ECE 286: Bayesian Machine Perception

Class 7: Factor Graphs and the Sum-Product Message Passing

Florian Meyer

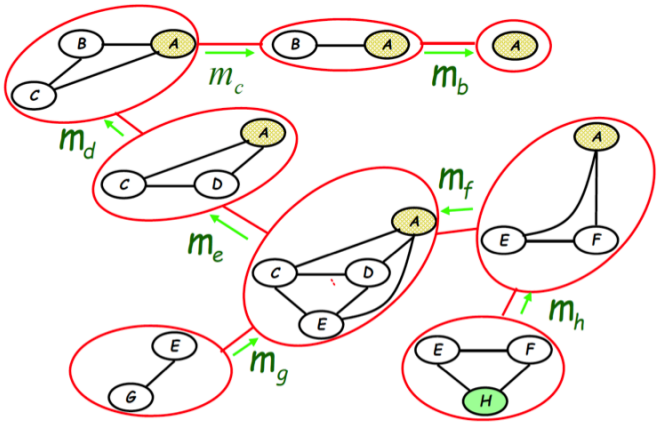
Electrical and Computer Engineering Department
University of California San Diego

Complexity of Variable Elimination

- The total computational complexity is determined by the size of the largest elimination clique:
  - The largest elimination clique can be determined by purely graph-theoretic reasoning
  - ``Good'' elimination order lead to small cliques and hence to a reduced computational complexity
  - Finding the best elimination ordering for a graph is an NP-hard problem
  - For many graphs ``obvious'' optimal or near-optimal elimination orders can be found

From Elimination to Message Passing

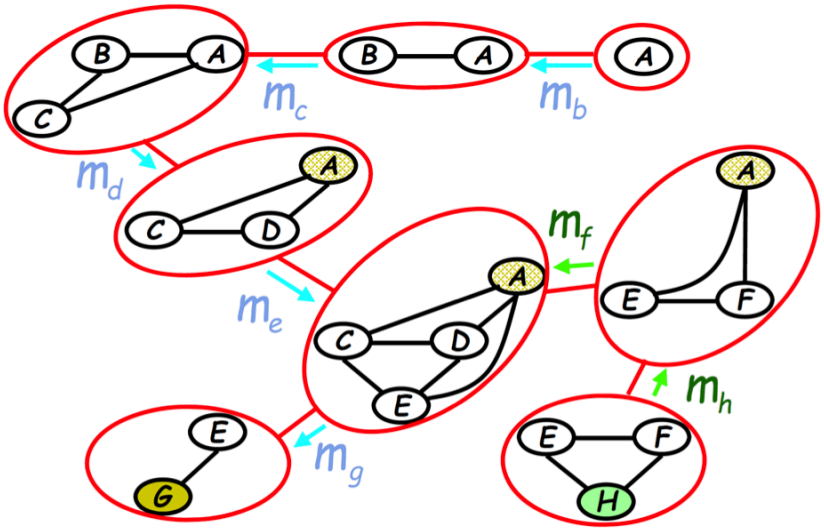
- Variable elimination provides one marginal posterior distribution ($p(a|h)$ in our example)
- Do we need to perform a complete elimination for every other posterior distribution?
- Recall that variable elimination can be interpreted as passing messages on a clique tree



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

From Elimination to Message Passing

- Let's consider the task of calculating $p(g|h)$
- Messages m_f and m_h are reused, others need to be recomputed



From Elimination to Message Passing

- Recall the variable elimination algorithm:
  1. Choose an elimination order in which the node j corresponding to the marginal distribution of interest is the final node
  2. Iteratively: Eliminate the next node i by grouping all potentials containing $\boldsymbol{x}_i$ and take sum/product over $\boldsymbol{x}_i$

Elimination on a Tree

- Let $m_{ji}(\boldsymbol{x}_i)$ denote the factor resulting from eliminating the ``leaf'' variables k, l and j

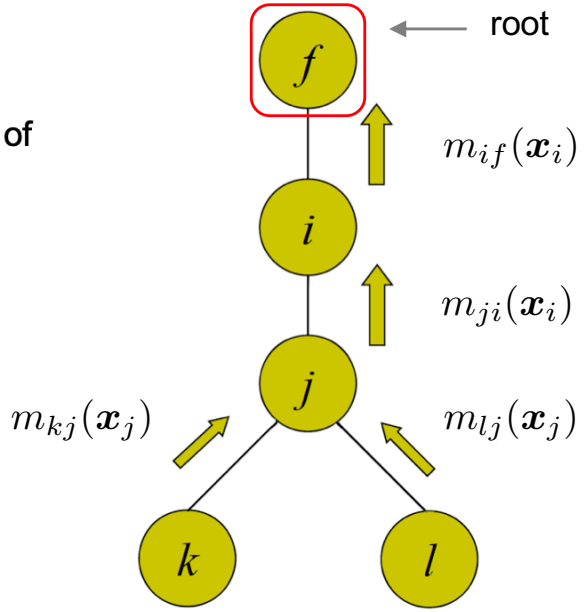
$$m_{ji}(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_j} \Big(\psi_i(\boldsymbol{x}_j) \psi_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \Big) \prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}(\boldsymbol{x}_j)$$

all nodes that are neighbors of node j in the graph

- This can be interpreted as a message send from j to i

- $m_{ji}(\boldsymbol{x}_i)$ represents the ``belief'' of $\boldsymbol{x}_i$ from the ``lower part'' of the graph

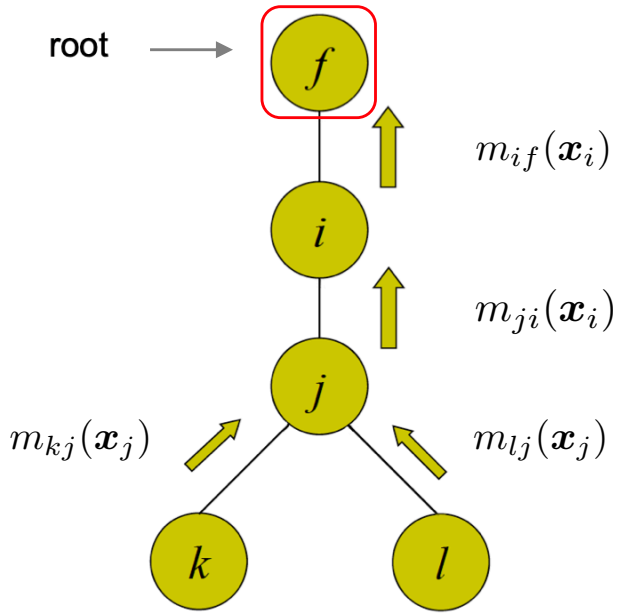
- The marginal distributions is obtained as product of local potentials and all incoming messages at the root



$$p(\boldsymbol{x}_f) \propto \psi(\boldsymbol{x}_f) \prod_{e \in \mathcal{N}(f)} m_{ef}(\boldsymbol{x}_f)$$
$$= \psi(\boldsymbol{x}_f)\, m_{if}(\boldsymbol{x}_f)$$

Elimination on a Tree

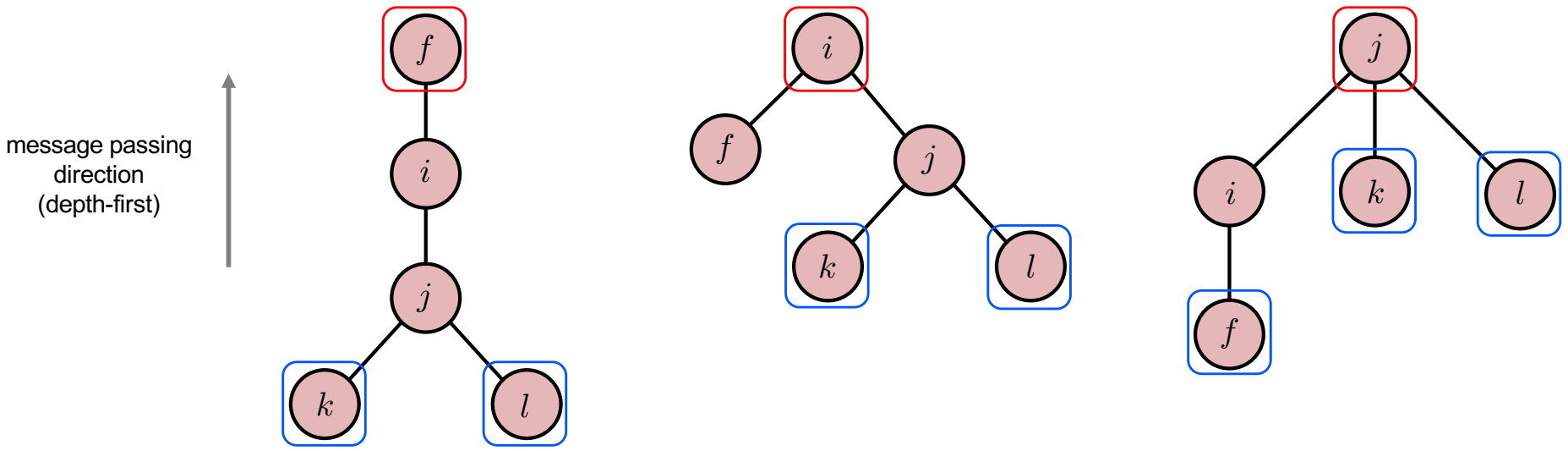
- Elimination on trees (to get marginal distribution corresponding to the root node) is equivalent to message passing along the edges of the graph



$$m_{ji}(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_j} \left(\psi_i(\boldsymbol{x}_j) \psi_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \prod_{k \in N(j) \setminus i} m_{kj}(\boldsymbol{x}_j)$$

Elimination on a Tree

- Every tree can be redrawn such that an arbitrary node is the root node
- Calculate marginal distribution corresponding to the root node: Start at leaf nodes and pass messages along the edges of the graph up to the root



From Elimination to Message Passing

- Recall the variable elimination algorithm:
  1. Choose an elimination order in which the node j corresponding to the marginal distribution of interest is the final node
  2. Iteratively: Eliminate the next node i by grouping all potentials containing $\boldsymbol{x}_i$ and take sum/product over $\boldsymbol{x}_i$

- Variable elimination by passing messages on a tree graph:
  1. Redraw the tree such that node j corresponding to the marginal distribution of interest is the final node
  2. Eliminate nodes in a depth-first order starting from leaf nodes (can be interpreted as messages passed along the edges of the graph)

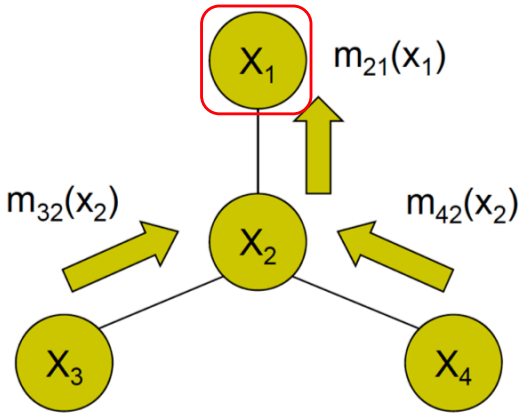
$\Longrightarrow$ Tree graphs are an ideal data-structure for inference

Message Passing on Trees

- Redrawing the graph and using a depth-first message order is equivalent to the message passing protocol:
  - A node can send a message to its neighbors when (and only when) it has received messages from all the other neighbors

- Naive approach to compute all node marginals:

Calculating $p(\boldsymbol{x}_1)$

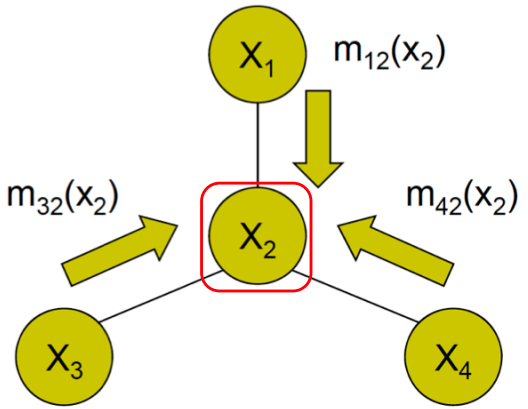


Message Passing on Trees

- Redrawing the graph and using a depth-first message order is equivalent to the message passing protocol:
  - A node can send a message to its neighbors when (and only when) it has received messages from all the other neighbors

- Naive approach to compute all node marginals:

Calculating $p(\boldsymbol{x}_2)$

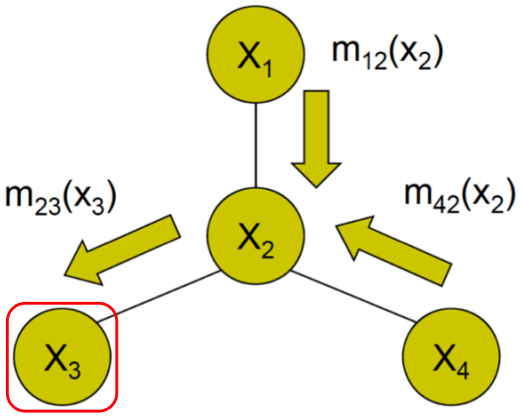


Message Passing on Trees

- Redrawing the graph and using a depth-first message order is equivalent to the message passing protocol:
  - A node can send a message to its neighbors when (and only when) it has received messages from all the other neighbors

- Naive approach to compute all node marginals:

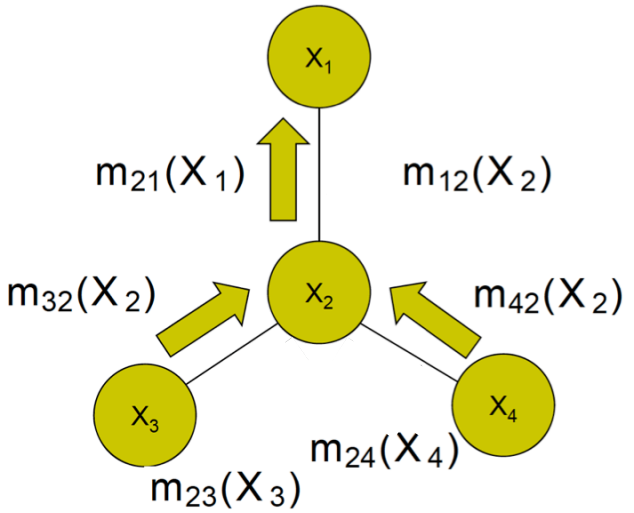
Calculating $p(\boldsymbol{x}_3)$



Message Passing on Trees

- Redrawing the graph and using a depth-first message order is equivalent to the message passing protocol:
  - A node can send a message to its neighbors when (and only when) it has received messages from all the other neighbors

- Dynamic programming approach to compute all node marginals (*The Forward-Backward Algorithm*):

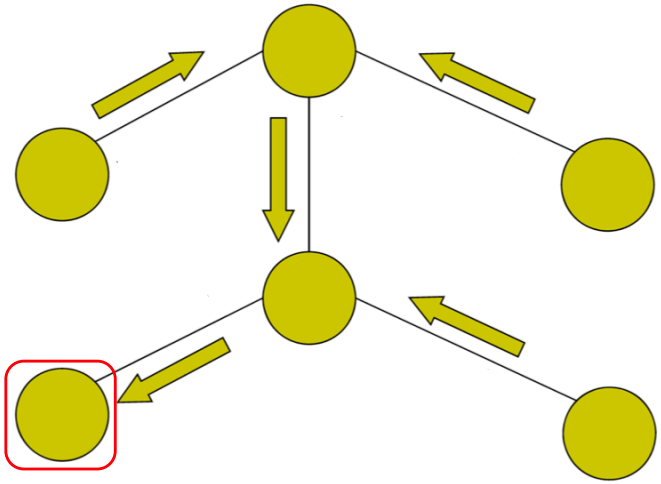


The Sum-Product Algorithm: Complexity

- For an arbitrary tree graph with V nodes, let C be the complexity of calculating the marginal corresponding to one node
- Complexity of naive approach: VC
- Complexity of dynamic programming approach: $2C$

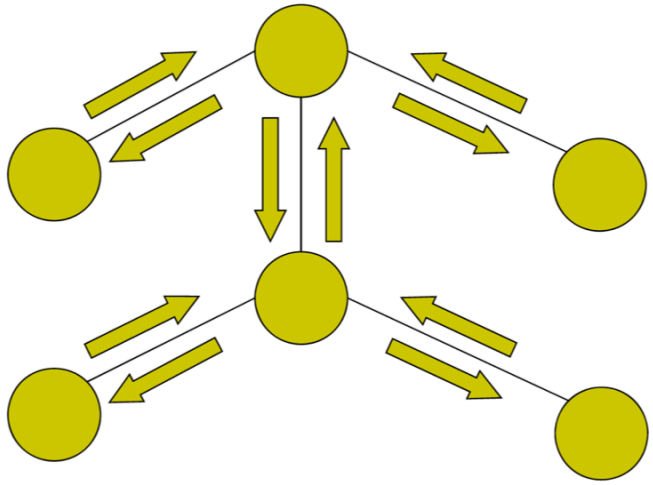
The Sum-Product Algorithm (Sequential Impl.)

- Define an arbitrary node as the root and
  - calculate all messages starting from leaf nodes (collect or forward pass)
  - redefine former leaves as roots and calculate missing messages (distribute or backward pass)
- Message passing is completed whenever a pair of messages has been computed for each edge $\Longrightarrow$ marginals can be computed for all nodes



The Sum-Product Algorithm (Sequential Impl.)

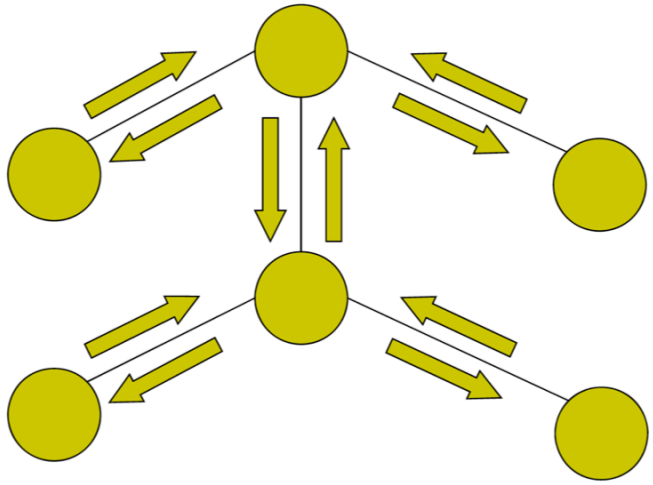
- Define an arbitrary node as the root and
  - calculate all messages starting from leaf nodes (collect or forward pass)
  - redefine former leaves as roots and calculate missing messages (distribute or backward pass)
- Message passing is completed whenever a pair of messages has been computed for each edge $\Longrightarrow$ marginals can be computed for all nodes



The Sum-Product Algorithm (Parallel Impl.)

- For a node of degree d, whenever messages have arrived on any subset of $d-1$ nodes, compute the messages for the remaining edge and send

- Message passing is completed whenever a pair of messages has been computed for each edge $\Longrightarrow$ marginals can be computed for all nodes



Correctness of the Sum-Product Algorithm

- Corollary: The synchronous implementation is ``non-blocking''
- Theorem: Message passing guarantees that all marginals are obtained in a tree-graph
- What graphs that are not trees?

Factor Graphs

- **Definition:** A factor graph is a bipartite graph that expresses the structure of a factorization $f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \prod_{I \in \mathcal{I}} f_I(\boldsymbol{x}_I)$

- **Example:**

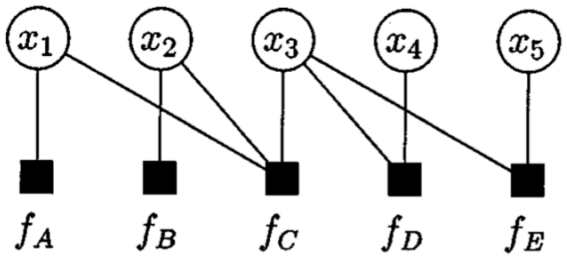
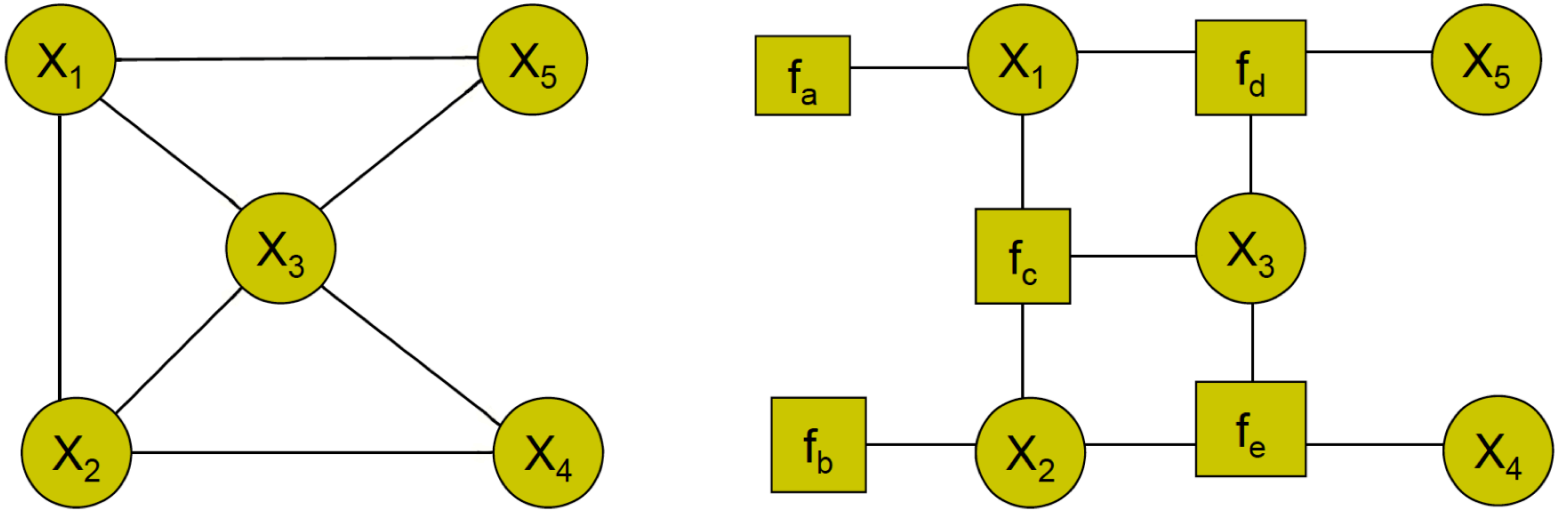


Fig. 1. A factor graph for the product $f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3) \cdot f_D(x_3, x_4)f_E(x_3, x_5)$.

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Trans. Inf. Theory, 2001.

Factor Graphs

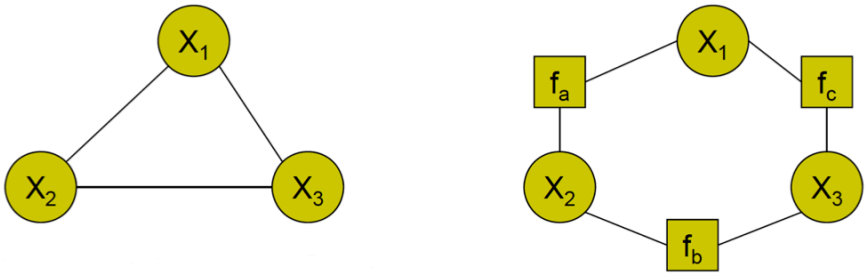
- Example 1:



$$\underbrace{p(\boldsymbol{x}_1)}_{g_a(\boldsymbol{x}_1)}\ \underbrace{p(\boldsymbol{x}_2)}_{g_b(\boldsymbol{x}_2)}\ \underbrace{p(\boldsymbol{x}_3|\boldsymbol{x}_1,\boldsymbol{x}_2)}_{g_c(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3)}\ \underbrace{p(\boldsymbol{x}_5|\boldsymbol{x}_1,\boldsymbol{x}_3)}_{g_d(\boldsymbol{x}_5,\boldsymbol{x}_1,\boldsymbol{x}_3)}\ \underbrace{p(\boldsymbol{x}_4|\boldsymbol{x}_2,\boldsymbol{x}_3)}_{g_e(\boldsymbol{x}_4,\boldsymbol{x}_2,\boldsymbol{x}_3)}$$

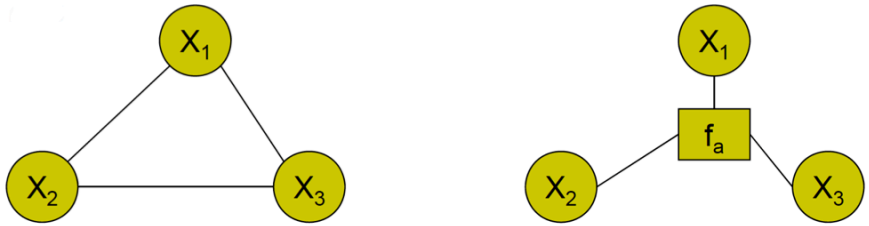
Factor Graphs

- Example 2:



$$\psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = g_a(\boldsymbol{x}_1, \boldsymbol{x}_2) g_b(\boldsymbol{x}_2, \boldsymbol{x}_3) g_c(\boldsymbol{x}_3, \boldsymbol{x}_1)$$

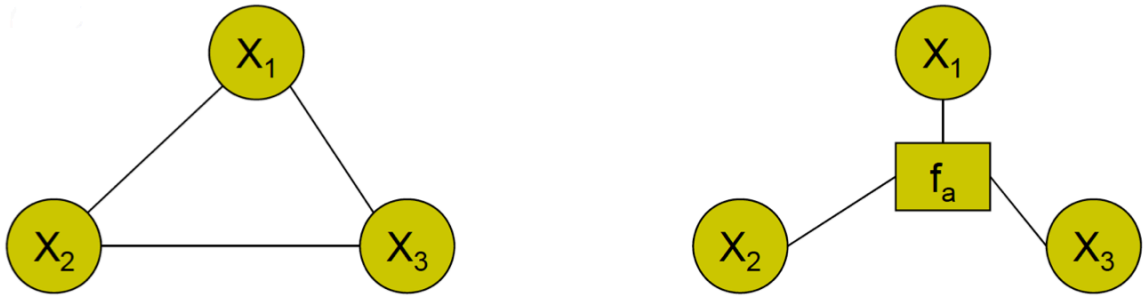
- Example 3:



$$\psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = g_a(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$$

Cycle-Free Factor Graph

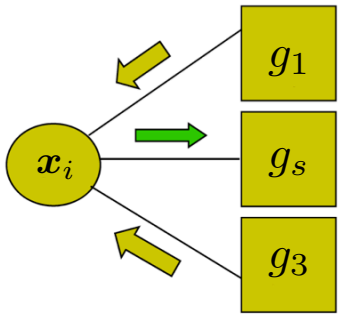
- A factor graph is cycle-free if the if the graph obtained by ignoring the distinction between variable nodes and factor nodes is an undirected tree
- A cycle-free factor graph is also referred to as factor tree



$$\psi(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) = g_a(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$$

Message Passing on a Cycle Free Factor Graph

- Two types of messages
  - ν from variables to factors
  - ϕ from factors to variables

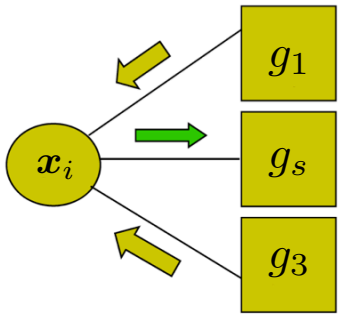


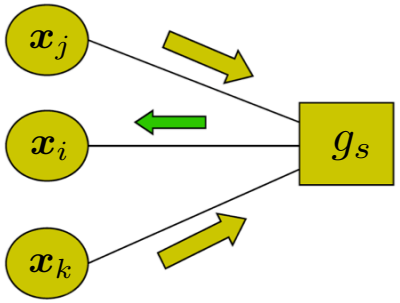
$$\nu_{is}(\boldsymbol{x}_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \phi_{ti}(\boldsymbol{x}_i)$$

$$\phi_{si}(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_{\mathcal{N}(s) \setminus i}} \left(g_s(\boldsymbol{x}_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(\boldsymbol{x}_j) \right)$$

Message Passing on a Cycle Free Factor Graph

- Message passing protocol: A message to a neighboring node can only be send when it has received messages from all its other neighbors
- Marginal distribution can be calculated as $b(\boldsymbol{x}_i) \propto \prod_{t \in \mathcal{N}(i)} \phi_{ti}(\boldsymbol{x}_i) = \phi_{ti}(\boldsymbol{x}_i)\nu_{it}(\boldsymbol{x}_i)$



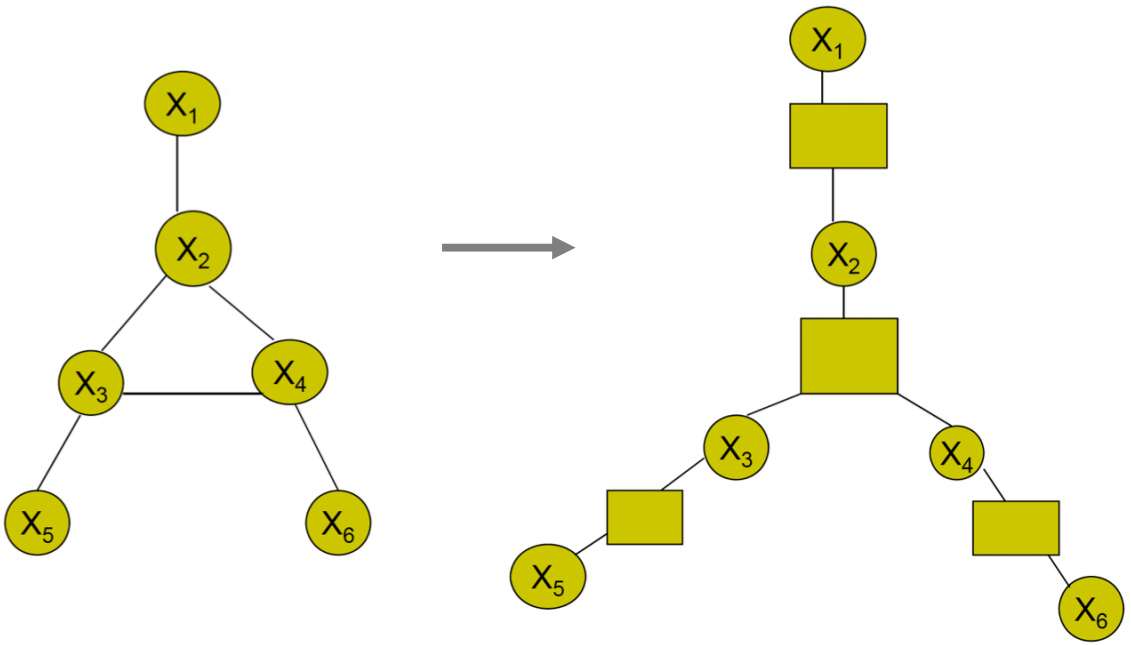


$$\nu_{is}(\boldsymbol{x}_i) = \prod_{t \in \mathcal{N}(i) \setminus s} \phi_{ti}(\boldsymbol{x}_i)$$

$$\phi_{si}(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_{\mathcal{N}(s) \setminus i}} \left(g_s(\boldsymbol{x}_{\mathcal{N}(s)}) \prod_{j \in \mathcal{N}(s) \setminus i} \nu_{js}(\boldsymbol{x}_j) \right)$$

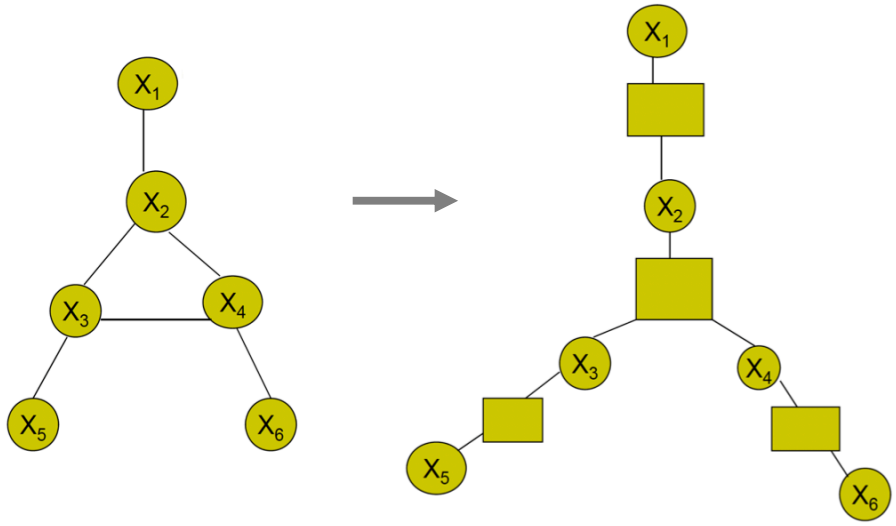
Why Factor Graphs?

- Tree-like Markov random fields may turn into cycle-free factor graphs
- A cycle free factor graphs guarantees correctness of sum-product message passing



Why Factor Graphs?

"When a factor graph is cycle-free, the factor graph not only encodes in its structure the factorization of the global function, but also encodes arithmetic expressions by which the marginal functions associated with the global function may be computed"

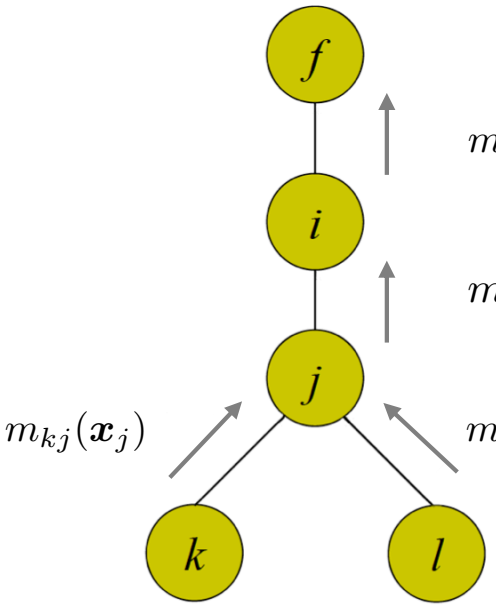


F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Trans. Inf. Theory, 2001.

Max-Product Algorithm: The Joint MAP Estimate

- The joint MAP estimate can be calculated by performing the an similar message passing procedure as the one performed by the sum-product algorithm

$$\max_{\boldsymbol{x}} p(\boldsymbol{x}) = \max_{\boldsymbol{x}_f} \left(\psi(\boldsymbol{x}_f)\, m_{if}(\boldsymbol{x}_f) \right)$$

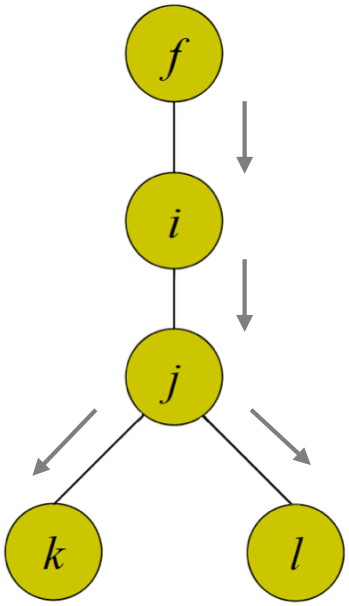


maximization instead of summation

$$m_{ji}(\boldsymbol{x}_i) = \max_{\boldsymbol{x}_j} \left(\psi_i(\boldsymbol{x}_j)\psi_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \prod_{k \in N(j) \setminus i} m_{kj}(\boldsymbol{x}_j)$$

Max-Product Algorithm: The Joint MAP Estimate

- The joint MAP estimate can be calculated by performing the an similar message passing procedure as the one performed by the sum-product algorithm with a bookkeeping backward pass



$$\hat{\boldsymbol{x}}_f = \text{argmax}_{\boldsymbol{x}_f}\big(\psi(\boldsymbol{x}_f)\, m_{if}(\boldsymbol{x}_f)\big)$$

$$\hat{\boldsymbol{x}}_i = \text{argmax}_{\boldsymbol{x}_i}\big(\psi(\boldsymbol{x}_i)\, \psi(\hat{\boldsymbol{x}}_f, \boldsymbol{x}_i)\, m_{ji}(\boldsymbol{x}_i)\big)$$

$$\hat{\boldsymbol{x}}_j = \text{argmax}_{\boldsymbol{x}_j}\big(\psi(\boldsymbol{x}_j)\, \psi(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j)\, m_{kj}(\boldsymbol{x}_j)\, m_{lj}(\boldsymbol{x}_j)\big)$$

$$\hat{\boldsymbol{x}}_l = \text{argmax}_{\boldsymbol{x}_l}\big(\psi(\boldsymbol{x}_l)\, \psi(\boldsymbol{x}_l, \hat{\boldsymbol{x}}_j)\big)$$

$$\hat{\boldsymbol{x}}_k = \text{argmax}_{\boldsymbol{x}_k}\big(\psi(\boldsymbol{x}_k)\, \psi(\boldsymbol{x}_k, \hat{\boldsymbol{x}}_j)\big)$$

Summary

- Sum-Product message passing can be used to calculated marginal distributions corresponding to all nodes in the graph efficiently
- If a probability distribution can be represented by a cycle free factor graph (factor tree), the sum-product algorithm is guaranteed do produce exact marginal posterior distributions
- The joint MAP estimate can be computed by replacing sum with max in the sum-product algorithm and performing an extra bookkeeping step