The Sum-Product Algorithm

Prof. Nicholas Zabaras
Center for Informatics and Computational Science
https://cics.nd.edu/
University of Notre Dame
Notre Dame, IN, USA

Email: nzabaras@gmail.com
URL: https://www.zabaras.com/

February 13, 2018

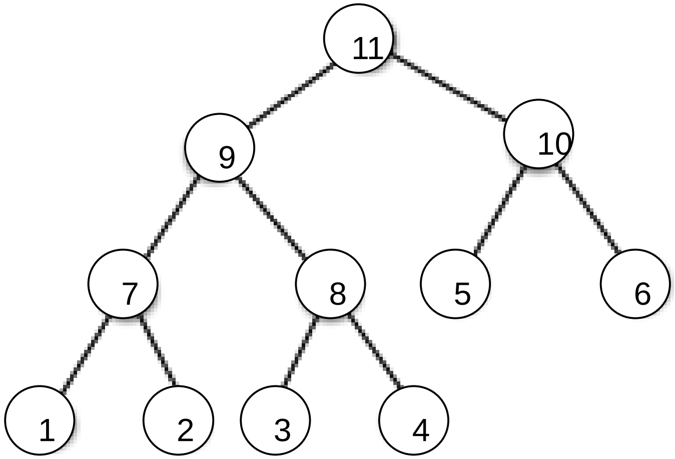
Contents

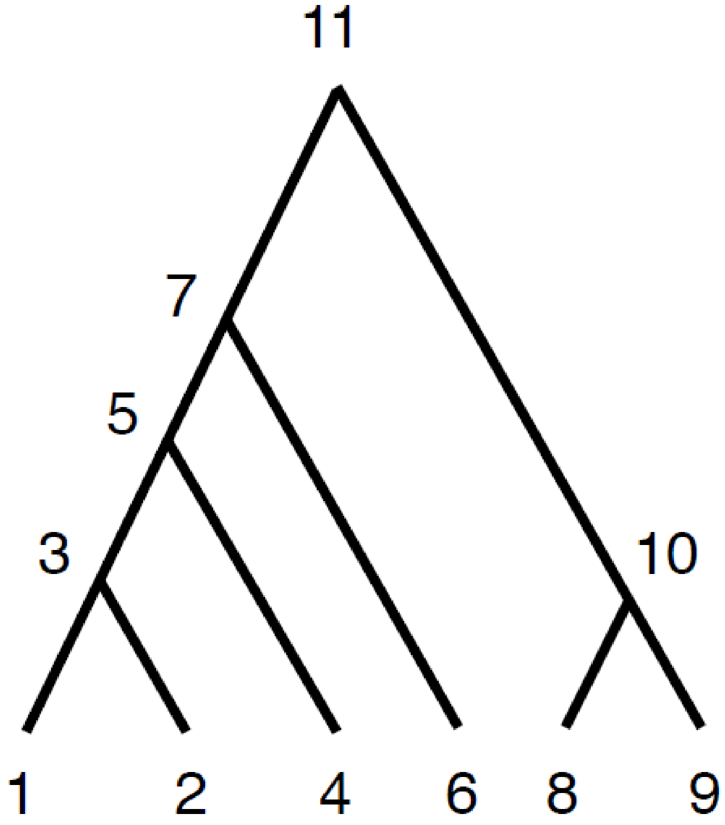
- PostOrder – Optimal Elimination, Sum-Product Algorithm for a Tree
- Factor Graphs from Directed Graphs, Factor Graphs from Undirected Graphs, Factor Graphs from Polytrees, Conditional Independence and Factor Graphs
- I-Map, D-Map and Perfect Map, Venn Diagram
- Factor Trees and the Sum-Product Algorithm, Messages from Factor Nodes to Variable Nodes, Initialization, Sum-Product algorithm for Markov Chains, Sum-Product algorithm example, Incorporating Evidence, Marginal associated with each factor, Marginals not associated with factors, Marginal associated with each variable node
- Sum-Product Algorithm for Factor Trees, Equivalence with the Sum-Product algorithm

- Kevin Murphy, Machine Learning: A probabilistic Perspective, Chapter 19
- Chris Bishop, Pattern Recognition and Machine Learning, Chapter 8
- Jordan, M. I. (2007). An introduction to probabilistic graphical models. In preparation (Chapters 4).
- Video Lectures on Machine Learning, Z. Gahramani, C. Bishop and others.
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.
- Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local computations with probabalities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society* **50**, 157–224.

Postorder – Optimal Elimination

```
postorder(u)
if u is a leaf
  print u;
else
  postorder(u →leftChild);
  postorder(u →rightChild);
  print u;
end
```

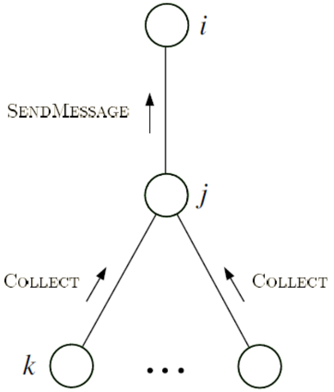




Sum Product Algorithm for a Tree

```
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
        $\psi^E(x_i)=\psi(x_i)$
```

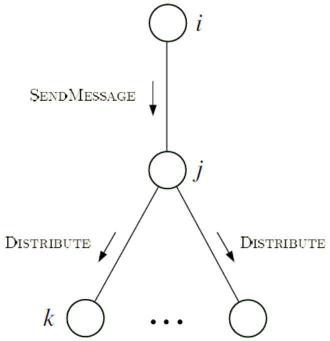


```
SendMessage(j,i)
    $m_{ji}(x_i)=\sum_{x_j}\left(\psi^E(x_j)\psi(x_i,x_j)\prod_{k\in\mathcal{N}(j)\setminus i} m_{kj}(x_j)\right)$
```

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

```
Collect(i,j)
    for $k\in\mathcal{N}(j)\setminus i$
        Collect(j,k)
    SendMessage(j,i)
```

```
Distribute(i,j)
    SendMessage(i,j)
    for $k\in\mathcal{N}(j)\setminus i$
        Distribute(j,k)
```



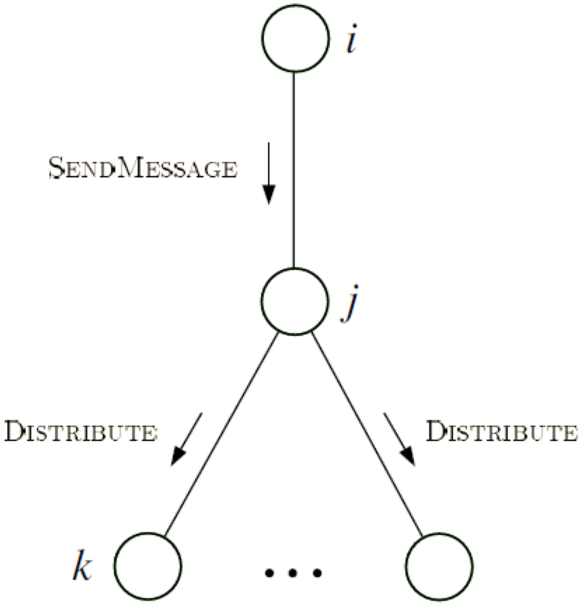
Sum Product Algorithm for a Tree

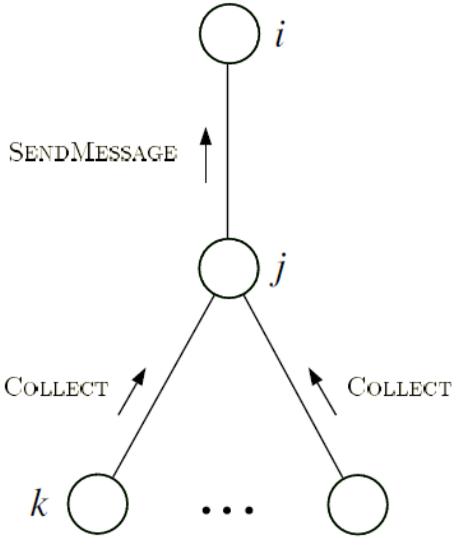
```
Sum-Product($\mathcal{T}$, E)
    Evidence(E)
    f=ChooseRoot($\mathcal{V}$)
    for e $\in \mathcal{N}$(f)
        Collect(f,e)
    for e $\in \mathcal{N}$(f)
        Distribute(f,e)
    for i $\in \mathcal{V}$
        ComputeMarginal(i)
```

Choose any root (unspecified here)

Messages flow from the leaves to the root

Messages flow outward from the root to the leaves

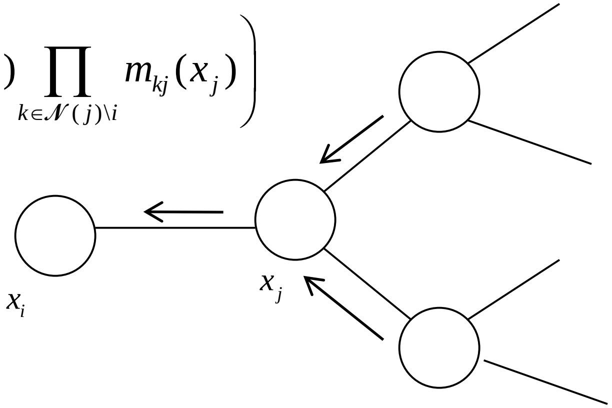




Belief Propagation: Sum-Product Algorithm

- The message passing algorithm is extended to any tree-structured graph (no loops).
- For each node to compute the outgoing message:
  - Form product of incoming messages and local evidence (if e.g. node *j* here is connected to an observed node)
  - Marginalize over x_j to give outgoing message at x_j
  - One message in each direction across each link
- The algorithm fails if there are loops in the graph

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

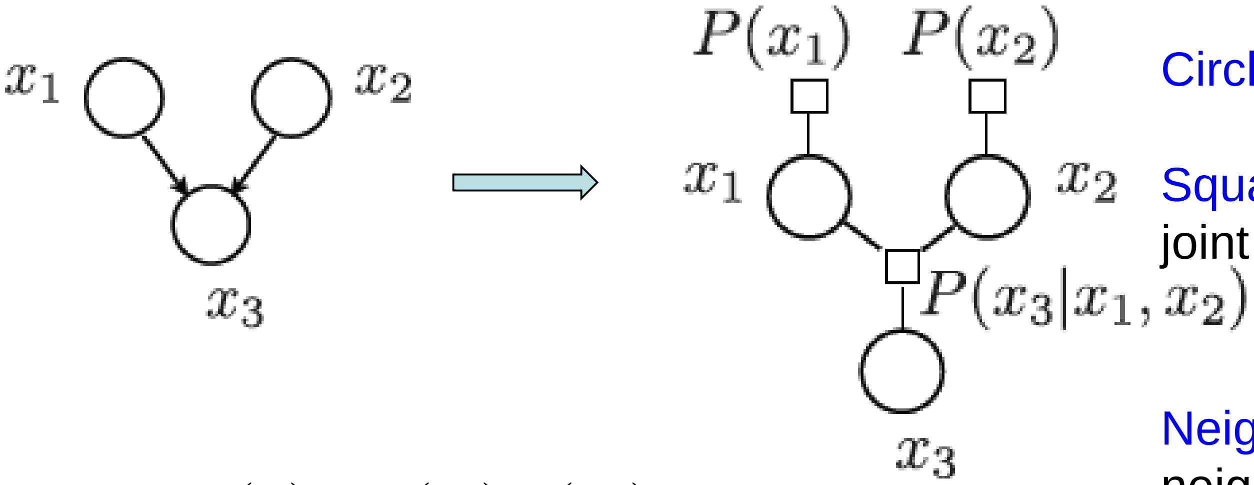


To implement this, propagate messages from the root node to the leaf node and reversely, and save all messages along each and every edge.

Factor Graphs From Directed Graphs

- ❑ Factor graphs explicate how the joint distribution factors into smaller components

- ❑ Each factor node is connected to all the variable nodes that the corresponding factor depends on.



Circles: random variables

Squares: Factors in the joint distribution

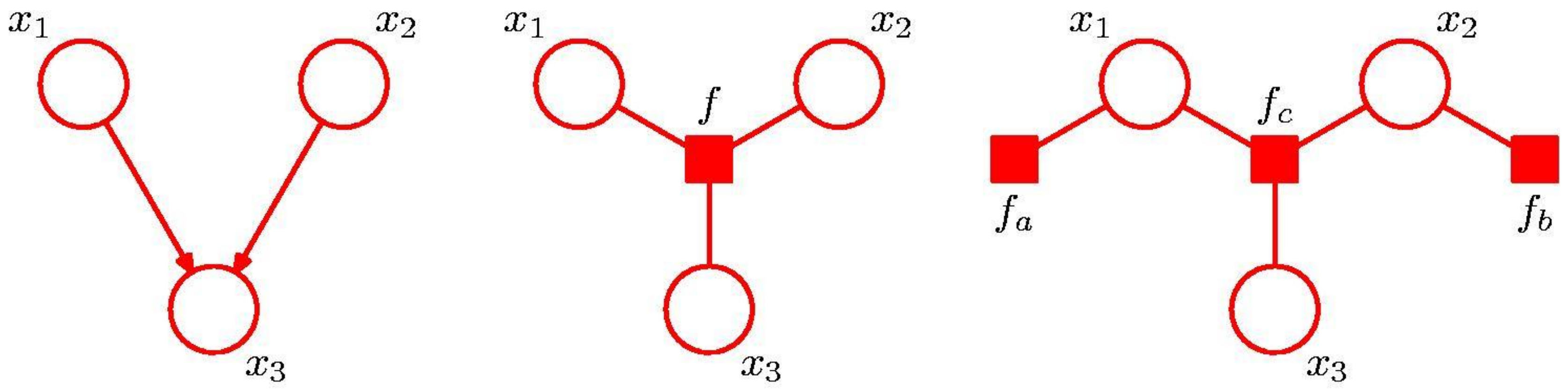
Neighbors: Two nodes are neighbors if they share a factor

$$p(x) = \underset{\psi_1(x_1)}{p(x_1)}\,\underset{\psi_2(x_2)}{p(x_2)}\,\underbrace{p(x_3|x_1,x_2)}_{\psi_{1,2,3}(x_1,x_2,x_3)}$$

- Frey, B. J. (1998). *Graphical Models for Machine Learning and Digital Communication*. MIT Press.
- Kschischnang, F. R., B. J. Frey, and H. A. Loeliger (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* **47**(2), 498–519.

Factor Graphs from Directed Graphs

- The conversion of a directed graph to a factor graph is illustrated in the Figure below



$$p(x) = p(x_1)\,p(x_2)\,p(x_3|x_1,x_2)$$

$$f(x_1,x_2,x_3) = p(x_1)\,p(x_2)\,p(x_3|x_1,x_2)$$

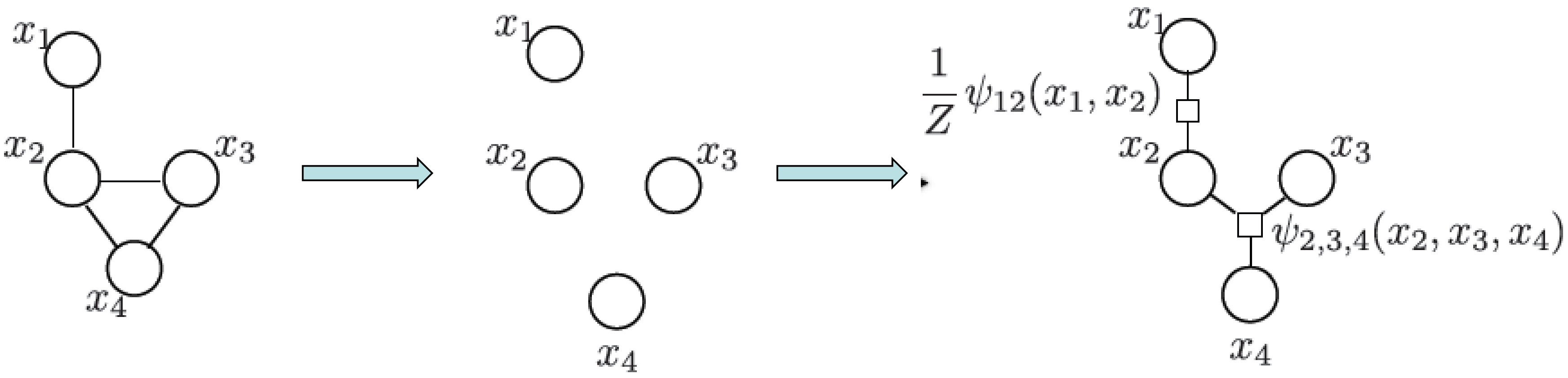
$$f_a(x_1) = p(x_1)$$
$$f_b(x_2) = p(x_2)$$
$$f_c(x_1,x_2,x_3) = p(x_3|x_1,x_2)$$

There can be multiple factor graphs all of which correspond to the same directed graph.

Factor Graphs From Undirected Graphs

- ❑ We can also define a factor graph representation of an undirected graph.

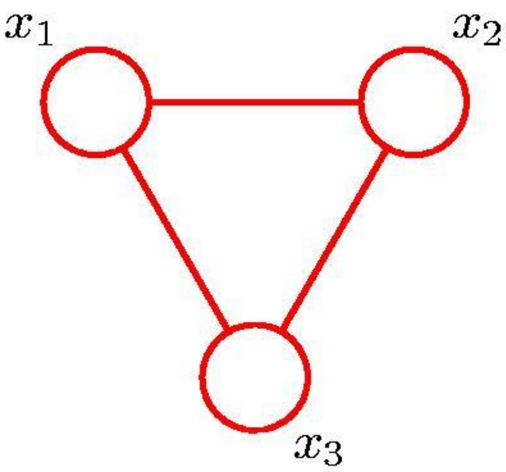
- ❑ Each factor node is connected to all the variable nodes that the corresponding factor depends on.


x_1
x_2
x_3
x_4
x_1
x_2
x_3
x_4
$\frac{1}{Z}\psi_{12}(x_1, x_2)$
x_1
x_2
x_3
$\psi_{2,3,4}(x_2, x_3, x_4)$
x_4

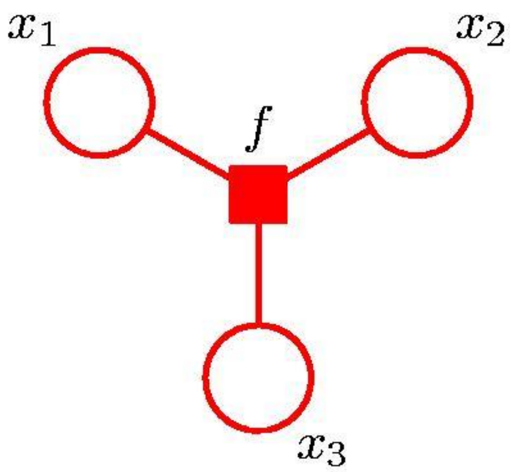
$$p(x) = \underbrace{\frac{1}{Z}\psi_{12}(x_1, x_2)}_{1st\ \ factor}\underbrace{\psi_{234}(x_2, x_3, x_4)}_{2nd\ \ factor}$$

Factor Graphs from Undirected Graphs

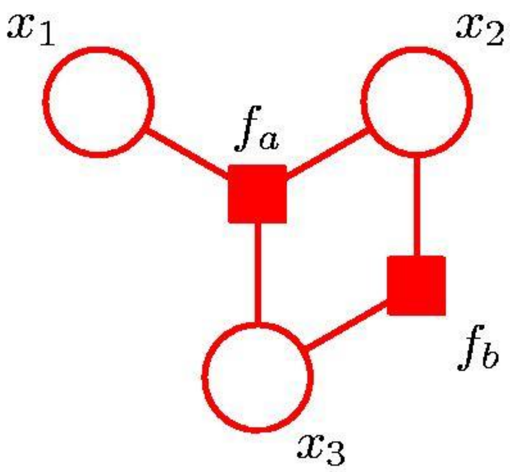
- An undirected graph can be readily convert it to a factor graph.



$$\psi\left(x_1, x_2, x_3\right)$$



$$f\left(x_1, x_2, x_3\right) = \psi\left(x_1, x_2, x_3\right)$$

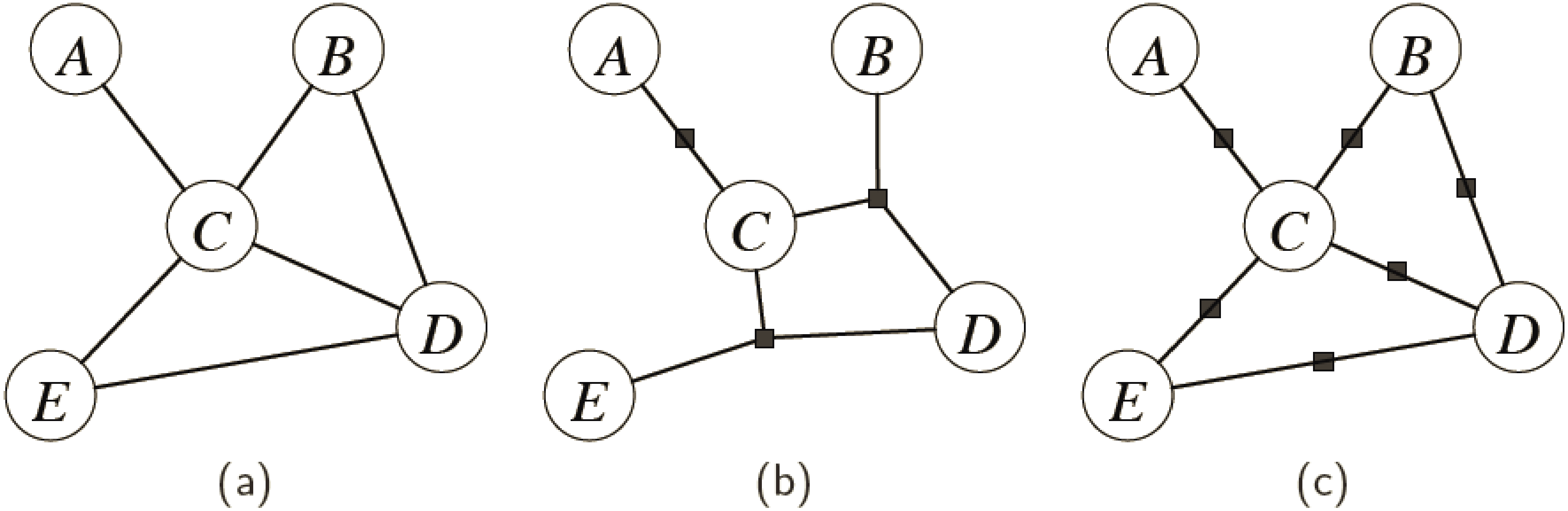


$$f_a\left(x_1, x_2, x_3\right) f_b\left(x_2, x_3\right) = \psi\left(x_1, x_2, x_3\right)$$

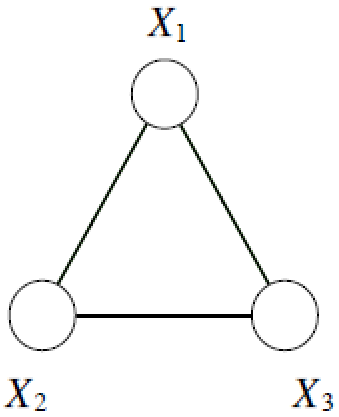
- *There may be several different factor graphs that correspond to the same undirected graph*.

Undirected Graphs and Factor Graphs

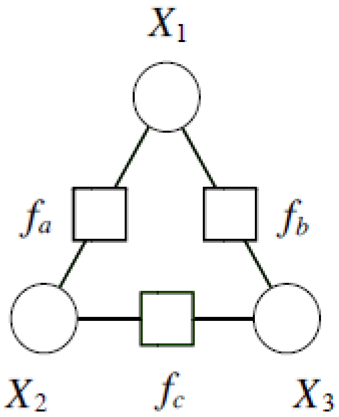
- All nodes in (a), (b), and (c) have exactly the same neighbors and these three graphs represent exactly the same conditional independence relationships.
- In (c) the probability factors into a product of pairwise functions.
- Consider the case where each variables is discrete and can take on K possible values. The functions in (a) and (b) are tables with $\mathcal{O}(K^3)$ cells, whereas in (c) they are $\mathcal{O}(K^2)$.



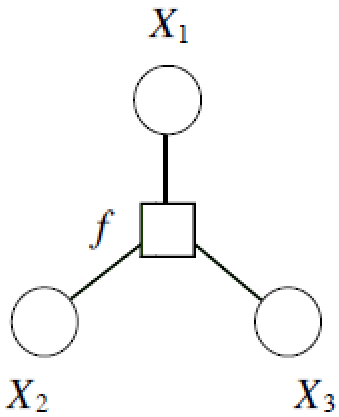
Converting to Factor Graphs

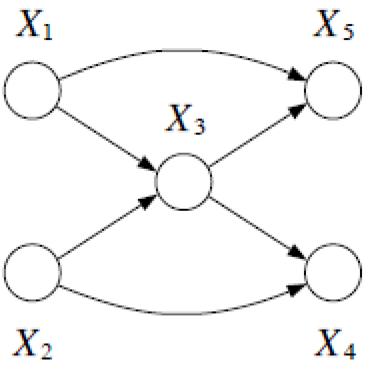


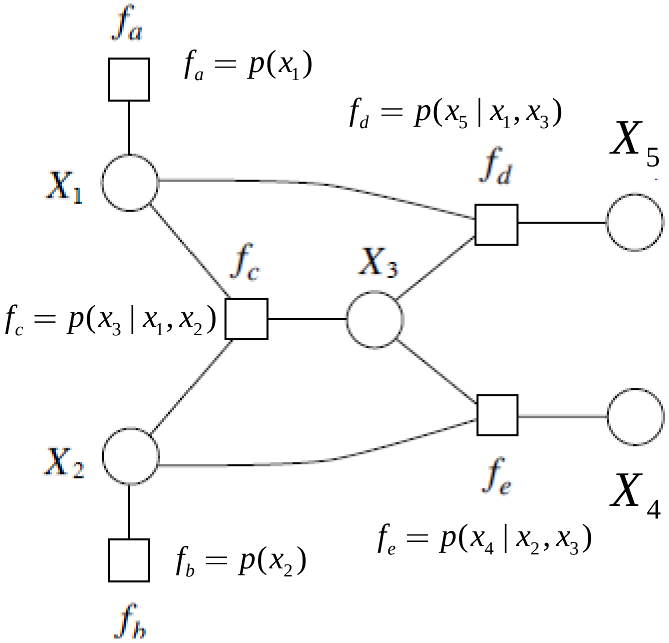
$\psi \quad x_1, x_2, x_3 \quad = f_a(x_1, x_2)\, f_b(x_1, x_3)\, f_c(x_2, x_3)$



$\psi \quad x_1, x_2, x_3$







$f_a = p(x_1)$

$f_d = p(x_5 \mid x_1, x_3)$

$f_c = p(x_3 \mid x_1, x_2)$

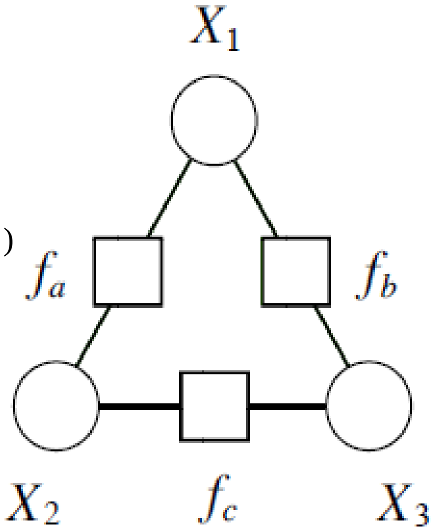
$f_b = p(x_2)$

$f_e = p(x_4 \mid x_2, x_3)$

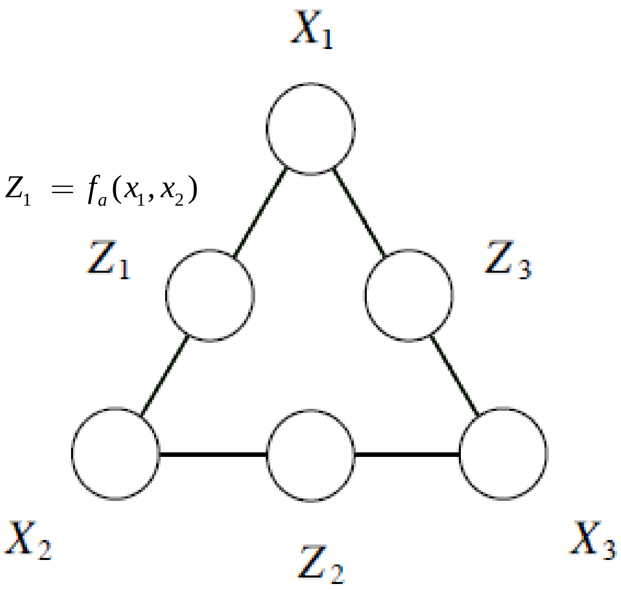
Converting From Factor Graphs

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

$f_a(x_1, x_2)$



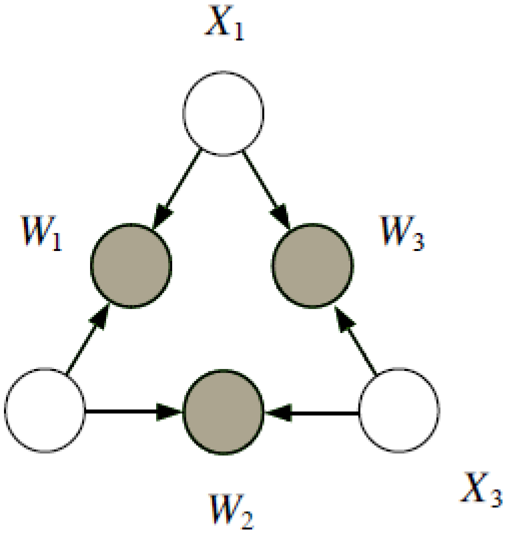
Z$_1$ an indicator variable taking 4 values for each combination of the values of X$_1$ and X$_2$ (assumed binary)



$\psi\;Z_1\;= f_a(x_1, x_2)$

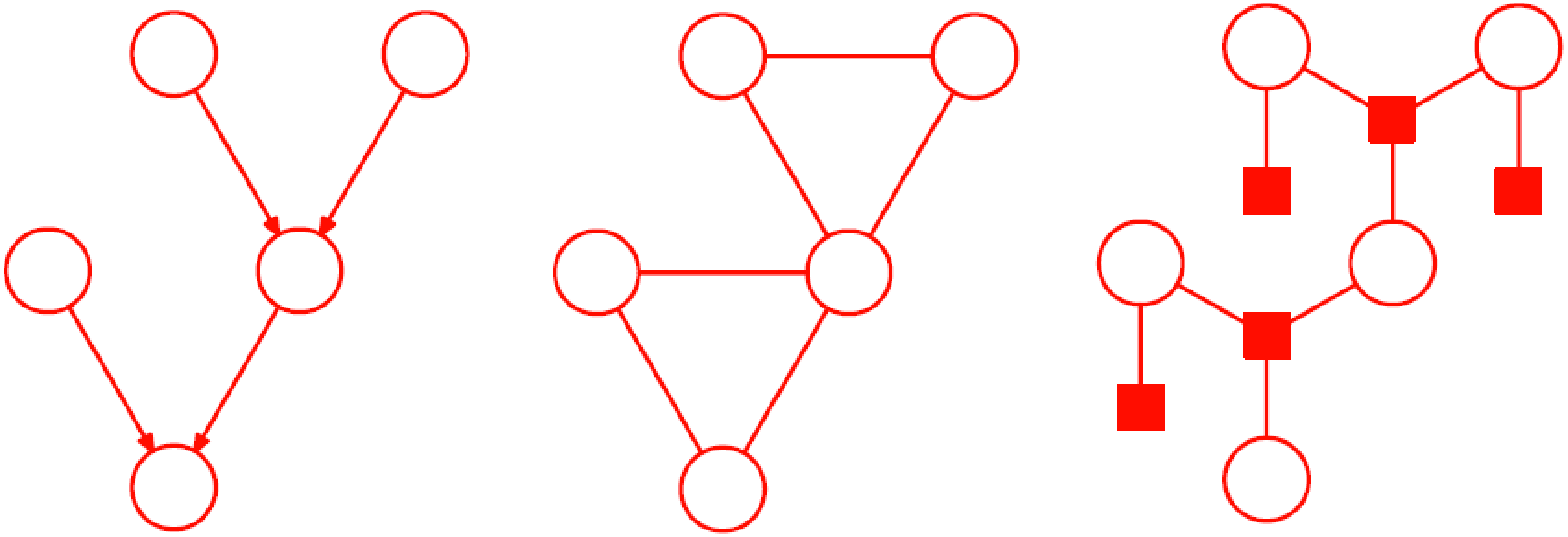
$p\;W_1 = 1 | x_1, x_2\;= f_a(x_1, x_2)$

W$_1$ binary variable always set to 1



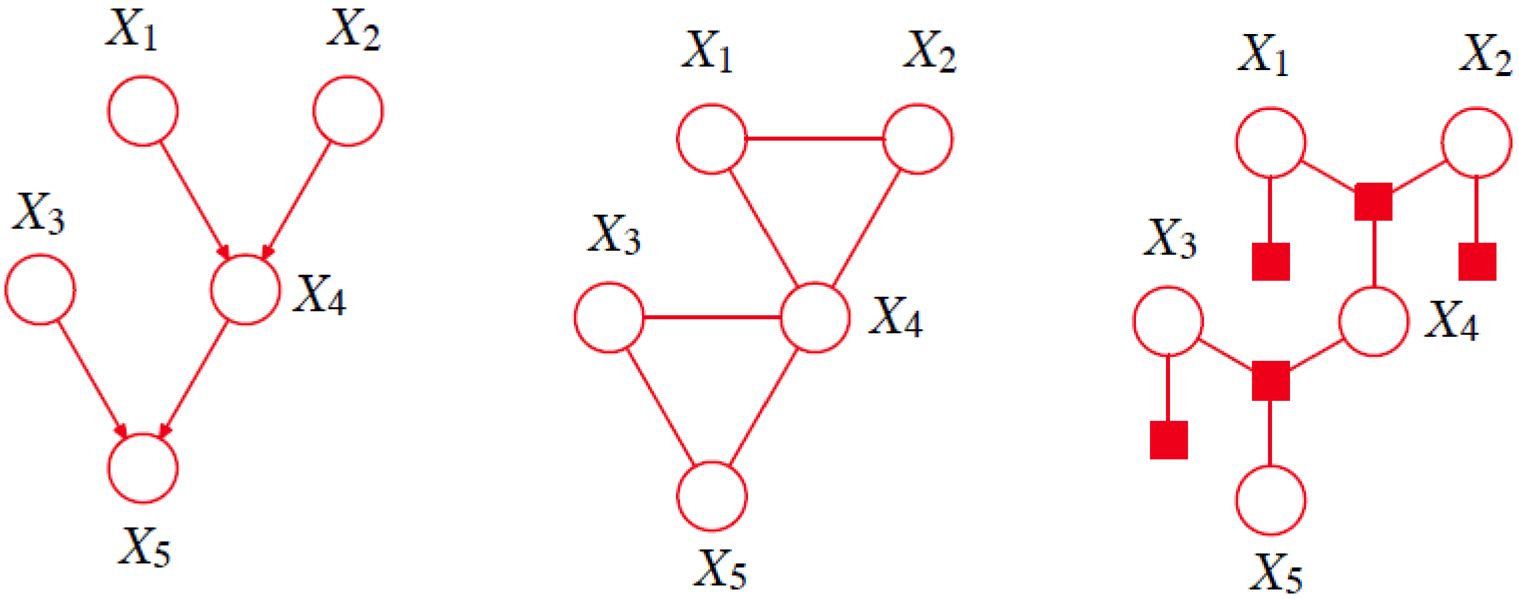
Factor Graphs from Polytrees

- In the case of a directed polytree,
  - *conversion to an undirected graph results in loops due to the moralization step,*
  - *whereas conversion to a factor graph results in a tree.*



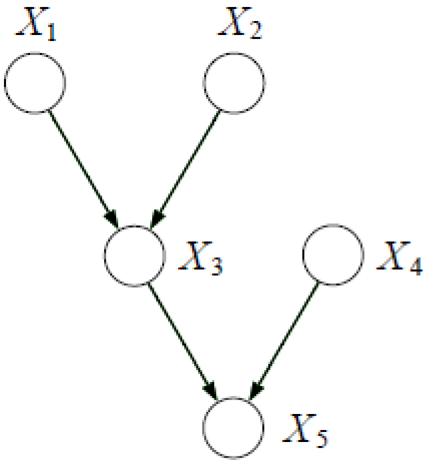
Factor Trees and Polytrees

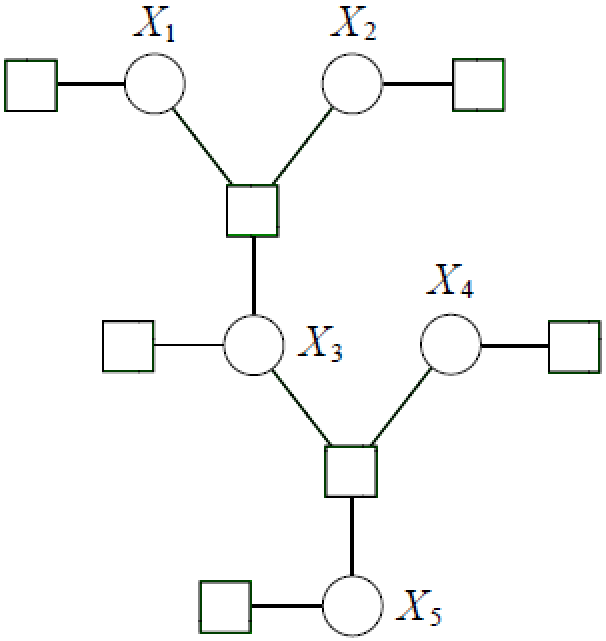
- *Factor trees are factor graphs that are trees, ignoring the distinction between variable nodes and factor nodes*
- Directed and undirected trees can trivially be represented as factor trees
- Polytrees can also be represented as factor trees



SUM-PRODUCT Applied to Factortrees

- The SUM-PRODUCT algorithm applies to factor trees.





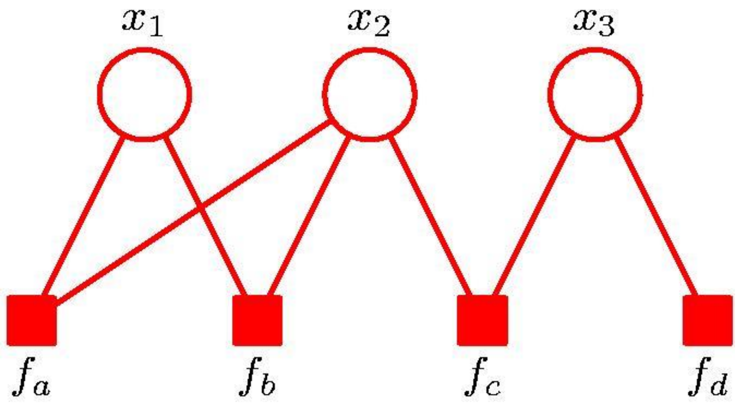
Factor Graphs

- Let us write the joint distribution over a set of variables in the form of a product of factors

$$p(x) = \prod_s f_s(x_s)$$

- For example, a distribution below can be expressed as a factor graph shown in the figure.

$$p(x) = f_a(x_1, x_2)\, f_b(x_1, x_2)\, f_c(x_2, x_3)\, f_d(x_3)$$



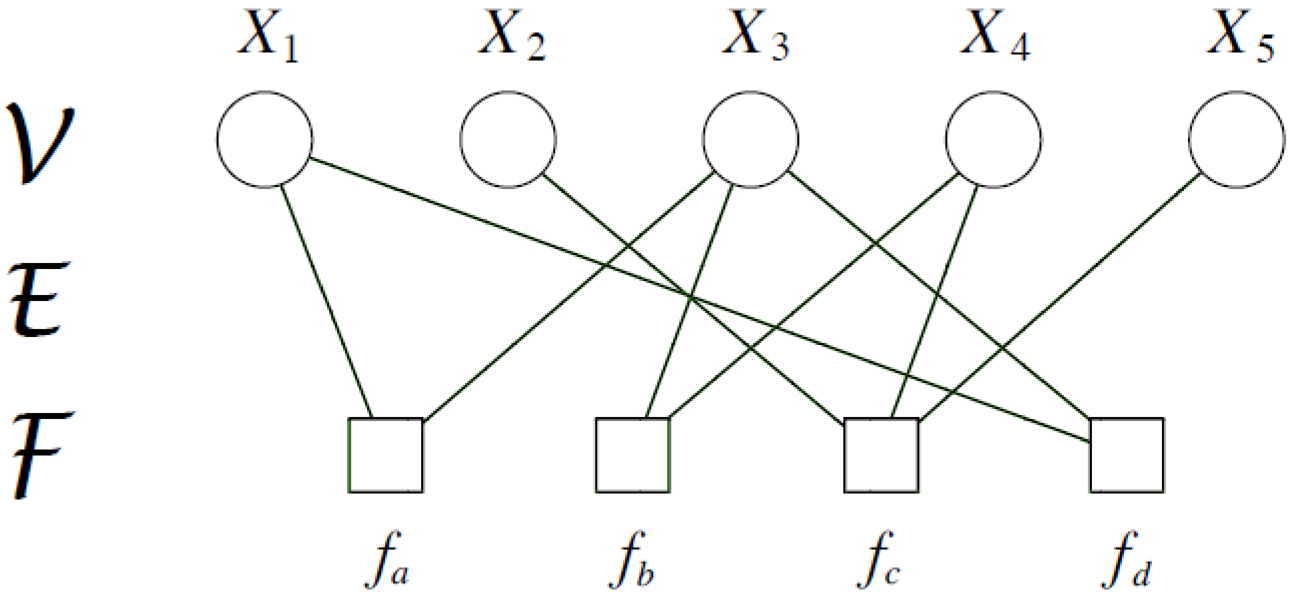
Circles: random variables

Filled dots: Factors in the joint distribution

Neighbors: Two nodes are neighbors if they share a factor

Bipartite Factor Graphs

- The graph is denoted $\mathcal{G}(\mathcal{V}, \mathcal{F}, \mathcal{E})$, with variables $\mathcal{V}$, factors $\mathcal{F}$, and edges $\mathcal{E}$. For example:

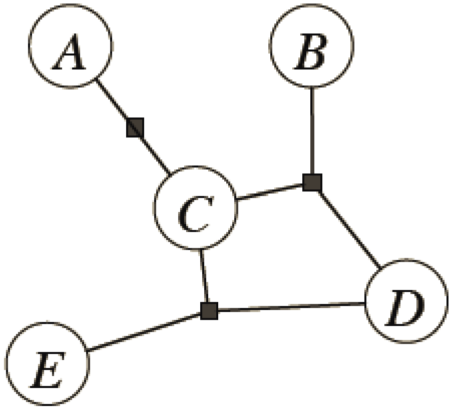


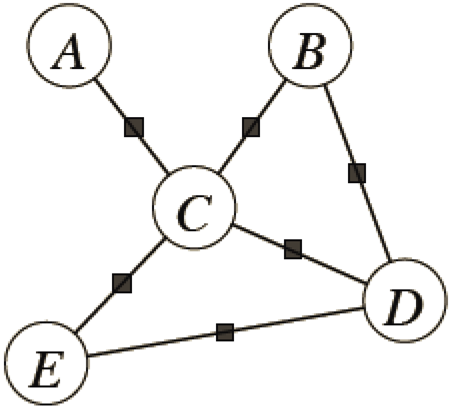
$$f(x_1, x_2, x_3, x_4, x_5) = f_a(x_1, x_3)\, f_b(x_3, x_4)\, f_c(x_2, x_4, x_5)\, f_d(x_1, x_3)$$

$$\mathcal{C} = \{1,3\}, \{3,4\}, \{2,4,5\}, \{1,3\}$$

A bipartite graph is a set of **graph** vertices decomposed into two disjoint sets such that no two **graph** vertices within the same set are adjacent.

Factor Graphs





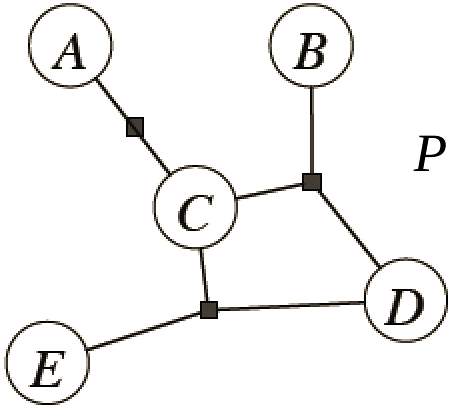
$$P(A,B,C,D,E) = \frac{1}{Z} g_1(A,C)\, g_2(B,C,D)\, g_3(C,D,E)$$

$$P(\mathrm{A,B,C,D,E}) = \frac{1}{Z} \mathrm{g}_1(\mathrm{A,C})\, \mathrm{g}_2(\mathrm{B,C})\, \mathrm{g}_3(\mathrm{C,D})\, \mathrm{g}_4(\mathrm{B,D})\, \mathrm{g}_5(\mathrm{C,E})\, \mathrm{g}_6(\mathrm{D,E})$$

- ❑ The g_i are non-negative functions of their arguments, and Z is a normalization constant e.g. in the fig. on the left, if all variables are discrete and take values in $\mathcal{A} \times \mathcal{B} \times \mathcal{C} \times \mathcal{D} \times \mathcal{E}$, then

$$Z = \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}} \sum_{d \in \mathcal{D}} \sum_{e \in \mathcal{E}} g_1(A=a, C=c)\, g_2(B=b, C=c, D=d)\, g_3(C=c, D=d, E=e)$$

Factor Graphs and CI Relations



$$P(A,B,C,D,E)=\frac{1}{Z}g_1(A,C)g_2(B,C,D)g_3(C,D,E)$$

- A path is a sequence of neighboring nodes.

- $X \perp\!\!\!\perp Y \mid V$ if every path between X and Y contains some node $V \in \mathcal{V}$

- Given the neighbors of X, the variable X is conditionally independent of all other variables (same as in undirected graphs):

$$X \perp Y \mid ne(X),\ \forall Y \notin \left\{X \cup ne(X)\right\}$$

Every path from X to Y has to go through its neighbors.

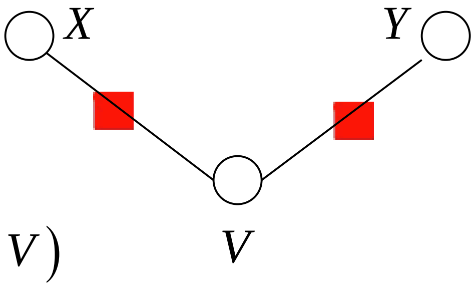
Conditional Independence and Factorization

- Lets consider the following conditional independence:

$$X \perp Y \mid V \Leftrightarrow p(X \mid Y,V) = p(X \mid V)$$

- This independence relation is represented with the factorization:

$$P(X,Y,V) = \frac{1}{Z} g_1(X,V)\, g_2(Y,V)$$



- Indeed:

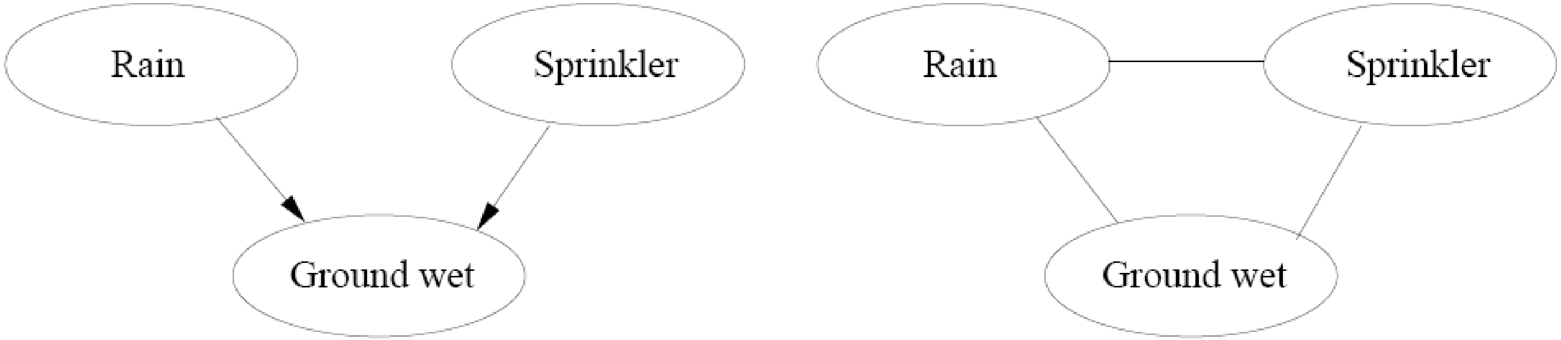
and $$P(Y,V) = \sum_X P(X,Y,V) = \frac{1}{Z} \sum_X g_1(X,V) g_2(Y,V)$$

$$P(X \mid Y,V) = \frac{P(X,Y,V)}{P(Y,V)} = \frac{\frac{1}{Z} g_1(X,V)\, g_2(Y,V)}{\frac{1}{Z} \sum_X g_1(X,V) g_2(Y,V)} = \frac{g_1(X,V)}{\sum_X g_1(X,V)} (\text{independent of } Y)$$

- Once more *we go from factorization to independence relations.*

Problems with Undirected Graphs & Factor Graphs

- In UGs and FGs, many useful independencies are unrepresented—two variables are connected merely because some other variable depends on them.
- This highlights the difference between marginal independence and conditional independence.



- R and S are marginally independent (i.e. given nothing), but they are conditionally dependent given G. This relation cannot be represented with UG or FGs.
- “Explaining Away”: Observing that the spinkler is on, would explain away the observation that the ground was wet, making it less probable that it rained.

D-Map, I-Map and Perfect Map

- **D map:** A graph is said to be a D map (for 'dependency map') of a distribution if every conditional independence statement satisfied by the distribution is reflected in the graph.

  A completely disconnected graph (no links) will be a trivial D map for any distribution.

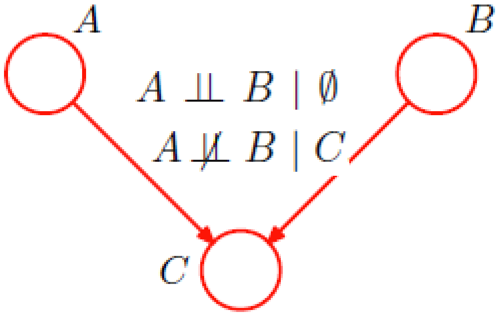
- **I map:** every conditional independence statement implied by a graph is satisfied by a specific distribution, then the graph is said to be an I map (for 'independence map') of that distribution.

  Clearly a fully connected graph will be a trivial I map for any distribution.

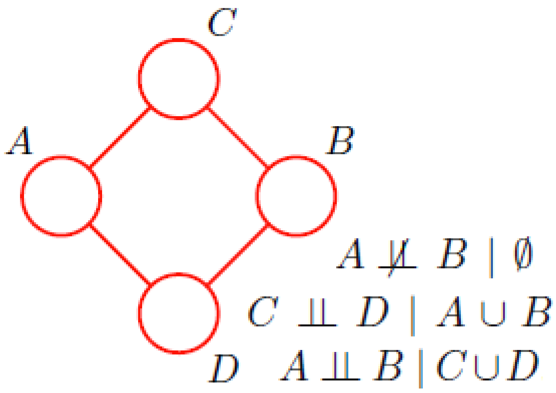
- **Perfect map:** every conditional independence property of the distribution is reflected in the graph, and vice versa.

Venn Diagram

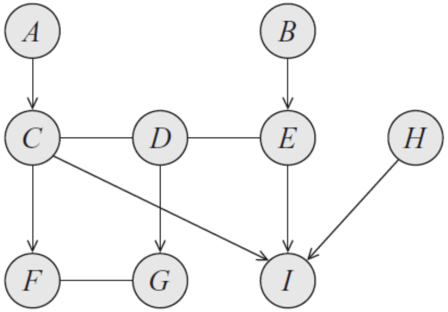
- Let P be the set of all distributions over a set of variables. The Venn diagram consists:
  - the set of distributions such that for each distribution there exists a directed graph that is a perfect map(D).
  - the set of distributions such that for each distribution there exists an undirected graph that is a perfect map(U).
  - Other distributions (chain graphs) for which neither directed nor undirected graphs offer a perfect map.
- *Chain graphs represent perfect maps for distributions broader than those corresponding to either directed or undirected graphs*.
- There are distributions that even chain graphs cannot provide a perfect map.

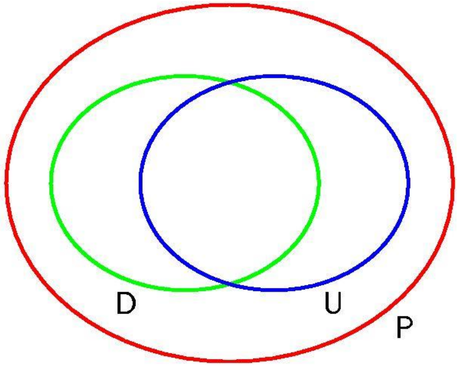


A
B
$A \perp\!\!\!\perp B \mid \emptyset$
$A \not\!\perp\!\!\!\perp B \mid C$
C



C
A
B
$A \not\!\perp\!\!\!\perp B \mid \emptyset$
$C \perp\!\!\!\perp D \mid A \cup B$
D
$A \perp\!\!\!\perp B \mid C \cup D$





- Lauritzen, S. and N. Wermuth (1989). Graphical models for association between variables, some of which are qualitative some quantitative. *Annals of Statistics* **17**, 31–57.
- Frydenberg, M. (1990). The chain graph Markov property. *Scandinavian Journal of Statistics* **17**, 333–353

Summary: Factor Graphs

- *Factor Graphs* are closely related to directed and undirected graphical models, but *start with factorization rather than with CI*
- They generalize both directed and undirected graphical models
- A slightly modified sum-product algorithm works for *factor trees*
- This turns out to be a simple way of *extending sum-product to polytrees. Exact Belief propagation is NP hard but in polytrees takes linear time.*
- Factor Graphs also provide a gateway to factor analysis, probabilistic PCA, Kalman filters, etc.

- Frey, B. J. and D. J. C. MacKay (1998). A revolution: Belief propagation in graphs with cycles. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems*, Volume 10. MIT Press.
- Kschischnang, F. R., B. J. Frey, and H. A. Loeliger (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* **47**(2), 498–519.

Summary: Factor Graphs

- Given variables $\{x_1, ..., x_n\}$, let $\mathcal{C}$ be a (multi)set of subsets of the indices $\{1, ..., n\}$

- For example, $\mathcal{C} = \{\{1,3\}, \{3,4\}, \{2,4,5\}, \{1,3\}\}$ for $\{x_1, ..., x_5\}$

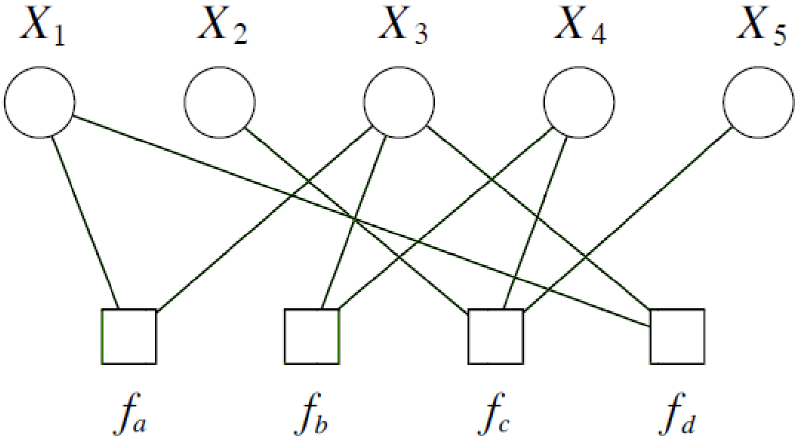
- Let there be a function $f_s(x_{C_s})$ associated with each $C_s \in \mathcal{C}$. It is called a factor

- Let the multivariate function f be defined:

$\mathcal{V}$

$\mathcal{E}$

$\mathcal{F}$



$$f(x_1,..,x_n) = \prod_{C_s \in \mathcal{C}} f_x \;\; x_{C_s}$$

- *This function need not be a probability distribution*.

The Sum-Product Algorithm and Factor Graphs

- We assume that our graph is an undirected tree or a directed tree or polytree, so that the corresponding factor graph has a tree structure. The sum-product algorithm is applied in all of these three cases.

- We convert the original graph into a factor graph so that we can deal with both directed and undirected models using the same framework.

- Objective:
  1) obtain an efficient, exact inference algorithm for finding marginals;
  2) When several marginals are required to allow computations to be shared efficiently.

- Key idea: *Distributive Law*

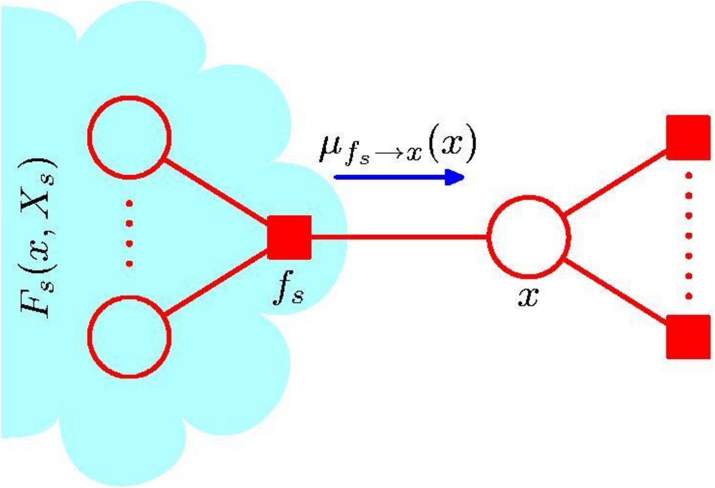
$$ab + ac = a\left(b + c\right)$$

The Sum-Product Algorithm

- We begin by considering the problem of finding the marginal $p(x)$ for particular variable node x. We consider factor-graphs with a tree structure.
- The marginal is

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

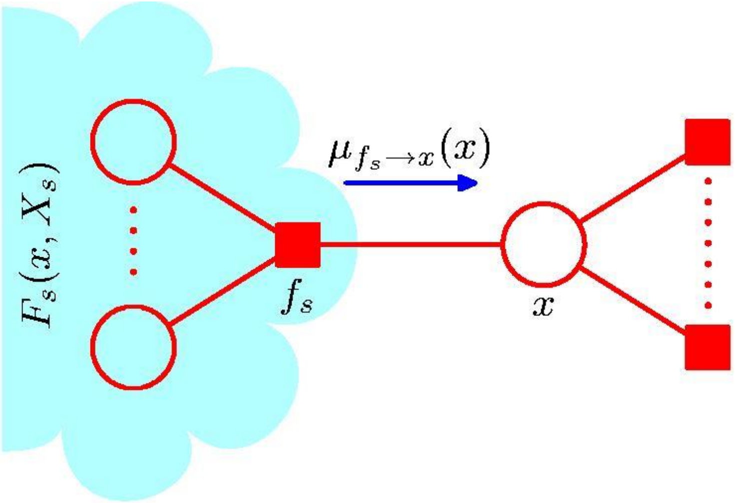
where $\boldsymbol{x} \setminus x$ denotes the set of variables in $\boldsymbol{x}$ with variable x omitted.



The joint distribution:

$$p(\mathbf{x}) = \prod_{s \in ne(x)} F_s(x, X_s)$$

The Sum-Product Algorithm

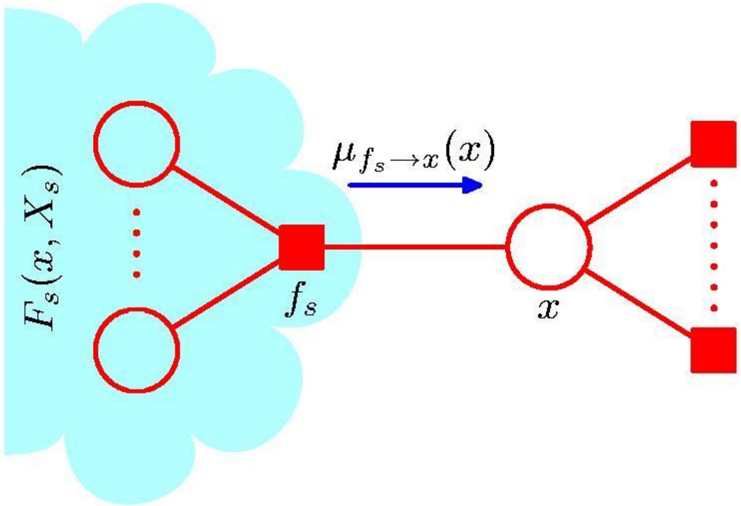


The joint distribution:

$$p(\mathbf{x}) = \prod_{s \in ne(x)} F_s(x, X_s)$$

- ❑ Here $ne(x)$ denotes the set of factor nodes that are neighbors of x
- ❑ X_s denotes the set of *all variables in the subtree connected to the variable node x via the factor node* f_s, and
- ❑ $F_s(x, Xs)$ represents the product of all the factors in the group associated with factor f_s.

Messages from Factor Nodes to Variable Nodes



Marginal :

$$p(x) = \prod_{s \in ne(x)} \left[\sum_{X_s} F_s(x, X_s) \right] = \prod_{s \in ne(x)} \mu_{f_s \to x}(x)$$

- Here we define a function

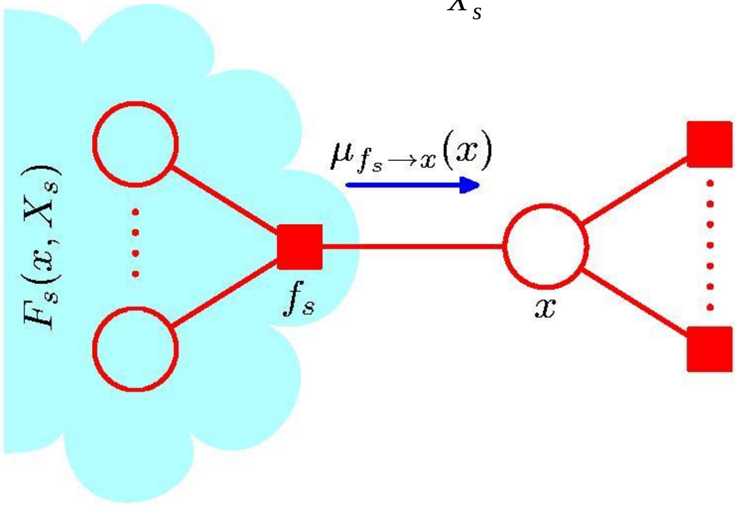
$$\mu_{f_s \to x}(x) = \sum_{X_s} F_s(x, X_s)$$

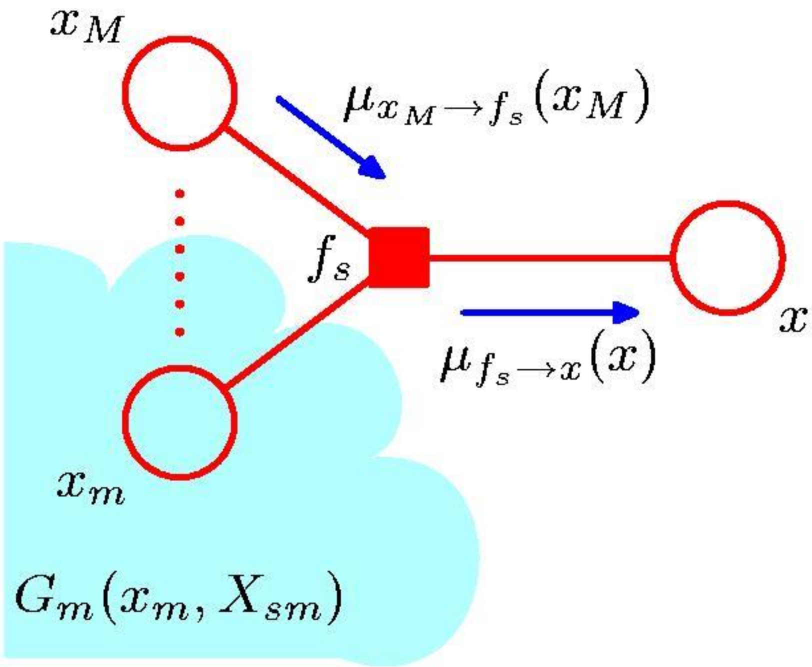
which can be viewed as message from the factor node f_s to the variable node x.

- We see that the required marginal $p(x)$ is given by the product of all the incoming messages arriving at node x.

The Sum-Product Algorithm

$$\mu_{f_s \to x}(x) = \sum_{X_s} F_s(x, X_s)$$





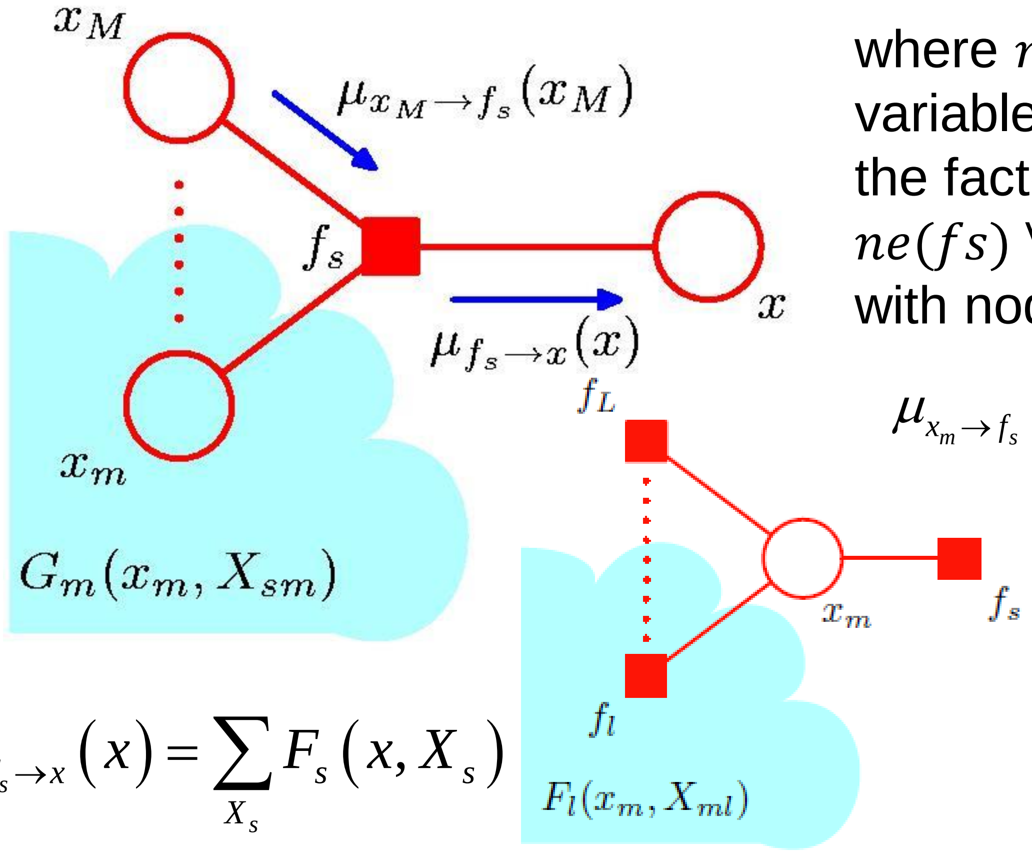
Here, we have denoted the variables associated with factor f_s, in addition to x, by $x_1, \ldots, x_M$.

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \cdots G_M(x_M, X_{sM})$$

The Sum-Product Algorithm

$$\mu_{f_s\to x}(x)=\sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{s\in ne(f_s)\backslash x}\left[\sum_{X_{x_m}} G_m(x_m,X_{sm})\right]$$

$$=\sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{s\in ne(f_s)\backslash x}\mu_{x_m\to f_s}(x_m)$$



where $ne(f_s)$ denotes the set of variable nodes that are neighbors of the factor node f_s, and $ne(fs)\setminus x$ denotes the same set but with node x removed.

$$\mu_{x_m\to f_s}(x_m)\equiv\sum_{X_{sm}} G_m(x_m,X_{sm})$$

$$=\sum_{X_{ml}}\prod_{l\in ne(x_m)\backslash f_s} F_l(x_m,X_{ml})$$

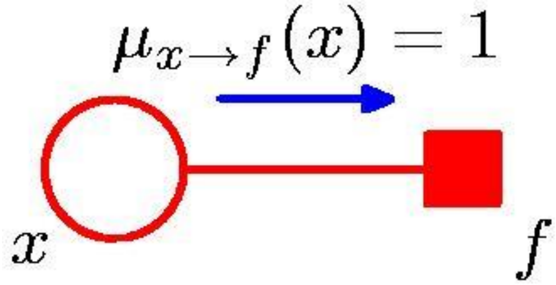
$$=\prod_{l\in ne(x_m)\backslash f_s}\mu_{f_l\to x_m}(x_m)$$

$$\mu_{f_s\to x}(x)=\sum_{X_s} F_s(x,X_s)$$

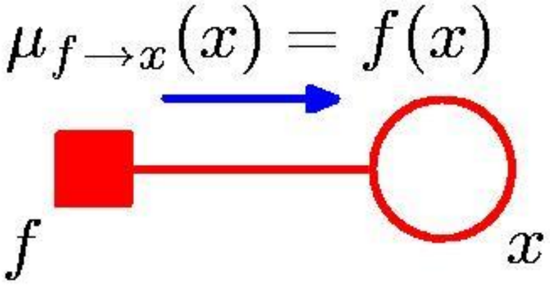
Initialization

- In order to start this recursion, we can view the node x as the root of the tree and begin at the leaf nodes.

- The sum-product algorithm begins with messages sent by the leaf nodes, which depend on whether the leaf node is (left) a variable node, or (right) a factor node.

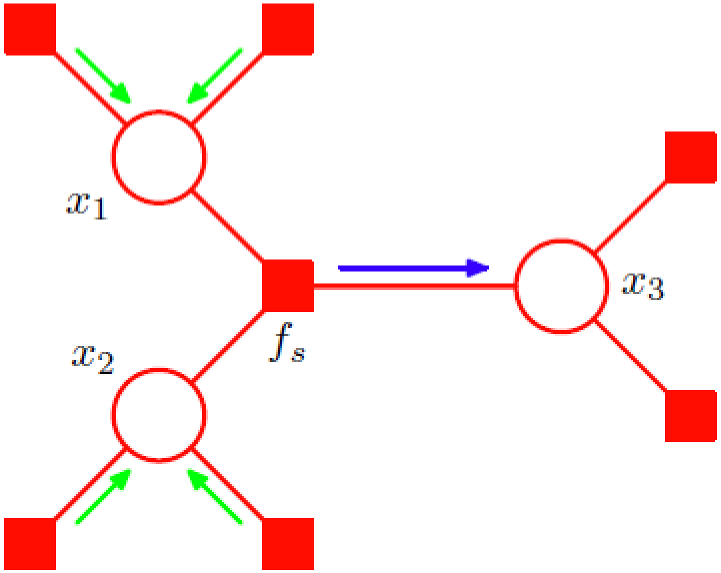


$$\mu_{x \to f}(x) = 1$$



$$\mu_{f \to x}(x) = f(x)$$

The Sum-Product Algorithm



- □ The sum-product algorithm can be *viewed purely in terms of messages sent out by factor nodes to other factor nodes*.

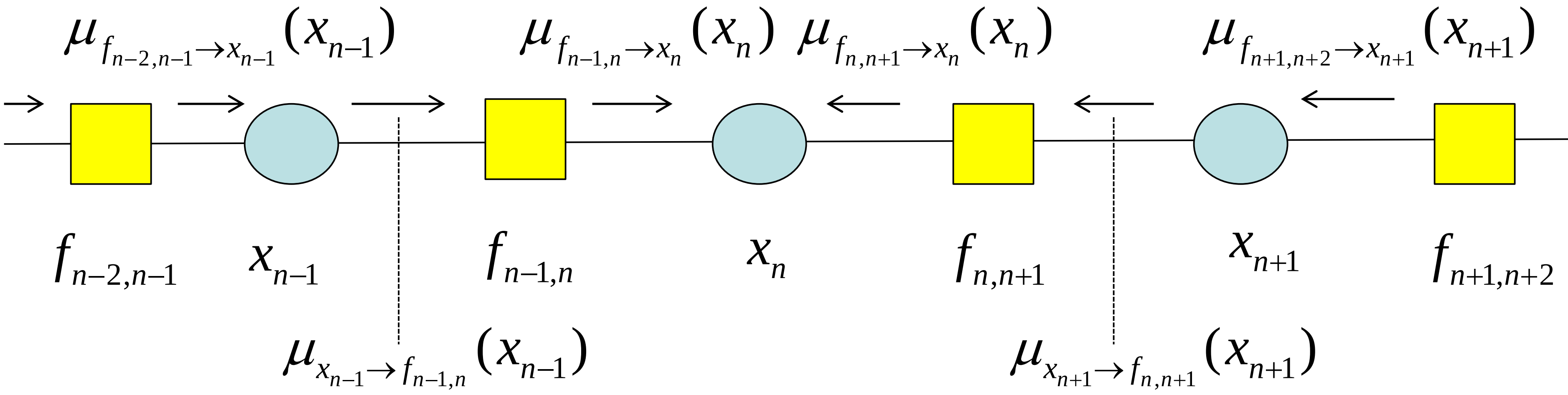
- □ The outgoing message shown by the blue arrow is obtained by
  - taking the product of all the incoming messages shown by green arrows,
  - multiplying by the factor f_s, and
  - marginalizing over the variables x_1 and x_2.

The Sum-Product Algorithm

- To compute local marginals:
  - Pick an arbitrary node as a root
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.

Sum-Product Algorithm for Markov Chains

- Applying the Sum-Product Algorithm to a Markov Chain gives as expected the same results as the elimination algorithm using α- and β-messages.


$\mu_{f_{n-2,n-1}\to x_{n-1}}(x_{n-1})$
$\mu_{f_{n-1,n}\to x_n}(x_n)$
$\mu_{f_{n,n+1}\to x_n}(x_n)$
$\mu_{f_{n+1,n+2}\to x_{n+1}}(x_{n+1})$
$f_{n-2,n-1}$
x_{n-1}
$f_{n-1,n}$
x_n
$f_{n,n+1}$
x_{n+1}
$f_{n+1,n+2}$
$\mu_{x_{n-1}\to f_{n-1,n}}(x_{n-1})$
$\mu_{x_{n+1}\to f_{n,n+1}}(x_{n+1})$

$$p(x_n)=\mu_{f_{n-1,n}\to x_n}(x_n)\mu_{f_{n,n+1}\to x_n}(x_n)=\sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\mu_{x_{n-1}\to f_{n-1,n}}(x_{n-1})\sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\mu_{x_{n+1}\to f_{n,n+1}}(x_{n+1})$$

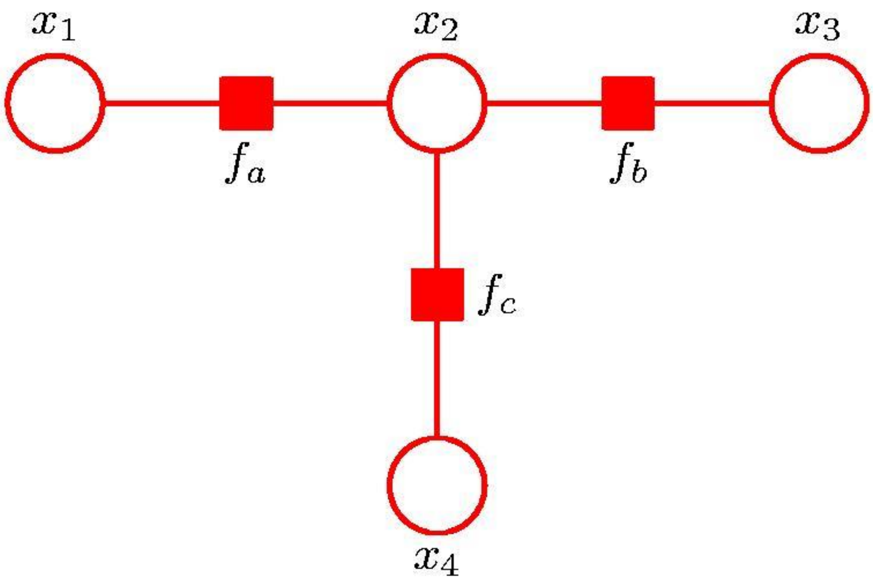
$$=\sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\underbrace{\mu_{f_{n-2,n-1}\to x_{n-1}}(x_{n-1})}_{\mu_\alpha(x_{n-1})}\sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\underbrace{\mu_{f_{n+1,n+2}\to x_{n+1}}(x_{n+1})}_{\mu_\beta(x_{n+1})}$$

- The end nodes are variable nodes so they send unit messages:

$$\mu_\alpha(x_2)=\sum_{x_1}\psi_{1,2}(x_1,x_2)\underbrace{\mu_{x_1\to f_{1,2}}(x_1)}_{1}=\sum_{x_1}\psi_{1,2}(x_1,x_2)$$

Sum-Product: Example

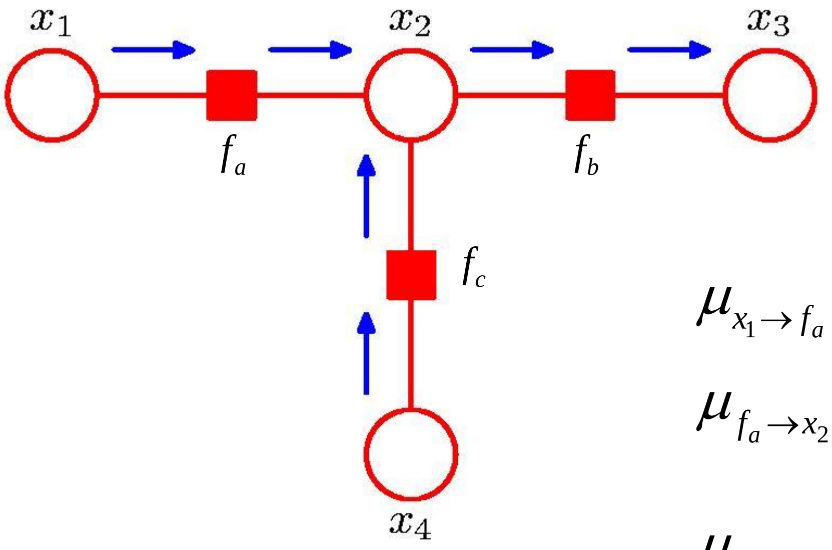
Example: a simple 4-node factor graph



Un-normalized joint distribution:

$$\tilde{p}(x) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

Sum-Product: Example



$$\mu_{x_1\to f_a}(x_1)=1$$

$$\mu_{f_a\to x_2}(x_2)=\sum_{x_1} f_a(x_1,x_2)$$

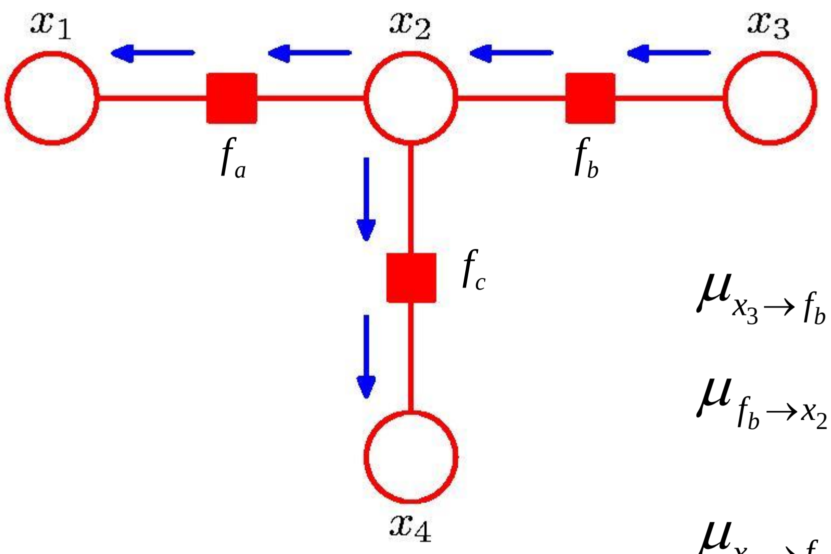
$$\mu_{x_4\to f_c}(x_4)=1$$

$$\mu_{f_c\to x_2}(x_2)=\sum_{x_4} f_c(x_2,x_4)$$

$$\mu_{x_2\to f_b}(x_2)=\mu_{f_a\to x_2}(x_2)\mu_{f_c\to x_2}(x_2)$$

$$\mu_{f_b\to x_3}(x_3)=\sum_{x_2} f_b(x_2,x_3)\mu_{x_2\to f_b}(x_2)$$

Sum-Product: Example



$$\mu_{x_3 \to f_b}(x_3) = 1$$

$$\mu_{f_b \to x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3)$$

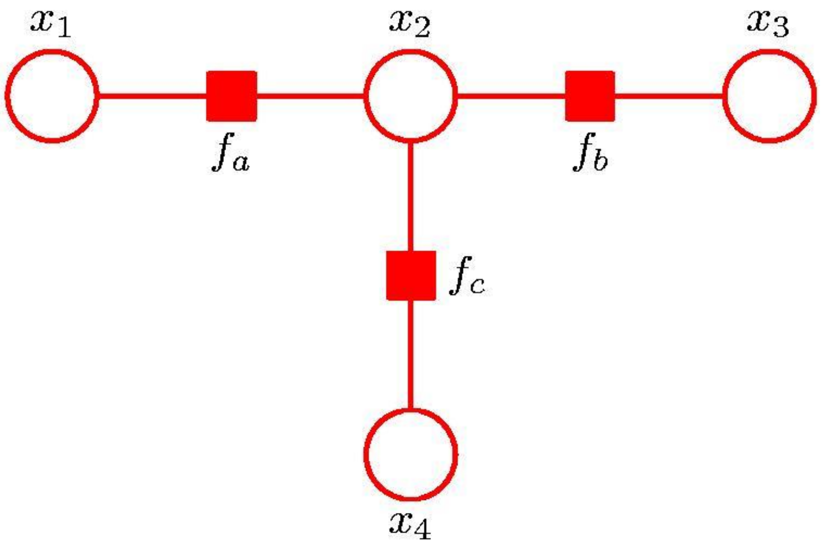
$$\mu_{x_2 \to f_a}(x_2) = \mu_{f_b \to x_2}(x_2)\, \mu_{f_c \to x_2}(x_2)$$

$$\mu_{f_a \to x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2)\, \mu_{x_2 \to f_a}(x_2)$$

$$\mu_{x_2 \to f_c}(x_2) = \mu_{f_a \to x_2}(x_2)\, \mu_{f_b \to x_2}(x_2)$$

$$\mu_{f_c \to x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4)\, \mu_{x_2 \to f_c}(x_2)$$

Sum-Product: Example



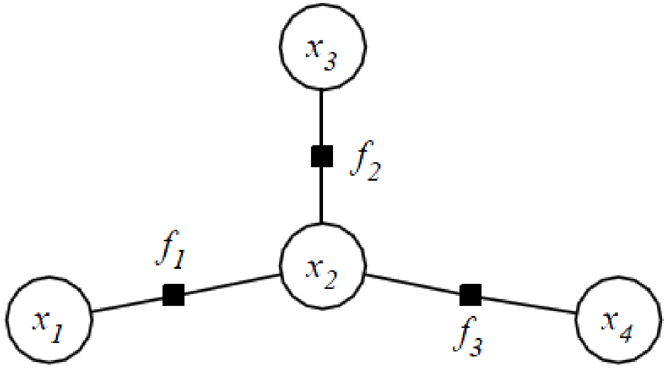
$$\tilde{p}(x_2) = \mu_{f_a\to x_2}(x_2)\mu_{f\,b\to x_2}(x_2)\mu_{f_c\to x_2}(x_2)$$

$$= \left[\sum_{x_1} f_a(x_1,x_2)\right]\left[\sum_{x_3} f_b(x_2,x_3)\right]\left[\sum_{x_4} f_c(x_2,x_4)\right]$$

$$= \sum_{x_1}\sum_{x_3}\sum_{x_4} f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_2,x_4)$$

$$= \sum_{x_1}\sum_{x_3}\sum_{x_4} \tilde{p}(\mathbf{x})$$

Propagation in Factor Graphs

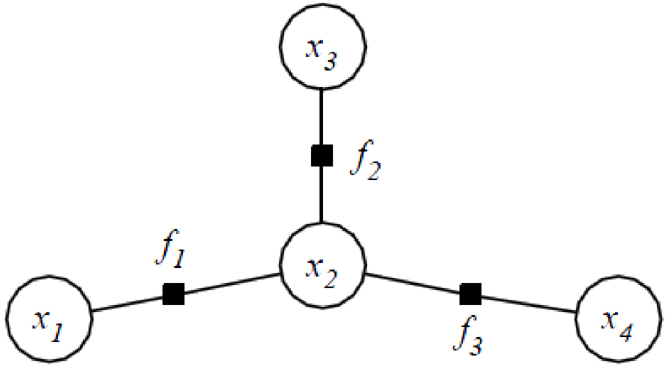


- ❑ Another example schedule of messages resulting in computing $p(x_4)$
- ❑ Initialize all messages to be 1

message direction	message value
$x_1 \to f_1$	$1(x_1)$
$x_3 \to f_2$	$1(x_3)$
$f_1 \to x_2$	$\sum_{x_1} f_1(x_1,x_2)1(x_1)$
$f_2 \to x_2$	$\sum_{x_3} f_2(x_3,x_2)1(x_3)$
$x_2 \to f_3$	$\left(\sum_{x_1} f_1(x_1,x_2)\right)\left(\sum_{x_3} f_2(x_3,x_2)\right)$
$f_3 \to x_4$	$\sum_{x_2} f_3(x_2,x_4)\left(\sum_{x_1} f_1(x_1,x_2)\right)\left(\sum_{x_3} f_2(x_3,x_2)\right)$

If a variable (here x_1) has only one factor as a neighbor, it can initiate message propagation

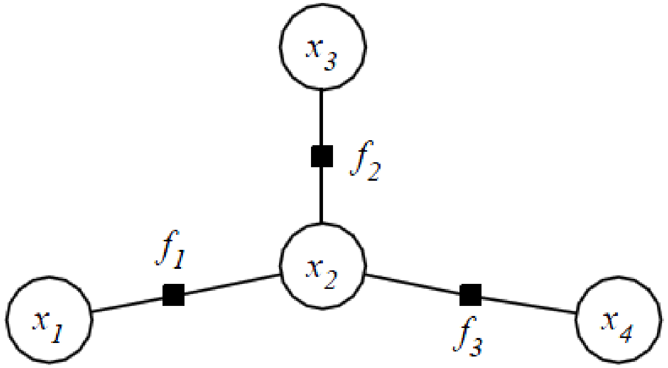
Propagation in Factor Graphs



- Once a variable has received all the messages from its neighboring factors, we can compute the probability of that variable by multiplying all the messages and renormalizing:

$$p(x) \propto \prod_{h \in ne(x)} \mu_{h \to x}(x)$$

Incorporating Evidence



- Initialize all messages to be 1
- An example schedule of messages resulting in computing $p(x_4|x_1 = a)$:

message direction	message value
$x_1 \to f_1$	$\delta(x_1 = a)$
$x_3 \to f_2$	$1(x_3)$
$f_1 \to x_2$	$\sum_{x_1} f_1(x_1, x_2)\delta(x_1 = a) = f_1(x_1 = a, x_2)$
$f_2 \to x_2$	$\sum_{x_3} f_2(x_3, x_2)1(x_3)$
$x_2 \to f_3$	$f_1(x_1 = a, x_2)\left(\sum_{x_3} f_2(x_3, x_2)\right)$
$f_3 \to x_4$	$\sum_{x_2} f_3(x_2, x_4)\, f_1(x_1 = a, x_2)\left(\sum_{x_3} f_2(x_3, x_2)\right)$

Marginal Associated with Each Factor

❑ The marginal distributions $p(x_s)$ over the sets of variables x_s associated with each of the factors $f_s(x_s)$ in a factor graph can be found by first running the sum-product message passing algorithm and then evaluating the required marginals.

The marginal: $$p(x_s) = \sum_{\mathbf{x}\backslash x_s} p(\mathbf{x})$$

$$p(x_s) \equiv \sum_{\mathbf{x}\backslash x_s} f_s(x_s) \prod_{i\in ne(f_s)} \prod_{j\in ne(x_i)\backslash f_s} F_j(x_i, X_{ij})$$

Product of all the factors in the group associated with factor j connected to node i. X_{ij} are all variables on the sub-tree ij.

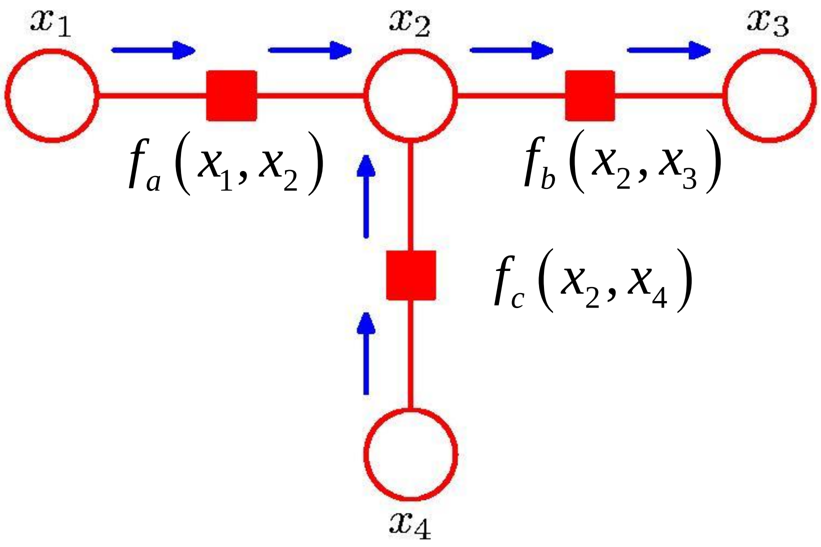
$$= f_s(x_s) \prod_{i\in ne(f_s)} \sum_{\mathbf{x}\backslash x_s} \prod_{j\in ne(x_i)\backslash f_s} F_j(x_i, X_{ij})$$

$$= f_s(x_s) \prod_{i\in ne(f_s)} \mu_{x_i\to f_s}(x_i)$$

Marginal Associated with Each Factor

- As an application, let us consider computing $p(x_1, x_2)$ in the factor graph below:

$$p(x_s) = f_s(x_s) \prod_{i \in ne(f_s)} \mu_{x_i \to f_s}(x_i)$$



$$p(x_1, x_2) = f_a(x_1, x_2)\mu_{x_1 \to f_a}(x_1)\mu_{x_2 \to f_a}(x_2)$$

$$= f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2)$$

$$= f_a(x_1, x_2)\mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2)$$

$$= f_a(x_1, x_2)\sum_{x_3} f_b(x_2, x_3)\sum_{x_4} f_c(x_2, x_4)$$

$$= \sum_{x_3}\sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) =$$

$$= \sum_{x_3}\sum_{x_4} \tilde{p}(x)$$

Marginal Not Associated with Factors

- Suppose we want to compute $p(x_a, x_b)$ where the set of variables x_a and x_b do not belong to the same factor.

$$p\left(x_a, x_b\right) = p\left(x_b \mid x_a\right) p(x_a)$$

- The marginal $p(x_a)$ can be computed by using the sum-product algorithm over all variables including x_b.

- To compute the conditional $p(x_b|x_a)$, fix the evidence x_a and for each of its allowed values run the sum-product algorithm to compute$p(x_b|x_a)$ by marginalizing over all variables except x_b and x_a (that remains at its fixed value).

Marginal Associated with each Variable Node

The marginal p(x_i) can also be written as the product of the incoming message along any one of the links with the outgoing message along the same link.

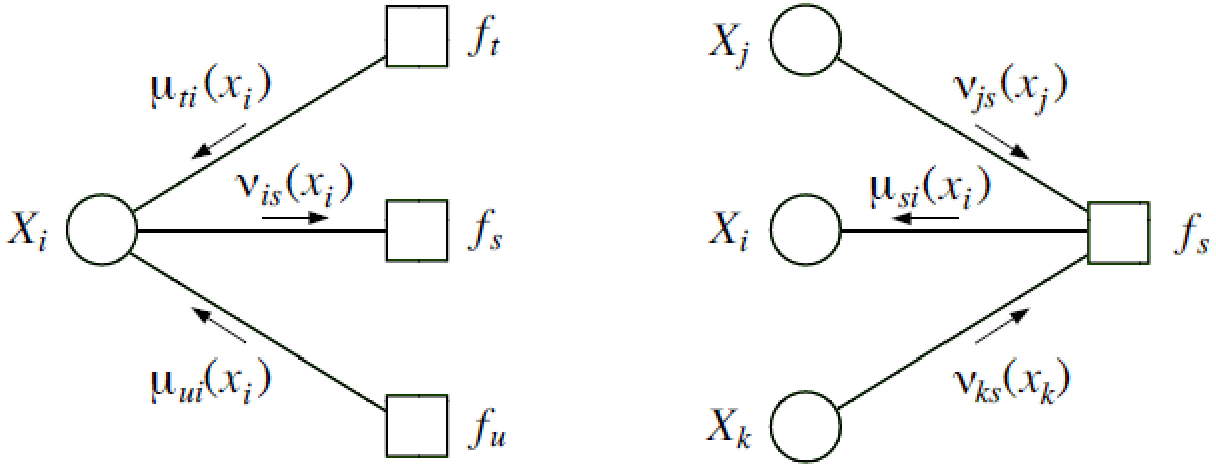
$$p(x_i) \equiv \prod_{s \in ne(x_i)} \mu_{f_s \to x_i}(x_i)$$

$$= \mu_{f_s \to x_i}(x_i) \prod_{t \in ne(x_i) \backslash f_s} \mu_{f_t \to x_i}(x_i)$$

$$= \mu_{f_s \to x_i}(x_i) \mu_{x_i \to f_s}(x_i)$$

Sum Product for Factor Trees

- We often use different notation for the two type of messages:
  - Variables→factors (ν)
  - Factors →Variables (μ)



- Both products of incoming messages but only variables require summing:

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

Sum Product Algorithm for a Factor-Tree

```
Sum-Product($\mathcal{T}$, E)
    Evidence(E)
    f=ChooseRoot($\mathcal{V}$)
    for s $\in \mathcal{N}$(f)
        $\mu$-Collect(f,s)
    for s $\in \mathcal{N}$(f)
        $\nu$-Distribute(f,s)
    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
        $\psi^E(x_i)=\psi(x_i)$
```

```
$\mu$-Collect(i,s)
    $for\ \ j \in \mathcal{N}(s)\setminus i$
        $\nu$-Collect(s,j)
    $\mu$-SendMessage(s,i)
```

```
$\nu$-Collect(s,i)
    $for\ t \in \mathcal{N}(i)\setminus s$
        $\mu$-Collect(i,t)
    $\nu$-SendMessage(i,s)
```

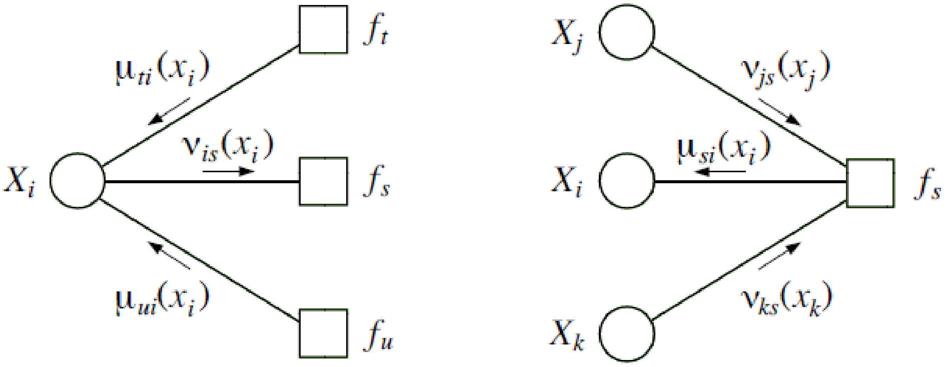
```
$\mu$-Distribute(s,i)
    $\mu$-SendMessage(s,i)
    $for\ t \in \mathcal{N}(i)\setminus s$
        $\nu$-Distribute(i,t)
```

```
$\nu$-Distribute(i,s)
    $\nu$-SendMessage(i,s)
    $for\ \ j \in \mathcal{N}(s)\setminus i$
        $\mu$-Distribute(s,j)
```

Sum Product Algorithm for a Factor-Tree

μ-SendMessage(s,i)

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



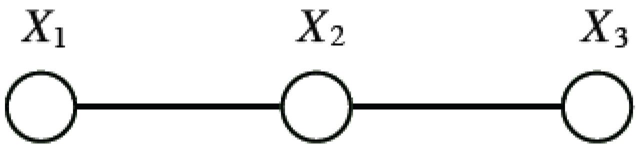
ν-SendMessage(j,i)

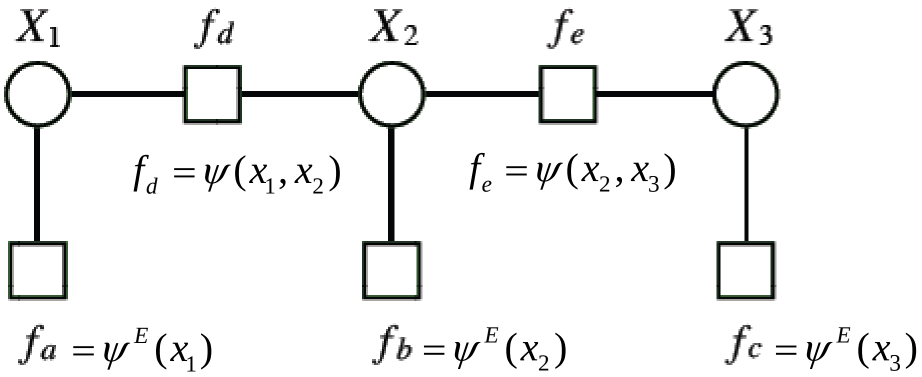
$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i)\setminus\{s\}} \mu_{ti}(x_i)$$

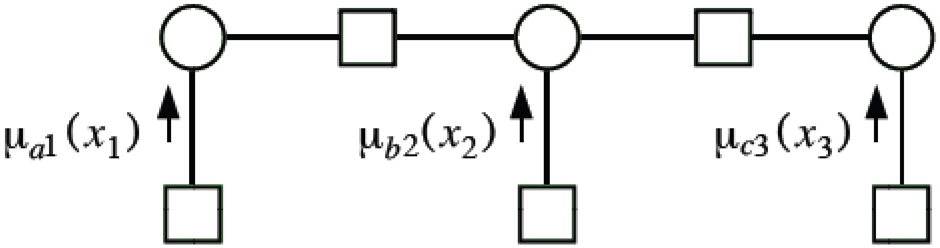
ComputeMarginal(i)

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

Example

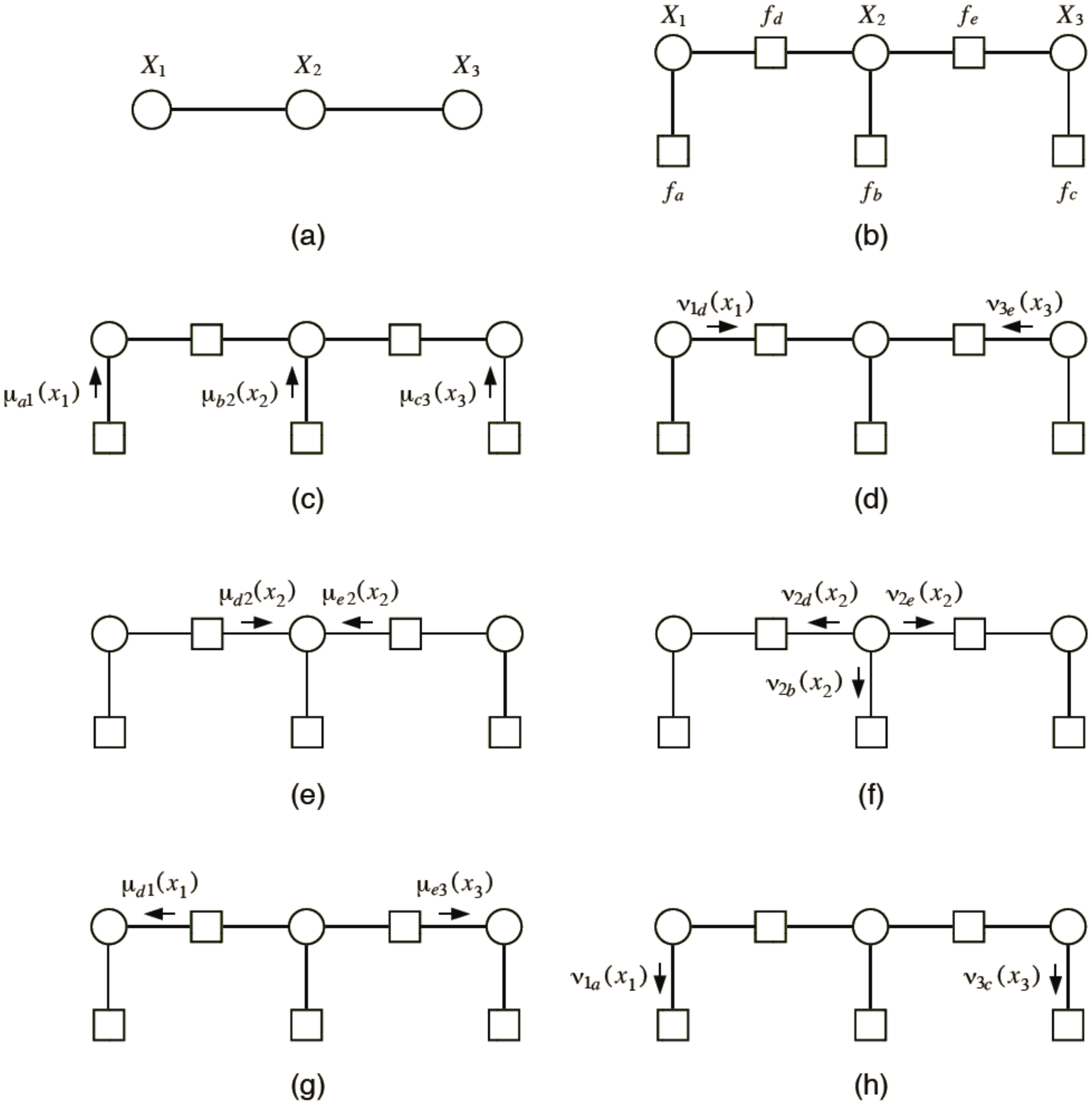






$$\mu_{a1}(x_1) = \sum_{x_{\mathcal{N}(a)\setminus\{1\}}} \left(f_a(x_{\mathcal{N}(a)}) \prod_{j\in\mathcal{N}(a)\setminus\{1\}} \nu_{ja}(x_j) \right) = f_a(x_1) = \psi^E(x_1),\ \mu_{b2}(x_2) = \psi^E(x_2),\ \mu_{c3}(x_3) = \psi^E(x_3)$$

Example



(a)

(b)

(c)

(d)

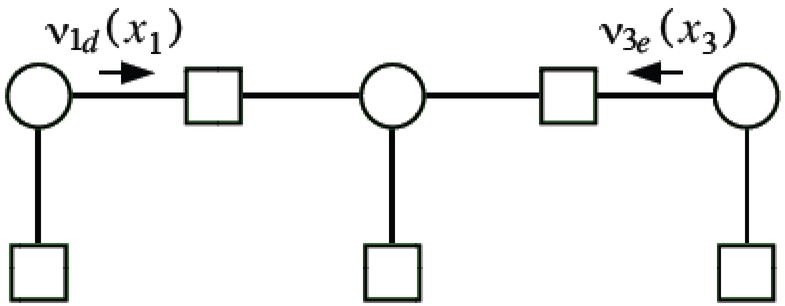
(e)

(f)

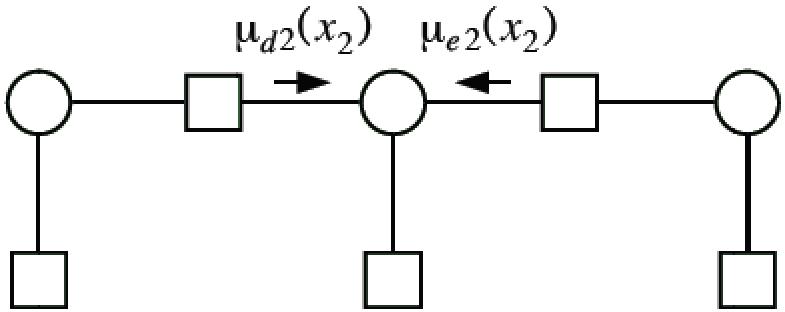
(g)

(h)

Example

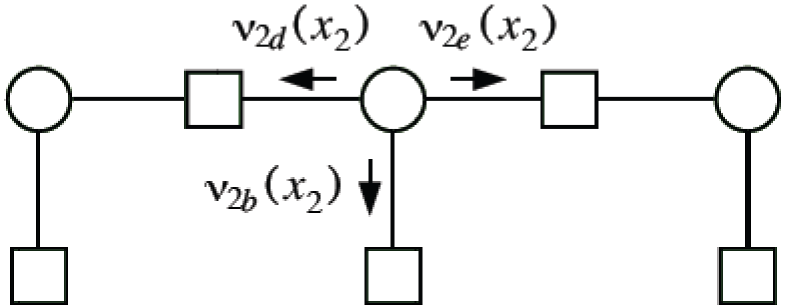


$$\nu_{1d}(x_1) = \prod_{t\in\mathcal{N}(1)\backslash\{d\}} \mu_{t1}(x_1) = \mu_{a1}(x_1) = \psi^E(x_1),\ \nu_{3e}(x_3) = \psi^E(x_3)$$



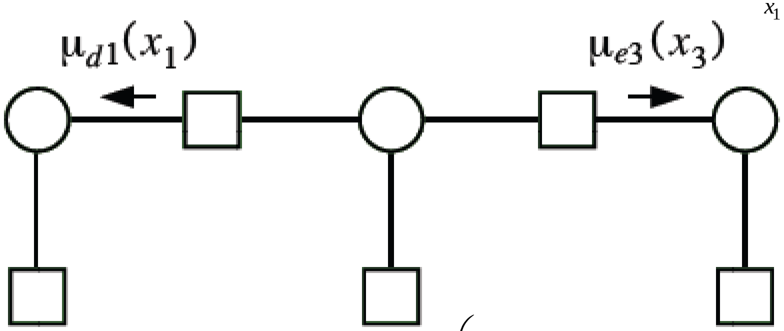
$$\mu_{d2}(x_2) = \sum_{x_{\mathcal{N}(d)\backslash\{2\}}} \left(f_d(x_{\mathcal{N}(d)}) \prod_{j\in\mathcal{N}(d)\backslash\{2\}} \nu_{jd}(x_j) \right) = \sum_{x_1} \psi(x_1,x_2)\psi^E(x_1),\ \mu_{e2}(x_2) = \sum_{x_3} \psi(x_2,x_3)\psi^E(x_3)$$

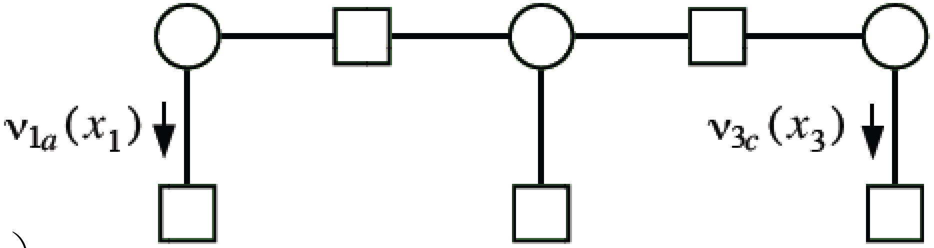
Example



$$\nu_{2d}(x_2) = \prod_{t\in\mathcal{N}(2)\backslash\{d\}} \mu_{t2}(x_2) = \psi^E(x_2)\sum_{x_3}\psi^E(x_3)\psi(x_2,x_3),\ \nu_{2e}(x_2) = \psi^E(x_2)\sum_{x_1}\psi^E(x_1)\psi(x_1,x_2)$$

$$\nu_{2b}(x_2) = \sum_{x_1}\psi^E(x_1)\psi(x_1,x_2)\sum_{x_3}\psi^E(x_3)\psi(x_2,x_3)$$





$$\mu_{d1}(x_1) = \sum_{x_{\mathcal{N}(d)\backslash\{1\}}}\left(f_d(x_{\mathcal{N}(d)})\prod_{j\in\mathcal{N}(d)\backslash\{1\}}\nu_{jd}(x_j)\right) = \sum_{x_2}\psi^E(x_2)\psi(x_1,x_2)\sum_{x_3}\psi^E(x_3)\psi(x_2,x_3) = \nu_{1a}(x_1)$$

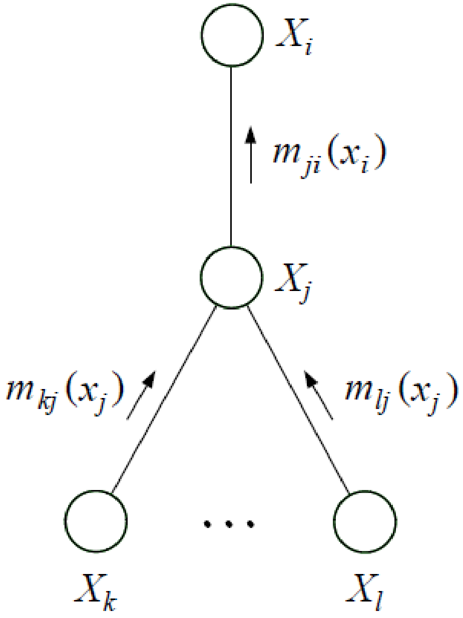
$$\mu_{e3}(x_1) = \sum_{x_2}\psi^E(x_1)\psi(x_1,x_2)\sum_{x_3}\psi^E(x_2)\psi(x_2,x_3) = \nu_{3c}(x_3)$$

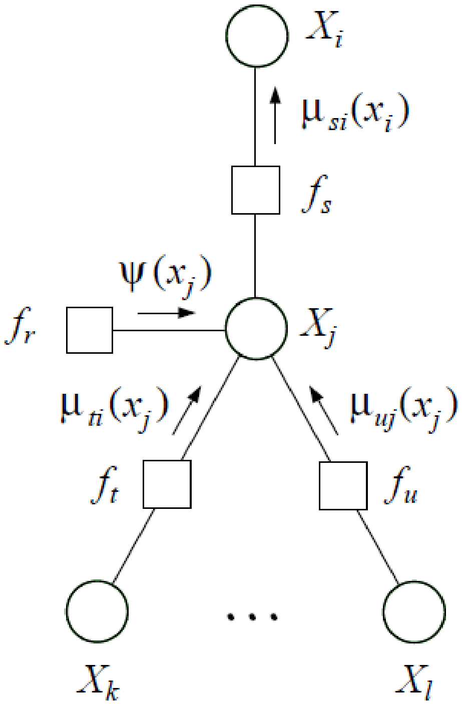
Note that these messages are the same as the corresponding messages that would pass in a run of the SUM-PRODUCT algorithm in the corresponding undirected graph, e.g.

$$\mu_{d1}(x_1) = m_{21}(x_1),\ \mu_{e3}(x_3) = m_{23}(x_3)$$

Equivalence with the Sum-Product Algorithm

- Converting an undirected graph to a factor graph gives m-messages that are the same as the m messages obtained from the SUM-PRODUCT algorithm.





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

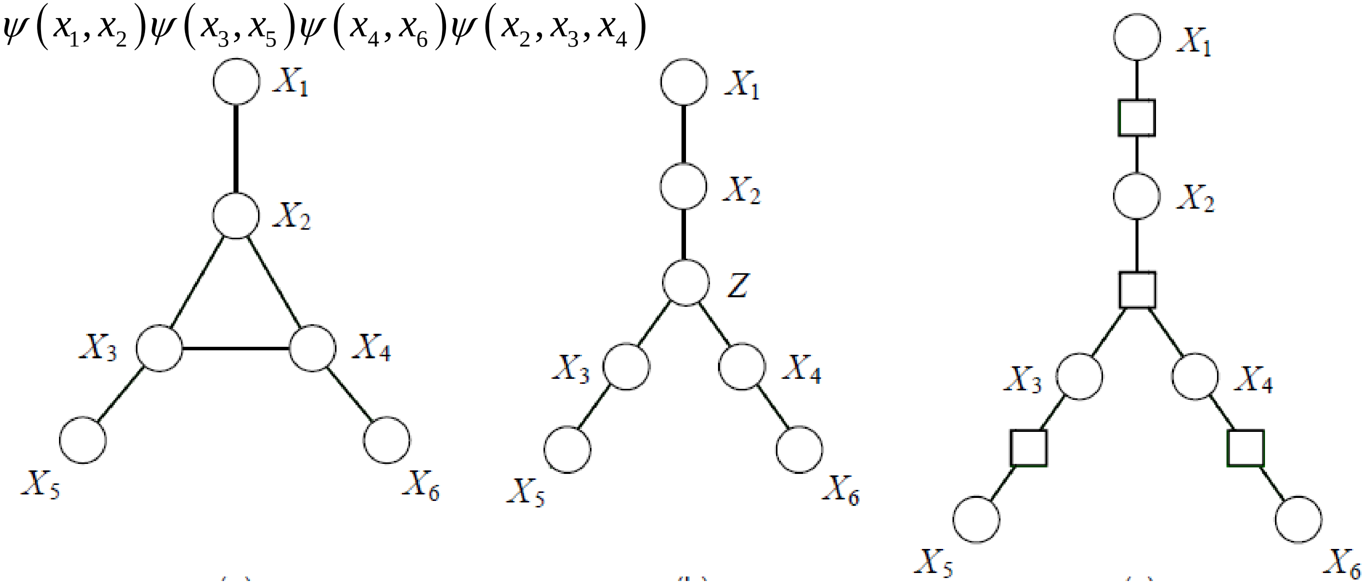
$$= \sum_{x_j} \psi(x_i, x_j) \psi^E(x_j) \prod_{t\in\mathcal{N}'(j)\backslash\{s\}} \mu_{tj}(x_j)$$

In the N'(j) omitting the singleton factor

SUM-PRODUCT Applied to Factor Trees

- If a graph (directed or undirected) is originally a tree, there is little to gain by transforming it to factor graph.
- However, there is significant merit in transforming to factor graphs *various `tree-like' graphs.* The SUM-PRODUCT applies directly to factor trees.

$$p(x) \propto \psi\left(x_1, x_2\right)\psi\left(x_3, x_5\right)\psi\left(x_4, x_6\right)\psi\left(x_2, x_3, x_4\right)$$



(a) An undirected graph with an unfactorized potential $\psi(x_2,x_3,x_4)$

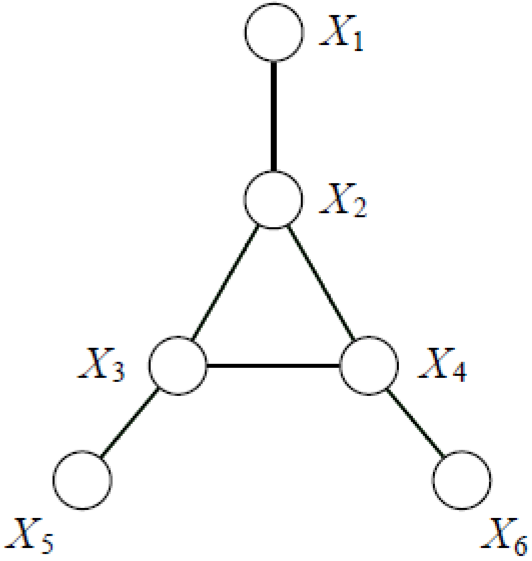
(b) An equivalent undirected model using a super variable Z (range the Cartesian product of the range of X_2,X_3,X_4). Create new potentials $\psi\left(x_2,Z\right)\psi\left(x_3,Z\right)\psi\left(x_4,Z\right)\psi\left(Z\right)$

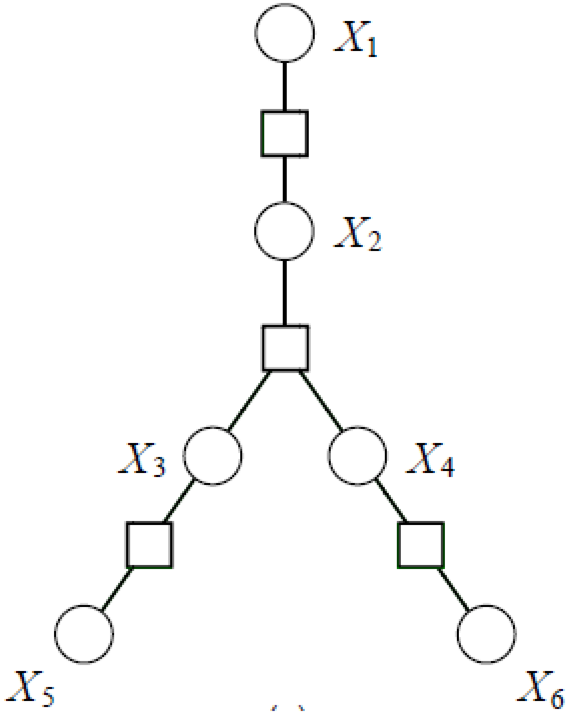
(c) An equivalent factor graph that is a factor-tree. No need for new potentials.

SUM-PRODUCT Applied to Factor Trees

- If the variables in the undirected graph can be clustered in non-overlapping cliques, and the parameterization of each clique is a general non-factorized potential, then the corresponding factor graph is a tree and the SUM-PRODUCT algorithm applies.

$$p(x) \propto \psi(x_1, x_2)\psi(x_3, x_5)\psi(x_4, x_6)\psi(x_2, x_3, x_4)$$





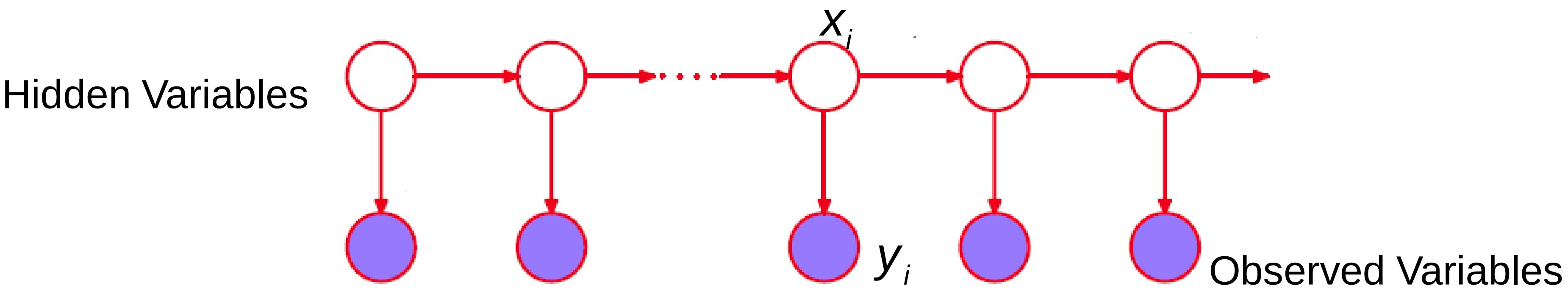
Application in State Space Models

- We will see in another lecture tht nference in HMM involves one forward and one backward pass

$$P(Y_1 = y_1, \ldots, Y_m = y_m, X_1 = x_1, \ldots, X_m = x_m) =$$

$$P\left(X_1 = x_1\right)\prod_{j=2}^{m} P\left(X_j = x_j \mid X_{j-1} = x_{j-1}\right)\prod_{j=1}^{m} P\left(Y_j = y_j \mid X_j = x_j\right)$$

- The computational cost grows linearly with the length of the chain.
- Similarly for the Kalman Filter



Belief Propagation for DAGS

- Belief propagation is an algorithm for exact inference on directed graphs without loops and is equivalent to a special case of the sum-product algorithm.

- We will discuss in the following lecture Belief Propagation for general directed graphs.

- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann.
- Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local computations with probabalities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society* **50**, 157–224.