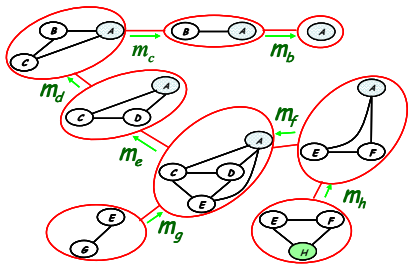
School of Computer Science
Carnegie Mellon

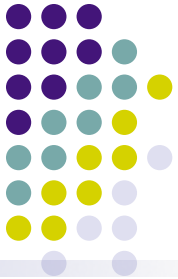
Probabilistic Graphical Models

Exact Inference:
Elimination and Message Passing
The Sum Product Algorithm



Eric Xing
Reading: MJ-chap 3,4
Lecture 7, February 4, 2015

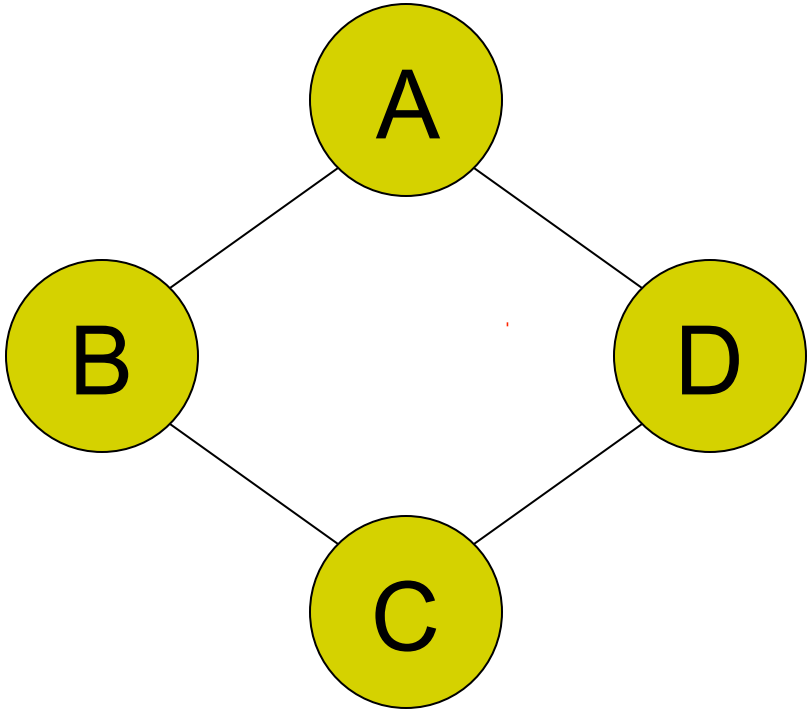
Recap:



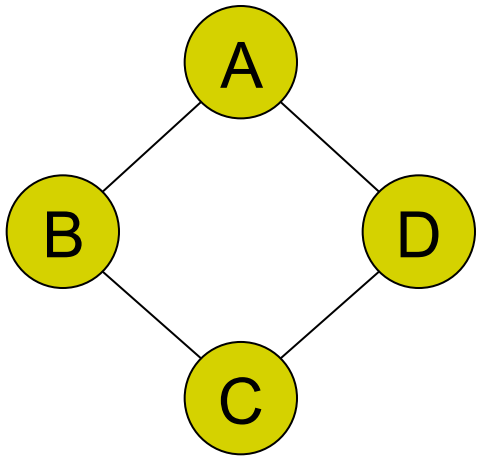
- Defn: A DAG G is a **perfect map** (P-map) for a distribution P if $I(P)=I(G)$.

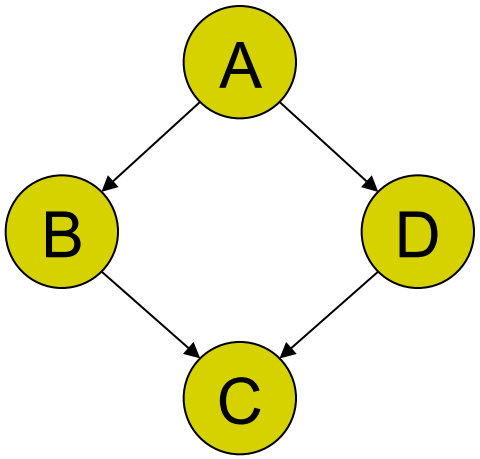
Question: Is there a BN that is a perfect map for a given MN?

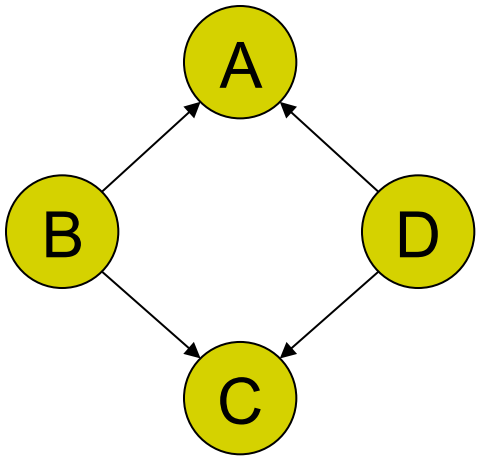
- The "diamond" MN



Question: Is there a BN that is a perfect map for a given MN?







$A \perp C \mid \{B,D\}$

$B \perp D \mid \{A,C\}$

$A \perp C \mid \{B,D\}$

$B \perp D \mid A$ ✗

$A \perp C \mid \{B,D\}$

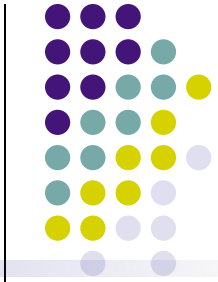
$B \perp D$ ✗

- This MN does not have a perfect I-map as BN!

Summary

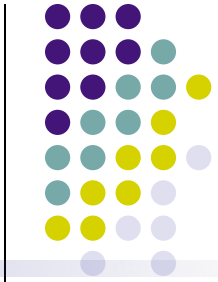
- Investigated the relationship between BNs and MNs
  - They represent different families of independence assumptions
- Not mentioned: Chain networks → superset of both BNs and MNs
- Why we care about this:
  - BN and MN offer different semantics for designer to capture or expression (conditional) independences among variables
  - Under certain condition BN can be represented as an MN and vice versa
    - Trees, Trangulated graphs
  - In the future, for certain operation (i.e., inference), we will be using a single representation as the "data structure" for which an algorithm can operate on.
  - This makes algorithm design, and analysis of the algorithms simpler

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*?
    i. We use **learning** as a name for the process of obtaining point estimate of *M*.
    ii. But for *Bayesian*, they seek $p(M|D)$, which is actually an **inference** problem.
    iii. When not all variables are observable, even computing point estimate of *M* need to do **inference** to impute the *missing data*.

Query 1: Likelihood



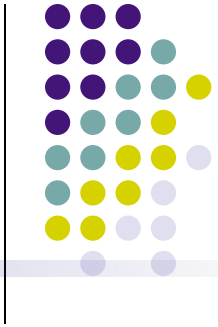
- Most of the queries one may ask involve **evidence**
  - Evidence $\mathbf{e}$ is an assignment of values to a set $\mathbf{E}$ variables in the domain
  - Without loss of generality $\mathbf{E} = \{ X_{k+1}, \ldots, X_n \}$

- 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 $\mathbf{e}$

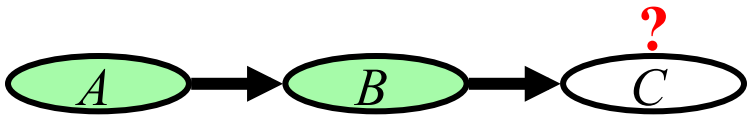
- We usually query a subset $\mathbf{Y}$ of all domain variables $\mathbf{X}=\{\mathbf{Y},\mathbf{Z}\}$ and "don't care" about the remaining, $\mathbf{Z}$:

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

  - 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.

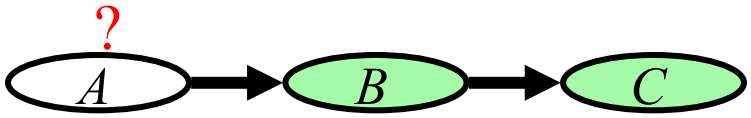
Applications of a posteriori Belief

- **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

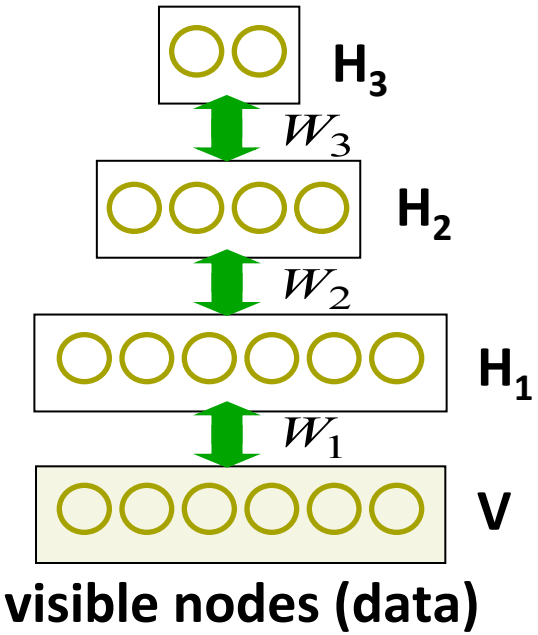


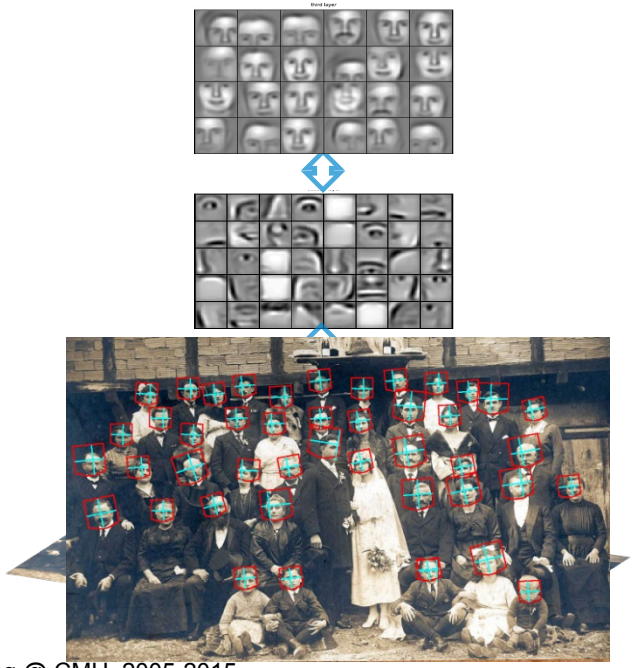
  - the query node an ancestor of the evidence

- **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

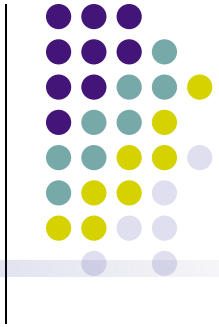
Example: Deep Belief Network

- 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



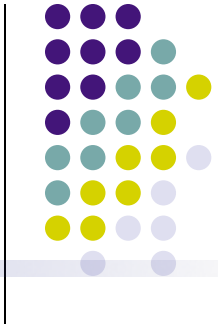
- In this query we want to find the **most probable joint assignment** (MPA) for *some* variables of interest

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

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

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

Applications of MPA



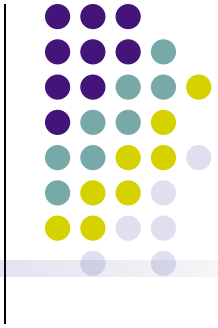
- Classification
  - find most likely label, given the evidence
- Explanation
  - what is the most likely scenario, given the evidence

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

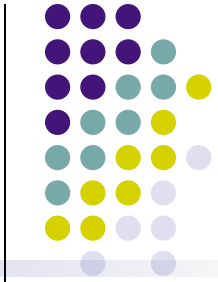


Thm:

Computing $P(\mathbf{X} = \mathbf{x} \mid \mathbf{e})$ 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

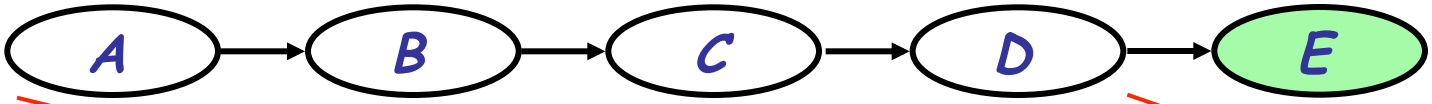


- Exact inference algorithms
  - The elimination algorithm
  - Message-passing algorithm (sum-product, belief propagation)
  - The junction tree algorithms

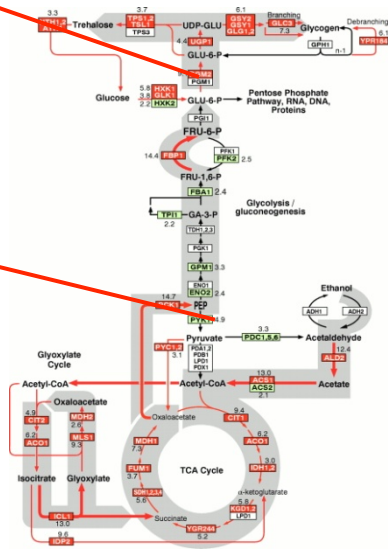
- Approximate inference techniques
  - Stochastic simulation / sampling methods
  - Markov chain Monte Carlo methods
  - Variational algorithms

Marginalization and Elimination

- A signal transduction pathway:



What is the likelihood that protein E is active?



- Query: $P(e)$

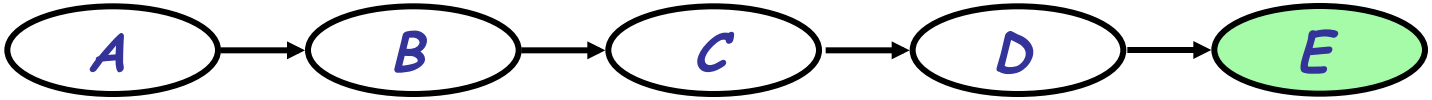
$$P(e) = \sum_d \sum_c \sum_b \sum_a P(a,b,c,d,e)$$

a naïve summation needs to enumerate over an exponential number of terms

- By chain decomposition, we get

$$= \sum_d \sum_c \sum_b \sum_a P(a)P(b \mid a)P(c \mid b)P(d \mid c)P(e \mid d)$$

Elimination on Chains

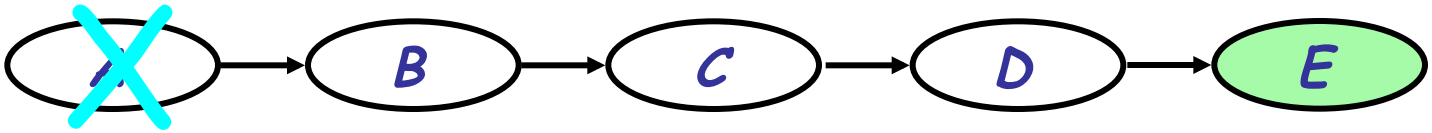


- Rearranging terms ...

$$P(e) = \sum_d \sum_c \sum_b \sum_a P(a)P(b|a)P(c|b)P(d|c)P(e|d)$$

$$= \sum_d \sum_c \sum_b P(c|b)P(d|c)P(e|d) \sum_a P(a)P(b|a)$$

Elimination on Chains

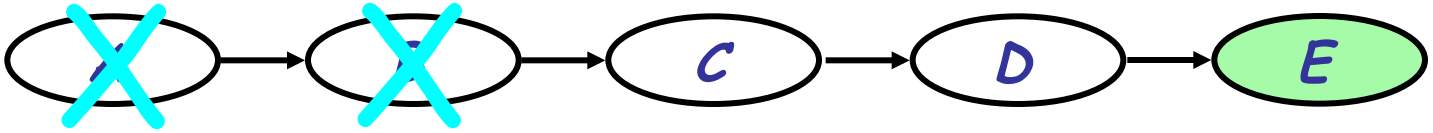


- Now we can perform innermost summation

$$P(e) = \sum_d \sum_c \sum_b P(c|b)P(d|c)P(e|d) \sum_a P(a)P(b|a)$$
$$= \sum_d \sum_c \sum_b P(c|b)P(d|c)P(e|d)\,p(b)$$

- This summation "eliminates" one variable from our summation argument at a "local cost".

Elimination in Chains



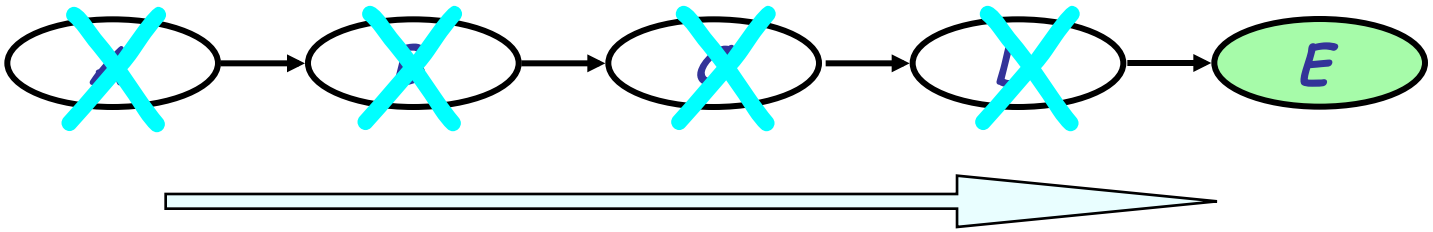
- Rearranging and then summing again, we get

$$P(e) = \sum_d \sum_c \sum_b P(c \mid b) P(d \mid c) P(e \mid d) p(b)$$

$$= \sum_d \sum_c P(d \mid c) P(e \mid d) \sum_b P(c \mid b) p(b)$$

$$= \sum_d \sum_c P(d \mid c) P(e \mid d) p(c)$$

Elimination in Chains

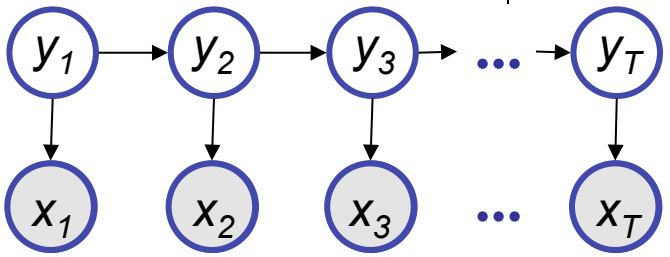


- Eliminate nodes one by one all the way to the end, we get

$$P(e) = \sum_{d} P(e \mid d)\, p(d)$$

- Complexity:
  - Each step costs $O(|Val(X_i)|*|Val(X_{i+1})|)$ operations: $O(nk^2)$
  - Compare to naïve evaluation that sums over joint values of $n-1$ variables $O(k^n)$

Hidden Markov Model

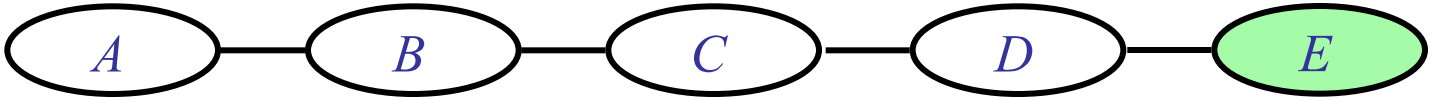


$$p(\mathbf{x}, \mathbf{y}) = p(x_1 \ldots\ldots x_T, y_1, \ldots\ldots, y_T)$$
$$= p(y_1)\, p(x_1 \mid y_1)\, p(y_2 \mid y_1)\, p(x_2 \mid y_2) \ldots p(y_T \mid y_{T-1})\, p(x_T \mid y_T)$$

Conditional probability:

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

Undirected Chains



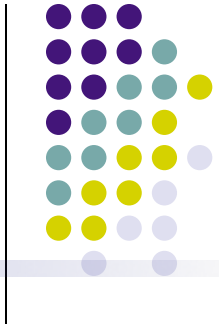
- Rearranging terms ...

$$P(e) = \sum_d \sum_c \sum_b \sum_a \frac{1}{Z} \phi(b,a)\phi(c,b)\phi(d,c)\phi(e,d)$$

$$= \frac{1}{Z} \sum_d \sum_c \sum_b \phi(c,b)\phi(d,c)\phi(e,d) \sum_a \phi(b,a)$$

$$= \cdots$$

The Sum-Product Operation



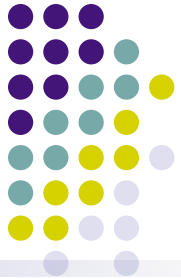
- In general, we can view the task at hand as that of computing the value of an expression of the form:

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

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

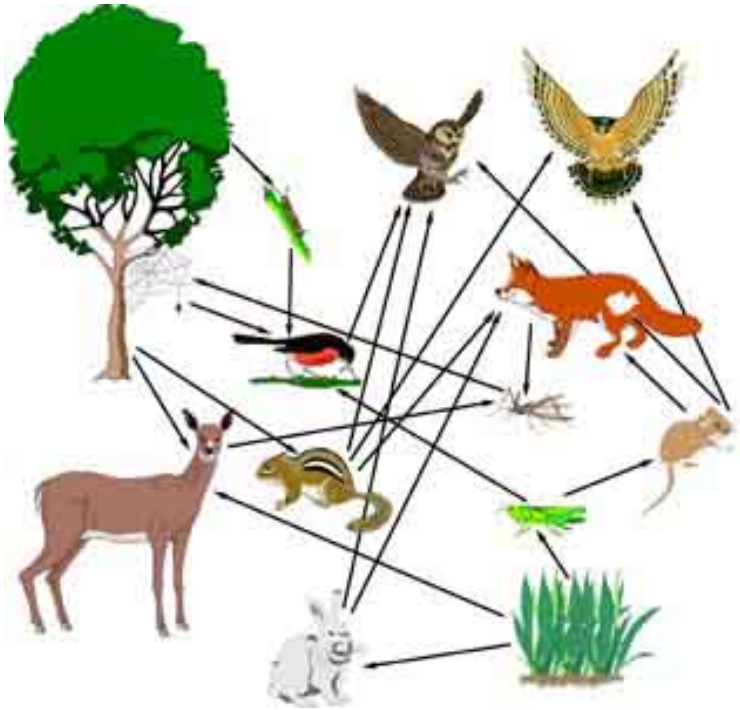
- Iteratively
  - Move all irrelevant terms outside of innermost sum
  - Perform innermost sum, getting a new term
  - Insert the new term into the product

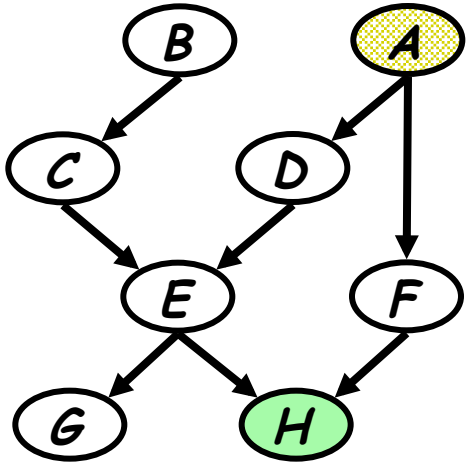
- wrap-up

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

A more complex network

A food web





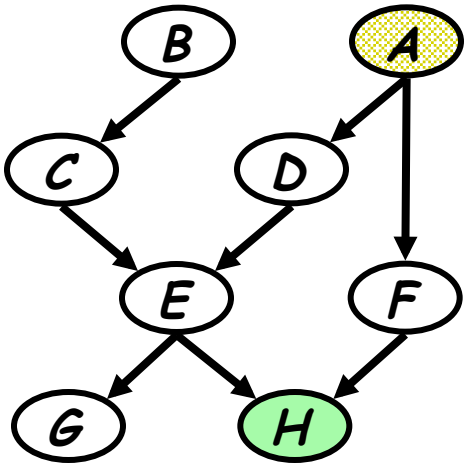
What is the probability that hawks are leaving given that the grass condition is poor?

Example: Variable Elimination

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

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

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

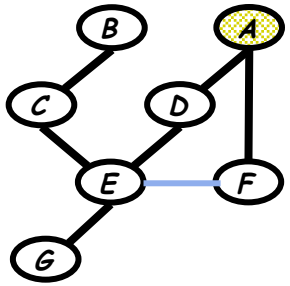


- Step 1:
  - **Conditioning** (fix the evidence node (i.e., h) on its observed value (i.e., $\tilde{h}$)):

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

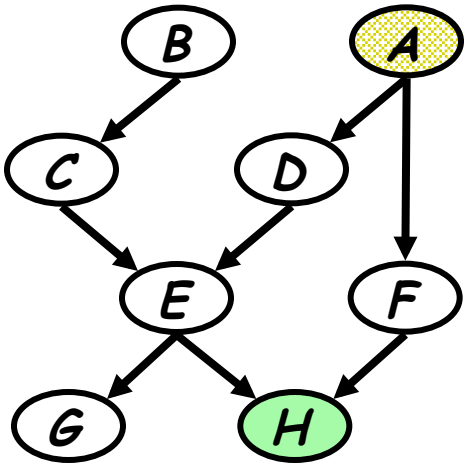
  - This step is isomorphic to a marginalization step:

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



Example: Variable Elimination

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

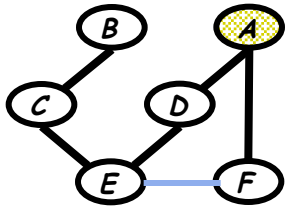


- Initial factors:

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

- Step 2: Eliminate G
- compute

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

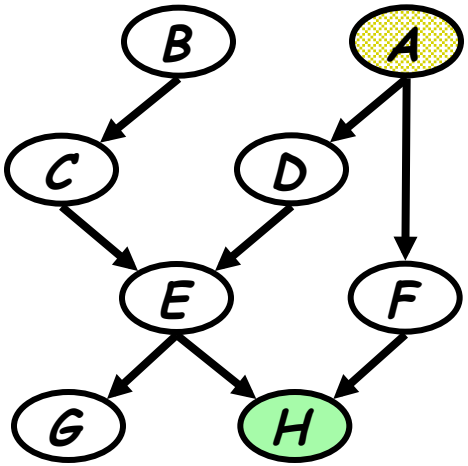


$$\Rightarrow P(a)P(b)P(c \mid b)P(d \mid a)P(e \mid c,d)P(f \mid a)m_g(e)m_h(e,f)$$
$$= P(a)P(b)P(c \mid b)P(d \mid a)P(e \mid c,d)P(f \mid a)m_h(e,f)$$

- Keep eliminating F,E,D,C,B in order

Example: Variable Elimination

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



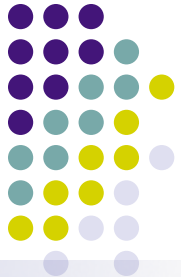
- Initial factors:

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

- Final Step: Wrap-up $p(a,\tilde{h}) = p(a)m_b(a), \quad p(\tilde{h}) = \sum_a p(a)m_b(a)$

$$\Rightarrow P(a \mid \tilde{h}) = \frac{p(a)m_b(a)}{\sum p(a)m_b(a)}$$

Complexity of variable elimination



- Suppose in one elimination step we compute

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

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

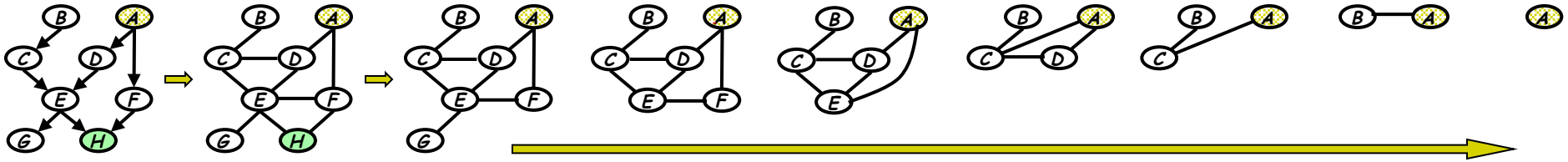
This requires

- $k \cdot |\mathrm{Val}(X)| \cdot \prod_i |\mathrm{Val}(\mathbf{y}_{C_i})|$ multiplications
  - For each value for $x, y_1, \ldots, y_k$, we do k multiplications
- $|\mathrm{Val}(X)| \cdot \prod_i |\mathrm{Val}(\mathbf{y}_{C_i})|$ 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

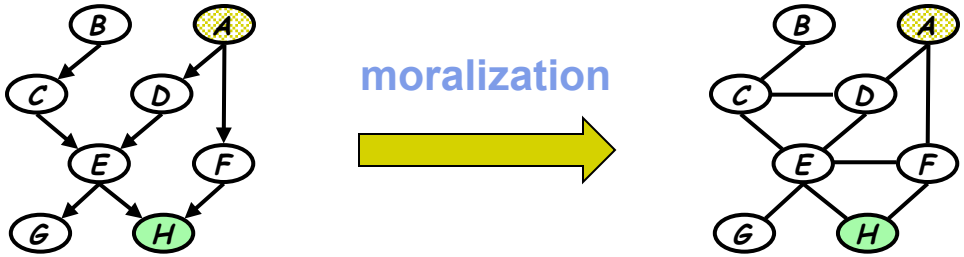
- A graph elimination algorithm



Moralization

- Convert from a directed acyclic graph (DAG) to equivalent undirected graph

- Moralization procedure
  - Starting from an input DAG
  - Connect nodes if they share a common child
  - Make directed edges to undirected edges

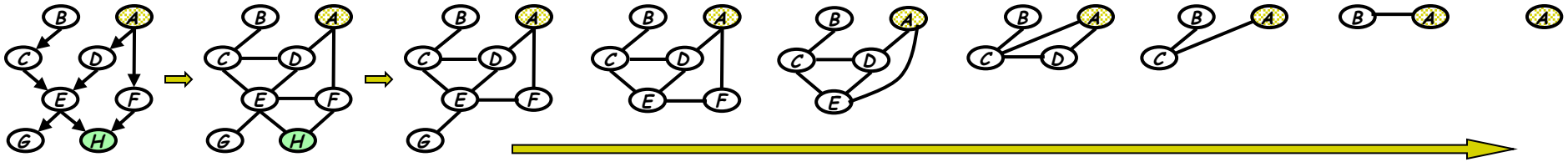


Graph elimination

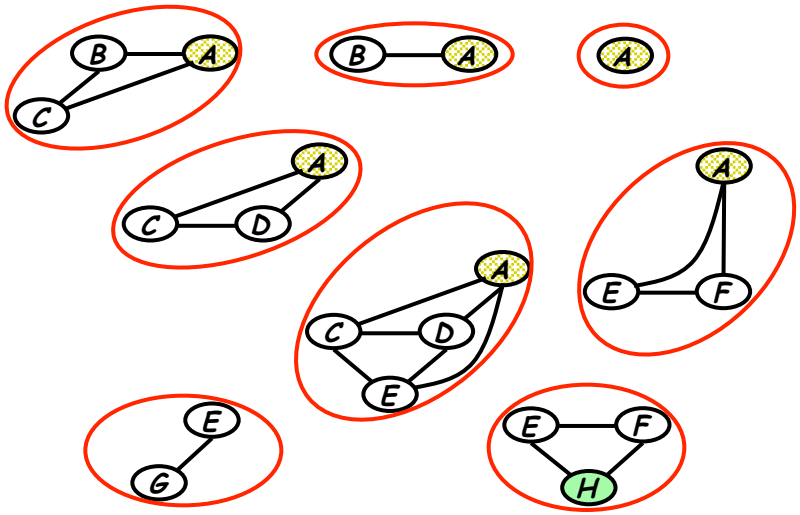
- Begin with the undirected GM or moralized BN
- Graph $G(V, E)$ and elimination ordering I
- Eliminate next node in the ordering I
  - Removing the node from the graph
  - Connecting the remaining neighbors of the nodes
- The reconstituted graph $G'(V, E')$
  - Retain the edges that were created during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

Understanding Variable Elimination

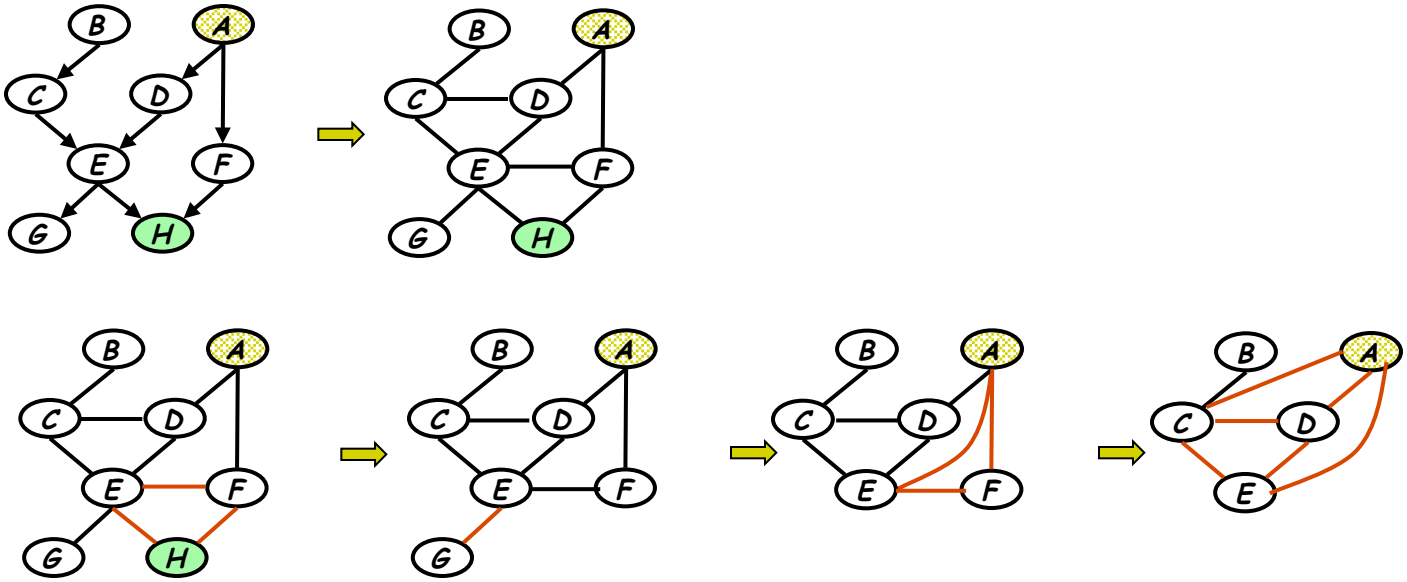
- A graph elimination algorithm



- Intermediate terms correspond to the cliques resulted from elimination



Elimination Cliques



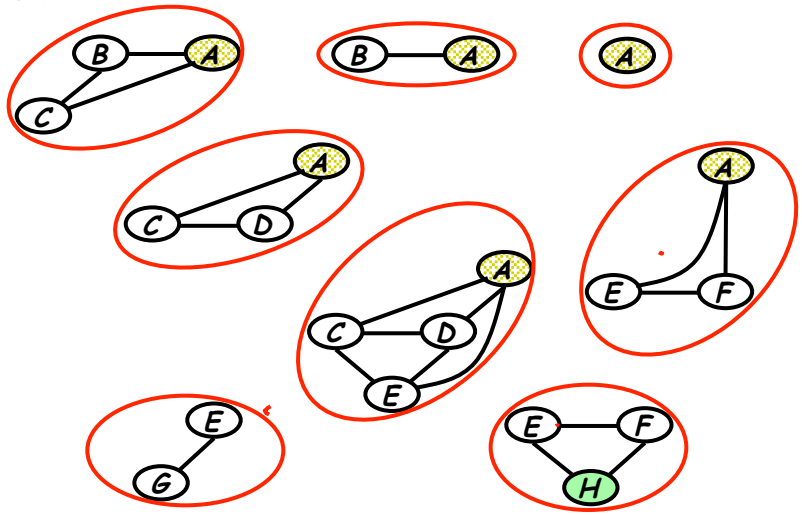
$m_h(e,f)$ $m_g(e)$ $m_f(e,a)$ $m_e(a,c,d)$

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

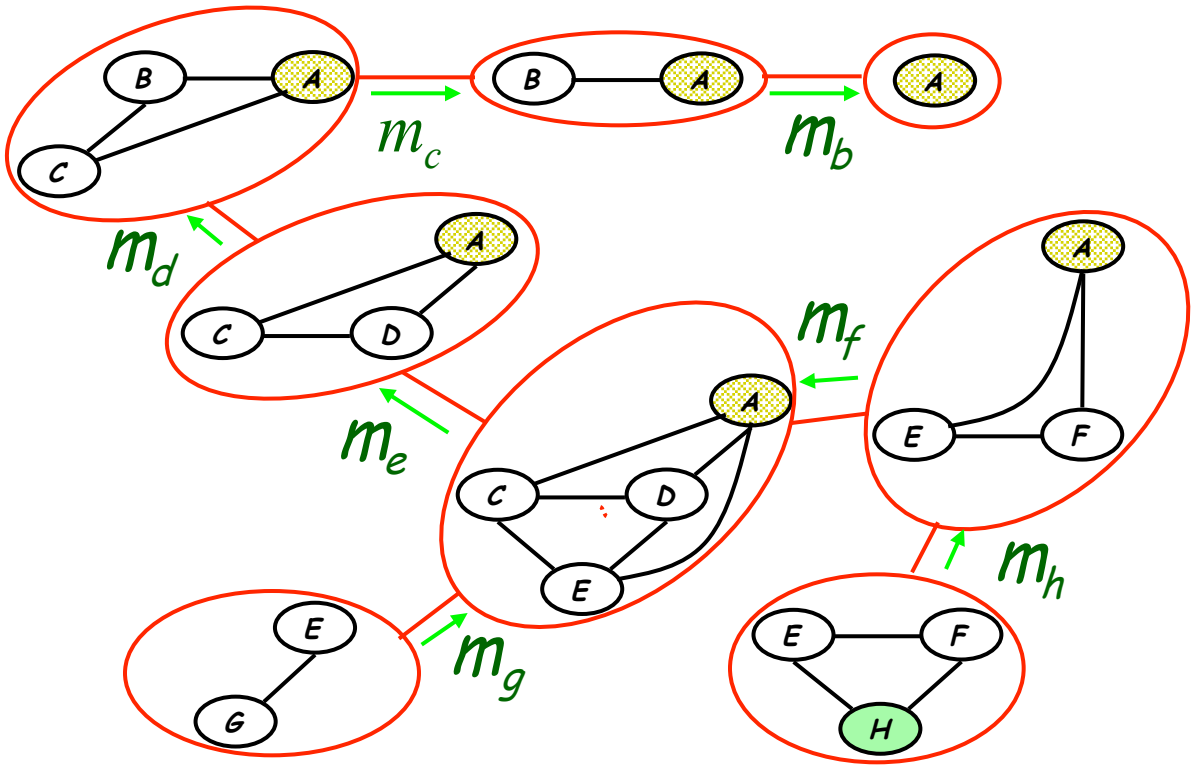
Graph elimination and marginalization

- 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



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

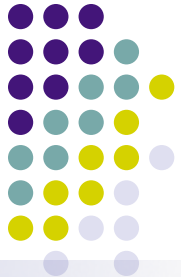
Complexity

- 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

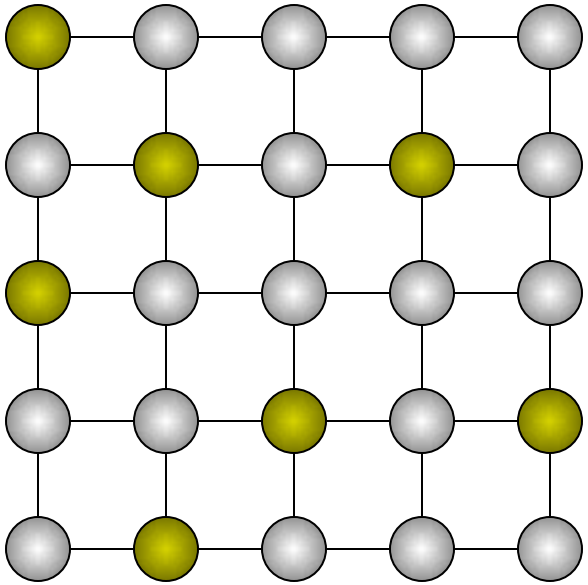
Examples



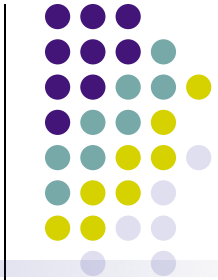
- Star

- 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

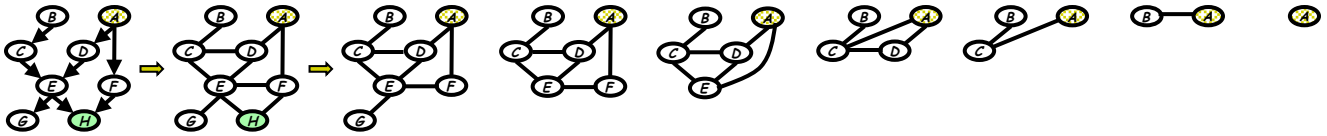
- What can we say about the overall computational complexity of the algorithm? In particular, how can we control the "size" of the summands that appear in the sequence of summation operation.

- 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.

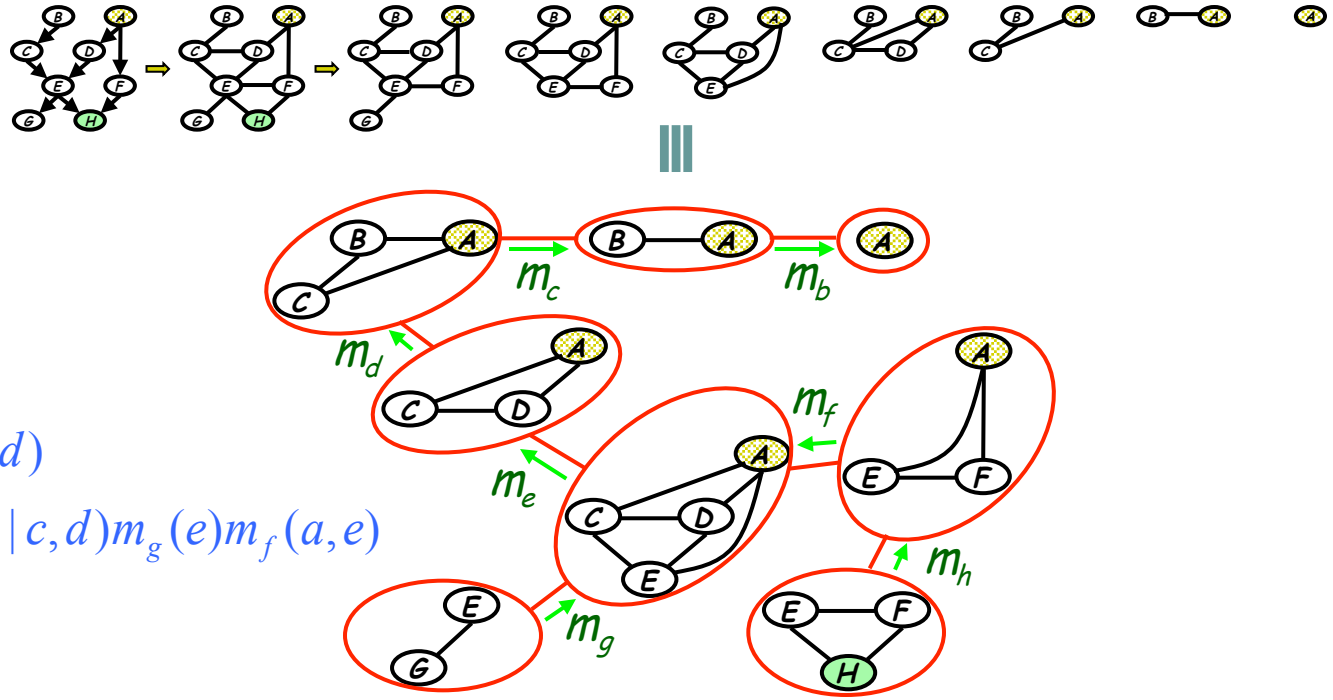
Limitation of Procedure Elimination

- Limitation



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**

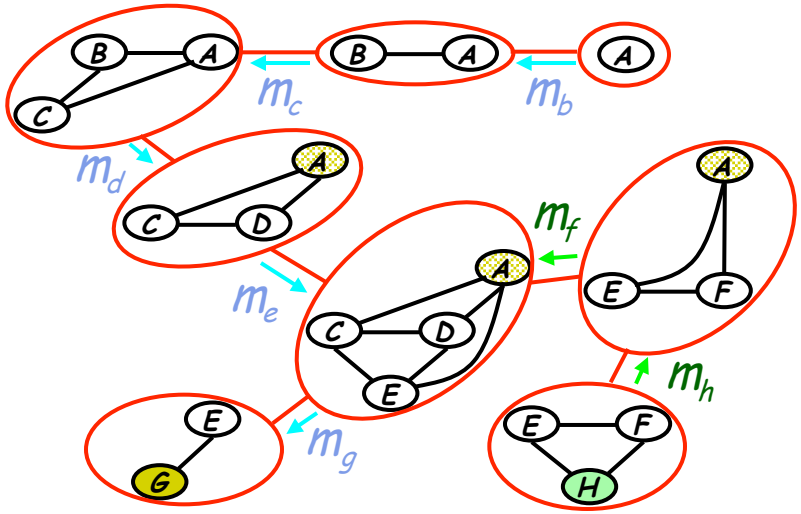


$$m_e(a,c,d) = \sum_e p(e \mid c,d) m_g(e) m_f(a,e)$$

- Messages can be reused

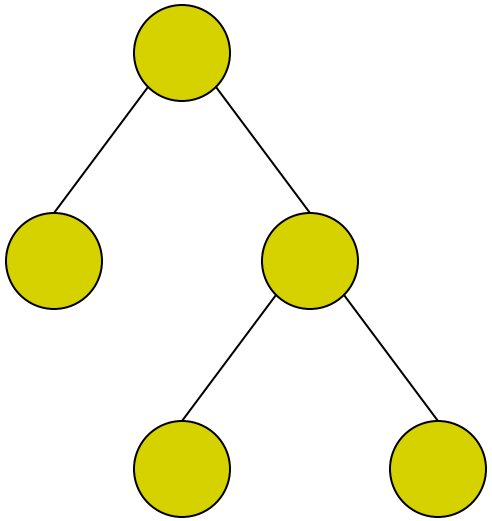
From Elimination to Message Passing

- 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 ...**

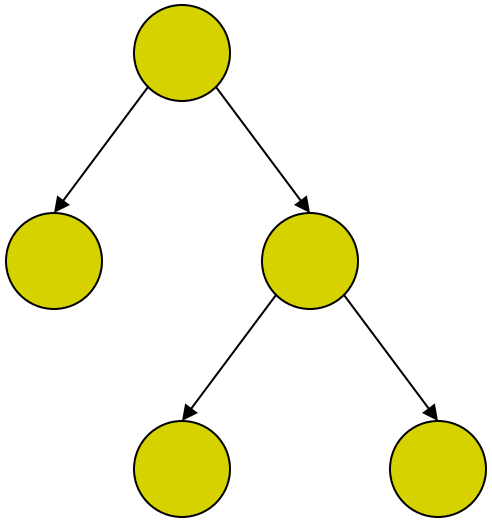


- Messages m_f and m_h are reused, others need to be recomputed

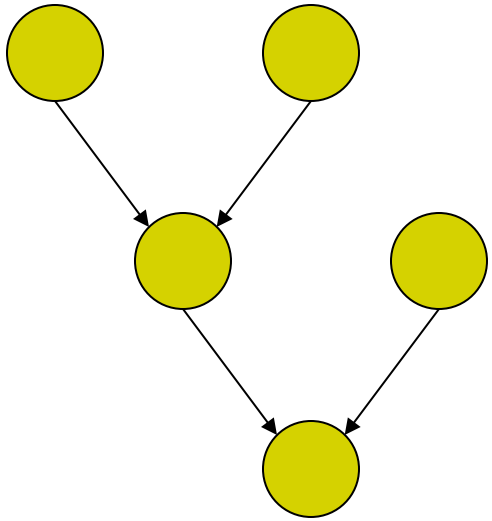
Tree GMs



Undirected tree: a unique path between any pair of nodes



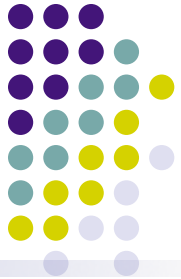
Directed tree: all nodes except the root have exactly one parent



Poly tree: can have multiple parents

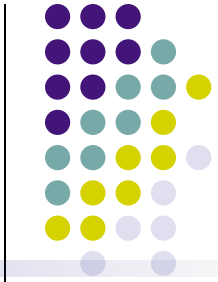
We will come back to this later

Equivalence of directed and undirected trees



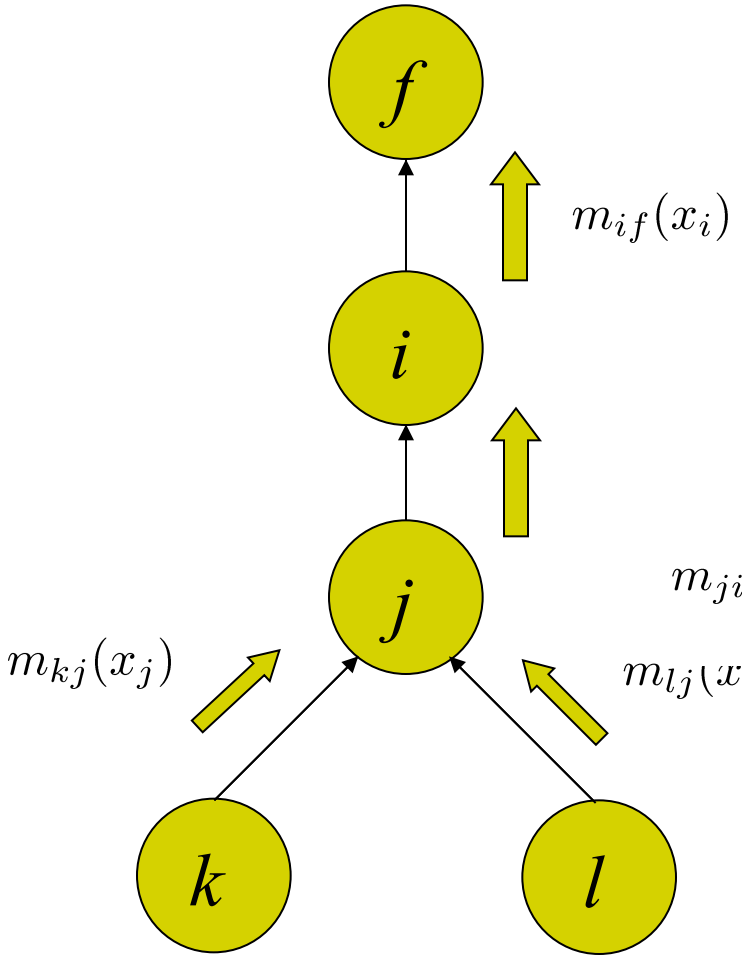
- Any undirected tree can be converted to a directed tree by choosing a root node and directing all edges away from it
- A directed tree and the corresponding undirected tree make the same conditional independence assertions
- Parameterizations are essentially the same.
  - Undirected tree: $$p(x) =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

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

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

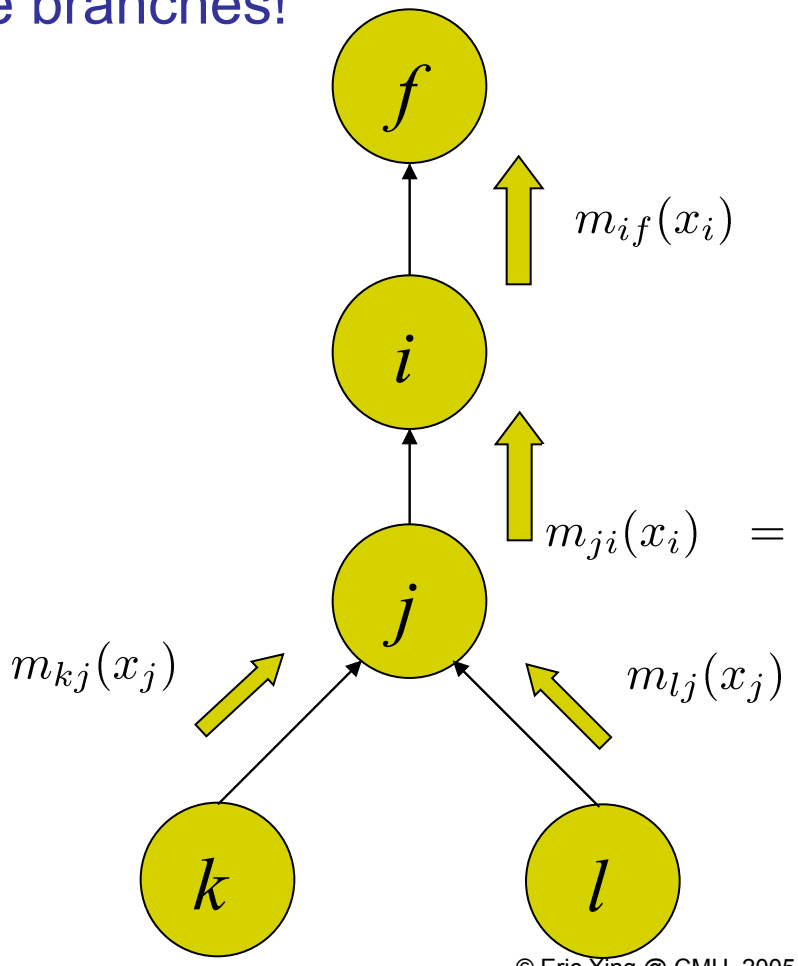
$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i,x_j) \prod_{k \in N(j)\setminus i} m_{kj}(x_j) \right)$$

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

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

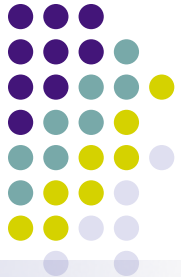
Message passing on a tree

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



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

From elimination to message passing



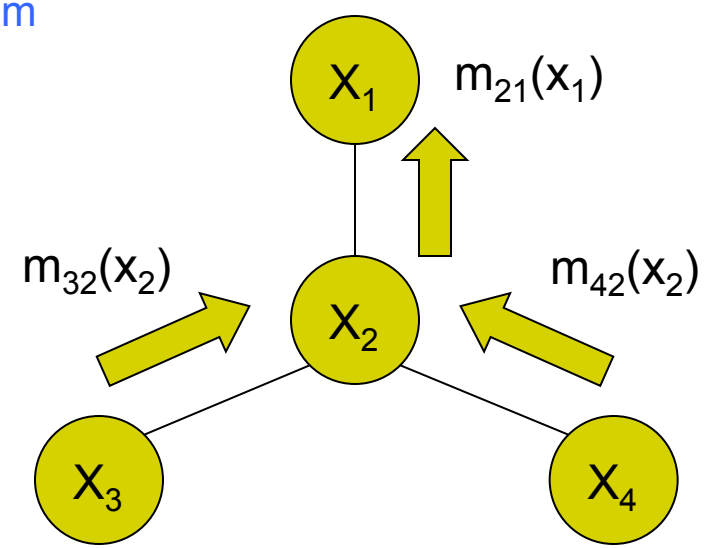
- Recall ELIMINATION algorithm:
  - Choose an ordering Z in which query node f is the final node
  - Place all potentials on an active list
  - Eliminate node i by removing all potentials containing i, take sum/product over x_i.
  - Place the resultant factor back on the list

- 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:

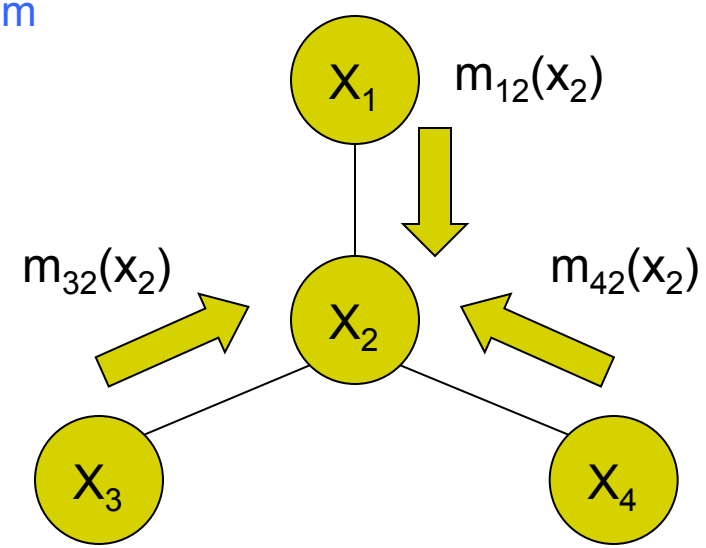
- A node can send a message to its neighbors when (and only when) it has received messages from all its ***other*** neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



Computing $P(X_1)$

The message passing protocol:

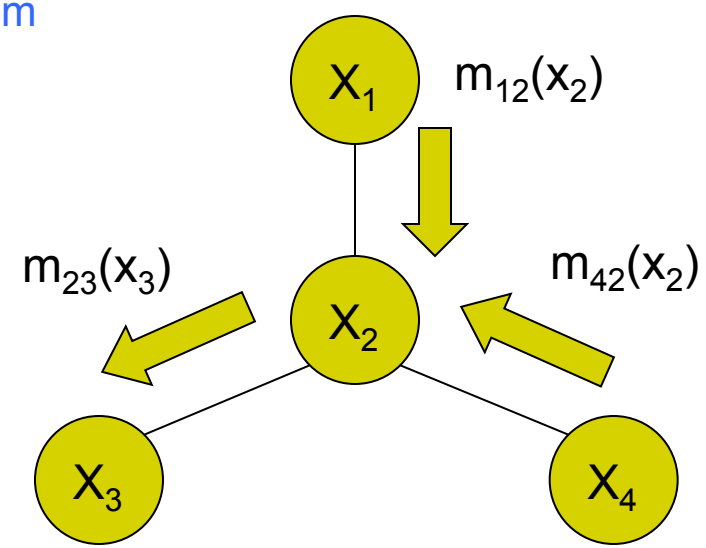
- A node can send a message to its neighbors when (and only when) it has received messages from all its ***other*** neighbors.
- Computing node marginals:
  - Naïve approach: consider each node as the root and execute the message passing algorithm



Computing $P(X_2)$

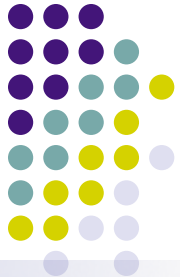
The message passing protocol:

- 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 $P(X_3)$

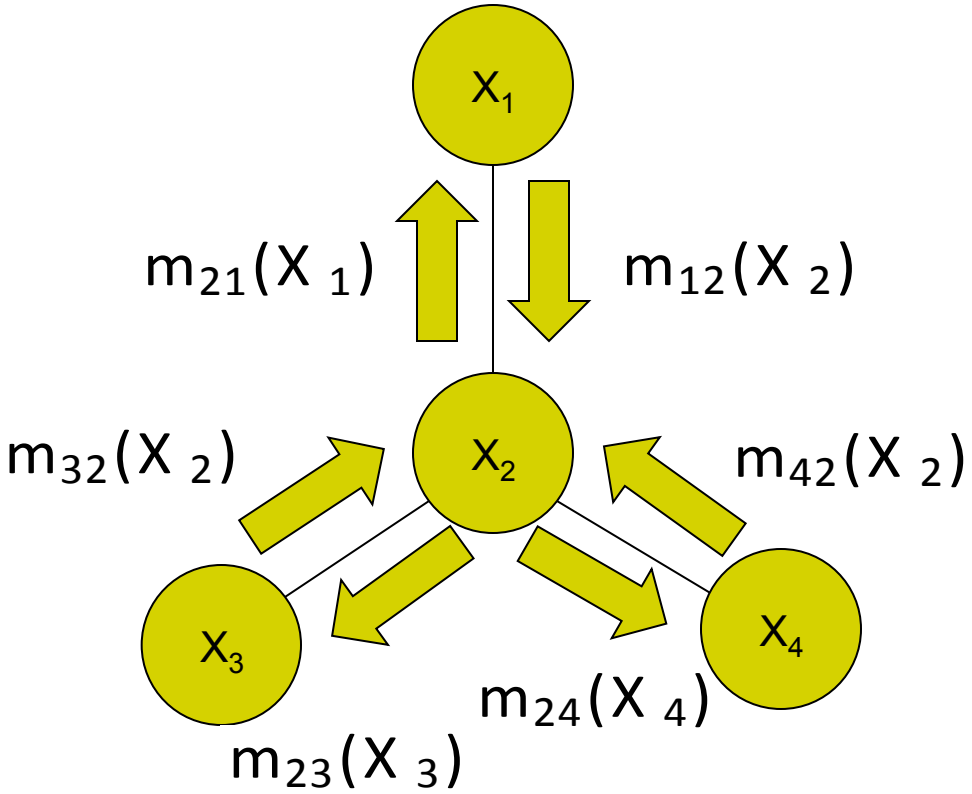
Computing node marginals



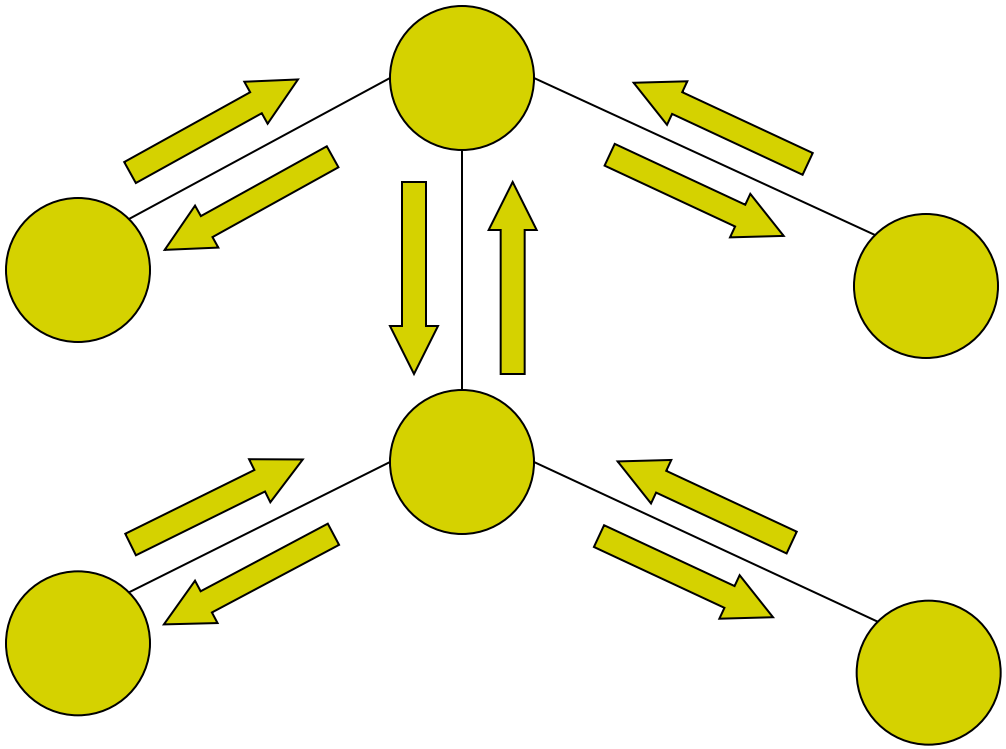
- Naïve approach:
  - Complexity: NC
    - N is the number of nodes
    - C is the complexity of a complete message passing
- Alternative dynamic programming approach
  - 2-Pass algorithm (next slide ➔)
  - Complexity: 2C!

The message passing protocol:

- A two-pass algorithm:

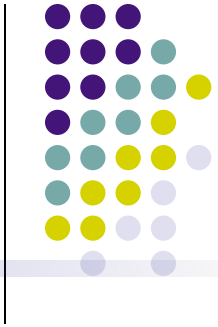


Belief Propagation (SP-algorithm): Parallel synchronous implementation



- 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



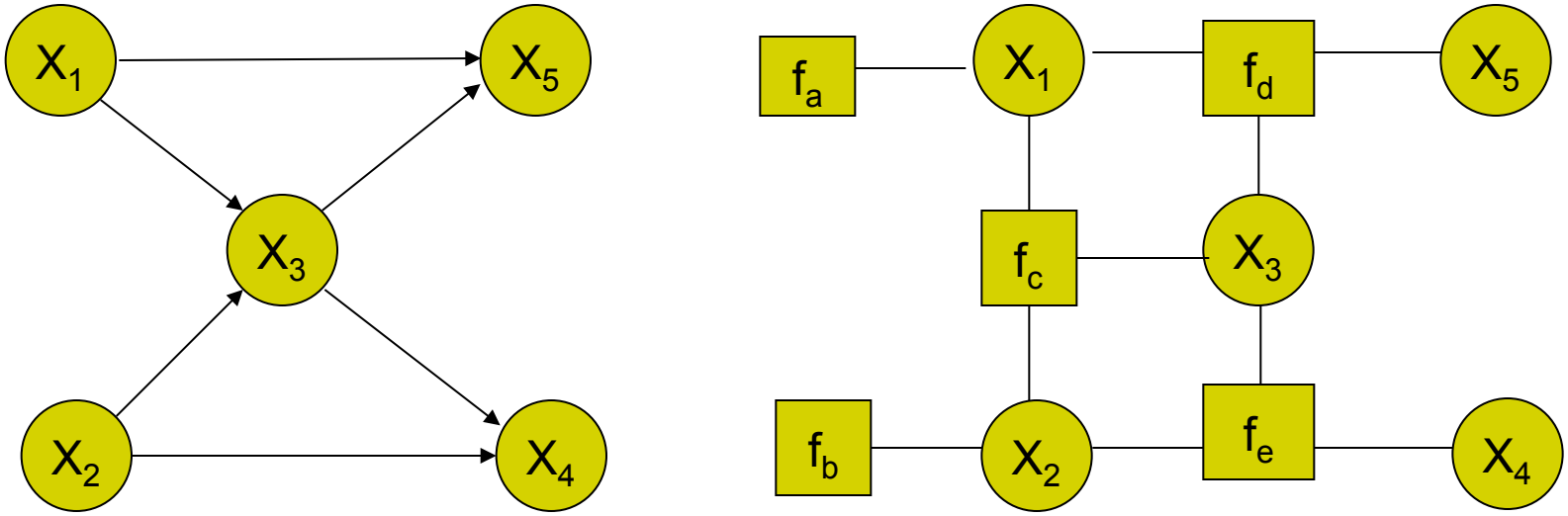
- Collollary: the synchronous implementation is "non-blocking"
- Thm: The Message Passage Guarantees obtaining all marginals in the tree

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

- What about non-tree?

Another view of SP: Factor Graph

- Example 1

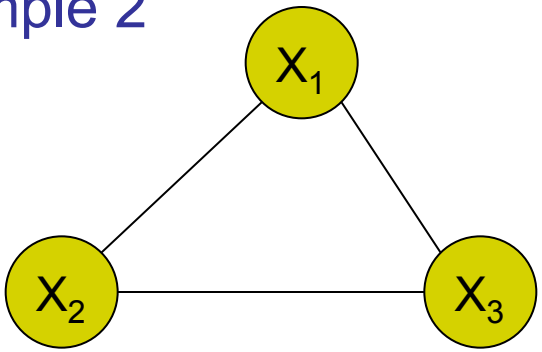


$P(X_1)$ $P(X_2)$ $P(X_3|X_1,X_2)$ $P(X_5|X_1,X_3)$ $P(X_4|X_2,X_3)$

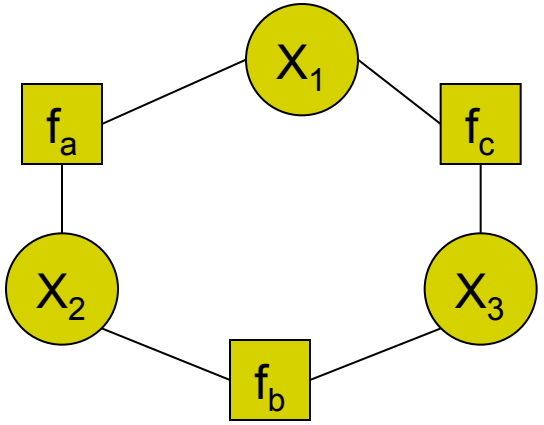
$f_a(X_1)$ $f_b(X_2)$ $f_c(X_3,X_1,X_2)$ $f_d(X_5,X_1,X_3)$ $f_e(X_4,X_2,X_3)$

Factor Graphs

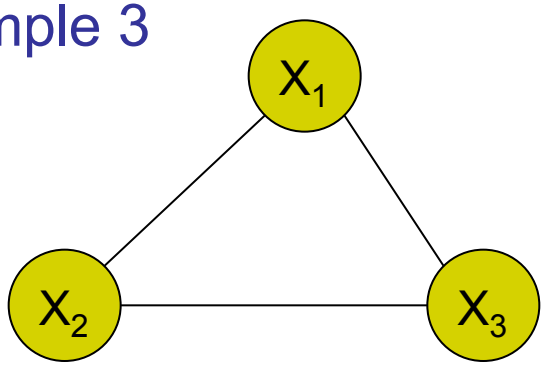
- Example 2



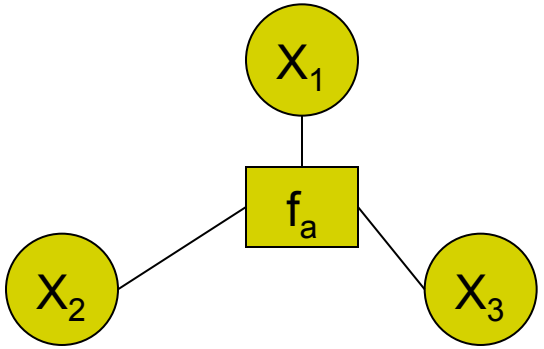
$\psi(x_1,x_2,x_3) = f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_3,x_1)$



- Example 3

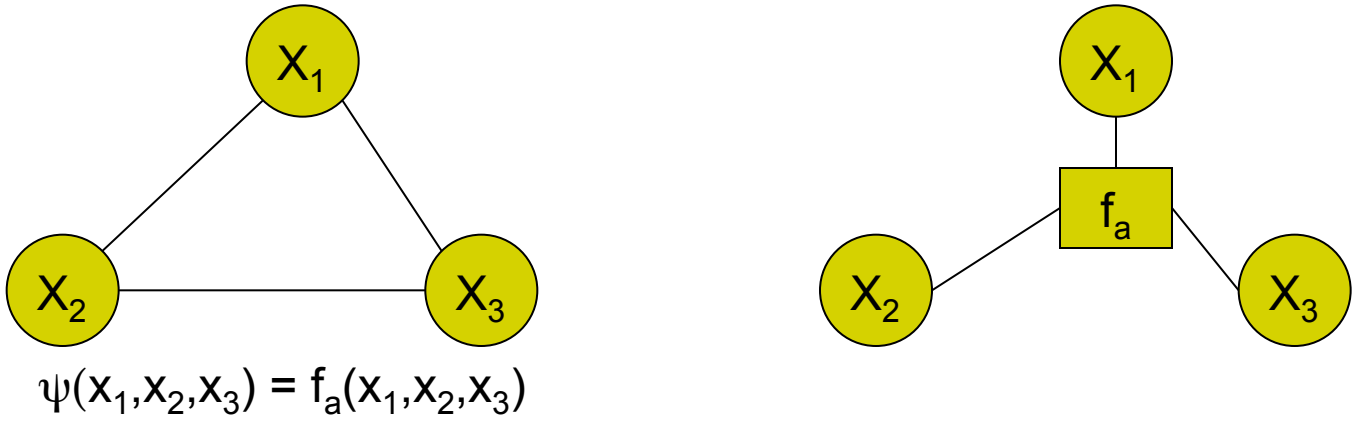


$\psi(x_1,x_2,x_3) = f_a(x_1,x_2,x_3)$

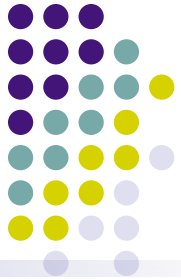


Factor Tree

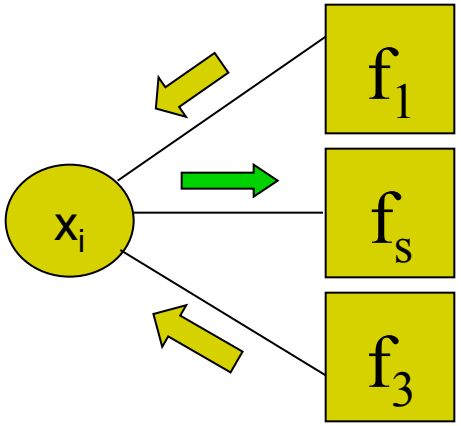
- A Factor graph is a Factor Tree if the undirected graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree

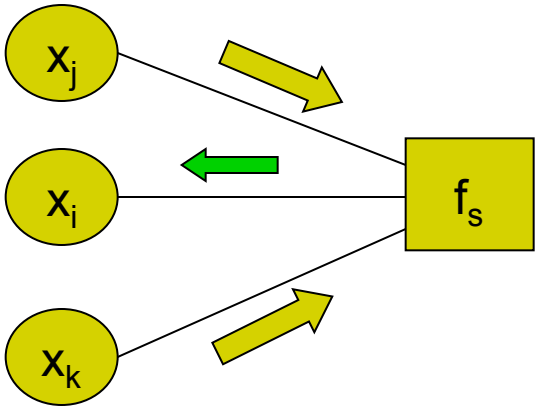


Message Passing on a Factor Tree



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



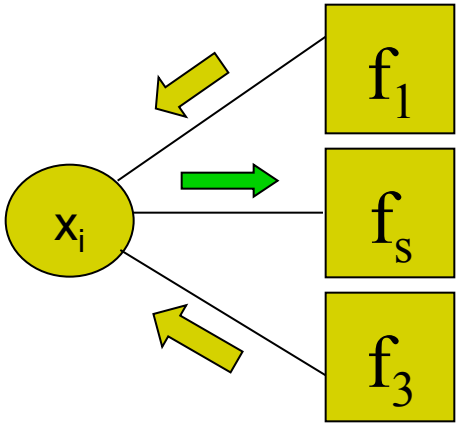


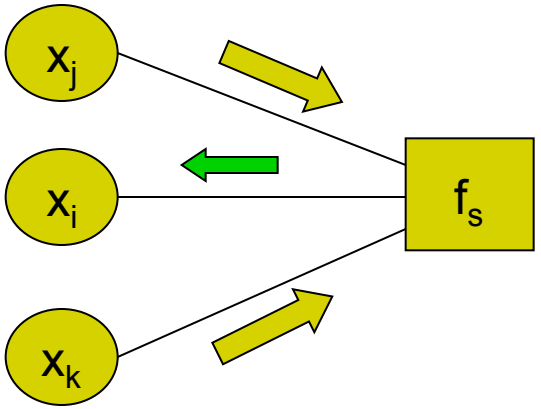
$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \mu_{ti}(x_i)$$

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

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:

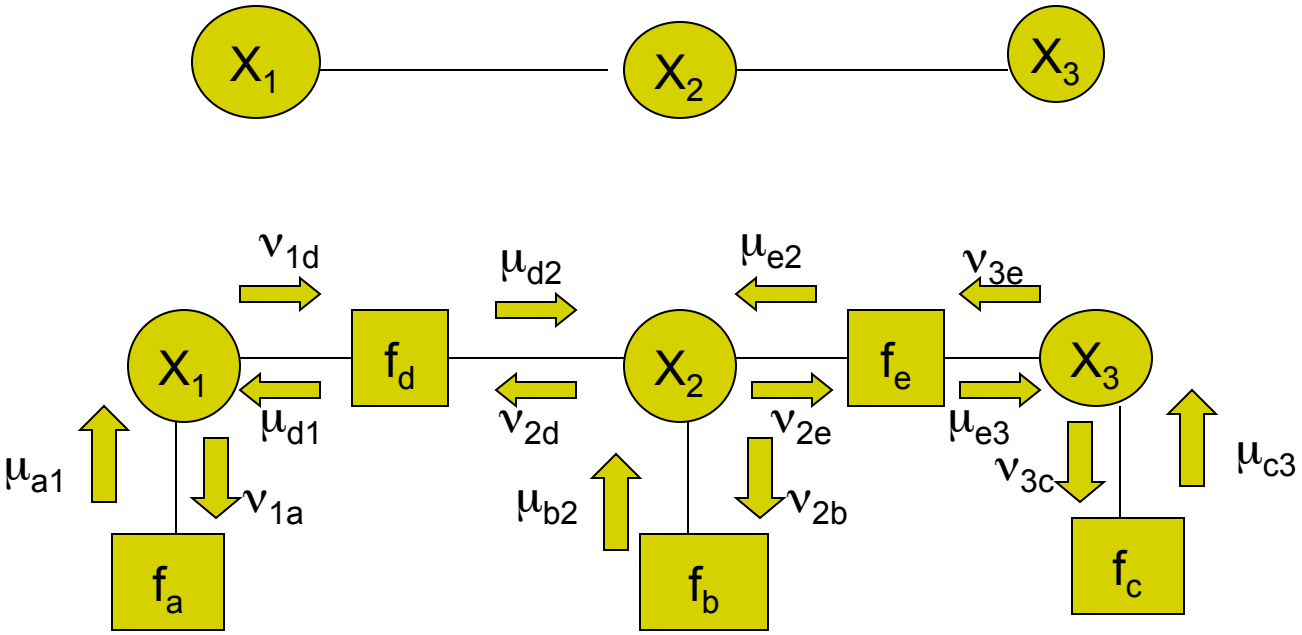




$$P(x_i) \propto \prod_{s \in N(i)} \mu_{si}(x_i)$$

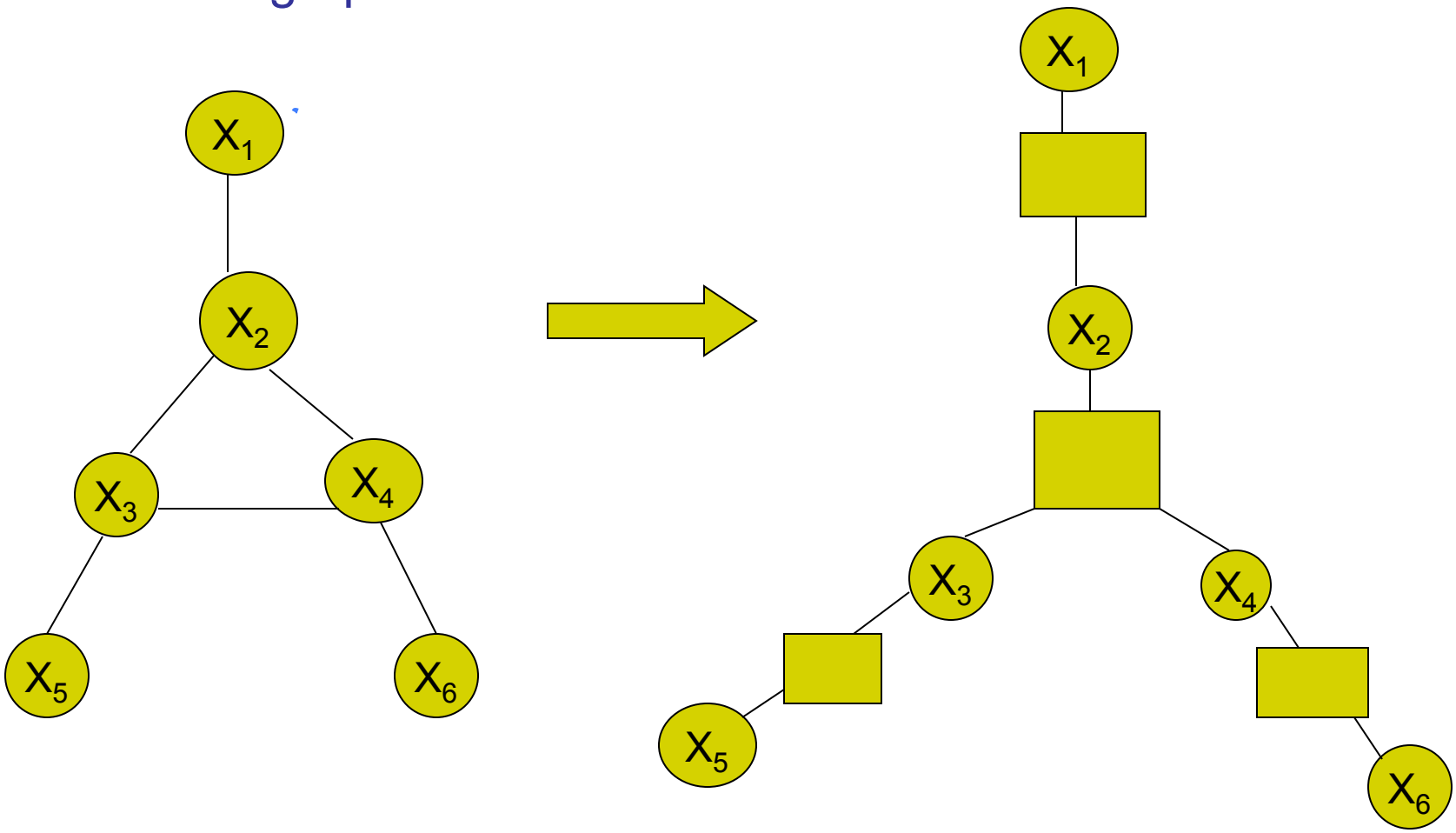
$$\propto \nu_{is}(x_i)\mu_{si}(x_i)$$

BP on a Factor Tree

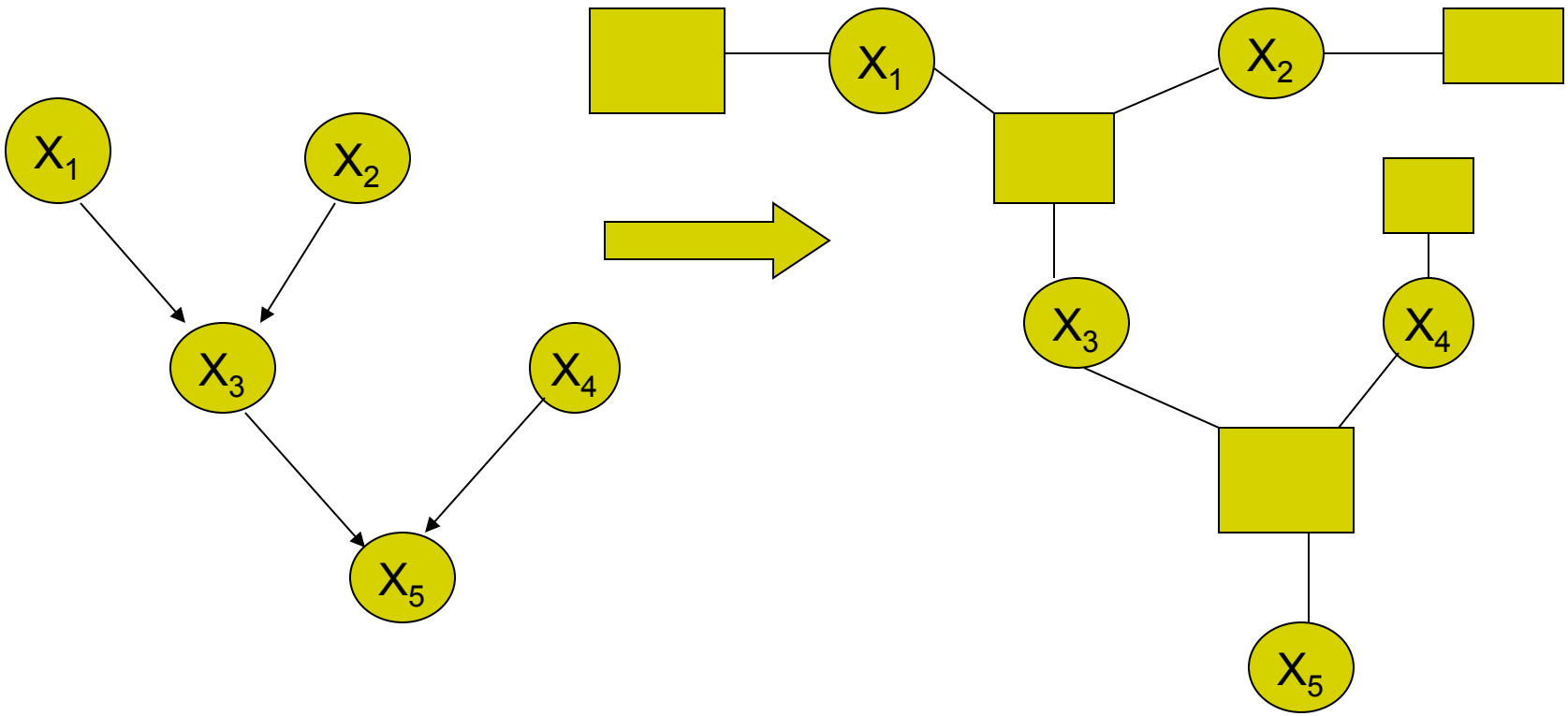


Why factor graph?

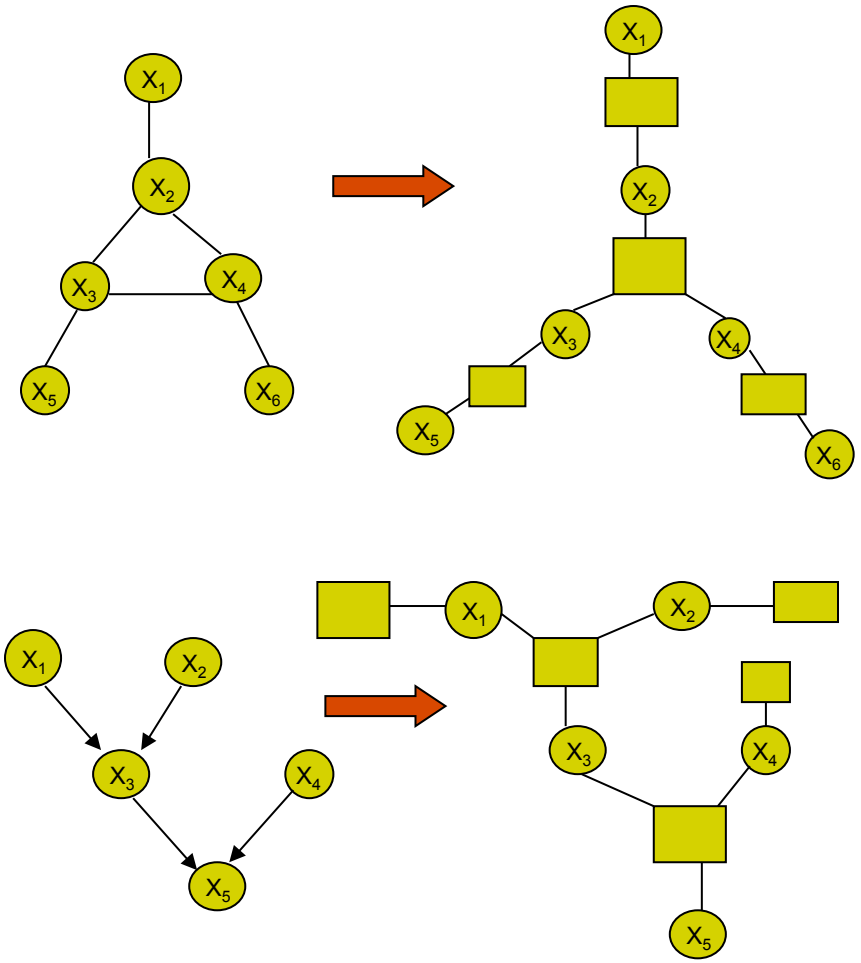
- Tree-like graphs to Factor trees



Poly-trees to Factor trees

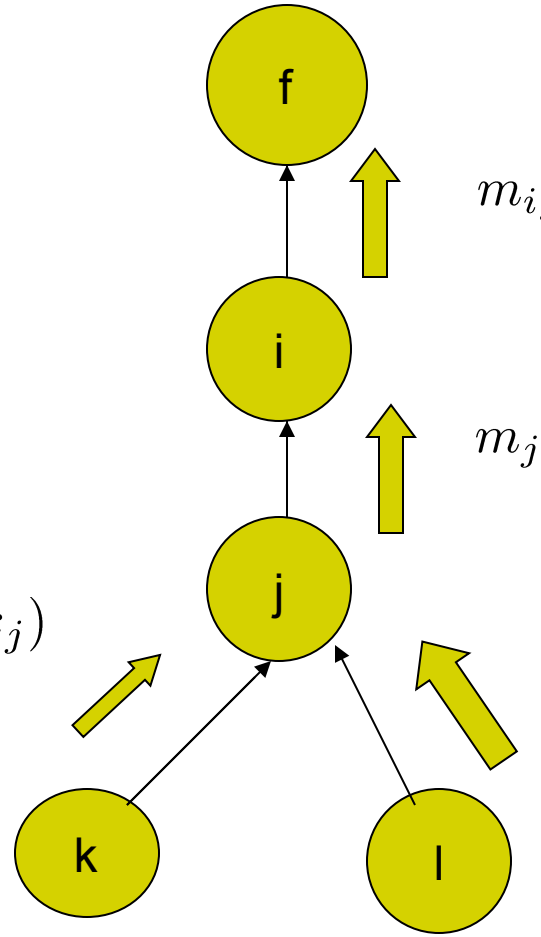


Why factor graph?



- 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



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

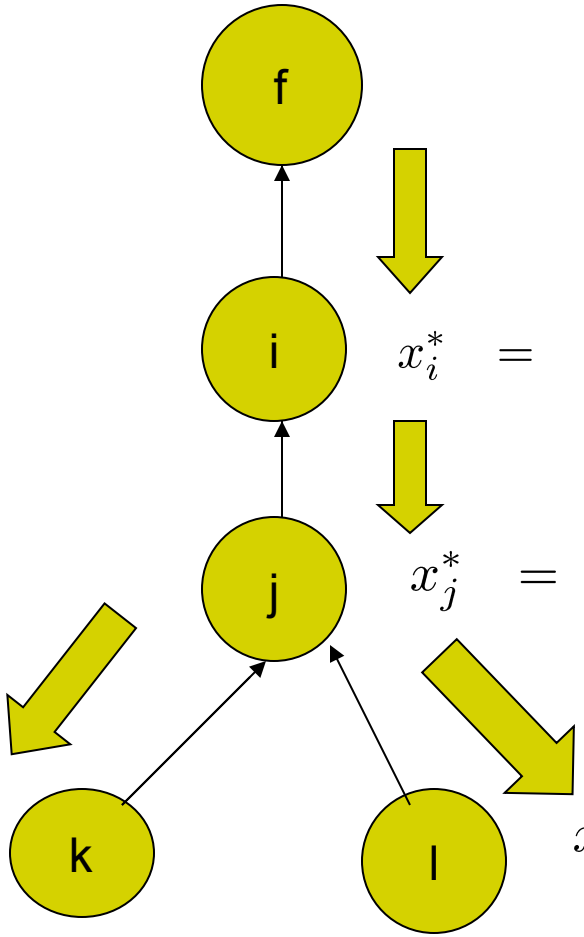
$m_{if}(x_f)$

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

$m_{kj}(x_j)$

$m_{lj}(x_j)$

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



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

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

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

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

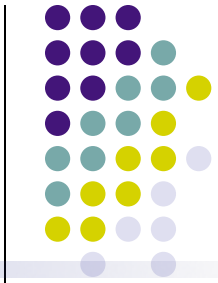
$$x_k^* = \arg\max_{x_k} \left(\psi(x_k)\psi(x_k, x_j^*)\right)$$

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

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

- Junction tree data-structure for exact inference on general graphs