Probabilistic Graphical Models
(continued)

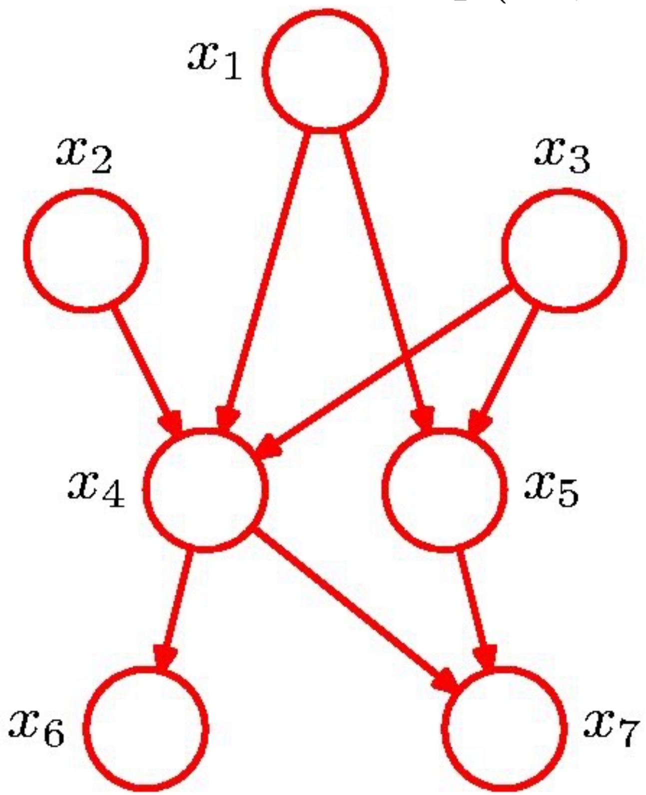
Xin Zhang
Peking University

Adapted from the slides of "Pattern Recognition and Machine Learning" Chapter 8

Recap: Bayesian Networks

- Directed Acyclic Graph (DAG)

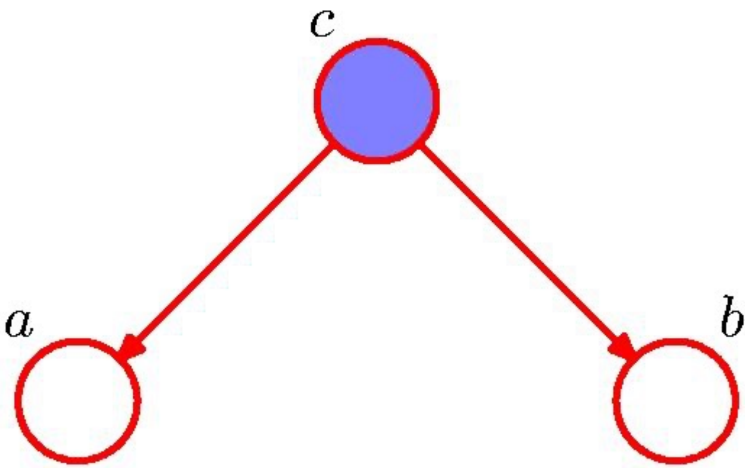
$$p(x_1, \ldots, x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3) \\ p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5)$$



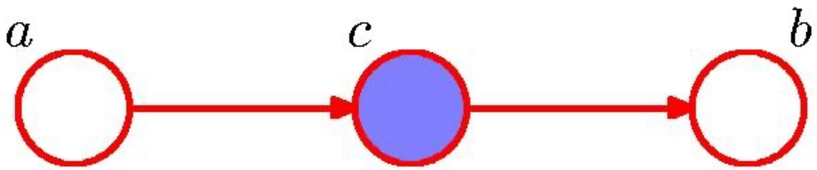
General Factorization

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$

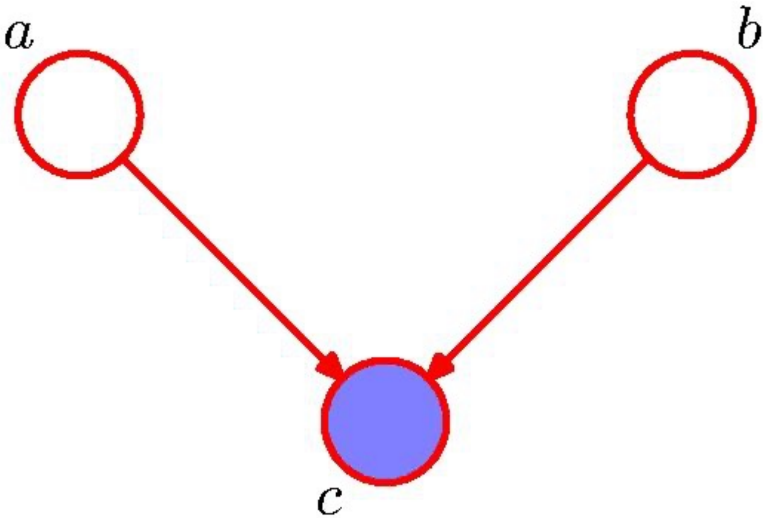
Recap: Conditional Independence



$$a \perp\!\!\!\perp b \mid c$$



$$a \perp\!\!\!\perp b \mid c$$



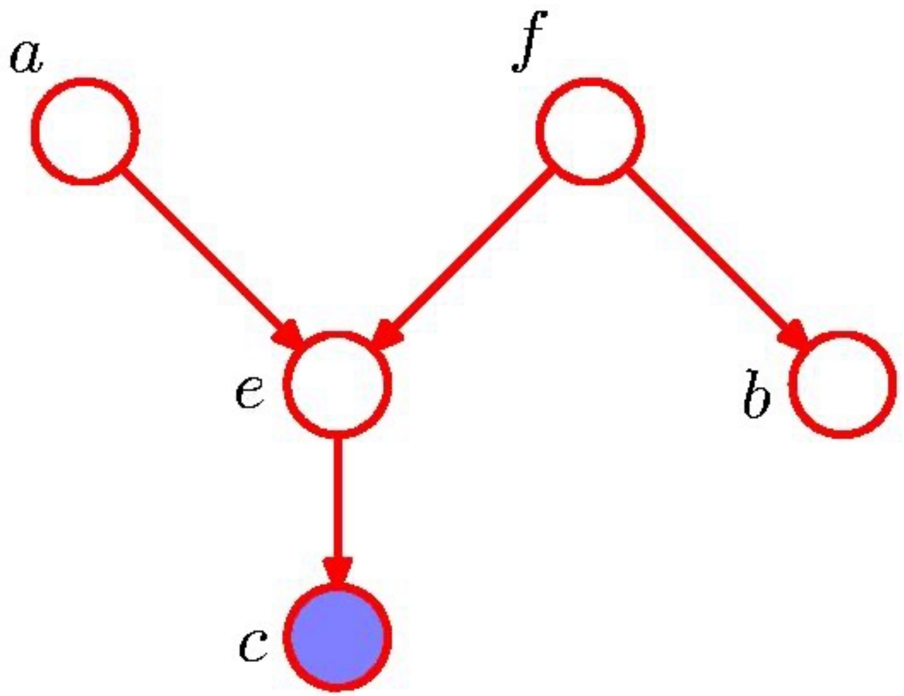
$$a \not\!\perp\!\!\!\perp b \mid c$$

Shaded nodes are observed.

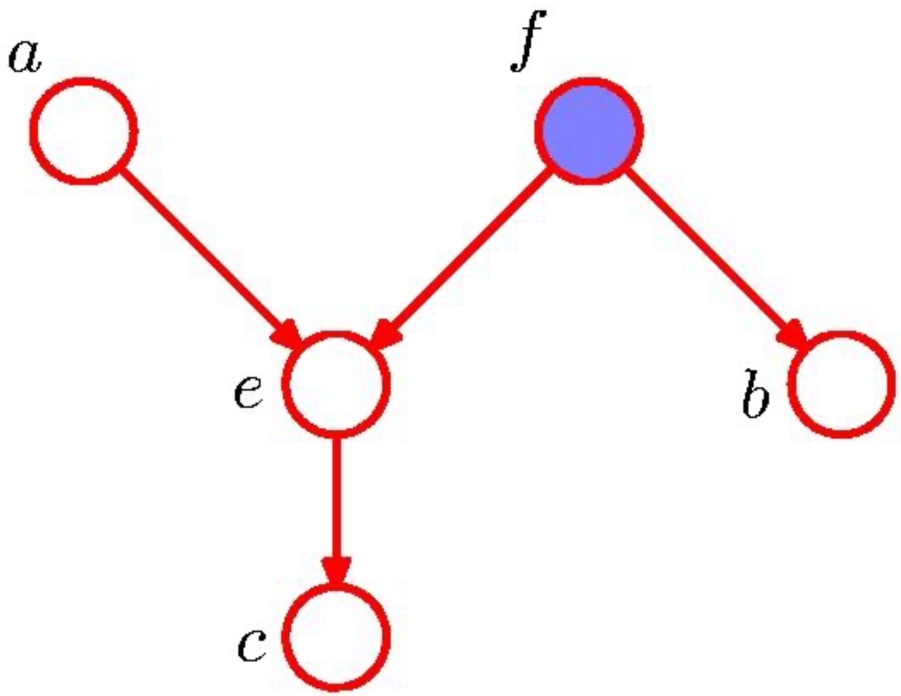
Recap: D-Separation

- A, B, and C are non-intersecting subsets of nodes in a directed graph.
- A path from A to B is blocked if it contains a node such that either
  a) the arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set C, or
  b) the arrows meet head-to-head at the node, and neither the node, nor any of its descendants, are in the set C.
- If all paths from A to B are blocked, A is said to be d-separated from B by C.
- If A is d-separated from B by C, the joint distribution over all variables in the graph satisfies .

D-separation: Example



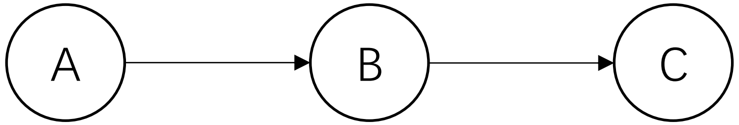
$$a \not\perp\!\!\!\perp b \mid c$$



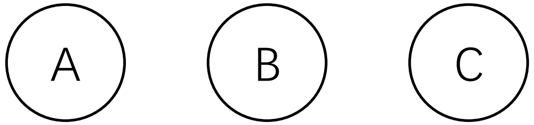
$$a \perp\!\!\!\perp b \mid f$$

Which graph(s) can describe the following distribution?

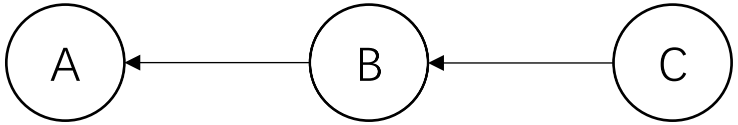
- $A \sim N(0, 1)$, $B \sim N(A, 1)$, $C \sim N(B, 1)$



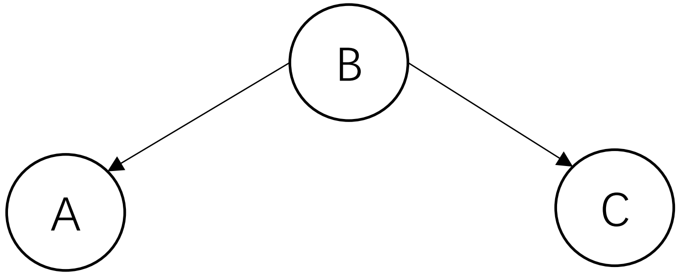
1



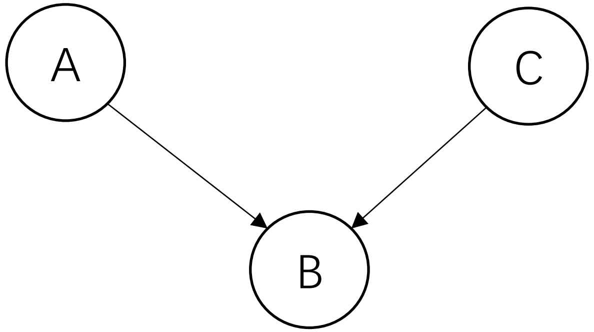
2



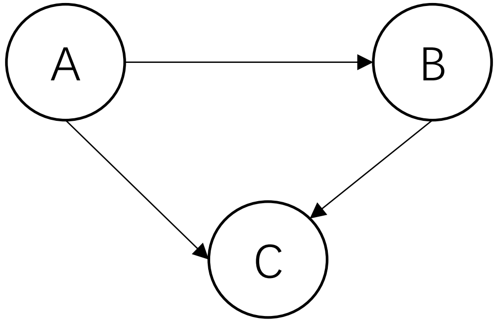
3



4

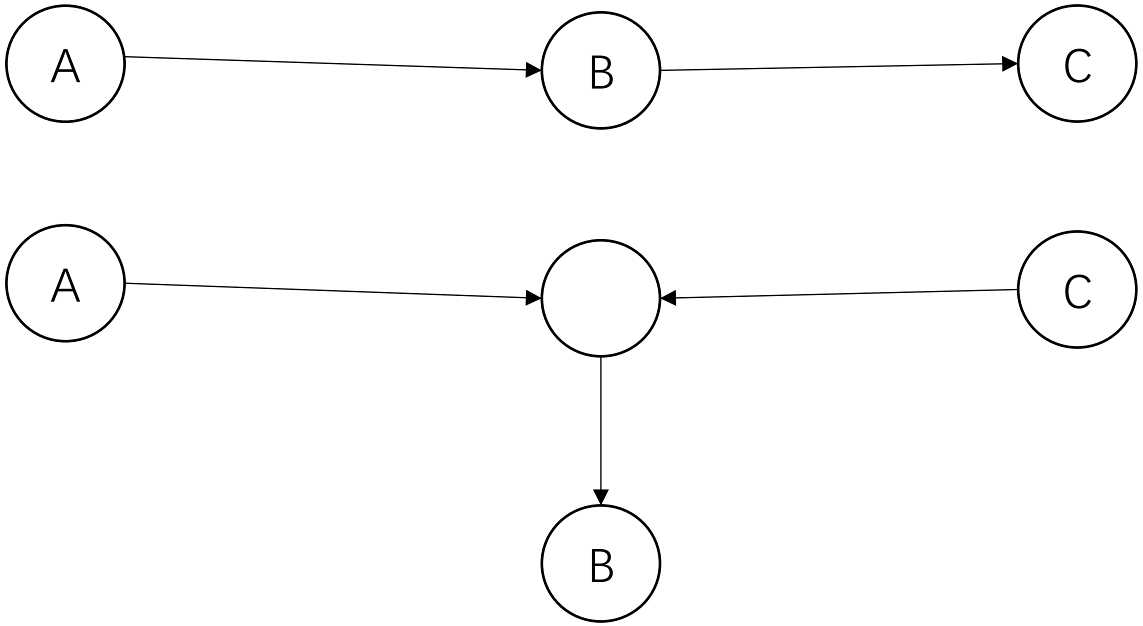


5

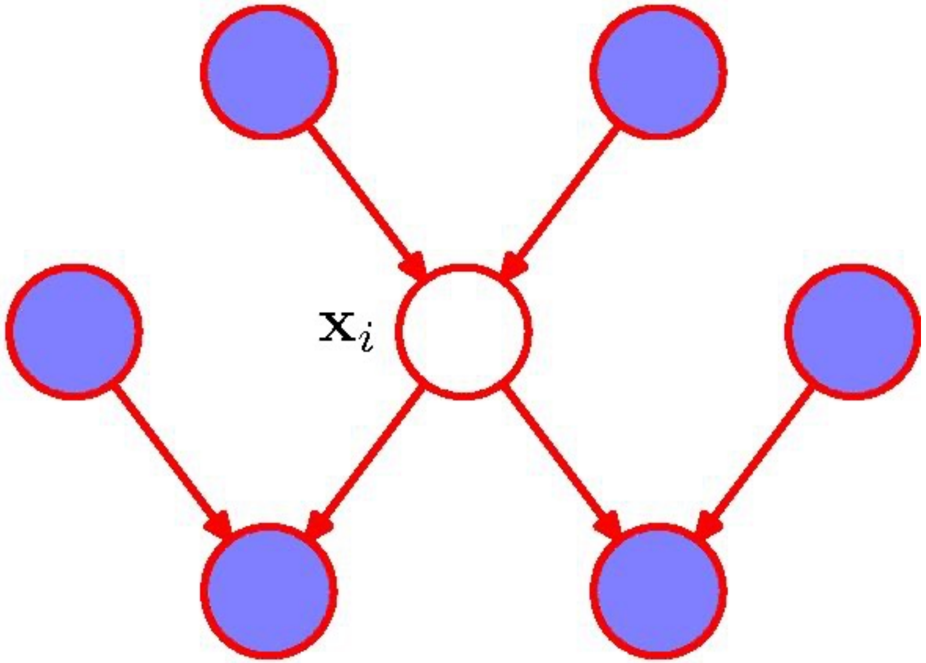


6

Is A d-separated from C by B?



Recap: The Markov Blanket



$$p(\mathbf{x}_i|\mathbf{x}_{\{j\neq i\}}) = \frac{p(\mathbf{x}_1,\ldots,\mathbf{x}_M)}{\int p(\mathbf{x}_1,\ldots,\mathbf{x}_M)\,\mathrm{d}\mathbf{x}_i}$$

$$= \frac{\prod_k p(\mathbf{x}_k|\mathrm{pa}_k)}{\int \prod_k p(\mathbf{x}_k|\mathrm{pa}_k)\,\mathrm{d}\mathbf{x}_i}$$

Factors independent of $\mathbf{x}_i$ cancel between numerator and denominator.

Recap: Markov Random Field

- Undirected, can have cycles

- Markov networks

- Reason about conditional independence using graph reachability

Recap: Markov Random Field

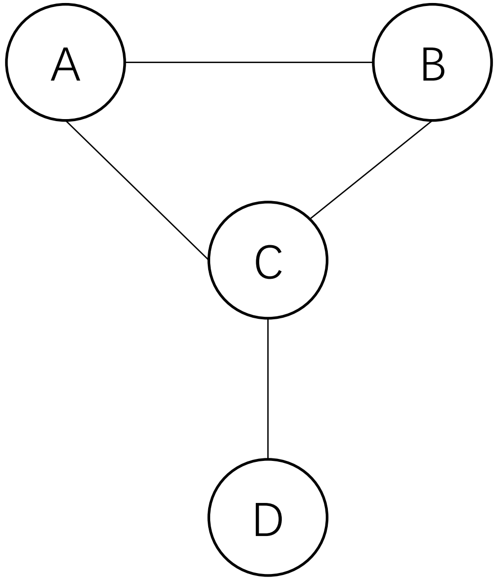
$$p(\mathbf{x}) = \frac{1}{Z}\prod_C \psi_C(\mathbf{x}_C)$$

- where $\psi_C(\mathbf{x}_C)$ is the potential over maximal clique C and

$$Z = \sum_{\mathbf{x}}\prod_C \psi_C(\mathbf{x}_C)$$

- is the normalization coefficient.

Recap: Markov Random Field



$$P(A = True, B = True, C = True, D = True)$$

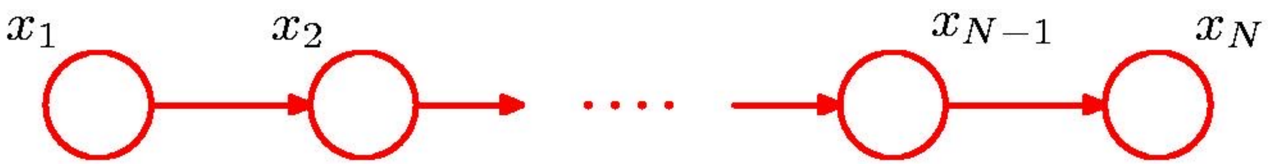
$$= \frac{\psi_{A,B,C}(True, True, True) \times \psi_{C,D}(True, True)}{\Sigma_{A,B,C,D}\psi_{A,B,C}(A, B, C) \times \psi_{C,D}(C, D)}$$

This Class

- Relationship between directed and undirected models

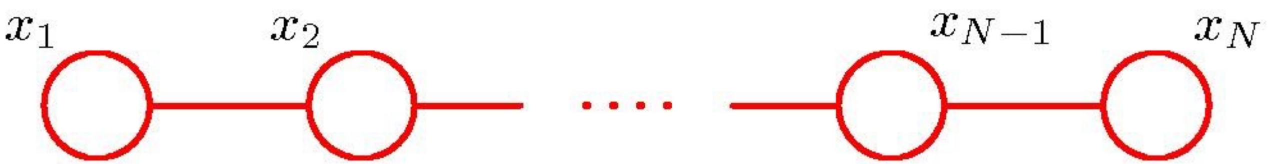
- Inference (“Exact”)

Converting Directed to Undirected Graphs

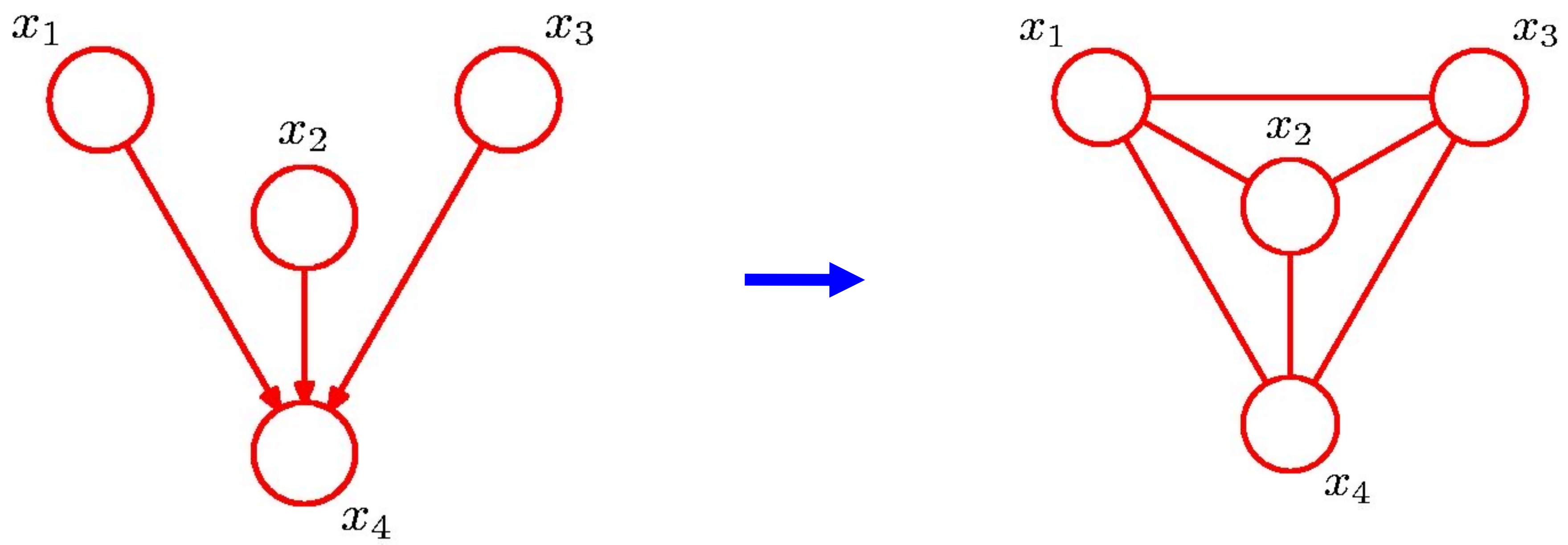


$$p(\mathbf{x}) = \underbrace{p(x_1)p(x_2|x_1)}\, p(x_3|x_2) \cdots p(x_N|x_{N-1})$$

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



Converting Directed to Undirected Graphs



Steps in Converting Directed to Undirected

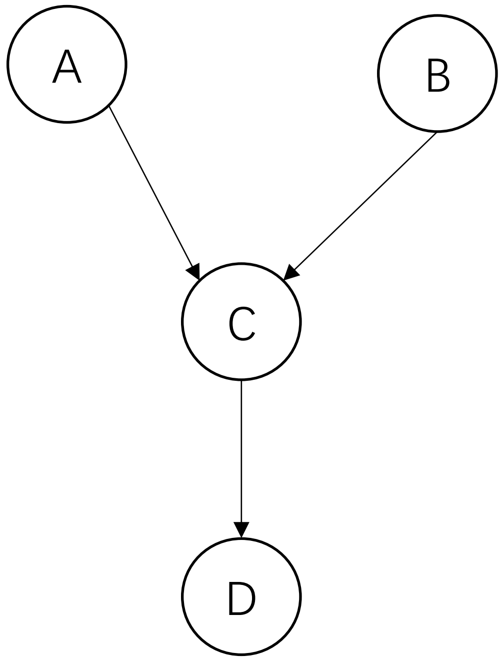
1. Add links between all pairs of parents for each node (moralization)

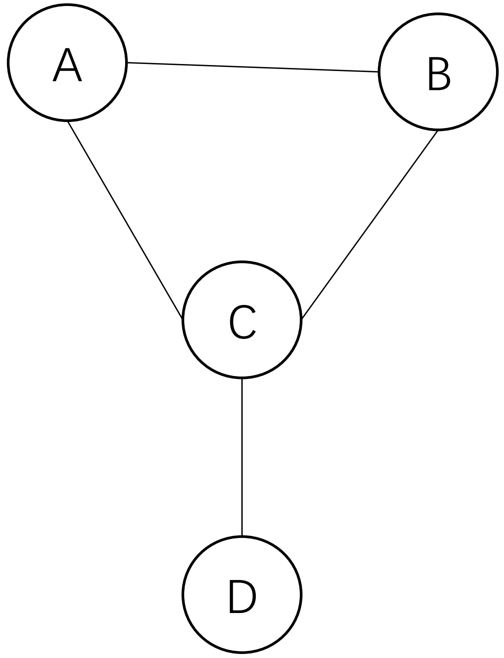
2. Drop arrows, which results in a moral graph

3. Initialize all of the clique potentials to 1. Take each conditional distribution factor and multiply it into one of the clique potentials

4. $Z = 1$

Example



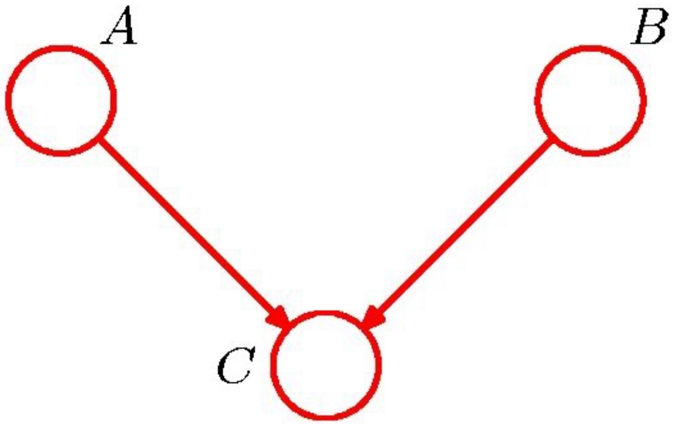


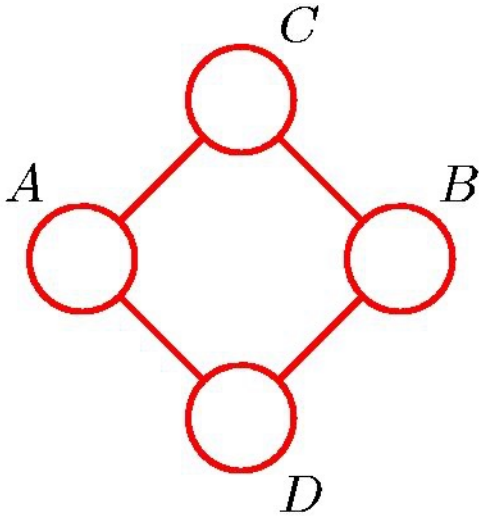
$$\psi_{A,B,C} = P(A)\times P(B)\times P(C|A,B)$$

$$\psi_{C,D} = P(D|C)$$

Directed vs. Undirected Graphs

Can you convert the following graphs and keep the conditional indecencies?





$$A \perp\!\!\!\perp B \mid \emptyset$$

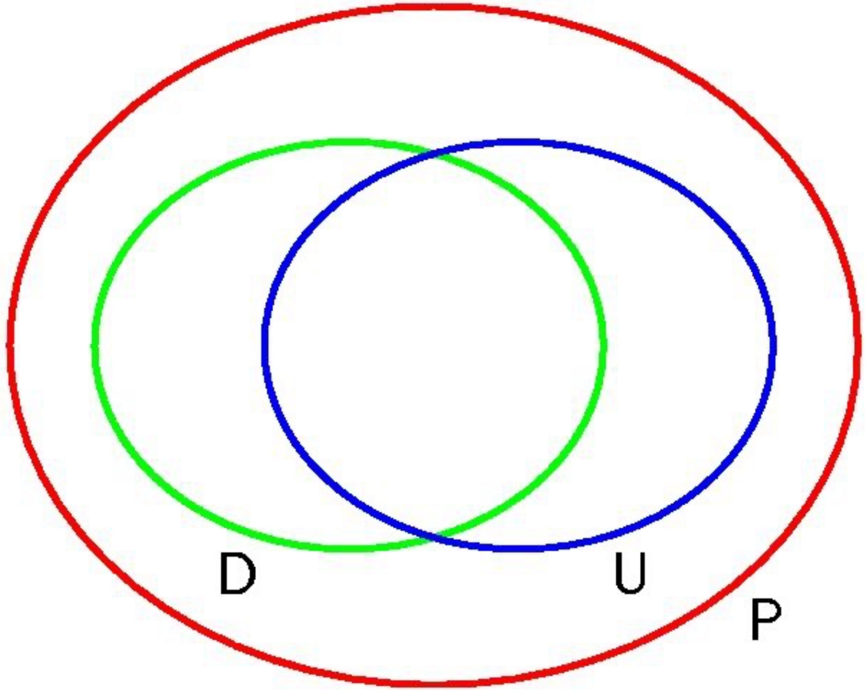
$$A \not\!\perp\!\!\!\perp B \mid C$$

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

$$A \perp\!\!\!\perp B \mid C \cup D$$

$$C \perp\!\!\!\perp D \mid A \cup B$$

Directed vs. Undirected Graphs



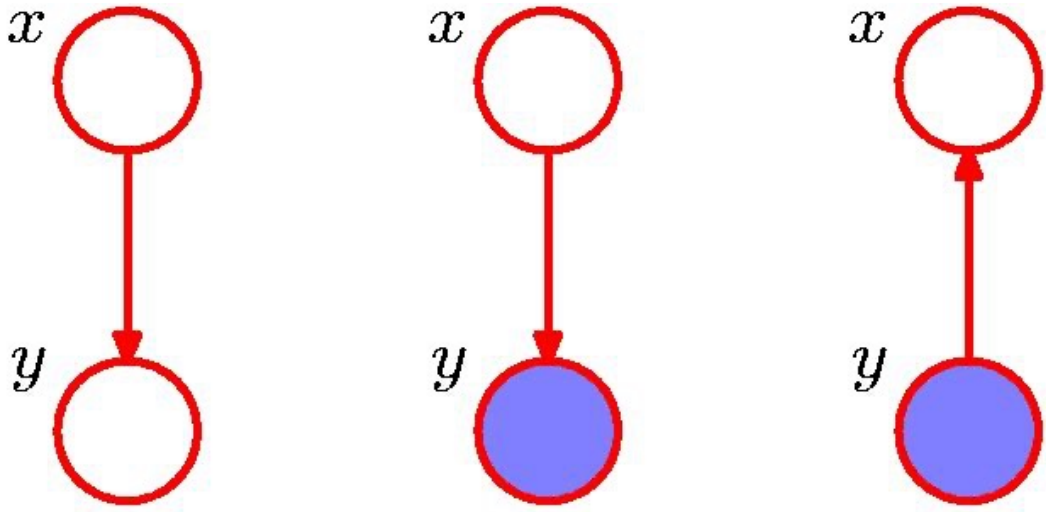
Distributions that can be perfectly represented by two types of graphs in terms of conditional independence

Inference in Graphical Models

- Marginal probabilities: p(x) or p(x,y)
- Conditional probabilities: p(x|o) or p(x,y|o)

What is the computational complexity regarding the number of variables?

Inference in Graphical Models

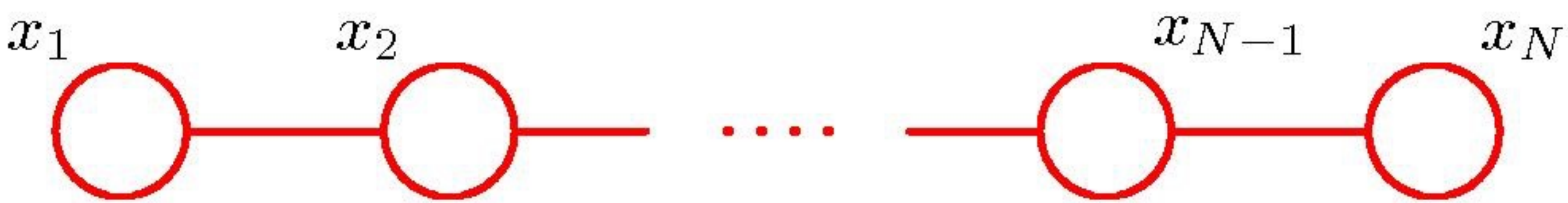


Shaded nodes are observed.

$$p(y) = \sum_{x'} p(y|x')p(x')$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

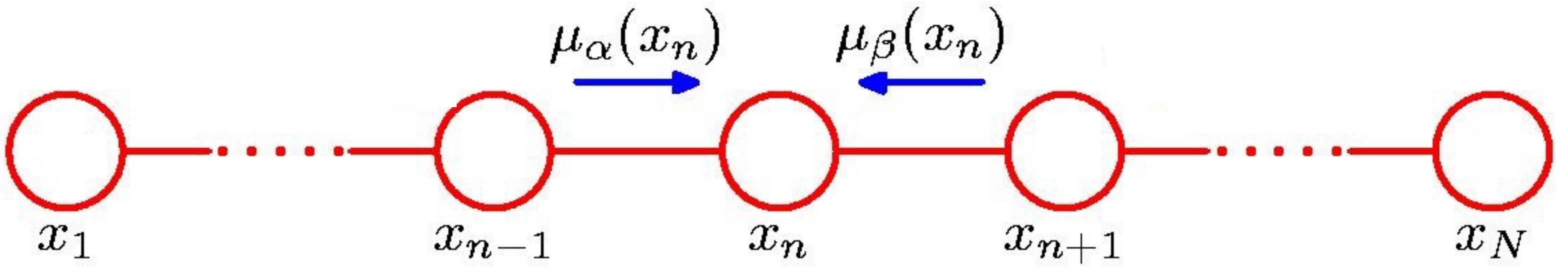
Inference on a Chain



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

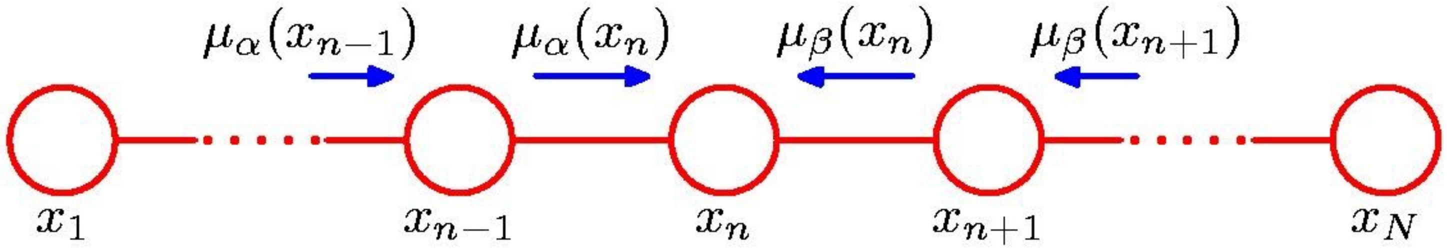
$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$$

Inference on a Chain



$$
\begin{aligned}
p(x_n) &= \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \cdots \right]}_{\mu_\alpha(x_n)} \\
&\quad \underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right]}_{\mu_\beta(x_n)}
\end{aligned}
$$

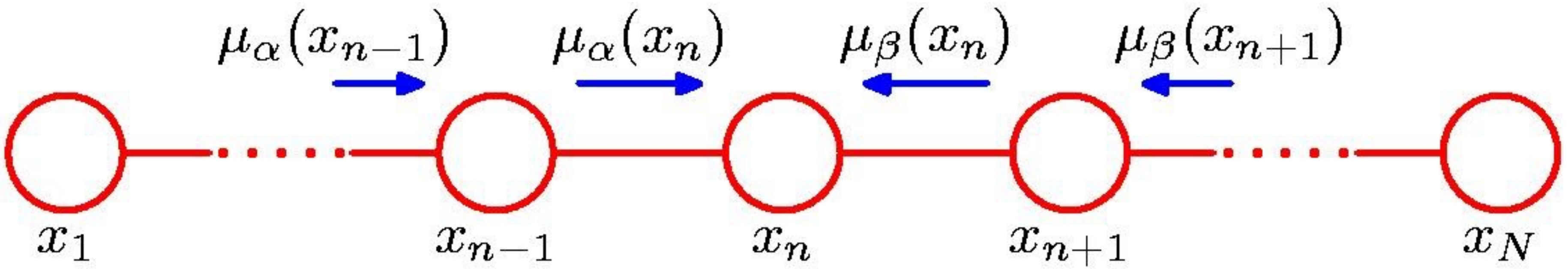
Inference on a Chain



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

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

Inference on a Chain



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

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

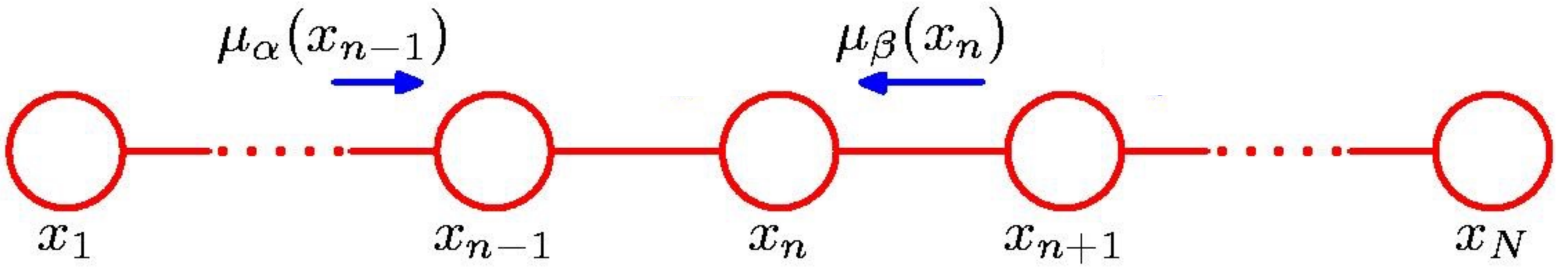
Inference on a Chain

- 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_m
  - Compute

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

  for all variables required.

What about $p(x_{n-1}, x_n)$?



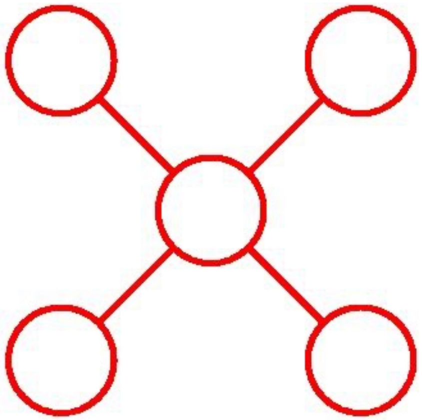
$$
\begin{aligned}
\mathrm{p}(x_{n-1}, x_n) &= \frac{1}{Z}\Sigma_{x_1}..\Sigma_{x_{n-2}}\Sigma_{x_{n+1}} \ldots \Sigma_{x_N}\psi_{1,2}(x_1, x_2) \ldots \psi_{N-1,N}(x_{N-1}, x_N) \\
&= \frac{1}{Z}\psi_{n-1,n}(x_{n-1}, x_n)\Sigma_{x_1}..\Sigma_{x_{n-2}}\psi_{1,2}(x_1, x_2) \ldots \psi_{n-2,n-1}(x_{n-2}, x_{n-1}) \\
&\quad \Sigma_{x_{n+1}}..\Sigma_{x_N}\psi_{n,n+1}(x_n, x_{n+1}) \ldots \psi_{N-1,N}(x_{N-1}, x_N) \\
&= \frac{1}{Z}\psi_{n-1,n}(x_{n-1}, x_n)\mu_\alpha(x_{n-1})\mu_\beta(x_n)
\end{aligned}
$$

What about $p(x_n|x_m=V)$

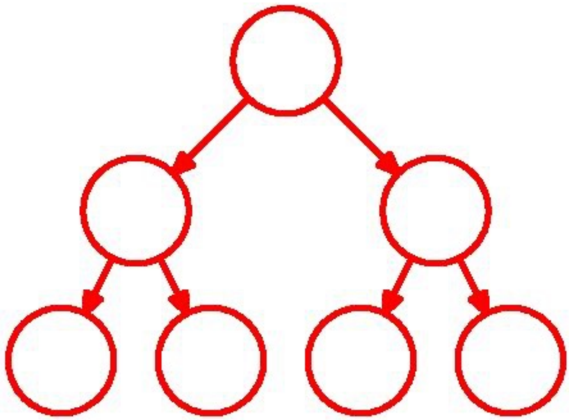
- Simply fix x_m to V instead of doing summarization over x_m!
- Z will also be changed accordingly

More Complex Graphs: Trees

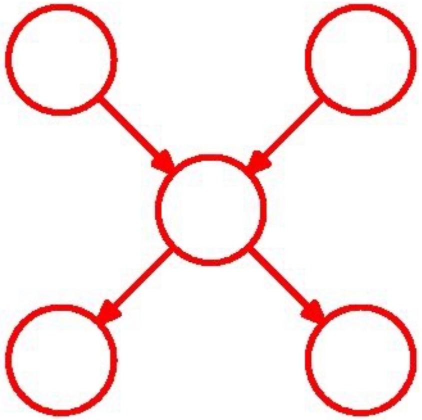
Undirected Tree



Directed Tree



Polytree

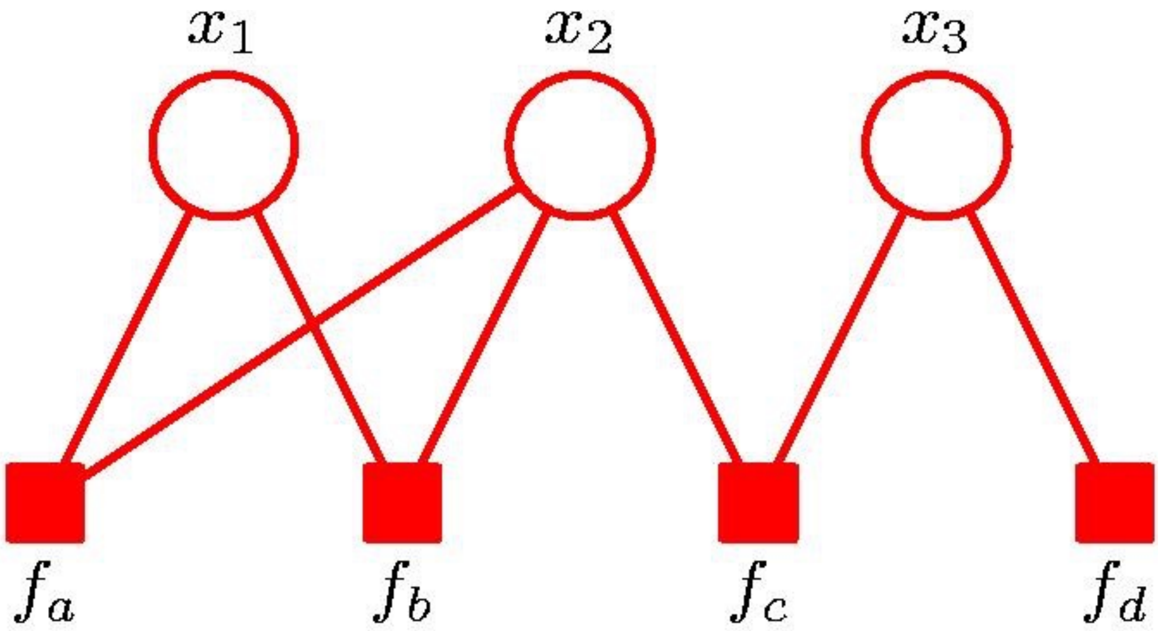


On these graphs, we can perform efficient exact inference using local message passing!

Before introducing algorithms, we first introduce a new model

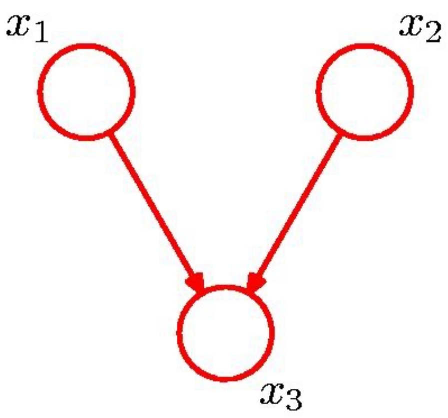
Factor Graphs

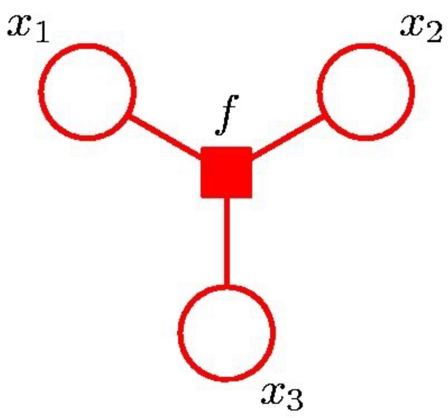
- Bipartite graph
- Two kinds of nodes:
  - Regular random variables
  - Factor nodes
- Factor node represents a function that maps assignments to its neighbors to a real number
- $p(\boldsymbol{x}) = \prod_s f_s(\boldsymbol{x}_s)$

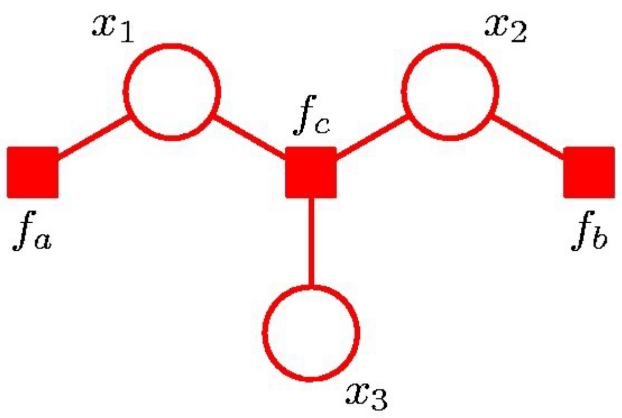


$$p(x_1, x_2, x_3) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

Factor Graphs from Directed Graphs





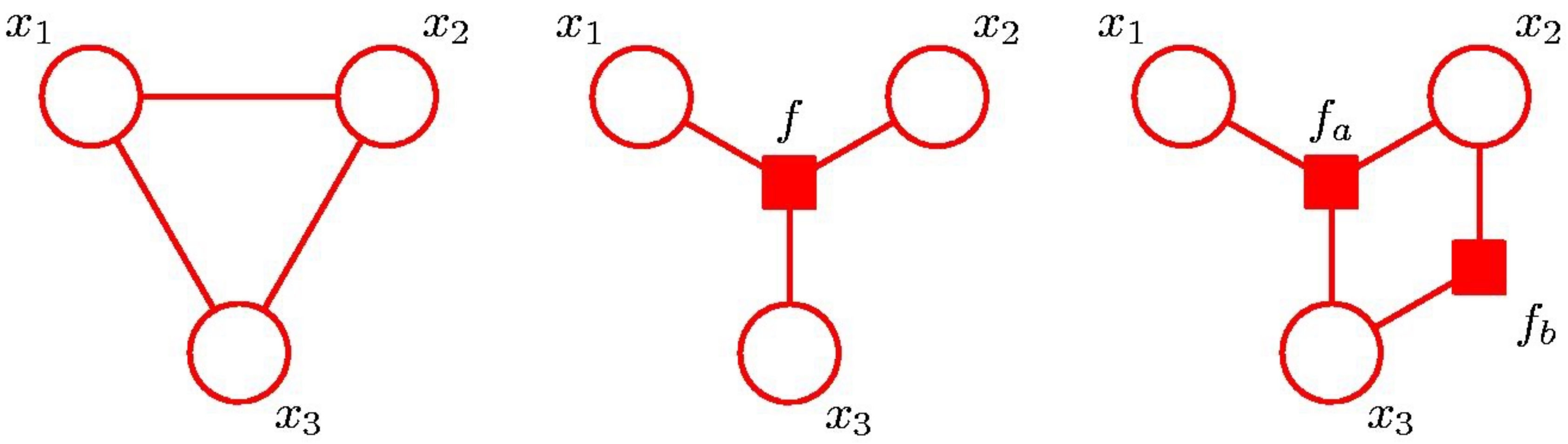


$$\begin{aligned} p(\mathbf{x}) &= p(x_1)p(x_2) \\ &\quad p(x_3|x_1, x_2) \end{aligned}$$

$$\begin{aligned} &f(x_1, x_2, x_3) = \\ &\quad p(x_1)p(x_2)p(_3|x_1, x_2) \end{aligned}$$

$$\begin{aligned} f_a(x_1) &= p(x_1) \\ f_b(x_2) &= p(x_2) \\ f_c(x_1, x_2, x_3) &= p(x_3|x_1, x_2) \end{aligned}$$

Factor Graphs from Undirected Graphs



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

$$\begin{aligned} f(x_1, x_2, x_3) \\ = \quad \psi(x_1, x_2, x_3) \end{aligned}$$

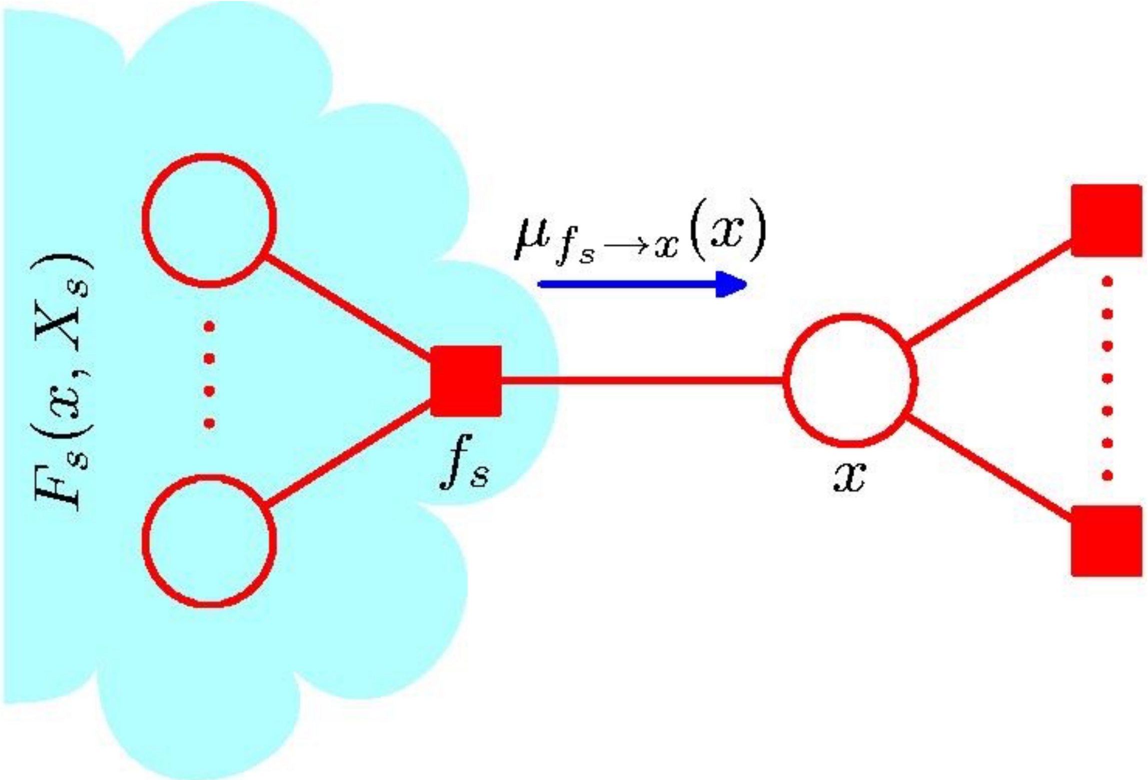
$$\begin{aligned} f_a(x_1, x_2, x_3) f_b(x_2, x_3) \\ = \quad \psi(x_1, x_2, x_3) \end{aligned}$$

The Sum-Product Algorithm

- Objective:
  i. to obtain an efficient, exact inference algorithm for finding marginals on tree-structure graphs;
  ii. in situations where several marginals are required, to allow computations to be shared efficiently.
- Key idea: Distributive Law

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

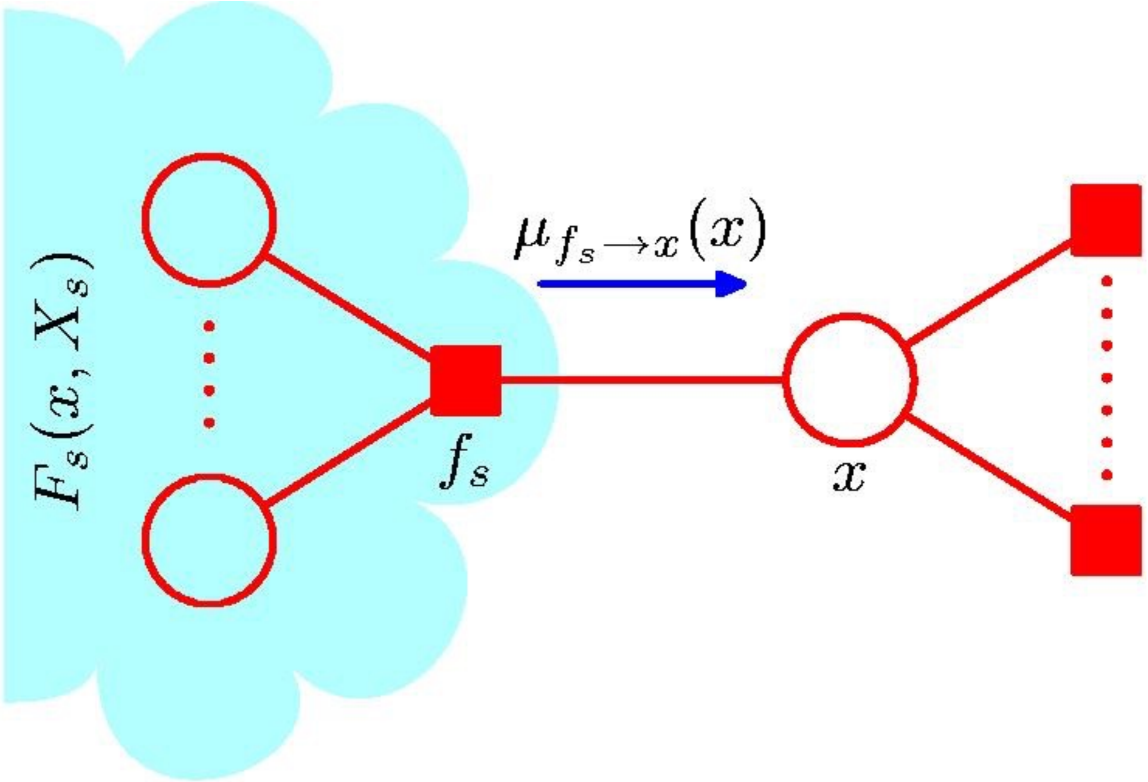
The Sum-Product Algorithm



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

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

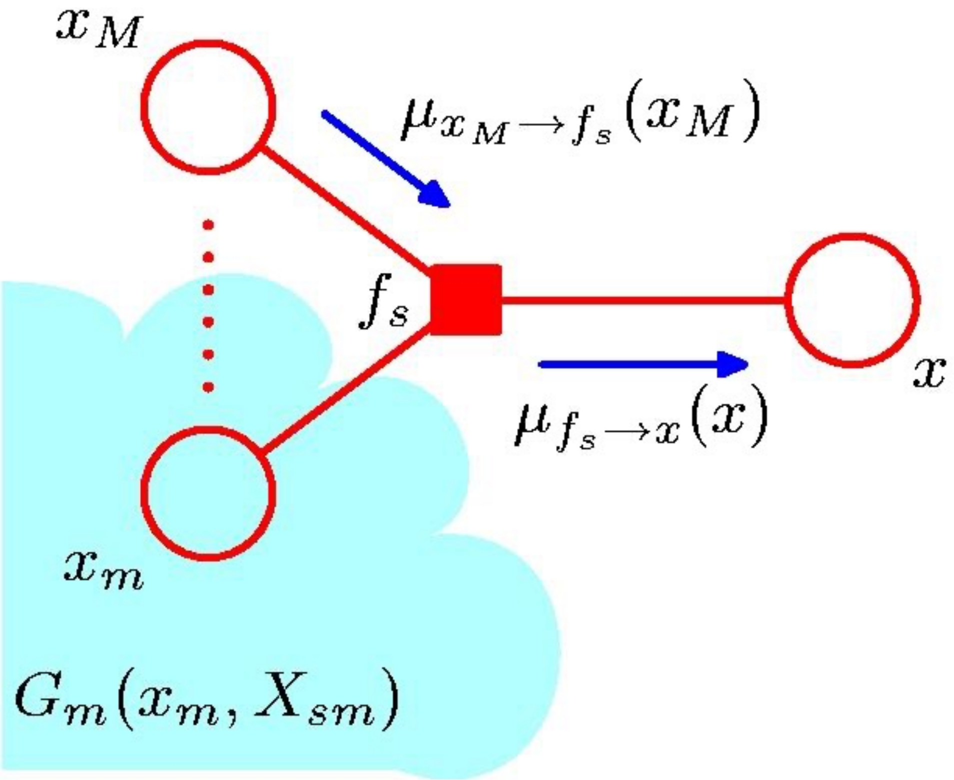
The Sum-Product Algorithm



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

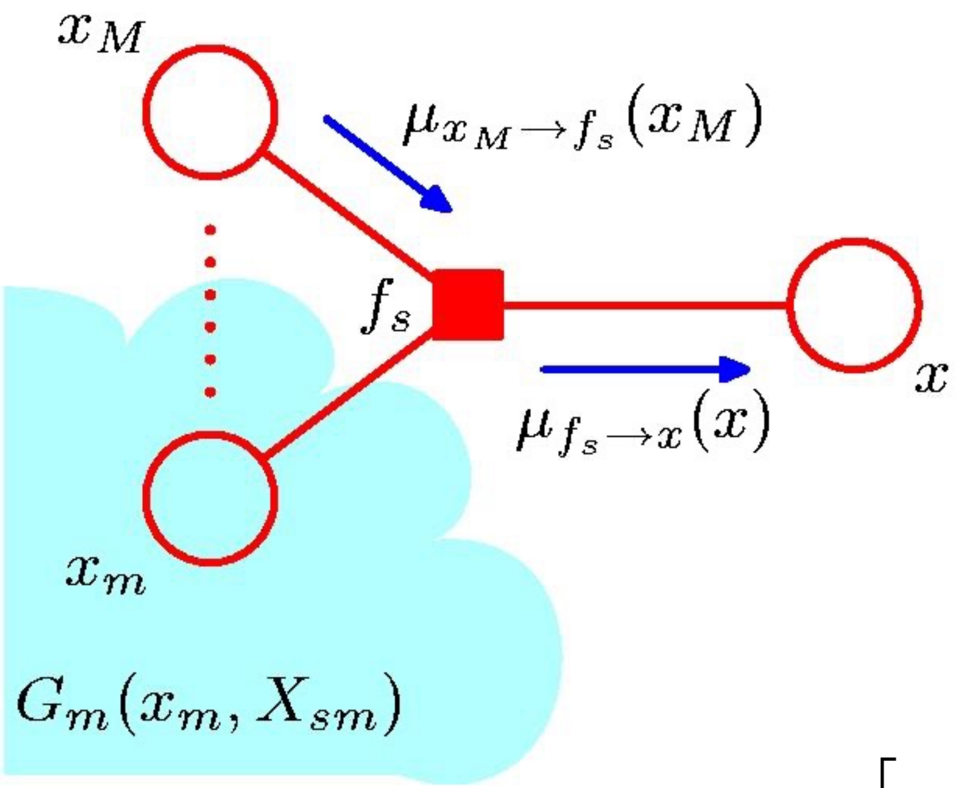
$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

The Sum-Product Algorithm



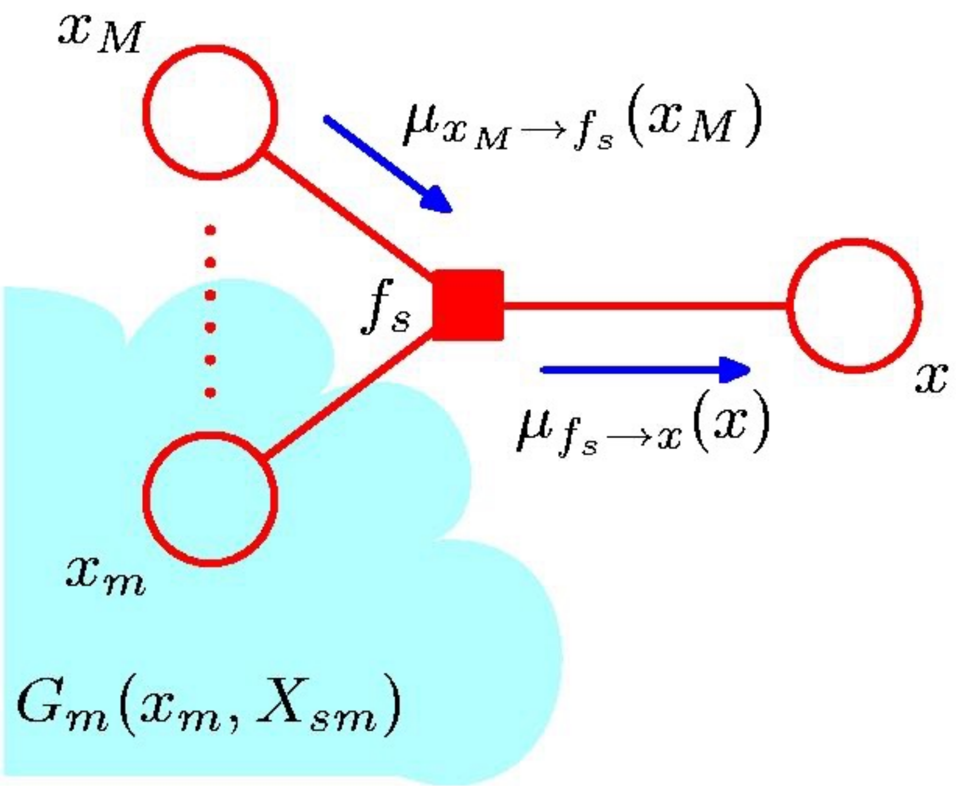
$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \ldots G_M(x_M, X_{sM})$$

The Sum-Product Algorithm



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

The Sum-Product Algorithm

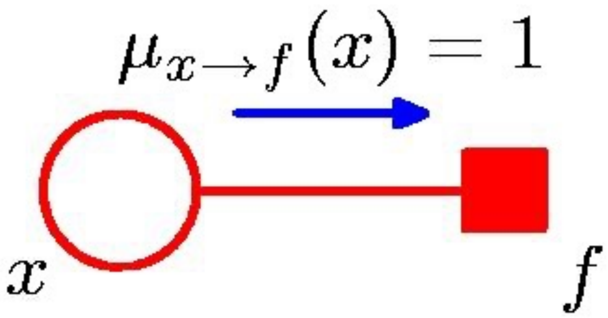


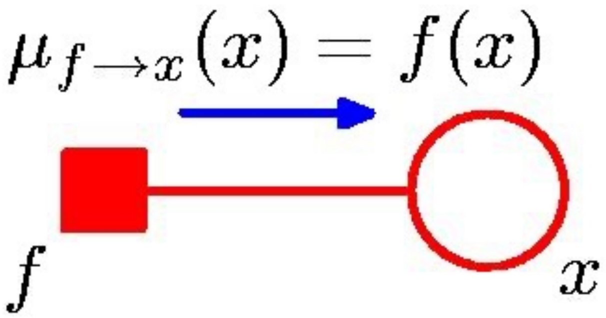
$$\mu_{x_m \to f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm}) = \sum_{X_{sm}} \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} F_l(x_m, X_{ml})$$

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

The Sum-Product Algorithm

- Initialization





The Sum-Product Algorithm

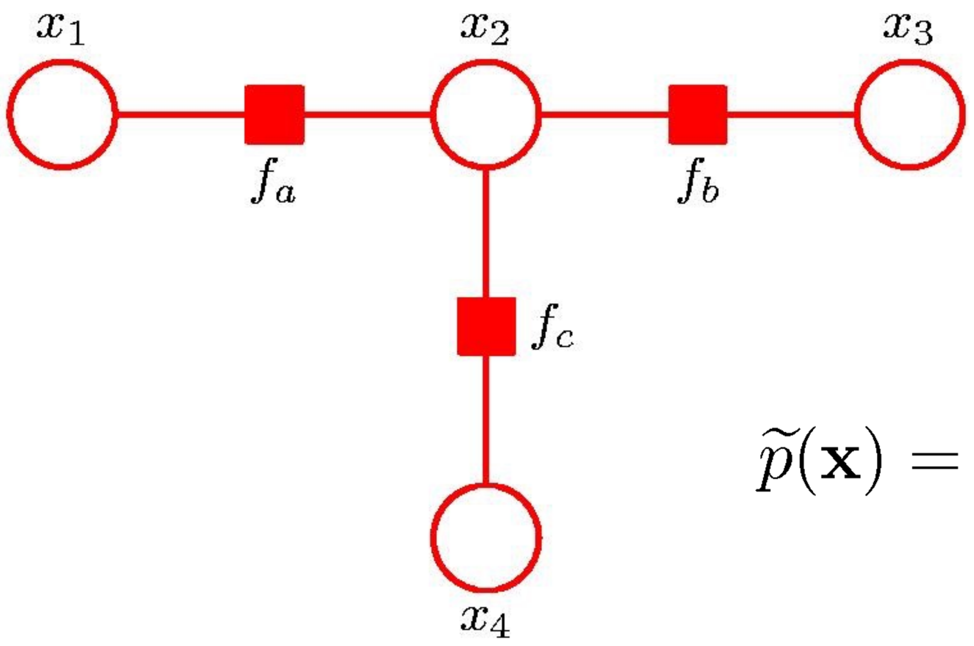
- To compute local marginals:
  - Pick an arbitrary node as root
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product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

Marginal Inference on A Set

- What if I want to know $p(\boldsymbol{x_s})$ where $\boldsymbol{x_s}$ are nodes in a factor s?

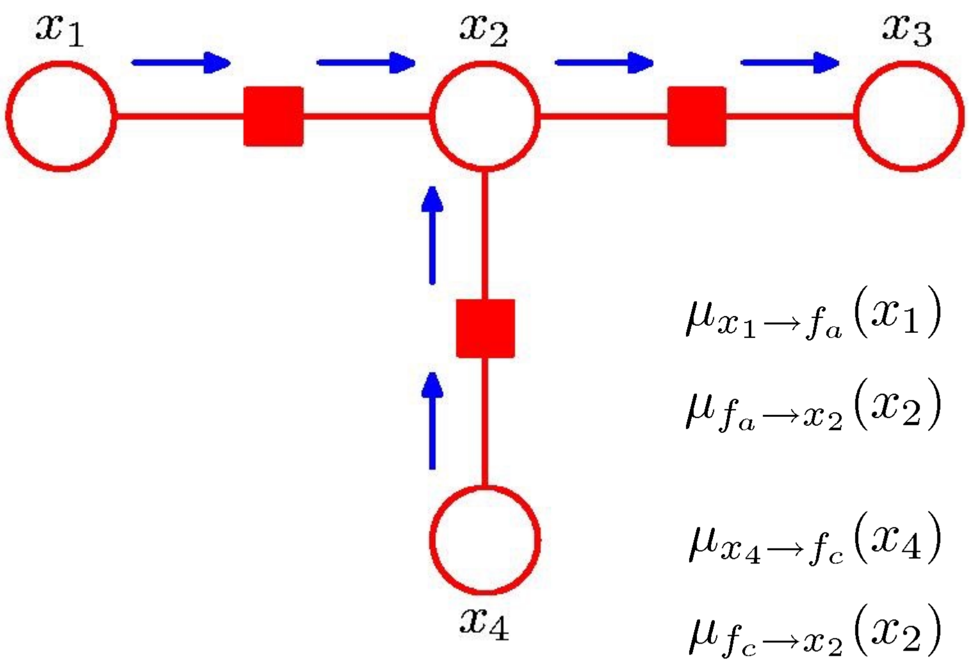
$$p(\boldsymbol{x_s}) = f_s(\boldsymbol{x_s}) \prod_{i \in ne(f_s)} \mu_{x_i \to f_s}(x_i)$$

Sum-Product: Example



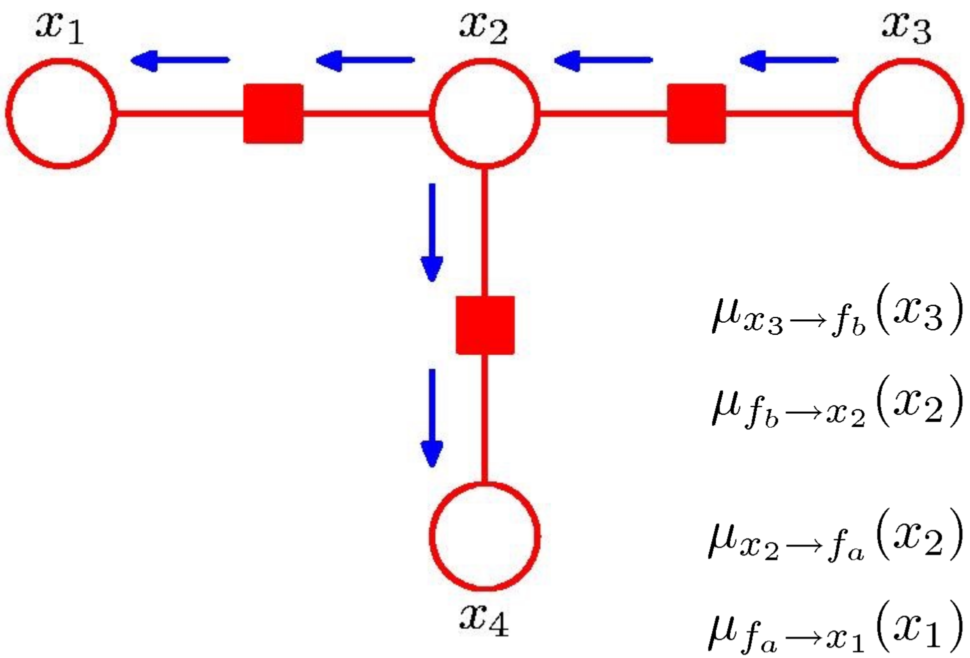
$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

Sum-Product: Example



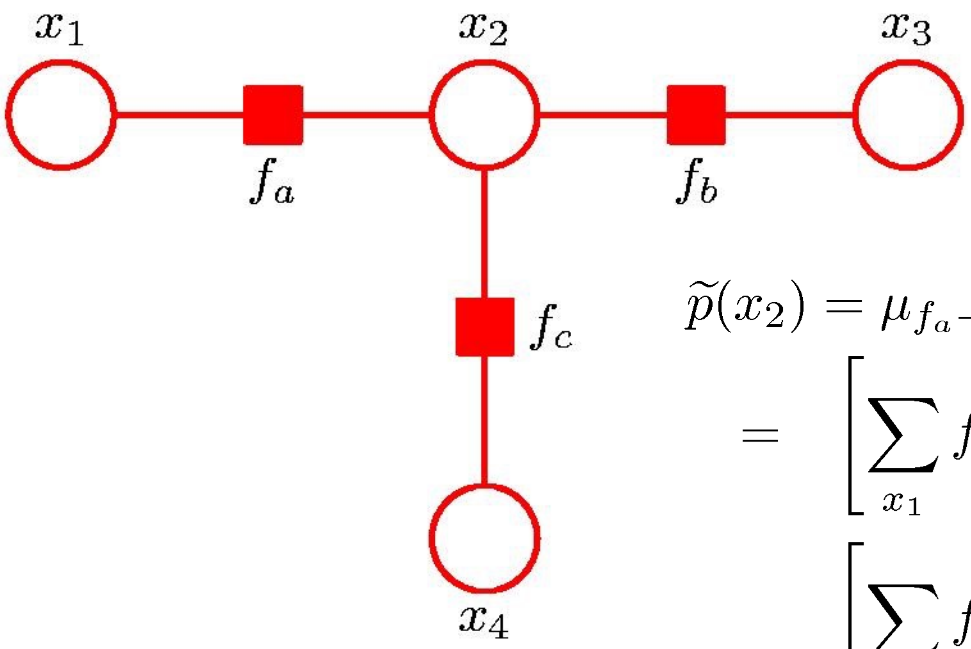
$$
\begin{aligned}
\mu_{x_1 \to f_a}(x_1) &= 1 \\
\mu_{f_a \to x_2}(x_2) &= \sum_{x_1} f_a(x_1, x_2) \\
\mu_{x_4 \to f_c}(x_4) &= 1 \\
\mu_{f_c \to x_2}(x_2) &= \sum_{x_4} f_c(x_2, x_4) \\
\mu_{x_2 \to f_b}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_b \to x_3}(x_3) &= \sum_{x_2} f_b(x_2, x_3)\mu_{x_2 \to f_b}(x_2)
\end{aligned}
$$

Sum-Product: Example



$$\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2) \\
\mu_{f_c \to x_4}(x_4) &= \sum_{x_2} f_c(x_2, x_4)\mu_{x_2 \to f_c}(x_2)
\end{aligned}$$

Sum-Product: Example



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

What about conditional probabilities?

- Fix the observed variables
- Or add a factor node
- Both need normalization

What if I want to know values of all variables that have the highest probability?

$\text{argmax}_{\mathbf{x}}\ p(\mathbf{x})$

The Max-Sum Algorithm

Objective: an efficient algorithm for finding

i. the value x^{max} that maximises $p(x)$;

ii. the value of $p(x^{max})$.

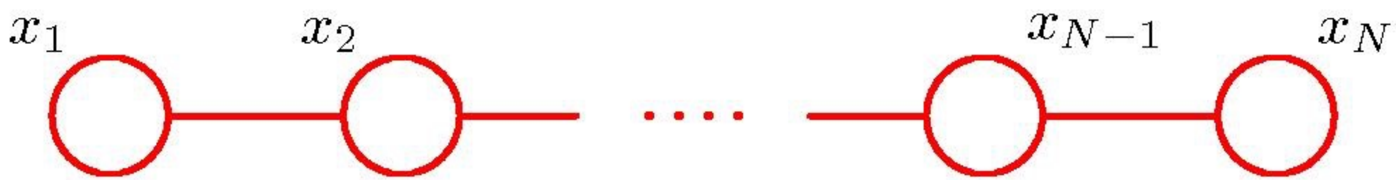
In general, maximum marginals ≠ joint maximum

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

$$\arg\max_x p(x, y) = 1 \qquad \arg\max_x p(x) = 0$$

The Max-Sum Algorithm

- Maximizing over a chain (max-product)



$$
\begin{aligned}
p(\mathbf{x}^{\max}) &= \max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \ldots \max_{x_M} p(\mathbf{x}) \\
&= \frac{1}{Z} \max_{x_1} \cdots \max_{x_N} \left[\psi_{1,2}(x_1, x_2) \cdots \psi_{N-1,N}(x_{N-1}, x_N)\right] \\
&= \frac{1}{Z} \max_{x_1} \left[\max_{x_2} \left[\psi_{1,2}(x_1, x_2) \left[\cdots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots\right]\right]
\end{aligned}
$$

The Max-Sum Algorithm

- Generalizes to tree-structured factor graph

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_n} \prod_{f_s \in \mathrm{ne}(x_n)} \max_{X_s} f_s(x_n, X_s)$$

- maximizing as close to the leaf nodes as possible

max(ab, ac) = a max(b,c)

The Max-Sum Algorithm

- Max-Product → Max-Sum
  - For numerical reasons, use

$$\ln\left(\max_{\mathbf{x}} p(\mathbf{x})\right) = \max_{\mathbf{x}} \ln p(\mathbf{x}).$$

  - Again, use distributive law

$$\max(a+b, a+c) = a + \max(b, c).$$

The Max-Sum Algorithm

- Initialization (leaf nodes)

$$\mu_{x \to f}(x) = 0 \qquad \mu_{f \to x}(x) = \ln f(x)$$

- Recursion

$$\mu_{f \to x}(x) = \max_{x_1,\ldots,x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_{\cdot}) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

$$\phi(x) = \arg\max_{x_1,\ldots,x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_{\cdot}) \setminus x} \mu_{x_m \to f}(x_m) \right]$$ Track the values

$$\mu_{x \to f}(x) = \sum_{l \in \mathrm{ne}(x) \setminus f} \mu_{f_l \to x}(x)$$

Max-Sum Algorithm

- Termination (root node)

$$
\begin{aligned}
p^{\max} &= \max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right] \\
x^{\max} &= \arg\max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]
\end{aligned}
$$

- Back-track, for all nodes i with l factor nodes to the root (l=0)

$$
\mathbf{x}_l^{\max} = \phi(x_{i,l-1}^{\max})
$$

Sum-Product vs. Max-Sum

Sum-Product

$$\mu_{f\to x}(x) = \sum_{x_1} \dots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{x_m \in ne(f)\backslash \mathrm{x}} \mu_{x_m \to f}(x_m)$$

$$\mu_{x\to f}(x) = \prod_{l \in ne(x)\backslash \mathrm{f}} \mu_{f_l \to x}(x)$$

a(b+c) = ab+bc

Max-Sum

$$\mu_{f\to x}(x) = \max_{x_1,\dots,x_M} [ln f(x, x_1, \dots, x_M) + \sum_{x_m \in ne(f)\backslash \mathrm{x}} \mu_{x_m \to f}(x_m)]$$

$$\mu_{x\to f}(x) = \sum_{l \in ne(x)\backslash \mathrm{f}} \mu_{f_l \to x}(x)$$

a+max(b,c) =max(a+b, a+c)

What about inference on general graphs?

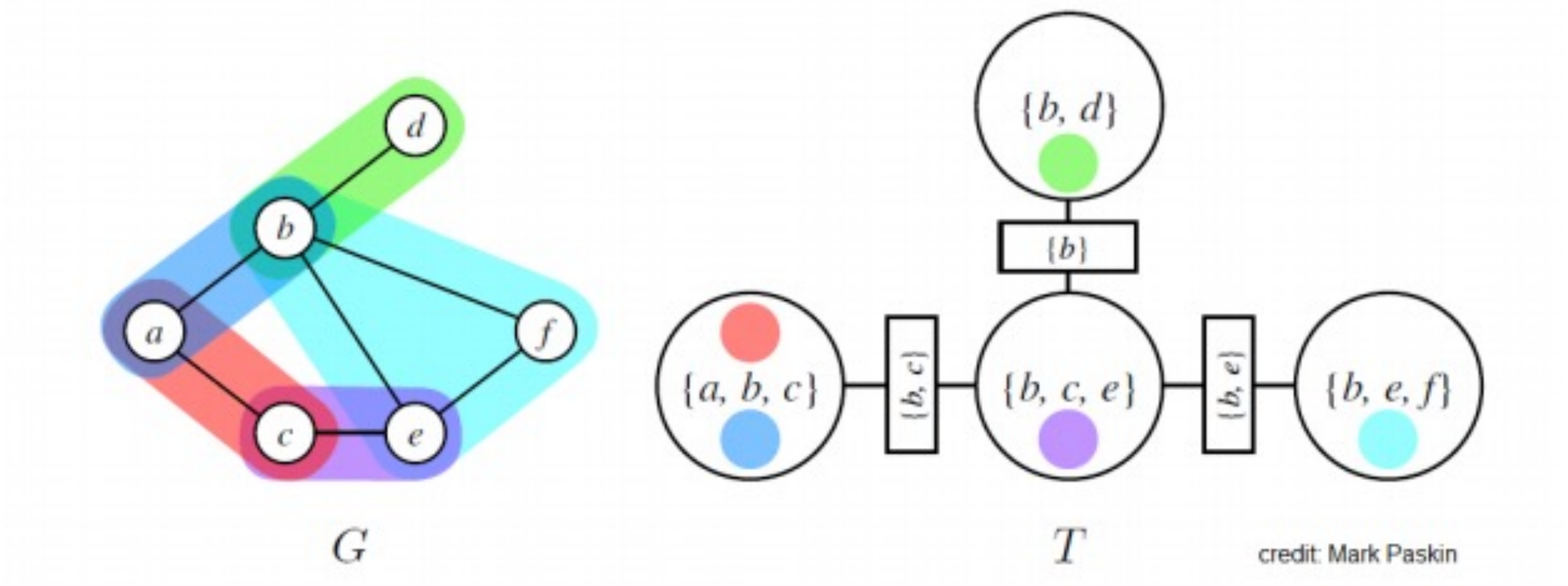
- NP-complete

- Counting problem

The Junction Tree Algorithm

- *Exact* inference on general graphs
- Works by turning the initial graph into a *junction tree* and then running a sum-product-like algorithm
- *Intractable* on graphs with large cliques

The Junction Tree Algorithm



Loopy Belief Propagation

- Sum-Product on general graphs
- Initial unit messages passed across all links, after which messages are passed around until convergence (not guaranteed!)
- *Approximate* but *tractable* for large graphs
- Sometime works well, sometimes not at all

Recap

- Bayesian networks → Markov Random Fields
  - Connect parents
  - Drop arrows
  - Multiply conditional probabilities to get potentials

- Factor graph
  - Random variable nodes
  - Factor nodes
  - $F(\boldsymbol{x}) = \prod_f f(x_1, x_2, \dots, x_n)$

Recap

- Marginal inference on tree-structure factor graph
  - Sum-product algorithm: a message-passing algorithm
  - Exchange sum and product using the distribution law
  - Messages from a factor to a node: sum over products of messages from other nodes to the factor
  - Messages from a node to a factor: product over messages from other factors to the node

- Inferring settings with the highest probability
  - Max-sum algorithm

Recap

- Inference on general graphs with loops is NPC
  - Exact: junction algorithm
  - Approximate: loopy belief propagation

Next Class

- Approximate inference
    - Sampling methods