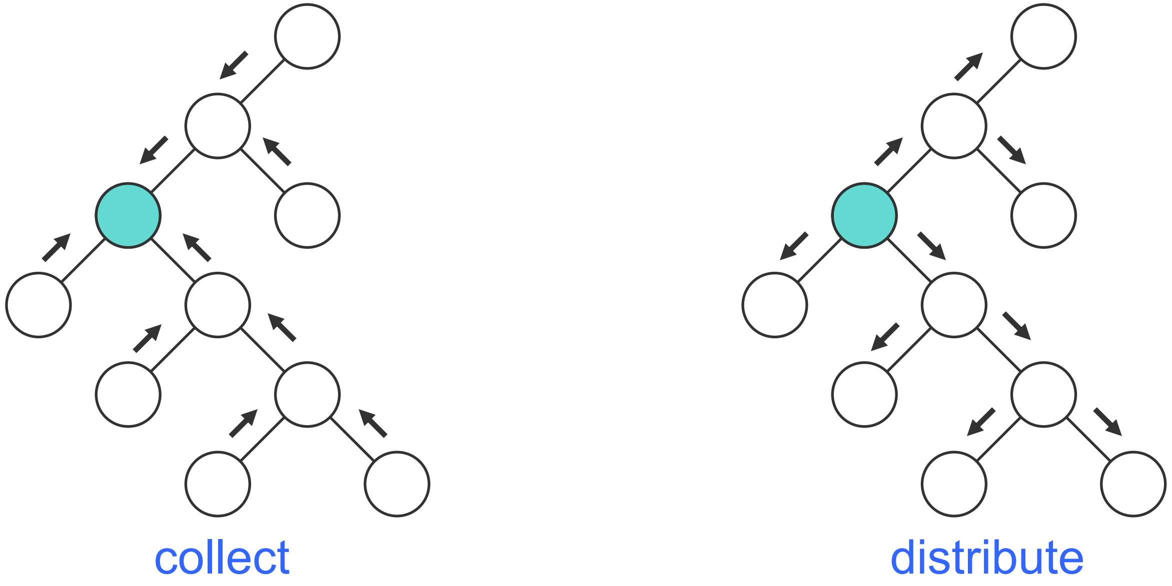


- Intermediate terms correspond to the cliques resulted from elimination
  - "good" elimination orderings lead to small cliques and hence reduce complexity (what will happen if we eliminate "e" first in the above graph?)
  - finding the optimum ordering is NP-hard, but for many graph optimum or near-optimum can often be heuristically found

A junction tree



Message-passing algorithms



- Message update
  - The Hugin update
  - The Shafer-Shenoy update

$$\boxed{\phi_S^* = \sum_{V \setminus S} \psi_V \qquad \psi_W^* = \frac{\phi_S^*}{\phi_S} \psi_W}$$

$$\boxed{m_{i \to j}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq j} m_{k \to i}(S_{ki})}$$

A Sketch of the Junction Tree Algorithm

- The algorithm

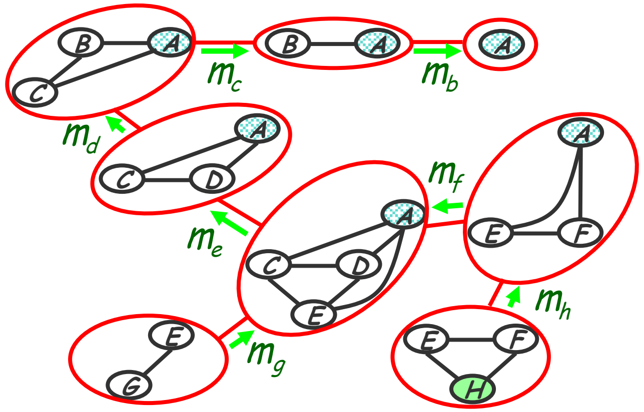
1. Moralize the graph (trivial)
2. Triangulate the graph (good heuristic exist, but actually NP hard)
3. Build a clique tree (e.g., using a maximum spanning tree algorithm
4. Propagation of probabilities --- a local message-passing protocol

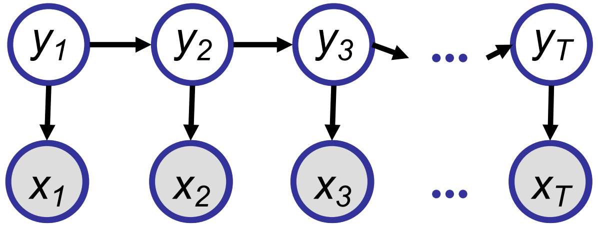
- Results in marginal probabilities of all cliques --- solves all queries in a single run
- A generic exact inference algorithm for any GM
- Complexity: exponential in the size of the maximal clique --- a good elimination order often leads to small maximal clique, and hence a good (i.e., thin) JT

Recall the Elimination and Message Passing Algorithm

- Elimination ≡ message passing on a clique tree

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



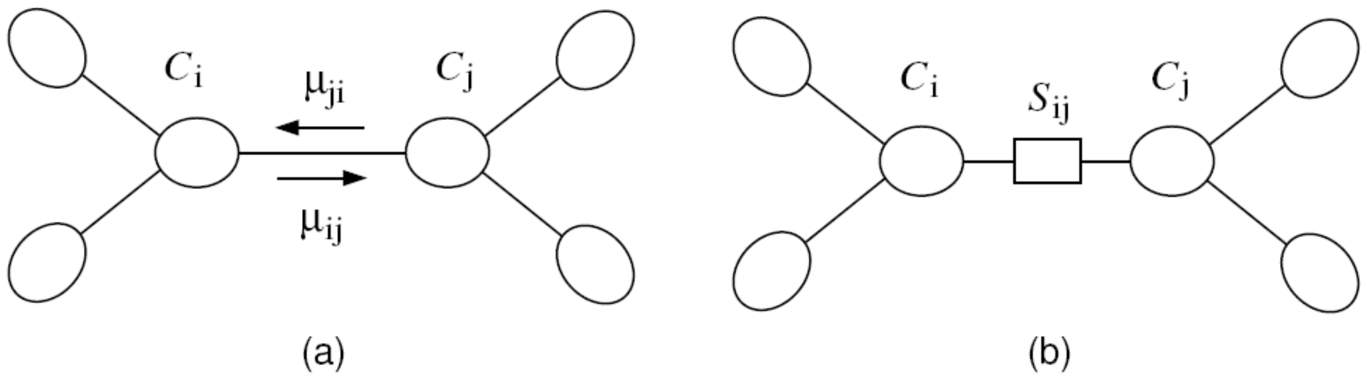


$$P(\mathbf{x}) = \sum_k \alpha_T^k$$

$$\alpha_t^k = p(x_t \mid y_t^k = 1) \sum_i \alpha_{t-1}^i a_{i,k}$$

Shafer Shenoy for HMMs

- Recap: Shafer-Shenoy algorithm



  - Message from clique *i* to clique *j* :

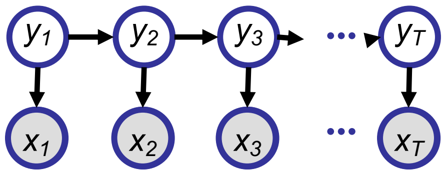
$$\mu_{i \to j} = \sum_{C_i \setminus S_{ij}} \psi_{C_i} \prod_{k \neq j} \mu_{k \to i}(S_{ki})$$

  - Clique marginal

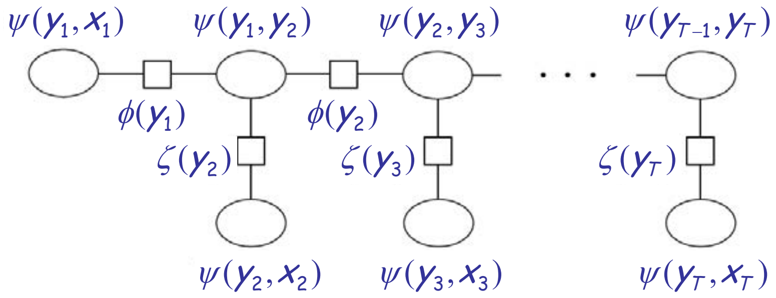
$$p(C_i) \propto \psi_{C_i} \prod_{k} \mu_{k \to i}(S_{ki})$$

Message Passing for HMMs (cont.)

- A junction tree for the HMM



$\Rightarrow$

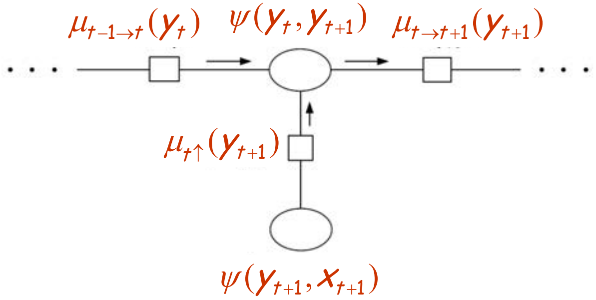


- Rightward pass

$$\mu_{t\to t+1}(y_{t+1}) = \sum_{y_t} \psi(y_t, y_{t+1})\mu_{t-1\to t}(y_t)\mu_{t\uparrow}(y_{t+1})$$

$$= \sum_{y_t} p(y_{t+1} \mid y_t)\mu_{t-1\to t}(y_t)p(x_{t+1} \mid y_{t+1})$$

$$= p(x_{t+1} \mid y_{t+1})\sum a_{y_t, y_{t+1}}\mu_{t-1\to t}(y_t)$$

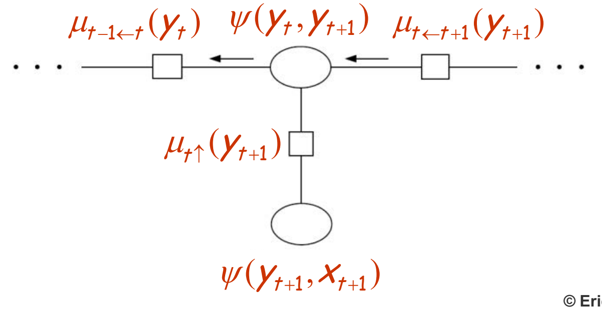


  - This is exactly the *forward algorithm*!

- Leftward pass …

$$\mu_{t-1\leftarrow t}(y_t) = \sum_{y_{t+1}} \psi(y_t, y_{t+1})\mu_{t\leftarrow t+1}(y_{t+1})\mu_{t\uparrow}(y_{t+1})$$

$$= \sum_{y_{t+1}} p(y_{t+1} \mid y_t)\mu_{t\leftarrow t+1}(y_{t+1})p(x_{t+1} \mid y_{t+1})$$



  - This is exactly the *backward algorithm*!

Summary

- Junction tree data-structure for exact inference on general graphs
- Two methods
  - Shafer-Shenoy
  - Belief-update or Lauritzen-Speigelhalter
- Constructing Junction tree from chordal graphs
  - Maximum spanning tree approach