Inference in Probabilistic Graphical Models: Chains and Trees, 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/

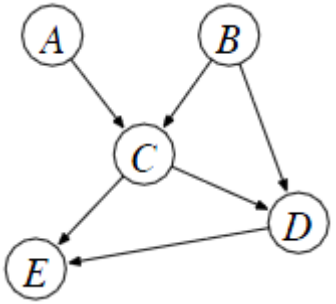
February 7, 2018

Contents

- Introduction, Inference on a Chain, Message Passing: Chapman-Kolmogorov Equations, Computing All Marginals, Max Product Algorithm on a Chain
- Inference on Trees, Parametrization, Evidence Potentials, Pruning Leaves, Elimination in Trees, The Sum-Product Algorithm, Message Passing Protocol, Bottom Up Dynamic Programming, Postorder, Summary of the Algorithm
- Factor Graphs, Sum Product Algorithm for Factor Trees
- Sum-Product Like Algorithm for Polytrees

- 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.

Inference in DAGs



- Consider the graph shown which represents:

$$p(A,B,C,D,E) = p(A)p(B)p(C|A,B)p(D|B,C)p(E|C,D)$$

- **Inference:** evaluate the probability distribution over some set of variables, given the values of another set of variables.
- For example, how can we compute P(A|C = c)? Assume each variable is binary.

Naive method (total 20 operations):

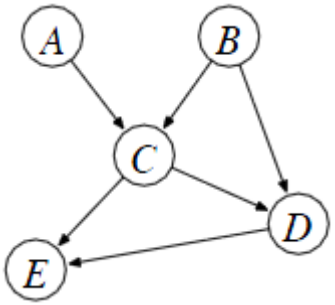
$$p(A, C=c) = \sum_{B,D,E} p(A, B, C=c, D, E)$$ [16 operations, 8 sums on B, D, E for each of the two values of A]

$$p(C=c) = \sum_{A} p(A, C=c)$$ [2 operations]

$$p(A|C=c) = \frac{p(A, C=c)}{p(C=c)}$$ [2 operations one for each of the values of A]

Inference in DAGs

- Consider the right graph which represents:



$$p(A,B,C,D,E) = p(A)p(B)p(C|A,B)p(D|B,C)p(E|C,D)$$

Computing p(A|C = c).

- **More efficient method (total 8 operations):**

$$p(A, C=c) = \sum_{B,D,E} p(A)\,p(B)\,p(C=c|A,B)\,p(D|B,C=c)\,p(E|C=c,D)$$

$$= \sum_{B} p(A)\,p(B)\,p(C=c|A,B) \underbrace{\sum_{D} p(D|B,C=c)}_{1} \underbrace{\sum_{E} p(E|C=c,D)}_{1}$$

$$= \sum_{B} p(A)\,p(B)\,p(C=c|A,B)$$

[4 operations, sum on 2 values of B for each of the 2 values of A]

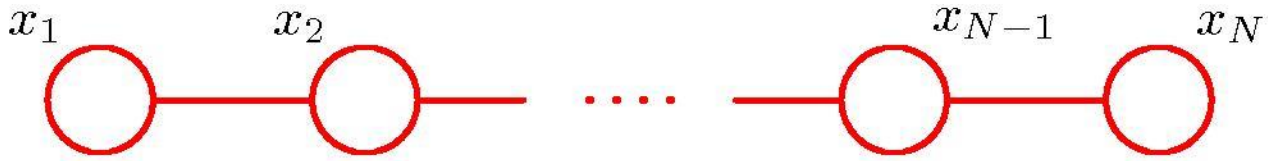
- Total: 4+2+2 = 8 operations
- Belief propagation methods use the conditional independence relationships in a graph to do efficient inference (for singly connected graphs, exponential gains in efficiency!).

Moving Beyond the Elimination Algorithm

- The Elimination Algorithm allow us to compute a single marginal.
- The Sum-Product algorithm (also known as Belief Propagation) will allow us to compute all marginals and conditionals. They key idea of the algorithm is the *development of a calculus of intermediate factors common to many elimination orderings.* We will also discuss computing maximum a posteriori probabilities.
- However, the Sum-Product algorithm *works only for trees and tree-like graphs* (using the factor approach we can handle polytrees – a class of directed graphical models where nodes have multiple parents).
- We will generalize in another lecture to the junction tree algorithm (exact inference problem).
- Before starting our discussion with trees, we first introduce ideas through a one-dimensional chain like graph.

Inference on a Chain

- Consider an undirected chain (many missing links, tree structure)



- The marginal distribution is given as

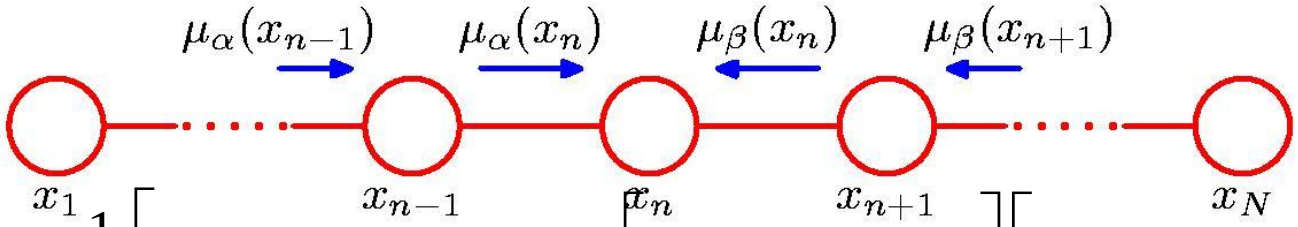
$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} p(\boldsymbol{x})$$

with a cost of $\mathcal{O}(M^N)$ if we have M configurations per variable (exponential in the length of the chain)

- But we can exploit the structure of the graph since the joint distribution is given as:

$$p(x) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Inference on a Chain: Message Passing



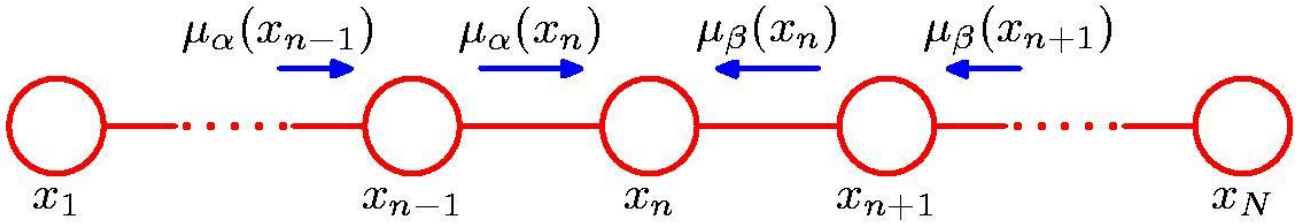
$$p(x_n) = \frac{1}{Z}\underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3)\right]\left[\sum_{x_1} \psi_{1,2}(x_1, x_2)\right]\right]}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_{N-1}} \psi_{N-2,N-1}(x_{N-2}, x_{N-1})\right]\right]\left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right]}_{\mu_\beta(x_n)}$$

$$= \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

- The μ_α term is computed by working forward across the chain. The μ_β term works backwards. The cost is now $\mathcal{O}(NM^2)$ *(N Tables of 2 variables)* – linear in *N*.
- Can be extended to any graph with no loops.

Chapman-Kolmogorov Equations



- $\mu_\alpha(x_n)$ is a message passed forwards along the chain from node x_{n-1} to node x_n.
- Similarly, $\mu_\beta(x_n)$ can be viewed as a message passed backwards along the chain to node x_n from node x_{n+1}.
- These messages comprise each a set of M values.
- $\mu_\alpha(x_n)$ and $\mu_\beta(x_n)$ can be evaluated recursively

$$\mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \left[\sum_{x_{n-2}} \cdots \right] = \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \mu_\alpha(x_{n-1})$$

$$Initialize: \mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2)$$

$$\mu_\beta(x_n) = \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \left[\sum_{x_{n+2}} \cdots \right] = \sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \mu_\beta(x_{n+1})$$

- Papoulis, A. (1984). *Probability, Random Variables, and Stochastic Processes* (Second ed.). McGraw-Hill.

Algorithm: Computing All Marginals

- To compute all local marginals (cost of $\mathcal{O}(2 \times NM^2)$):
  - Compute and store all forward messages, $\mu_\alpha(x_n)$.
  - Compute and store all backward messages, $\mu_\beta(x_n)$.
  - Compute Z once at *any* node x_n ($\sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$, cost $\mathcal{O}(M)$)
  - Compute

$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

    for all variables required.
  - Using these marginals, one can find the x_n that maximizes $p(x_n)$

- The joint distribution for two neighbouring nodes x_{n-1} and x_n on the chain can also be computed as

$$p(x_{n-1}, x_n) = \frac{1}{Z}\mu_\alpha(x_{n-1})\psi_{n-1,n}(x_{n-1}, x_n)\mu_\beta(x_n)$$

Algorithm: Computing Conditionals

- Assume we want to compute $p(x_n|x_N)$ for all n=1,…,N-1.
  - One needs to only change the initial message for the β-recursion:

$$\mu_\beta(x_{N-1}) = \psi_{N-1,N}(x_{N-1}, \bar{x}_N)$$

  - Note that *there is no summation now on x_N.*
  - The α- and β- recursions remain the same apart from the above change and one computes for each n=1,…,N-1:

$$p(x_n \mid \bar{x}_N) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

Max-Product Algorithm on a Chain

- Our goal here is to *find the mode of the joint distribution*:

$$p\left(\mathbf{x}^{\max}\right) = \max_{\mathbf{x}} p\left(\mathbf{x}\right) = \max_{x_1} \cdots \max_{x_N} p\left(x_1, ..., x_N\right)$$

$$= \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} \left[\psi_{1,2}\left(x_1, x_2\right) \cdots \psi_{N-1,N}\left(x_{N-1}, x_N\right)\right]$$

$$= \frac{1}{Z} \max_{x_1} \left[\max_{x_2} \left[\psi_{1,2}\left(x_1, x_2\right) \left[\cdots \max_{x_N} \psi_{N-1,N}\left(x_{N-1}, x_N\right)\right]\right]\right]$$

- Message passing algorithm (linear in the number of nodes) with `sum' replaced by `max'
- This is a generalization of the Viterbi algorithm for HMMs
- Define

$$\phi\left(x_i\right) = \max_{x_1} \cdots \max_{x_{i-1}} \max_{x_{i+1}} \cdots \max_{x_N} p(x_1, x_2, ..., x_N)$$

- Then

$$x_i^{MAP} = \arg\max_{x_i} \phi\left(x_i\right)$$

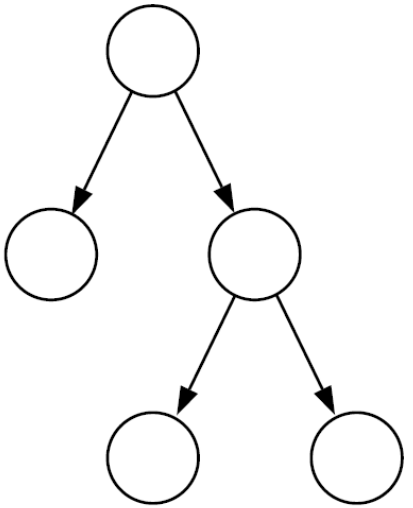
- We will discuss this in detail in a forthcoming lecture.

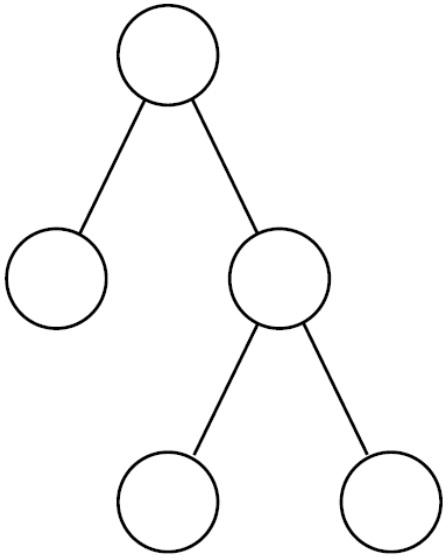
Sum-Product Algorithm

- The elimination algorithm discussed in an earlier lecture works for all graphs, but it must be run separately for each single-node marginal.
- *The sum-product algorithm efficiently computes all marginals in the special case of trees* (already shown earlier for the case of one-dimensional chains).
- *Using factor graphs*, *we can extend the sum-product algorithm to polytrees.*
- One can generalize for arbitrary graphs using the so called junction tree algorithm.

Trees

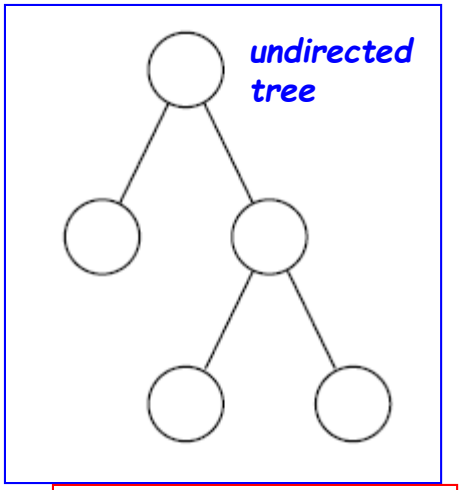
- *Undirected Tree: There is only one path between any pairs of nodes.*
- *Directed Tree: Its moralized graph is an undirected tree.*
- *Directed and undirected trees make the same CI assumptions*
- This is why moralization of a directed tree adds no edges
- However, *undirected trees are not locally normalized*
- The two are essentially equivalent for the purposes of probabilistic inference
- We focus on the undirected case

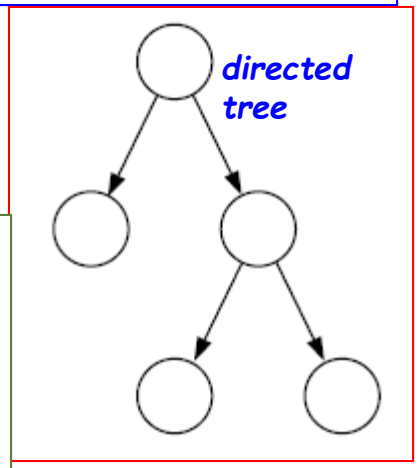


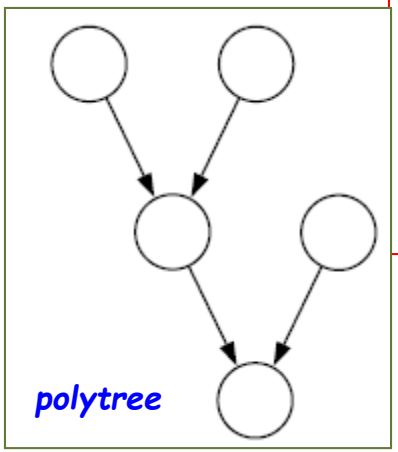


Inference on Trees

- **A tree** is an undirected graph in which there is only and only one path between any pair of nodes.
- ***A directed graph*** *is a tree when its moralized graph is a tree. Directed trees have a single node that has no parent (the root node) and that all other nodes have exactly one parent.*
- ***A polytree*** *is not a directed tree. It has nodes with multiple parents and its moralized graph has loops.*
- Any undirected tree can be converted to a directed tree – choose a root node and orient all edges pointing away from the root. The two have the same conditional independence relations.

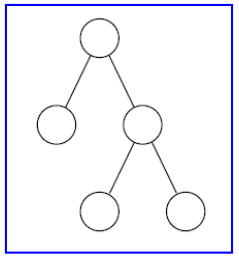






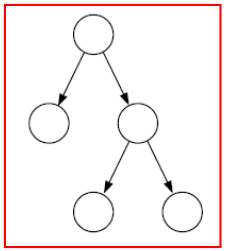
Parametrization

- For undirected trees, the cliques are pairs of nodes and single nodes.



$$p(x) = \frac{1}{Z}\prod_{i\in\mathcal{V}}\psi(x_i)\prod_{(i,j)\in\mathcal{E}}\psi(x_i,x_j),\ \textit{for a tree}\ \mathcal{T}(\mathcal{V},\mathcal{E})$$

- For directed trees:



$$p(x) = p(x_r)\prod_{(i,j)\in\mathcal{E}} p(x_j \mid x_i),\ where\ (i,j) = directed\ edge, \{i\} = \pi_j$$

- We can represent the directed tree as undirected tree (r=root node):

$$\psi(x_r) = p(x_r), \psi(x_i) = 1\ \forall\ i \neq r, Z = 1$$
$$\psi(x_i, x_j) = p(x_j \mid x_i), \{i\} = \pi_j$$

Evidence Potentials and Conditioning

- Recall the definition of evidence potentials

$$\psi_i^E(x_i) \triangleq \begin{cases} \psi_i\ (x_i)\delta(x_i, \bar{x}_i), & i \in E \\ \psi_i\ (x_i), & i \notin E \end{cases}$$

- Using this, we can write conditional distributions as follows:

$$p(x \mid \bar{x}_E) = \frac{1}{Z^E} \prod_{i \in \mathcal{V}} \psi^E(x_i) \prod_{(i,j) \in \mathcal{E}} \psi(x_i, x_j), \ \textit{for a tree } \mathcal{T}(\mathcal{V}, \mathcal{E})$$

where the normalization factor is

$$Z^E = \sum_x \left(\prod_{i \in \mathcal{V}} \psi^E(x_i) \prod_{(i,j) \in \mathcal{E}} \psi(x_i, x_j) \right)$$

- Note that *with this parametrization, the conditional and unconditional cases are practically the same (but note Z^E is not 1 in the conditional case).*

Incorporating Evidence

- Let us consider another example of accounting for the evidence $x_1=1$ for a distribution having the form:

$$P(x_1, x_2, \ldots x_n; \theta) = \frac{1}{Z(\theta)} \prod_{(i,j)\in E} \psi_{i,j}(x_i, x_j)$$

- An easy way to introduce this evidence is by modifying the clique potentials for all edges $(1, j) \in E$ as follows:

$$\psi_{1,j}(x_1, x_j) = 0 \;\forall\; x_1 \neq 1$$

- We keep all other definitions of $\psi_{i,j}(x_i, x_j)$ with no changes.

- Our Bayesian network is then of the form

$$P(x_2, \ldots x_n; X_1 = 1, \theta) = \frac{1}{Z(\theta)} \prod_{(i,j)\in E} \psi_{i,j}(x_i, x_j)$$

Elimination Algorithm for Trees

- We specialize the elimination algorithm to trees.
  - Treat f as the root and view the tree as a directed tree by directing all edges of the tree to point away from f.
  - Consider an elimination ordering in which a node is eliminated only after all of its children in the directed version of the tree are eliminated.
  - This elimination proceeds inwards from the leaves and *generates elimination cliques of size at most two* (treewidth=1).

Pruning Leaves – Directed Case

- *In the directed case, any unobserved leaf (or subtree) can be pruned from the tree*
  - The reason is that the corresponding summation must reduce to unity

$$\sum_{x_j} p(x_j \mid x_i) = 1$$

- In the unconditional case, we can arrange things so all sums are of this form (elimination ordering that begins with the leaves and proceeds to the root) and Z=1 (Something similar happens with undirected graphs even though Z is affected).

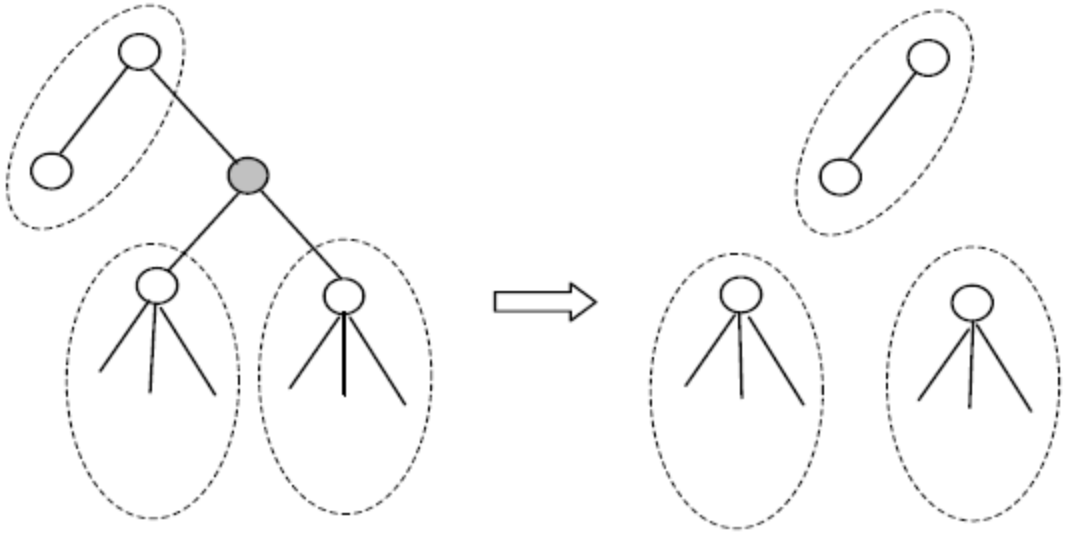
- When we condition, however, the resulting product of potentials is unnormalized (and Z^E not equal to 1). This brings the directed case closer to the undirected case.

- *We can prune any subtree that contains only variables that are not conditioned on by eliminating backwards.*

Pruning Leaves

- Thus, *we can assume that all leaves are observed (evidence nodes) and all of the sums have to be computed explicitly.*
- *There is no difference in this case between directed and undirected cases.*
- It may also be reasonable to *assume all nonleaves are unobserved, because observed nonleaves would split the tree; thus E would equal the set of leaves*

- Conditioning on a set of nodes creates conditional independencies through graph separation that will subdivide the graph into smaller trees whose marginal probabilities are solved independently of each other.
- To work with only one tree, we treat the evidence by modifying the definition of the self potentials.

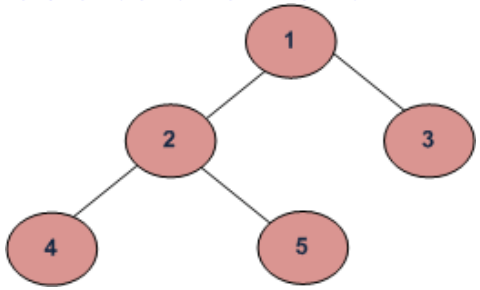


Message Passing

- Consider the case of computing the marginal probability of an arbitrary ancestral node in an undirected tree by elimination
- *An optimal elimination ordering proceeds from leaves to root (postorder) in a tree rooted at the query node*
- Under such an ordering, *the intermediate functions generated during the elimination process will pass like messages from descendants to ancestors*
- Each message summarizes evidence in the subtree beneath the node
- *Messages can be reused from one node to the next*

```
Algorithm Postorder(tree)
1.  Traverse the left subtree, i.e., call Postorder(left-subtree)
2.  Traverse the right subtree, i.e., call Postorder(right-subtree)
3.  Visit the root.
```

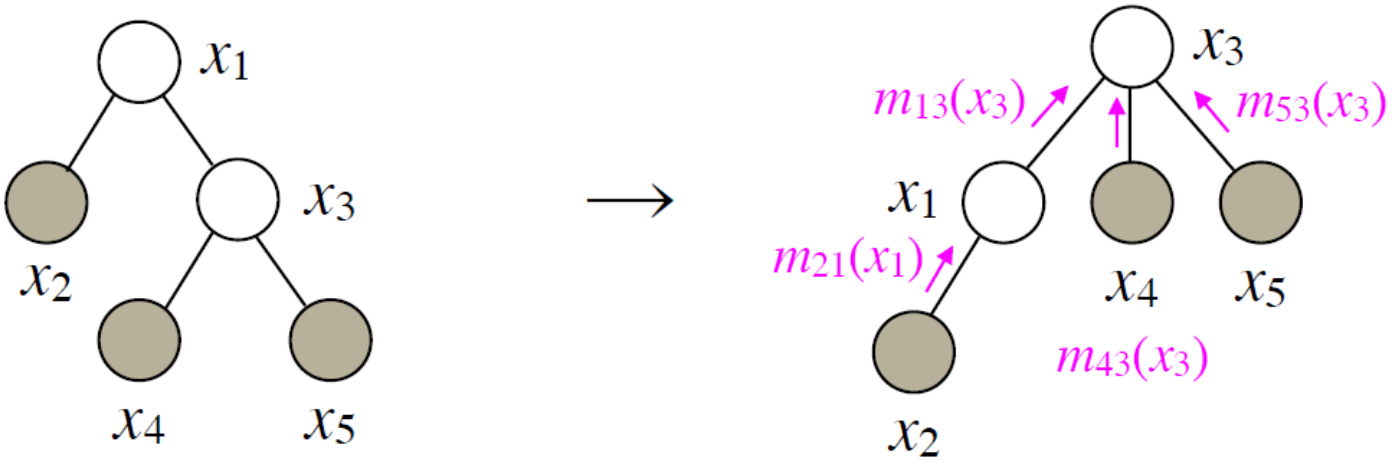
Postorder (Left, Right, Root) : 4 5 2 3 1



Elimination Algorithm

- Recall the general elimination algorithm
  - Choose an elimination I in which the query node f is the final node
  - Place all potentials on an active list
  - Eliminate a node i by removing all potentials referencing the node from the active list, taking the product, summing over x_i, and placing the resulting intermediate factor on the active list.
- Consider *depth-first traversal of the tree*. *A node is eliminated only after all of its children in the directed version of the tree are eliminated.* This proceeds inwards from the leaves and generates cliques of size 2 (tree-width=1). Let us see an example.

Elimination and Message Passing



$$p\ x_3 \mid \bar{x}_2, \bar{x}_4, \bar{x}_5 = \frac{1}{Z^E} \prod_{i \in \mathcal{V}} \psi^E(x_i) \prod_{(i,j) \in \mathcal{E}} \psi(x_i, x_j)$$

$$= \frac{1}{Z^E} \sum_{x_1} \psi^E\ x_3\ \psi^E\ x_1\ \psi(x_1, x_3) \underbrace{\psi^E\ x_2\ \psi(x_2, x_1)}_{m_{21}\ x_1} \underbrace{\psi^E\ x_4\ \psi(x_4, x_3)}_{m_{43}\ x_3} \underbrace{\psi^E\ x_5\ \psi(x_5, x_3)}_{m_{53}\ x_3}$$

$$= \frac{1}{Z^E} \psi^E\ x_3 \underbrace{\left(\sum_{x_1} \psi^E\ x_1\ \psi(x_1, x_3) m_{21}\ x_1 \right)}_{m_{13}\ x_3} m_{43}\ x_3\ m_{53}\ x_3$$

$$= \frac{1}{Z^E} \psi^E\ x_3\ m_{13}\ x_3\ m_{43}\ x_3\ m_{53}\ x_3$$

$$= \frac{\psi^E\ x_3\ m_{13}\ x_3\ m_{43}\ x_3\ m_{53}\ x_3}{\sum_{x_3'} \psi^E\ x_3'\ m_{13}\ x_3'\ m_{43}\ x_3'\ m_{53}\ x_3'}$$

Message Passing in Trees

- Consider the node i closer to the root than node j.
- What is the intermediate factor when j is eliminated?

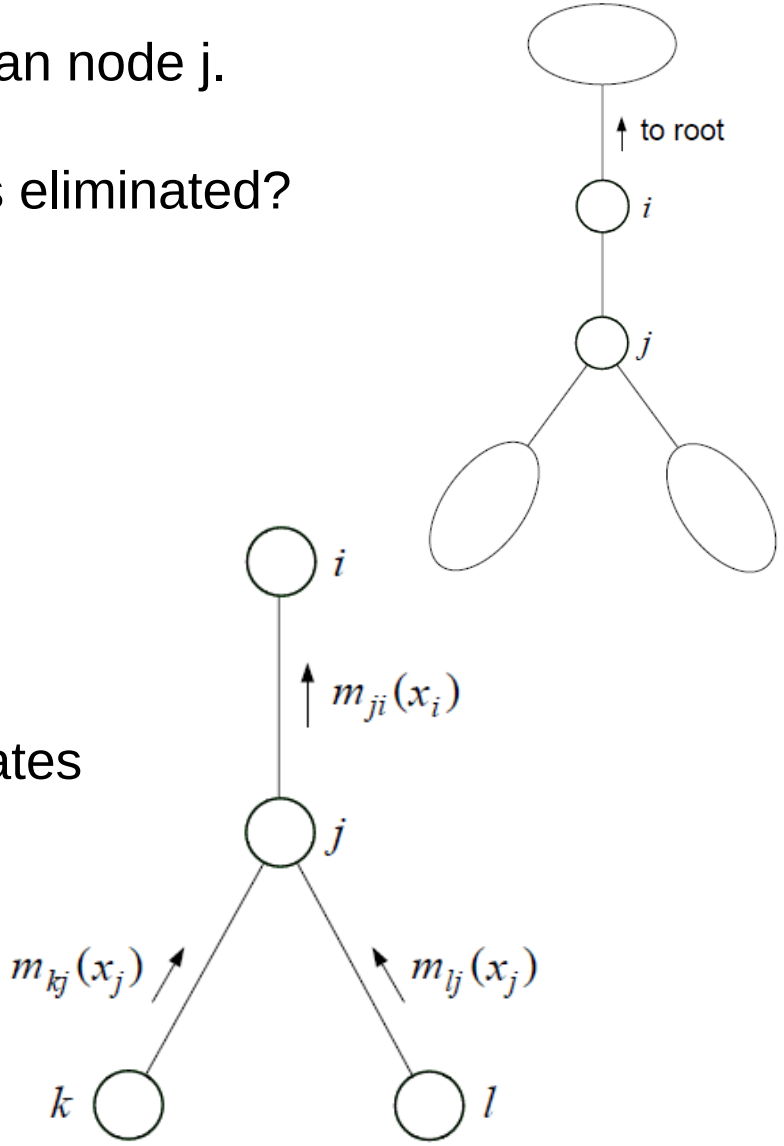
$m_{ji}(x_i) =$ message that j sends to i

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

Inference results in a coupled set of eqs for $m_{ji}(x_i)$ (Complexity $\mathcal{O}(m^2)$, m=states per node)

- For the final node f, we have:

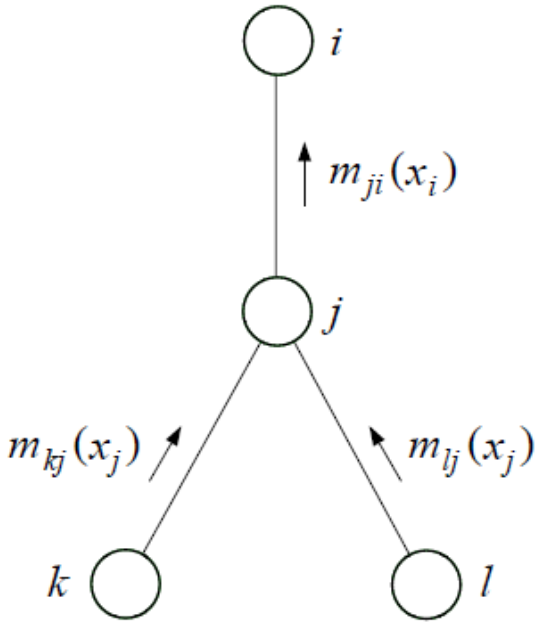
$$p(x_f \mid \bar{x}_E) = \frac{\psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}(x_f)}{\sum_{x'_f} \psi^E(x'_f) \prod_{e \in \mathcal{N}(f)} m_{ef}(x'_f)}$$



The Sum Product Algorithm

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

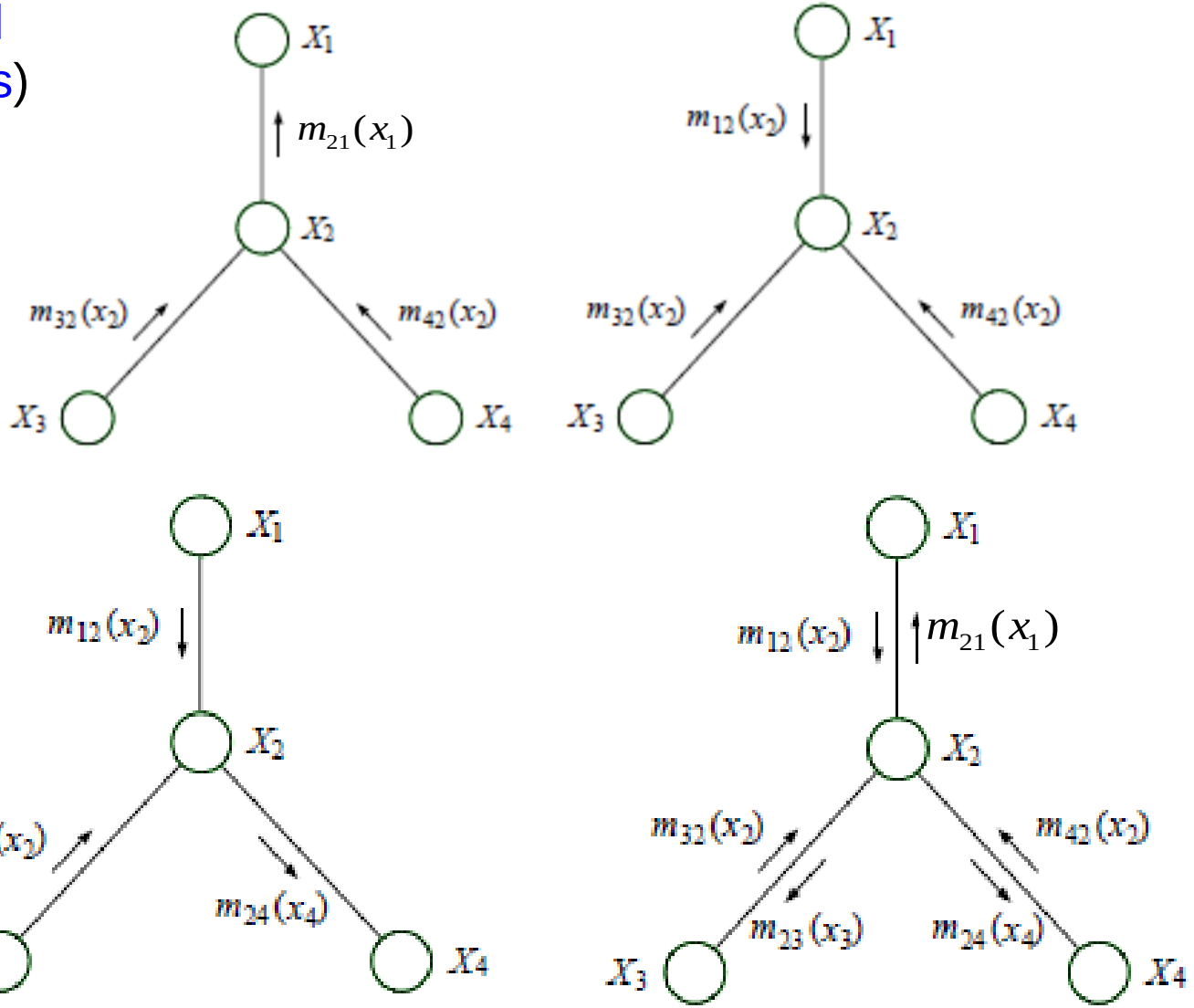
$$p(x_f \mid \bar{x}_E) \propto \psi^E(x_f) \prod_{e \in \mathcal{N}(f)} m_{ef}(x_f)$$



- We can obtain all marginals by simply doubling the work needed to compute a single marginal.
- After having passed messages from the leaves to an arbitrary root, we can pass messages from the root to back to the leaves
- We can then *use similar Eqs to the ones above to find the marginal at every node*.
- The complete set of messages scales linearly with the size of the tree.

The Sum Product Algorithm on Trees

- Below are the messages for computing all marginal (reuse the computed messages)



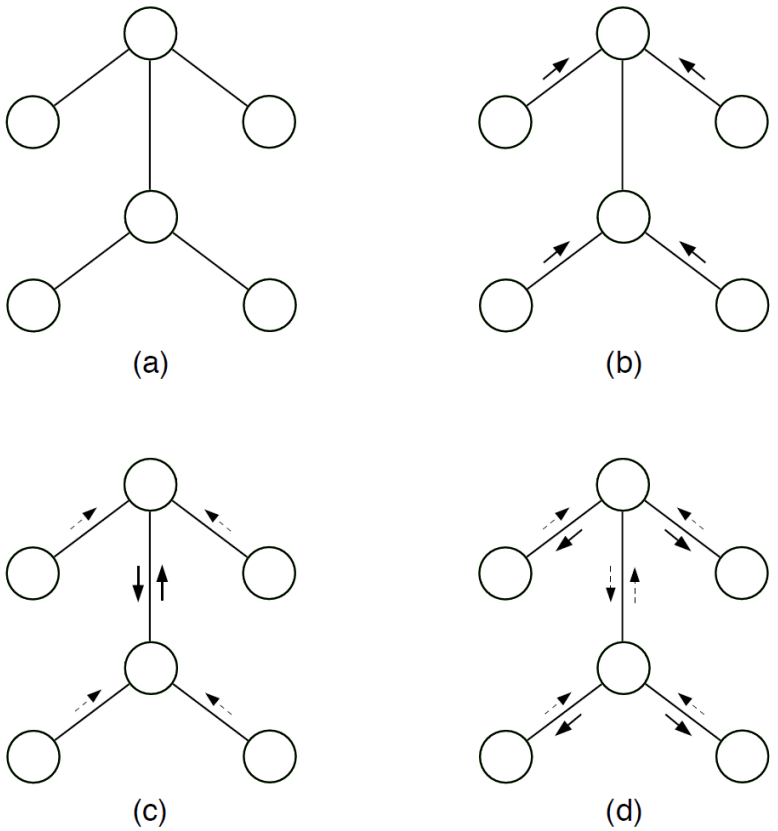
Message-Passing Protocol

- All that is *needed at each query node is the complete set of incoming messages*
- However, there are dependencies to consider: *a message cannot be sent from j to i until messages have been received from all **other** neighbors of j*
- Think of this as a parallel algorithm: each node has a processor that polls its neighbors, sends each message when all others received
- The messages will flow in from the leaves.

Synchronous Parallel Protocol

- ***Protocol***: *node j sends a message to neighbor i when (and only when) it has received messages from all other neighbors*

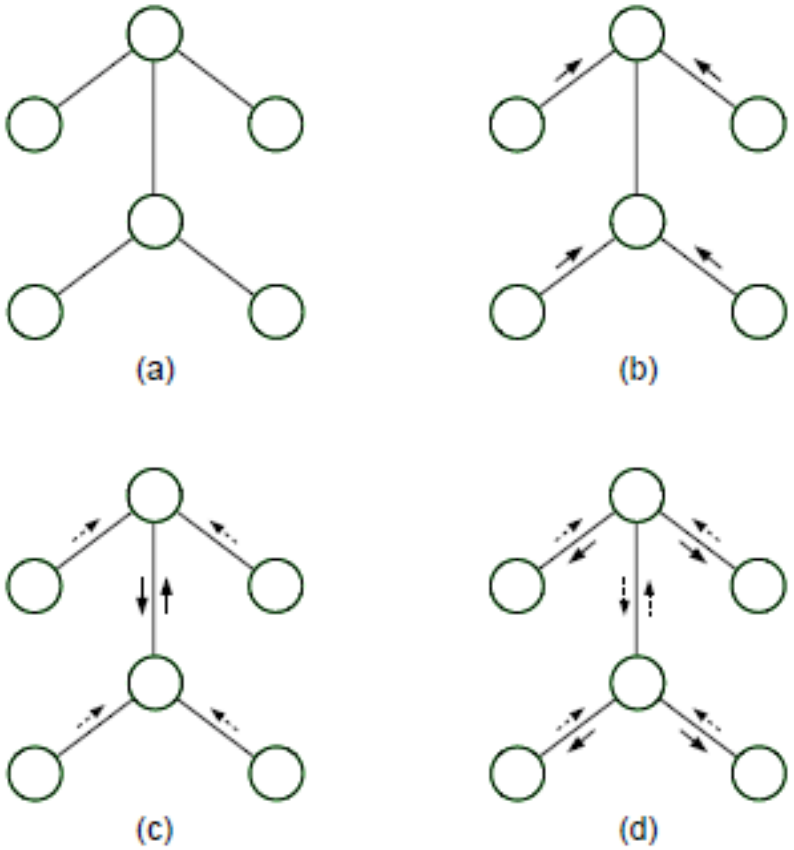
- All incoming messages received by all nodes in $\mathcal{O}(2|\mathcal{E}|)$ steps



- ✓ Solid arrows: messages passed at a given time
- ✓ Dashed arrows: messages passed at earlier times

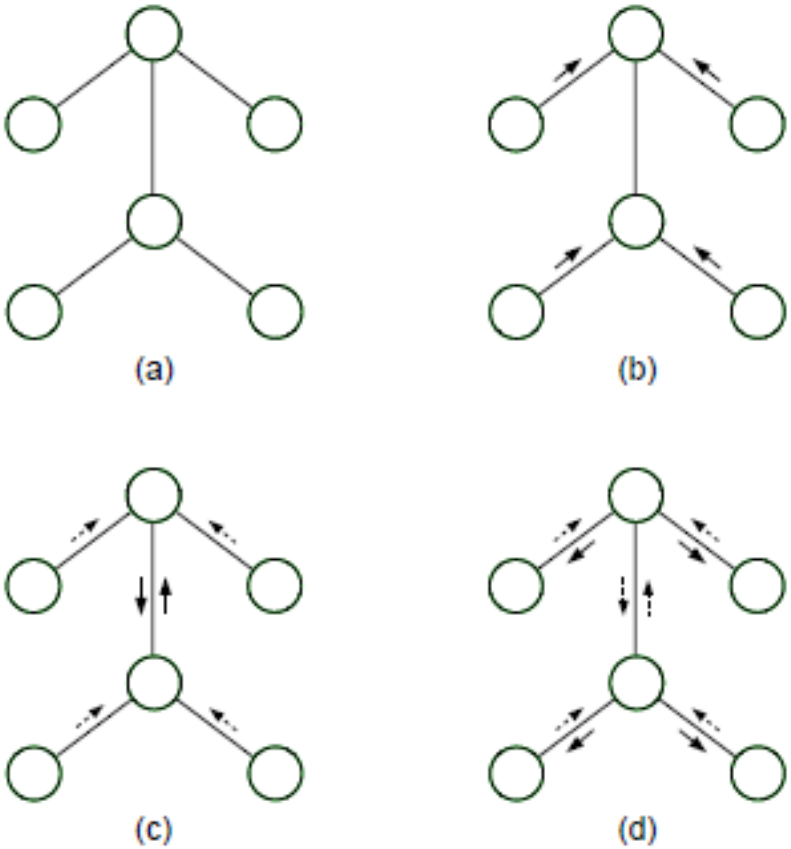
The Sum Product Algorithm on Trees

- The solid lines are the messages passed at a given time step and the dotted lines are the messages passed at the earlier step.



The Sum Product Algorithm on Trees

- Messages start to flow from the leaves. When the algorithm terminates, there are *two messages for each edge - one* in each direction. *We need to be sure that the algorithm never blocks.*



The Sum Product Algorithm on Trees

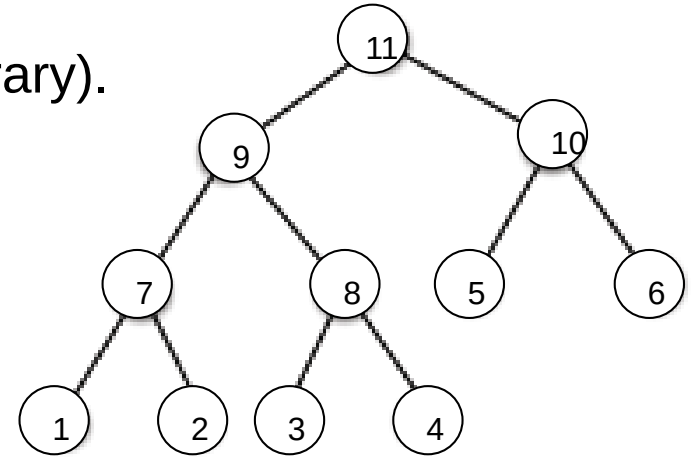
- Let us consider the MRF model shown. We want to compute the partition function as well as marginal probabilities.

$$P(x_1, x_2, \ldots x_n; \theta) = \frac{1}{Z(\theta)} e^{\sum_{(i,j)\in E} \theta_{ij}(x_i, x_j)} = \frac{1}{Z(\theta)} \prod_{(i,j)\in E} e^{\theta_{ij}(x_i, x_j)} = \frac{1}{Z(\theta)} \prod_{(i,j)\in E} \psi_{i,j}(x_i, x_j)$$

$$Z(\theta) = \sum_{x_1, x_2, \ldots, x_n} \prod_{(i,j)\in E} e^{\theta_{ij}(x_i, x_j)}$$

$$P(X_i = x; \theta) = \frac{1}{Z(\theta)} \sum_{x_1, x_2, \ldots, x_n} \delta(x_i, x) \prod_{(i,j)\in E} \psi_{i,j}(x_i, x_j)$$

- We *pick node 11 as the root node* (arbitrary).



Bottom Up Dynamic Programming

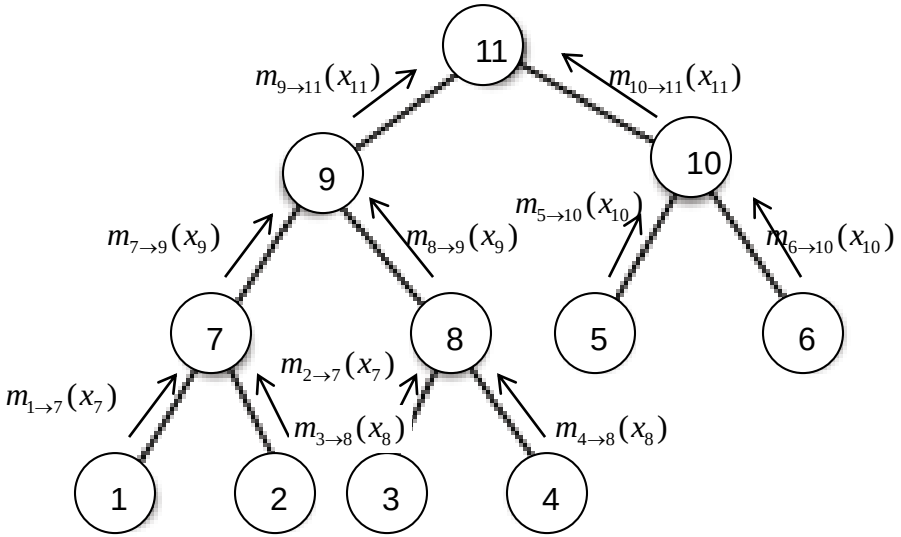
- We first send messages up through the tree.

$$m_{i\to j}(x_j)=\sum_{x_i}\psi_{i,j}(x_i,x_j)\prod_{k\in N(i),k\neq j}m_{k\to i}(x_i),\ where\ N(i)=\{j:(i,j)\in E\}$$

Note if N(i)={j}, then the message is simply

$$m_{i\to j}(x_j)=\sum_{x_i}\psi_{i,j}(x_i,x_j)$$

- The particular calculations of these messages are shown next.



Bottom Up Dynamic Programming

$$m_{1\to 7}(x_7)=\sum_{x_1}\psi_{1,7}(x_1,x_7) \qquad m_{3\to 8}(x_8)=\sum_{x_3}\psi_{3,8}(x_3,x_8) \qquad m_{5\to 10}(x_{10})=\sum_{x_5}\psi_{5,10}(x_5,x_{10})$$

$$m_{2\to 7}(x_7)=\sum_{x_2}\psi_{2,7}(x_2,x_7) \qquad m_{4\to 8}(x_8)=\sum_{x_4}\psi_{4,8}(x_4,x_8) \qquad m_{6\to 10}(x_{10})=\sum_{x_6}\psi_{6,10}(x_6,x_{10})$$

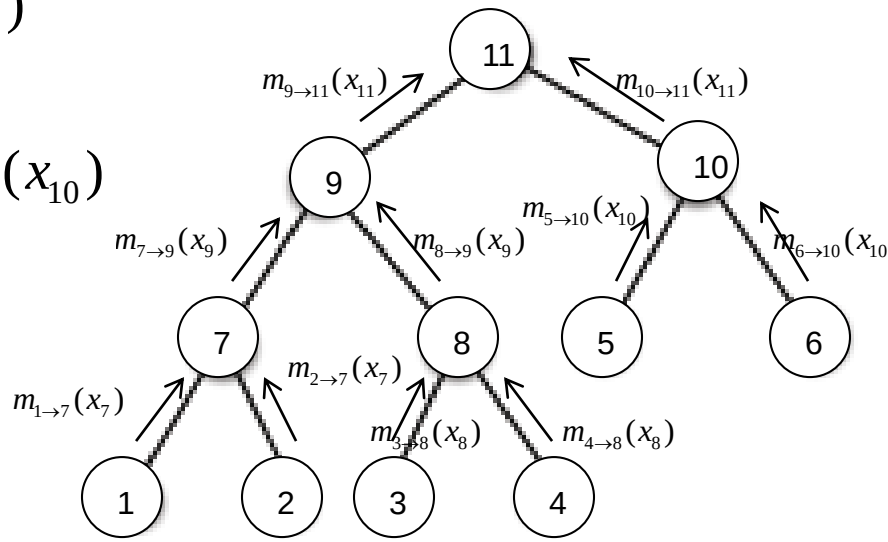
$$m_{7\to 9}(x_9)=\sum_{x_7}\psi_{7,9}(x_7,x_9)m_{1\to 7}(x_7)m_{2\to 7}(x_7)$$

$$m_{8\to 9}(x_9)=\sum_{x_8}\psi_{8,9}(x_8,x_9)m_{3\to 8}(x_8)m_{4\to 8}(x_8)$$

$$m_{9\to 11}(x_{11})=\sum_{x_9}\psi_{9,11}(x_9,x_{11})m_{7\to 9}(x_9)m_{8\to 9}(x_9)$$

$$m_{10\to 11}(x_{11})=\sum_{x_{10}}\psi_{10,11}(x_{10},x_{11})m_{5\to 10}(x_{10})m_{6\to 10}(x_{10})$$

$$Z(\theta)=\sum_{x_{11}} m_{9\to 11}(x_{11})m_{10\to 11}(x_{11})$$

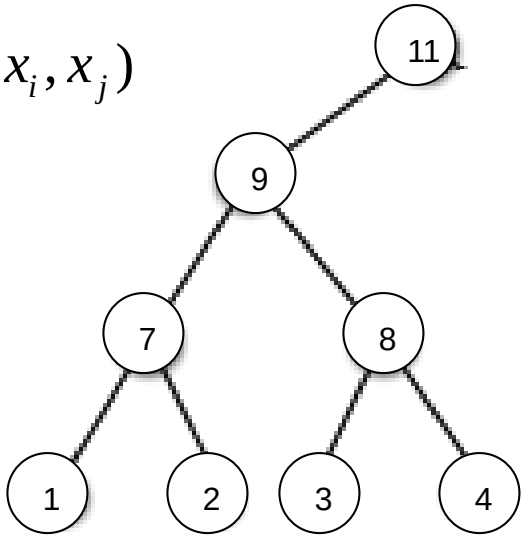


Meaning of Messages

- Let us consider the message $m_{9\to 11}(x_{11})$

- We define the subtree T(9,11) – it contains the edge 9,11 and the *subtree rooted at node 9 that is the component when the edge (9,11) is removed from the graph.*

$$m_{9\to 11}(x_{11}) = \sum_{x_1,x_2,x_3,x_4,x_7,x_8,x_9} \prod_{(i,j)\in T(9,11)} \psi_{i,j}(x_i, x_j)$$



Downward Messages

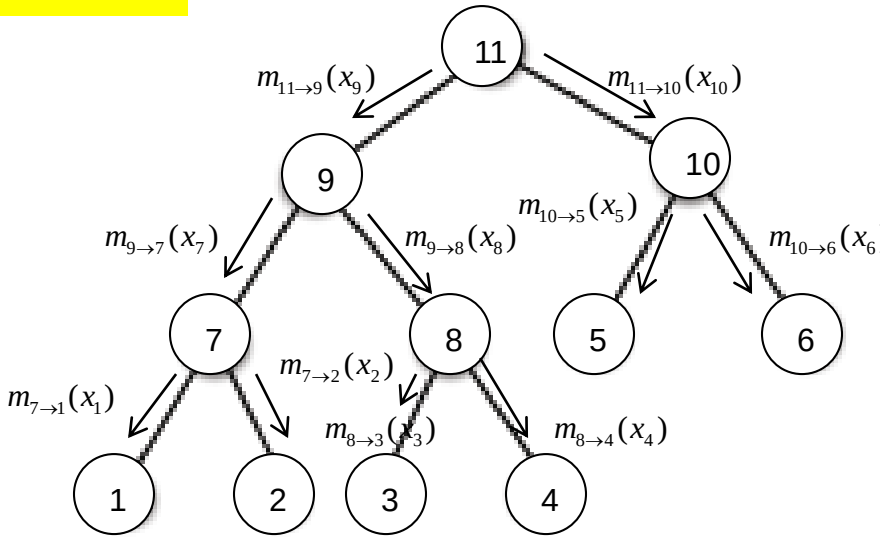
- We now compute messages send from the root to the leafs. The same equations are applied as before.

$$m_{i\to j}(x_j)=\sum_{x_i}\psi_{i,j}(x_i,x_j)\prod_{k\in N(i),k\neq j}m_{k\to i}(x_i),\ where\ N(i)=\{j:(i,j)\in E\}$$

Note if N(i)={j}, then the message is simply

$$m_{i\to j}(x_j)=\sum_{x_i}\psi_{i,j}(x_i,x_j)$$

- The particular calculations of these messages are shown next.



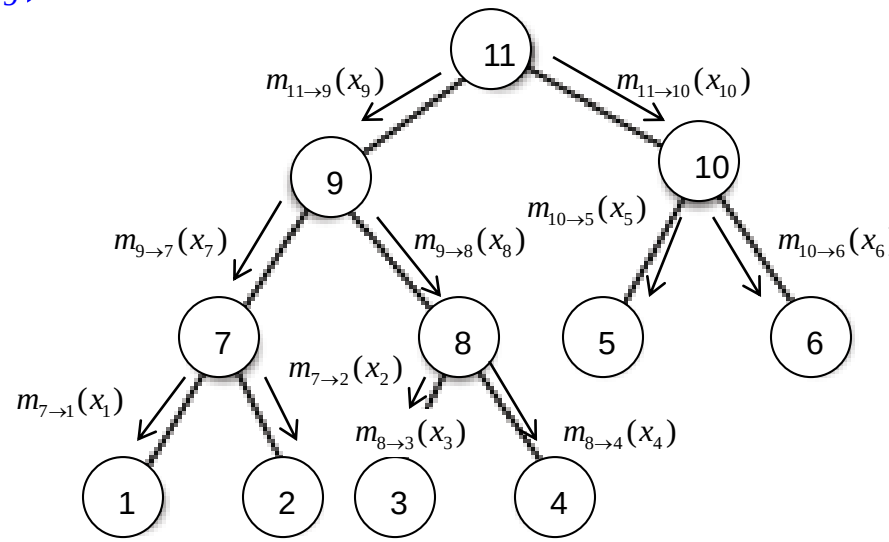
Downward Messages

$$m_{11\to 9}(x_9) = \sum_{x_{11}} \psi_{11,9}(x_{11}, x_9) m_{10\to 11}(x_{11})$$

$$m_{11\to 10}(x_{10}) = \sum_{x_{11}} \psi_{11,10}(x_{11}, x_{10}) m_{9\to 11}(x_{11})$$

$$m_{9\to 7}(x_7) = \sum_{x_9} \psi_{9,7}(x_9, x_7) m_{11\to 9}(x_9) m_{8\to 9}(x_9)$$

$$m_{9\to 8}(x_8) = \sum_{x_9} \psi_{9,8}(x_9, x_8) m_{11\to 9}(x_9) m_{7\to 9}(x_9)$$



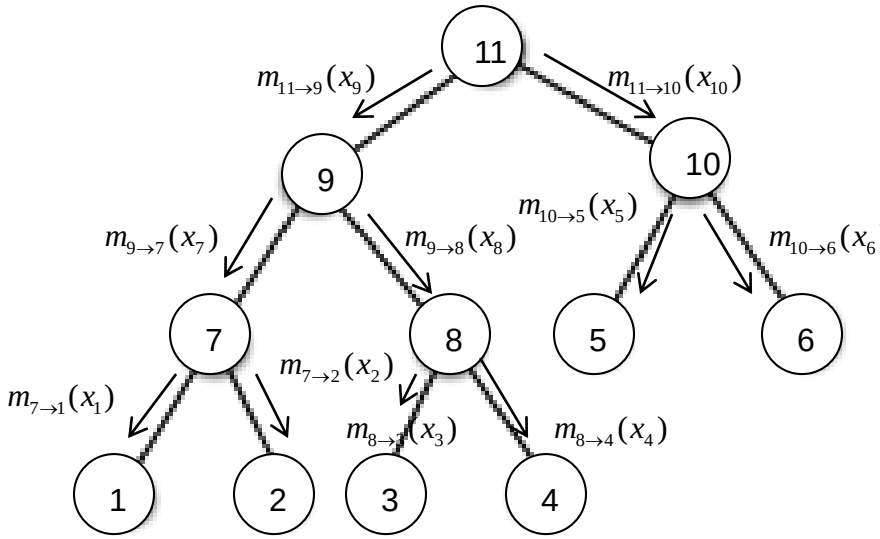
Computing Marginals

- Once all messages are computed, one can easily compute all the marginals.

$$P(X_i = x;\theta) = \frac{1}{Z(\theta)} \prod_{j \in N(i)} m_{j \to i}(x), \text{ where } N(i) = \{ j : (i,j) \in E \}$$

- For example:

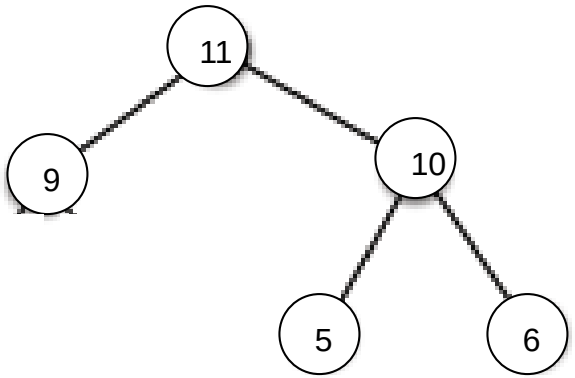
$$P(X_9 = x;\theta) = \frac{1}{Z(\theta)} m_{7 \to 9}(x) m_{8 \to 9}(x) m_{11 \to 9}(x)$$



Meaning of the Downward Messages

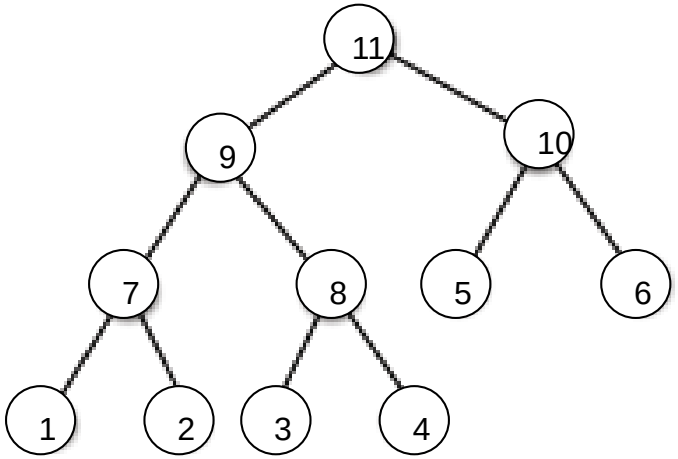
- Consider a typical message $m_{11\to 9}(x_9)$
- We consider the subtree T(11,9) that contains the edge 11,9 and the *subtree rooted at node 11 that is the component when the edge (11,9) is removed from the graph.*

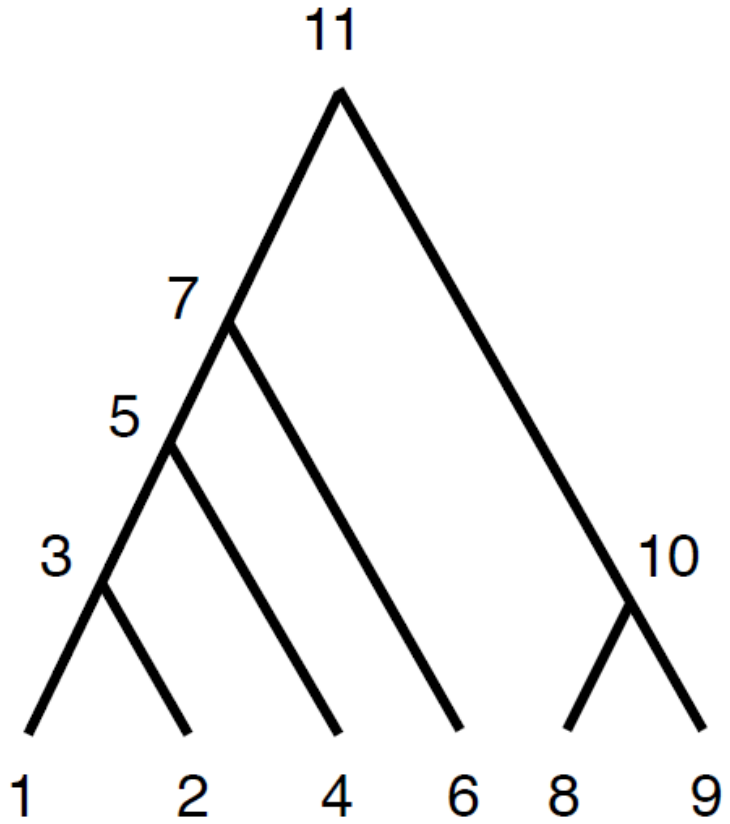
$$m_{11\to 9}(x_9) = \sum_{x_5, x_6, x_{10}, x_{11}} \prod_{(i,j)\in T(11,9)} \psi_{i,j}(x_i, x_j)$$



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)

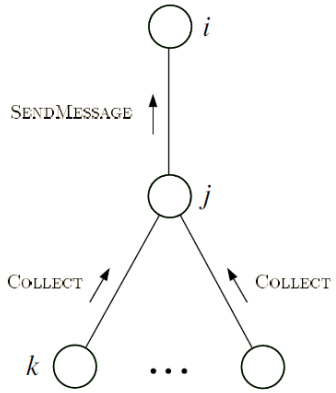
Evidence(E)

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)

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

Collect(i,j)

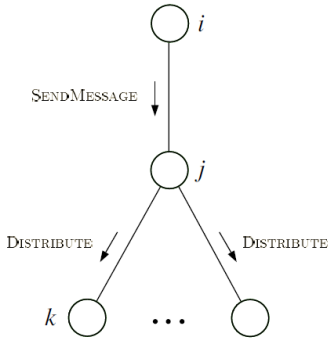
for $k \in \mathcal{N}(j)\setminus i$
    Collect(j,k)
SendMessage(j,i)

ComputeMarginal(i)

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

Distribute(i,j)

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



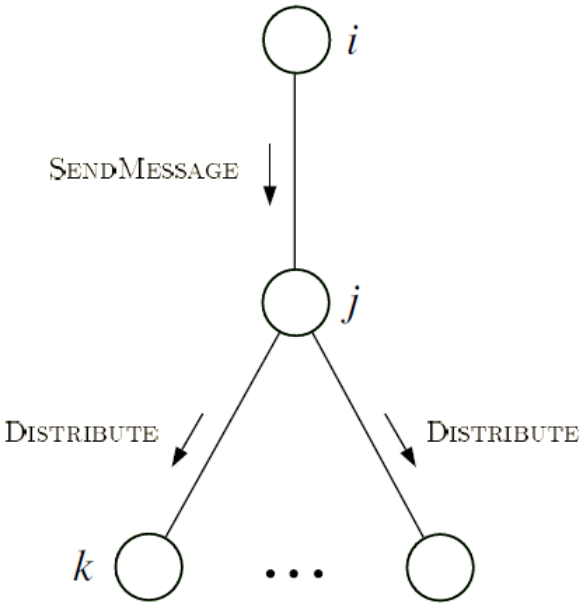
Sum Product Algorithm for a Tree

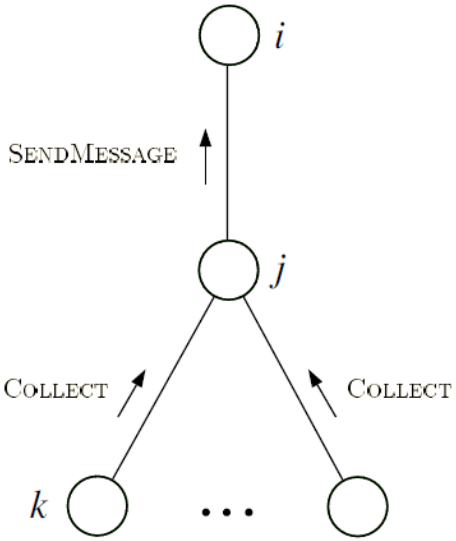
```
Sum-Product(𝒯, E)
    Evidence(E)
    f=ChooseRoot(𝒱)
    for e ∈ 𝒩(f)
        Collect(f,e)
    for e ∈ 𝒩(f)
        Distribute(f,e)
    for i ∈ 𝒱
        ComputeMarginal(i)
```

Choose any root (unspecified here)

Messages flow from the leaves to the root

Messages flow outward from the root to the leaves

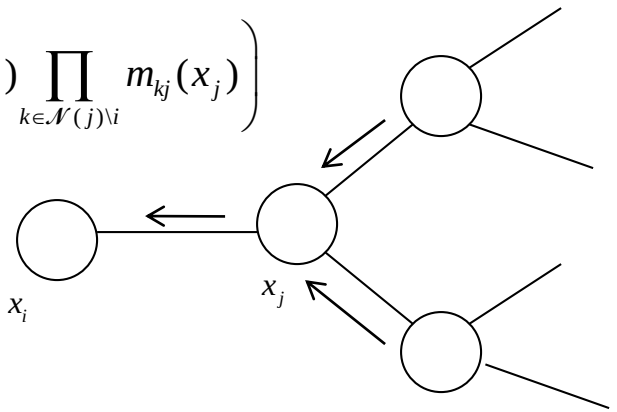




Belief Propagation: Sum-Product Algorithm

- The message passing algorithm is thus 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

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



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

- *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.

Notation for Factor Graphs

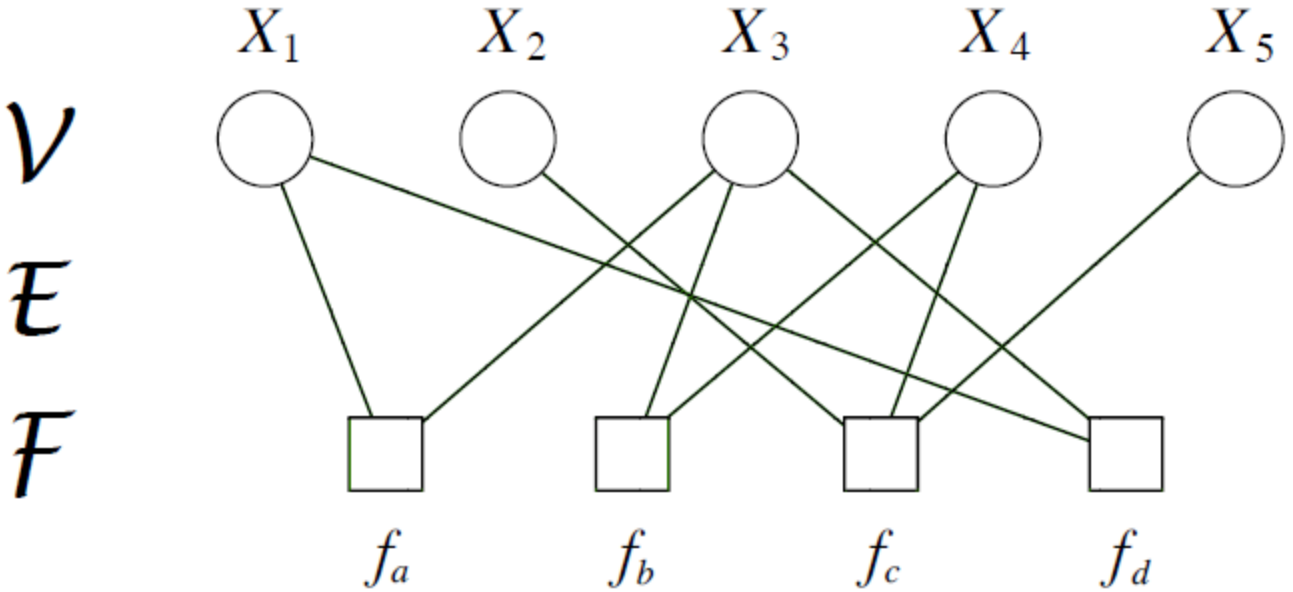
- Given variables $\{x_1, ..., x_n\}$, let $\mathcal{C}$ be a (multi)set of subsets of the indices $\{1, ..., n\}$
- For example, $\mathcal{C} = \{\{1,3\}, \{3,4\}, \{2,4,5\}, \{1,3\}\}$ for $\{x_1, ..., x_5\}$
- Let there be a function $f_s(x_{C_s})$ associated with each $C_s \in \mathcal{C}$. It is called a factor
- Let the multivariate function f be defined:

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

- *This function need not be a probability distribution*, but we will assume it is

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:

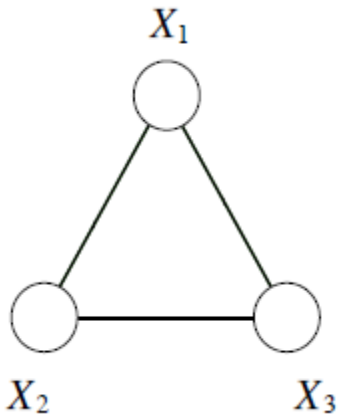


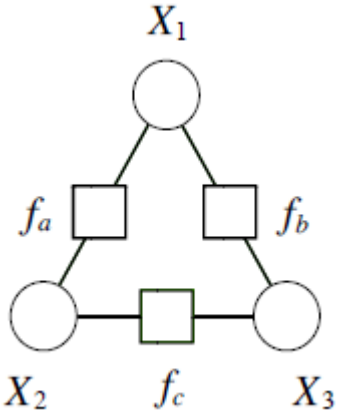
$$f(x_1, x_2, x_3, x_4, x_5) = f_a(x_1, x_3)\, f_b(x_3, x_4)\, f_c(x_2, x_4, x_5)\, f_d(x_1, x_3)$$

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

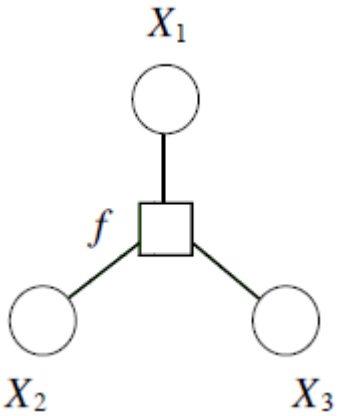
Converting to Factor Graphs

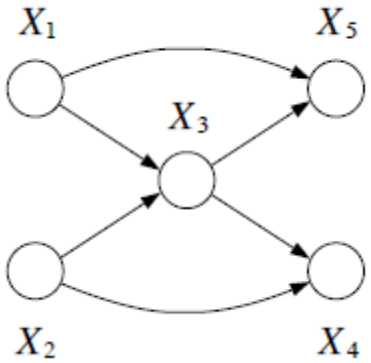
$$\psi(x_1, x_2, x_3) = f_a(x_1, x_2)\, f_b(x_1, x_3)\, f_c(x_2, x_3)$$

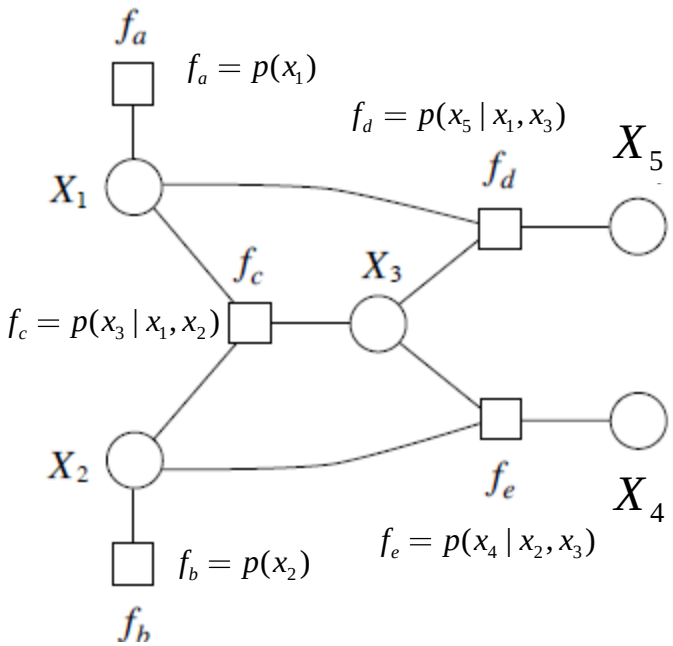




$\psi(x_1, x_2, x_3)$

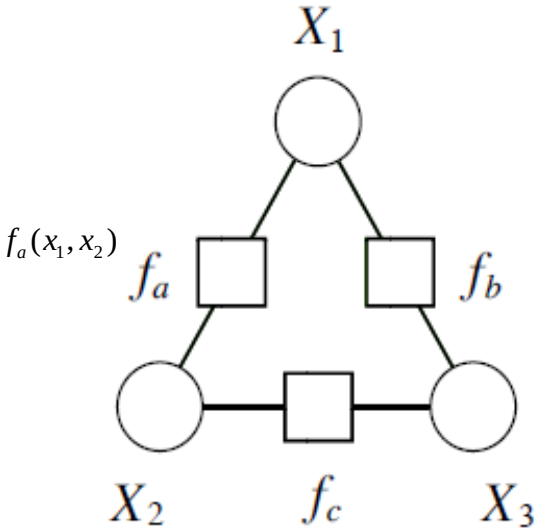






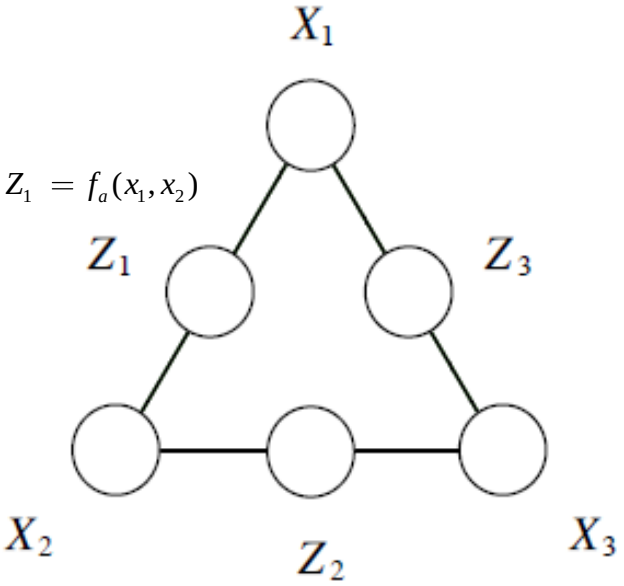
Converting From Factor Graphs

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



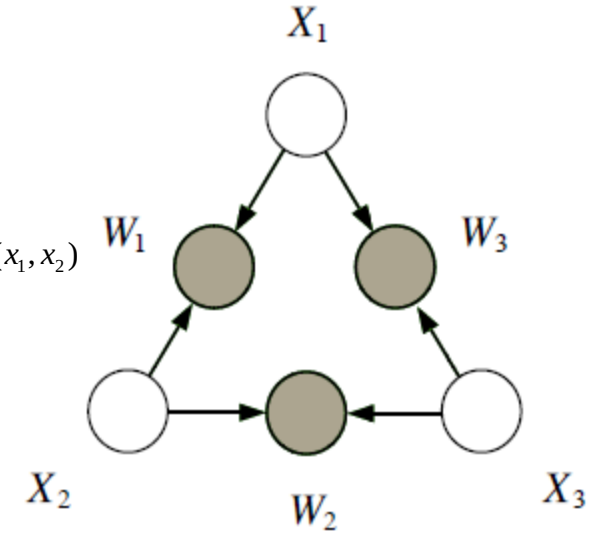
Z_1 an indicator variable taking 4 values for each combination of the values of X_1 and X_2 (assumed binary)

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



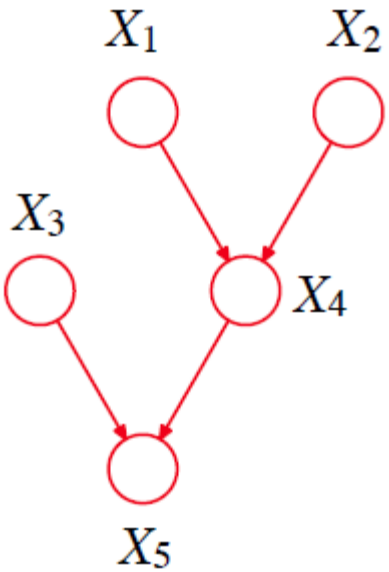
$$p\ W_1 = 1 |\ x_1, x_2\ = f_a(x_1, x_2)$$

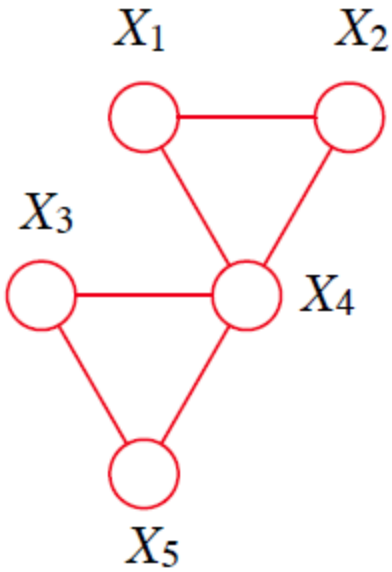
W_1 binary variable always set to 1

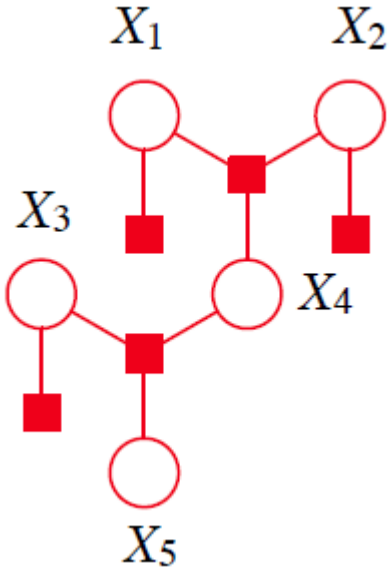


Factor Trees and Polytrees

- *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







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) to obtain an efficient, exact inference algorithm for finding marginals;
  2) in situations where several marginals are required, to allow computations to be shared efficiently.
- Key idea: *Distributive Law*

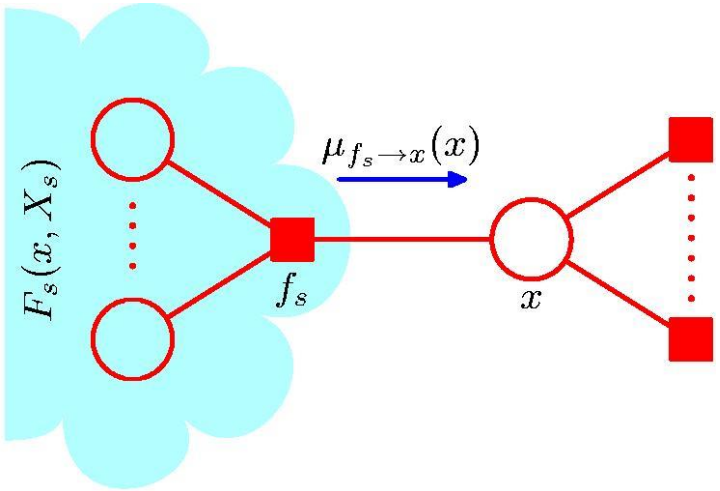
$$ab + ac = a(b + c)$$

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

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

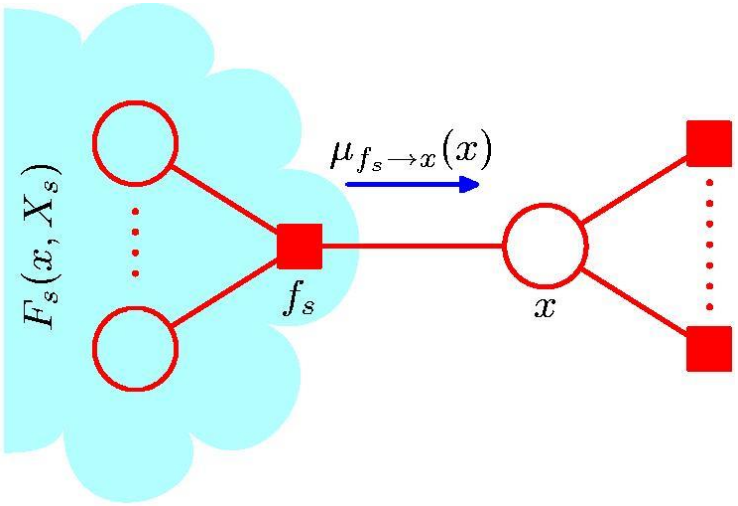
where **x** \ x denotes the set of variables in **x** with variable x omitted.



The joint distribution:

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

The Sum-Product Algorithm

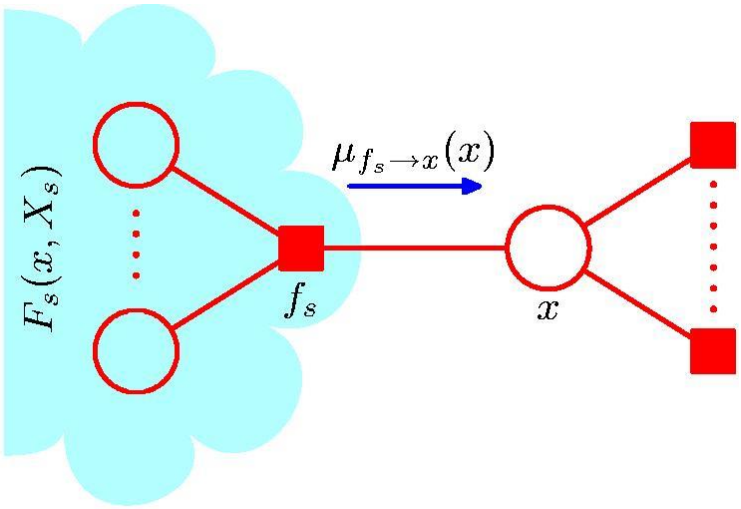


The joint distribution:

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

- 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
- $F_s(x, X_s)$ 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

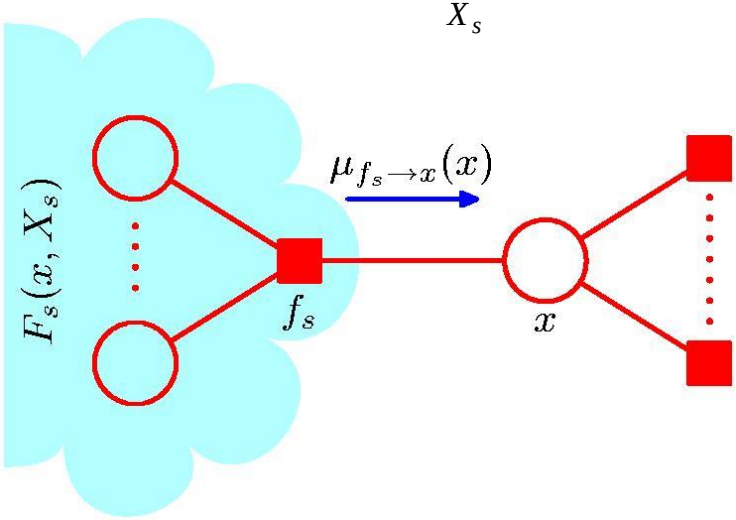
$$\mu_{f_s \to x}(x) = \sum_{X_s} F_s(x, X_s)$$

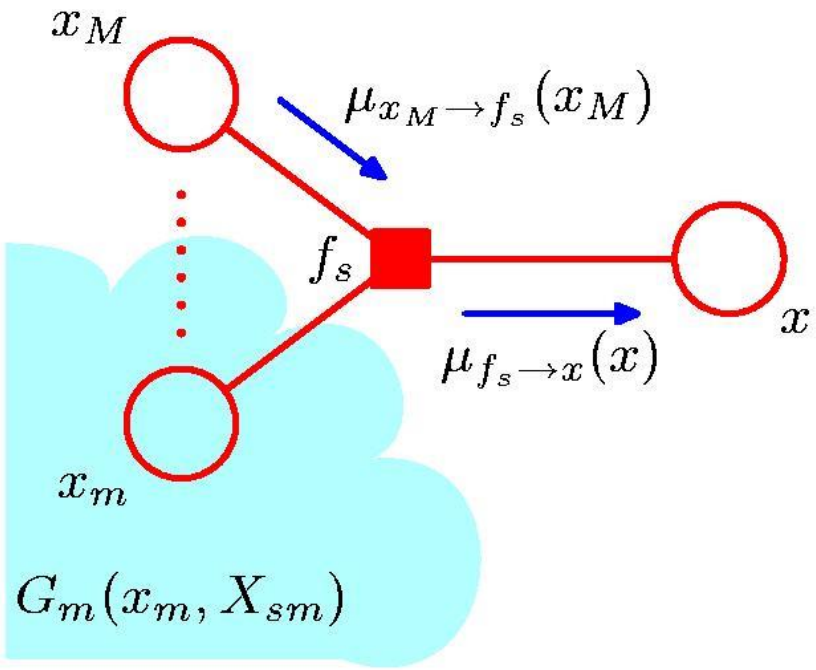
which can be viewed as message from the factor node f_s to the variable node x.

- 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

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





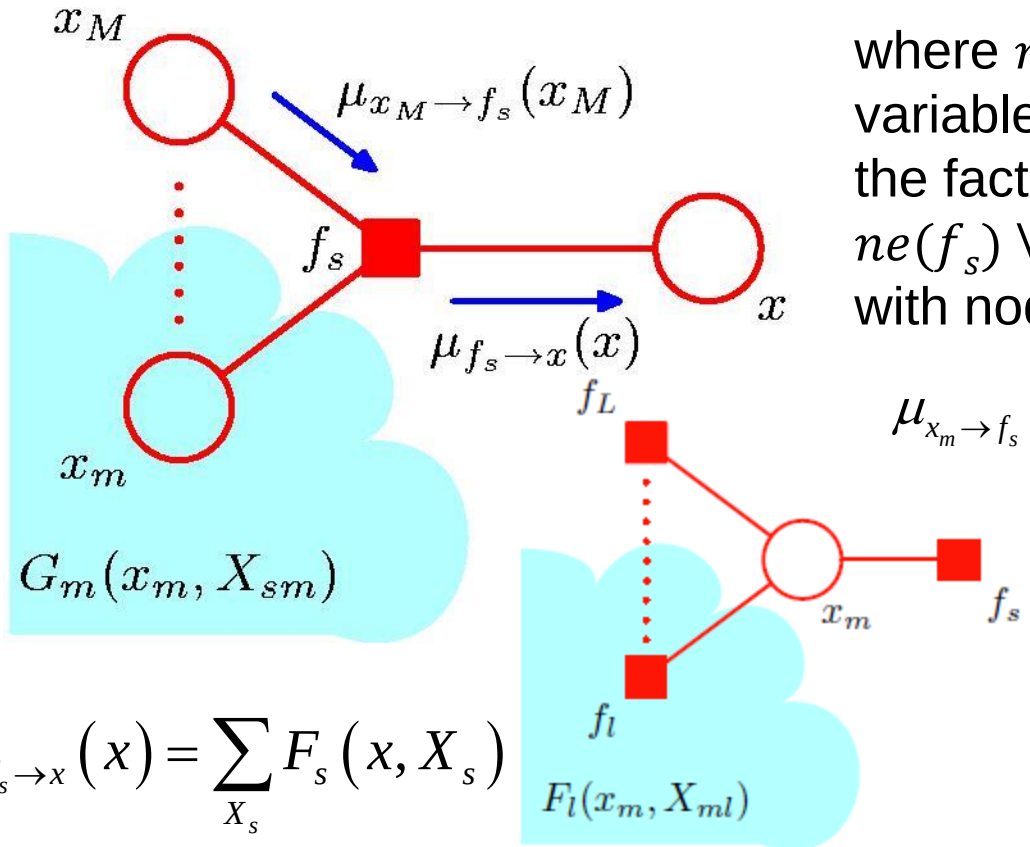
Here, we have denoted the variables associated with factor f_s, in addition to x, by $x_1, \ldots, x_M$.

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

The Sum-Product Algorithm

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

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



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

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

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

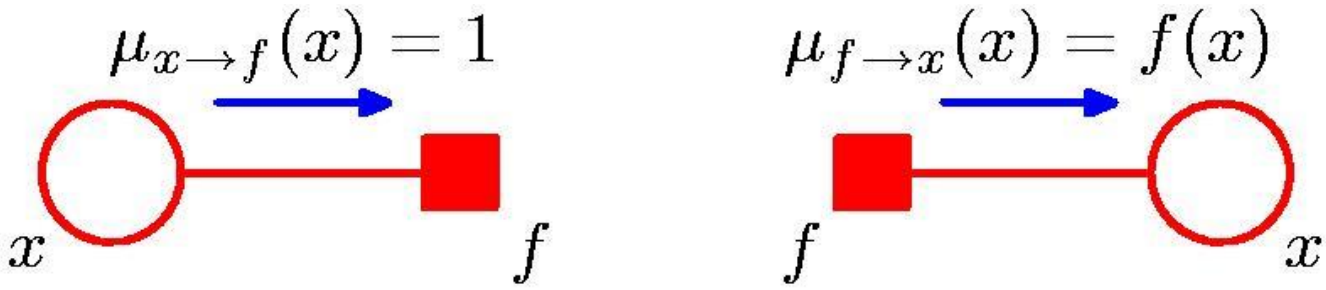
$$= \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

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

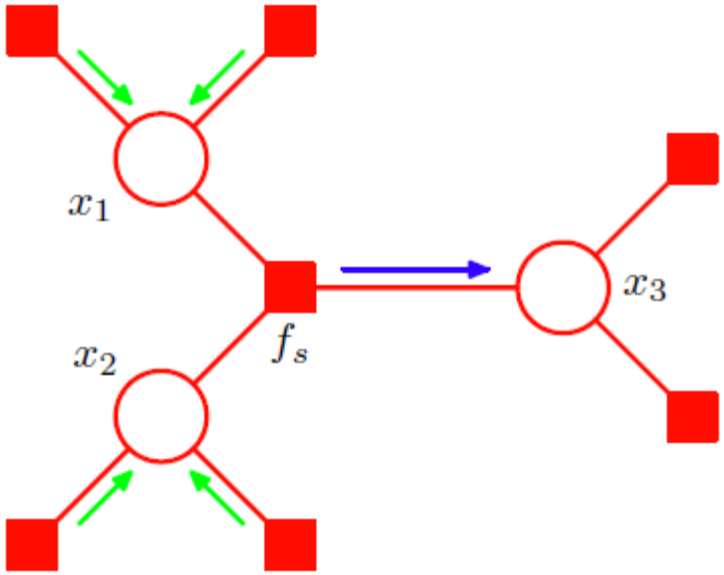
Initialization

- 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.



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*.

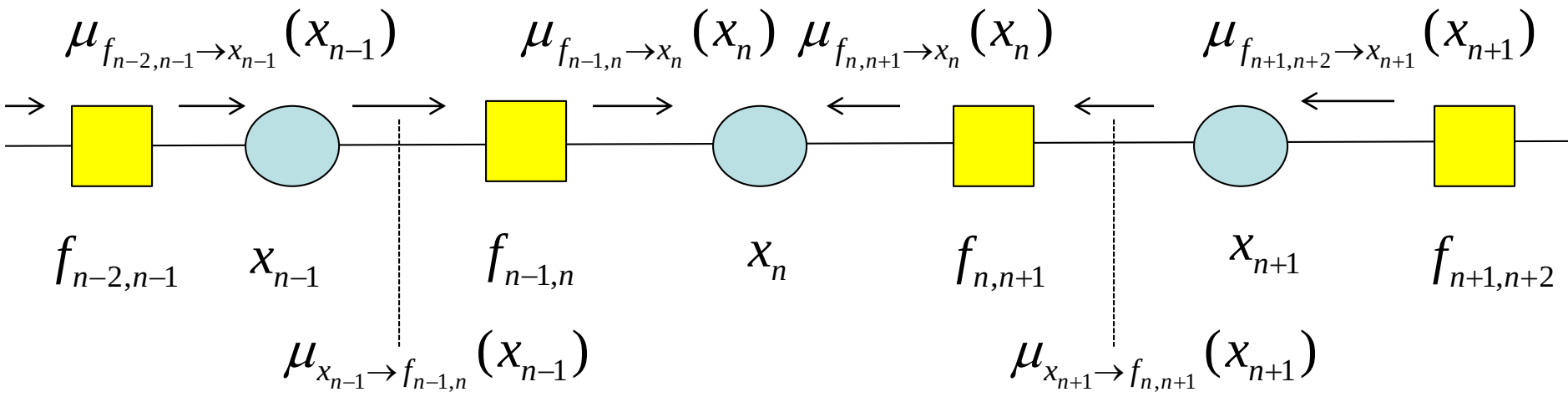
- The outgoing message shown by the blue arrow is obtained by
  - taking the product of all the incoming messages shown by green arrows,
  - multiplying by the factor f_s, and
  - marginalizing over the variables x_1 and x_2.

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.



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

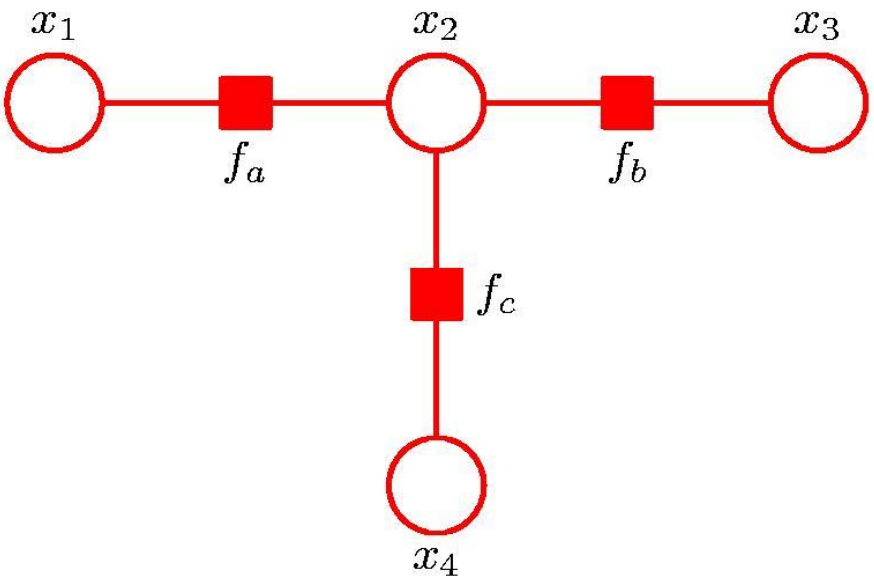
$$=\sum_{x_{n-1}}\psi_{n-1,n}(x_{n-1},x_n)\underbrace{\mu_{f_{n-2,n-1}\to x_{n-1}}(x_{n-1})}_{\mu_\alpha(x_{n-1})}\sum_{x_{n+1}}\psi_{n,n+1}(x_n,x_{n+1})\underbrace{\mu_{f_{n+1,n+2}\to x_{n+1}}(x_{n+1})}_{\mu_\beta(x_{n+1})}$$

- 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

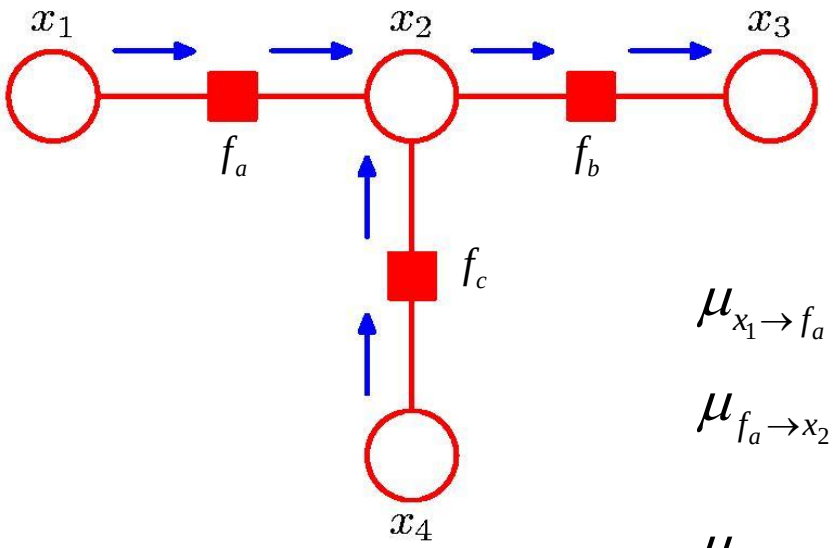
Example: a simple 4-node factor graph



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



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

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

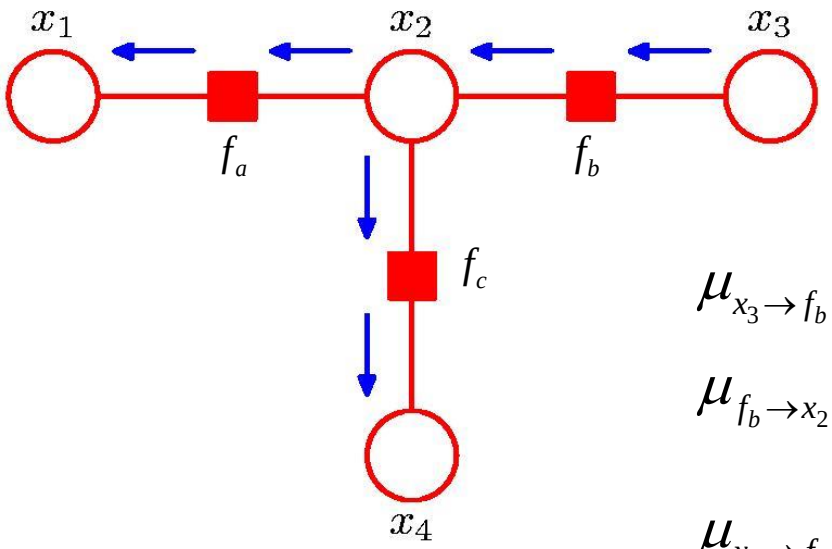
$$\mu_{x_4 \to f_c}(x_4) = 1$$

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

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

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

Sum-Product: Example



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

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

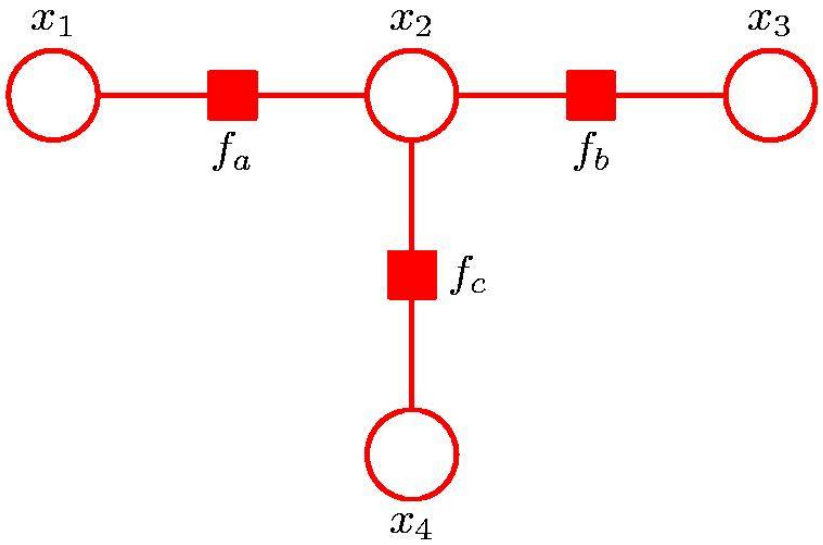
$$\mu_{x_2 \to f_a}(x_2) = \mu_{f_b \to x_2}(x_2)\,\mu_{f_c \to x_2}(x_2)$$

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

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

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

Sum-Product: Example



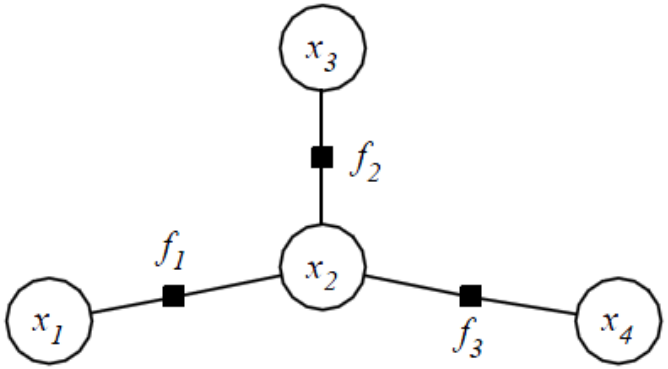
$$\tilde{p}(x_2) = \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2)$$

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

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

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

Propagation in Factor Graphs

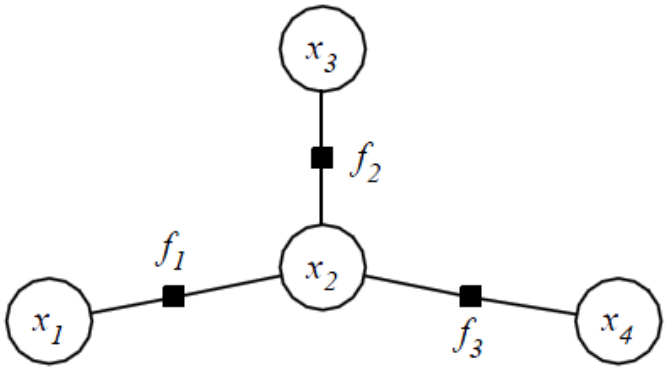


- Another example schedule of messages resulting in computing $p(x_4)$
- Initialize all messages to be 1

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

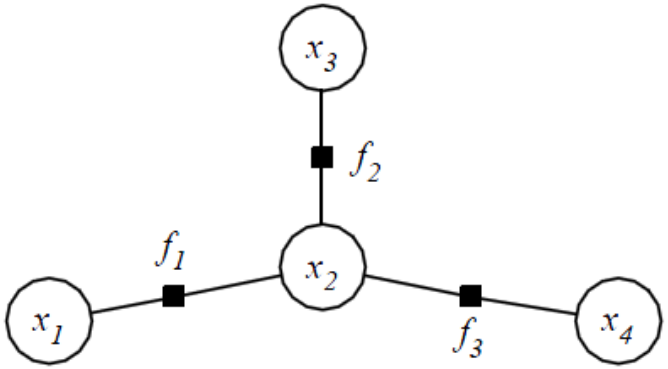
Propagation in Factor Graphs



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

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

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.

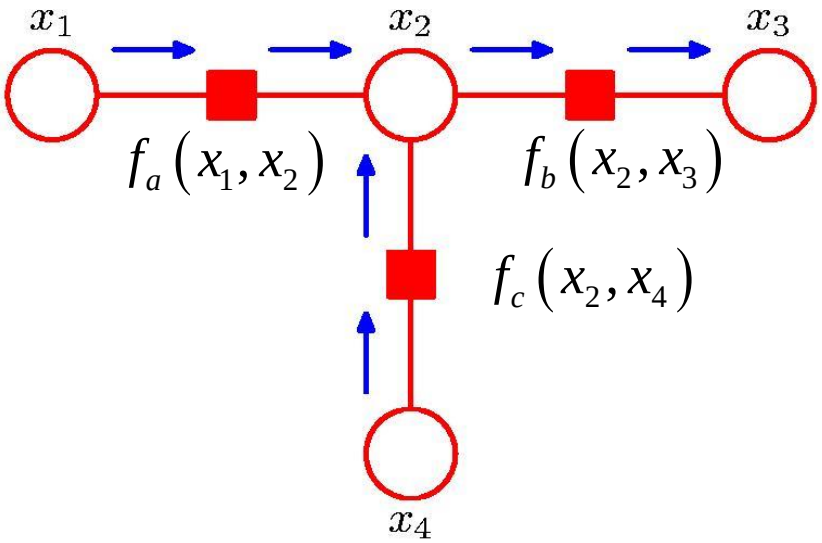
$$= f_s(x_s) \prod_{i \in ne(f_s)} \sum_{\mathbf{x} \setminus x_s} \prod_{j \in ne(x_i) \setminus f_s} F_j(x_i, X_{ij})$$

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

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.

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

- 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).

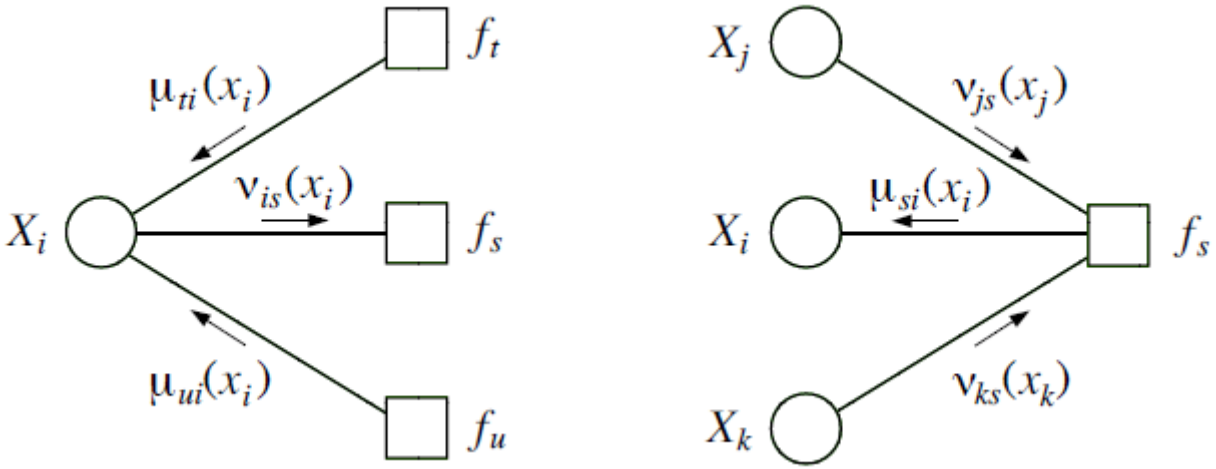
Marginal Associated with each Variable Node

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.

$$
\begin{aligned}
p(x_i) &\equiv \prod_{s \in ne(x_i)} \mu_{f_s \to x_i}(x_i) \\
&= \mu_{f_s \to x_i}(x_i) \prod_{t \in ne(x_i) \setminus f_s} \mu_{f_t \to x_i}(x_i) \\
&= \mu_{f_s \to x_i}(x_i)\, \mu_{x_i \to f_s}(x_i)
\end{aligned}
$$

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:

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

Sum Product Algorithm for a Factor-Tree

Sum-Product($\mathcal{T}$, E)
- Evidence(E)
- f=ChooseRoot($\mathcal{V}$)
- for s $\in \mathcal{N}$(f)
  - μ-Collect(f,s)
- for s $\in \mathcal{N}$(f)
  - ν-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

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

μ-Collect(i,s)
- for $j \in \mathcal{N}(s) \setminus i$
  - ν-Collect(s,j)
- μ-SendMessage(s,i)

ν-Collect(s,i)
- for $t \in \mathcal{N}(i) \setminus s$
  - μ-Collect(i,t)
- ν-SendMessage(i,s)

μ-Distribute(s,i)
- μ-SendMessage(s,i)
- for $t \in \mathcal{N}(i) \setminus s$
  - ν-Distribute(i,t)

ν-Distribute(i,s)
- ν-SendMessage(i,s)
- for $j \in \mathcal{N}(s) \setminus i$
  - μ-Distribute(s,j)

Sum Product Algorithm for a Factor-Tree

μ-SendMessage(s,i)

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

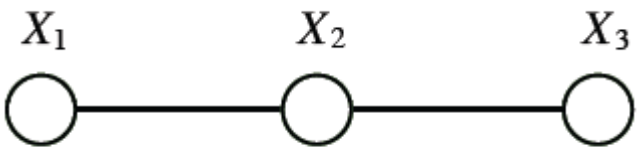
ν-SendMessage(j,i)

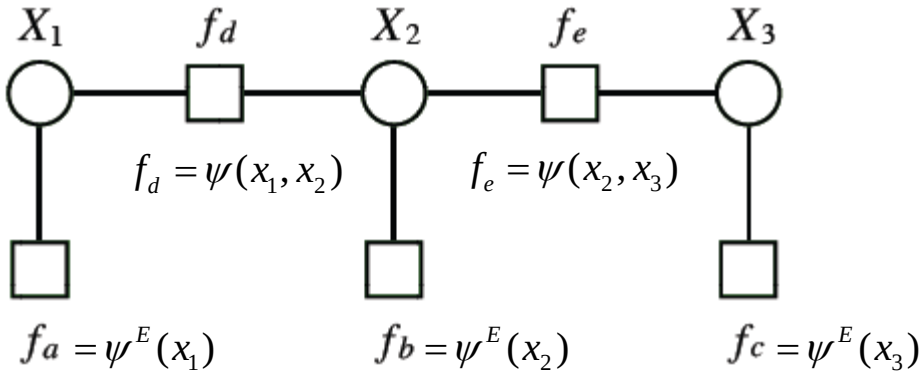
$$\nu_{is}(x_i) = \prod_{t \in \mathcal{N}(i)\setminus\{s\}} \mu_{ti}(x_i)$$

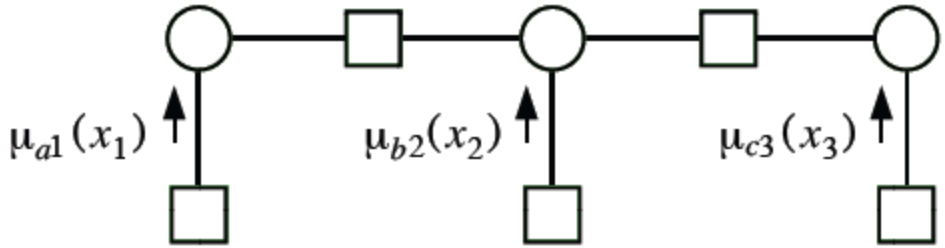
ComputeMarginal(i)

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

Another Example

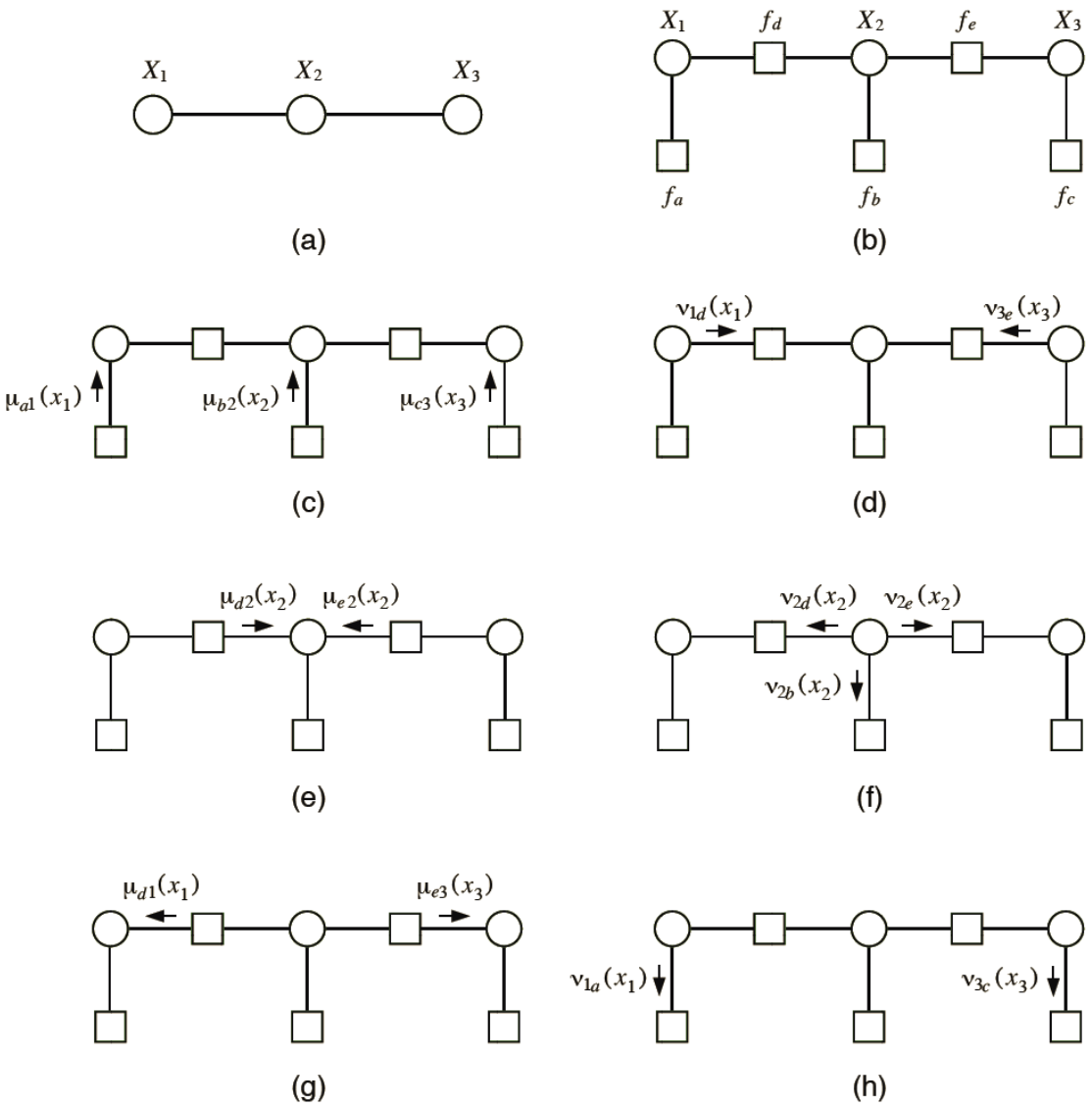




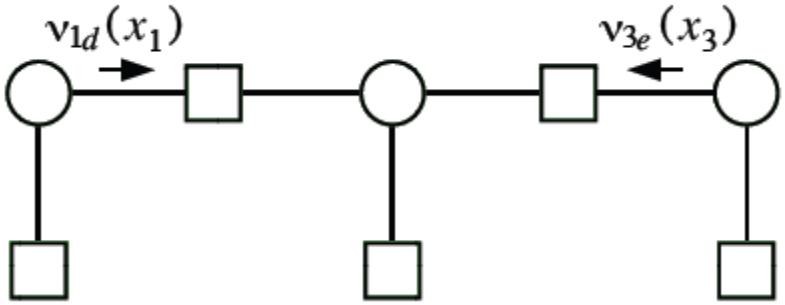


$$\mu_{a1}(x_1) = \sum_{x_{\mathcal{N}(a)\setminus\{1\}}} \left(f_a(x_{\mathcal{N}(a)}) \prod_{j \in \mathcal{N}(a)\setminus\{1\}} \nu_{ja}(x_j) \right) = f_a(x_1) = \psi^E(x_1),\ \mu_{b2}(x_2) = \psi^E(x_2),\ \mu_{c3}(x_3) = \psi^E(x_3)$$

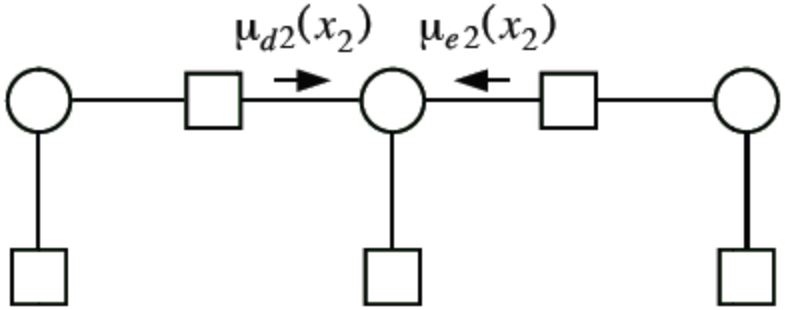
Another Example



Example

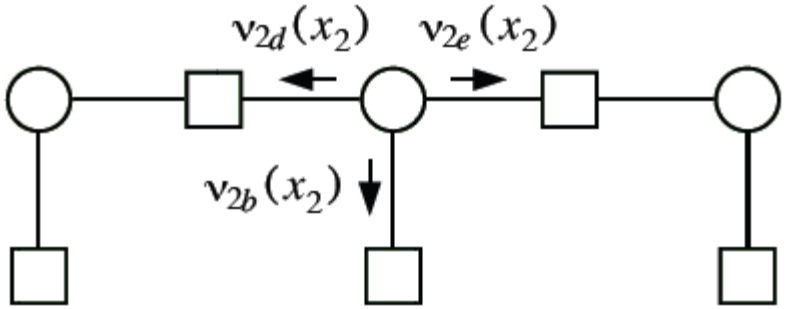


$$\nu_{1d}(x_1) = \prod_{t \in \mathcal{N}(1)\setminus\{d\}} \mu_{t1}(x_1) = \mu_{a1}(x_1) = \psi^E(x_1), \nu_{3e}(x_3) = \psi^E(x_3)$$



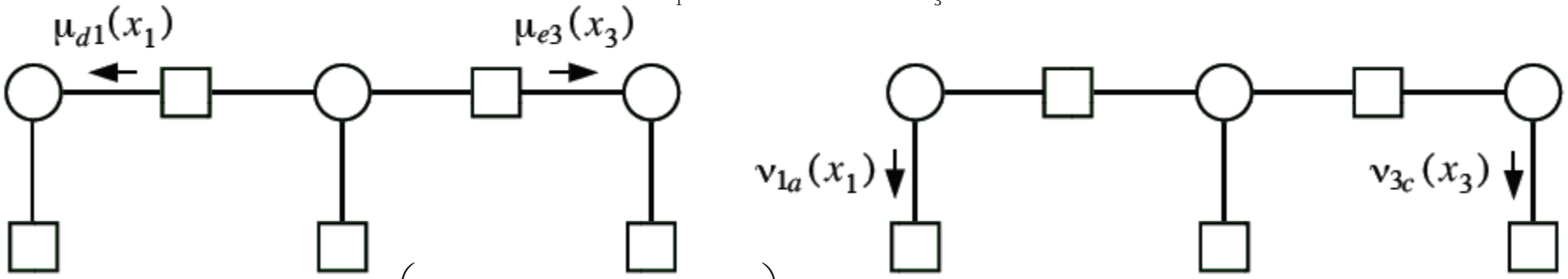
$$\mu_{d2}(x_2) = \sum_{x_{\mathcal{N}(d)\setminus\{2\}}} \left(f_d(x_{\mathcal{N}(d)}) \prod_{j \in \mathcal{N}(d)\setminus\{2\}} \nu_{jd}(x_j) \right) = \sum_{x_1} \psi(x_1, x_2)\psi^E(x_1), \mu_{e2}(x_2) = \sum_{x_3} \psi(x_2, x_3)\psi^E(x_3)$$

Example



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

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



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

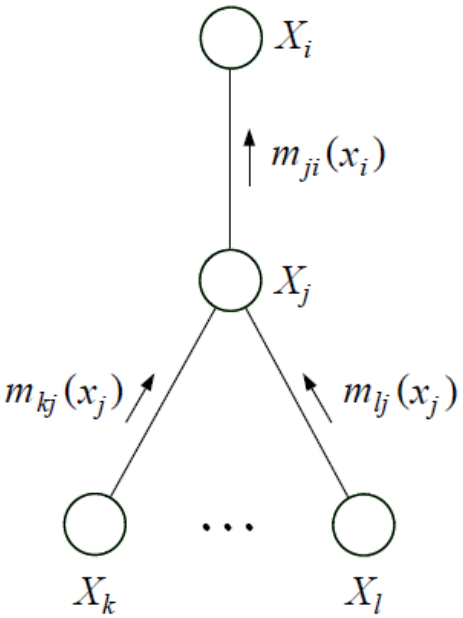
$$\mu_{e3}(x_1) = \sum_{x_2} \psi^E(x_1)\psi(x_1, x_2) \sum_{x_3} \psi^E(x_2)\psi(x_2, x_3) = \nu_{3c}(x_3)$$

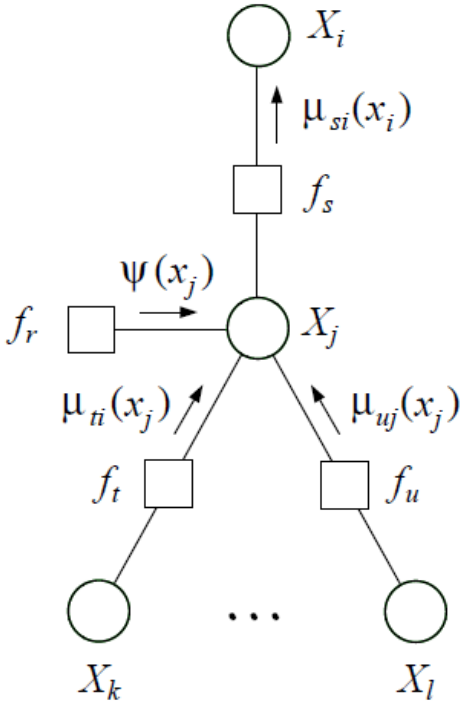
Note that these messages are the same as the corresponding messages that would pass in a run of the SUM-PRODUCT algorithm in the corresponding undirected graph, e.g.

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

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.





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

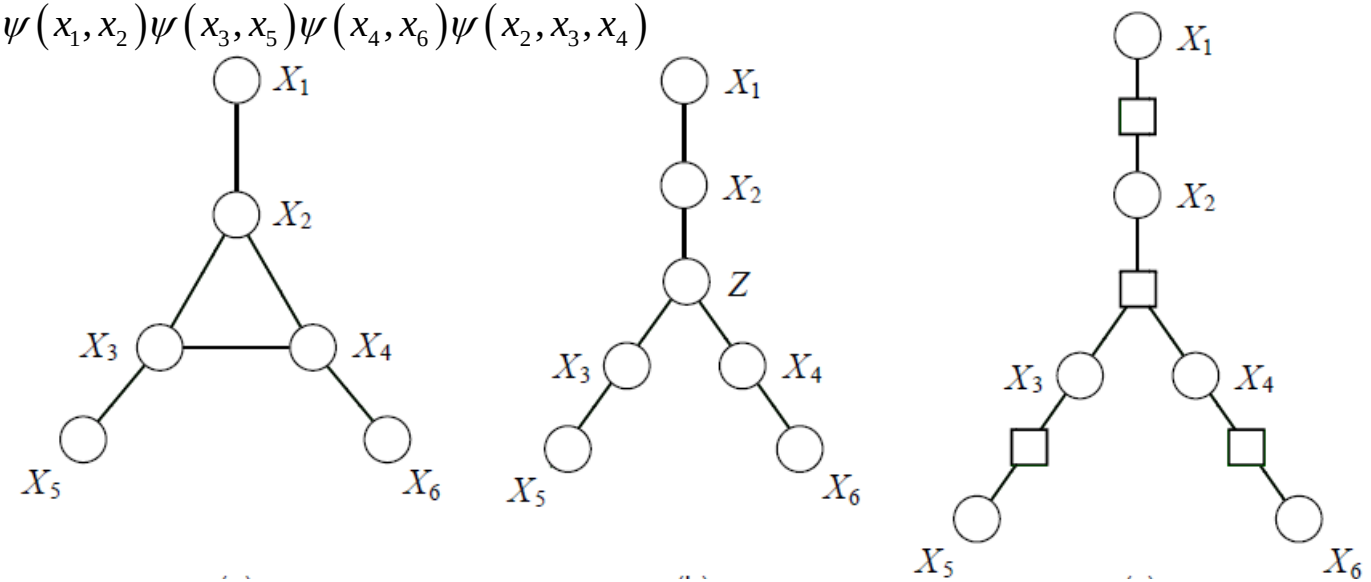
$$= \sum_{x_j} \psi(x_i,x_j)\psi^E(x_j) \prod_{t\in\mathcal{N}'(j)\setminus\{s\}} \mu_{tj}(x_j)$$

In the N'(j) omitting the singleton factor

SUM-PRODUCT is 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.

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



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

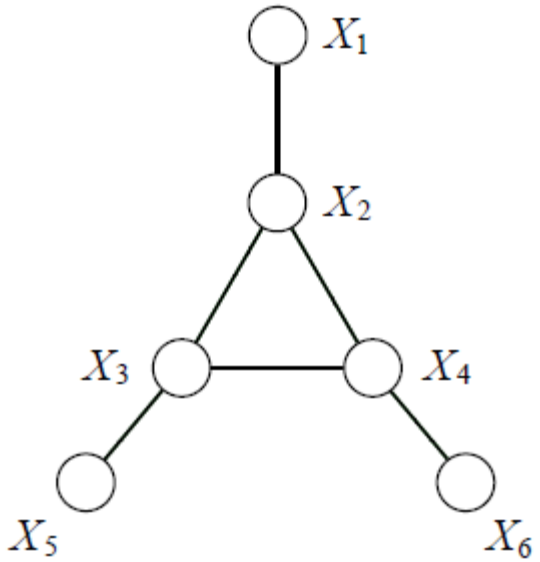
(b) An equivalent undirected model using a super variable Z (range the Cartesian product of the range of X_2,X_3,X_4). Create new potentials $\psi(x_1, Z)\psi(x_5, Z)\psi(x_6, Z)\psi(Z)$

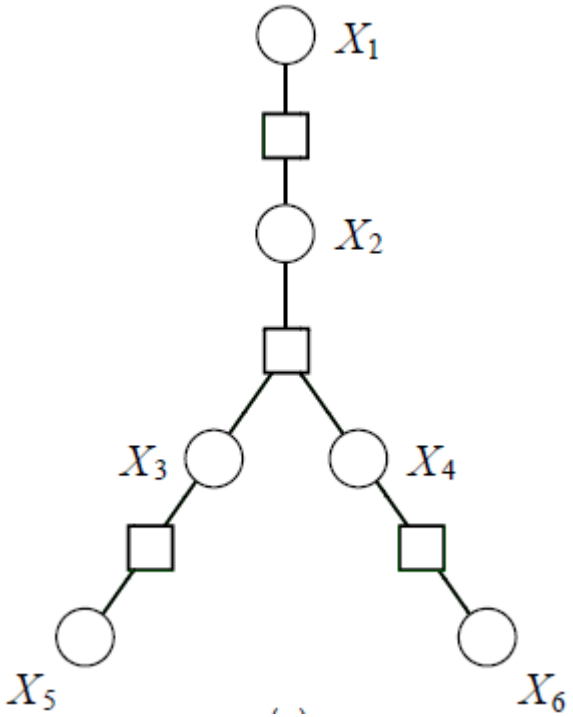
(c) An equivalent factor graph that is a factor-tree. No need for new potentials.

SUM-PRODUCT is Applied to Factor Trees

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

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

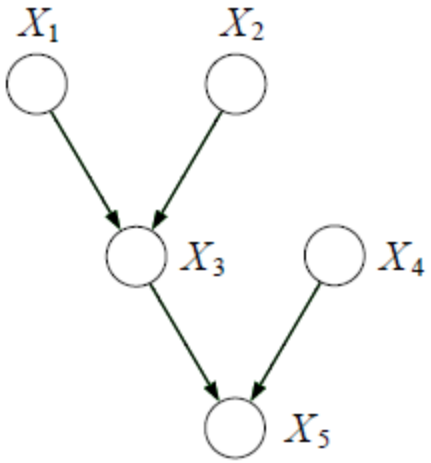


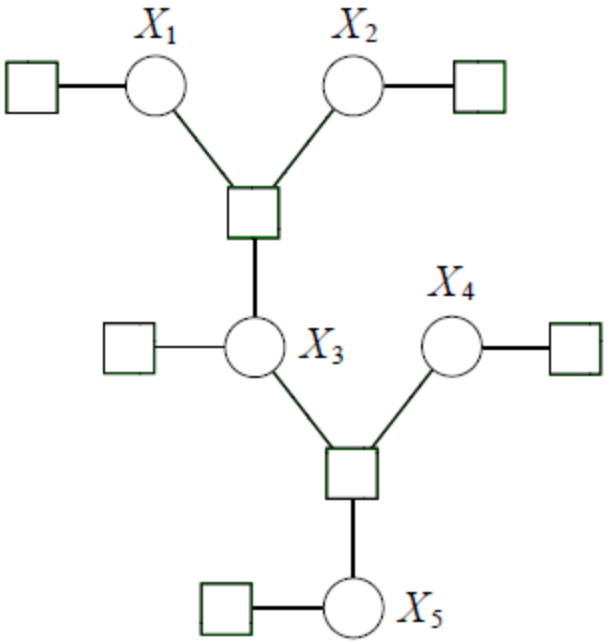


SUM-PRODUCT Applied to Polytrees

- Recall that a polytree is a directed graph that reduces to an undirected tree if we convert each directed edge to an undirected edge. Polytrees have no loops in their underlying undirected graph.

- The corresponding factor graph is a tree with a factor for each family $p(x_i|x_{p(i)})$ and the SUM-PRODUCT algorithm applies.





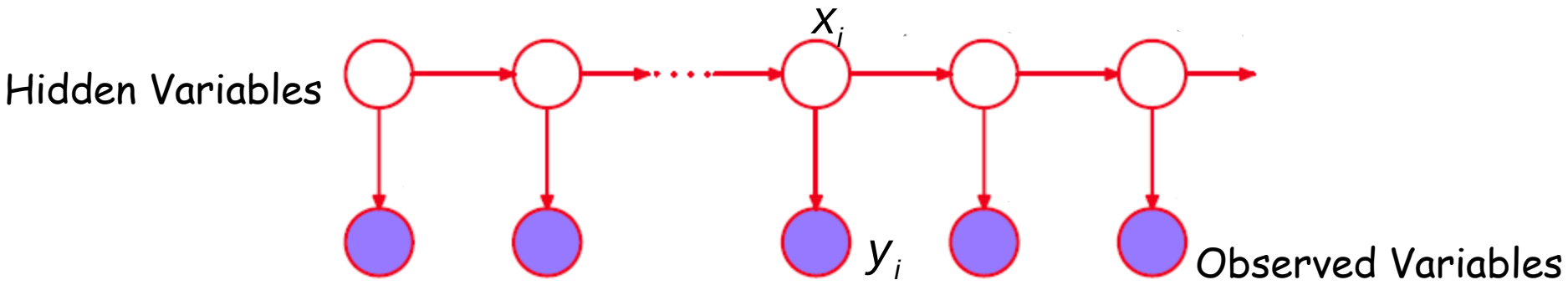
Application in State Space Models

- Inference in HMM involves one forward and one backward pass

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

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

- 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.