Inference in Probabilistic Graphical Models

Luigi Portinale

University of Piemonte Orientale, Italy

March 7-10, 2017

Introduction

The task of inference in a PGM is to compute the posterior probability of a set of query nodes, given a set of evidence nodes.

Inference in a PGM

Given a set of random variables X whose distribution is represented as a PGM, given a subset $Q \subseteq X$ (called query variables) and a subset $E \subset X$ of observed variables such that $Q \cap E = \emptyset$, the inference task consists in computing $\mathcal{P}(Q|\mathbf{e})$ for a given state $\mathbf{e}$ of variables in E.

- Inference can be efficiently implemented when the structure of the PGM is a tree (actually a poly-tree)
- Exact approaches try to exploit such a feature

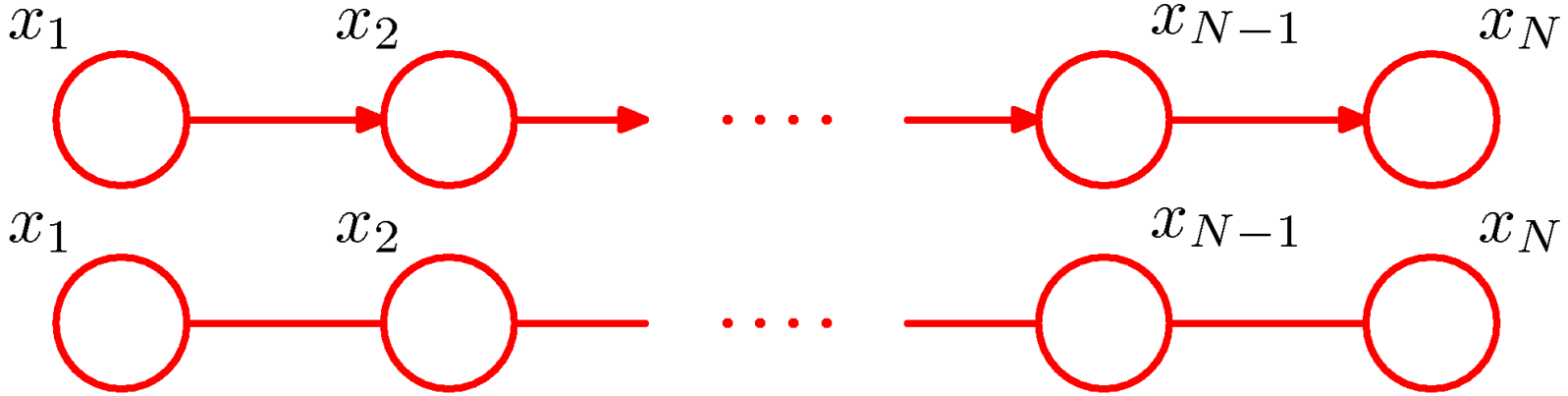
- We have seen that a directed model can be converted in an undirected model by discarding some independencies (moralization);
- We have seen the notion of a chordal graph; they have the property of having a junction tree (definition later on);
- We will see how to build such a tree by transforming a given graphical model, and we will discuss how to exploit such a tree structure to perform inference;
- NB. Transformations will possibly discard some independencies present in the original model, but the resulting distribution has the same joint probability as the original one

Outline

- Inference on a chain
- Inference on polytrees
- Sum-Product Algorithm
- Junction tree Algorithm

Inference on a chain

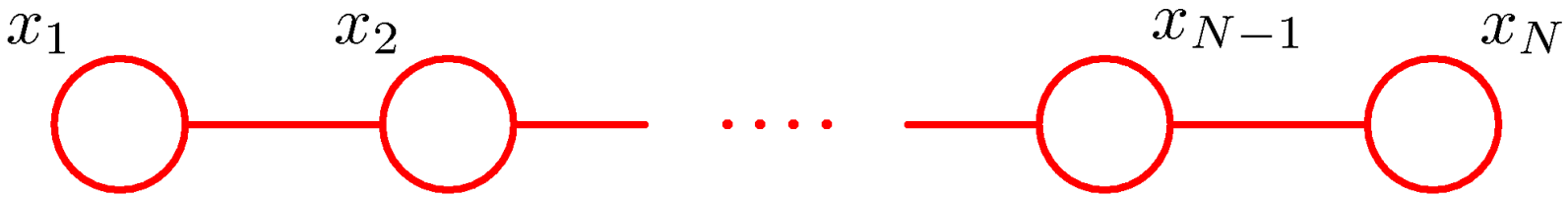
Consider the following chains (they represent the same set of conditional independencies)



Consider the undirected version

$$\mathcal{P}(\mathbf{x}) = \frac{1}{Z}\Psi_{1,2}(x_1, x_2)\Psi_{2,3}(x_2, x_3)\ldots\Psi_{N-1,N}(x_{N-1}, x_N)$$

If each variable has K states, each potential is a $K \times K$ table and we have $(N-1)K^2$ parameters



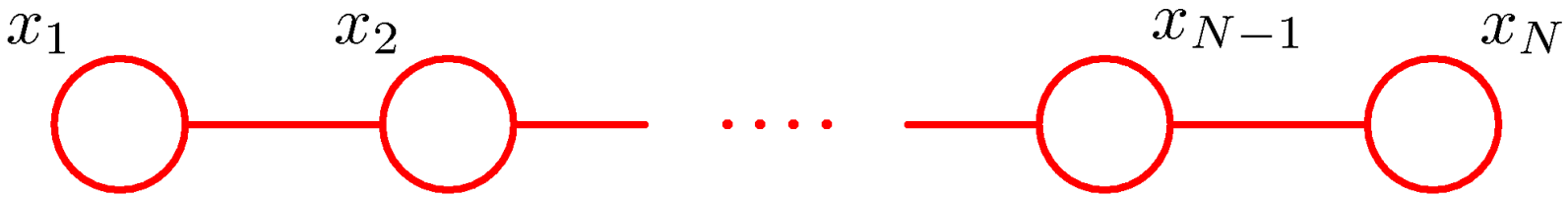
$$\mathcal{P}(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} \mathcal{P}(\mathbf{x})$$

$$= \frac{1}{Z} \sum_{x_1} \Psi_{1,2}(x_1, x_2) \cdots \sum_{x_{n-1}} \Psi_{n-1,n}(x_{n-1}, x_n)$$

$$\sum_{x_{n+1}} \Psi_{n+1,n+2}(x_{n+1}, x_{n+2}) \cdots \sum_{x_N} \Psi_{N-1,N}(x_{N-1}, x_N)$$

Now observe that the summation over x_N involves the last term, producing a factor of x_{N-1} ($\Psi'_{N-1} = \sum_{x_N} \Psi_{N-1,N}(x_{N-1}, x_N)$) and the summation over x_{N-1} will involve only $\Psi'_{N-1}\Psi_{N-2,N-1}(x_{N-2}, x_{N-1})$ and so on.

Simlar considerations apply for x_1 towards x_2, towards x_3, etc...



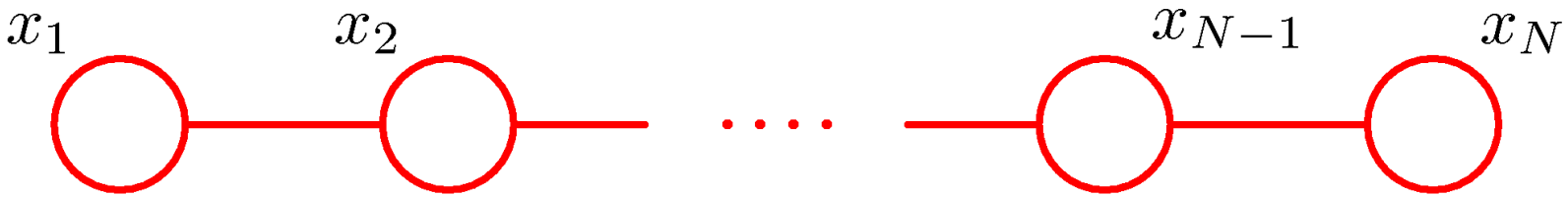
$$\mathcal{P}(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} \mathcal{P}(\mathbf{x})$$

$$= \frac{1}{Z} \sum_{x_1} \Psi_{1,2}(x_1, x_2) \ldots \sum_{x_{n-1}} \Psi_{n-1,n}(x_{n-1}, x_n)$$

$$\sum_{x_{n+1}} \Psi_{n+1,n+2}(x_{n+1}, x_{n+2}) \ldots \sum_{x_N} \Psi_{N-1,N}(x_{N-1}, x_N)$$

Now observe that the summation over x_N involves the last term, producing a factor of x_{N-1} ($\Psi'_{N-1} = \sum_{x_N} \Psi_{N-1,N}(x_{N-1}, x_N)$) and the summation over x_{N-1} will involve only $\Psi'_{N-1}\Psi_{N-2,N-1}(x_{N-2}, x_{N-1})$ and so on.

Simlar considerations apply for x_1 towards x_2, towards x_3, etc...



$$\mathcal{P}(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} \mathcal{P}(\mathbf{x})$$

$$= \frac{1}{Z} \sum_{x_1} \Psi_{1,2}(x_1, x_2) \ldots \sum_{x_{n-1}} \Psi_{n-1,n}(x_{n-1}, x_n)$$

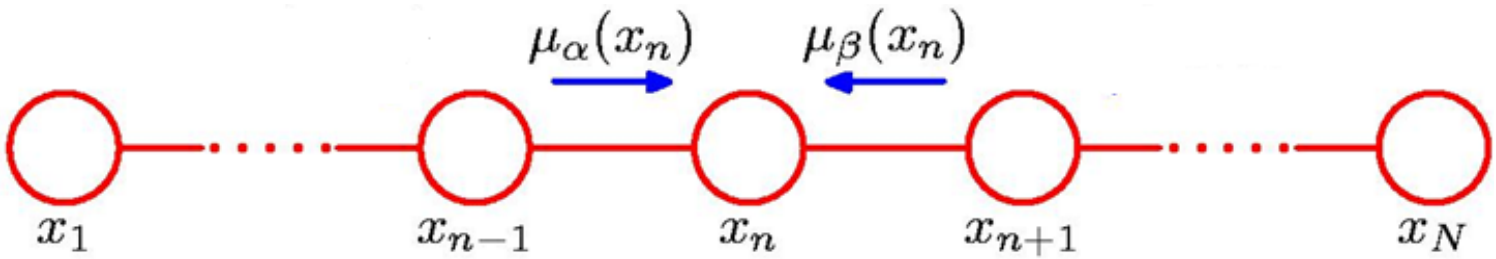
$$\sum_{x_{n+1}} \Psi_{n+1,n+2}(x_{n+1}, x_{n+2}) \ldots \sum_{x_N} \Psi_{N-1,N}(x_{N-1}, x_N)$$

Now observe that the summation over x_N involves the last term, producing a factor of x_{N-1} ($\Psi'_{N-1} = \sum_{x_N} \Psi_{N-1,N}(x_{N-1}, x_N)$) and the summation over x_{N-1} will involve only $\Psi'_{N-1}\Psi_{N-2,N-1}(x_{N-2}, x_{N-1})$ and so on.

Simlar considerations apply for x_1 towards x_2, towards x_3, etc...

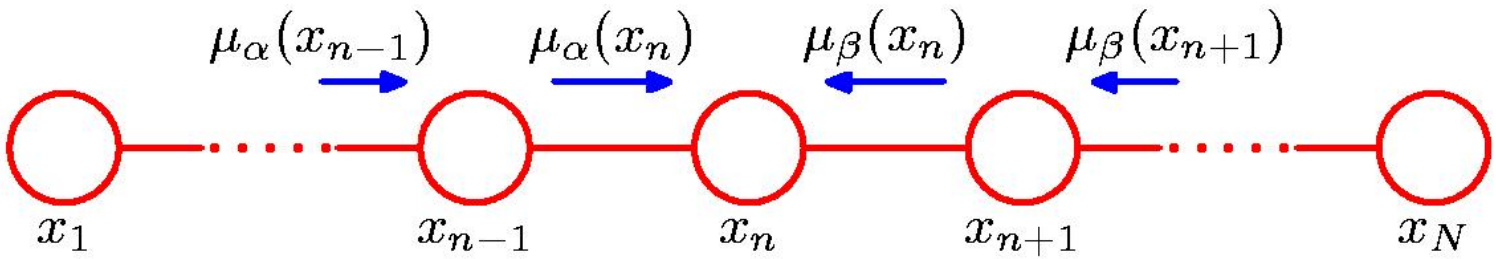
$$\Psi'_{2,3} = [\sum_{x_1} \Psi_{1,2}(x_1, x_2)]\Psi_{2,3}(x_2, x_3)$$
$$\Psi'_{N-2,N-1} = [\sum_{x_N} \Psi_{N-1,N}(x_{n-1}, x_N)]\Psi_{N-2,N-1}(x_{N-2}, x_{N-1})$$

- Every summation eliminates a variable
- Variable elimination occurs from left to right from x_1 towards x_n, and from right to left from x_N towards x_n.
- It is like passing messages containing the summation factor to the neighbor node which then perform the multiplication with the factor shared with the other neighbor



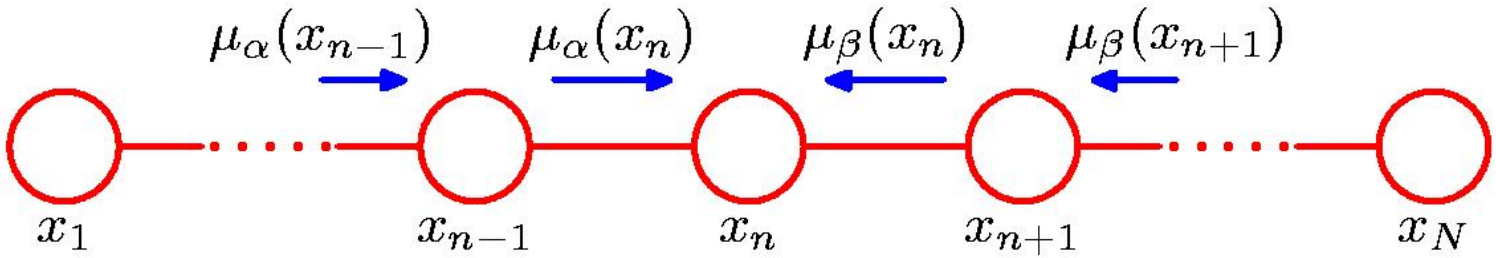
$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1},x_n) \cdots \left[\sum_{x_1} \psi_{1,2}(x_1,x_2)\right] \cdots\right]}_{\mu_\alpha(x_n)}$$

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



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

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



$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \qquad \mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

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

To compute local marginals:

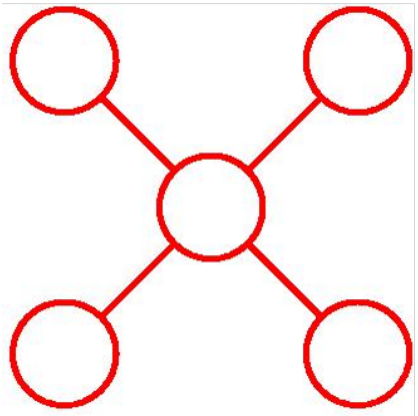
- compute and store all forward messages $\mu_\alpha(x_n)$.
- compute and store all backward messages $\mu_\beta(x_n)$.
- compute Z at any node x_n.
- compute

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

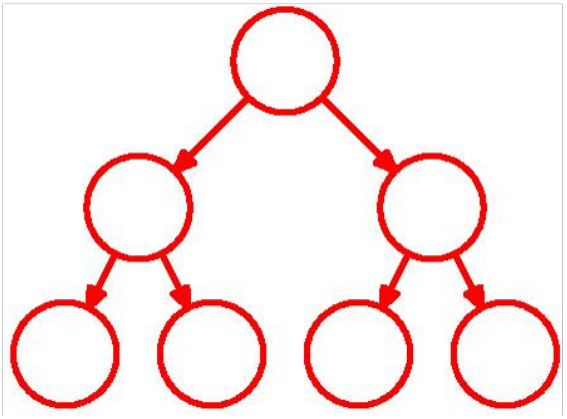
  for all variables required
- If some variables are observed (conditional queries): evidence variables are *clamped* to their observed values, thus disappearing from the corresponding factors (no need for summation)

Inference on Trees and Polytrees

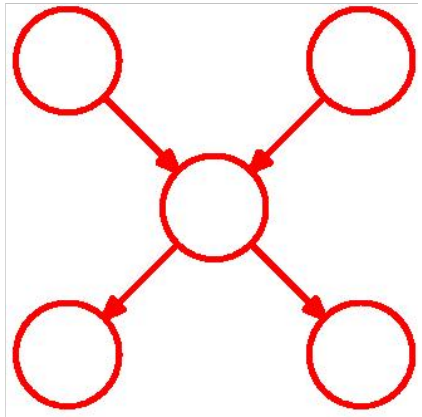
Undirected Tree



Directed Tree

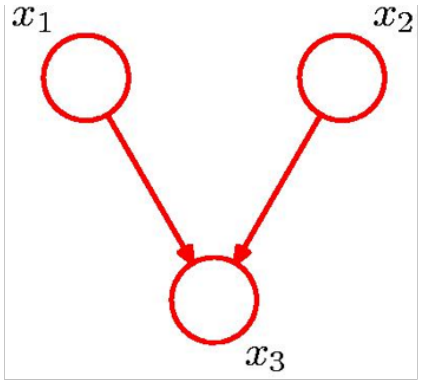


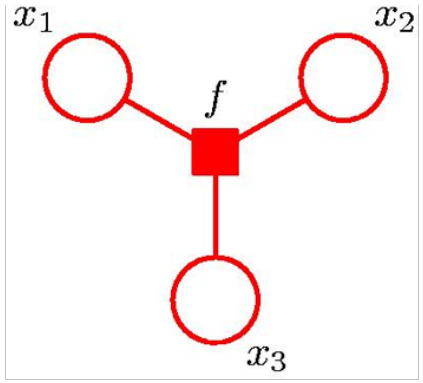
Polytree

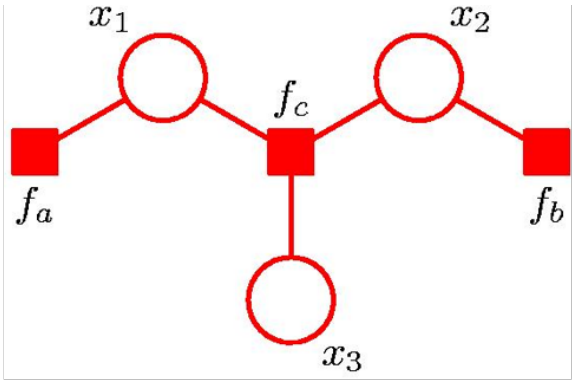


- Undirected tree: only one path between any pair of nodes (no loops)
- Directed tree: only one node with no parents (the root) and other nodes having exactly one parent; the moralized undirected graph is the underlying undirected graph.
- Directed polytree: the underlying undirected graph is a tree (more than one root); the moralized undirected graph has loops.

Factor Graphs from a DAG







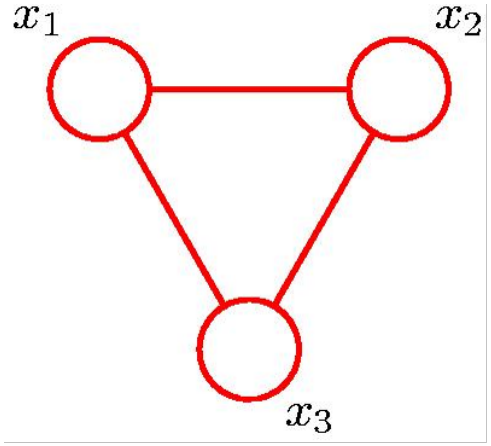
$$p(\mathbf{x}) = p(x_1)p(x_2)\, p(x_3|x_1,x_2) \qquad f(x_1,x_2,x_3) = p(x_1)p(x_2)p(_3|x_1,x_2)$$

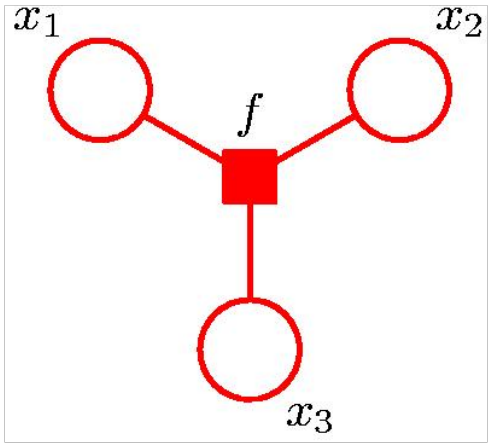
$$f_a(x_1) = p(x_1)$$

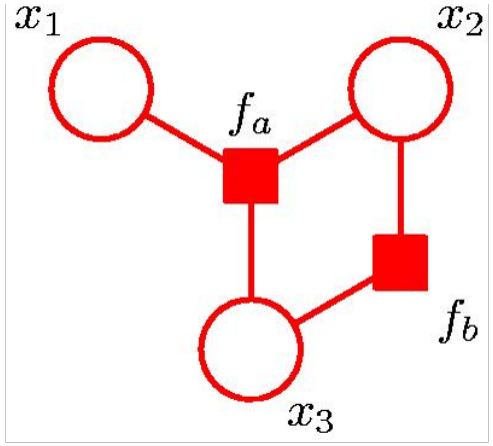
$$f_b(x_2) = p(x_2)$$

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

Factor Graphs from an UGM

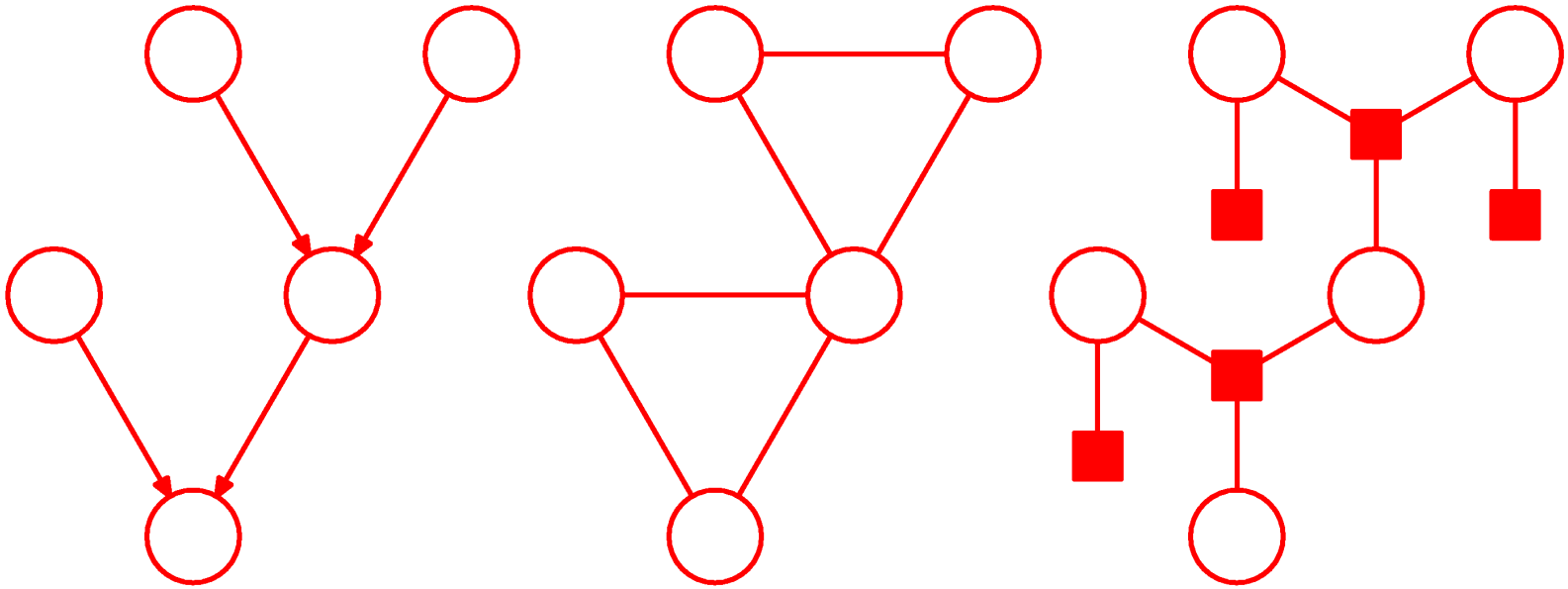






$$\psi(x_1, x_2, x_3) \qquad f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3) \qquad f_a(x_1, x_2, x_3) f_b(x_2, x_3) = \psi(x_1, x_2, x_3)$$

If we create a FG from a tree (either directed or indirected), the FG is a tree as well



If a DAG is a polytree, the corresponding moral graph has loops, but the FG is still a tree

Sum-Product Algorithm

Sum-Product Algorithm

- Objective:
  - to obtain an efficient, exact inference algorithm for finding marginals;
  - in situations where several marginals are required, to allow computations to be shared efficiently.
- Assumptions:
  - the original graph is an undirected tree or a directed tree or a polytree
  - we convert the orginal graph into a FG, so that we can deal with both directed and undirected models using the same framework

Remember

$$\mathcal{P}(\mathbf{x}) = \prod_{s=1}^{k} f_s(\mathbf{x_s})$$

where k is the number of factors and $\mathbf{x_s}$ is the subset of variables in the scope of factor f_s.

Sum-Product Algorithm

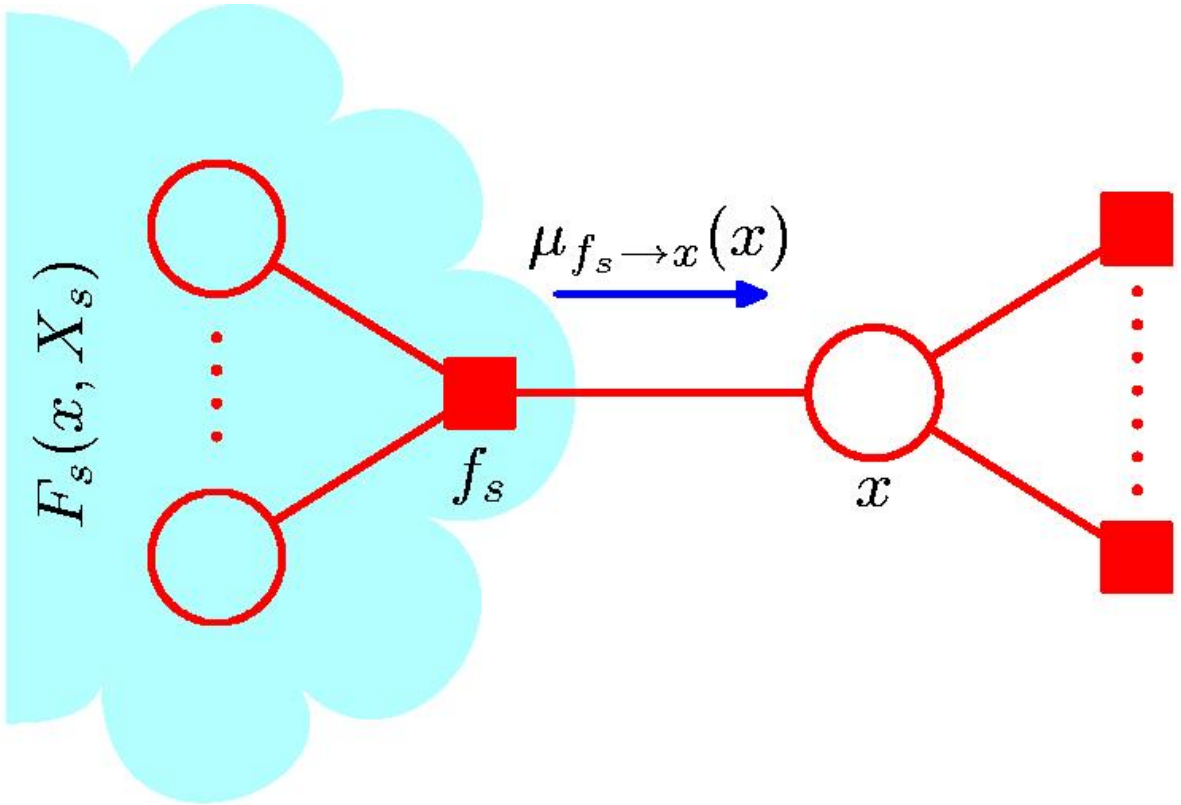
- Objective:
  - to obtain an efficient, exact inference algorithm for finding marginals;
  - in situations where several marginals are required, to allow computations to be shared efficiently.
- Assumptions:
  - the original graph is an undirected tree or a directed tree or a polytree
  - we convert the orginal graph into a FG, so that we can deal with both directed and undirected models using the same framework

Remember

$$\mathcal{P}(\mathbf{x}) = \prod_{s=1}^{k} f_s(\mathbf{x_s})$$

where k is the number of factors and $\mathbf{x_s}$ is the subset of variables in the scope of factor f_s.

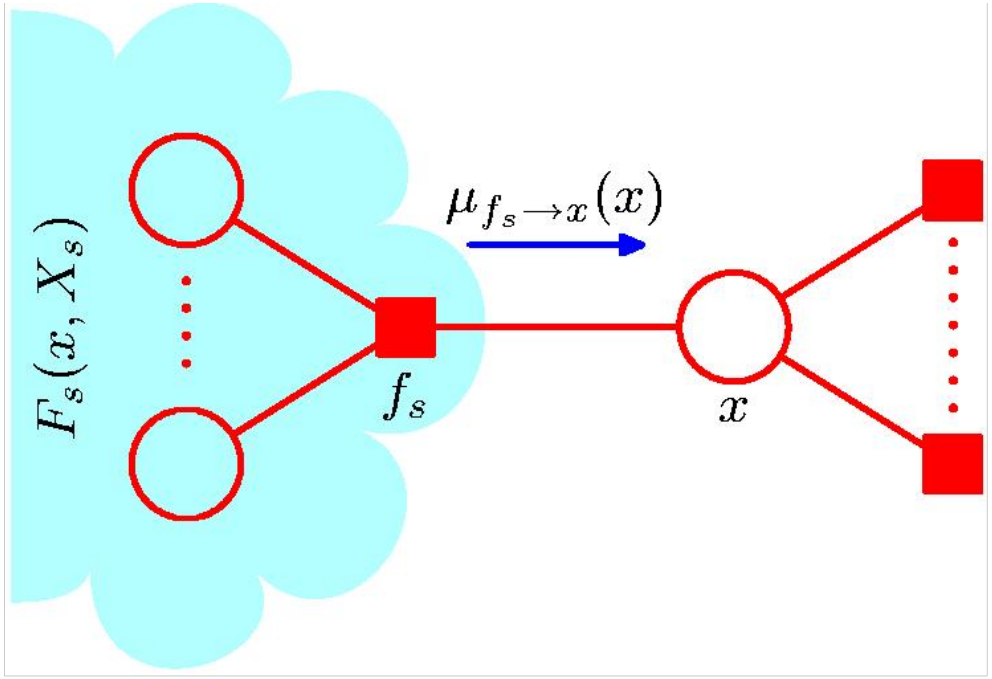
- Consider the computation of a single marginal: the tree structure of the FG allows us to partition the factors in the joint into groups with one group associated with each of the factor nodes that is a neighbor of the variable node x;
- let $ne(x)$ denotes the set of factor nodes that are neighbors of x;
- let X_s denotes the set of all variables in the subtree connected to the variable node x via the factor node f_s;
- let $F_s(x, X_s)$ represents the product of all the factors in the group associated with factor f_s.



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

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

From factor f_s, node x can receive the marginalization of the factor product wrt X_s ($\mu_{f_s \to x}(x)$).



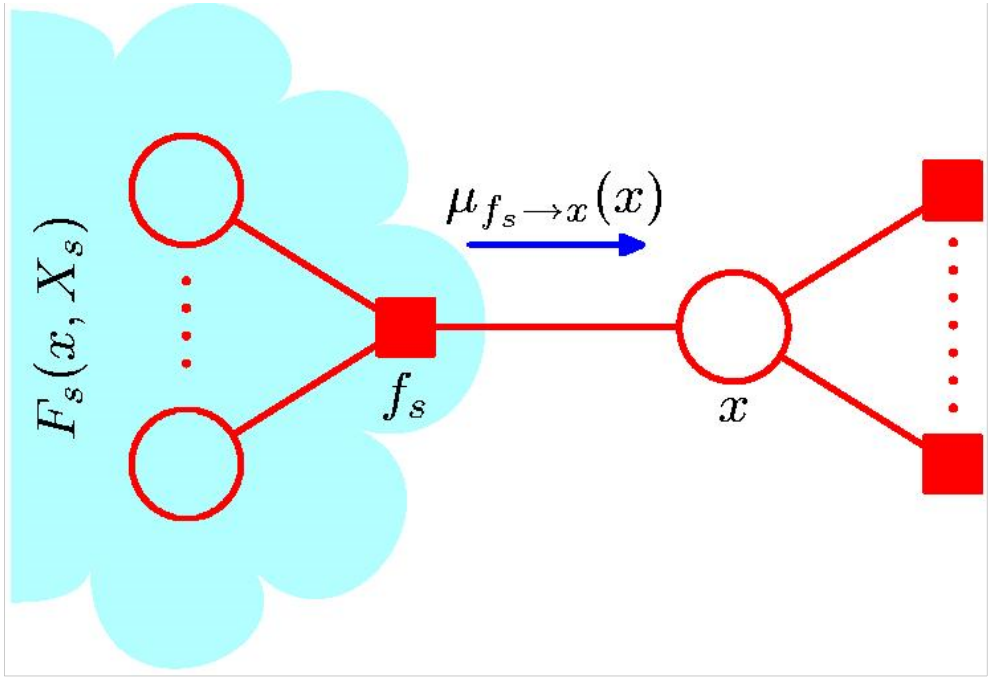
$$p(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right]$$

$$= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x).$$

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

Marginal of x is the product of incoming messages

From factor f_s, node x can receive the marginalization of the factor product wrt X_s ($\mu_{f_s \to x}(x)$).



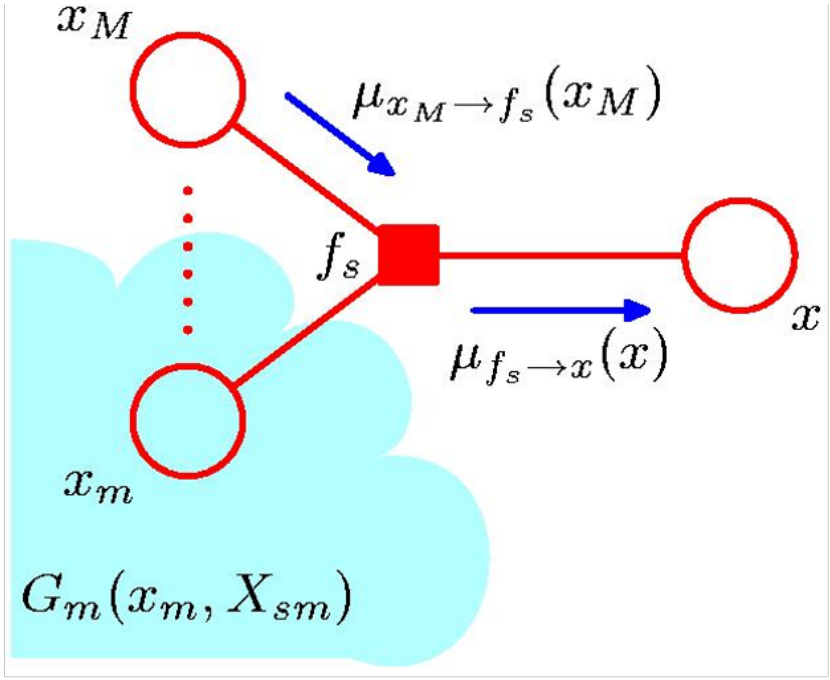
$$p(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right]$$

$$= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x).$$

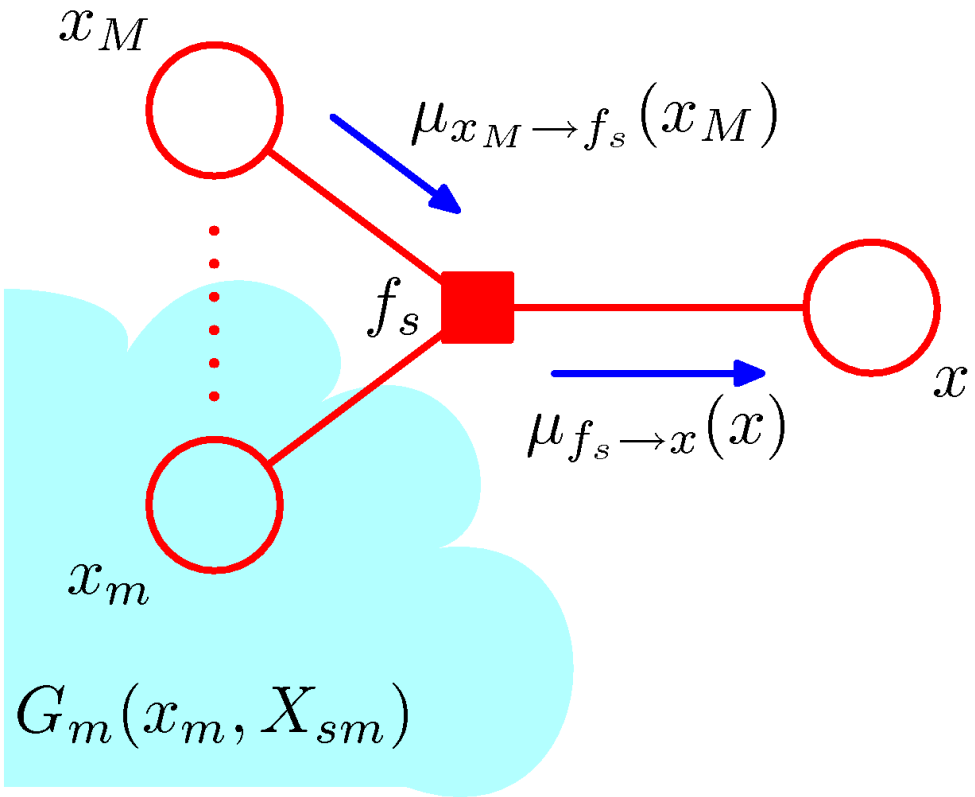
$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

Marginal of x is the product of incoming messages

Each factor $F_s(x, X_s)$, can in turns be factorized into different sub-factors, one for each variable in the scope of f_s other than x (let call them $x_1, \ldots x_M$), plus the factor $f_s(x, x_1, \ldots x_M)$

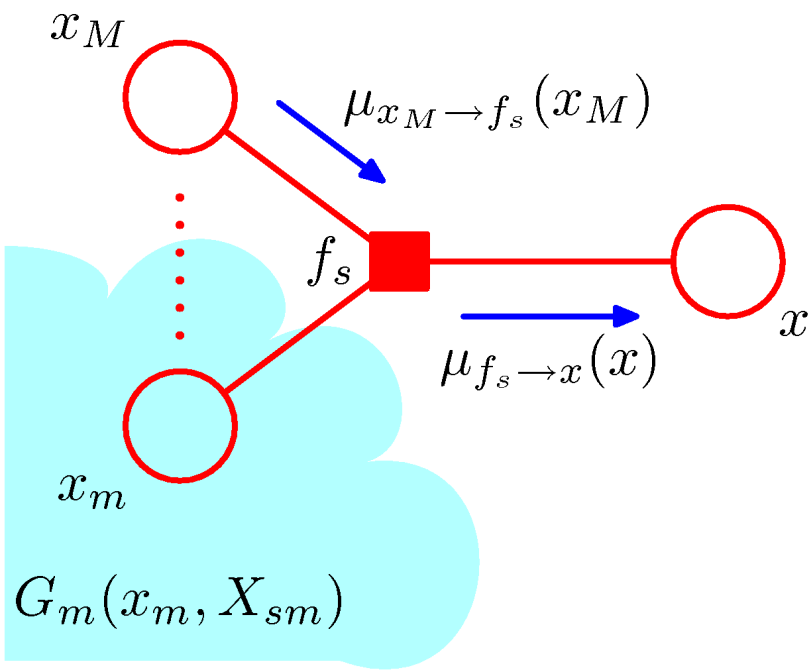


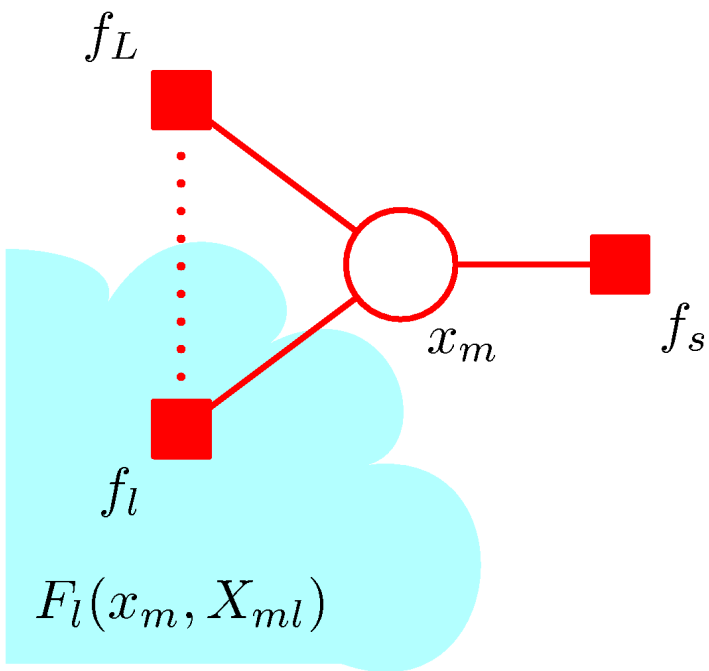
$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \ldots G_M(x_M, X_{sM})$$



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

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



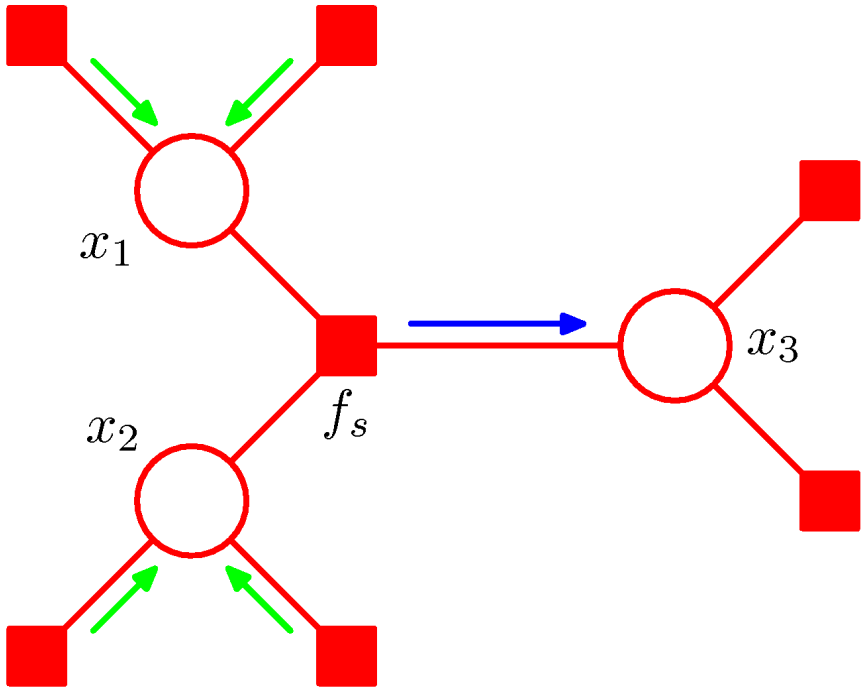


$$\mu_{x_m \to f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm}) = \sum_{X_{sm}} \prod_{l \in \text{ne}(x_m) \setminus f_s} F_l(x_m, X_{ml})$$

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

Initialization

- If the leaf is a variable node, since the sent message must be the product of incoming messages (see previous slide), then the sent message is 1;
- if the leaf is a factor node, then the sent message is the factor itself (see two previous slides)



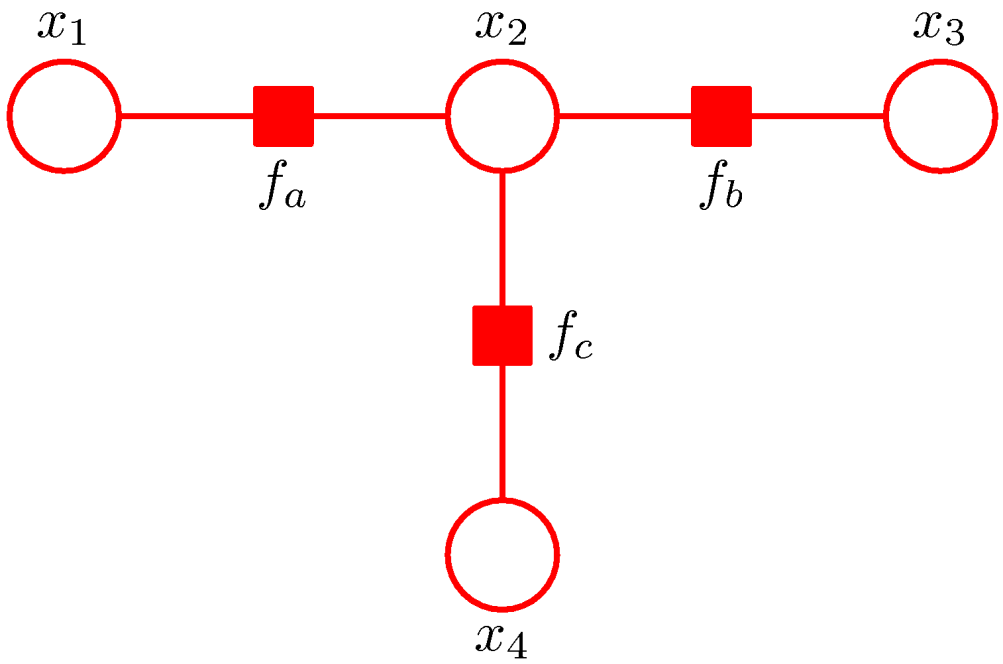
The outgoing message shown by the blue arrow is obtained by taking the product of all the incoming messages shown by green arrows, multiplying by the factor f_s, and marginalizing over the variables x_1 and x_2.

Sum-Product Algorithm: Marginals

To compute local marginarls:

- Pick an arbitrary node as *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

Example

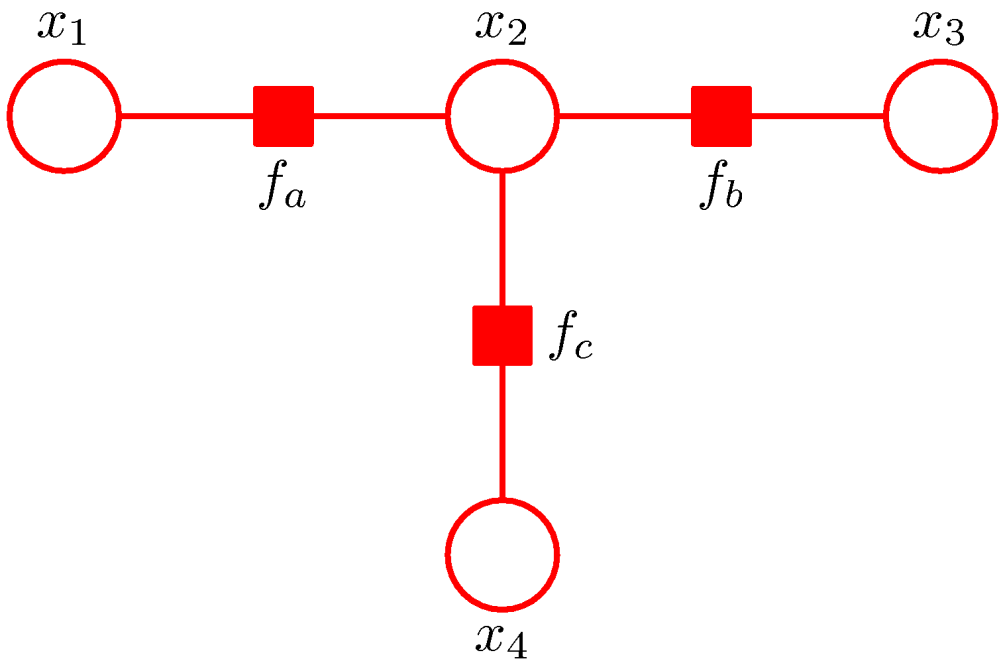


$$\tilde{\mathcal{P}}(x_1, x_2, x_3) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

($\tilde{\mathcal{P}}$ unormalized distribution)

Consider x_3 as root node ...

Example

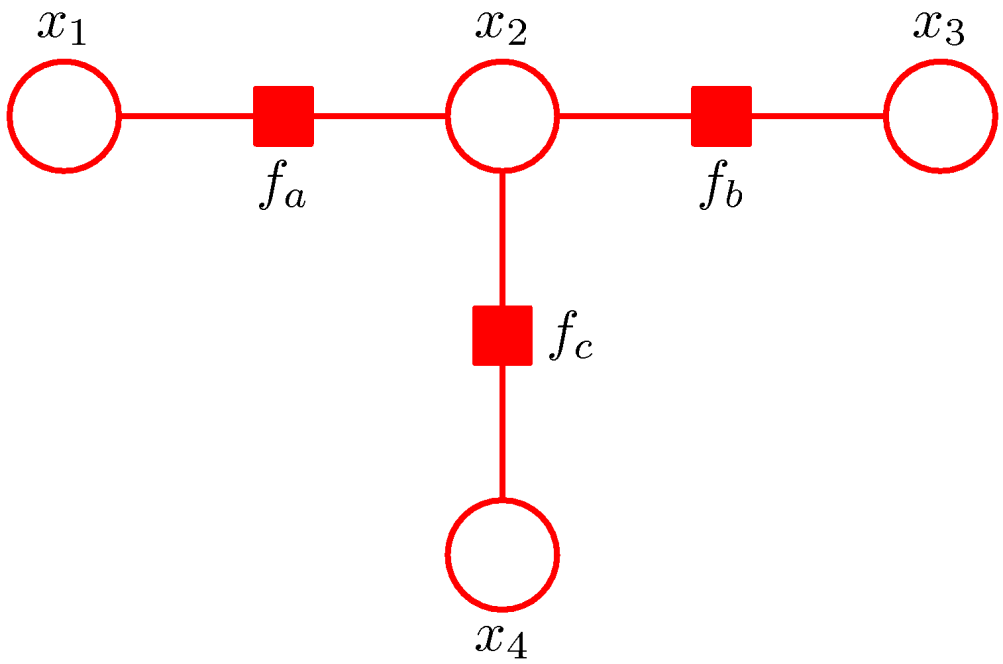


$$\tilde{\mathcal{P}}(x_1, x_2, x_3) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

($\tilde{\mathcal{P}}$ unormalized distribution)

Consider x_3 as root node ...

Example

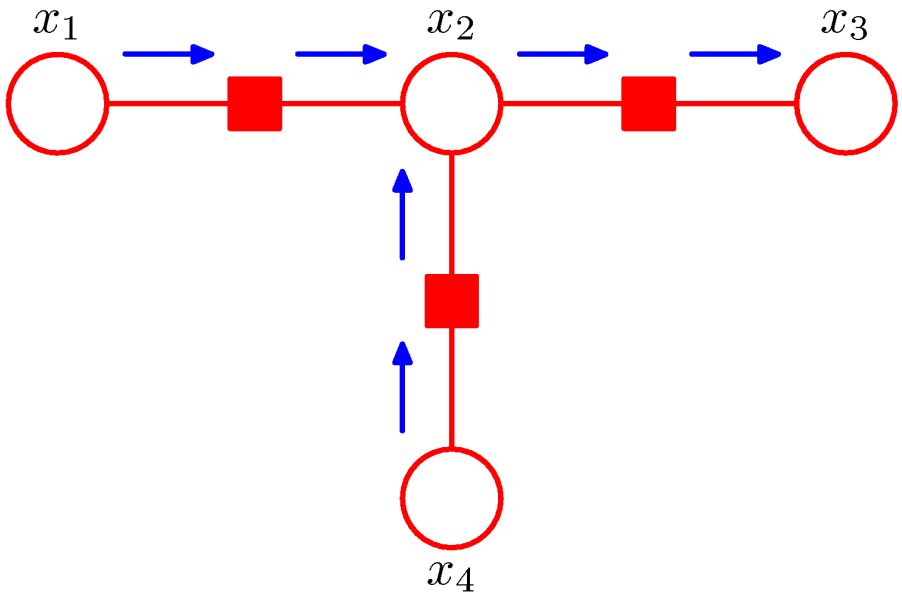


$$\tilde{\mathcal{P}}(x_1, x_2, x_3) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

($\tilde{\mathcal{P}}$ unormalized distribution)

Consider x_3 as root node ...

Example



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

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

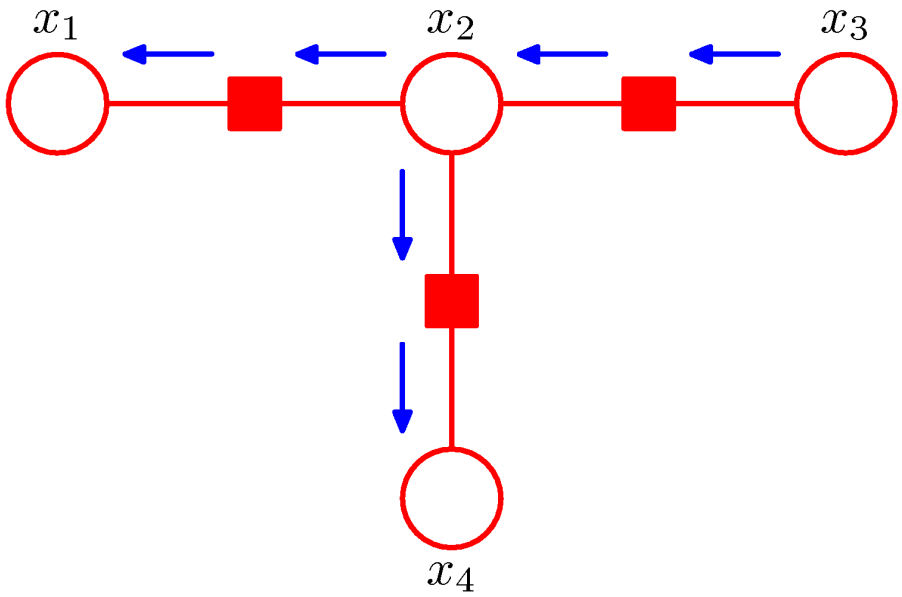
$$\mu_{x_4 \to f_c}(x_4) = 1$$

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

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

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

Example



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

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

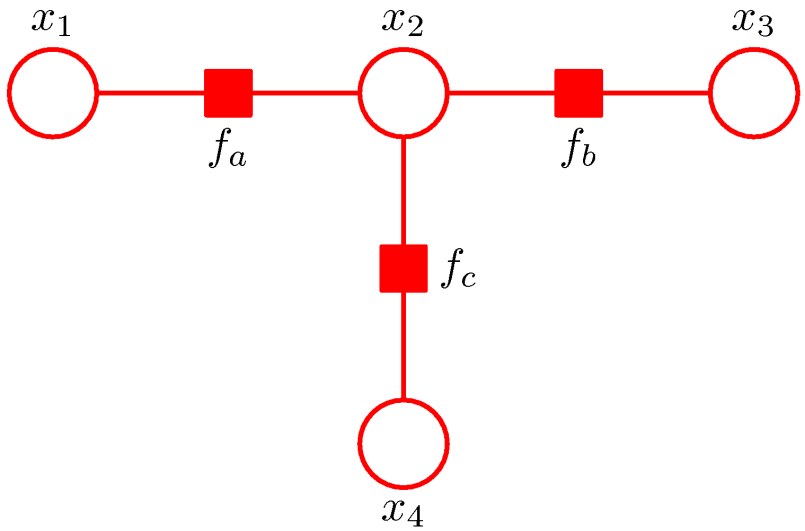
$$\mu_{x_2 \to f_a}(x_2) = \mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2)$$

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

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

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

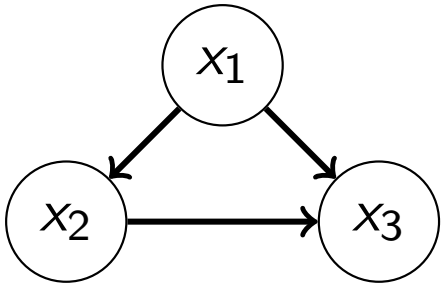
Example

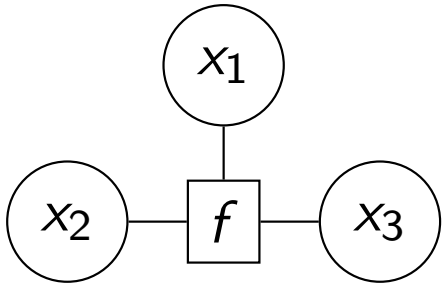


$$\begin{aligned}
\widetilde{p}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
&= \left[\sum_{x_1} f_a(x_1, x_2)\right]\left[\sum_{x_3} f_b(x_2, x_3)\right] \\
&\quad \left[\sum_{x_4} f_c(x_2, x_4)\right] \\
&= \sum_{x_1}\sum_{x_3}\sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\
&= \sum_{x_1}\sum_{x_3}\sum_{x_4} \widetilde{p}(\mathbf{x})
\end{aligned}$$

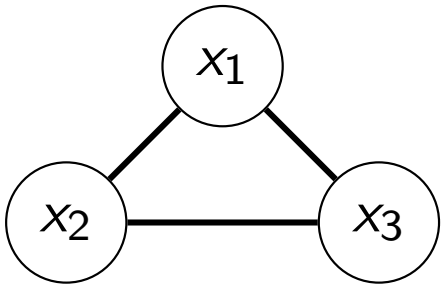
In conclusion:

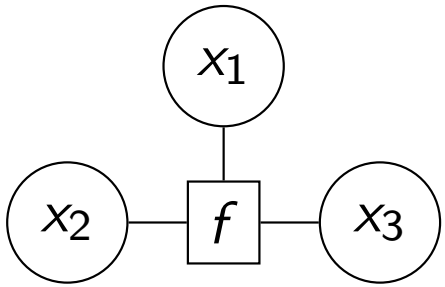
- if the original graph produces a tree FG, then we can use the Sum-Product message passing algorithm;
- this certainly occurs when the original graph is an undirected tree, a directed tree or a polytree
- it may happen that the FG is a tree even if the original graph is not a tree or a polytree (see figure)
- visiting twice all the nodes in the graph we compute every marginal
- we compute posterior probabilities over each variable given the evidence, by just reducing the factors where a variable is observed (technically it corresponds to multiply the factor for the *Kronecker delta function* of each variable and the observed ones)





$$f(x_1, x_2, x_3) = \mathcal{P}(x_1)\mathcal{P}(x_2|x_1)\mathcal{P}(x_3|x_1, x_2)$$





$$f(x_1, x_2, x_3) = \Psi(x_1, x_2, x_3)$$

Kronecker Delta Function

$$\delta(x_i, x_j) = \begin{cases} 0 & \text{if } x_i \neq x_j \\ 1 & \text{if } x_i = x_j \end{cases}$$

Suppose there is a factor $f(x_1, x_2, x_3)$ and suppose x_2 is observed to value $\mathbf{x_2}$;

then we modify the factor f to
$f'(x_1, x_2, x_3) = f(x_1, x_2, x_3)\delta(x_2, \mathbf{x_2})$

now, since x_2 is observed it is eliminated by summation producing the new factor $f''(x_1, x_3) = \sum_{x_2} f'(x_1, x_2, x_3)$

it corresponds to eliminating from f all the entry with $x_2 \neq \mathbf{x_2}$ and to consider only x_1, x_3

Example of evidence: $x_2 = 0$

x_1, x_2, x_3	$f(x_1, x_2, x_3)$
0 0 0	10
0 0 1	10
0 1 0	5
0 1 1	5
1 0 0	100
1 0 1	10
1 1 0	20
1 1 1	5

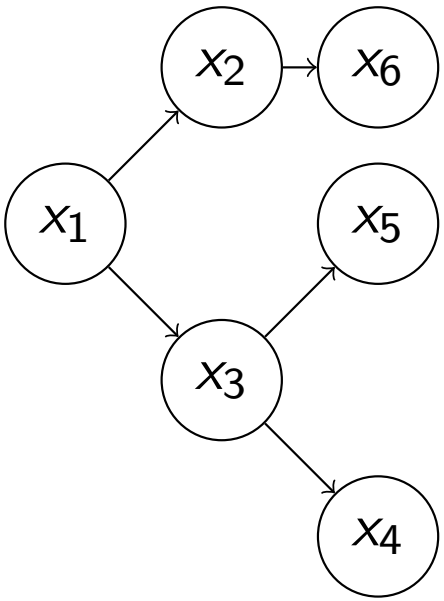
x_1, x_2, x_3	$f(x_1, x_2, x_3)\delta(x_2, 0)$
0 0 0	10
0 0 1	10
0 1 0	**0**
0 1 1	**0**
1 0 0	100
1 0 1	10
1 1 0	**0**
1 1 1	**0**

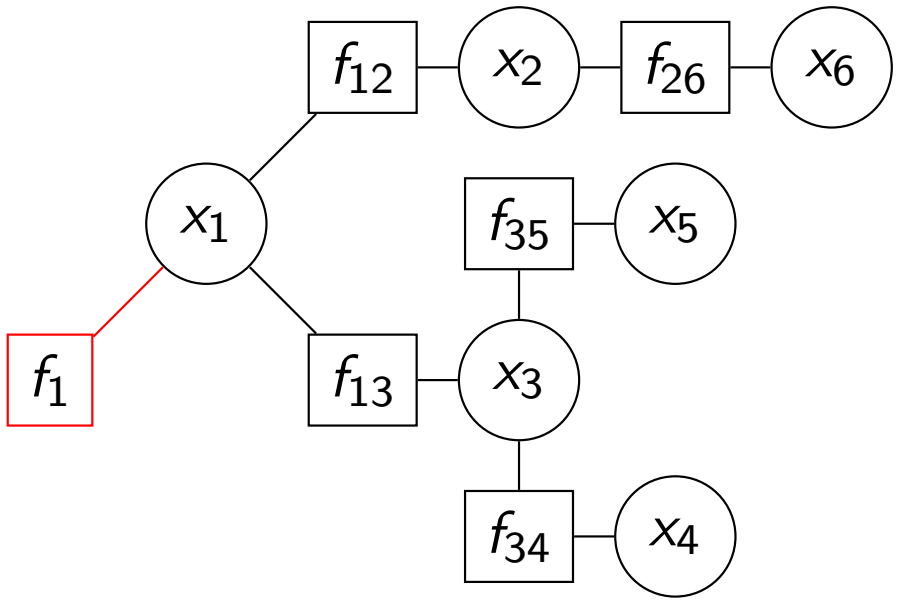
x_1, x_3	$\sum_{x_2} f(x_1, x_2, x_3)\delta(x_2, 0)$
0 0	10
0 1	10
1 0	100
1 1	10

Inference on Graphs

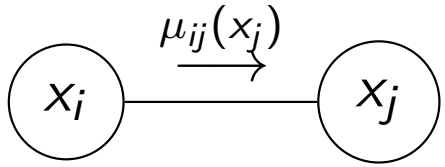
A digression on Belief Propagation on Trees

- In case of trees, the propagation rules can be defined directly between variable nodes (with no need for factor nodes)
- In a tree, each edge between two variables corresponds to a factor $f_{ij}(x_i, x_j) = \Psi(x_i, x_j)$; exception the factor for the prior of the root node in case of a DAG





Propagation rule



$$\mu_{ij}(x_j) = \sum_{x_i}(\Psi(x_i)\Psi(x_i,x_j)\prod_{k\neq j}\mu_{ki}(x_i))$$

if x_i is the root of a DAG

$$\mu_{ij}(x_j) = \sum_{x_i}(\Psi(x_i,x_j)\prod_{k\neq j}\mu_{ki}(x_i))$$

otherwise

Thus the message sent from one node x_i to another node x_j is the marginalization wrt to the first node x_i of the product of all the messages entering x_i and sent by other nodes different than x_j, times the potential(s) involving the nodes.

Example

Cooping with loops: Idea!

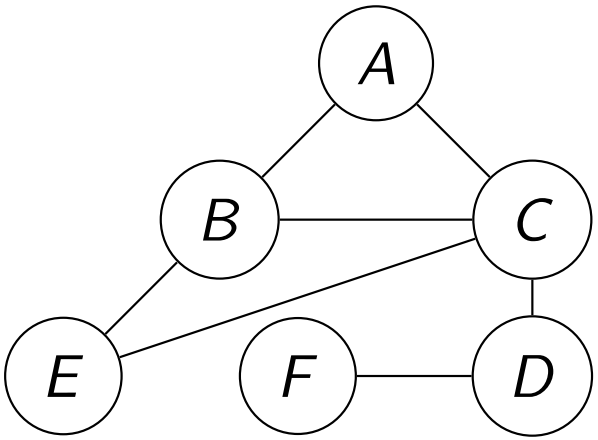
If I can transform the original (loopy) graph into a tree, then I can use the Belief Propagation (Sum Product) algorithm to perform inference.
A suitable rule should then be provided to feed back the obtained results into the original graph.

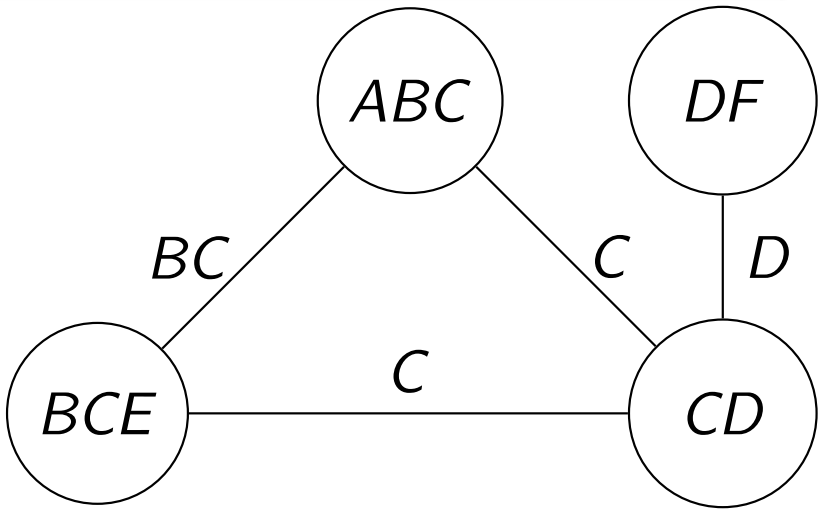
Solution: Junction Tree

Junction Tree Algorithm

Clique Graph

Given an undirected graph $G = \langle V, E \rangle$, the clique graph of G is the graph $CG = \langle C, F \rangle$ where $C \subseteq 2^V$ is the set of cliques of G and an edge $\langle c_i, c_j \rangle \in F$ if $c_i \cap c_j \neq \emptyset$; each edge is labeled by the intersection of the connected cliques and is called a *separator*.





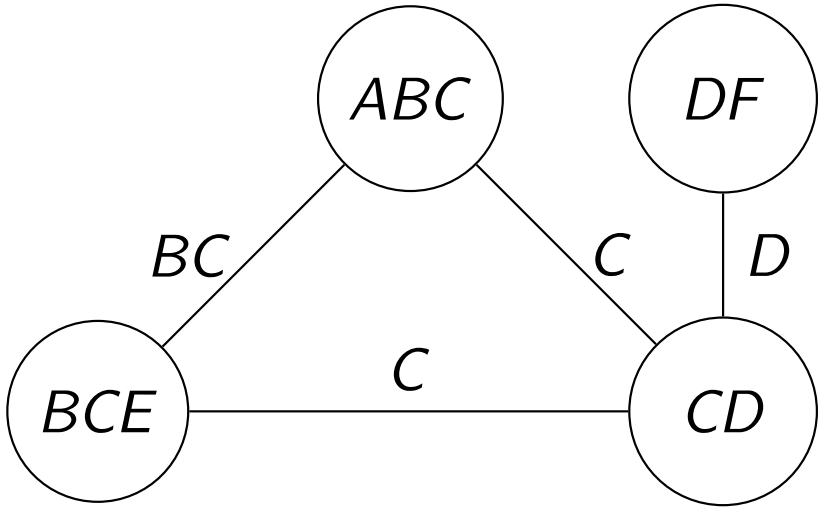
Separators between c_i and c_j are denoted as s_{ij} and can be indicated as additional "square nodes" instead of labels.

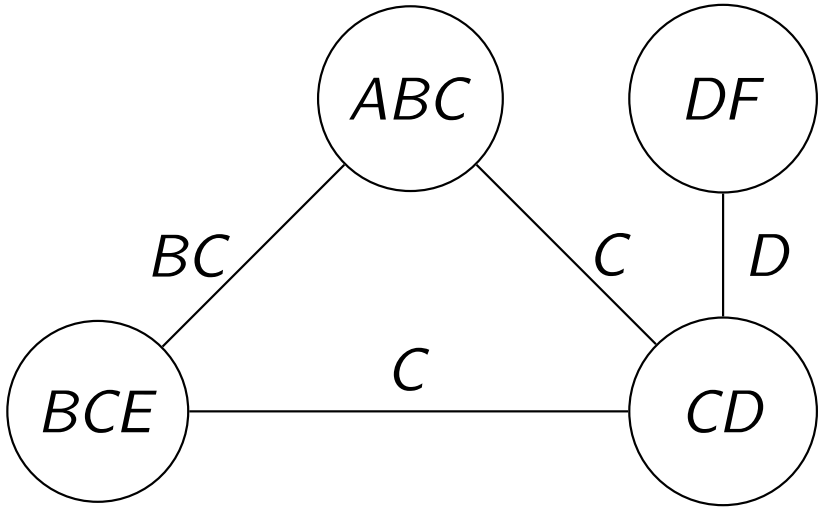
Junction (Join) Tree

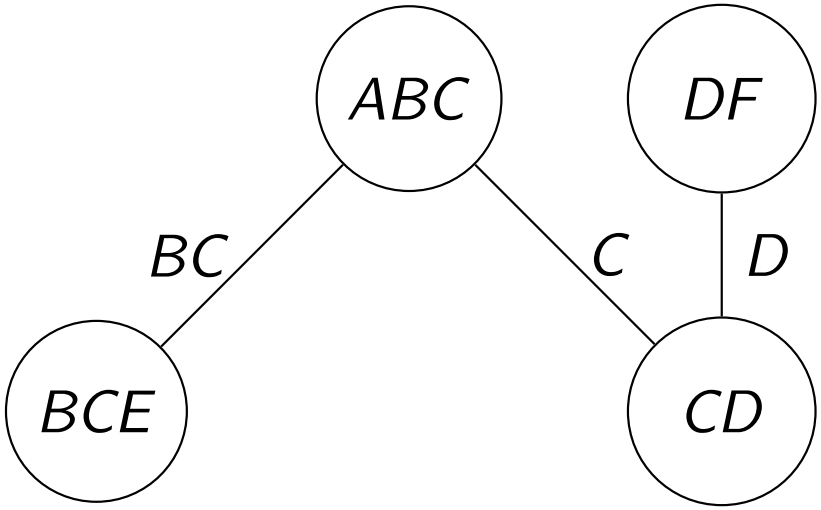
A junction or join tree of un undirected graph G, is a subtree of the clique graph of G satisfying the *running intersection property*: for each pair of cliques c_i, c_j with separator s_{ij}, then all cliques in the path between c_i and c_j contain s_{ij}.

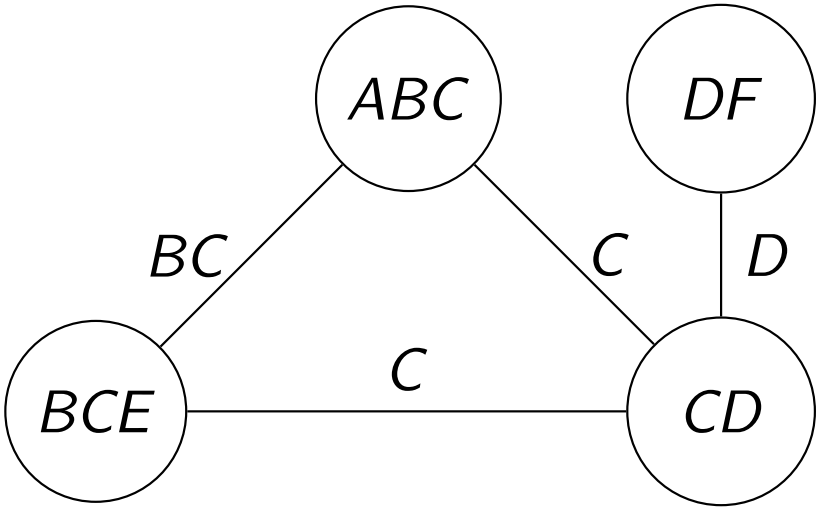
Chordal Graphs and Junction Trees

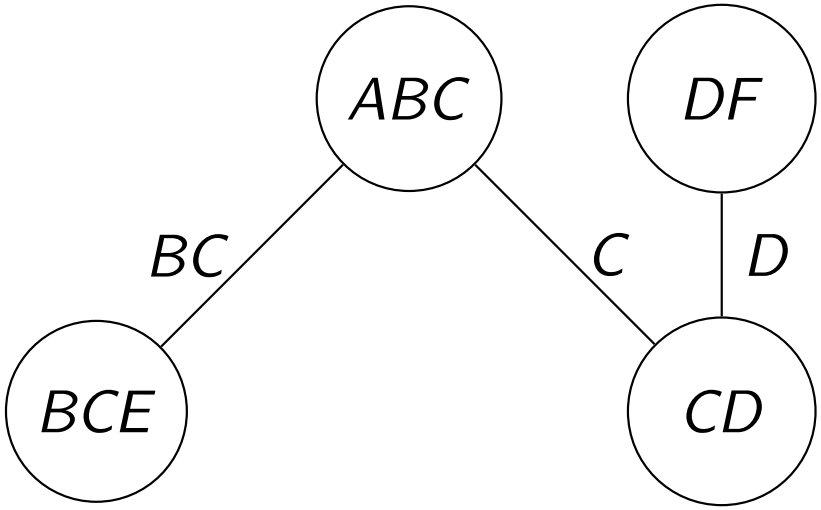
A graph is chordal if and only if it has a junction tree

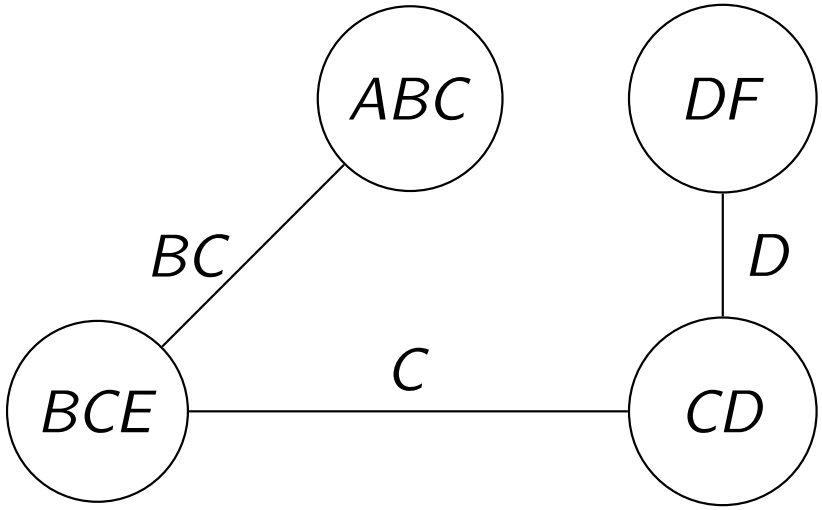




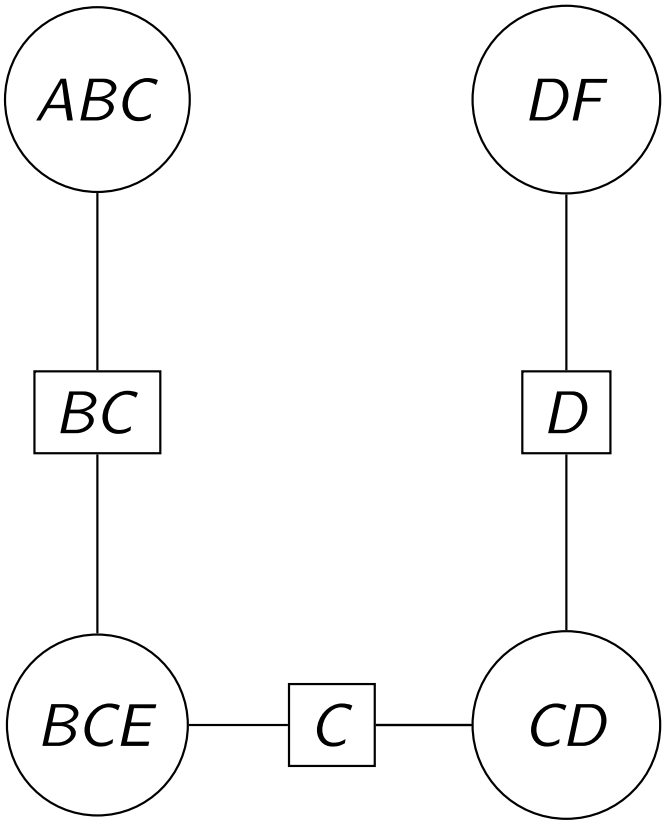








An alternative representation: explicit separator nodes



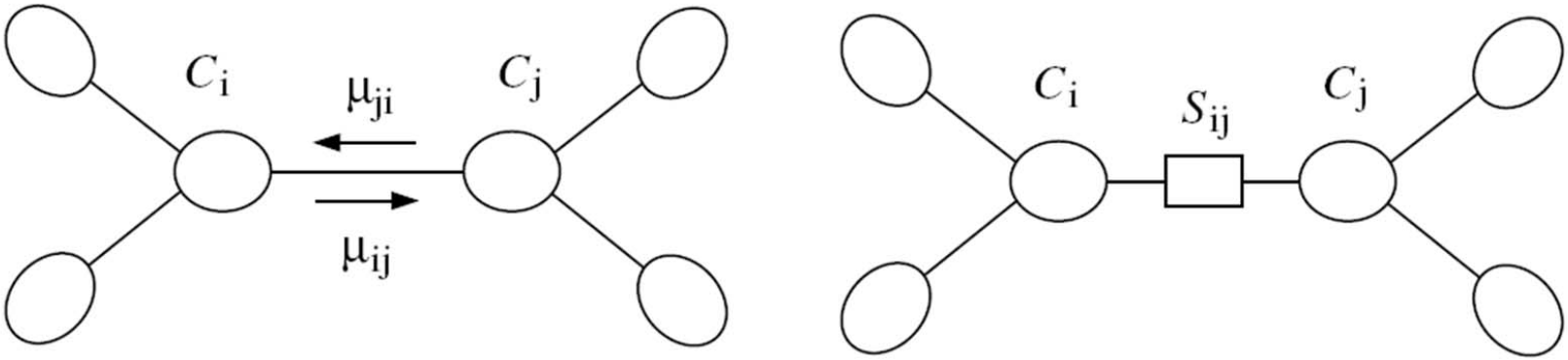
Path between ABC to CD: all cliques (and all separators) contain C

Junction Tree Algorithm

- If the graph is a DAG, then *moralize* the graph
- Make the graph chordal, i.e., triangulate the graph from previous step
- Build the *junction tree* of the chordal graph
- Run the Sum-Product algorithm on the junction tree (Shafer-Shenoy architecture)
- The updated potential of each clique will result in the joint probability of the clique's variables.

- graph triangulation is NP-hard, but there exists good heuristic allgorithms
- to build the junction tree, one can run a maximum spanning tree algorithm on the clique graph, by weighting each arc with the separator's cardinality

Shafer-Shenoy Propagation



- Message from clique i to clique j

$$\mu_{ij} = \sum_{c_i \setminus s_{ij}} \Psi_{c_i} \prod_{k \neq j} \mu_{ki}$$

- Clique Marginal

$$\mathcal{P}(c_i) \propto \Psi_{c_i} \prod_{k} \mu_{ki}$$

Example

An alternative propagation rule: the Hugin architecture

- An alternative to sum-product algorithm exploits the following equation; let C be the set of cliques and S the set of separators, then

$$\mathcal{P}(X) = \frac{\prod_{c \in C} \Psi_c}{\prod_{s \in S} \Psi_s}$$

- The message sent from node c_i to node c_j through separator s_{ij} is the following

$$\mu_{ij} = \frac{\sum_{c_i \setminus s_{ij}} \Psi_{c_i}}{\Psi_{s_{ij}}}$$

- The clique potential of c_j is then updated as $\Psi_{c_j} \leftarrow \Psi_{c_j}\, \mu_{ij}$ while the separator potential is updated as $\Psi_{s_{ij}} = \sum_{c_i \setminus s_{ij}} \Psi_{c_i}$

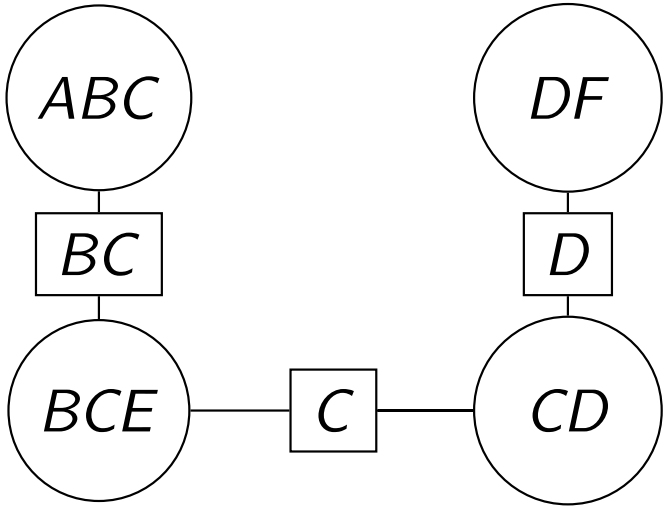
Considerations

- treewidth of a graph: is the size of the largest clique in the junction tree
- the complexity of the algorithm is exponential in the treewidth of the graph
- the triangulation ordering affects the efficiency (as the elimination order of the variables in the joint)
- a proxy for the treewidth is the number of parents in each node
- when the treewidth is too large, approximate algorithms are the only solutions

Considerations (cont.)

- To deal with evidence, clamp clique potentials to the observed values
- To get marginal probability of a variable x, marginalize the potential of any clique containing x
- To get the joint probability of a set of variables contained in a given clique, then marginalize the clique potential over such a set
- To get the joint probability of a set of variables not contained in any clique:
  - $\mathcal{P}(x_1, \ldots x_k|\mathbf{e}) =$
    $\mathcal{P}(x_1|\mathbf{e})\mathcal{P}(x_2|x_1, \mathbf{e}) \ldots \mathcal{P}(x_k|x_1, \ldots x_{k-1}, \mathbf{e})$ (inefficient)
  - perform variable elimination on the calibrated junction tree (i.e. after propagation)

Example: joint probability computation on a JT



$\mathcal{P}(A, D|\mathbf{e}) = \sum_{B,C} \Psi_1^*(ABC) \sum_E \Psi_2^*(BCE)\Psi_3^*(CD)$

where Ψ_i^* is the calibrated (i.e., after propagation of evidence $\mathbf{e}$) potential of clique c_i.