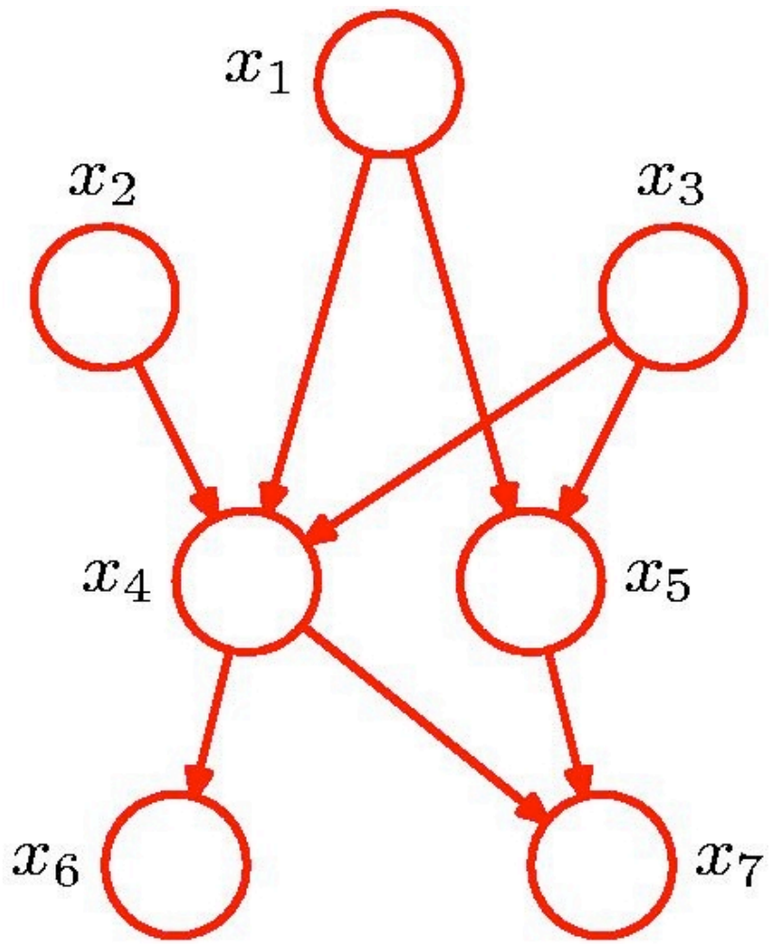
4. Probabilistic Graphical Models

Undirected Models

Repetition: Bayesian Networks

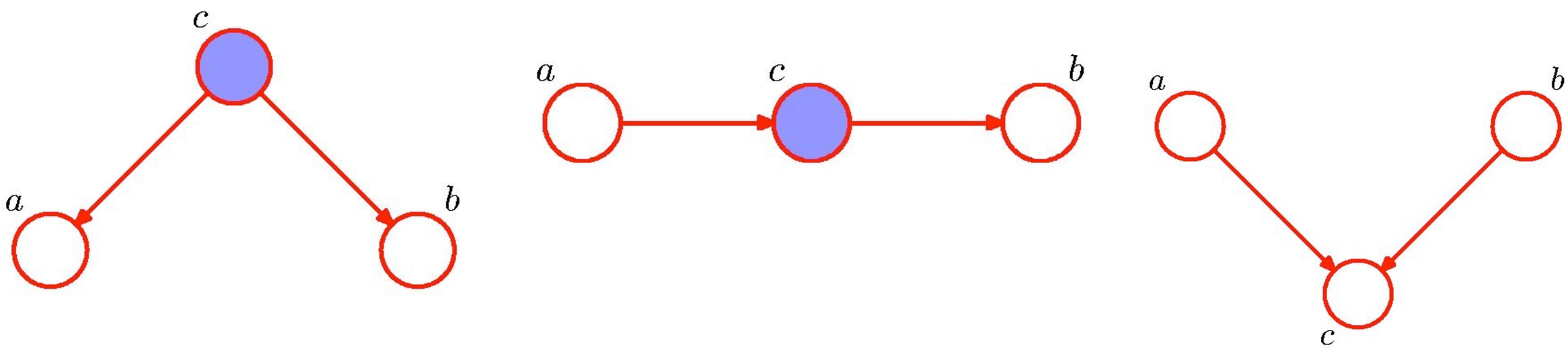


Directed graphical models can be used to represent **probability distributions**

This is useful to do **inference** and to **generate samples** from the distribution efficiently

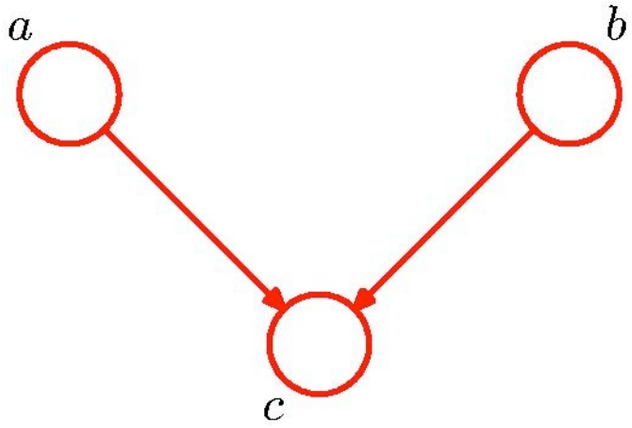
$$p(x_1, \ldots, x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3)$$
$$p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5)$$

Repetition: D-Separation



- D-separation is a property of graphs that can be easily determined
- An I-map assigns every d-separation a c.i. rel
- A D-map assigns every c.i. rel a d-separation
- Every Bayes net determines a unique prob. dist.

In-depth: The Head-to-Head Node



$p(a) = 0.9 \qquad p(b) = 0.9$

a	b	p(c)
1	1	0.8
1	0	0.2
0	1	0.2
0	0	0.1

Example:

a: Battery charged (0 or 1)

b: Fuel tank full (0 or 1)

c: Fuel gauge says full (0 or 1)

We can compute $p(\neg c) = 0.315$

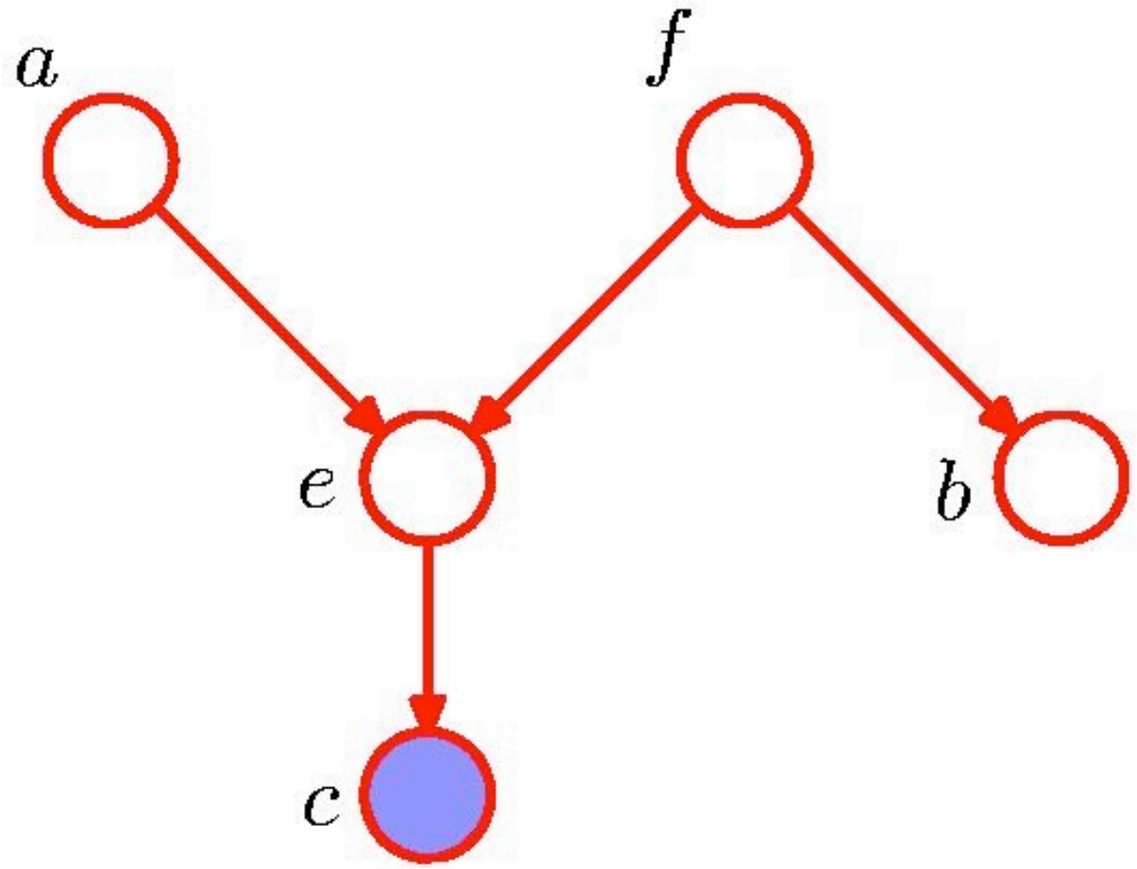
and $p(\neg c \mid \neg b) = 0.81$

and obtain $p(\neg b \mid \neg c) \approx 0.257$

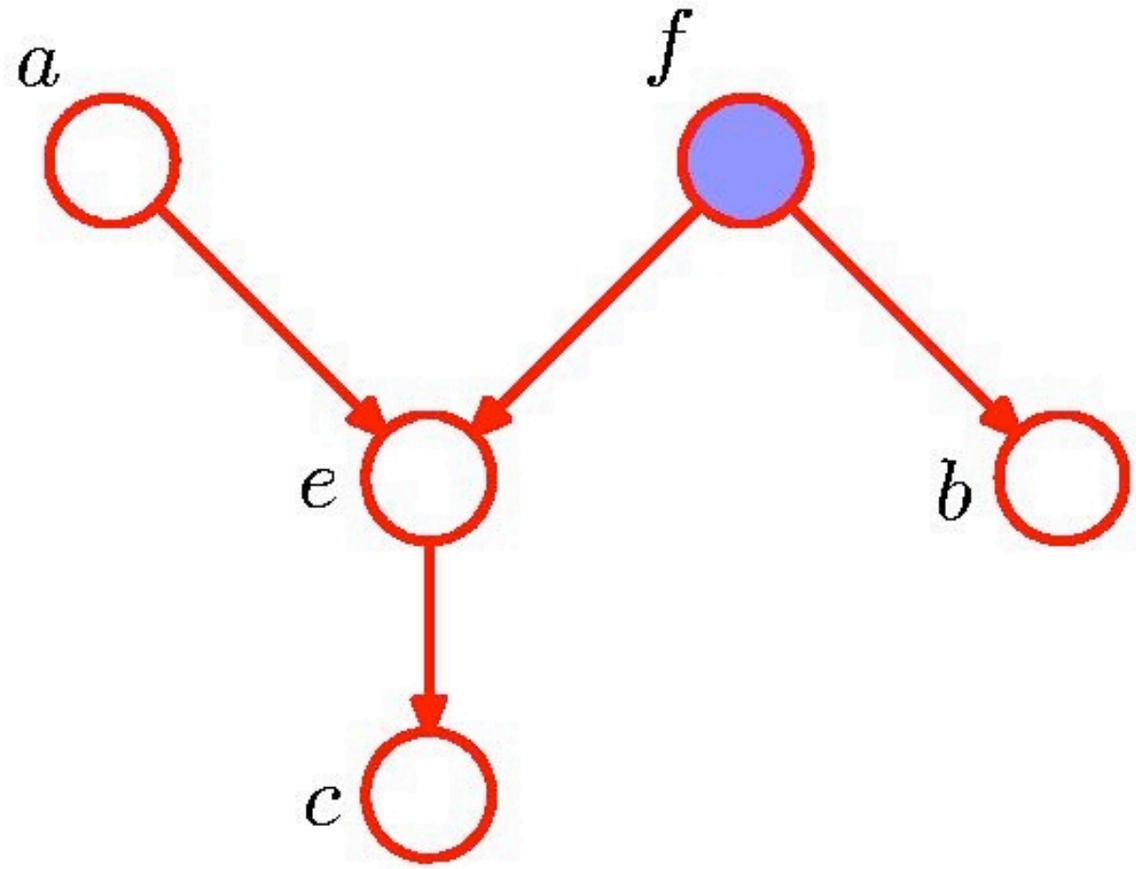
similarly: $p(\neg b \mid \neg c, \neg a) \approx 0.111$

“a **explains** c **away**”

Repetition: D-Separation



$$\neg \text{dsep}(a, b|c)$$



$$\text{dsep}(a, b|f)$$

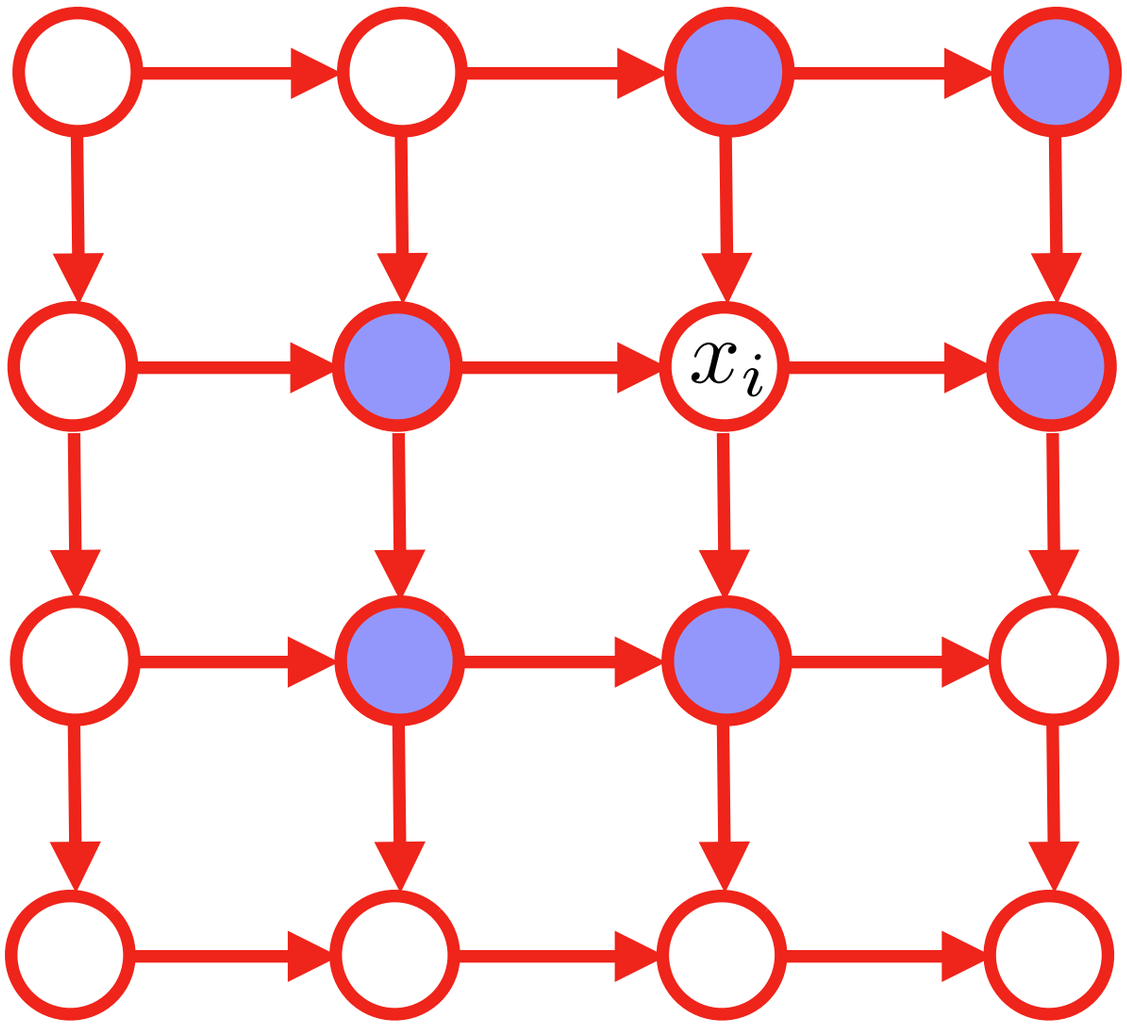
Directed vs. Undirected Graphs

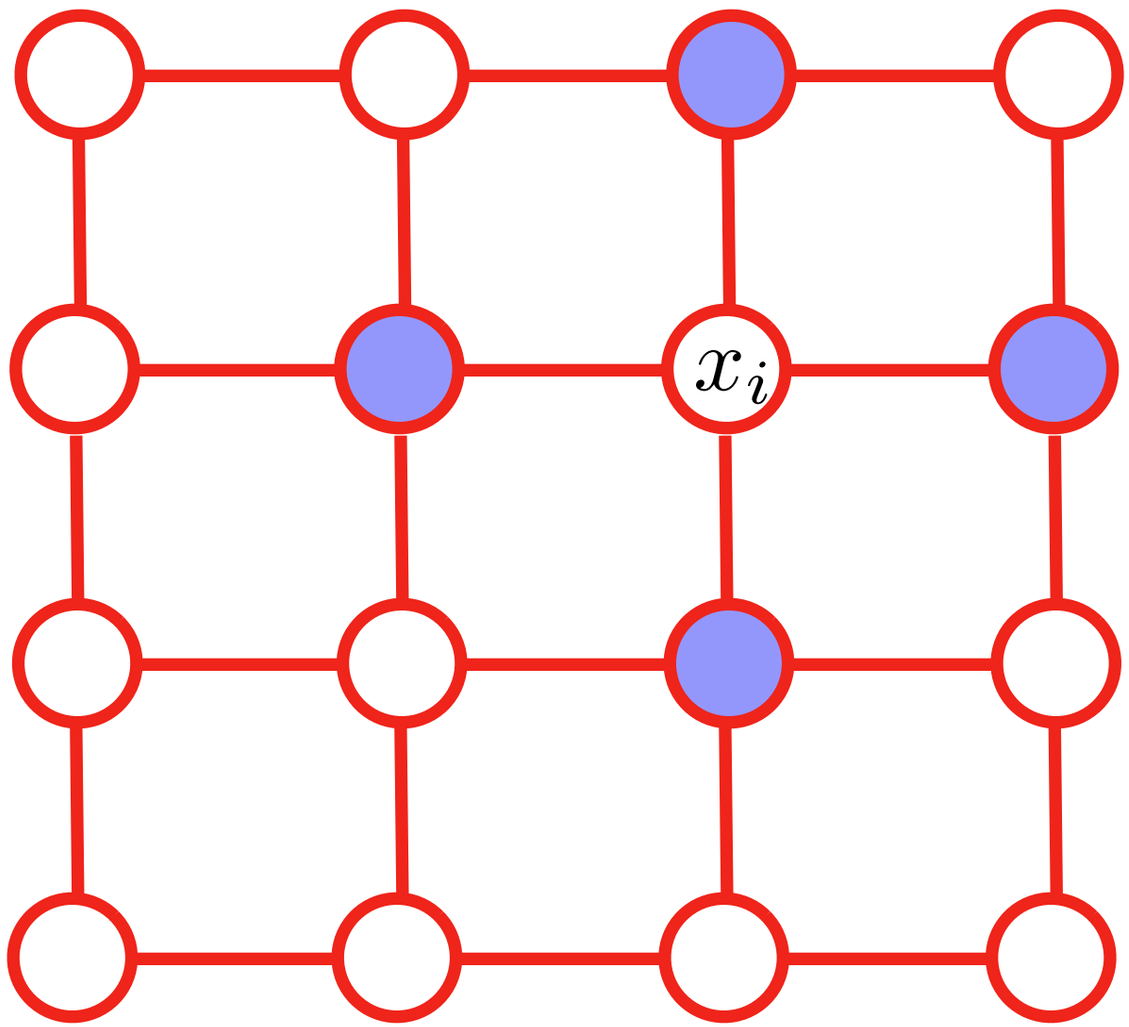
Using D-separation we can identify conditional independencies in directed graphical models, but:

- Is there a simpler, more intuitive way to express conditional independence in a graph?
- Can we find a representation for cases where an „ordering“ of the random variables is inappropriate (e.g. the pixels in a camera image)?

Yes, we can: by removing the directions of the edges we obtain an Undirected Graphical Model, also known as a Markov Random Field

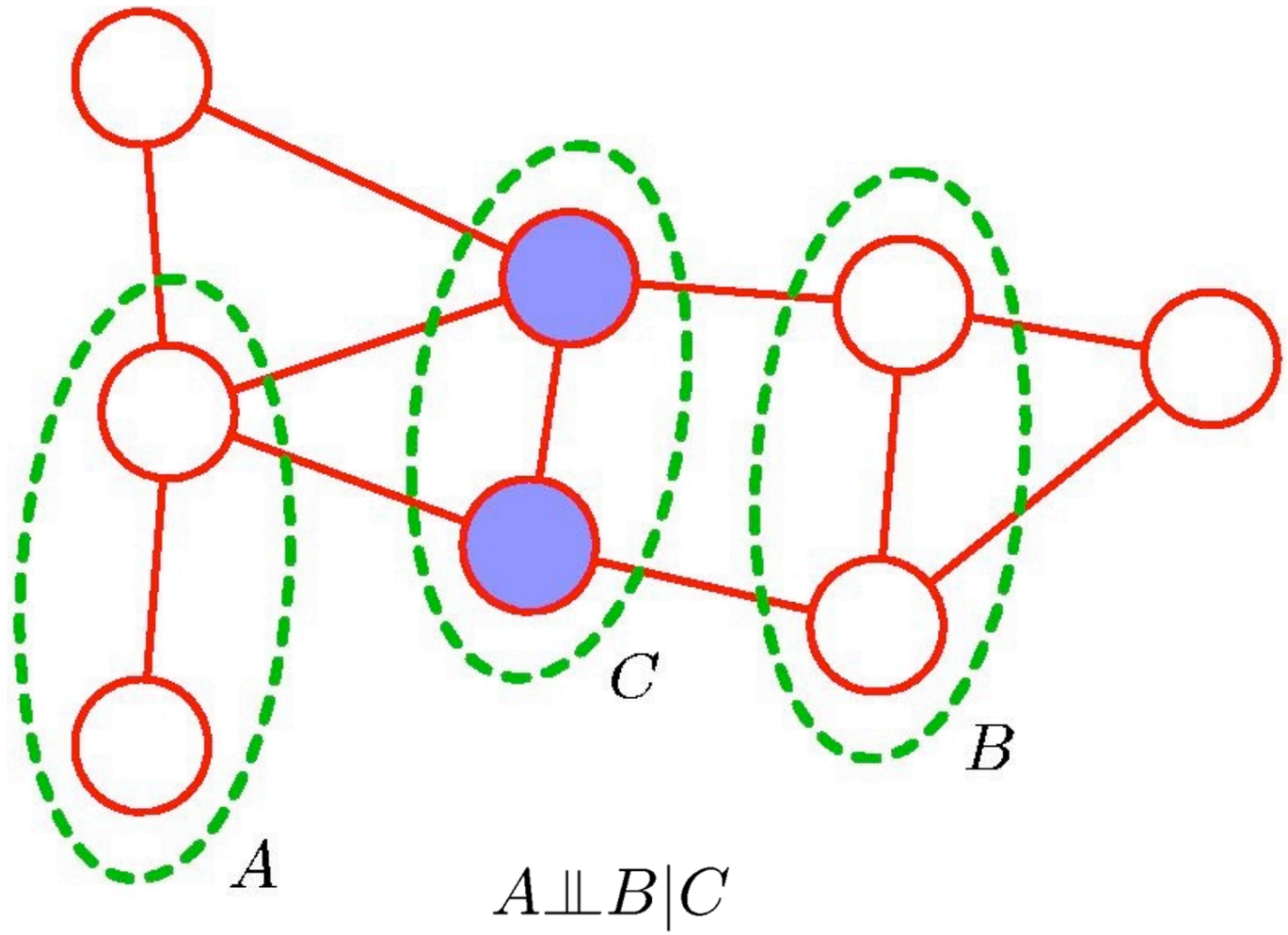
Example: Camera Image





- directions are counter-intuitive for images
- Markov blanket is not just the direct neighbors

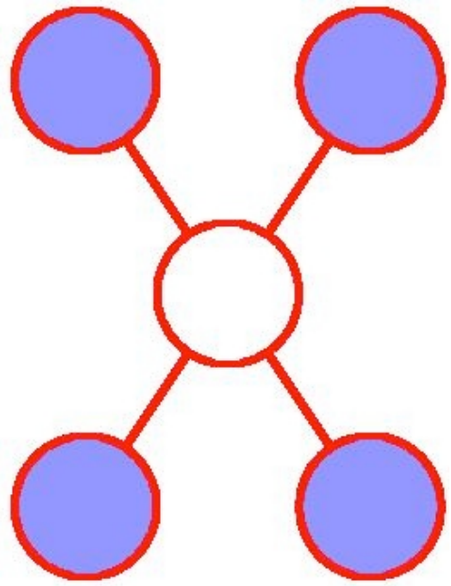
Markov Random Fields



$A \perp\!\!\!\perp B | C$

All paths from A to B go through C, i.e. C blocks all paths.

Markov Blanket



We only need to condition on the **direct neighbors** of $\mathbf{x}$ to get c.i., because these already block every path from $\mathbf{x}$ to any other node.

Factorization of MRFs

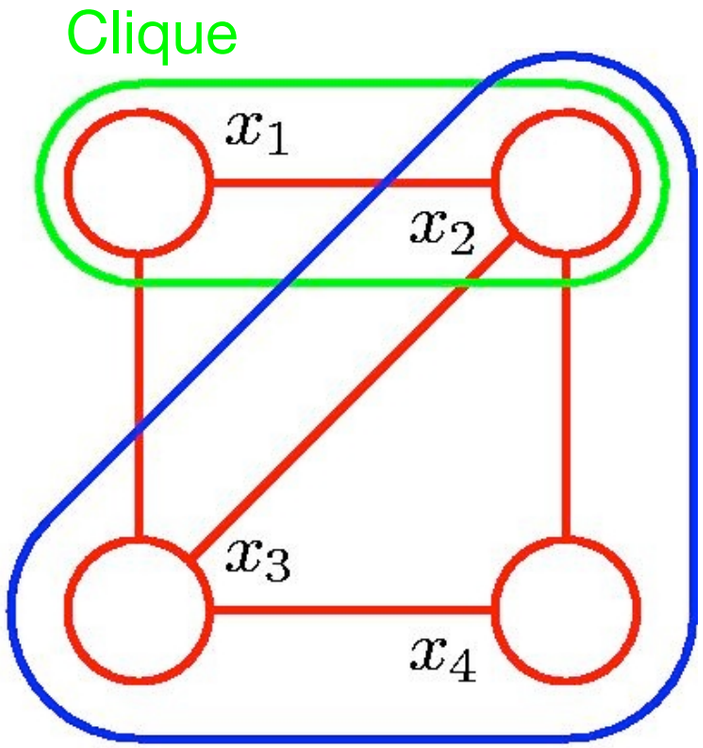
Any two nodes x_i and x_j that are not connected in an MRF are conditionally independent given all other nodes:

$$p(x_i, x_j \mid \mathbf{x}_{\setminus\{i,j\}}) = p(x_i \mid \mathbf{x}_{\setminus\{i,j\}}) p(x_j \mid \mathbf{x}_{\setminus\{i,j\}})$$

In turn: each factor contains only nodes that are connected

This motivates the consideration of cliques in the graph:

- A **clique** is a fully connected subgraph.
- A **maximal** clique can not be extended with another node without loosing the property of full connectivity.



Factorization of MRFs

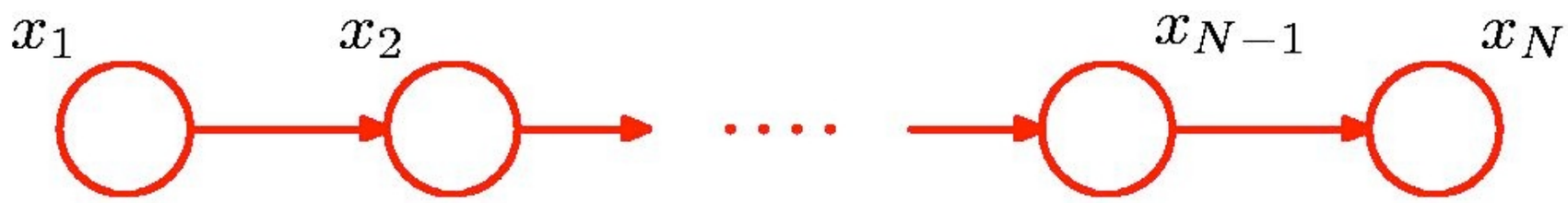
- In general, a Markov Random Field is factorized as

$$p(\mathbf{x}) = \frac{\prod_C \phi_C(\mathbf{x}_C)}{\sum_{\mathbf{x}'} \prod_C \phi_C(\mathbf{x}'_C)} = \frac{1}{Z} \prod_C \phi_C(\mathbf{x}_C) \qquad (4.1)$$

- where C is the set of all (maximal) cliques and Φ_C is a positive function of a given clique $\mathbf{x}_C$ of nodes, called the **clique potential**. Z is called the **partition function**.

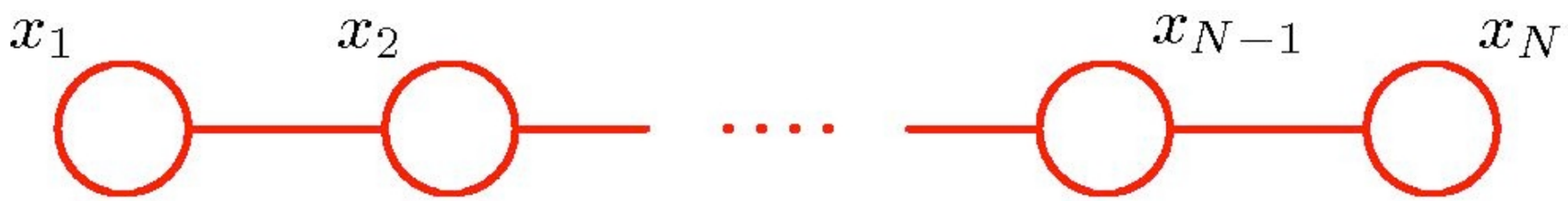
- Theorem (Hammersley/Clifford): Any undirected model with associated clique potentials Φ_C is a perfect map for the probability distribution defined by Equation (4.1).
- As a conclusion, all probability distributions that can be factorized as in (4.1), can be represented as an MRF.

Converting Directed to Undirected Graphs (1)



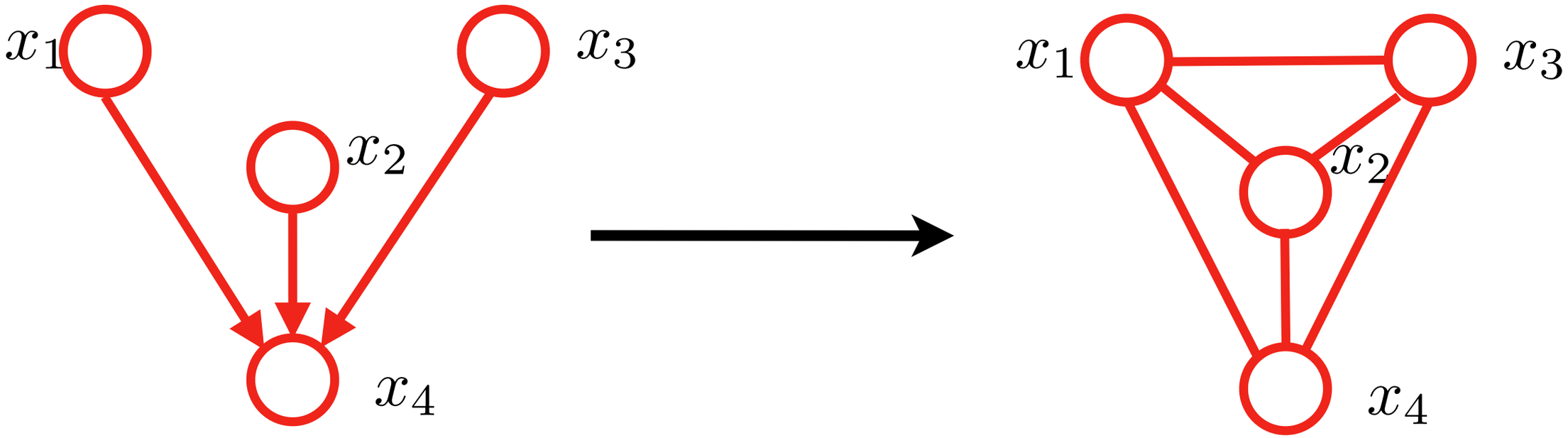
$$p(\mathbf{x}) = p(x_1)p(x_2|x_1)\; p(x_3|x_2) \cdots p(x_N|x_{N-1})$$

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



In this case: Z=1

Converting Directed to Undirected Graphs (2)



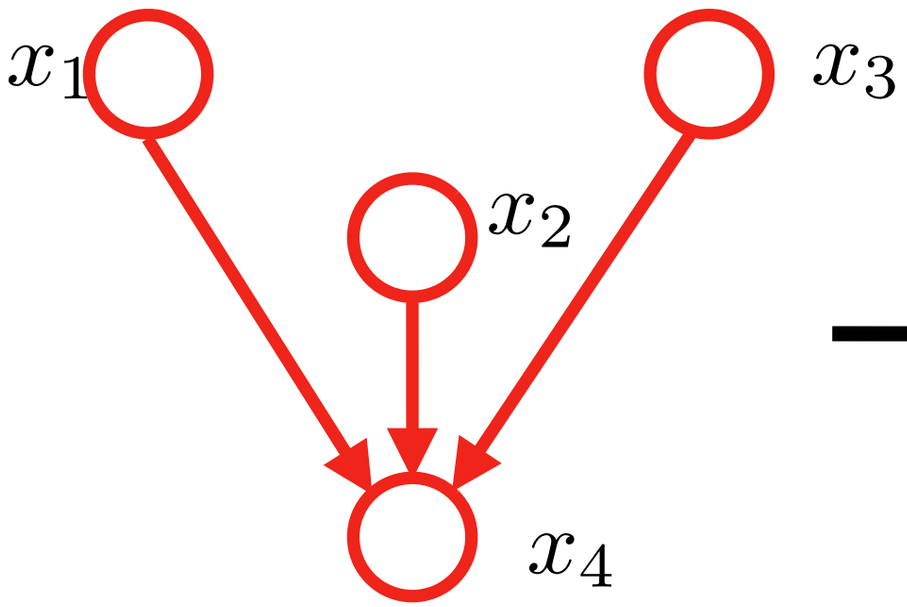
$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_2)p(x_4 \mid x_1, x_2, x_3)$$

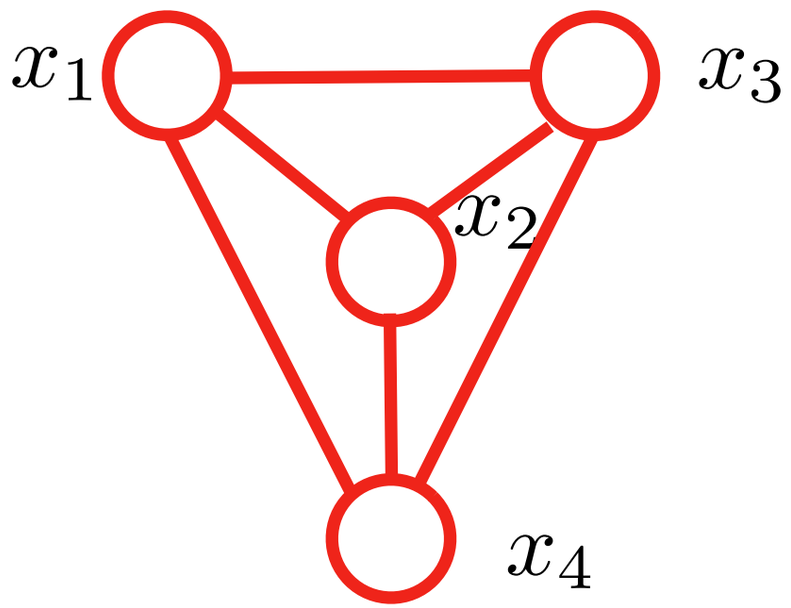
In general: conditional distributions in the directed graph are mapped to cliques in the undirected graph

However: the variables are **not** conditionally independent given the head-to-head node

Therefore: Connect all parents of head-to-head nodes with each other (**moralization**)

Converting Directed to Undirected Graphs (2)





$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_2)p(x_4 \mid x_1, x_2, x_3)$$

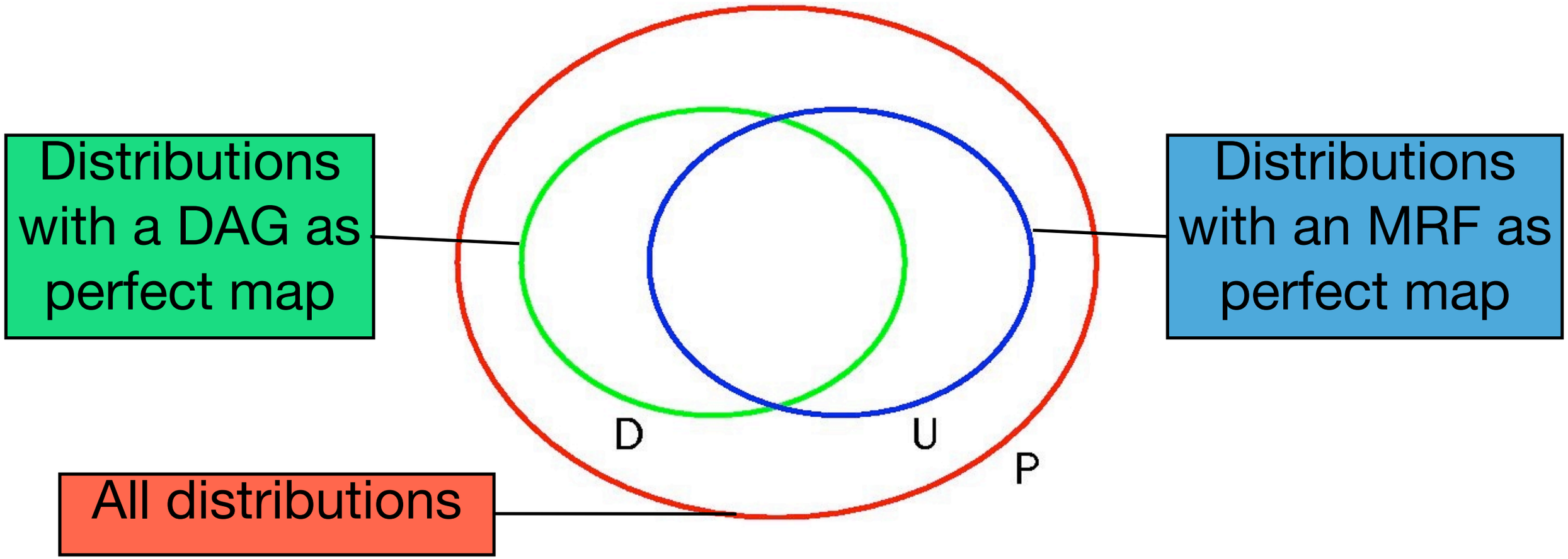
$$p(\mathbf{x}) = \phi(x_1, x_2, x_3, x_4)$$

Problem: This process can remove conditional independence relations (inefficient)

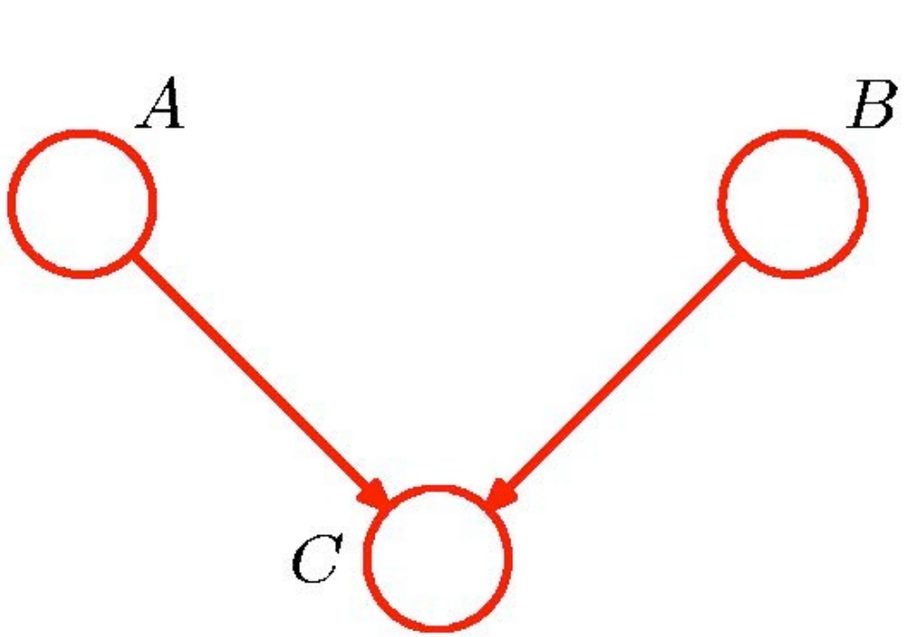
Generally: There is no one-to-one mapping between the distributions represented by directed and by undirected graphs.

Representability

- As for DAGs, we can define an I-map, a D-map and a perfect map for MRFs.
- The set of all distributions for which a DAG exists that is a perfect map is different from that for MRFs.

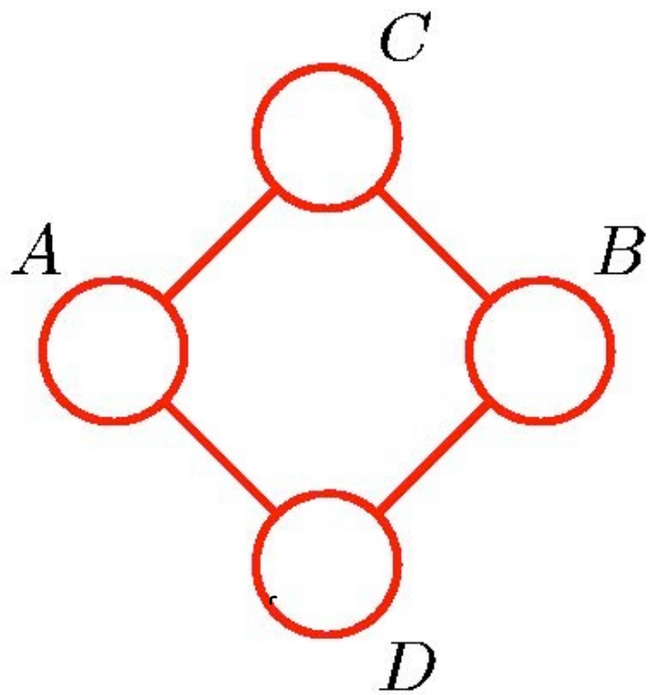


Directed vs. Undirected Graphs



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

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



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

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

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

Both distributions can not be represented in the other framework (directed/undirected) with all conditional independence relations.

Using Graphical Models

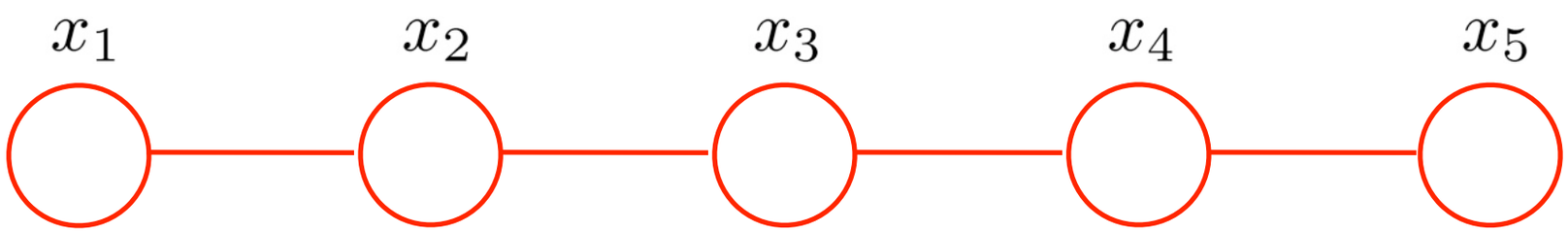
We can use a graphical model to do **inference**:

- Some nodes in the graph are **observed**, for others we want to find the posterior distribution
- Also, computing the local **marginal distribution** $p(x_n)$ at any node x_n can be done using inference.

Question: How can inference be done with a graphical model?

We will see that when exploiting conditional independences we can do efficient inference.

Inference on a Chain



The joint probability is given by

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)$$

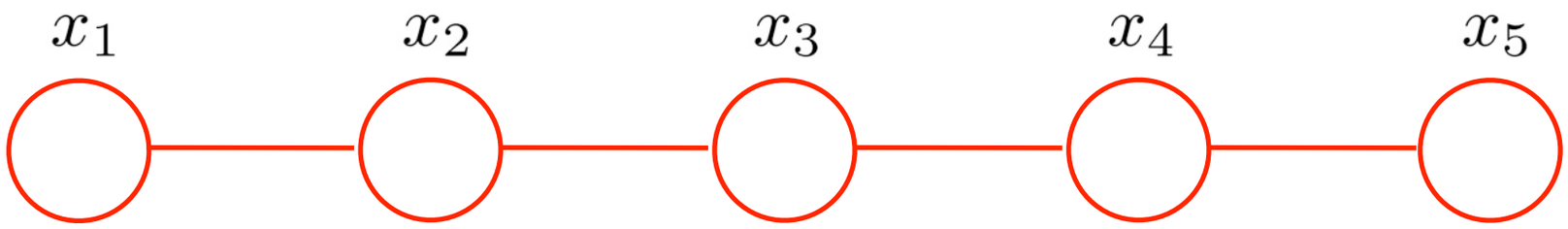
The marginal at x_3 is $$p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(\mathbf{x})$$

In the general case with N nodes we have

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

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

Inference on a Chain



$$p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(\mathbf{x})$$

- This would mean K^N computations! A more efficient way is obtained by rearranging:

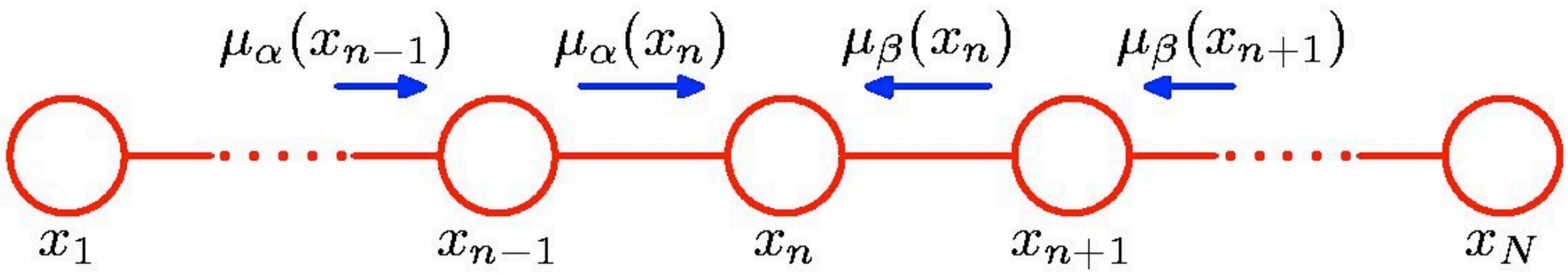
$$p(x_3) = \frac{1}{Z}\sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} \psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)$$

$$= \frac{1}{Z}\sum_{x_2}\sum_{x_1}\sum_{x_4}\sum_{x_5} \psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)$$

$$= \frac{1}{Z}\sum_{x_2}\psi_{2,3}(x_2,x_3)\sum_{x_1}\psi_{1,2}(x_1,x_2)\sum_{x_4}\psi_{3,4}(x_3,x_4)\sum_{x_5}\psi_{4,5}(x_4,x_5)$$

$\mu_\alpha(x_3)$ ← Vectors of size K → $\mu_\beta(x_3)$

Inference on a Chain



In general, we have

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

Inference on a Chain

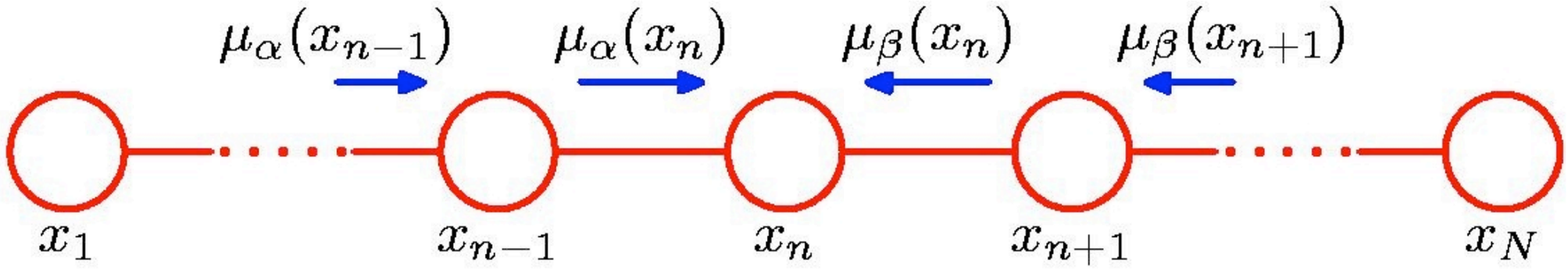
The messages μ_α and μ_β can be computed recursively:

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

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

Computation of μ_α starts at the first node and computation of μ_β starts at the last node.

Inference on a Chain



- The first values of μ_α and μ_β are:

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

- The partition function can be computed at any node:

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

- Overall, we have $O(NK^2)$ operations to compute the marginal $p(x_n)$

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 **once** at a node x_m: $Z = \sum_{x_m} \mu_\alpha(x_m)\mu_\beta(x_m)$
- Compute

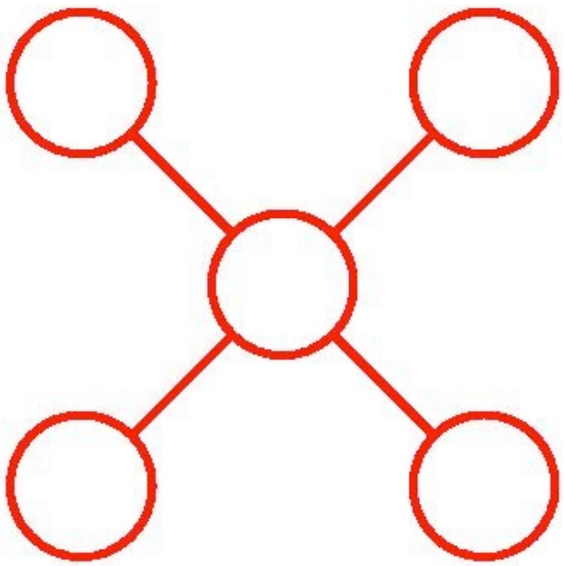
$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

for all variables required.

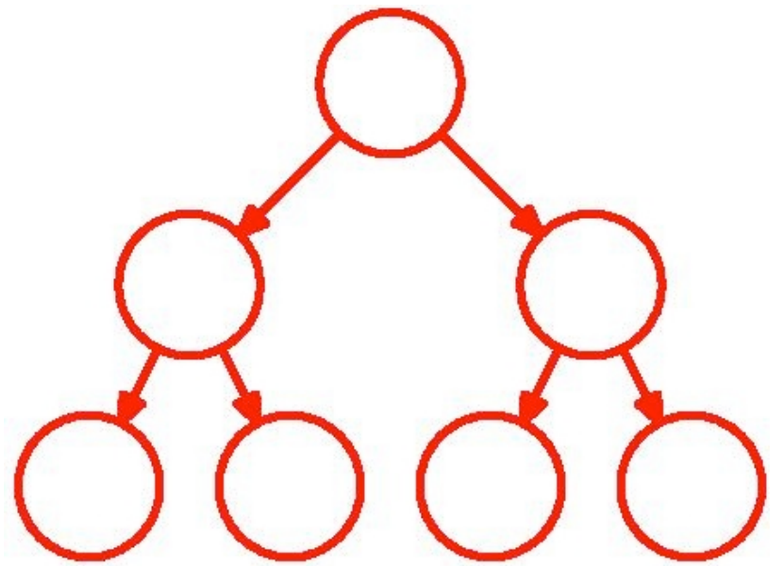
More General Graphs

The message-passing algorithm can be extended to more general graphs:

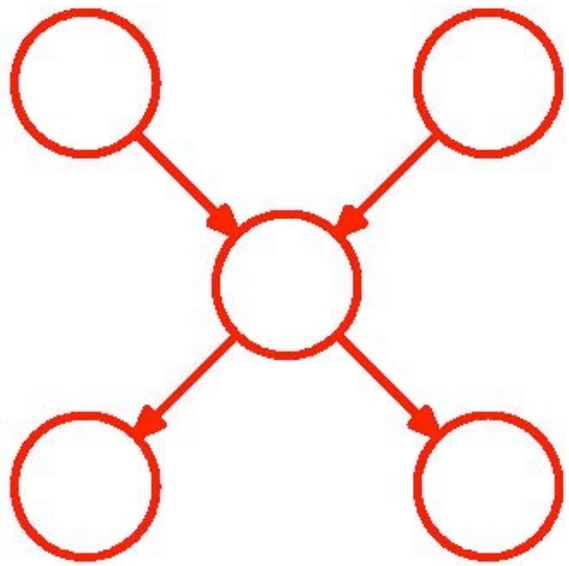
Undirected Tree



Directed Tree



Polytree



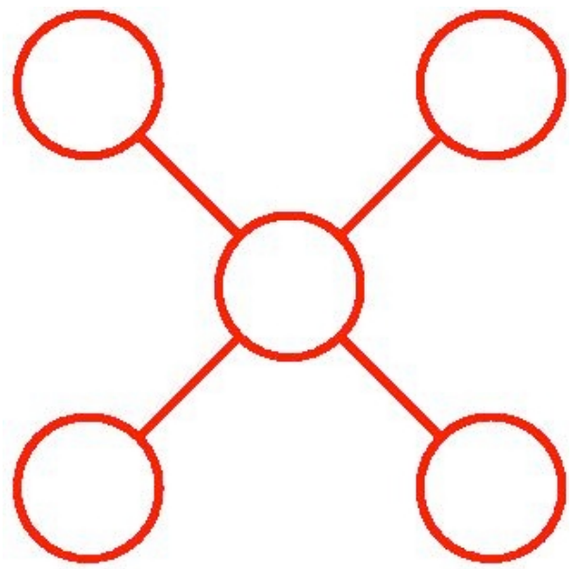
It is then known as the **sum-product algorithm.**

A special case of this is **belief propagation**.

More General Graphs

The message-passing algorithm can be extended to more general graphs:

Undirected Tree



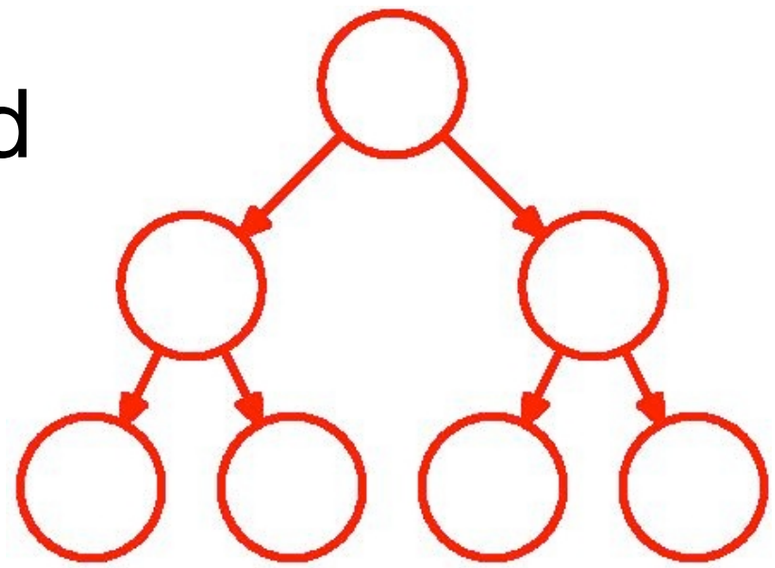
An **undirected tree** is defined as a graph that has exactly one path between any two nodes

More General Graphs

The message-passing algorithm can be extended to more general graphs:

A **directed tree** has only one node without parents and all other nodes have exactly one parent

Directed Tree



Conversion from a directed to an undirected tree is no problem, because no links are inserted

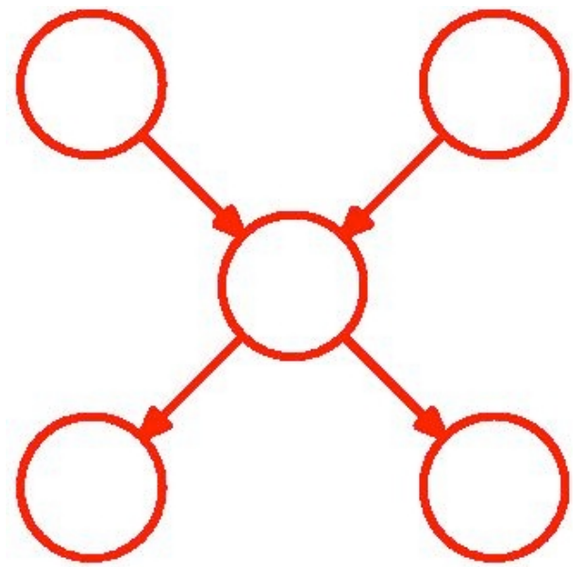
The same is true for the conversion back to a directed tree

More General Graphs

The message-passing algorithm can be extended to more general graphs:

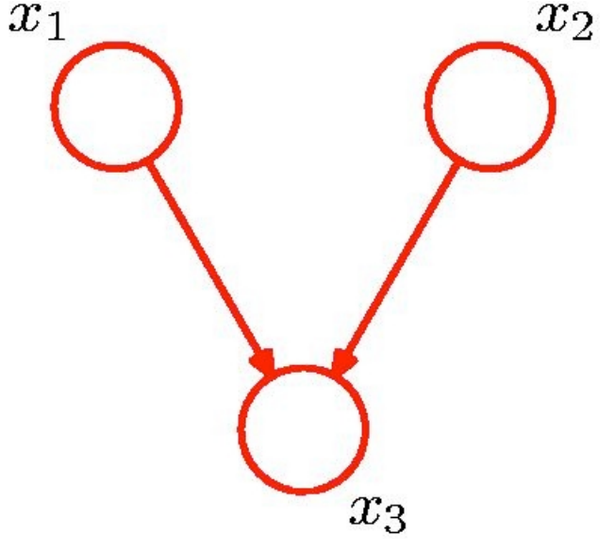
Polytrees can contain nodes with several parents, therefore moralization can remove independence relations

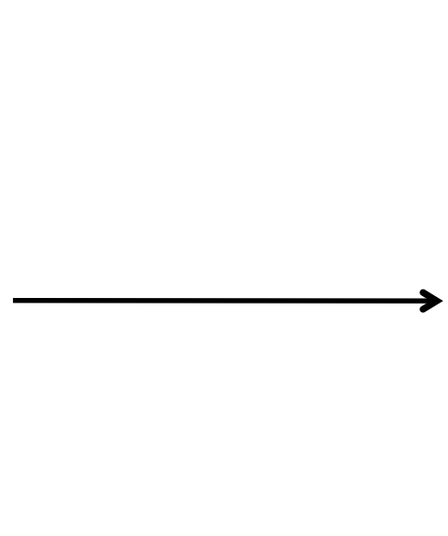
Polytree

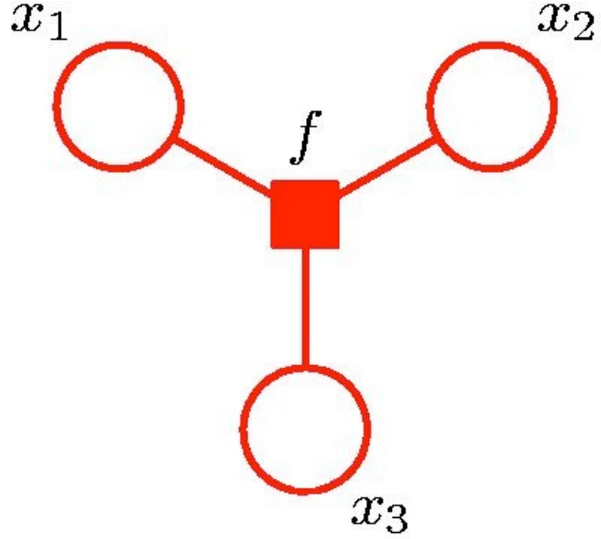


Factor Graphs

- The Sum-product algorithm can be used to do inference on undirected and directed graphs.
- A representation that generalizes directed and undirected models is the **factor graph**.







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

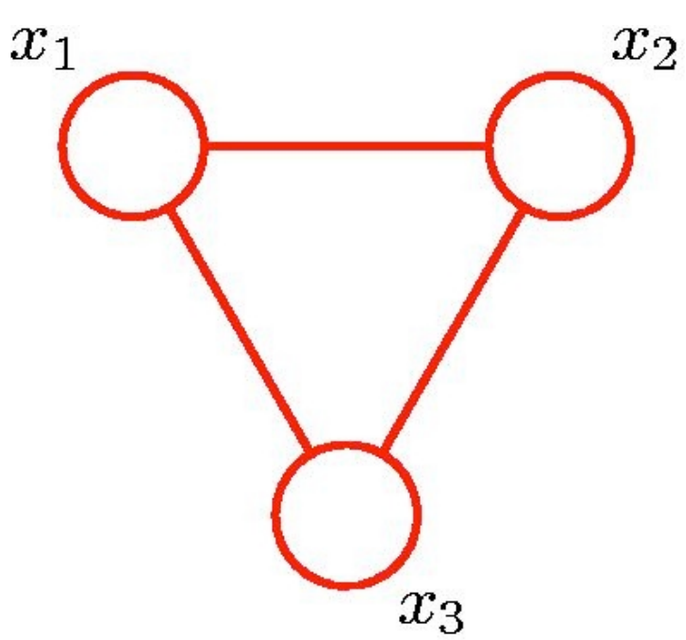
Directed graph

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

Factor graph

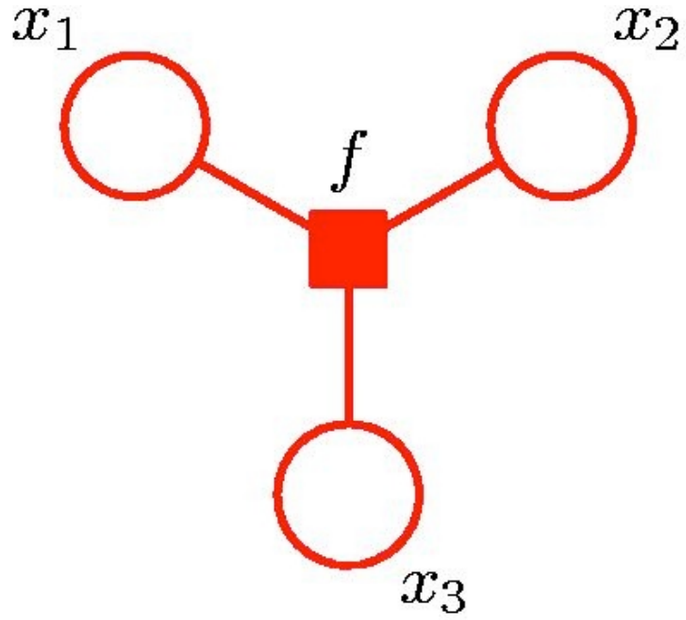
Factor Graphs

- The Sum-product algorithm can be used to do inference on undirected and directed graphs.
- A representation that generalizes directed and undirected models is the **factor graph**.



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

Undirected graph



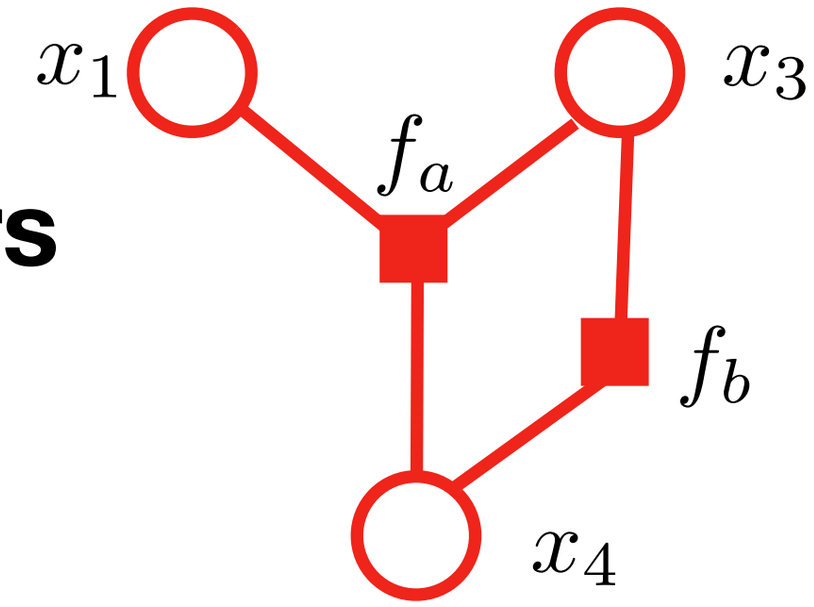
$$f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$$

Factor graph

Factor Graphs

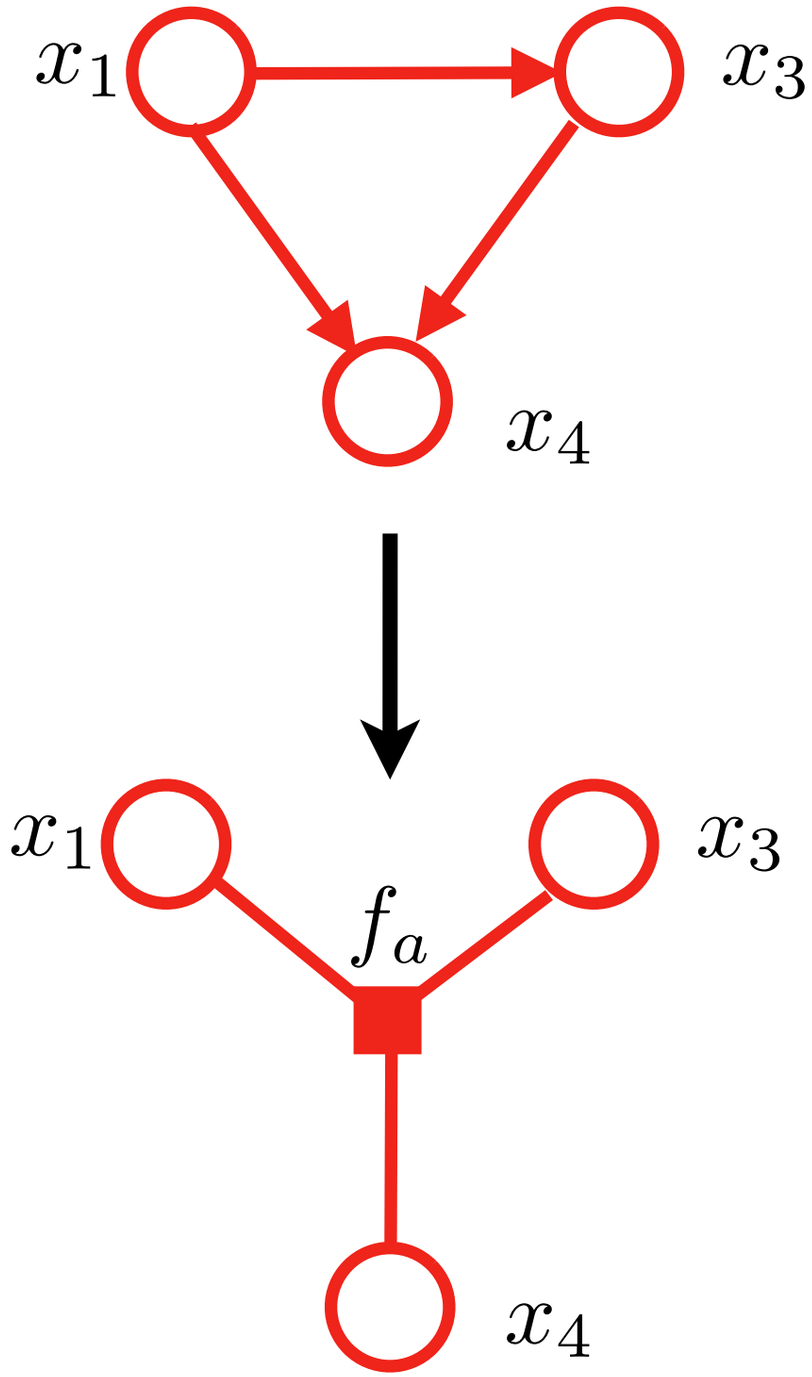
Factor graphs

- can contain **multiple factors** for the same nodes
- are more general than undirected graphs
- are **bipartite**, i.e. they consist of two kinds of nodes and all edges connect nodes of different kind



Factor Graphs

- Directed trees convert to tree-structured factor graphs
- The same holds for undirected trees
- Also: directed polytrees convert to tree-structured factor graphs
- And: Loops in a directed graph can be removed by converting to a factor graph



The Sum-Product Algorithm

Assumptions:

- all variables are discrete
- the factor graph has a tree structure

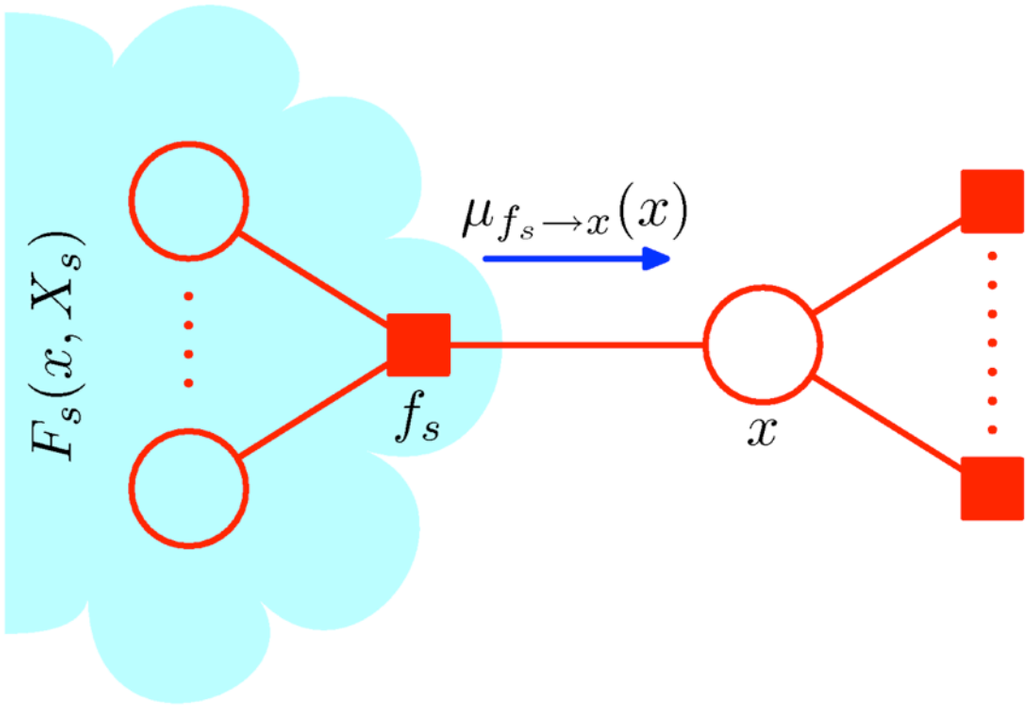
The factor graph represents the joint distribution as a product of factor nodes:

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

The marginal distribution at a given node x is

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

The Sum-Product Algorithm



For a given node x the joint can be written as

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

Product of all factors associated with f_s

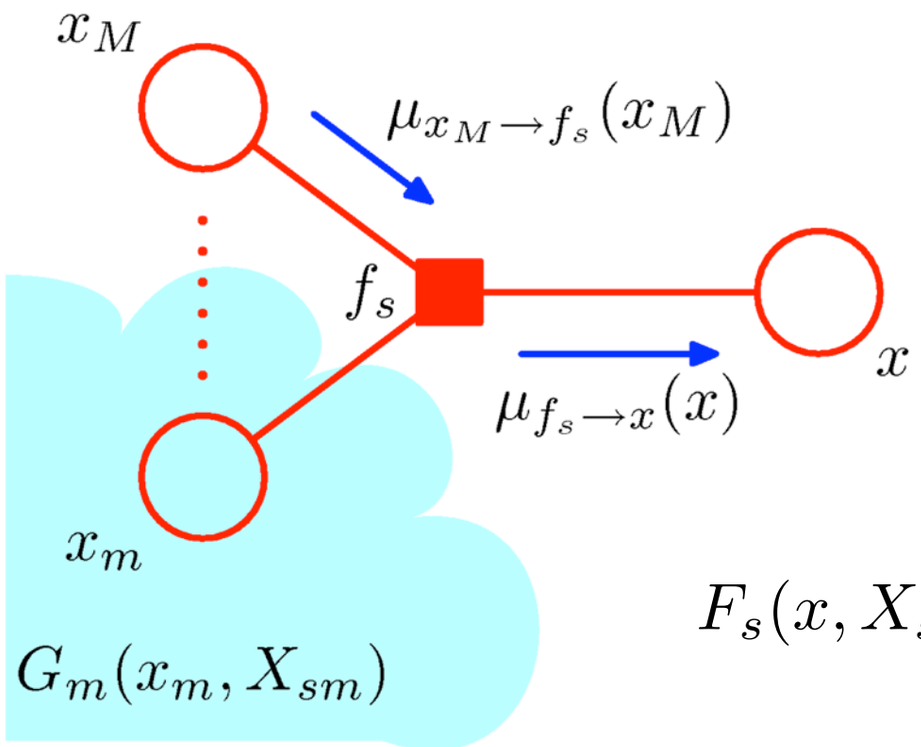
Thus, we have $p(x) = \sum_{\mathbf{x} \setminus x} \prod_{s \in \mathrm{ne}(x)} F_s(x, X_s)$

Key insight: Sum and product can be exchanged!

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

"Messages from factors to node x"

The Sum-Product Algorithm



The factors in the messages can be factorized further:

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s_1}) \ldots G_M(x_M, X_{s_M})$$

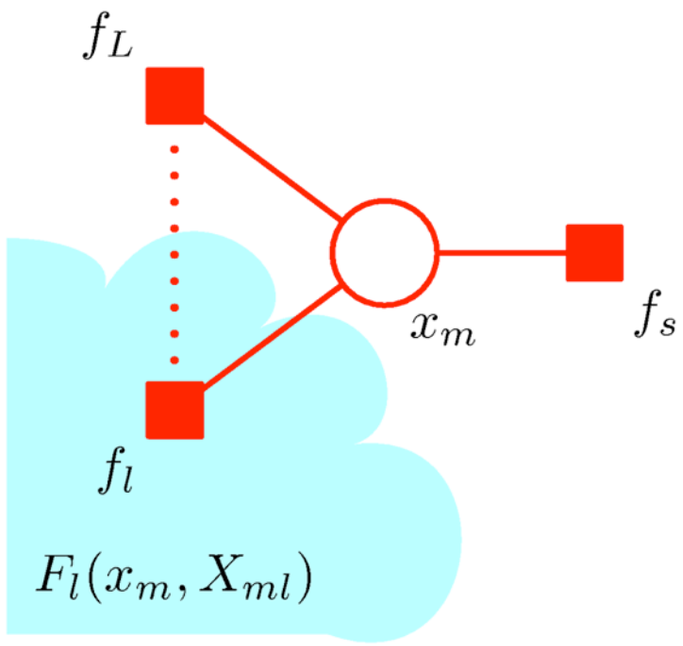
The messages can then be computed as

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

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

"Messages from nodes to factors"

The Sum-Product Algorithm



The factors G of the neighboring nodes can again be factorized further:

$$G_M(x_m, X_{s_m}) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} F_l(x_m, X_{m_l})$$

This results in the exact same situation as above!
We can now recursively apply the derived rules:

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

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

The Sum-Product Algorithm

Summary marginalization:

1. Consider the node x as a root note
2. Initialize the recursion at the leaf nodes as:

$$\mu_{f \to x}(x) = 1 \quad \text{(var) or} \quad \mu_{x \to f}(x) = f(x) \quad \text{(fac)}$$

3. Propagate the messages from the leaves to the root x
4. Propagate the messages back from the root to the leaves
5. We can get the marginals at every node in the graph by multiplying all incoming messages

The Max-Sum Algorithm

Sum-product is used to find the marginal distributions at every node, but:

How can we find the setting of all variables that **maximizes** the joint probability? And what is the value of that maximal probability?

Idea: use sum-product to find all marginals and then report the value for each node x that maximizes the marginal $p(x)$

However: this does not give the **overall** maximum of the joint probability

The Max-Sum Algorithm

Observation: the max-operator is distributive, just like the multiplication used in sum-product:

$$\max(ab, ac) = a \max(b, c) \qquad \text{if} \qquad a \geq 0$$

Idea: use max instead of sum as above and exchange it with the product

Chain example:

$$\max_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \ldots \max[\psi_{1,2}(x_1, x_2) \ldots \psi_{N-1,N}(x_{N-1}, x_N)]$$

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

Message passing can be used as above!

The Max-Sum Algorithm

To find the maximum value of $p(\mathbf{x})$, we start again at the leaves and propagate to the root.

Two problems:

- no summation, but many multiplications; this leads to numerical instability (very small values)
- when propagating back, multiple configurations of $\mathbf{x}$ can maximize $p(\mathbf{x})$, leading to wrong assignments of the overall maximum

Solution to the first:

Transform everything into log-space and use sums

The Max-Sum Algorithm

Solution to the second problem:

Keep track of the arg max in the forward step, i.e. store at each node which value was responsible for the maximum:

$$\phi(x_n) = \arg\max_{x_{n-1}} [\ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \to f_{n-1,n}}(x_n)]$$

Then, in the back-tracking step we can recover the arg max by recursive substitution of:

$$x_{n-1}^{\max} = \phi(x_n^{\max})$$

This is the Viterbi-Algorithm for HMMs!

Other Inference Algorithms

Junction Tree Algorithm:

- Provides exact inference on general graphs.
- Works by turning the initial graph into a **junction tree** and then running a sum-product-like algorithm
- A junction tree is obtained from an undirected model by **triangulation** and mapping cliques to nodes and connections of cliques to edges
- It is the maximal spanning tree of cliques

Problem: Intractable on graphs with large cliques.

Cost grows exponentially with the number of variables in the larges clique (“tree width”).

Other Inference Algorithms

Loopy Belief Propagation:

- Performs Sum-Product on general graphs, particularly when they have loops
- Propagation has to be done several times, until a convergence criterion is met
- No guarantee of convergence and no global optimum
- Messages have to be **scheduled**
- Initially, unit messages passed across all edges
- Approximate, but tractable for large graph

Conditional Random Fields

- A special case of MRFs is known as Conditional Random Field (CRF).
- CRFs are used for classification where labels are represented as discrete random variables $\mathbf{y}$ and features as continuous random variables $\mathbf{x}$
- A CRF represents the conditional probability

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{\prod_C \phi_C(\mathbf{x}_C, \mathbf{y}_C; \mathbf{w})}{\sum_{\mathbf{y}'} \prod_C \phi_C(\mathbf{x}_C, \mathbf{y}'_C; \mathbf{w})}$$

where $\mathbf{w}$ are parameters learned from training data.

Conditional Random Fields

Derivation of the formula for CRFs:

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})}{p(\mathbf{x} \mid \mathbf{w})} = \frac{p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})}{\sum_{y'} p(\mathbf{y}', \mathbf{x} \mid \mathbf{w})} = \frac{\prod_C \phi_C(\mathbf{x}_C, \mathbf{y}_C; \mathbf{w}) \quad Z}{Z \quad \sum_{\mathbf{y}'} \prod_C \phi_C(\mathbf{x}_C, \mathbf{y}'_C; \mathbf{w})}$$

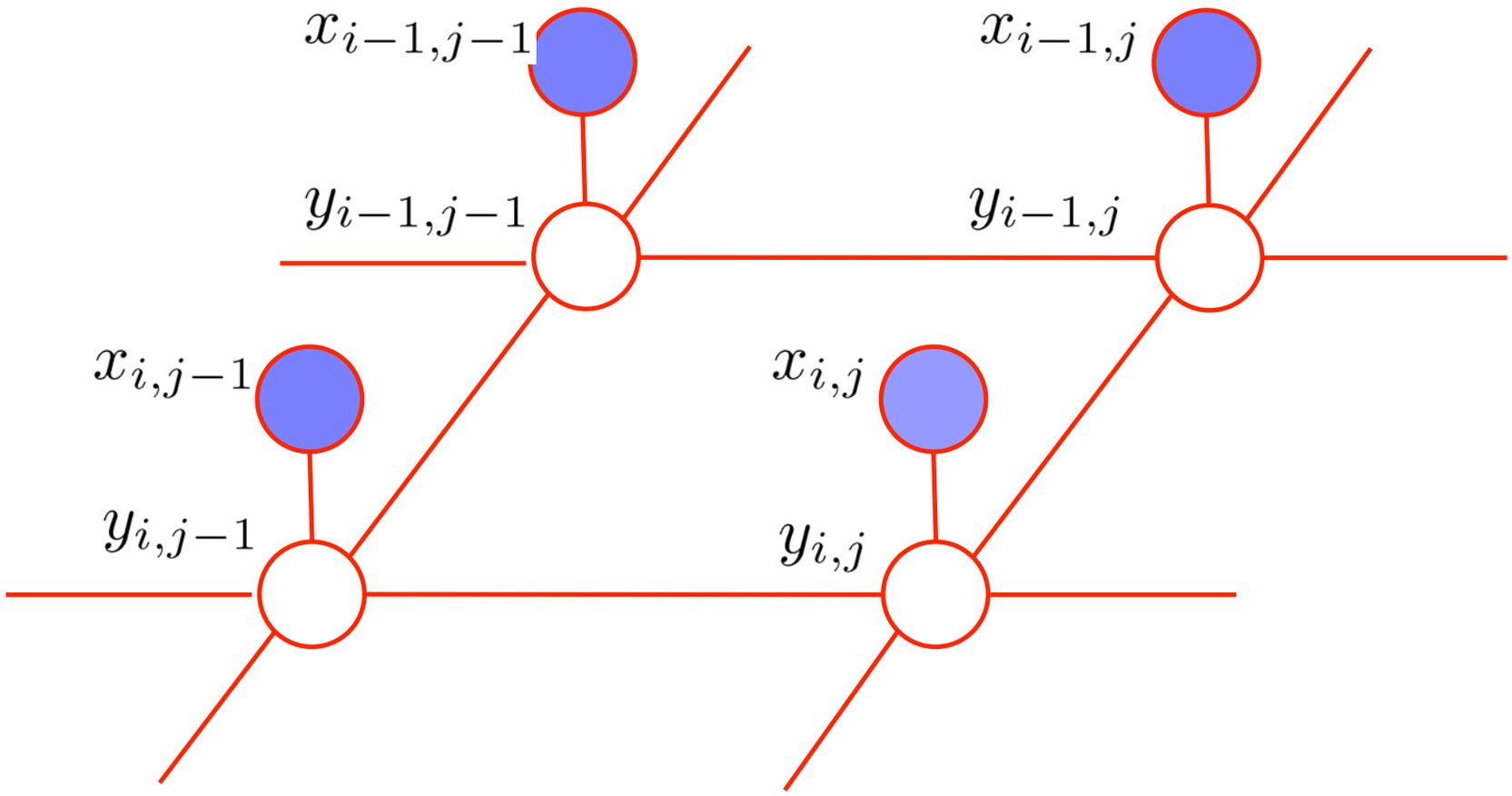
In the training phase, we compute parameters $\mathbf{w}$ that maximize the posterior:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}^*, \mathbf{y}^*) \propto p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) p(\mathbf{w})$$

where (x^*,y^*) is the training data and $p(w)$ is a Gaussian prior. In the inference phase we maximize

$$\arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}^*)$$

Conditional Random Fields



Typical example:
observed variables $x_{i,j}$ are intensity values of pixels in an image and **hidden** variables $y_{i,j}$ are object labels

Note: the definition of $x_{i,j}$ and $y_{i,j}$ is different from the one in C.M. Bishop (pg.389)!

CRF Training

We minimize the negative log-posterior:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \{-\ln p(\mathbf{w} \mid \mathbf{x}^*, \mathbf{y}^*)\} = \arg\min_{\mathbf{w}} \{-\ln p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) - \ln p(\mathbf{w})\}$$

Computing the likelihood is intractable, as we have to compute the partition function for each $\mathbf{w}$. We can approximate the likelihood using **pseudo-likelihood**:

where

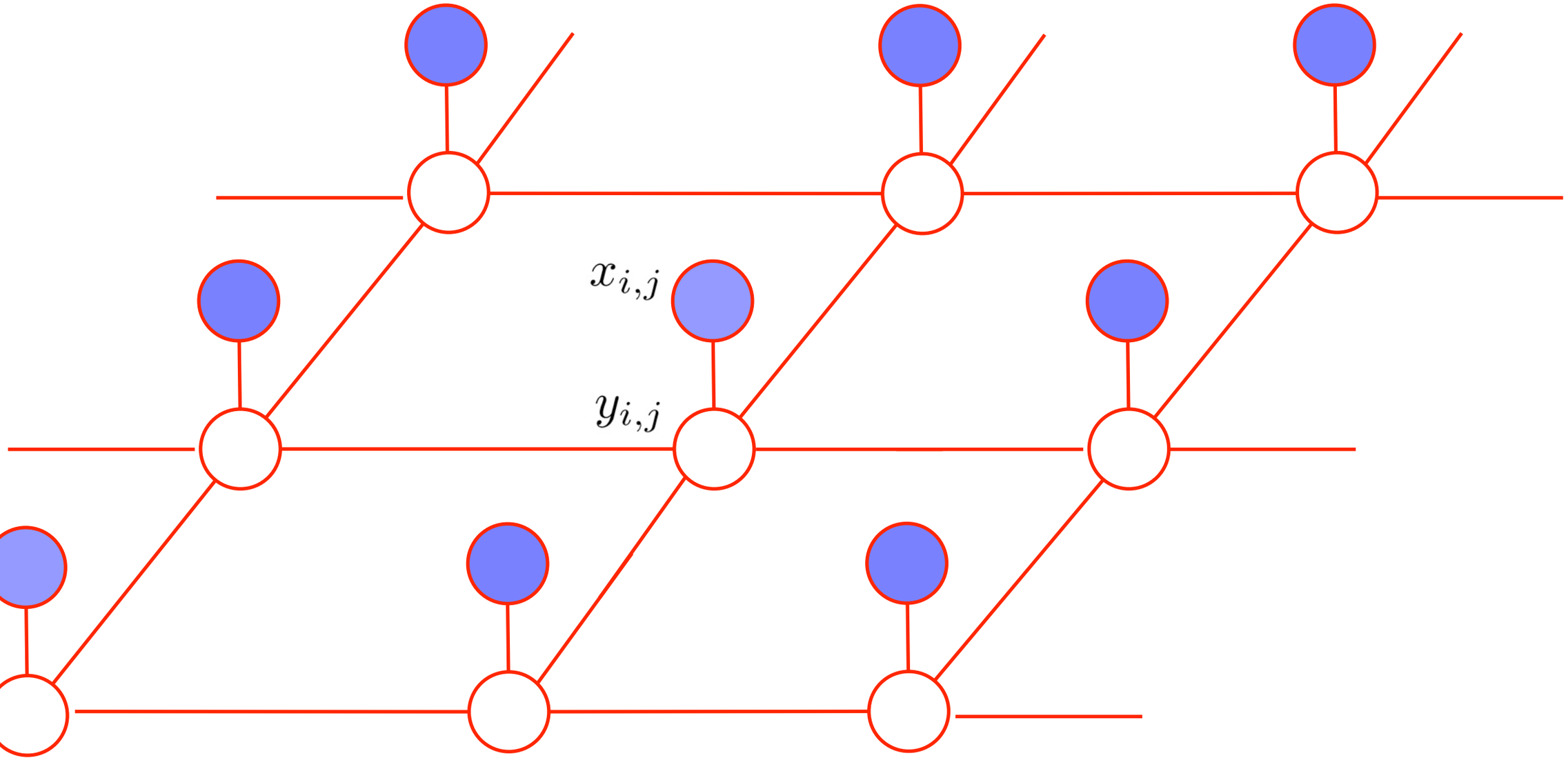
$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) \approx \prod_i p(y_i^* \mid \mathcal{M}(y_i^*), \mathbf{x}^*, \mathbf{w})$$

Markov blanket

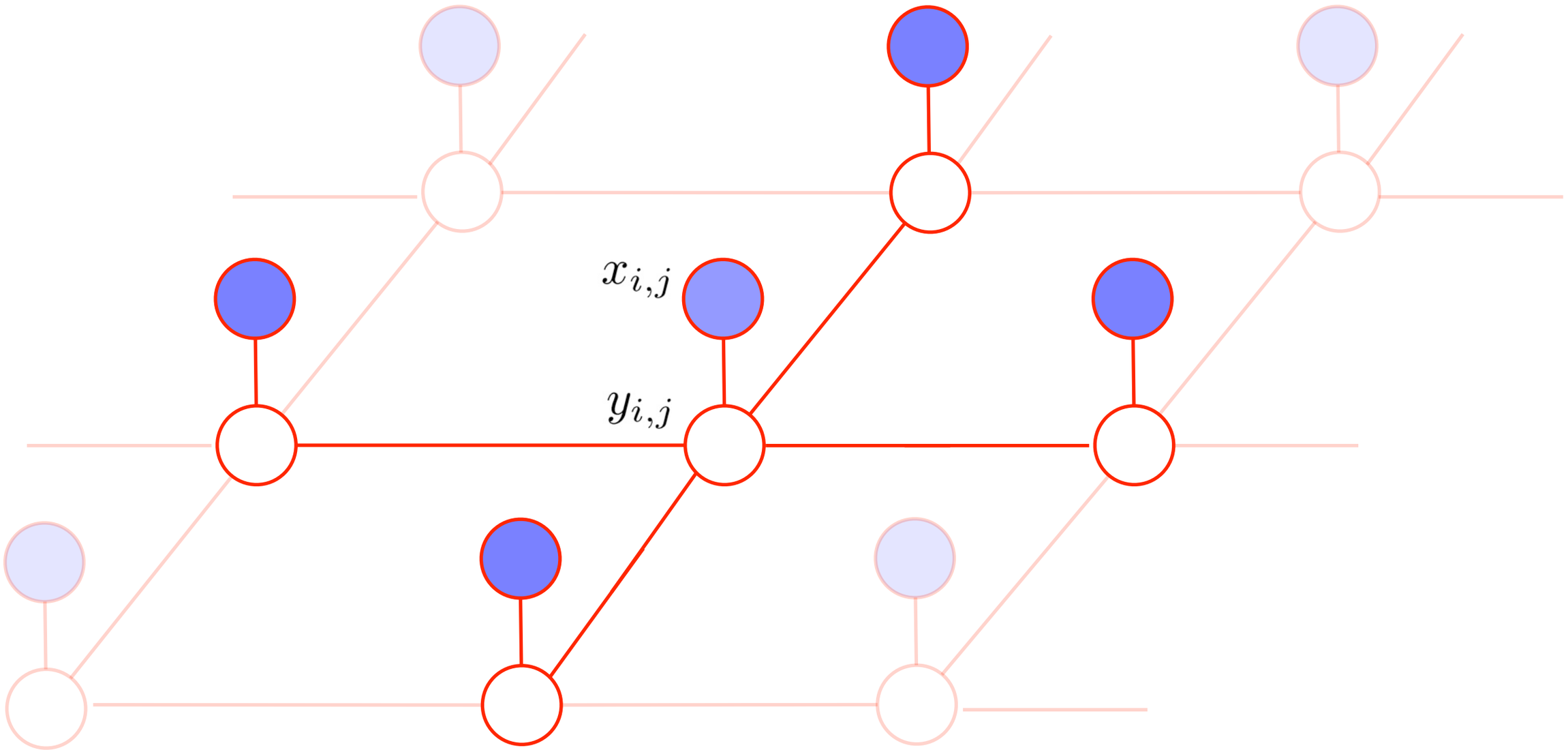
C_i: All cliques containing y_i

$$p(y_i^* \mid \mathcal{M}(y_i^*), \mathbf{x}^*, \mathbf{w}) = \frac{\prod_{C_i} \phi_{C_i}(\mathbf{x}_{C_i}^*, y_i^*, \mathbf{y}_{C_i}^*; \mathbf{w})}{\sum_{y_i'} \prod_{C_i} \phi_C(\mathbf{x}_{C_i}^*, y_i', \mathbf{y}_{C_i}^*; \mathbf{w})}$$

Pseudo Likelihood



Pseudo Likelihood



Pseudo-likelihood is computed only on the Markov blanket of y_i and its corresp. feature nodes.

Potential Functions

- The only requirement for the potential functions is that they are positive. We achieve that with:

$$\phi_C(\mathbf{x}_C, \mathbf{y}_C, \mathbf{w}) := \exp(\mathbf{w}^T f(\mathbf{x}_C, \mathbf{y}_C))$$

where f is a compatibility function that is large if the labels $\mathbf{y}_C$ fit well to the features $\mathbf{x}_C$.

- This is called the **log-linear model.**
- The function f can be, e.g. a local classifier

CRF Training and Inference

Training:

- Using pseudo-likelihood, training is efficient. We have to minimize:

$$L(\mathbf{w}) = -lpl(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) + \frac{1}{2\sigma^2}\mathbf{w}^T\mathbf{w}$$

Log-pseudo-likelihood

Gaussian prior

- This is a convex function that can be minimized using gradient descent

Inference:

- Only approximatively, e.g. using loopy belief propagation

Summary

- Undirected Graphical Models represent conditional independence more intuitively using graph separation
- Their factorization is done based on potential functions The normalizer is called the partition function, which in general is intractable to compute
- Inference in graphical models can be done efficiently using the sum-product algorithm (message passing).
- Another inference algorithm is loopy belief propagation, which is approximate, but tractable
- Conditional Random Fields are a special kind of MRFs and can be used for classification