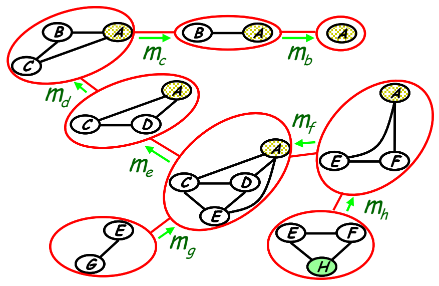
Machine Learning

Exact Inference

Eric Xing

Lecture 11, August 14, 2010



Reading:

Inference and Learning

- We now have compact representations of probability distributions: **GM**
- A BN *M* describes a unique probability distribution *P*
- Typical tasks:
  - Task 1: How do we answer **queries** about P?
    - We use **inference** as a name for the process of computing answers to such queries
  - Task 2: How do we estimate a **plausible model** $\mathcal{M}$ from data $\mathcal{D}$?

    i. We use **learning** as a name for the process of obtaining point estimate of $\mathcal{M}$.

    ii. But for *Bayesian*, they seek $p(\mathcal{M}\,|\,\mathcal{D})$, which is actually an **inference** problem.

    iii. When not all variables are observable, even computing point estimate of $\mathcal{M}$ need to do **inference** to impute the *missing data*.

Inferential Query 1: Likelihood

- Most of the queries one may ask involve **evidence**
  - Evidence $\mathbf{x}_{\mathrm{v}}$ is an assignment of values to a set $\mathbf{X}_{\mathrm{v}}$ of nodes in the GM over varialbe set $\mathbf{X}=\{X_1, X_2, \ldots, X_{\mathrm{n}}\}$
  - Without loss of generality $\mathbf{X}_{\mathrm{v}}=\{X_{k+1}, \ldots, X_{\mathrm{n}}\}$,
  - Write $\mathbf{X}_{\mathrm{H}}=\mathbf{X}\backslash\mathbf{X}_{\mathrm{v}}$ as the set of hidden variables, $\mathbf{X}_{\mathrm{H}}$ can be $\varnothing$ or $\mathbf{X}$

- Simplest query: compute probability of evidence

$$P(\mathbf{x}_{\mathrm{v}}) = \sum_{\mathbf{x}_{\mathrm{H}}} P(\mathbf{X}_{\mathrm{H}},, \mathbf{X}_{\mathrm{v}}) = \sum_{x_1} \cdots \sum_{x_k} P(x_1, \ldots, x_k, \mathbf{x}_{\mathrm{v}})$$

  - this is often referred to as computing the **likelihood** of $\mathbf{x}_{\mathrm{v}}$

Inferential Query 2: Conditional Probability

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

$$P(\mathbf{X}_\mathrm{H} \mid \mathbf{X}_\mathrm{V} = \mathbf{x}_\mathrm{V}) = \frac{P(\mathbf{X}_\mathrm{H}, \mathbf{x}_\mathrm{V})}{P(\mathbf{x}_\mathrm{V})} = \frac{P(\mathbf{X}_\mathrm{H}, \mathbf{x}_\mathrm{V})}{\sum_{\mathbf{x}_\mathrm{H}} P(\mathbf{X}_\mathrm{H} = \mathbf{x}_\mathrm{H}, \mathbf{x}_\mathrm{V})}$$

  - this is the ***a posteriori*** **belief** in $\mathbf{X}_\mathrm{H}$, given evidence $\mathbf{x}_\mathrm{v}$

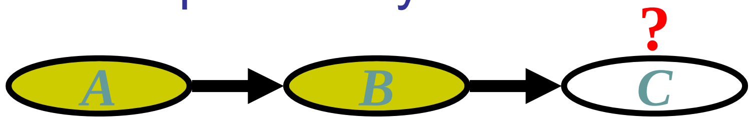
- We usually query a subset $\mathbf{Y}$ of all hidden variables $\mathbf{X}_\mathrm{H}=\{\mathbf{Y},\mathbf{Z}\}$ and "don't care" about the remaining, $\mathbf{Z}$:

$$P(\mathbf{Y} \mid \mathbf{x}_\mathrm{V}) = \sum_{\mathbf{z}} P(\mathbf{Y}, \mathbf{Z} = \mathbf{z} \mid \mathbf{x}_\mathrm{V})$$

  - the process of summing out the "don't care" variables *z* is called **marginalization**, and the resulting $P(\mathbf{Y}|\mathbf{x}_\mathrm{v})$ is called a **marginal** prob.

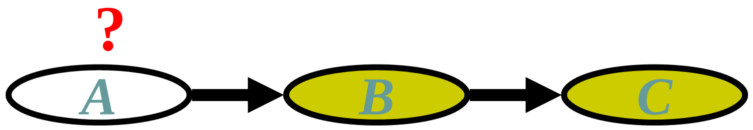
Applications of *a posteriori* Belief

- **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

Inferential 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 $\mathbf{x}_V$, and ignoring (the values of) other variables $\mathbf{Z}$:

$$\mathbf{Y}^* \mid \mathbf{x}_V = \arg\max_{\mathbf{y}} P(\mathbf{Y} \mid \mathbf{x}_V) = \arg\max_{\mathbf{y}} \sum_{\mathbf{z}} P(\mathbf{Y}, \mathbf{Z} = \mathbf{z} \mid \mathbf{x}_V)$$

  - this is the **maximum *a posteriori*** configuration of $\mathbf{Y}$.

Complexity of Inference

Thm:

Computing $P(\mathbf{X}_H=\mathbf{x}_H | \mathbf{x}_v)$ in an arbitrary 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
  - Belief propagation
  - The junction tree algorithms (but will not cover in detail here)
- Approximate inference techniques
  - Variational algorithms
  - Stochastic simulation / sampling methods
  - Markov chain Monte Carlo methods

Inference on General BN via Variable Elimination

General idea:

- Write query in the form

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

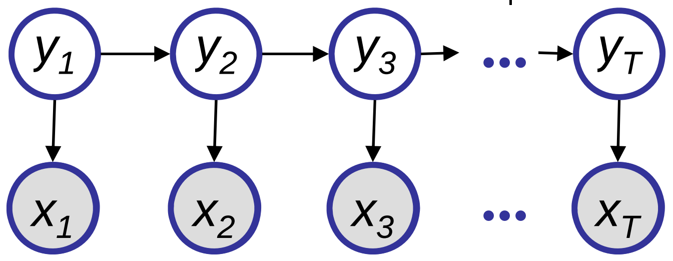
  - this suggests an "elimination order" of latent variables to be marginalized

- 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

$$P(X_1 \mid \boldsymbol{e}) = \frac{P(X_1, \boldsymbol{e})}{P(\boldsymbol{e})}$$

Hidden Markov Model



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

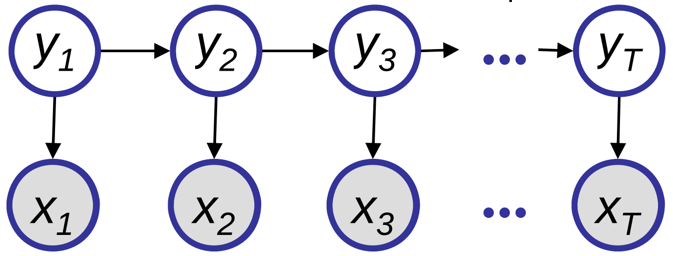
These are known as "forward message"

Conditional probability:

$$p(y_i|x_1,\ldots,x_T) = \sum_{y_1}\cdots\sum_{y_{i-1}}\sum_{y_{i+1}}\cdots\sum_{y_T} p(y_i,\ldots,y_T,x_1,\ldots,x_T)$$
$$= \sum_{y_1}\cdots\sum_{y_{i-1}}\sum_{y_{i+1}}\cdots\sum_{y_T} p(y_1)p(x_1|y_1)\ldots p(y_T|y_{T-1})p(x_T|y_T)$$
$$= \sum_{y_2}\cdots\sum_{y_T} P(x_2|y_2)P(y_3|y_2)\cdots\cdots \sum_{y_1} P(y_1)P(x_1|y_1)P(y_2|y_1)$$
$$\searrow P(x_1, y_2)$$
$$= \sum_{y_3}\cdots\sum_{y_T} P(x_3|y_3)P(y_4|y_3)\cdots\cdots \sum_{y_2} P(x_1,y_2)P(x_2|y_2)P(y_3|y_2)$$
$$\searrow P(x_1, x_2, y_3)$$

Hidden Markov Model

Conditional probability:



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

$$= \sum_{y_1}\cdots\sum_{y_{T-1}} P(y_1)\cdots P(x_{T-1}|y_{T-1}) \sum_{y_T} P(y_T|y_{T-1})P(x_T|y_T)$$

$$\rightarrow P(x_T|y_{T-1})$$

$$= \sum_{y_1}\cdots\sum_{y_{T-2}} P(y_1)\cdots P(x_{T-2}|y_{T-2}) \sum_{y_{T-1}} P(y_{T-1}|y_{T-2})P(x_{T-1}|y_{T-1})\, P(x_T|y_{T-1})$$

$$\rightarrow P(x_T, x_{T-1}|y_{T-2})$$

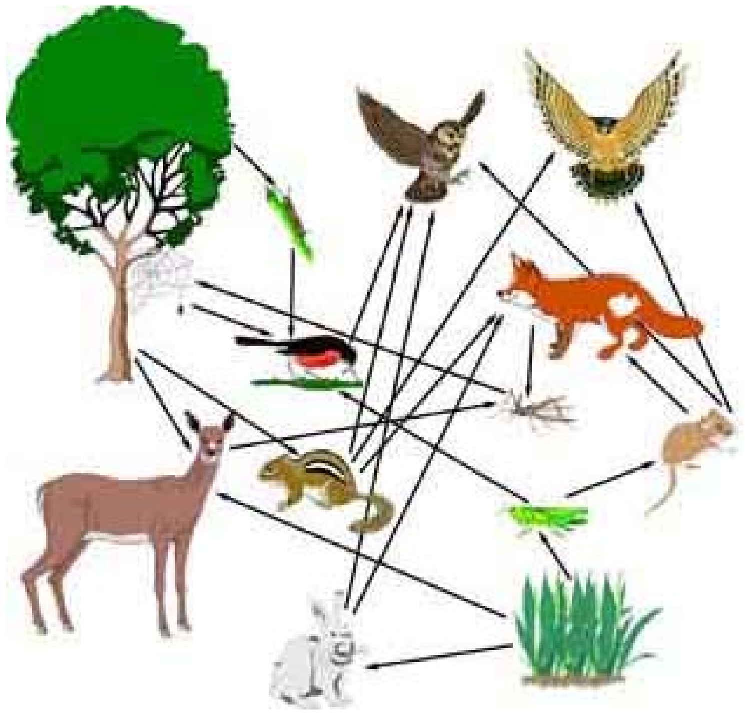
$$= \cdots\cdots$$

$$= \cdots$$

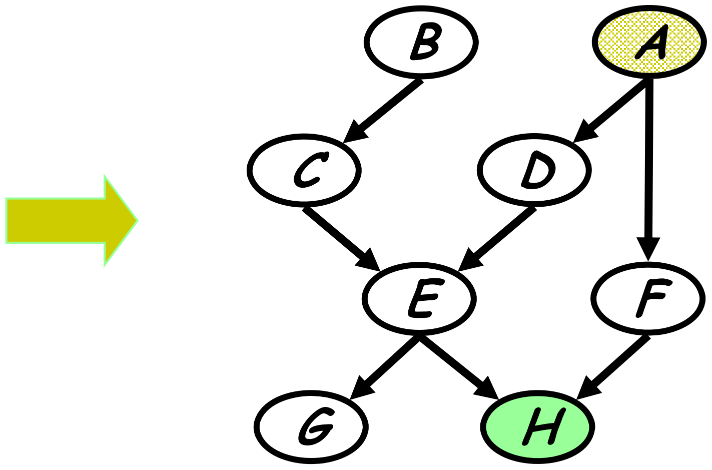
These are known as "backward message"

A Bayesian network

A food web



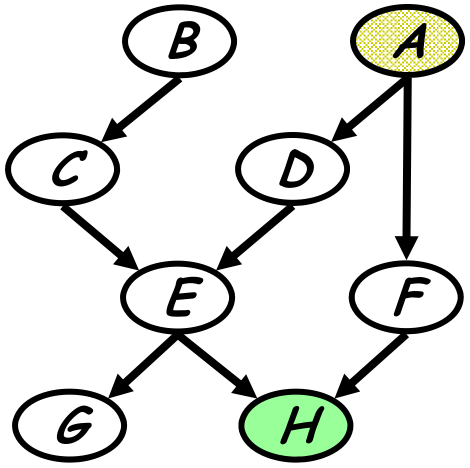
$$\begin{tikzcd} & B \arrow[dl] & & A \arrow[dl] \arrow[d] \\ C \arrow[dr] & & D \arrow[dl] & F \arrow[dl] \\ & E \arrow[dl] \arrow[r] & H & \\ G & & & \end{tikzcd}$$



What is the probability that hawks are leaving given that the grass condition is poor?

Example: Variable Elimination

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



- Initial factors:

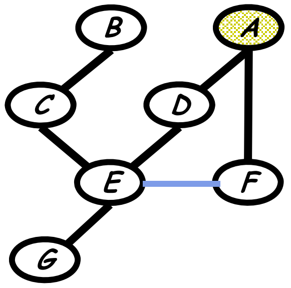
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$

- 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:



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

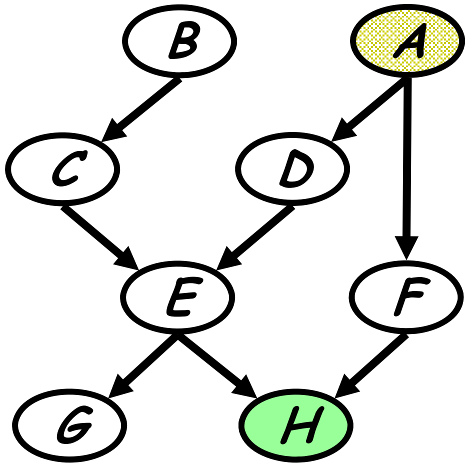
Example: Variable Elimination

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

- Initial factors:

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

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

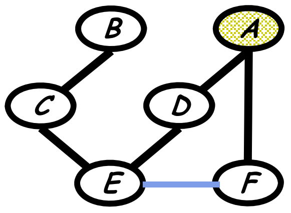


- Step 2: Eliminate *G*
  - compute

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

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

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

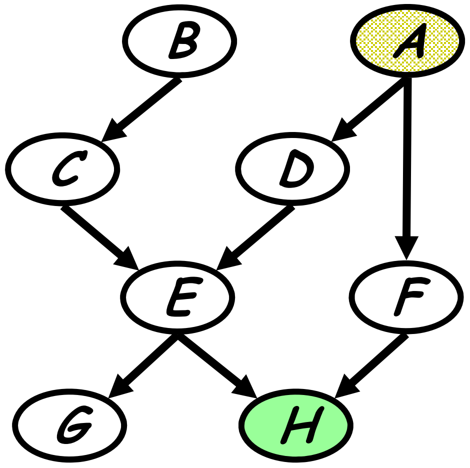


Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B,C,D,E,F*

- Initial factors:

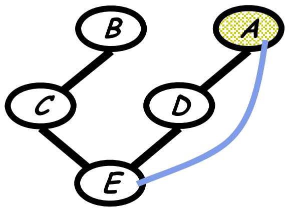
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)\underline{P(f|a)m_h(e,f)}$$



- Step 3: Eliminate *F*
  - compute

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

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

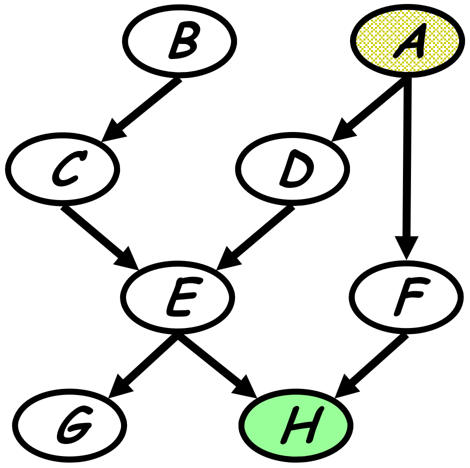


Example: Variable Elimination

- Query: *P(B | h)*
  - Need to eliminate: *B,C,D,E*

- Initial factors:

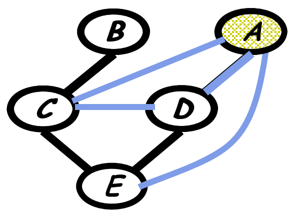
$$P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)P(h|e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)P(g|e)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)P(e|c,d)P(f|a)m_h(e,f)$$
$$\Rightarrow P(a)P(b)P(c|b)P(d|a)\underline{P(e|c,d)m_f(a,e)}$$



- Step 4: Eliminate *E*
  - compute

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

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

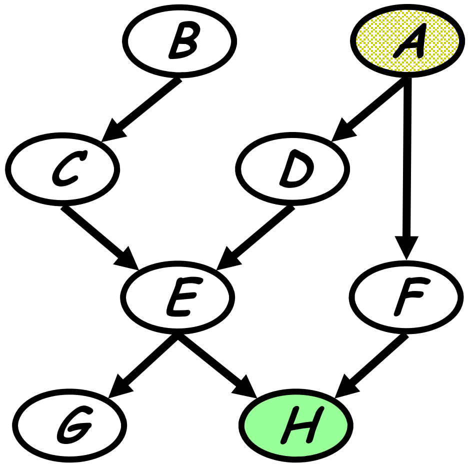


Example: Variable Elimination

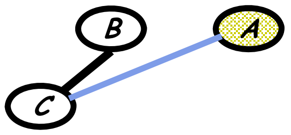
- Query: $P(B\,|\,h)$
  - Need to eliminate: B,C,D

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

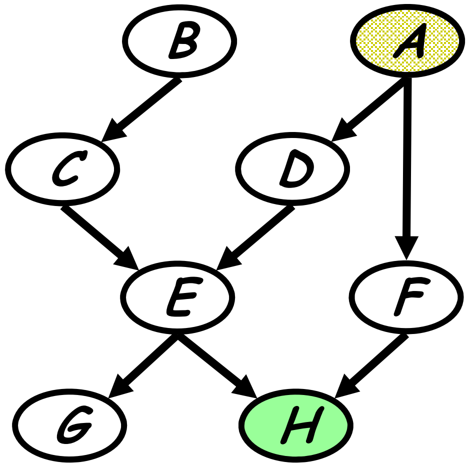


$$\Rightarrow P(a)P(b)P(c|d)\underline{m_d(a,c)}$$

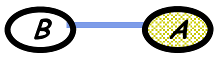
Example: Variable Elimination

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

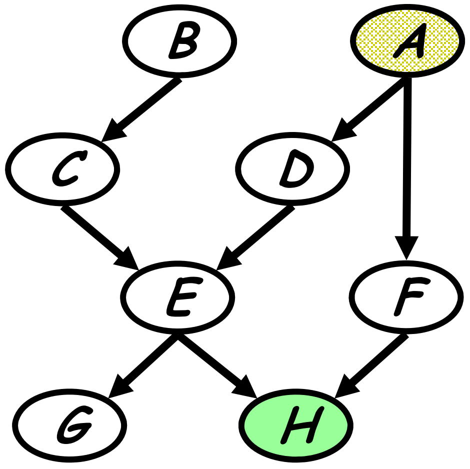


$$\Rightarrow P(a)P(b)P(c \mid d)\underline{m_d(a,c)}$$

Example: Variable Elimination

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

- Initial factors:



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

- Step 7: Eliminate B

  - compute

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

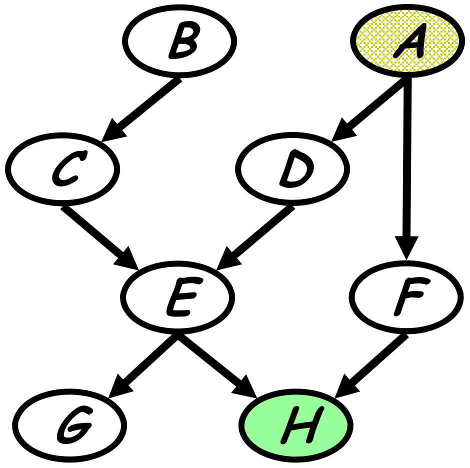
$$\Rightarrow P(a)\underline{m_b(a)}$$

Example: Variable Elimination

- 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

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

Complexity of variable elimination

- Suppose in one elimination step we compute

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

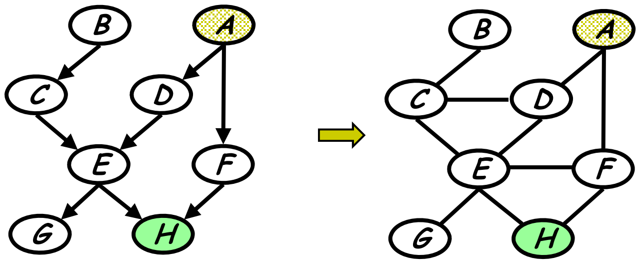
$$m'_x(x, y_1,\dots,y_k) = \prod_{i=1}^{k} m_i(x, \mathbf{y}_{c_i})$$

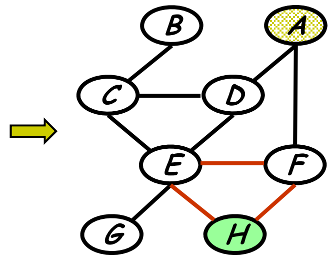
This requires

- $k \bullet |\mathrm{Val}(X)| \bullet \prod_i |\mathrm{Val}(\mathbf{Y}_{C_i})|$ multiplications
  - For each value of *x, y₁, …, yₖ*, we do *k* multiplications
- $|\mathrm{Val}(X)| \bullet \prod_i |\mathrm{Val}(\mathbf{Y}_{C_i})|$ additions
  - For each value of $y_1, \dots, y_k$, we do $|Val(X)|$ additions

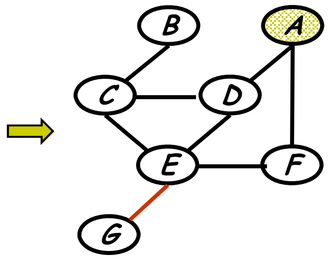
Complexity is **exponential** in number of variables in the intermediate factor

Elimination Cliques

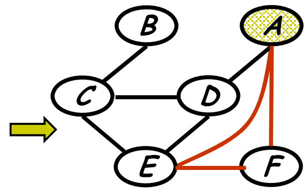




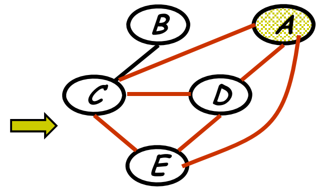
$m_h(e, f)$



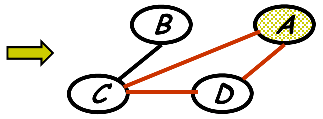
$m_g(e)$



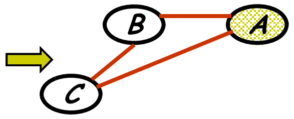
$m_f(e, a)$



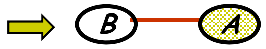
$m_e(a, c, d)$



$m_d(a, c)$



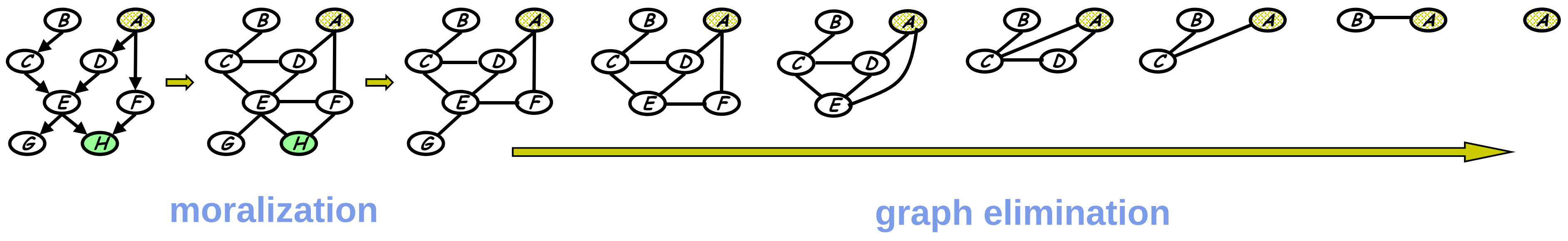
$m_c(a, b)$



$m_b(a)$

Understanding Variable Elimination

- A graph elimination algorithm

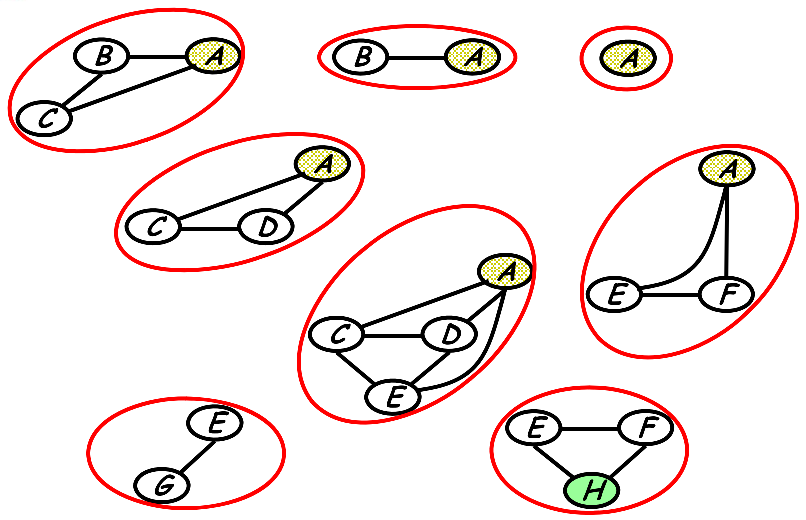


- Intermediate terms correspond to the cliques resulted from elimination
  - “good” elimination orderings lead to **small cliques** and hence reduce complexity (what will happen if we eliminate "e" first in the above graph?)
  - finding the optimum ordering is NP-hard, but for many graph optimum or near-optimum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

From Elimination to Belief Propagation

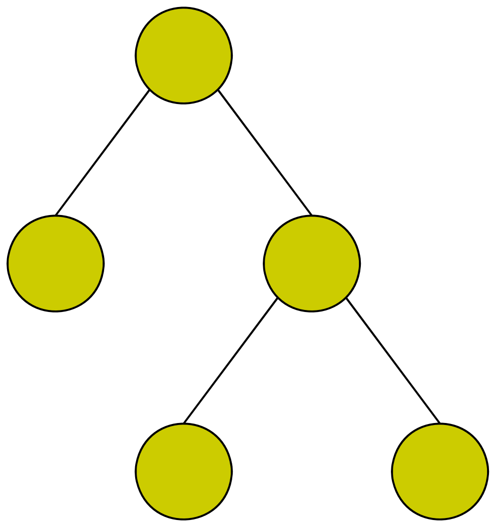
- Recall that Induced dependency during marginalization is captured in elimination cliques
  - Summation <-> elimination
  - Intermediate term <-> elimination clique

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

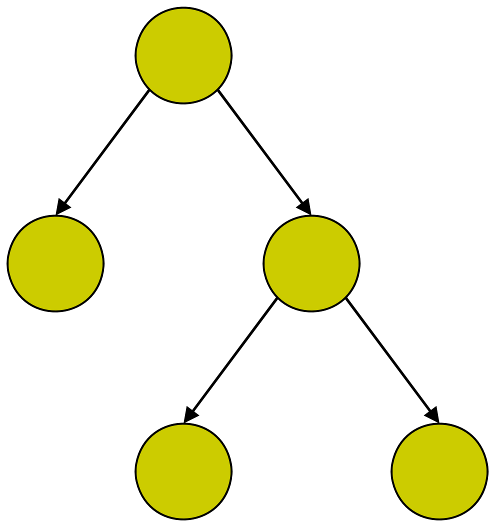


  - Can this lead to an generic inference algorithm?

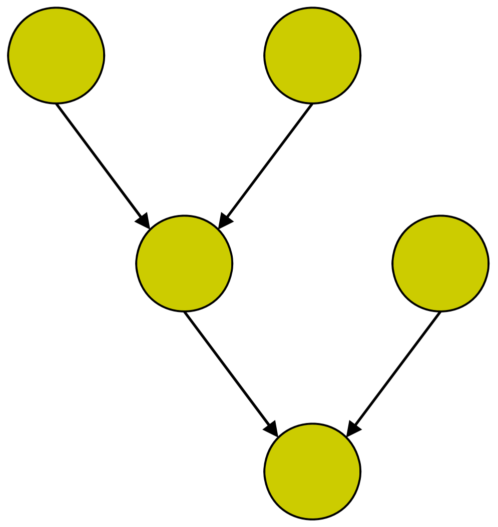
Tree GMs



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

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:?**

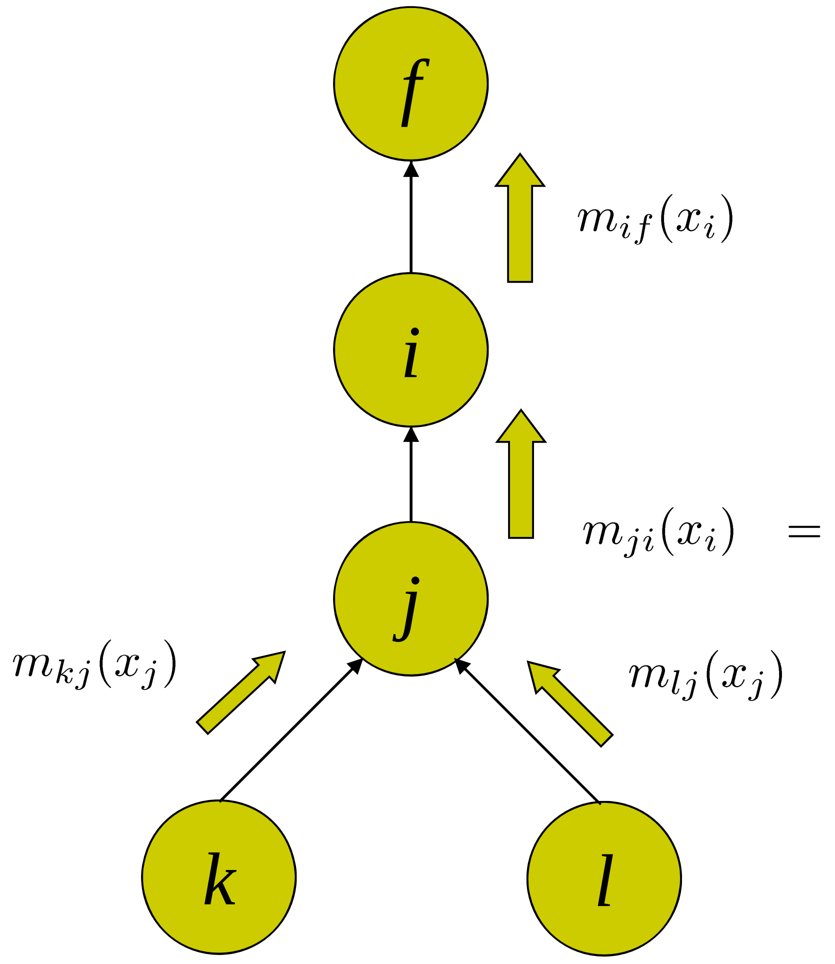
From elimination to message passing

- Recall ELIMINATION algorithm:
  - Choose an ordering $\mathcal{Z}$ 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!!

Message passing for trees



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

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

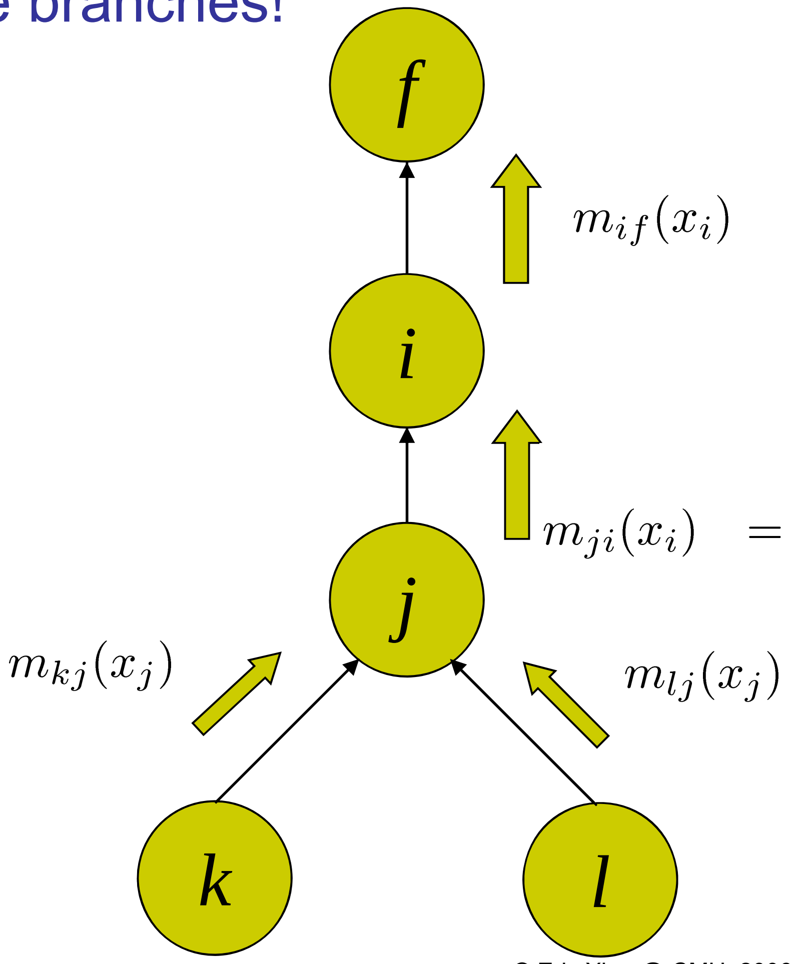
$$m_{ji}(x_i) = \sum_{x_j}\left(\psi(x_j)\psi(x_i,x_j)\prod_{k\in N(j)\backslash i} m_{kj}(x_j)\right)$$

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

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

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

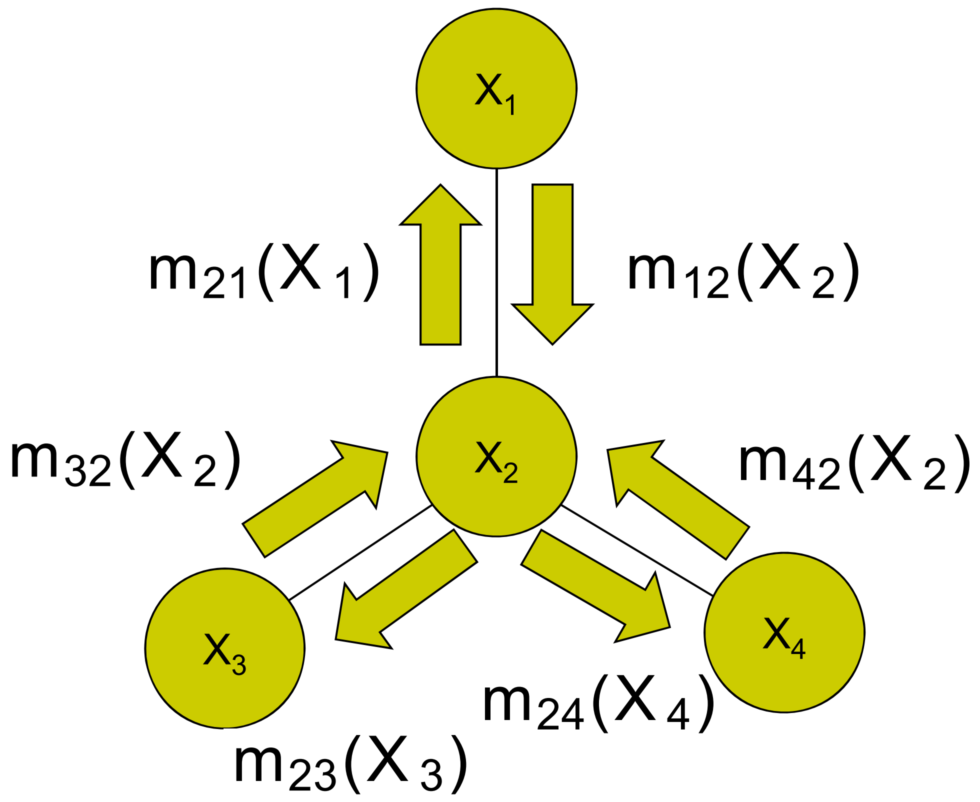
- Elimination on trees is equivalent to message passing along tree branches!



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

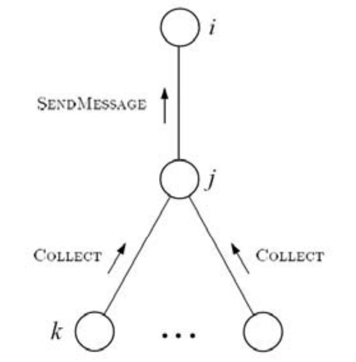
The message passing protocol:

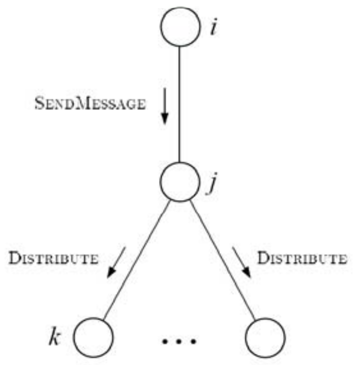
- A two-pass algorithm:



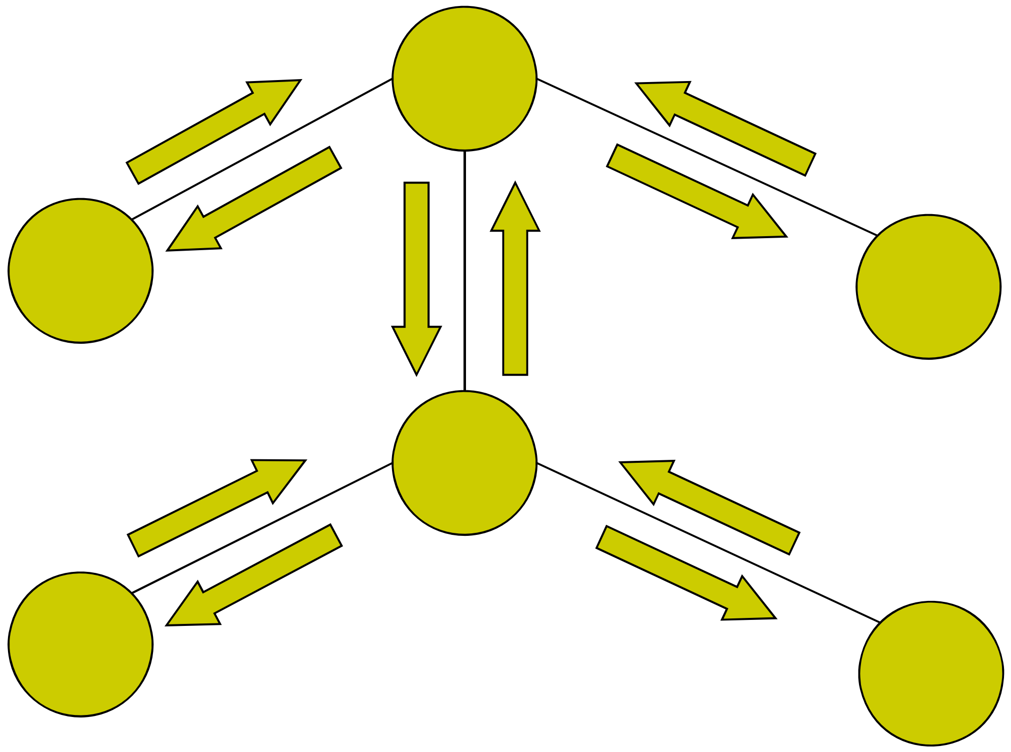
Belief Propagation (SP-algorithm): Sequential implementation

```
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

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

- What about non-tree?

Inference on general GM

- Now, what if the GM is not a tree-like graph?
- Can we still directly run message

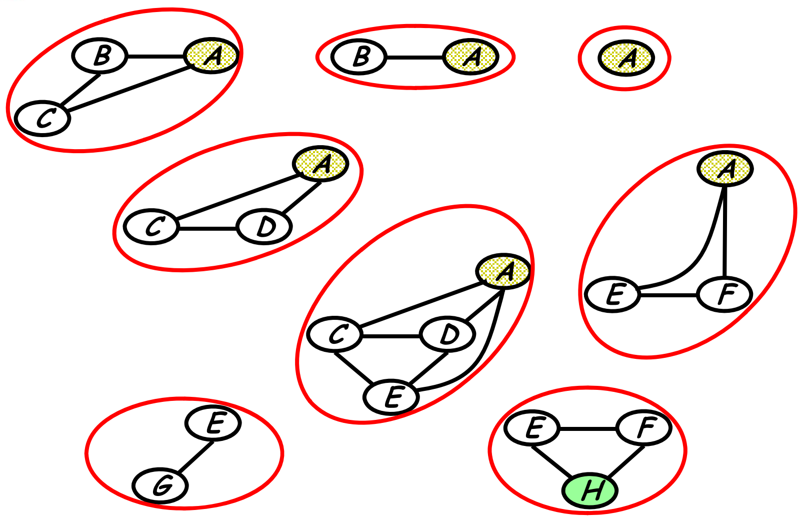
  message-passing protocol along its edges?
- For non-trees, we do not have the guarantee that message-passing will be consistent!
- Then what?
  - Construct a graph data-structure from P that has a tree structure, and run message-passing on it!

$\rightarrow$ Junction tree algorithm

Elimination Clique

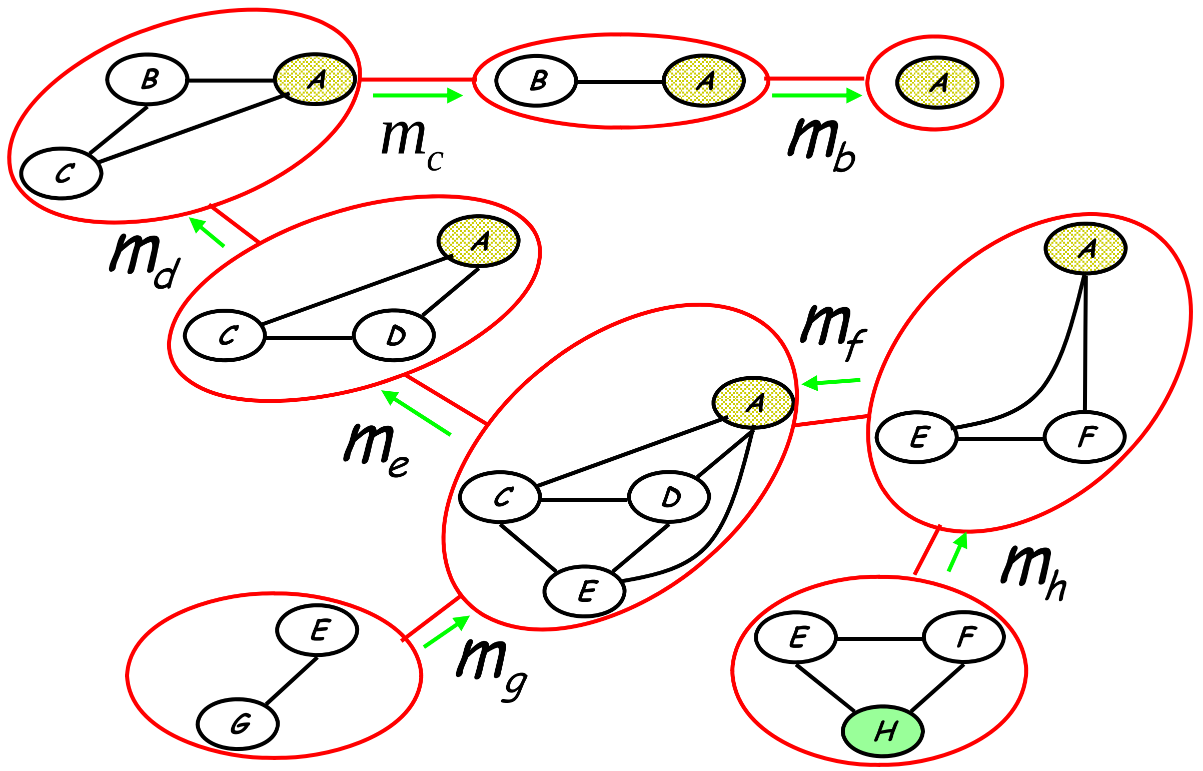
- Recall that Induced dependency during marginalization is captured in elimination cliques
    - Summation <-> elimination
    - Intermediate term <-> elimination clique

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



- Can this lead to an generic inference algorithm?

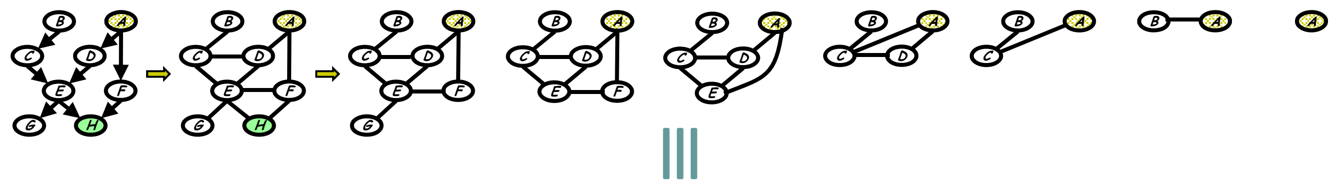
A Clique Tree

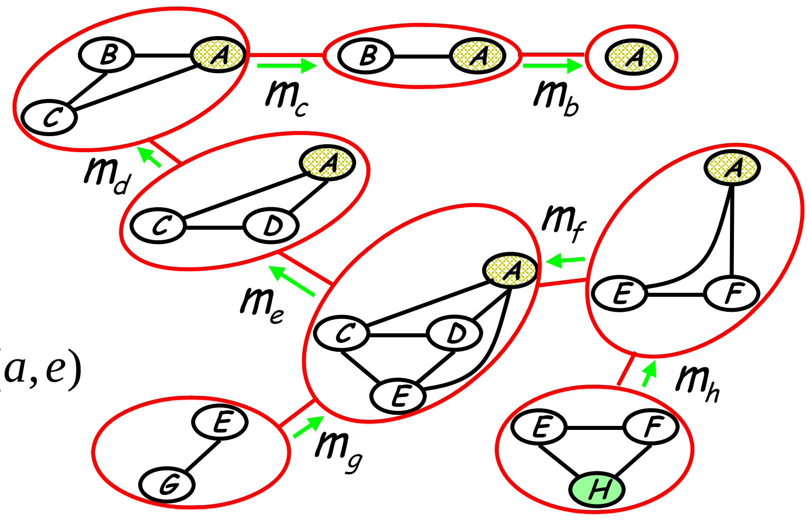


$$
\begin{aligned}
&m_e(a,c,d) \\
&= \sum_e p(e \mid c,d)\, m_g(e)\, m_f(a,e)
\end{aligned}
$$

From Elimination to Message Passing

- Elimination ≡ message passing on a **clique tree**



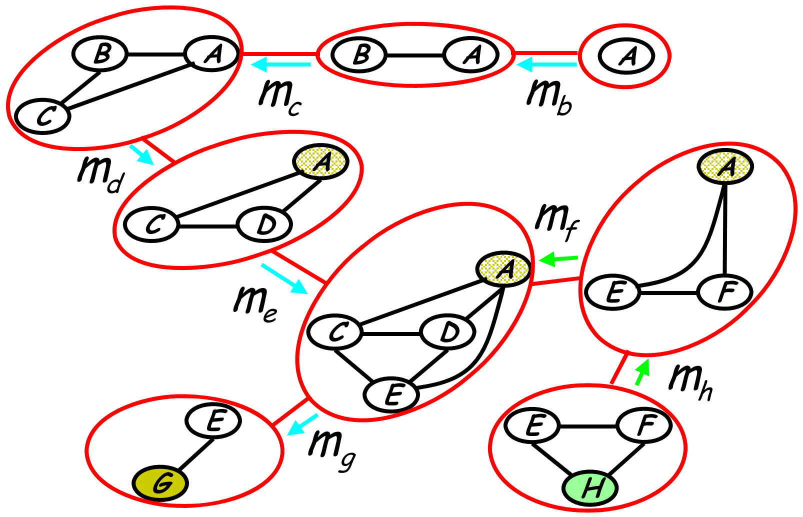


$$m_e(a,c,d) = \sum_e p(e|c,d)m_g(e)m_f(a,e)$$

- Messages can be reused

From Elimination to Message Passing

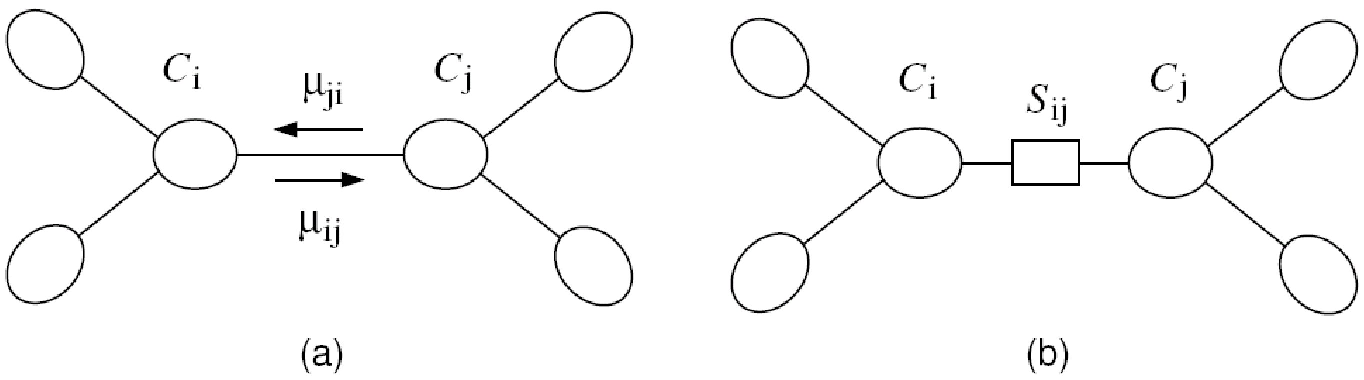
- Elimination ≡ message passing on a **clique tree**
  - **Another query ...**



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

The Shafer Shenoy Algorithm

- 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

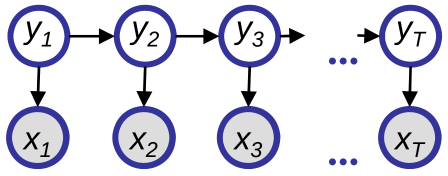
$$p(C_i) \propto \psi_{C_i} \prod_{k} \mu_{k \to i}(S_{ki})$$

A Sketch of the Junction Tree Algorithm

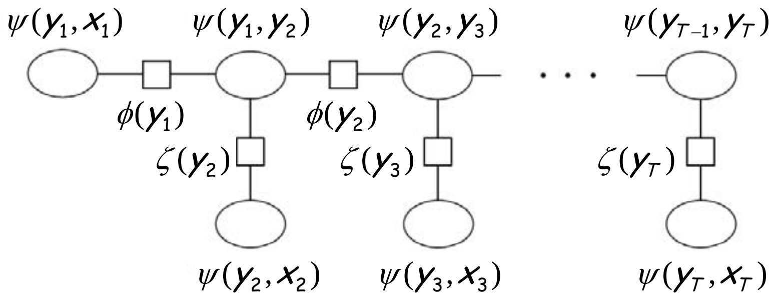
- **The algorithm**
  - Construction of junction trees --- a special **clique tree**
  - Propagation of probabilities --- a message-passing protocol
- Results in marginal probabilities of all cliques --- solves all queries in a single run
- A **generic** exact inference algorithm for any GM
- **Complexity**: 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
- Many well-known algorithms are special cases of JT
  - Forward-backward, Kalman filter, Peeling, Sum-Product ...

The Junction tree algorithm for HMM

- A junction tree for the HMM



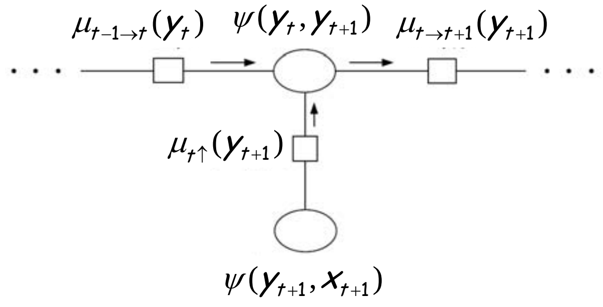
$\Rightarrow$



- Rightward pass

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

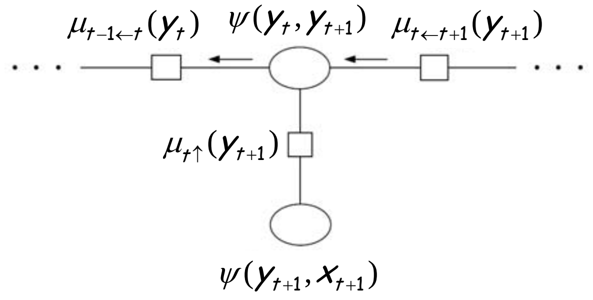
  - This is exactly the *forward algorithm*!



- 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*!



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

Summary

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

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

- 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 algorithms

- 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.