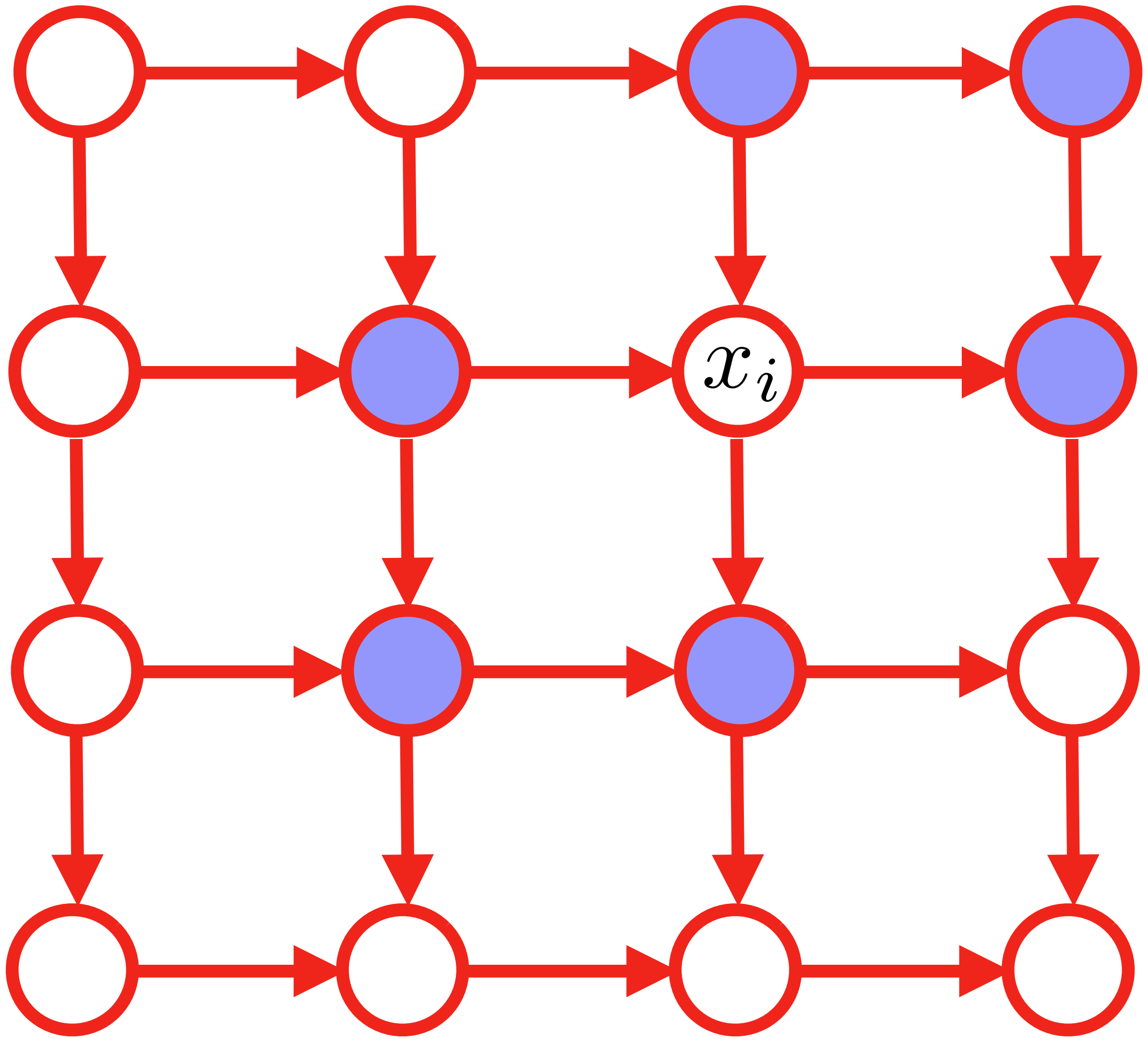
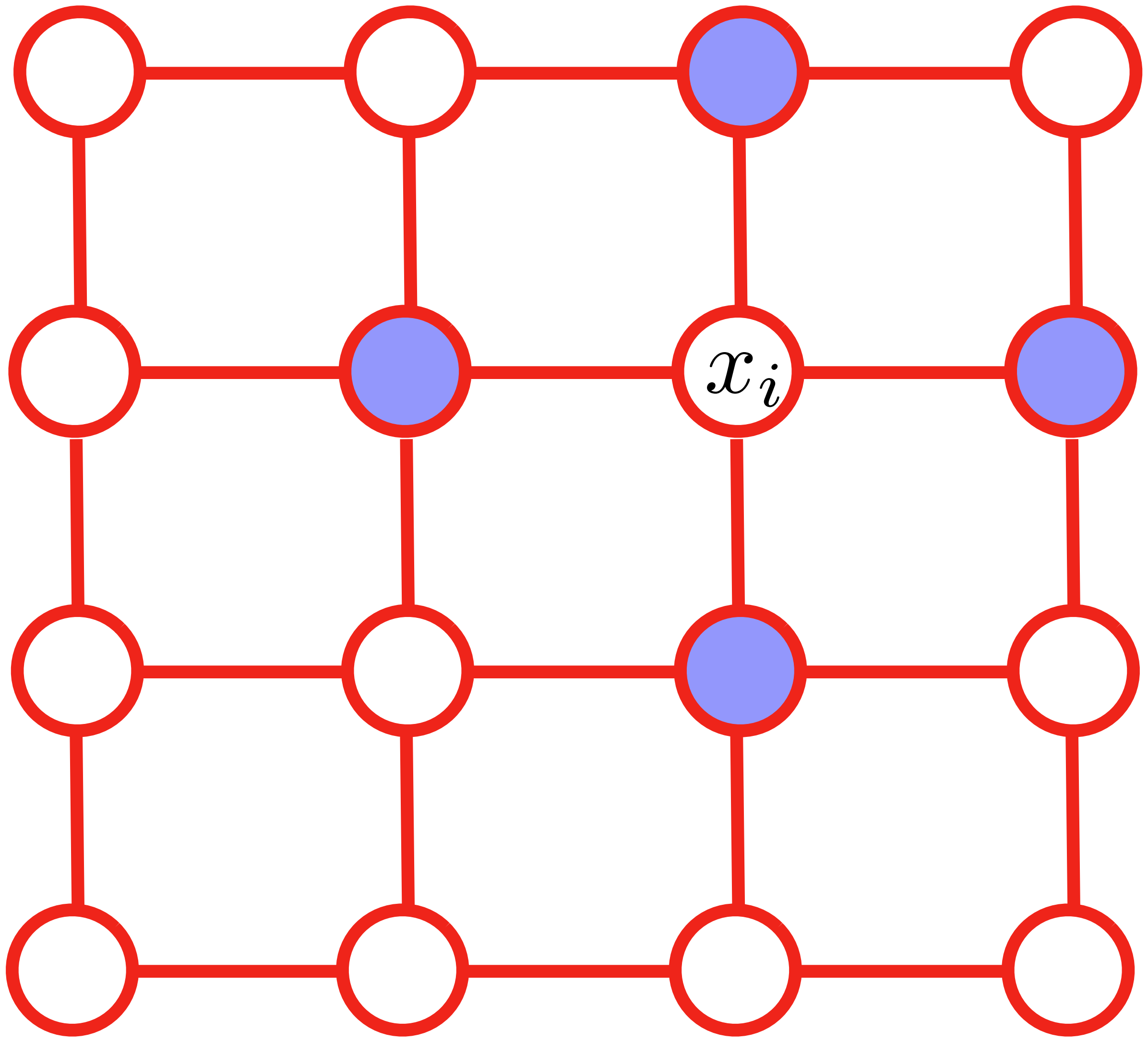
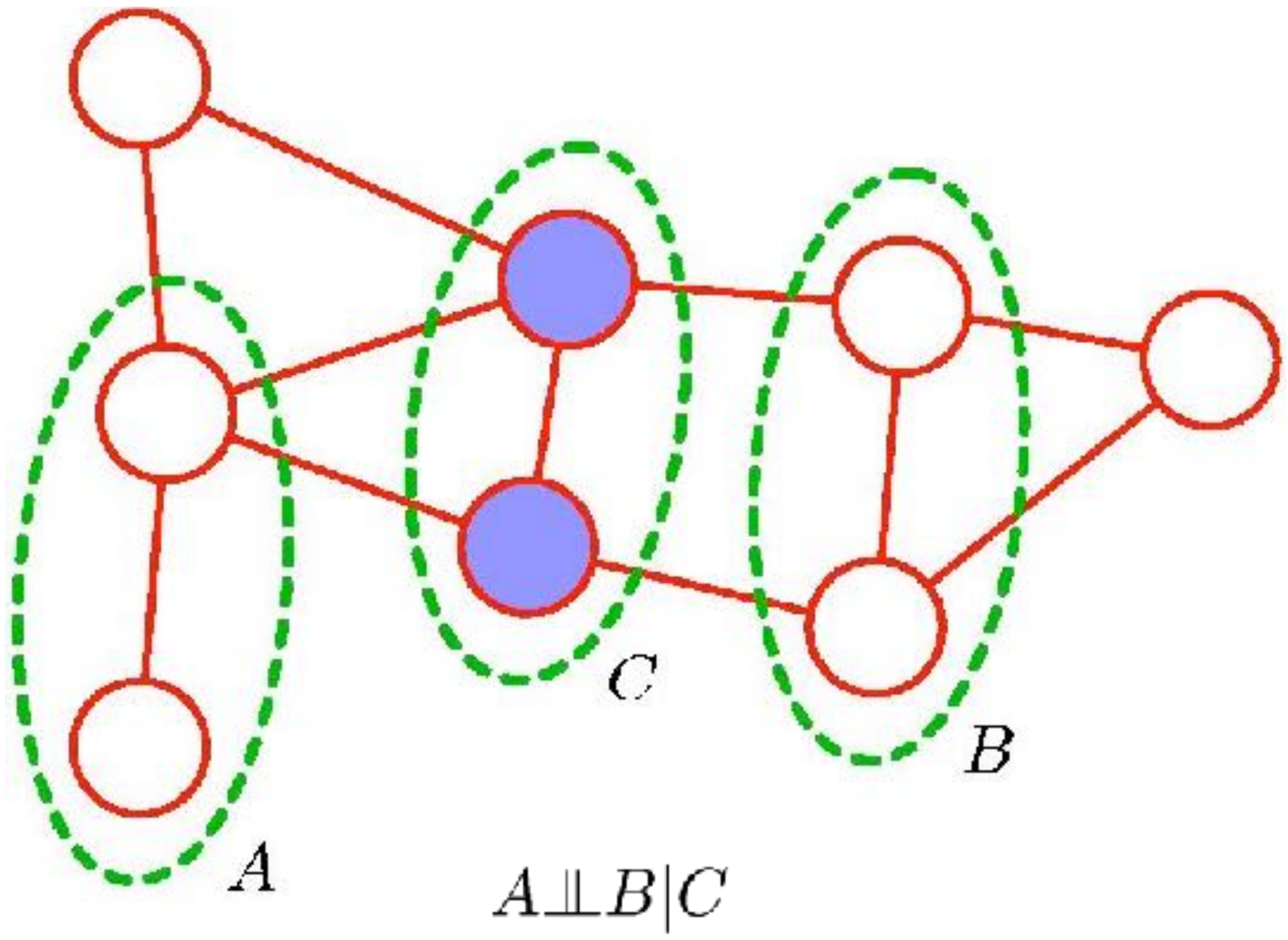
Example: Camera Image





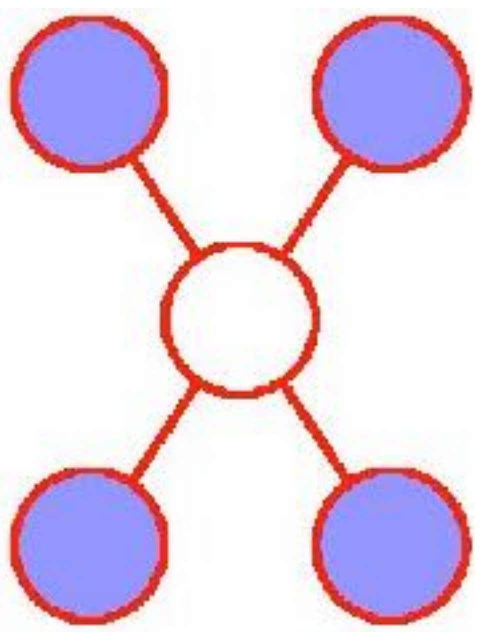
- directions are counter-intuitive for images
- Markov blanket is not just the direct neighbors when using a directed model

Markov Random Fields



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

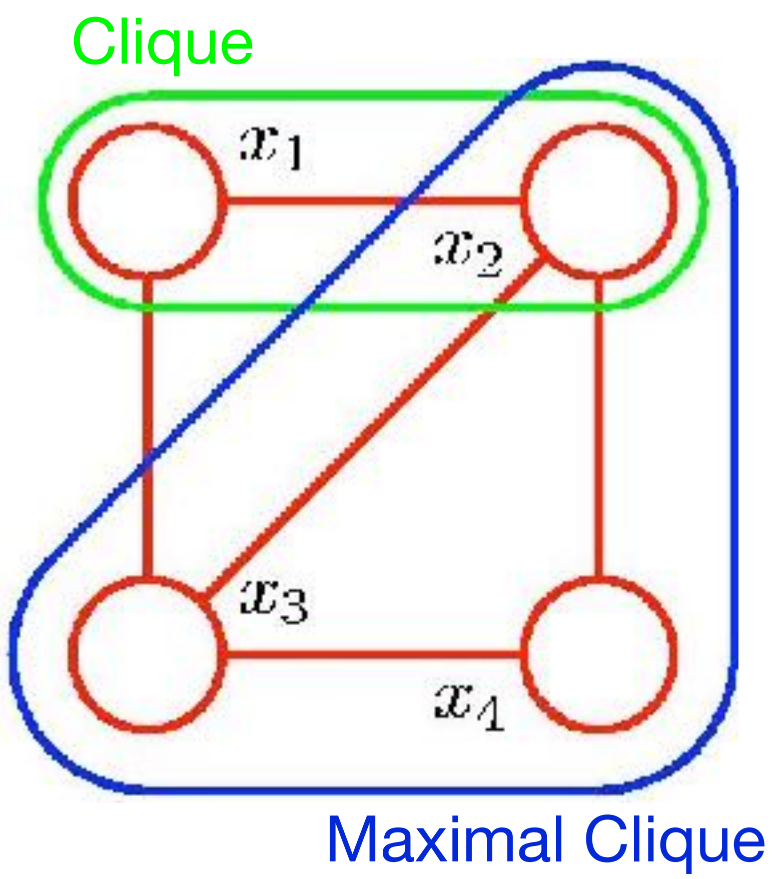
Any two nodes x_i and x_j that are not connected in an MRF are conditionally independent given all other nodes:

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

This means: 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

$$p(\mathbf{x}) = \frac{\prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c)}{\sum_{\mathbf{x}'} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}'_c)} = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c) \qquad (4.1)$$

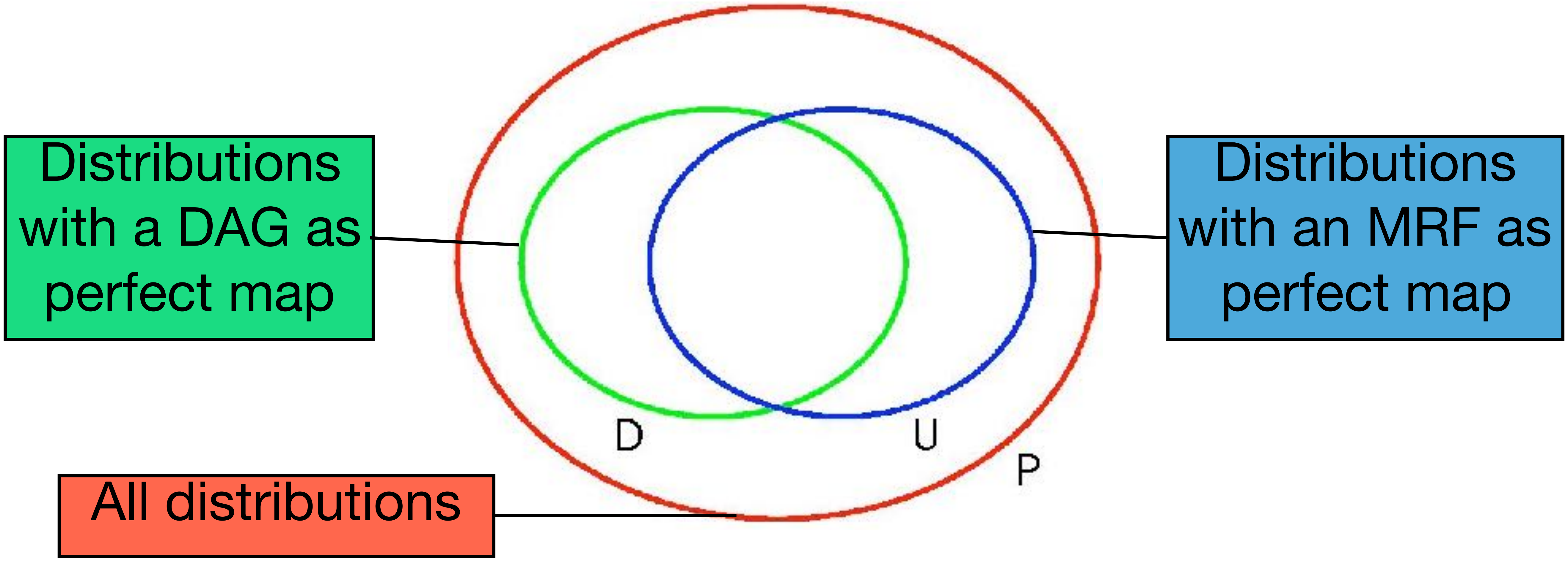
where $\mathcal{C}$ is the set of all (maximal) cliques and $\psi_c(\mathbf{x}_c)$ is a positive function of a given clique $\mathbf{x}_C$ of nodes, called the **clique potential**. Z is called the **partition function**.

Theorem (Hammersley/Clifford): Any undirected model with associated clique potentials ψ_c is a perfect map for the probability distribution defined by Equation (4.1).

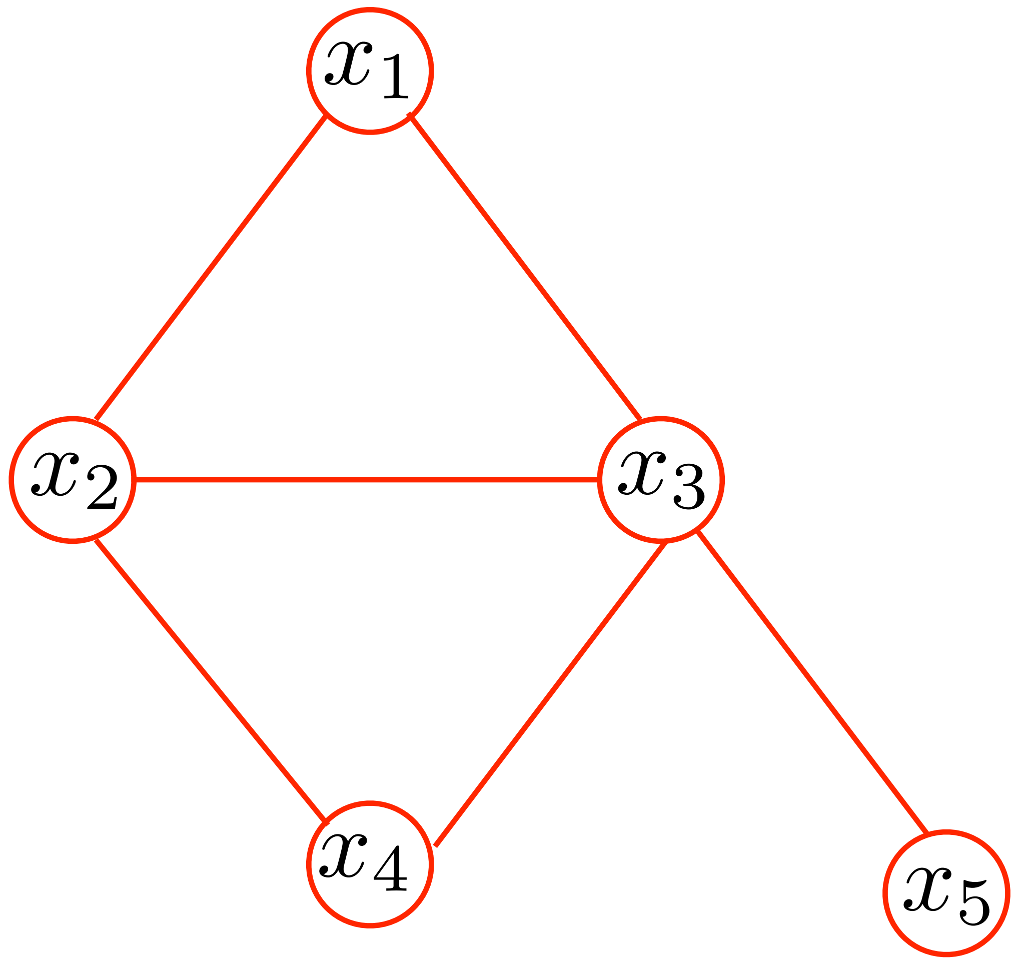
As a conclusion, all probability distributions that can be factorized as in (4.1), can be represented as an MRF.

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.


Distributions with a DAG as perfect map
Distributions with an MRF as perfect map
All distributions
D
U
P

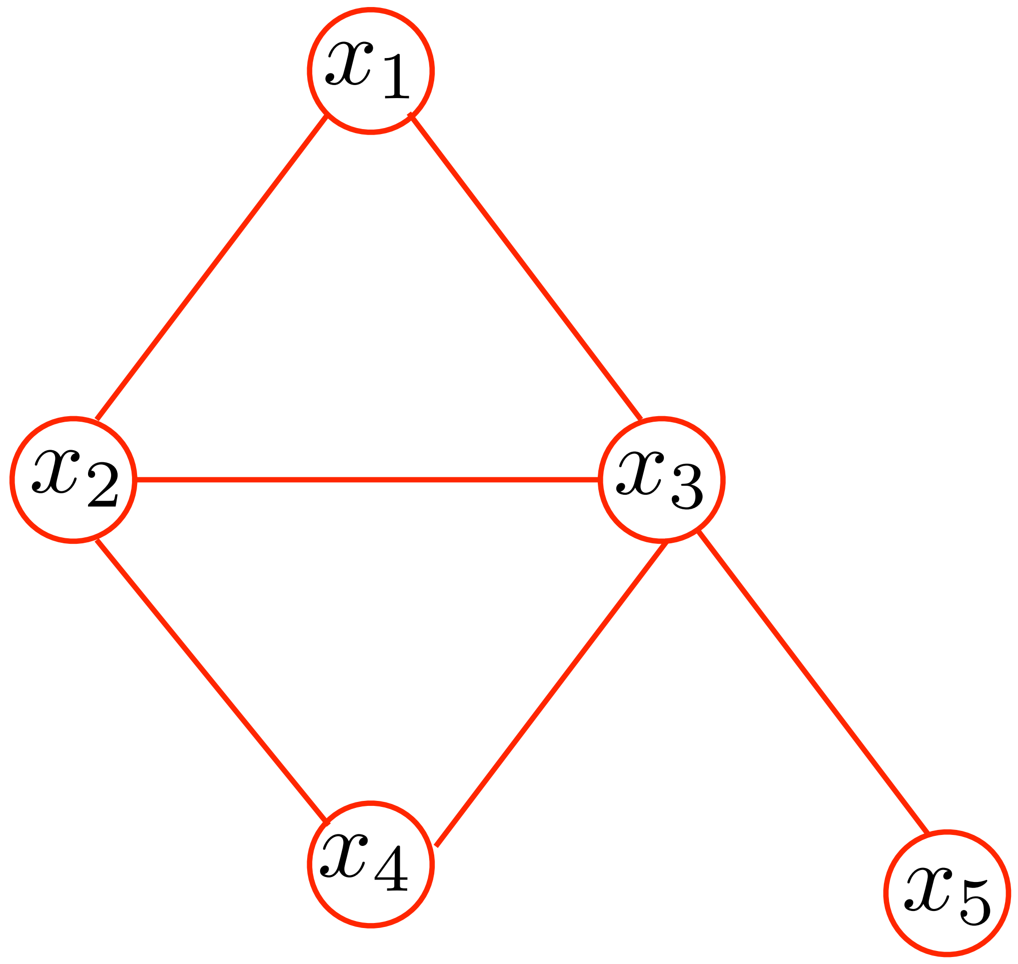
A Simple Example



If a distribution p satisfies all conditional independence relationships of this graph, then we can write p as

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{123}(x_1, x_2, x_3)\psi_{234}(x_2, x_3, x_4)\psi_{35}(x_3, x_5)$$

A Simple Example

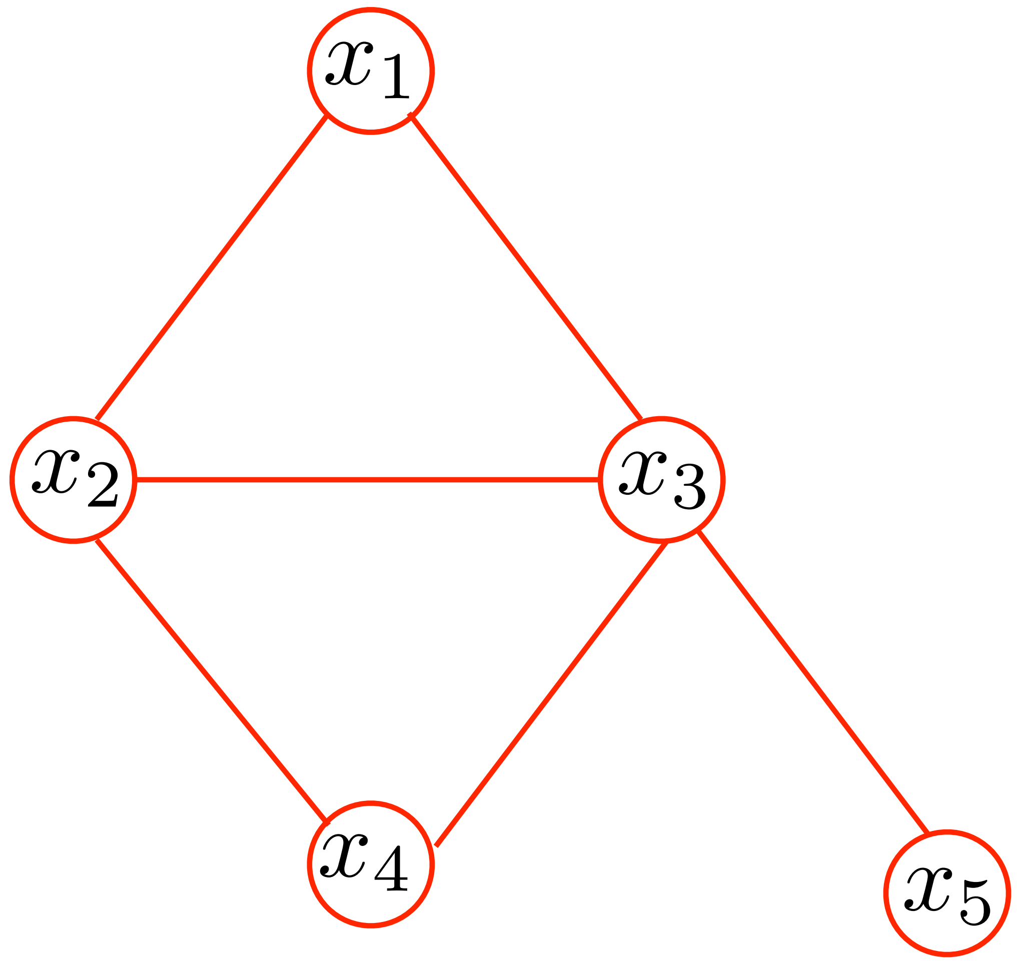


How to define the potentials?

- Intuitively, the potential of a clique should be high, iff the joint probability of the corres-ponding random variables is high.

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{123}(x_1, x_2, x_3)\psi_{234}(x_2, x_3, x_4)\psi_{35}(x_3, x_5)$$

A Simple Example



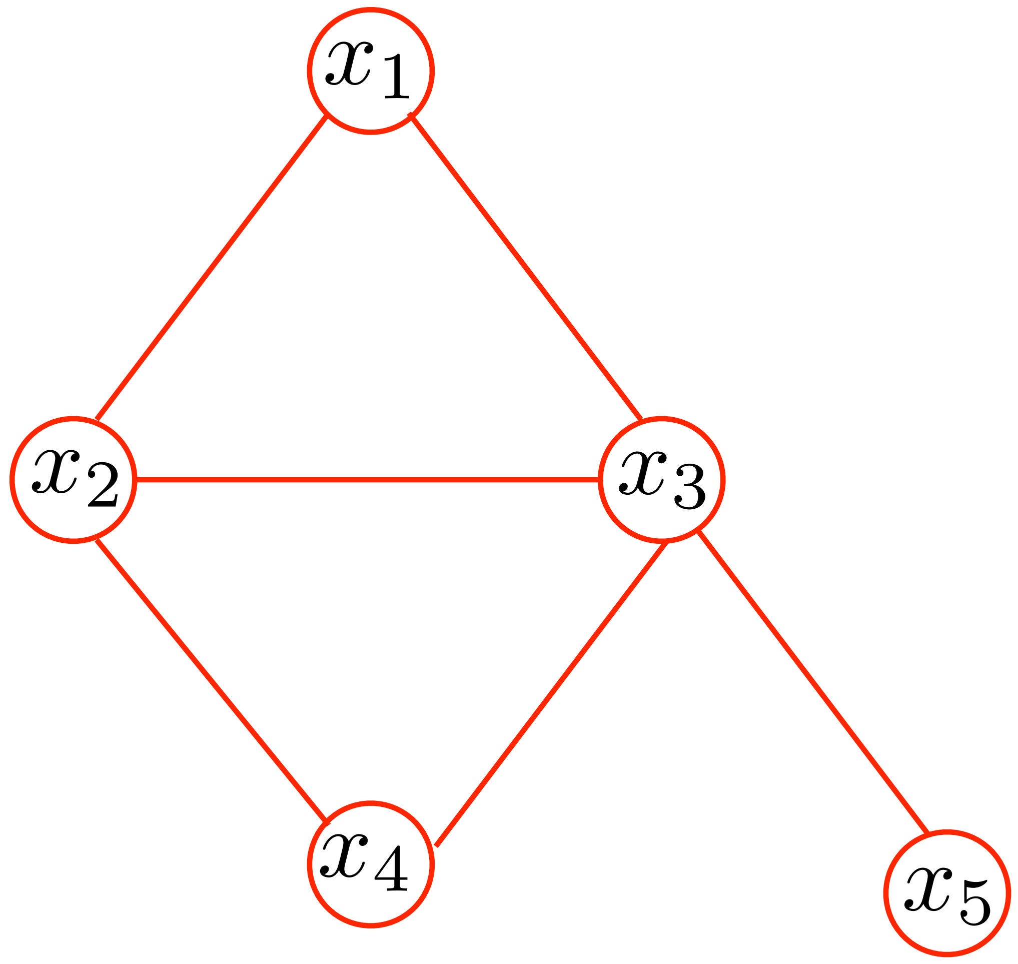
How to define the potentials?

- Intuitively, the potential of a clique should be high, iff the joint probability of the corres-ponding random variables is high.
- In most cases the potential is defined using a **log-linear** model:

$$\log \psi_c(\mathbf{x}_c) = \underbrace{\phi_c}_{\text{Feature function}}(\mathbf{x}_c)^T \underbrace{\boldsymbol{\theta}_c}_{\text{Parameters}}$$

Feature function **Parameters**

A Simple Example



How to define the potentials?

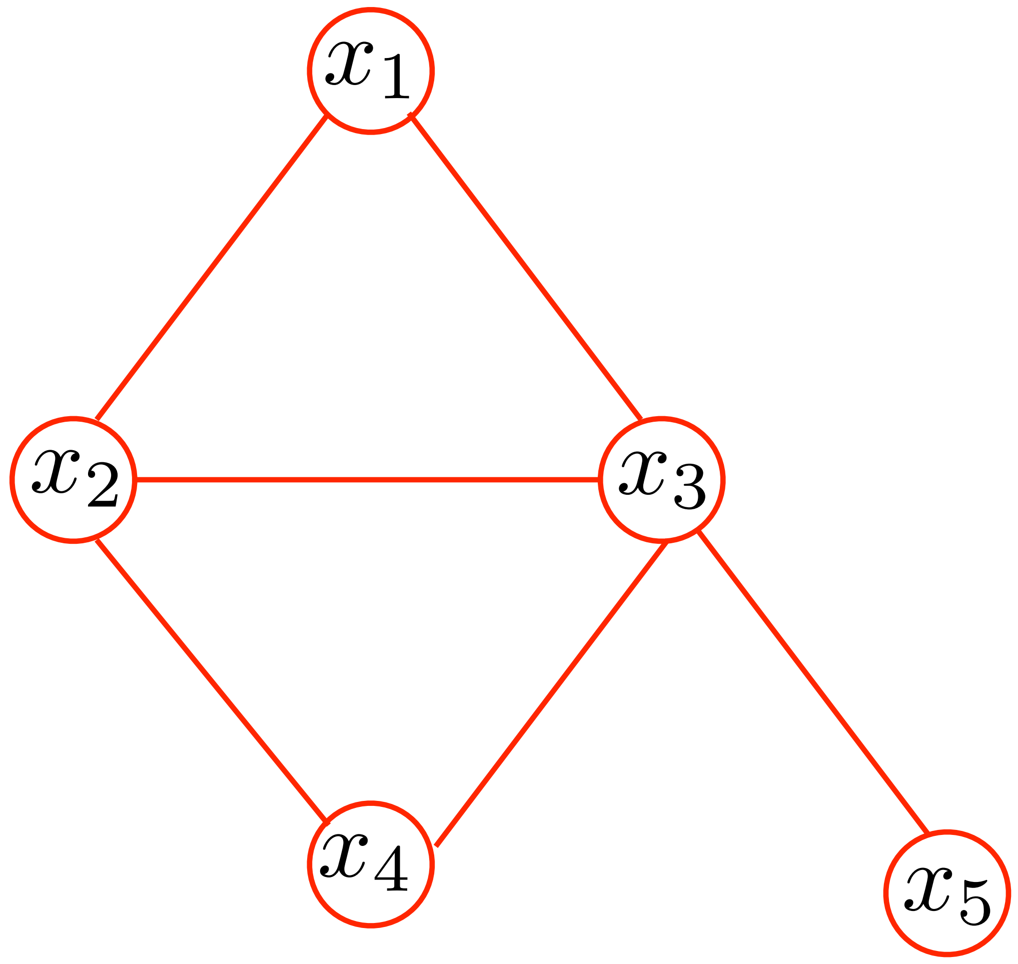
- Intuitively, the potential of a clique should be high, iff the joint probability of the corresponding random variables is high.
- In most cases the potential is defined using a **log-linear** model

$$\log \psi_c(\mathbf{x}_c) = \boldsymbol{\phi}_c(\mathbf{x}_c)^T \boldsymbol{\theta}_c$$

making the parameters explicit:

$$\Rightarrow \log p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{c \in \mathcal{C}} \boldsymbol{\phi}_c(\mathbf{x}_c)^T \boldsymbol{\theta}_c - \log Z(\boldsymbol{\theta})$$

A Simple Example



Using numbers, e.g.:

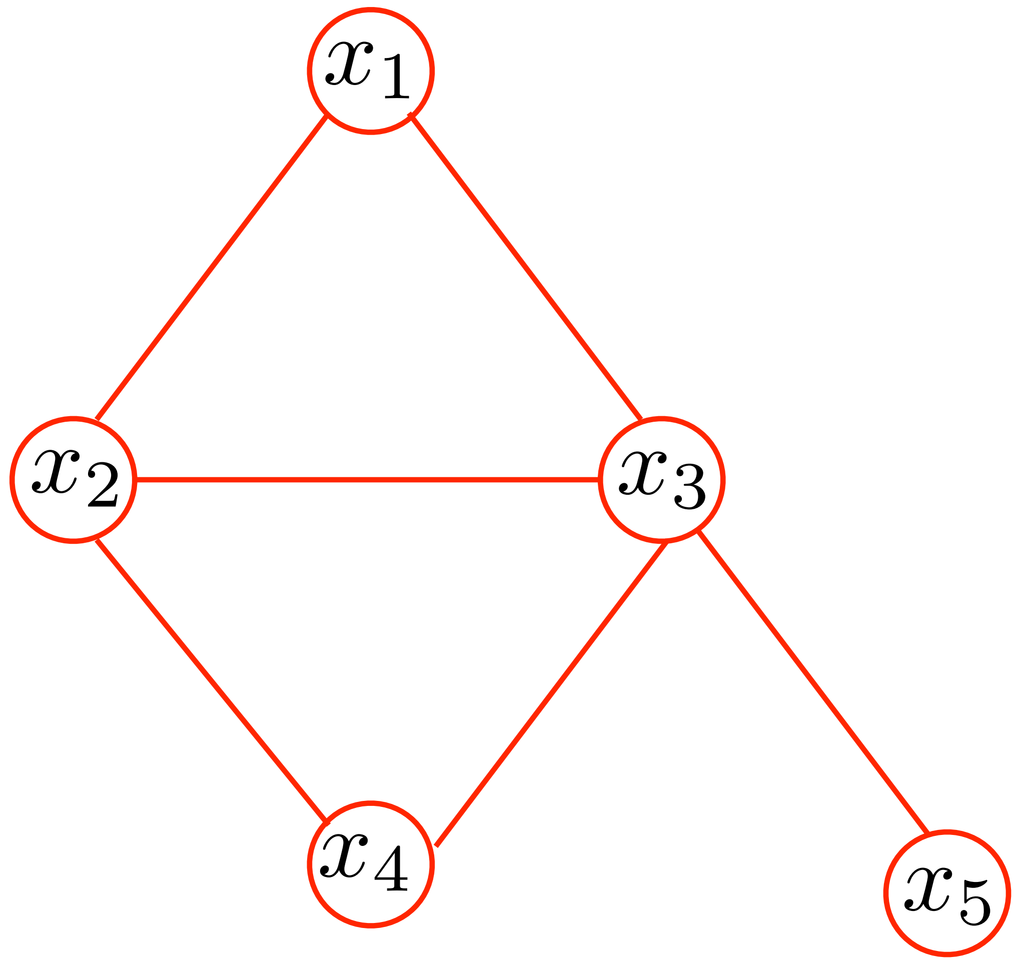
- Let all variables be binary:

$$x_i \in \{0, 1\}$$

- We can define **features** ϕ

$$\phi_{ijk}(x_i, x_j, x_k) = \begin{cases} 1 & \text{if } x_n = 1 \ \forall n \in \{i, j, k\} \\ 0 & \text{otherwise} \end{cases}$$

A Simple Example



Using numbers, e.g.:

- Let all variables be binary:

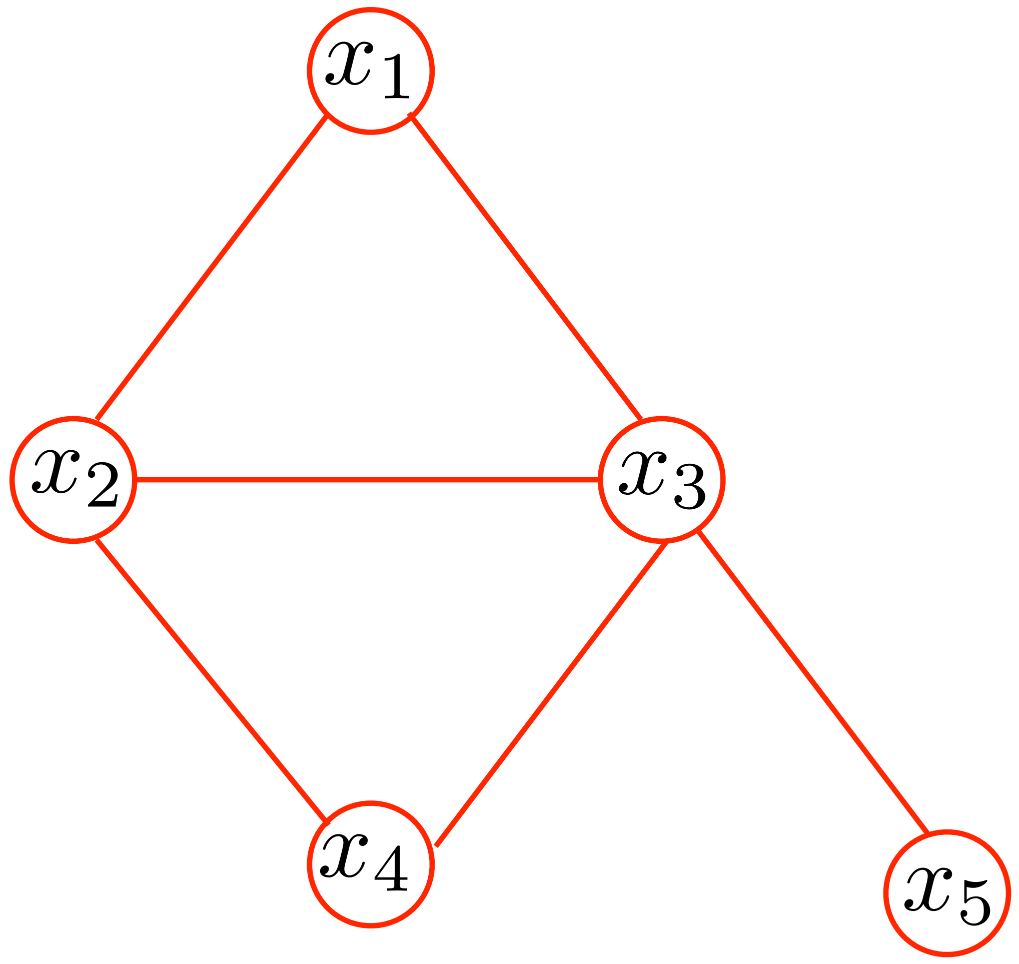
$$x_i \in \{0,1\}$$

- We can define **features** ϕ
- and determine **weights** θ

$$\phi_{ijk}(x_i, x_j, x_k) = \begin{cases} 1 & \text{if } x_n = 1 \; \forall n \in \{i,j,k\} \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{\theta} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)^T$$

A Simple Example

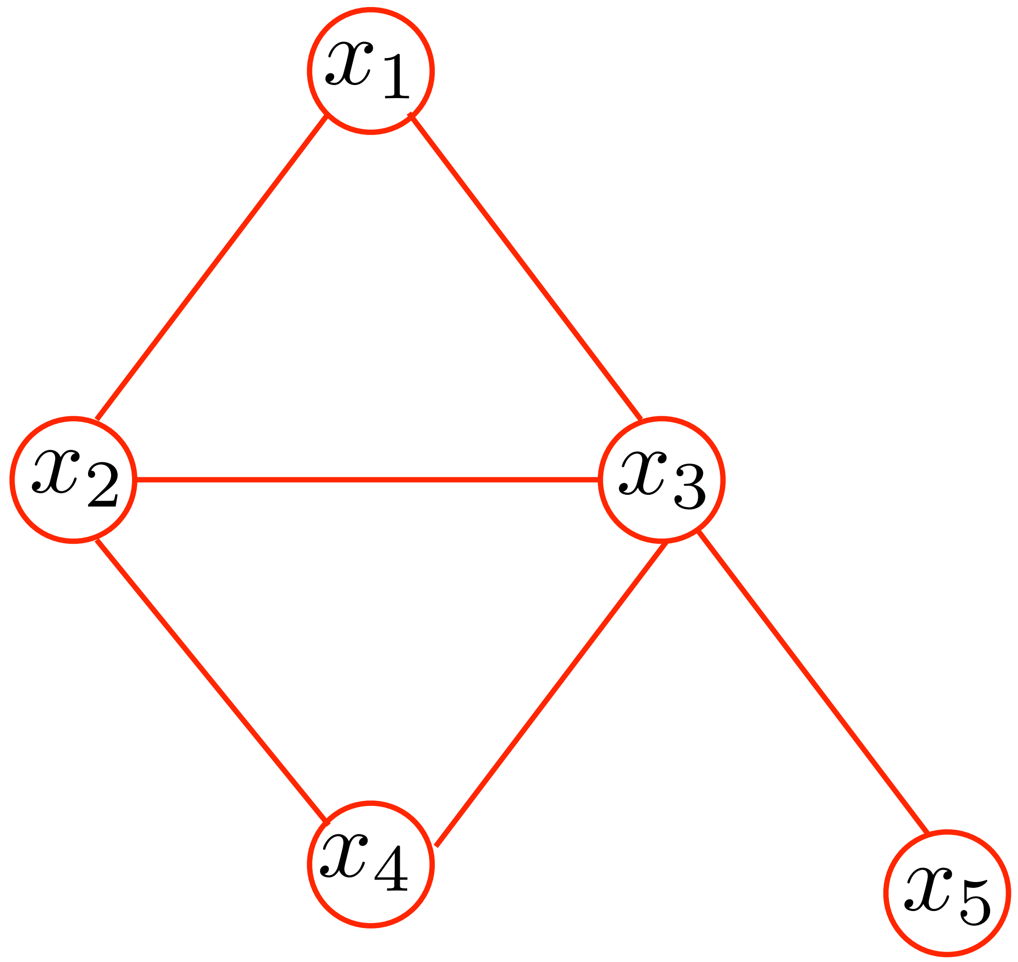


Using numbers, e.g.:

- Let all variables be binary:
$$x_i \in \{0, 1\}$$
- We can define **features** ϕ
- and determine **weights** θ
- Then, we can compute the (log of the) joint probability for each realisation of the x_i

$$\log p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c)^T \boldsymbol{\theta}_c - \log Z(\boldsymbol{\theta})$$

A Simple Example



Using numbers, e.g.:

- The same graph can also be interpreted as a **binary** MRF
- This a more specific representation, but it is less complex (and therefore more efficient)
- In Computer Vision, we almost always use **binary** MRFs; they are a specific case of general MRFs:

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{23}(x_2,x_3)\psi_{24}(x_2,x_4)\psi_{34}(x_3,x_4)\psi_{35}(x_3,x_5)$$

Using Graphical Models

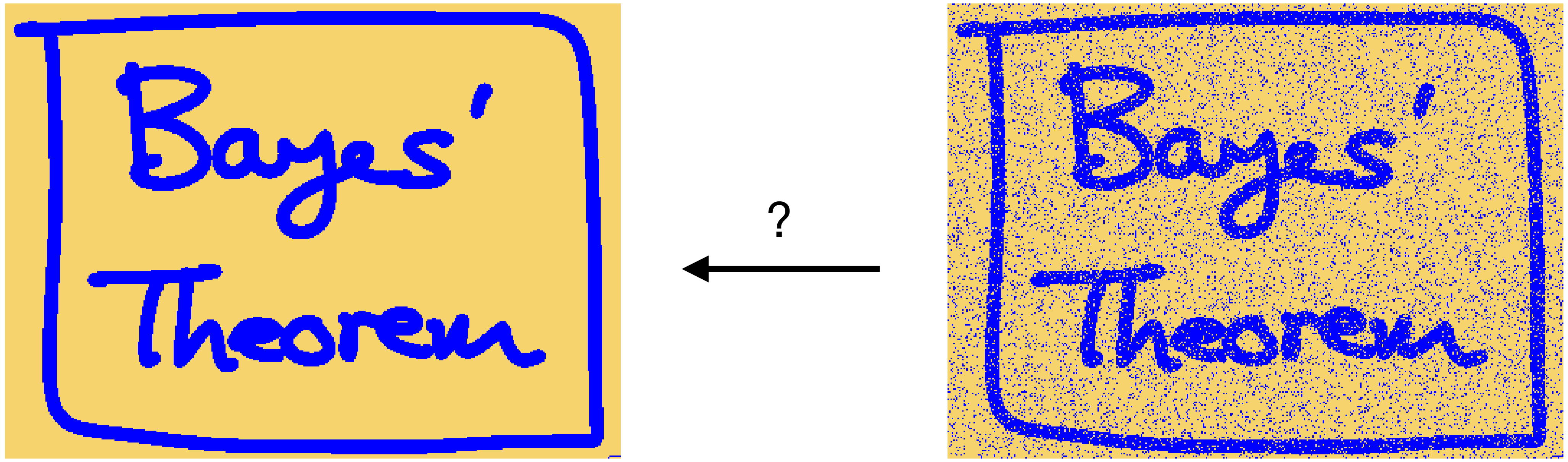
We can use a graphical model to do **inference**:

- We want to find $\arg\max_{\mathbf{x}} p(\mathbf{x})$
- 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.

Example Application: Denoising

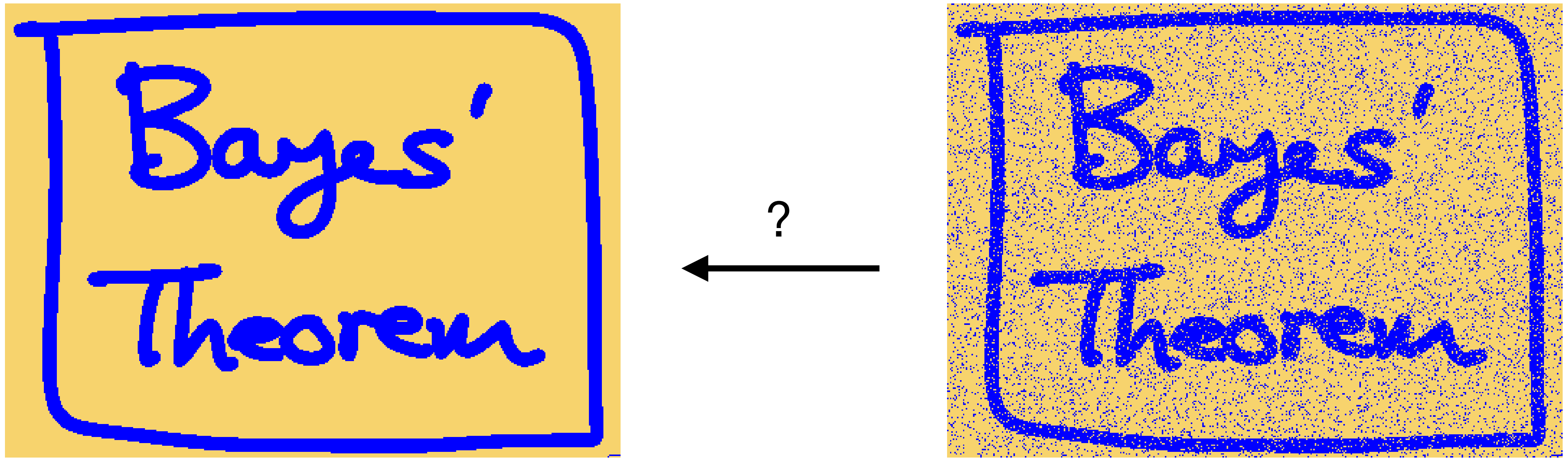


Noise-free image **Noisy image (observation)**

Aim: Recover the noise-free image from the noisy one

We model the original image with variables $x_i \in \{-1, 1\}$
and the noisy image with pixel values $y_i \in \{-1, 1\}$

Example Application: Denoising



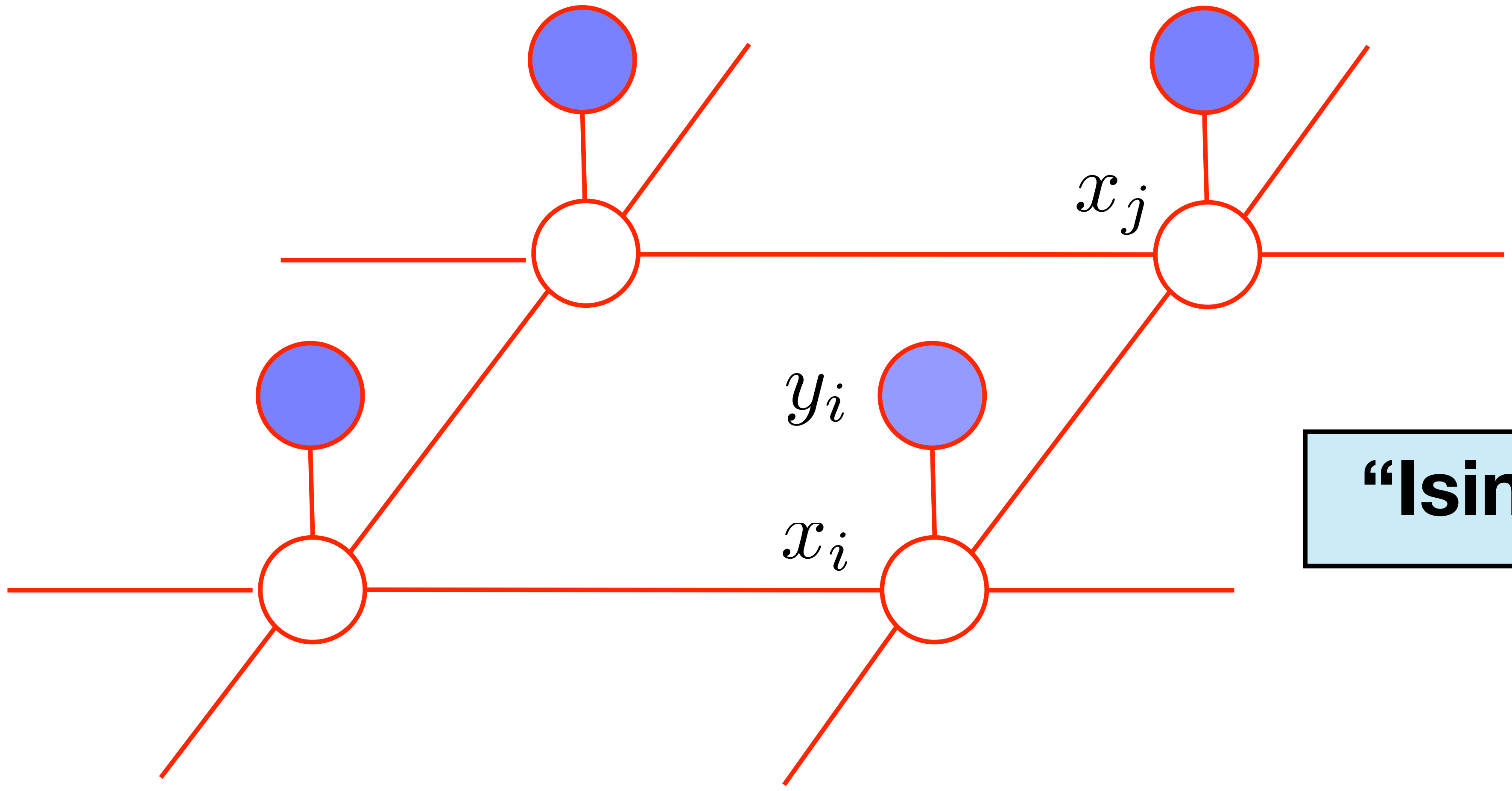
Noise-free image **Noisy image (observation)**

Aim: Recover the noise-free image from the noisy one

We model the original image with variables $x_i \in \{-1, 1\}$ and the noisy image with pixel values $y_i \in \{-1, 1\}$

We consider the true pixel vales as **hidden** or **latent**

Example Application: Denoising



“Ising model"

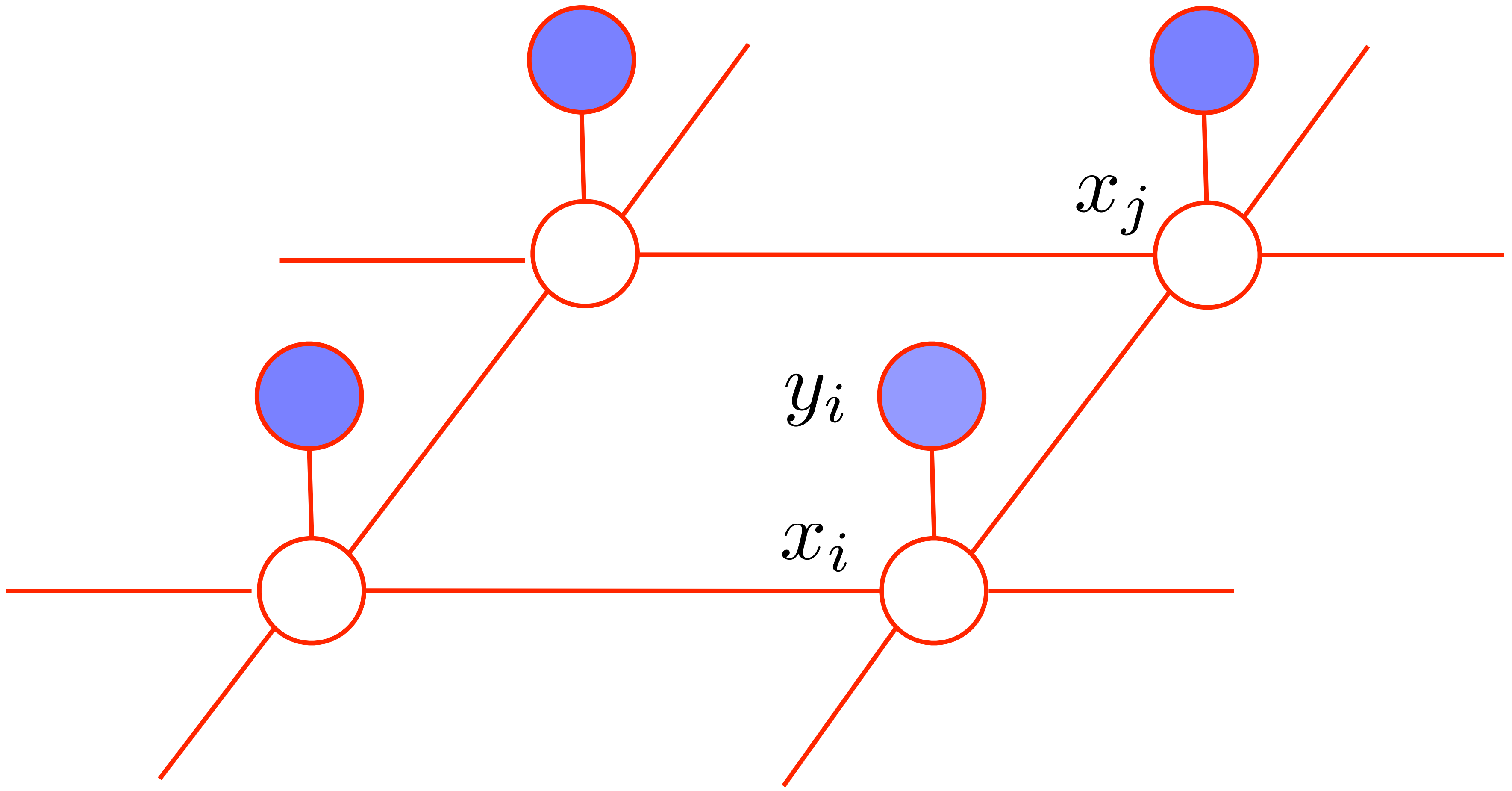
We define two simple edge features:

$$\phi(x_i, y_i) = x_i y_i \qquad \phi(x_i, x_j) = x_i x_j$$

These are multiplied by parameters β and η:

$$\log \psi(x_i, y_i) = \eta x_i y_i \qquad \log \psi(x_i, y_i) = \eta x_i y_i$$

Example Application: Denoising



With this, we can compute the joint:

$$p(\mathbf{x}, \mathbf{y} \mid \eta, \beta) = \frac{1}{Z} \prod_i \exp(\eta x_i y_i) \prod_{i,j} \exp(\beta x_i x_j)$$

and its log:

$$\log p(\mathbf{x}, \mathbf{y} \mid \eta, \beta) = \eta \sum_i x_i y_i + \beta \sum_{i,j} x_i x_j - \log(Z)$$

Example Application: Denoising

$$\log p(\mathbf{x}, \mathbf{y} \mid \eta, \beta) = \eta \sum_i x_i y_i + \beta \sum_{i,j} x_i x_j - \log(Z)$$

Our aim now is to find the hidden states x_i such that this log of the joint is maximal (or at least very high).

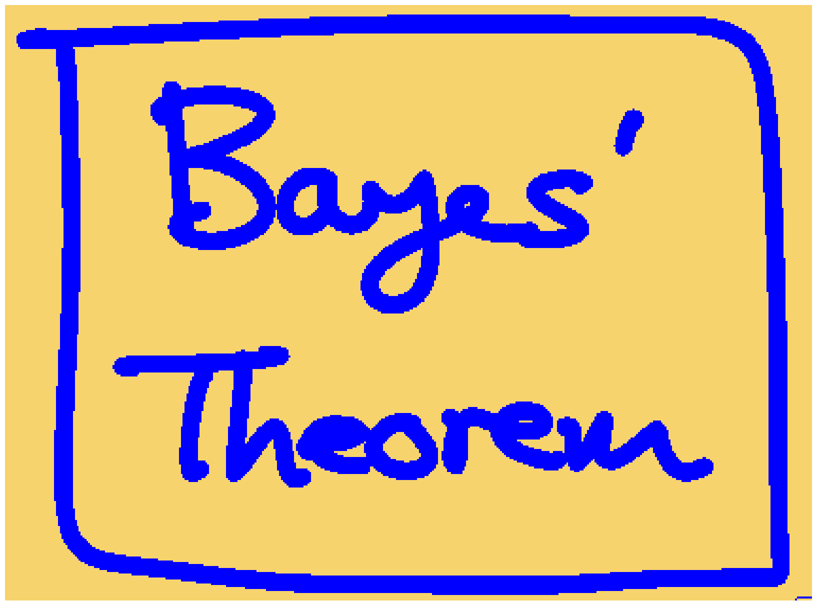
Simple approach is Iterated Conditional Modes (ICM):

1. Initialize all x_i by corresponding y_i
2. For all nodes x_i :
    1. set x_i to +1 and to -1 and evaluate $\log p(\mathbf{x}, \mathbf{y} \mid \eta, \beta)$
    2. keep the value that gives higher log joint

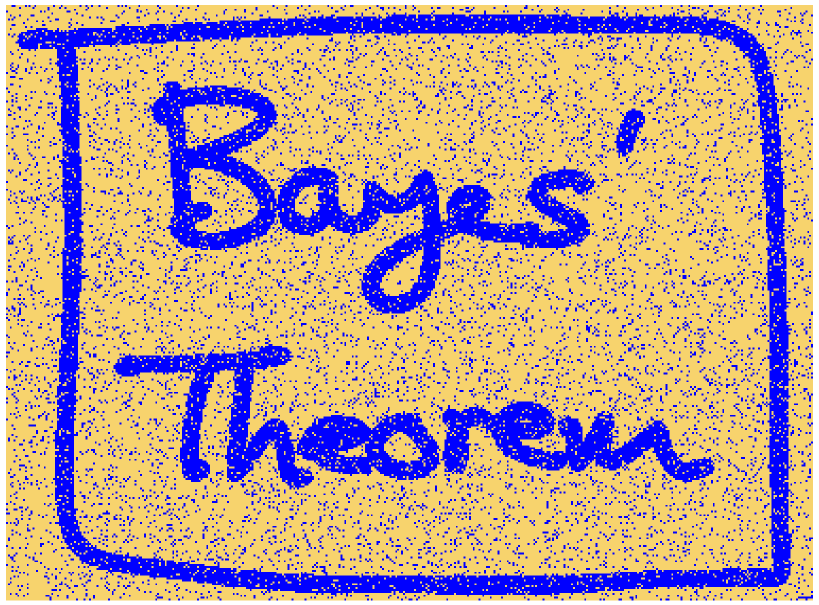
This will keep or increase the joint in every step

The nodes can be visited in order or randomly

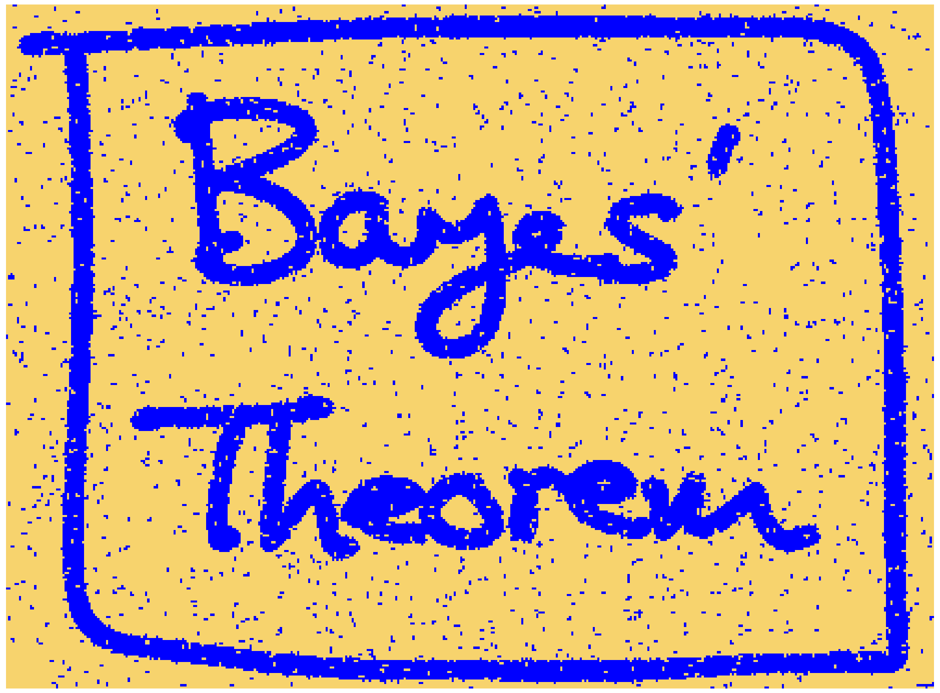
Result of ICM



Noise-free image



Noisy image (observation)

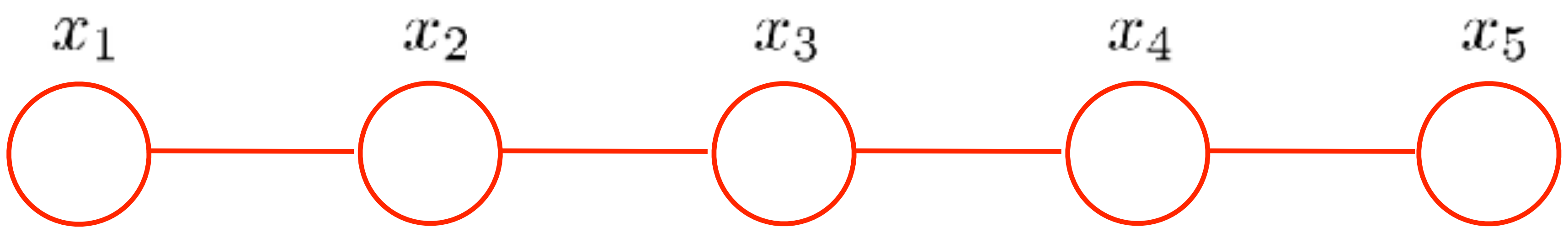


Result of ICM

General Inference in MRFs

- In general, we do not have such an easy model
- Therefore, we need more general inference methods for MRF
- The major aim is to exploit sparsity in the graphical model to make inference efficient

Inference on a Chain



The joint probability is given by

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

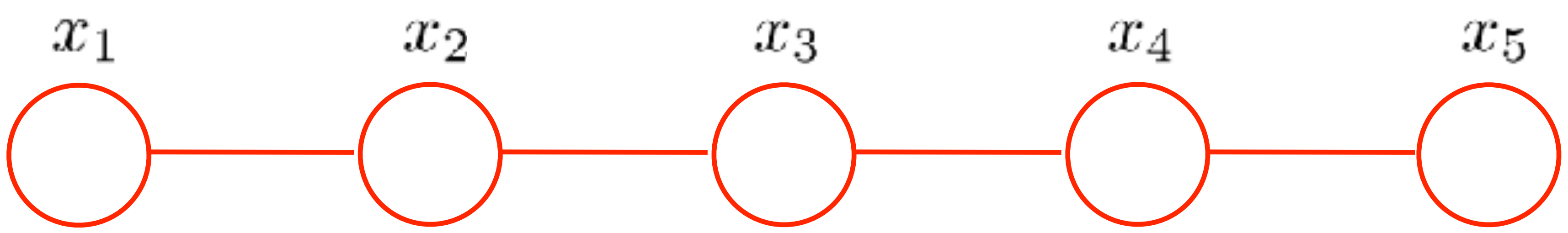
The marginal at x_3 is $$p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(\mathbf{x})$$

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



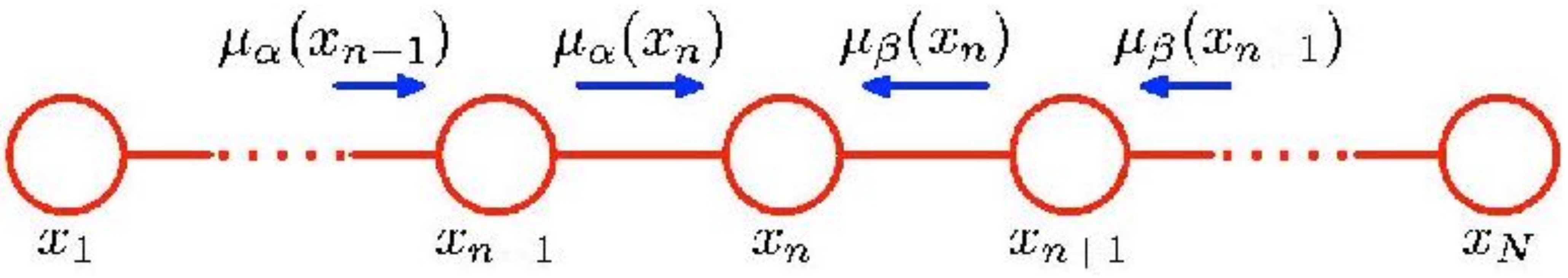
$$p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(\mathbf{x})$$

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

$$\begin{aligned}
p(x_3) &= \frac{1}{Z}\sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)\\
&= \frac{1}{Z}\sum_{x_2}\sum_{x_1}\sum_{x_4}\sum_{x_5}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)\psi_{4,5}(x_4,x_5)\\
&= \frac{1}{Z}\underbrace{\sum_{x_2}\psi_{2,3}(x_2,x_3)\sum_{x_1}\psi_{1,2}(x_1,x_2)}_{\mu_\alpha(x_3)}\underbrace{\sum_{x_4}\psi_{3,4}(x_3,x_4)\sum_{x_5}\psi_{4,5}(x_4,x_5)}_{\mu_\beta(x_3)}
\end{aligned}$$

$$\mu_\alpha(x_3) \leftarrow \boxed{\text{Vectors of size K}} \rightarrow \mu_\beta(x_3)$$

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

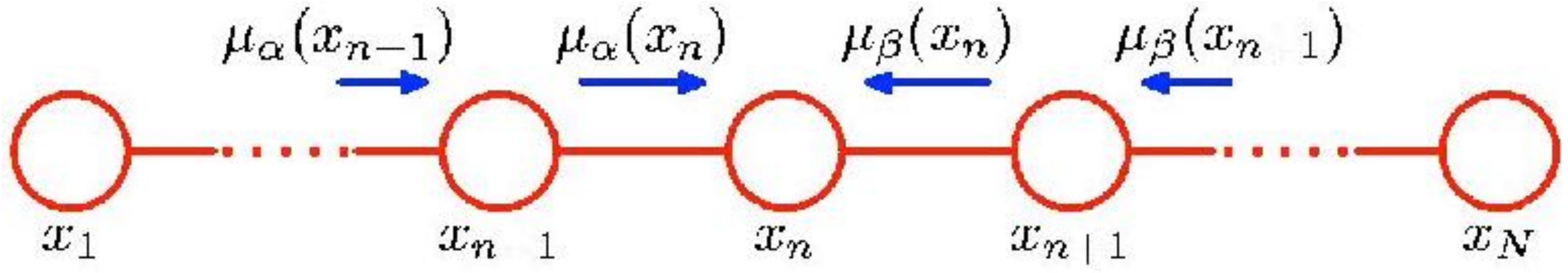
The **messages** μ_α and μ_β can be computed recursively:

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

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

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

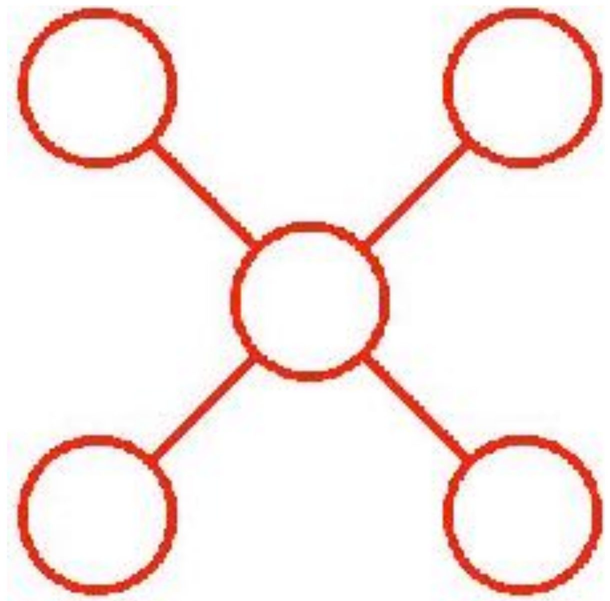
$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$

for all variables required.

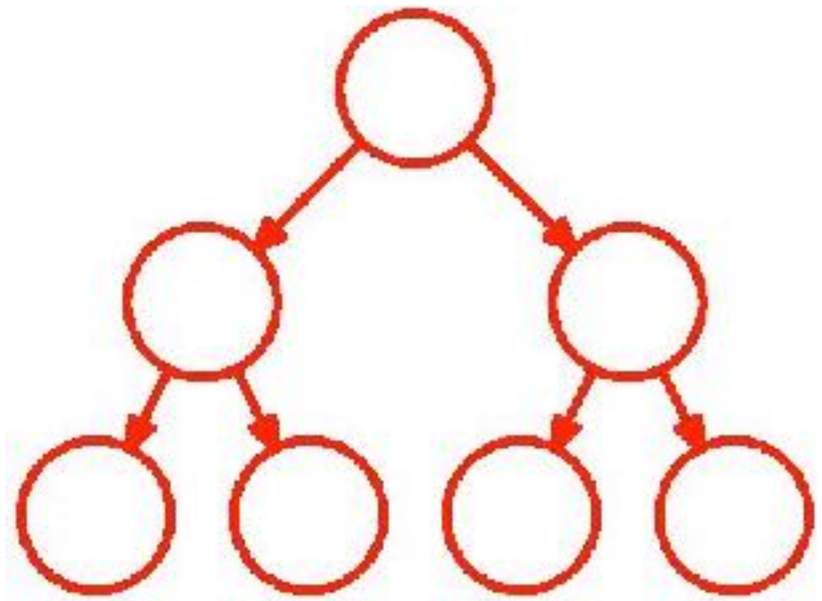
More General Graphs

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

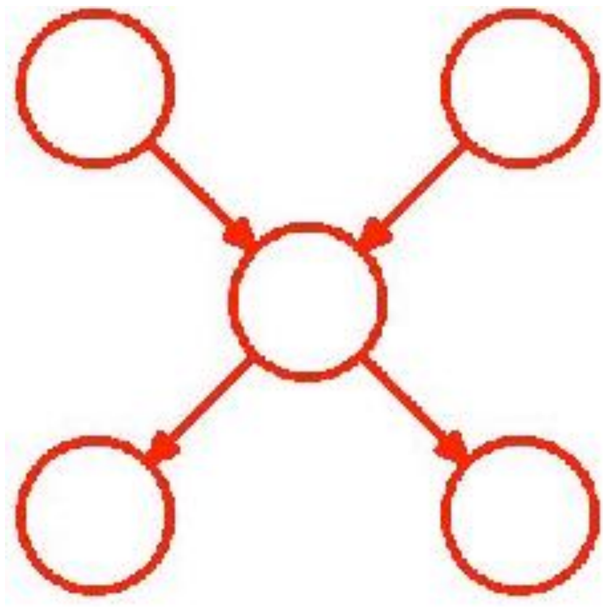
Undirected Tree



Directed Tree



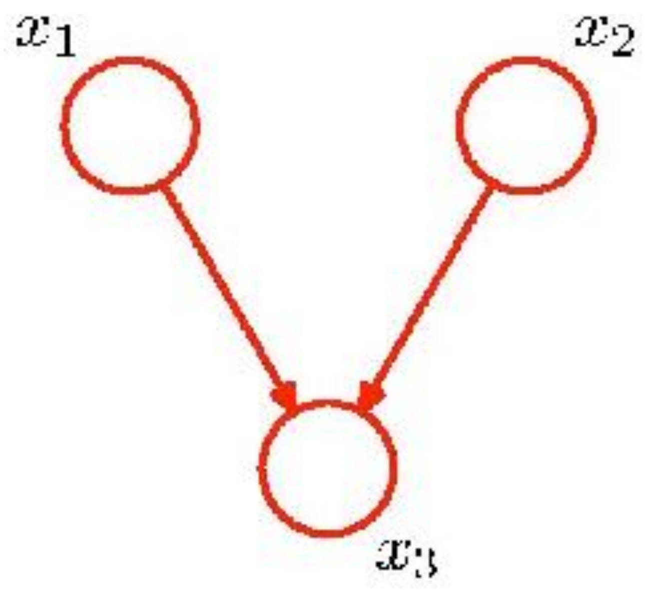
Polytree

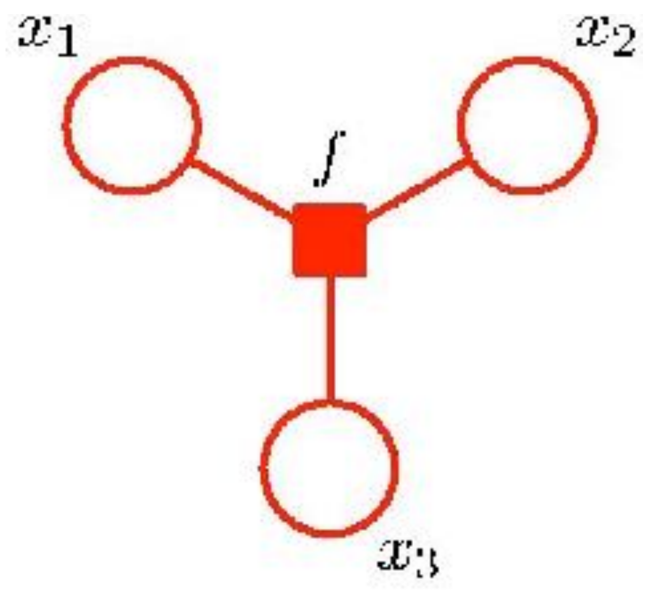


It is then known as the **sum-product algorithm.**
A special case of this is **belief propagation**.

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**.





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

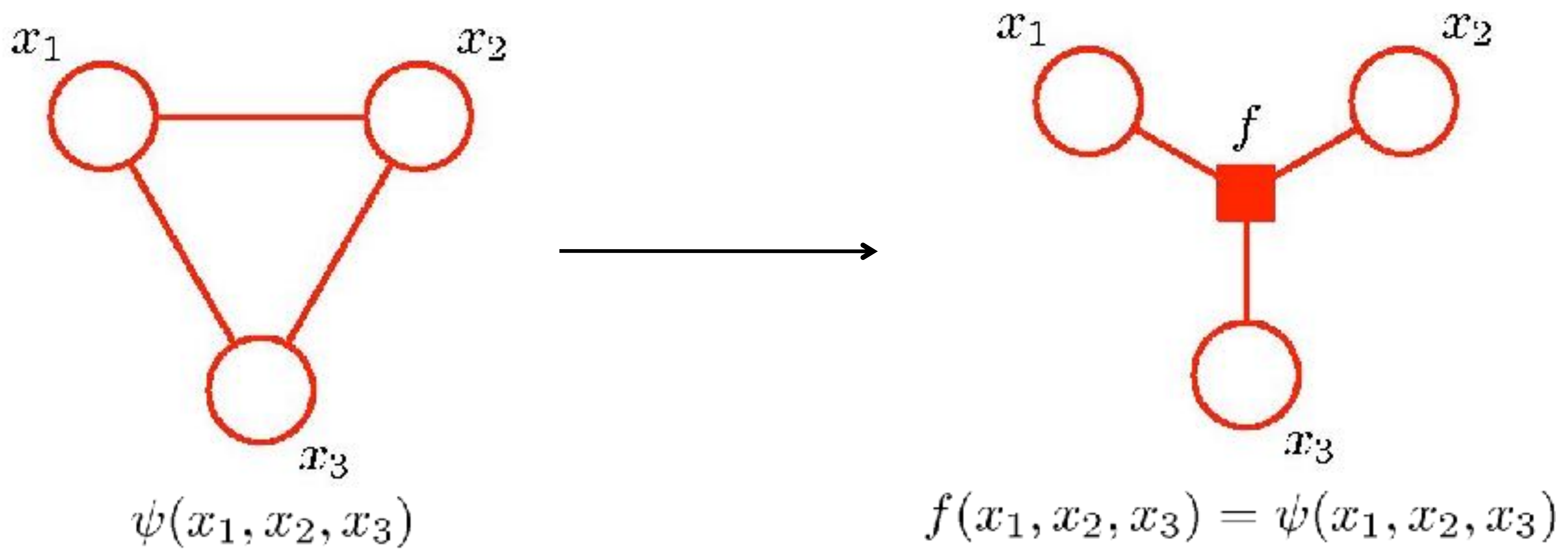
$$f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3 \mid x_1, x_2)$$

Directed graph

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**.



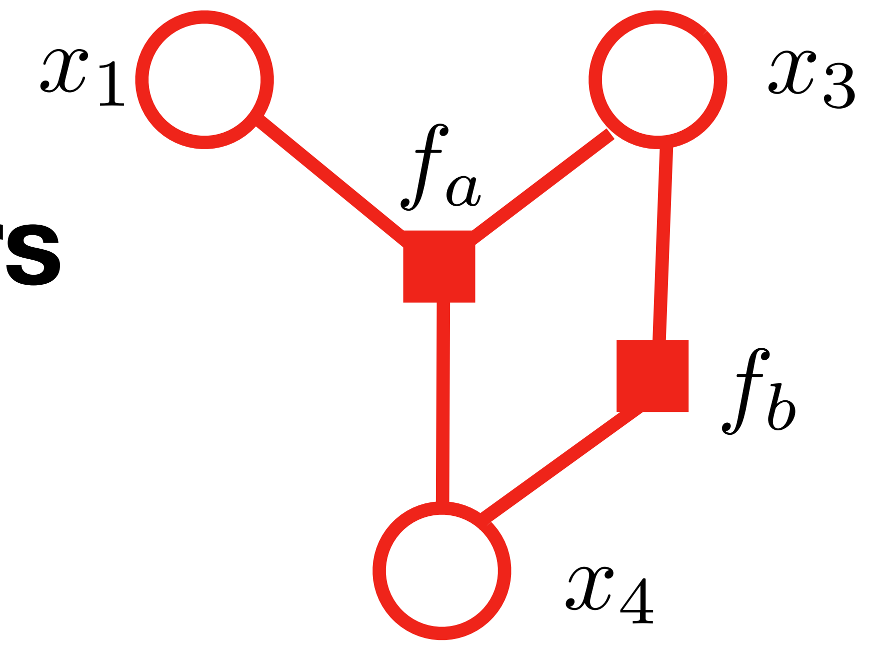
Undirected graph

Factor graph

Factor Graphs

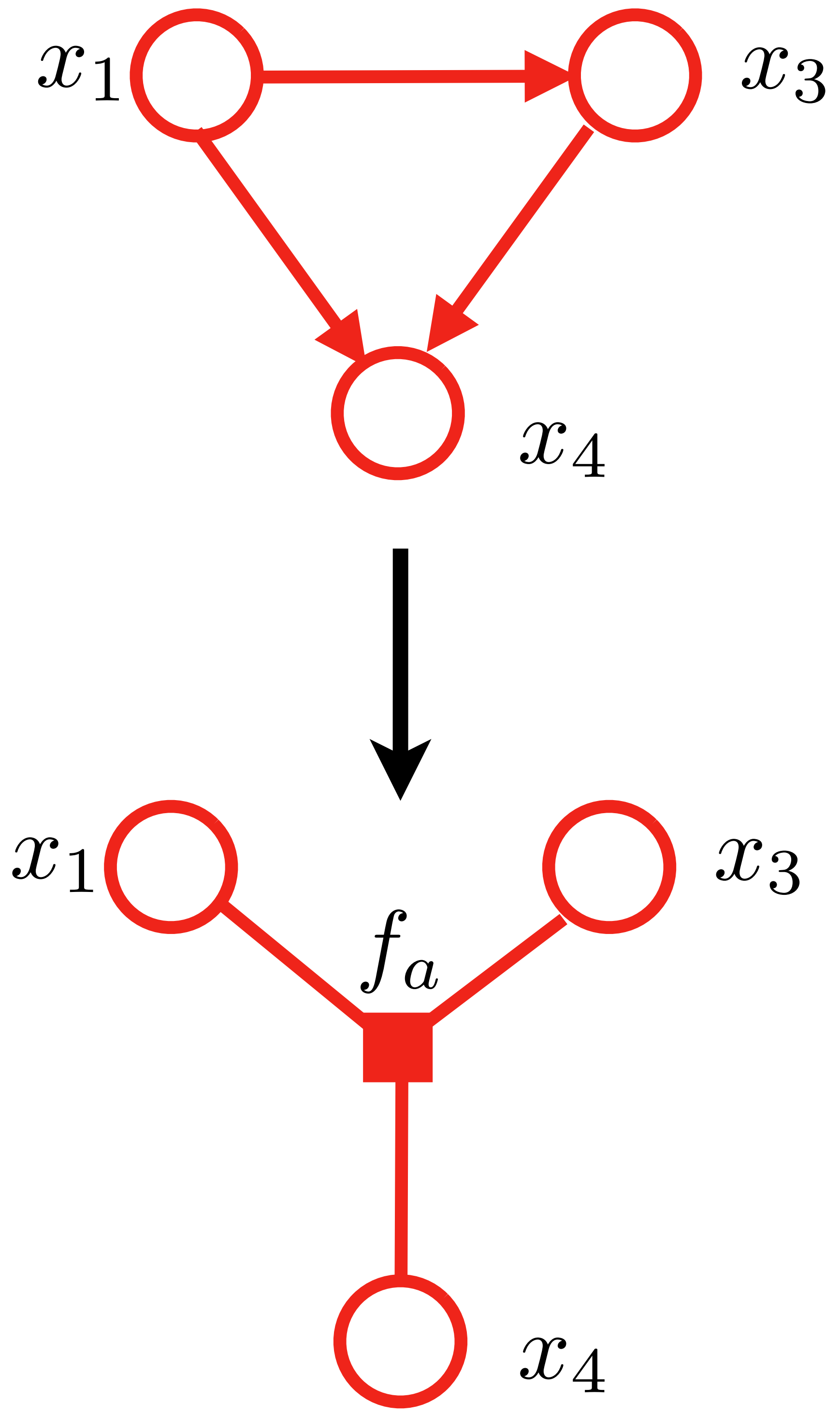
Factor graphs

- 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: Local cycles 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

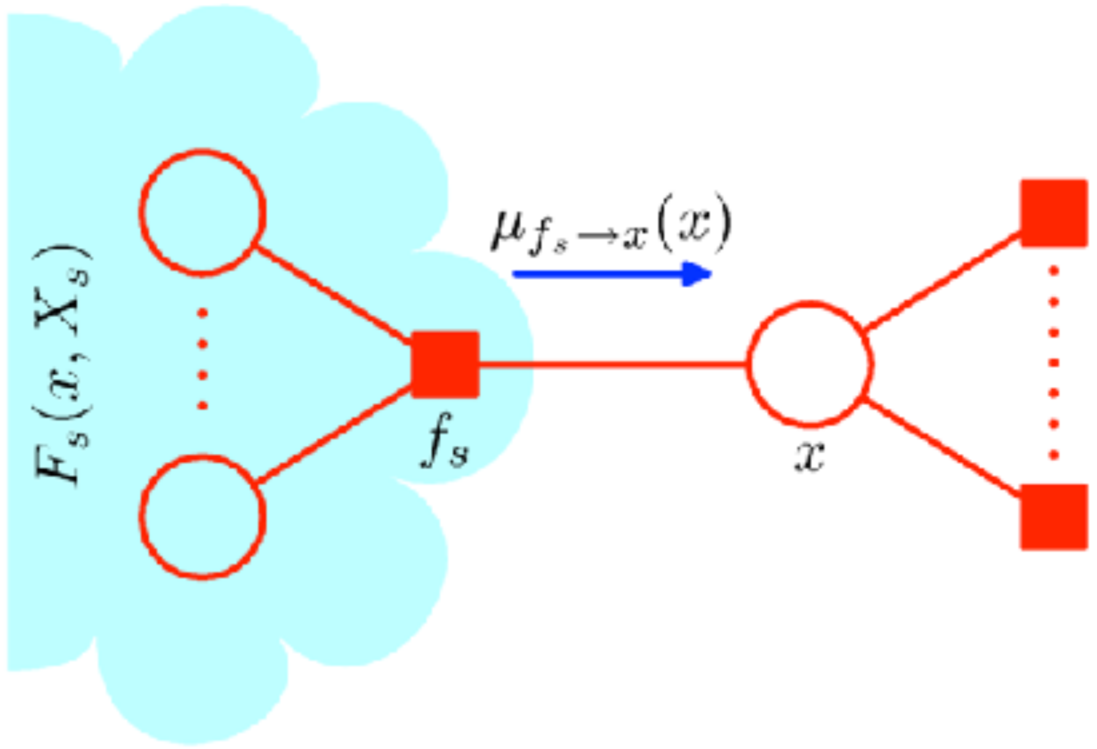
The factor graph represents the joint distribution as a product of factor nodes:

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

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

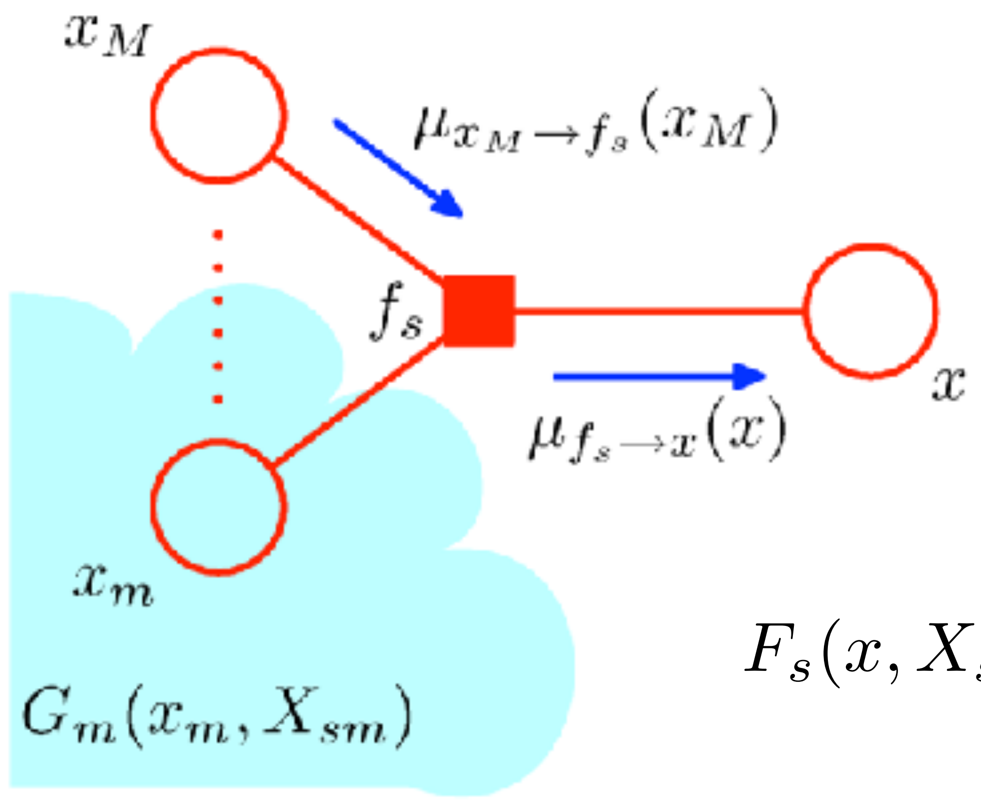
Thus, we have $p(x) = \sum_{\mathbf{x} \setminus x} \prod_{s \in \mathrm{ne}(x)} F_s(x, X_s)$

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:

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

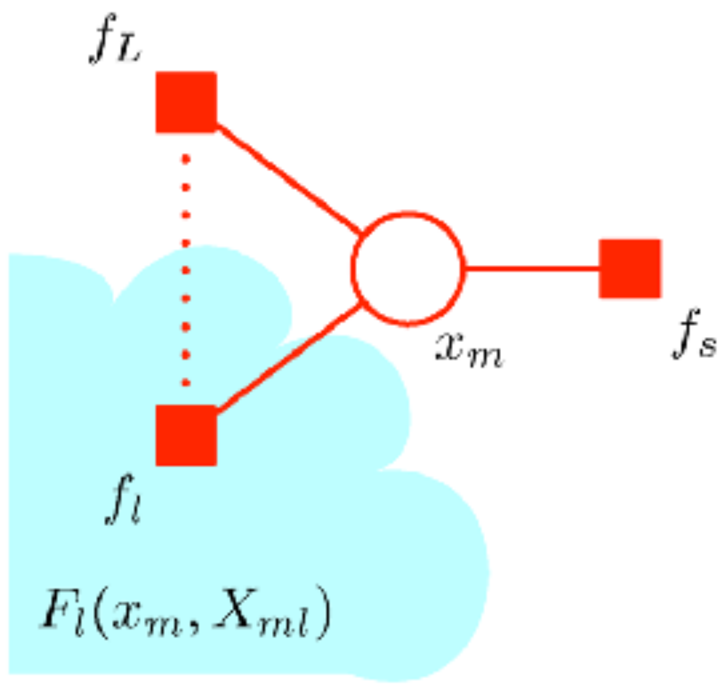
The messages can then be computed as

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

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

“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:

   $\mu_{f \to x}(x) = 1$ (var) or $\mu_{x \to f}(x) = f(x)$ (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:

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

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

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:

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

Sum-Product Inference in General Graphical Models

1. Convert graph (directed or undirected) into a **factor graph** (there are no cycles)
2. If the goal is to **marginalize** at node x, then consider x as a root node
3. Initialize the recursion at the leaf nodes as:

   $\mu_{f \to x}(x) = 1$ (var) or $\mu_{x \to f}(x) = f(x)$ (fac)
4. Propagate messages from the leaves to x
5. Propagate messages from x to the leaves
6. Obtain marginals at every node by multiplying all incoming messages

Other Inference Algorithms

- Max-Sum algorithm: used to **maximize** the joint probability of all variables (no marginalization)
- Junction Tree algorithm: exact inference for general graphs (even with loops)
- Loopy belief propagation: approximate inference on general graphs (more efficient)

Special kind of undirected GM:

- Conditional Random fields (e.g.: classification)

Conditional Random Fields

- Another kind of undirected graphical model 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.
- CRFs are **discriminative** and MRFs are **generative**

Conditional Random Fields

Derivation of the formula for CRFs:

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

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

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

where $(\mathbf{x}, \mathbf{y})$ is the training data and $p(\mathbf{w})$ is a Gaussian prior. In the inference phase we maximize

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

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**:

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

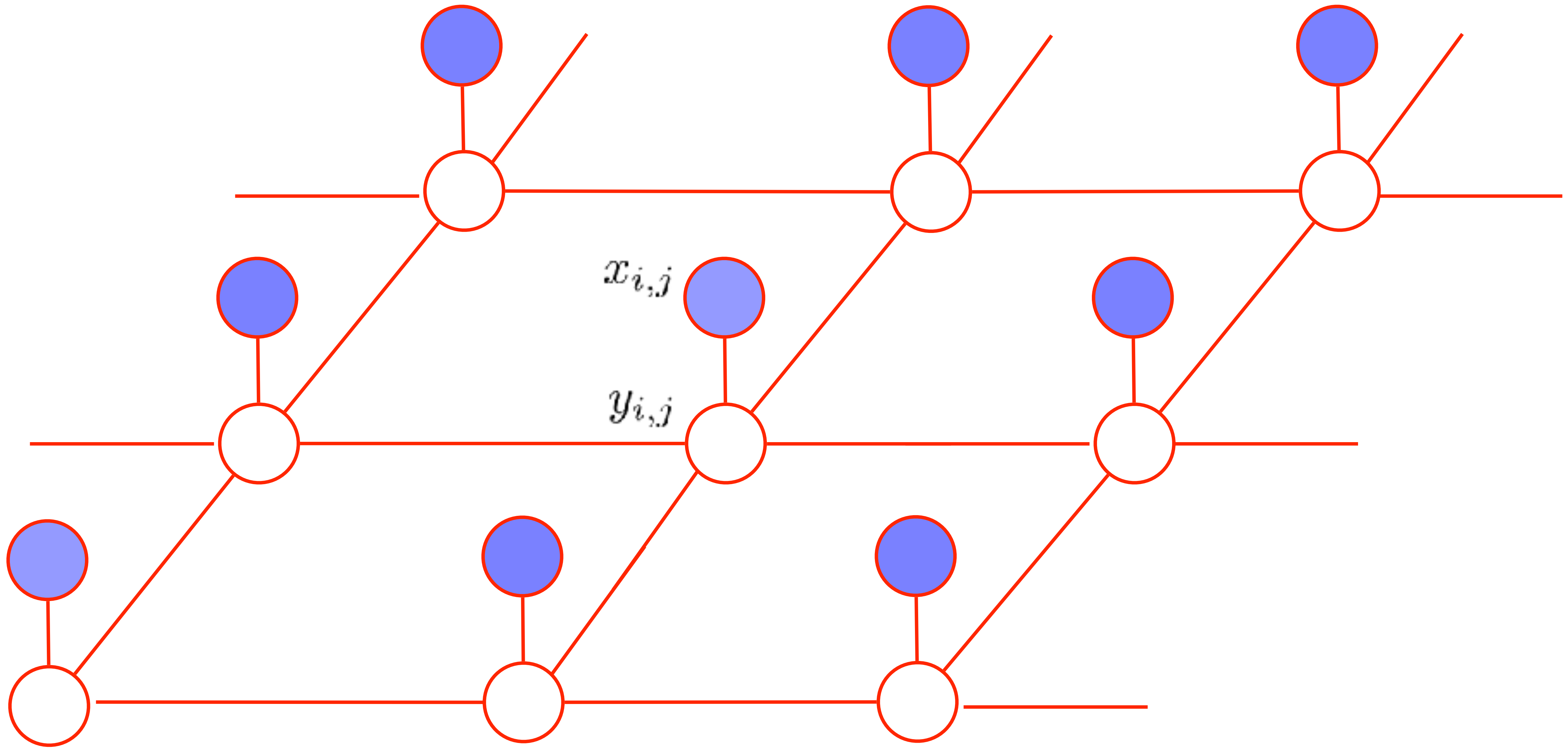
where

Markov blanket

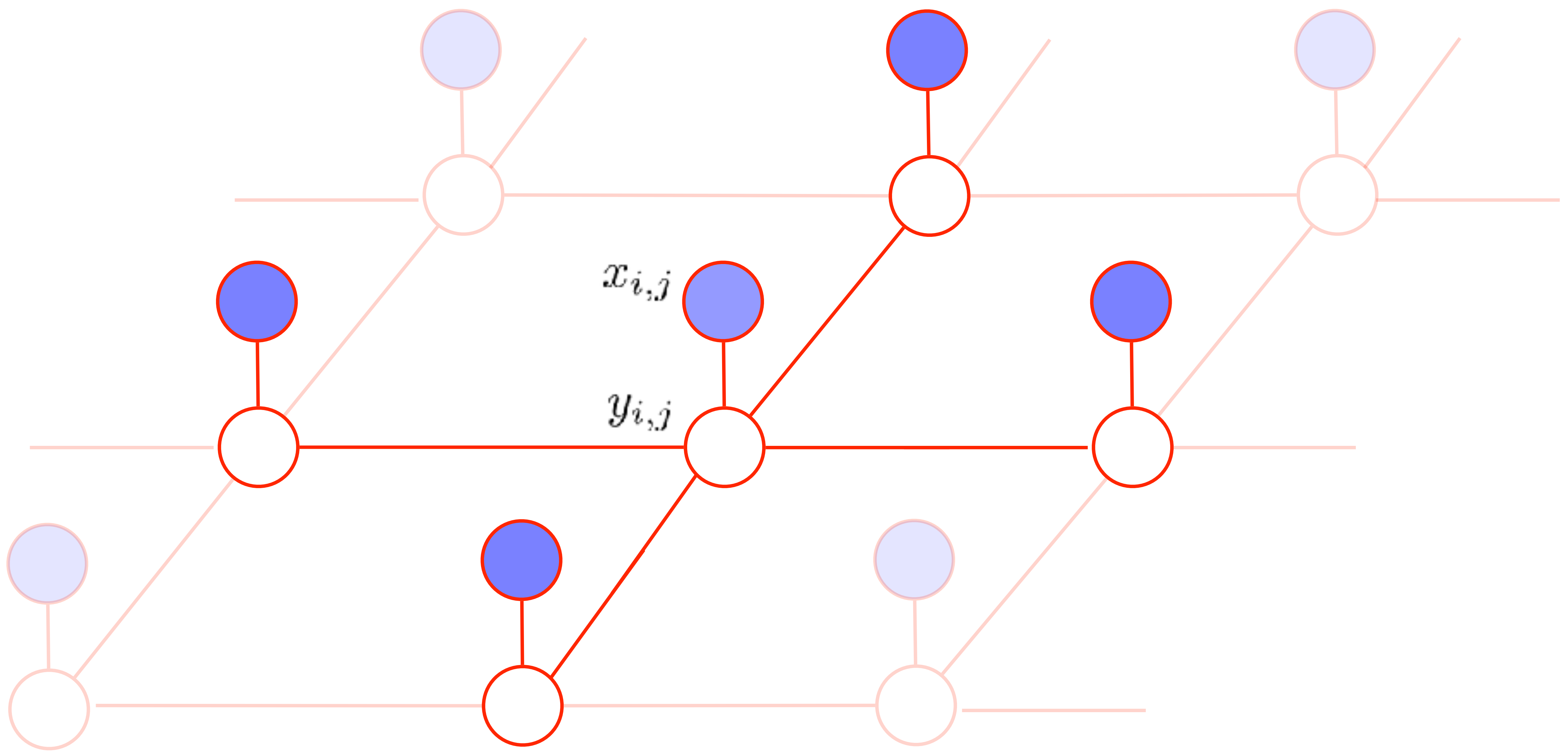
C_i: All cliques containing y_i

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

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:

$$L(\mathbf{w}) = \underbrace{-lpl(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w})}_{\text{Log-pseudo-likelihood}} + \underbrace{\frac{1}{2\sigma^2}\mathbf{w}^T\mathbf{w}}_{\text{Gaussian prior}}$$

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

Inference:

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

Summary

- Undirected models (aka Markov random fields) provide an intuitive representation of conditional independence
- An MRF is defined as a **factorization** over clique potentials and normalized globally
- Directed and undirected models have different representative power (no simple “containment”)
- Inference on undirected Markov chains is efficient using message passing
- Factor graphs are more general; exact inference can be done efficiently using sum-product