

# Graphical Models

Andrea Passerini  
passerini@disi.unitn.it

Complex Systems

## Why

- All probabilistic inference and learning amount at repeated applications of the sum and product rules
- *Probabilistic graphical models* are graphical representations of the *qualitative* aspects of probability distributions allowing to:
  - visualize the structure of a probabilistic model in a simple and intuitive way
  - discover properties of the model, such as conditional independencies, by inspecting the graph
  - express complex computations for inference and learning in terms of graphical manipulations
  - represent multiple probability distributions with the same graph, abstracting from their quantitative aspects (e.g. discrete vs continuous distributions)

# Bayesian Networks (BN)

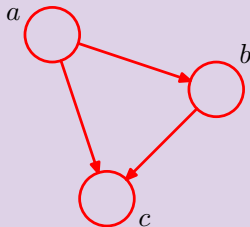
## Description

- Directed graphical models representing joint distributions

$$\begin{aligned}p(a, b, c) &= p(c|a, b)p(a, b) \\ &= p(c|a, b)p(b|a)p(a)\end{aligned}$$

- Fully connected graphs contain no independency assumption
- Joint probabilities can be decomposed in many different ways by the product rule
- Assuming an ordering of the variables, a probability distributions over  $K$  variables can be decomposed into:

$$p(x_1, \dots, x_K) = p(x_K|x_1, \dots, x_{K-1}) \dots p(x_2|x_1)p(x_1)$$



# Bayesian Networks

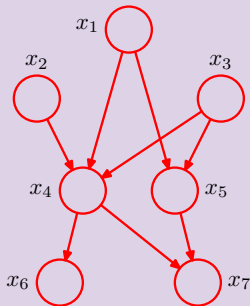
## Modeling independencies

- A graph not fully connected contains independency assumptions.
- The joint probability of a Bayesian network with  $K$  nodes is computed as the product of the probabilities of the nodes given their parents:

$$p(\mathbf{x}) = \prod_{i=1}^K p(x_i | \text{pa}_i)$$

- The joint probability for the BN in the figure is written as

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



# Conditional independence

## Introduction

- Two variables  $a, b$  are conditionally independent (written  $a \perp\!\!\!\perp b \mid \emptyset$ ) if:

$$p(a, b) = p(a)p(b)$$

- Two variables  $a, b$  are conditionally independent given  $c$  (written  $a \perp\!\!\!\perp b \mid c$ ) if:

$$p(a, b|c) = p(a|c)p(b|c)$$

- Independence assumptions can be verified by repeated applications of sum and product rules
- Graphical models allow to directly verify them through the *d-separation* criterion

## Tail-to-tail

- Joint distribution:

$$p(a, b, c) = p(a|c)p(b|c)p(c)$$

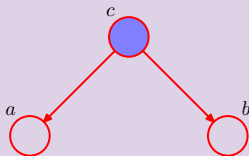
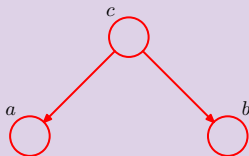
- $a$  and  $b$  are **not conditionally independent** (written  $a \not\perp\!\!\!\perp b \mid \emptyset$ ):

$$p(a, b) = \sum_c p(a|c)p(b|c)p(c) \neq p(a)p(b)$$

- $a$  and  $b$  are **conditionally independent given  $c$** :

$$p(a, b|c) = \frac{p(a, b, c)}{p(c)} = p(a|c)p(b|c)$$

- $c$  is *tail-to-tail* wrt to the path  $a \rightarrow b$  as it is connected to the tails of the two arrows



## Head-to-tail

- Joint distribution:



$$p(a, b, c) = p(b|c)p(c|a)p(a) = p(b|c)p(a|c)p(c)$$

- $a$  and  $b$  are **not conditionally independent**:

$$p(a, b) = p(a) \sum_c p(b|c)p(c|a) \neq p(a)p(b)$$

- $a$  and  $b$  are **conditionally independent given  $c$** :



$$p(a, b|c) = \frac{p(b|c)p(a|c)p(c)}{p(c)} = p(b|c)p(a|c)$$

- $c$  is *head-to-tail* wrt to the path  $a \rightarrow b$  as it is connected to the head of an arrow and to the tail of the other one

## Head-to-head

- Joint distribution:

$$p(a, b, c) = p(c|a, b)p(a)p(b)$$

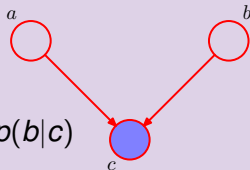
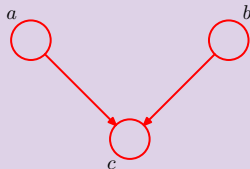
- $a$  and  $b$  are **conditionally independent**:

$$p(a, b) = \sum_c p(c|a, b)p(a)p(b) = p(a)p(b)$$

- $a$  and  $b$  are **not conditionally independent given  $c$** :

$$p(a, b|c) = \frac{p(c|a, b)p(a)p(b)}{p(c)} \neq p(a|c)p(b|c)$$

- $c$  is *tail-to-tail* wrt to the path  $a \rightarrow b$  as it is connected to the heads of the two arrows



## General Head-to-head

- Let a *descendant* of a node  $x$  be any node which can be reached from  $x$  with a path following the direction of the arrows
- A head-to-head node  $c$  unblocks the dependency path between its parents if either itself or *any of its descendants* receives evidence

# Example of head-to-head connection

## Setting

- A fuel system in a car:

**battery**  $B$ , either charged ( $B = 1$ ) or flat ( $B = 0$ )

**fuel tank**  $F$ , either full ( $F = 1$ ) or empty ( $F = 0$ )

**electric fuel gauge**  $G$ , either full ( $G = 1$ ) or empty ( $G = 0$ )

## Conditional probability tables (CPT)

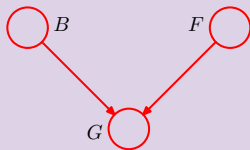
- Battery and tank have independent prior probabilities:

$$P(B = 1) = 0.9 \quad P(F = 1) = 0.9$$

- The fuel gauge is conditioned on both (unreliable!):

$$P(G = 1|B = 1, F = 1) = 0.8 \quad P(G = 1|B = 1, F = 0) = 0.2$$

$$P(G = 1|B = 0, F = 1) = 0.2 \quad P(G = 1|B = 0, F = 0) = 0.1$$



# Example of head-to-head connection

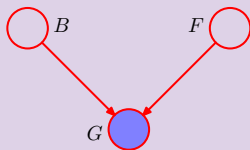
## Probability of empty tank

- Prior:

$$P(F = 0) = 1 - P(F = 1) = 0.1$$

- Posterior after observing empty fuel gauge:

$$P(F = 0 | G = 0) = \frac{P(G = 0 | F = 0)P(F = 0)}{P(G = 0)} \simeq 0.257$$



## Note

The probability that the tank is empty *increases* from observing that the fuel gauge reads empty (not as much as expected because of strong prior and unreliable gauge)

# Example of head-to-head connection

## Derivation

$$\begin{aligned}P(G = 0|F = 0) &= \sum_{B \in \{0,1\}} P(G = 0, B|F = 0) \\&= \sum_{B \in \{0,1\}} P(G = 0|B, F = 0)P(B|F = 0) \\&= \sum_{B \in \{0,1\}} P(G = 0|B, F = 0)P(B) = 0.81\end{aligned}$$

$$\begin{aligned}P(G = 0) &= \sum_{B \in \{0,1\}} \sum_{F \in \{0,1\}} P(G = 0, B, F) \\&= \sum_{B \in \{0,1\}} \sum_{F \in \{0,1\}} P(G = 0|B, F)P(B)P(F)\end{aligned}$$

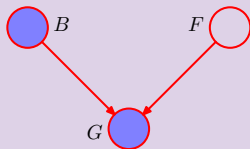
# Example of head-to-head connection

## Probability of empty tank

- Posterior after observing that the battery is also flat:

$$P(F = 0 | G = 0, B = 0) =$$

$$\frac{P(G = 0 | F = 0, B = 0)P(F = 0 | B = 0)}{P(G = 0)} \simeq 0.111$$



## Note

- The probability that the tank is empty *decreases* after observing that the battery is also flat
- The battery condition *explains away* the observation that the fuel gauge reads empty
- The probability is still greater than the prior one, because the fuel gauge observation still gives some evidence in favour of an empty tank

# General $d$ -separation criterion

## d-separation definition

- Given a generic Bayesian network
- Given  $A, B, C$  arbitrary nonintersecting sets of nodes
- The sets  $A$  and  $B$  are  $d$ -separated by  $C$  if:
  - All paths from any node in  $A$  to any node in  $B$  are *blocked*
- A path is blocked if it includes at least one node s.t. either:
  - the arrows on the path meet tail-to-tail or head-to-tail at the node and it is in  $C$ , or
  - the arrows on the path meet head-to-head at the node and neither it nor any of its descendants is in  $C$

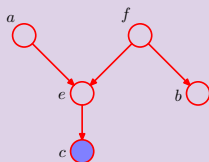
## d-separation implies conditional independency

The sets  $A$  and  $B$  are independent given  $C$  ( $A \perp\!\!\!\perp B \mid C$ ) if they are  $d$ -separated by  $C$ .

# Example of general d-separation

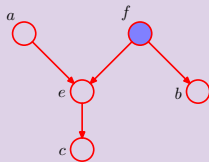
$$a \not\perp\!\!\!\perp b | c$$

- Nodes  $a$  and  $b$  are **not d-separated** by  $c$ :
  - Node  $f$  is tail-to-tail and not observed
  - Node  $e$  is head-to-head and its child  $c$  is observed



$$a \perp\!\!\!\perp b | f$$

- Nodes  $a$  and  $b$  are **d-separated** by  $f$ :
  - Node  $f$  is tail-to-tail and observed



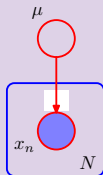
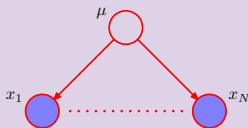
# Example of i.i.d. samples

## Maximum-likelihood

- We are given a set of instances  $\mathcal{D} = \{x_1, \dots, x_N\}$  drawn from an univariate Gaussian with unknown mean  $\mu$
- All paths between  $x_i$  and  $x_j$  are blocked if we condition on  $\mu$
- The examples are independent of each other given  $\mu$ :

$$p(\mathcal{D}|\mu) = \prod_{i=1}^N p(x_i|\mu)$$

- A set of nodes with the same variable type and connections can be compactly represented using the *plate* notation



# Markov blanket (or boundary)

## Definition

- Given a directed graph with  $D$  nodes
- The *markov blanket* of node  $x_i$  is the minimal set of nodes making it  $x_i$  independent on the rest of the graph:

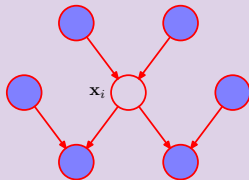
$$\begin{aligned} p(x_i | x_{j \neq i}) &= \frac{p(x_1, \dots, x_D)}{p(x_{j \neq i})} = \frac{p(x_1, \dots, x_D)}{\int p(x_1, \dots, x_D) dx_j} \\ &= \frac{\prod_{k=1}^D p(x_k | pa_k)}{\int \prod_{k=1}^D p(x_k | pa_k) dx_j} \end{aligned}$$

- All components which do not include  $x_i$  will cancel between numerator and denominator
- The only remaining components are:
  - $p(x_i | pa_i)$  the probability of  $x_i$  given its parents
  - $p(x_j | pa_j)$  where  $pa_j$  includes  $x_i \Rightarrow$  the children of  $x_i$  with their *co-parents*

# Markov blanket (or boundary)

## d-separation

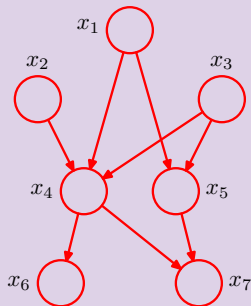
- Each parent  $x_j$  of  $x_i$  will be head-to-tail or tail-to-tail in the path btw  $x_i$  and any of  $x_j$  other neighbours  $\Rightarrow$  blocked
- Each child  $x_j$  of  $x_i$  will be head-to-tail in the path btw  $x_i$  and any of  $x_j$  children  $\Rightarrow$  blocked
- Each co-parent  $x_k$  of a child  $x_j$  of  $x_i$  be head-to-tail or tail-to-tail in the path btw  $x_j$  and any of  $x_k$  other neighbours  $\Rightarrow$  blocked



# Joint probability in Bayesian networks revised

## Why it works

- Let's number nodes in a BN by levels, starting from the roots and following the arrows.
- The product rule allows to decompose the joint probability in a way such that no node is conditioned on its descendants:



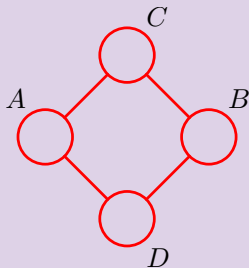
$$p(x_1, \dots, x_K) = p(x_K | x_1, \dots, x_{K-1}) \dots p(x_2 | x_1) p(x_1)$$

- The Markov blanket of a variable having no descendants is given by its parents only, thus the joint simplifies to:

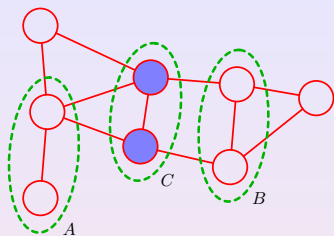
$$p(x_1, \dots, x_K) = \prod_{i=1}^K p(x_i | \text{pa}_i)$$

## Undirected graphical models

- Graphically model the joint probability of a set of variables encoding independency assumptions (as BN)
- Do not assign a directionality to pairwise interactions
- Do not assume that local interactions are proper conditional probabilities (more freedom in parametrizing them)

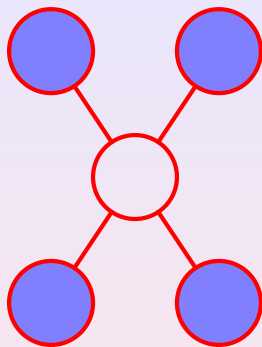


# Markov Networks (MN)



## d-separation definition

- Given  $A, B, C$  arbitrary nonintersecting sets of nodes
- The sets  $A$  and  $B$  are *d-separated* by  $C$  if:
  - All paths from any node in  $A$  to any node in  $B$  are *blocked*
- A path is blocked if it includes at least one node from  $C$  (much simpler than in BN)
- The sets  $A$  and  $B$  are independent given  $C$  ( $A \perp\!\!\!\perp B \mid C$ ) if they are d-separated by  $C$ .



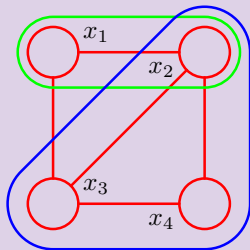
## Markov blanket

- The Markov blanket of a variable is simply the set of its immediate neighbour

# Markov Networks (MN)

## Factorization properties

- We need to decompose the joint probability in *factors* (like  $P(x_i|pa_i)$  in BN) in a way consistent with the independency assumptions.
- Two nodes  $x_i$  and  $x_j$  not connected by a link are independent given the rest of the network (any path between them contains at least another node)  $\Rightarrow$  they should stay in separate factors
- Factors are associated with *cliques* in the network (i.e. fully connected subnetworks)
- A possible option is to associate factors only with *maximal* cliques



## Joint probability (1)

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

- $\psi_C(\mathbf{x}_C) : \text{Val}(\mathbf{x}_C) \rightarrow \mathbb{R}^+$  is a positive function of the value of the variables in clique  $\mathbf{x}_C$  (called clique *potential*)
- The joint probability is the (normalized) product of clique potentials for all (maximal) cliques in the network
- The *partition function*  $Z$  is a normalization constant assuring that the result is a probability (i.e.  $\sum_{\mathbf{x}} p(\mathbf{x}) = 1$ ):

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

## Joint probability (2)

- In order to guarantee that potential functions are positive, they are typically represented as exponentials:

$$\psi_C(\mathbf{x}_C) = \exp(-E(\mathbf{x}_C))$$

- $E(\mathbf{x}_C)$  is called *energy* function: low energy means highly probable configuration
- The joint probability becomes a sum of exponentials:

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left(-\sum_C E(\mathbf{x}_C)\right)$$

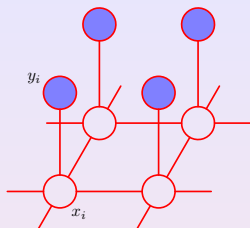
## Comparison to BN

- Advantage:
  - More freedom in defining the clique potentials as they don't need to represent a probability distribution
- Problem:
  - The partition function  $Z$  has to be computed summing over all possible values of the variables
- Solutions:
  - Intractable in general
  - Efficient algorithms exists for certain types of models (e.g. linear chains)
  - Otherwise  $Z$  is approximated

## Description

- We are given an image as a grid of binary pixels  $y_i \in \{-1, 1\}$
- The image is a noisy version of the original one, where each pixel was flipped with a certain (low) probability
- The aim is designing a model capable of recovering the clean version

# MN Example: image de-noising



## Markov network

- We represent the true image as grid of points  $x_i$ , and connect each point to its observed noisy version  $y_i$
- The MN has two maximal clique types:
  - $\{x_i, y_i\}$  for the true/observed pixel pair
  - $\{x_i, x_j\}$  for neighbouring pixels
- The MN has one unobserved non-maximal clique type: the true pixel  $\{x_i\}$

## Energy functions

- We assume low noise  $\rightarrow$  the true pixel should usually have same sign as the observed one:

$$E(x_i, y_i) = -\alpha x_i y_i \quad \alpha > 0$$

same sign pixels have lower energy  $\rightarrow$  higher probability

- We assume neighbouring pixels should usually have same sign (locality principle, e.g. background colour):

$$E(x_i, x_j) = -\beta x_i x_j \quad \beta > 0$$

- We assume the image can have preference for a certain pixel value (either positive or negative):

$$E(x_i) = \gamma x_i$$

## Joint probability

- The joint probability becomes:

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}) &= \frac{1}{Z} \exp \sum_{(\mathbf{x}, \mathbf{y})_c} (-E((\mathbf{x}, \mathbf{y})_c)) \\ &= \frac{1}{Z} \exp \left( \alpha \sum_{i,j} x_i x_j + \beta \sum_i x_i y_i - \gamma \sum_i x_i \right) \end{aligned}$$

- $\alpha, \beta, \gamma$  are *parameters* of the model.

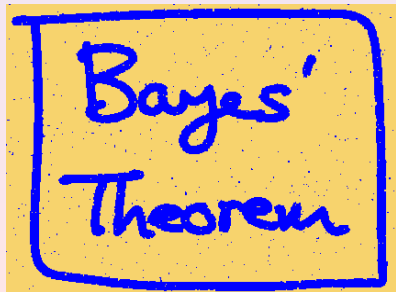
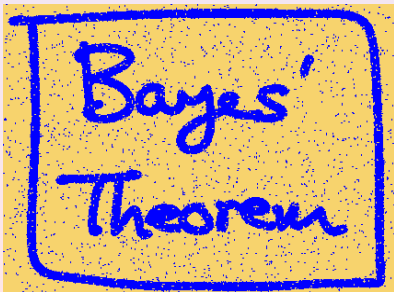
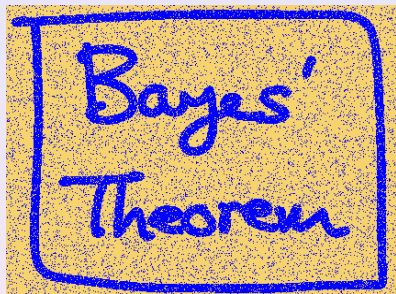
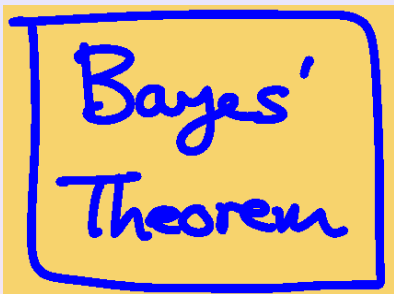
## Note

- all cliques of the same type share the same parameter (e.g.  $\alpha$  for cliques  $\{x_i, x_j\}$ )
- This *parameter sharing* (or tying) allows to:
  - reduce the number of parameters, simplifying learning
  - build a network *template* which will work on images of different sizes.

## Inference

- Assuming parameters are given, the network can be used to produce a clean version of the image:
  - 1 we replace  $y_i$  with its observed value for each  $i$
  - 2 we compute the most most probable configuration for the true pixels given the observed ones
- The second computation is complex and can be done in approximate or exact ways (we'll see) with results of different quality.

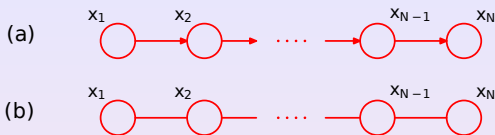
# MN Example: image de-noising



## Independency assumptions

- Directed and undirected graphs are different ways of representing conditional independency assumptions
- Some assumptions can be perfectly represented in both formalisms, some in only one, some in neither of the two
- However there is a tight connection between the two
- It is rather simple to convert a *specific* directed graphical model into an undirected one (but different directed graphs can be mapped to the same undirected graph)
- The converse is more complex because of normalization issues

# From an directed to an undirected model



## Linear chain

- The joint probability in the directed model (a) is:

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

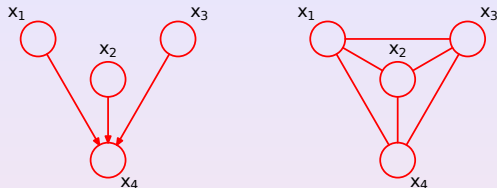
- The joint probability in the undirected model (b) is:

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

- In order for the latter to represent the same distribution as the former, we set:

$$\begin{aligned} \psi_{1,2}(x_1, x_2) &= p(x_1)p(x_2|x_1) & \psi_{2,3}(x_2, x_3) &= p(x_3|x_2) \\ \cdots & & \psi_{N-1,N}(x_{N-1}, x_N) &= p(x_N|x_{N-1}) & Z &= 1 \end{aligned}$$

# From an directed to an undirected model



## Multiple parents

- If a node has multiple parents, we need to construct a clique containing all of them, in order for its potential to represent the conditional probability:
  - We add an undirected link between each pair of parents of the node
  - The process is called *moralization* (marrying the parents! old fashioned..)

# From an directed to an undirected model

## Generic graph

- 1 Build the *moral graph* connecting the parents in the directed graphs and dropping all arrows
- 2 initialize all clique potentials to 1
- 3 for each conditional probability, choose a clique containing all its variables, and multiply its potential with the conditional probability
- 4 set  $Z = 1$

# Inference in graphical models

## Description

- Assume we have evidence  $\mathbf{e}$  on the state of a subset of variables in the model  $\mathbf{E}$
- Inference amounts at computing the posterior probability of a subset  $\mathbf{X}$  of the non-observed variables given the observations:

$$p(\mathbf{X} | \mathbf{E} = \mathbf{e})$$

## Note

- When we need to distinguish between variables and their values, we will indicate random variables with uppercase letters, and their values with lowercase ones.

## Efficiency

- We can always compute the posterior probability as the ratio of two joint probabilities:

$$p(\mathbf{X}|\mathbf{E} = \mathbf{e}) = \frac{p(\mathbf{X}, \mathbf{E} = \mathbf{e})}{p(\mathbf{E} = \mathbf{e})}$$

- The problem consists of estimating such joint probabilities when dealing with a large number of variables
- Directly working on the full joint probabilities requires time exponential in the number of variables
- For instance, if all  $N$  variables are discrete and take one of  $K$  possible values, a joint probability table has  $K^N$  entries
- We would like to exploit the structure in graphical models to do inference more efficiently.

# Inference in graphical models

## Inference on a chain (1)

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

- The marginal probability of an arbitrary  $x_n$  is:

$$p(x_n) = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} p(\mathbf{x})$$

- Only the  $\psi_{N-1, N}(x_{N-1}, x_N)$  is involved in the last summation which can be computed first:

$$\mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1, N}(x_{N-1}, x_N)$$

giving a function of  $x_{N-1}$  which can be used to compute the successive summation:

$$\mu_\beta(x_{N-2}) = \sum_{x_{N-1}} \psi_{N-2, N-1}(x_{N-2}, x_{N-1}) \mu_\beta(x_{N-1})$$

# Inference in graphical models

## Inference on a chain (2)

- The same procedure can be applied starting from the other end of the chain, giving

$$\mu_{\alpha}(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2)$$

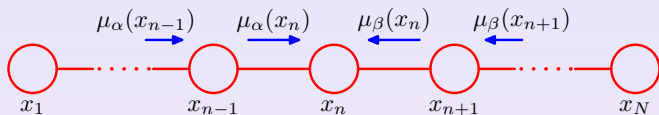
up to  $\mu_{\alpha}(x_n)$

- The marginal probability is thus computed from the normalized contribution coming from both ends as:

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

- The partition function  $Z$  can now be computed just summing  $\mu_{\alpha}(x_n) \mu_{\beta}(x_n)$  over all possible values of  $x_n$

# Inference in graphical models



## Inference as message passing

- We can think of  $\mu_\alpha(x_n)$  as a message passing from  $x_{n-1}$  to  $x_n$

$$\mu_\alpha(x_n) = \sum_{x_{n-1}} \psi_{x_{n-1}, x_n} \mu_\alpha(x_{n-1})$$

- We can think of  $\mu_\beta(x_n)$  as a message passing from  $x_{n+1}$  to  $x_n$

$$\mu_\beta(x_n) = \sum_{x_{n+1}} \psi_{x_n, x_{n+1}} \mu_\beta(x_{n+1})$$

- Each outgoing message is obtained multiplying the incoming message by the clique potential, and summing over the node values

## Full message passing

- Suppose we want to know marginal probabilities for a number of different variables  $x_i$ :
  - 1 We send a message from  $\mu_\alpha(x_1)$  up to  $\mu_\alpha(x_N)$
  - 2 We send a message from  $\mu_\beta(x_N)$  down to  $\mu_\beta(x_1)$
- If all nodes store messages, we can compute any marginal probability as

$$p(x_i) = \mu_\alpha(x_i)\mu_\beta(x_i)$$

for any  $i$  having sent just a double number of messages wrt a single marginal computation

# Inference in graphical models

## Adding evidence

- If some nodes  $\mathbf{x}_e$  are observed, we simply use their observed values instead of summing over all possible values when computing their messages
- After normalization by the partition function  $Z$ , this gives the conditional probability given the evidence

## Joint distribution for clique variables

- It is easy to compute the joint probability for variables in the same clique:

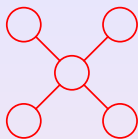
$$p(x_{n-1}, x_n) = \frac{1}{Z} \mu_\alpha(x_{n-1}) \psi_{x_{n-1}, x_n}(x_{n-1}, x_n) \mu_\beta(x_n)$$

## Directed models

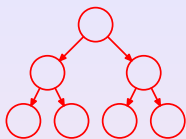
- The procedure applies straightforwardly to a directed model, replacing clique potentials with conditional probabilities
- The marginal  $p(X_n)$  does not need to be normalized as its already a correct distribution
- When adding evidence, the message passing procedure computes the joint probability of the variable and the evidence, and it has to be normalized to obtain the conditional probability:

$$p(X_n | \mathbf{X}_e = \mathbf{x}_e) = \frac{p(X_n, \mathbf{X}_e = \mathbf{x}_e)}{\sum_{X_n} p(X_n, \mathbf{X}_e = \mathbf{x}_e)}$$

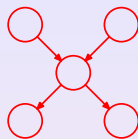
# Inference



(a)



(b)



(c)

## Inference on trees

- Efficient inference can be computed for the broadened family of tree-structured models:

**undirected trees (a)** undirected graphs with a single path for each pair of nodes

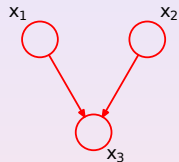
**directed trees (b)** directed graphs with a single node (the root) with no parents, and all other nodes with a single parent

**directed polytrees (c)** directed graphs with multiple parents for node and multiple roots, but still as single (undirected) path between each pair of nodes

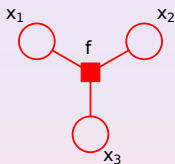
## Description

- Efficient inference algorithms can be better explained using an alternative graphical representation called *factor graph*
- A *factor graph* is a graphical representation of a (directed or undirected) graphical model highlighting its factorization (conditional probabilities or clique potentials)
- The factor graph has one node for each node in the original graph
- The factor graph has one additional node (of a different type) for each factor
- A factor node has undirected links to each of the node variables in the factor

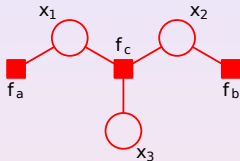
# Factor graphs: examples



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

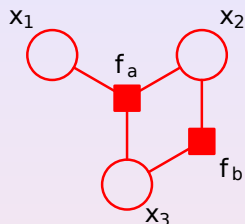
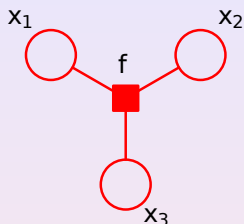
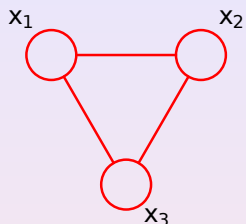


$$f(x_1, x_2, x_3) = p(x_3|x_1, x_2)p(x_1)p(x_2)$$



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

# Factor graphs: examples



## Note

- $f$  is associated to the whole clique potential
- $f_a$  is associated to a maximal clique
- $f_b$  is associated to a non-maximal clique

## The sum-product algorithm

- The *sum-product* algorithm is an efficient algorithm for exact inference on tree-structured graphs
- It is a message passing algorithm as its simpler version for chains
- We will present it on factor graphs, thus unifying directed and undirected models
- We will assume a tree-structured graph, giving rise to a factor graph which is a tree

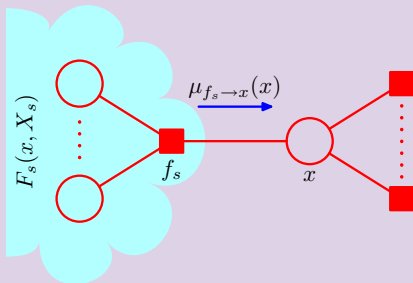
## Computing marginals

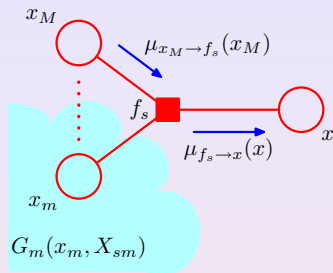
- We want to compute the marginal probability of  $x$ :

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

- Generalizing the message passing scheme seen for chains, this can be computed as the product of messages coming from all neighbouring factors  $f_s$ :

$$p(x) = \frac{1}{Z} \prod_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x)$$

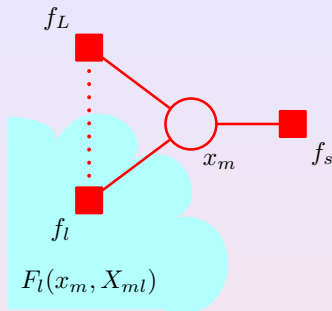




## Factor messages

- Each factor message is the product of messages coming from nodes other than  $x$ , times the factor, summed over all possible values of the factor variables other than  $x$  ( $x_1, \dots, x_M$ ):

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



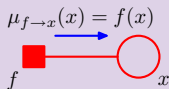
## Node messages

- Each message from node  $x_m$  to factor  $f_s$  is the product of the factor messages to  $x_m$  coming from factors other than  $f_s$ :

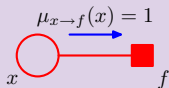
$$\mu_{x_m \rightarrow f_s}(X_m) = \prod_{l \in \text{ne}(x_m) \setminus f_s} \mu_{f_l \rightarrow x_m}(X_m)$$

## Initialization

- Message passing start from leaves, either factors or nodes
- Messages from leaf factors are initialized to the factor itself (there will be no  $x_m$  different from the destination on which to sum over)



- Messages from leaf nodes are initialized to 1



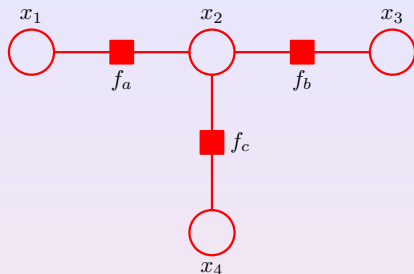
## Message passing scheme

- The node  $x$  whose marginal has to be computed is designed as root.
- Messages are sent from all leaves to their neighbours
- Each internal node sends its message towards the root as soon as it received messages from all other neighbours
- Once the root has collected all messages, the marginal can be computed as the product of them

## Full message passing scheme

- In order to be able to compute marginals for any node, messages need to pass in all directions:
  - 1 Choose an arbitrary node as root
  - 2 Collect messages for the root starting from leaves
  - 3 Send messages from the root down to the leaves
- All messages passed in all directions using only twice the number of computations used for a single marginal

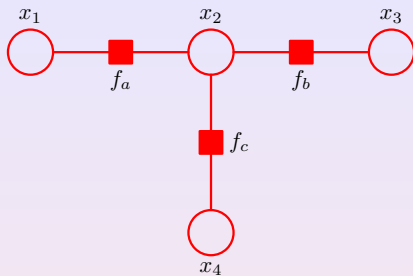
# Inference example



Consider the unnormalized distribution

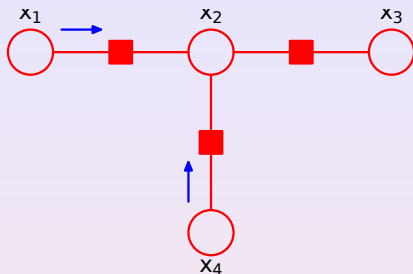
$$\tilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

# Inference example



Choose  $x_3$  as root

# Inference example

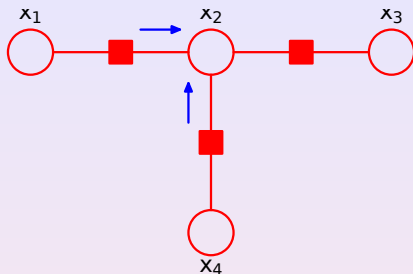


Send initial messages from leaves

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

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

# Inference example

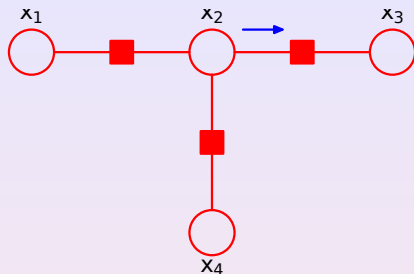


Send messages from factor nodes to  $x_2$

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

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

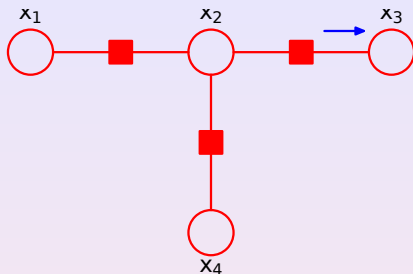
# Inference example



Send message from  $x_2$  to factor node  $f_b$

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

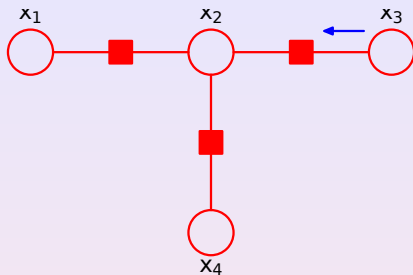
# Inference example



Send message from  $f_b$  to  $x_3$

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

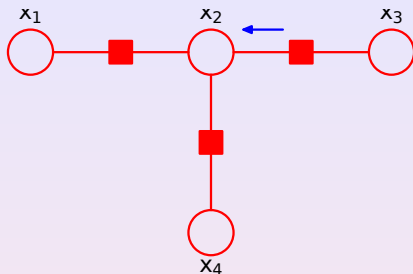
# Inference example



Send message from root  $x_3$

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

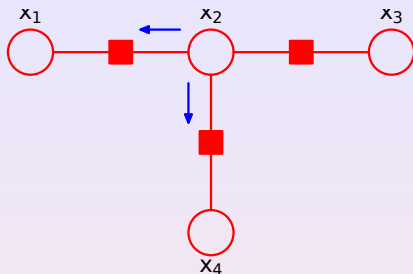
# Inference example



Send message from  $f_b$  to  $x_2$

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

# Inference example

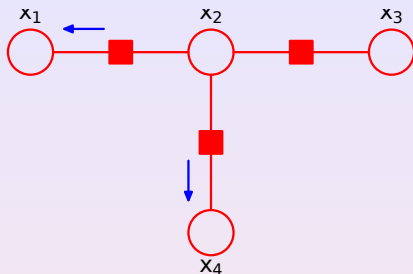


Send messages from  $x_2$  to factor nodes

$$\mu_{x_2 \rightarrow f_a}(X_2) = \mu_{f_b \rightarrow x_2}(X_2) \mu_{f_c \rightarrow x_2}(X_2)$$

$$\mu_{x_2 \rightarrow f_c}(X_2) = \mu_{f_b \rightarrow x_2}(X_2) \mu_{f_a \rightarrow x_2}(X_2)$$

# Inference example

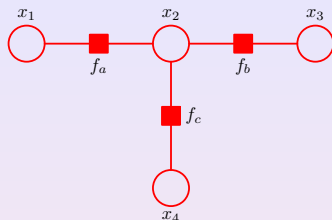


Send messages from factor nodes to leaves

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

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

# Inference example



Compute for instance the marginal for  $x_2$

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

## Normalization

$$p(x) = \frac{1}{Z} \prod_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x)$$

- If the factor graph came from a directed model, a correct probability is already obtained as the product of messages ( $Z = 1$ )
- If the factor graph came from an undirected model, we simply compute the normalization  $Z$  summing the product of messages over all possible values of the target variable:

$$Z = \sum_x \prod_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x)$$

## Factor marginals

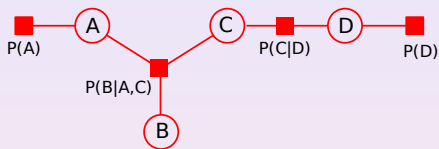
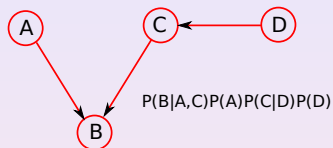
$$p(\mathbf{x}_s) = \frac{1}{Z} f_s(\mathbf{x}_s) \prod_{m \in \text{ne}(f_s)} \mu_{x_m \rightarrow f_s}(x_m)$$

- factor marginals can be computed from the product of neighbouring nodes messages and the factor itself

## Adding evidence

- If some nodes  $\mathbf{x}_e$  are observed, we simply use their observed values instead of summing over all possible values when computing their messages
- After normalization by the partition function  $Z$ , this gives the conditional probability given the evidence

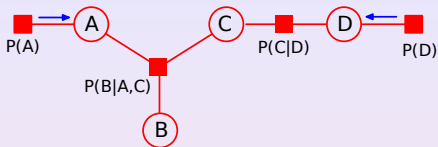
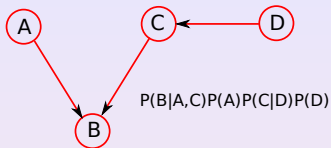
# Inference example



## Directed model

- Take a directed graphical models
- Build a factor graph representing it
- Compute the marginal for a variable (e.g.  $B$ )

# Inference example



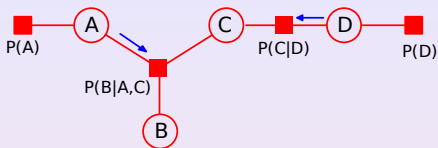
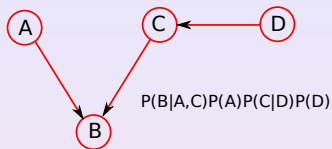
Compute the marginal for  $B$

- Leaf factor nodes send messages:

$$\mu_{f_A \rightarrow A} = P(A)$$

$$\mu_{f_D \rightarrow D} = P(D)$$

# Inference example



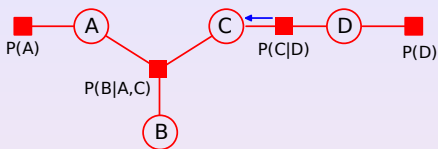
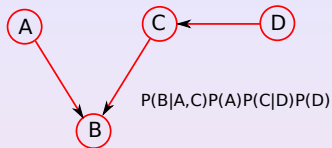
Compute the marginal for  $B$

- $A$  and  $D$  send messages:

$$\mu_{A \rightarrow f_{A,B,C}}(A) = \mu_{f_A \rightarrow A} = P(A)$$

$$\mu_{D \rightarrow f_{C,D}}(D) = \mu_{f_D \rightarrow D} = P(D)$$

# Inference example

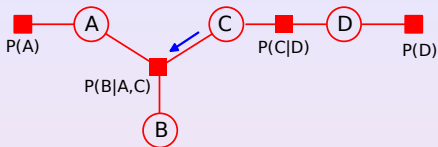
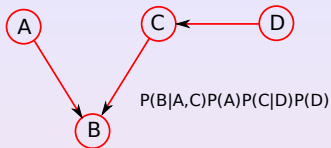


Compute the marginal for  $B$

- $f_{C,D}$  sends message:

$$\mu_{f_{C,D} \rightarrow C}(C) = \sum_D P(C|D) \mu_{f_{D \rightarrow D}} = \sum_D P(C|D) P(D)$$

# Inference example

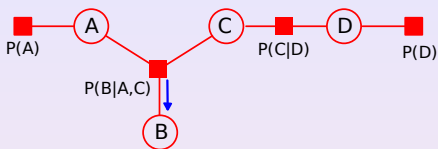
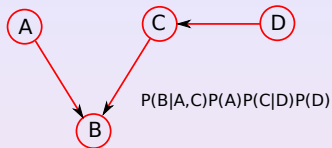


Compute the marginal for  $B$

- $C$  sends message:

$$\mu_{C \rightarrow A, B, C}(C) = \mu_{f_{C, D} \rightarrow C}(C) = \sum_D P(C|D)P(D)$$

# Inference example

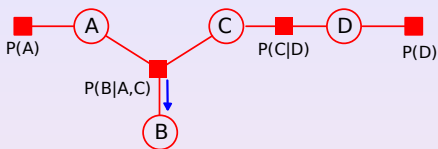
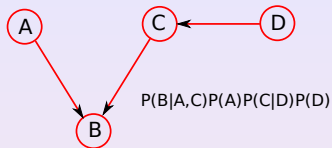


Compute the marginal for  $B$

- $f_{A,B,C}$  sends message:

$$\begin{aligned}\mu_{f_{A,B,C} \rightarrow B}(B) &= \sum_A \sum_C P(B|A, C) \mu_{C \rightarrow f_{A,B,C}}(C) \mu_{A \rightarrow f_{A,B,C}}(A) \\ &= \sum_A \sum_C P(B|A, C) P(A) \sum_D P(C|D) P(D)\end{aligned}$$

# Inference example

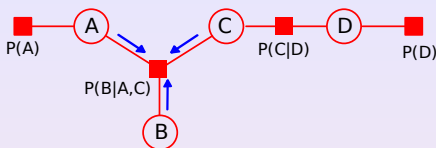
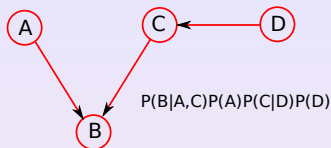


## Compute the marginal for $B$

- The desired marginal is obtained:

$$\begin{aligned}P(B) &= \mu_{f_{A,B,C} \rightarrow B}(B) = \sum_A \sum_C P(B|A, C)P(A) \sum_D P(C|D)P(D) \\ &= \sum_A \sum_C \sum_D P(B|A, C)P(A)P(C|D)P(D) \\ &= \sum_A \sum_C \sum_D P(A, B, C, D)\end{aligned}$$

# Inference example



## Compute the joint for $A, B, C$

- Send messages to factor node  $f_{A,B,C}$

$$\begin{aligned}P(A, B, C) &= f_{A,B,C}(A, B, C) \cdot \mu_{A \rightarrow f_{A,B,C}}(A) \cdot \mu_{C \rightarrow f_{A,B,C}}(C) \cdot \\ &\quad \cdot \mu_{B \rightarrow f_{A,B,C}}(B) \\ &= P(B|A, C) \cdot P(A) \cdot \sum_D P(C|D)P(D) \cdot 1 \\ &= \sum_D P(B|A, C)P(A)P(C|D)P(D) \\ &= \sum_D P(A, B, C, D)\end{aligned}$$

## Finding the most probable configuration

- Given a joint probability distribution  $p(\mathbf{x})$
- We wish to find the configuration for variables  $\mathbf{x}$  having the highest probability:

$$\mathbf{x}^{\max} = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{x})$$

for which the probability is:

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x})$$

## Note

- We want the configuration which is *jointly* maximal for all variables
- We cannot simply compute  $p(x_i)$  for each  $i$  (using the sum-product algorithm) and maximize it

## The max-product algorithm

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \cdots \max_{x_M} p(\mathbf{x})$$

- As for the sum-product algorithm, we can exploit the distribution factorization to efficiently compute the maximum
- It suffices to replace sum with max in the sum-product algorithm

## Linear chain

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

## Message passing

- As for the sum-product algorithm, the max-product can be seen as message passing over the graph.
- The algorithm is thus easily applied to tree-structured graphs via their factor trees:

$$\mu_{f \rightarrow x}(x) = \max_{x_1, \dots, x_M} \left[ f(x, x_1, \dots, x_M) \prod_{m \in ne(f) \setminus x} \mu_{x_m \rightarrow f}(x_m) \right]$$
$$\mu_{x \rightarrow f}(x) = \prod_{l \in ne(x) \setminus f} \mu_{f_l \rightarrow x}(x)$$

## Recovering maximal configuration

- Messages are passed from leaves to an arbitrarily chosen root  $x_r$
- The probability of maximal configuration is readily obtained as:

$$p(\mathbf{x}^{\max}) = \max_{x_r} \left[ \prod_{l \in ne(x_r) \setminus f} \mu_{f_l \rightarrow x_r}(x_r) \right]$$

- The maximal configuration for the root is obtained as:

$$x_r^{\max} = \operatorname{argmax}_{x_r} \left[ \prod_{l \in ne(x_r) \setminus f} \mu_{f_l \rightarrow x_r}(x_r) \right]$$

- We need to recover maximal configuration for the other variables

## Recovering maximal configuration

- When sending a message towards  $x$ , each factor node should store the configuration of the other variables which gave the maximum:

$$\phi_{f \rightarrow x}(x) = \operatorname{argmax}_{x_1, \dots, x_M} \left[ f(x, x_1, \dots, x_M) \prod_{m \in \operatorname{ne}(f) \setminus x} \mu_{x_m \rightarrow f}(x_m) \right]$$

- When the maximal configuration for the root node  $x_r$  has been obtained, it can be used to retrieve the maximal configuration for the variables in neighbouring factors from:

$$x_1^{\max}, \dots, x_M^{\max} = \phi_{f \rightarrow x_r}(x_r^{\max})$$

- The procedure can be repeated *back-tracking* to the leaves, retrieving maximal values for all variables

# Recovering maximal configuration

## Example for linear chain

$$x_N^{\max} = \operatorname{argmax}_{x_N} \mu_{f_{N-1,N} \rightarrow x_N}(x_N)$$

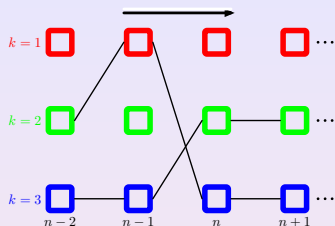
$$x_{N-1}^{\max} = \phi_{f_{N-1,N} \rightarrow x_N}(x_N^{\max})$$

$$x_{N-2}^{\max} = \phi_{f_{N-2,N-1} \rightarrow x_{N-1}}(x_{N-1}^{\max})$$

⋮

$$x_1^{\max} = \phi_{f_{1,2} \rightarrow x_2}(x_2^{\max})$$

# Recovering maximal configuration



## Trellis for linear chain

- A *trellis* or *lattice* diagram shows the  $K$  possible states of each variable  $x_n$  one per row
- For each state  $k$  of a variable  $x_n$ ,  $\phi_{f_{n-1,n} \rightarrow x_n}(x_n)$  defines a unique (maximal) previous state, linked by an edge in the diagram
- Once the maximal state for the last variable  $x_N$  is chosen, the maximal states for other variables are recovering following the edges backward.

## Underflow issues

- The max-product algorithm relies on products (no summation)
- Products of many small probabilities can lead to underflow problems
- This can be addressed computing the logarithm of the probability instead
- The logarithm is monotonic, thus the proper maximal configuration is recovered:

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

- The effect is replacing products with sums (of logs) in the max-product algorithm, giving the *max-sum* one

## Instances of inference algorithms

- The sum-product and max-product algorithms are generic algorithms for exact inference in tree-structured models.
- Inference algorithms for certain specific types of models can be seen as special cases of them
- We will see examples (forward-backward as sum-product and Viterbi as max-product) for Hidden Markov Models and Conditional Random Fields

## Exact inference on general graphs

- The sum-product and max-product algorithms can be applied to tree-structured graphs
- Many practical application require graphs with loops
- An extension of this algorithms to generic graphs can be achieved with the *junction tree algorithm*
- The algorithm does not work on factor graphs, but on *junction trees*, tree-structured graphs with nodes containing clusters of variables of the original graph
- A message passing scheme analogous to the sum-product and max-product algorithms is run on the junction tree

## Problem

- The complexity on the algorithm is exponential on the maximal number of variables in a cluster, making it intractable for large complex graphs.

## Approximate inference

- In cases in which exact inference is intractable, we resort to *approximate* inference techniques
- A number of techniques for approximate inference exist:
  - loopy belief propagation** message passing on the original graph even if it contains loops
  - variational methods** deterministic approximations, assuming the posterior probability (given the evidence) factorizes in a particular way
  - sampling methods** approximate posterior is obtained sampling from the network

## Loopy belief propagation

- Apply sum-product algorithm even if it is not guaranteed to provide an exact solution
- We assume all nodes are in condition of sending messages (i.e. they already received a constant 1 message from all neighbours)
- A *message passing schedule* is chosen in order to decide which nodes start sending messages (e.g. *flooding*, all nodes send messages in all directions at each time step)
- Information flows many times around the graph (because of the loops), each message on a link replaces the previous one and is only based on the most recent messages received from the other neighbours
- The algorithm can eventually converge (no more changes in messages passing through any link) depending on the specific model over which it is applied

## Sampling methods

- Given the joint probability distribution  $p(\mathbf{X})$
- A *sample* from the distribution is an instantiation of all the variables  $\mathbf{X}$  according to the probability  $p$ .
- Samples can be used to approximate the probability of a certain assignment  $\mathbf{Y} = \mathbf{y}$  for a subset  $\mathbf{Y} \subset \mathbf{X}$  of the variables:
  - We simply count the fraction of samples which are consistent with the desired assignment
- We typically need to sample from a posterior distribution given some evidence  $\mathbf{E} = \mathbf{e}$

## Markov Chain Monte Carlo (MCMC)

- We typically cannot directly sample from the posterior distribution (too expensive)
- We can instead build a random process which gradually samples from distributions closer and closer to the posterior
- The state of the process is an instantiation of all the variables
- The process can be seen as randomly traversing the graph of states moving from one state to another with a certain probability.
- After enough time, the probability of being in any particular state is the desired posterior.
- The random process is a Markov Chain

## Note

- The state graph is very different from the graphical model of the joint distribution:
  - nodes in the graphical model are variables
  - nodes in the state graph are instantiations of all the variables

## Definition

- A Markov chain consists of:
  - A space  $Val(\mathbf{X})$  of possible states (one for each possible instantiation of all variables  $\mathbf{X}$ )
  - A *transition probability model* defining for each state  $\mathbf{x} \in Val(\mathbf{X})$  a *next-state* distribution over  $Val(\mathbf{X})$ . We will represent the transition probability from state  $\mathbf{x}$  to  $\mathbf{x}'$  as:

$$T(\mathbf{x} \rightarrow \mathbf{x}')$$

- A Markov chain defines a stochastic process that evolves from state to state

## Homogeneous chains

- A Markov chain is *homogeneous* if its transition probability model does not change over time
- Transition probabilities between states do not depend on the particular time instant in the process
- We will be interested in such chains for sampling

## Chain dynamics

- Consider a sequence of states  $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots$  of the random process
- Being random, the state of the process at time  $t$  can be seen as a random variable  $\mathbf{X}^{(t)}$
- Assume the initial state  $\mathbf{X}^{(0)}$  is distributed according to some initial probability  $p^{(0)}(\mathbf{X}^{(0)})$
- The distribution over subsequent states can be defined using the chain dynamics as:

$$p^{(t+1)}(\mathbf{X}^{(t+1)} = \mathbf{x}') = \sum_{\mathbf{x} \in \text{Val}(\mathbf{X})} p^{(t)}(\mathbf{X}^{(t)} = \mathbf{x}) \mathcal{T}(\mathbf{x} \rightarrow \mathbf{x}')$$

- The probability of being in a certain state  $\mathbf{x}$  at time  $t$  is the sum of the probabilities of being in any possible state  $\mathbf{x}'$  at time  $t - 1$  times the probability of a transition from  $\mathbf{x}'$  to  $\mathbf{x}$

## Convergence

- As the process converges, we expect that  $p^{(t+1)}$  becomes close to  $p^{(t)}$ :

$$p^{(t)}(\mathbf{x}') \approx p^{(t+1)}(\mathbf{x}') = \sum_{\mathbf{x} \in \text{Val}(\mathbf{X})} p^{(t)}(\mathbf{x}) T(\mathbf{x} \rightarrow \mathbf{x}')$$

- At convergence, we expect to have reached an equilibrium distribution  $\pi(\mathbf{X})$ :
  - The probability of being in a state is the same as that of transitioning into it from a random predecessor

## Stationary distribution

- A distribution  $\pi(\mathbf{X})$  is a *stationary distribution* for a Markov Chain  $\mathcal{T}$  if:

$$\pi(\mathbf{X} = \mathbf{x}') = \sum_{\mathbf{x} \in \text{Val}(\mathbf{X})} \pi(\mathbf{X} = \mathbf{x}) \mathcal{T}(\mathbf{x} \rightarrow \mathbf{x}')$$

## Requirements

- We are interested in Markov Chains with:
  - a *unique* stationary distribution
  - which is reachable from any starting distribution  $p^{(0)}$
- There are various conditions to guarantee this property (e.g. *ergodicity*)
- For a Markov Chain with a finite state space  $\text{Val}(\mathbf{X})$ , *regularity* is a sufficient and necessary condition

## Regular chains

A Markov chain is *regular* if there exists a number  $k$  such that for all state pairs  $\mathbf{x}, \mathbf{x}' \in \text{Val}(\mathbf{X})$  the probability of getting from  $\mathbf{x}$  to  $\mathbf{x}'$  in exactly  $k$  steps is greater than zero

## Conditions ensuring regularity

- It is possible to get from any state to any other state using a positive probability path in the state graph
- For every state  $\mathbf{x}$  there is a positive probability of transitioning from  $\mathbf{x}$  to  $\mathbf{x}$  in one step (self loop)

## Markov chains for graphical models

- The typical use is estimating the probability of a certain instantiation  $\mathbf{x}$  of the variables  $\mathbf{X}$  given some evidence  $\mathbf{E} = \mathbf{e}$
- This requires sampling from the posterior distribution  $p(\mathbf{X}|\mathbf{E} = \mathbf{e})$
- We wish to define a chain for which  $p(\mathbf{X}|\mathbf{E} = \mathbf{e})$  is the stationary distribution
- States in the chain should be the subset of all possible instantiations which is consistent with the evidence  $\mathbf{e}$ :

$$\mathbf{x} \in \text{val}(\mathbf{X}) \text{ s.t. } \mathbf{x}_{\mathbf{E}} = \mathbf{e}$$

## Transition model

- A simple transition model consists of updating variables in  $\hat{\mathbf{X}} = \mathbf{X} - \mathbf{E}$  one at a time
- This can be seen as made of a set of  $k = |\hat{\mathbf{X}}|$  *local* transition models  $\mathcal{T}_i$ , one for each variable  $X_i \in \hat{\mathbf{X}}$
- Let  $\mathbf{U}_i = \mathbf{X} - X_i$  and  $\mathbf{u}_i$  a possible instantiation of  $\mathbf{U}_i$
- The local transition model  $\mathcal{T}_i$  defines transitions between states  $(\mathbf{u}_i, x_i)$  and  $(\mathbf{u}_i, x'_i)$
- By defining a transition as a sequence of  $k$  local transitions, one for each variable  $X_i$ , we obtain a homogeneous (i.e. not time-dependent) global transition model

## Gibbs Sampling

- Gibbs sampling is an efficient and effective simple Markov chain for factored state spaces (like the ones in graphical models)
- Local transitions  $\mathcal{T}_i$  are defined “forgetting” about the value of  $X_i$  in the current state.
- This allows to sample a new value according to the posterior probability of  $X_i$  given the rest of the current state:

$$\mathcal{T}_i((\mathbf{u}_i, x_i) \rightarrow (\mathbf{u}_i, x'_i)) = p(x'_i | \mathbf{u}_i)$$

- The Gibbs chain samples a new state performing  $k$  subsequent local moves
- We only take samples after a full round of local moves

## Regularity of Gibbs chains

- Gibbs chains are ensured to be regular if the distribution is *strictly* positive: every value of  $X_i$  given any instantiation of  $\mathbf{U}_i$  has non-zero probability
- In this case we can get from any state to any state in at most  $k = |\hat{\mathbf{X}}|$  local steps
- Gibbs chains have the desired posterior  $p(\mathbf{X}|\mathbf{E} = \mathbf{e})$  as stationary distribution

## Positivity

- Positivity of the distributions is guaranteed if no conditional probability distribution (for BN) or clique potential (for MN) has zero value for any configuration of the variables

## Computing local transition probabilities

- Local transition probabilities can be computed very efficiently for discrete graphical models.
- Only the Markov blanket of  $X_i$  is needed in order to compute its posterior  $p(X_i|\mathbf{u}_i)$
- Other models for which such posterior is not efficiently computable require more complex Markov chain methods such as the *Metropolis-Hastings*

## Generating samples

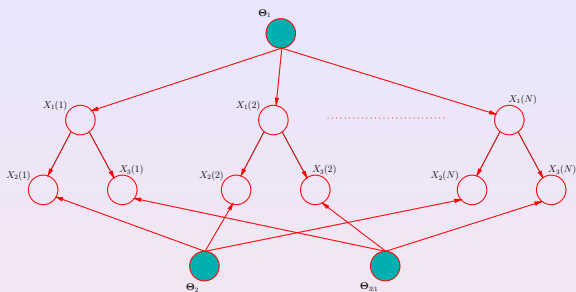
- In order to generate samples from the correct posterior distribution, we need to wait until the Markov chain has converged to the stationary distribution
- Once at convergence, all subsequent samples could be used to estimate the desired probability
- However, consecutive samples are definitely not independent
- A possible approach consists of letting an interval of  $d$  samples before collecting the next sample
- In practice it is often the case that using all samples leads to better estimates if a fixed amount of time is provided (simply because they are based on more even if less independent samples)

## Parameter estimation

- We assume the structure of the model is given
- We are given a dataset of examples  $\mathcal{D} = \{\mathbf{x}(1), \dots, \mathbf{x}(N)\}$
- Each example  $\mathbf{x}(i)$  is a configuration for *all* (complete data) or *some* (incomplete data) variables in the model
- We need to estimate the parameters of the model (CPD or weights of clique potentials) from the data
- The simplest approach consists of learning the parameters maximizing the *likelihood* of the data:

$$\theta^{\max} = \operatorname{argmax}_{\theta} p(\mathcal{D}|\theta) = \operatorname{argmax}_{\theta} \mathcal{L}(\mathcal{D}, \theta)$$

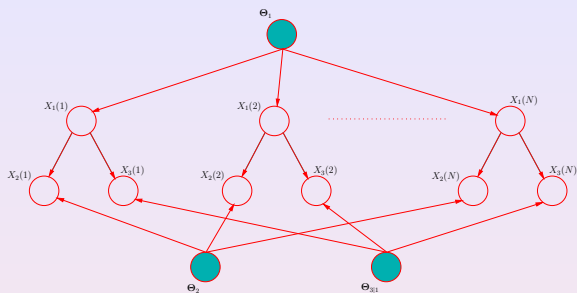
# Learning Bayesian Networks



Maximum likelihood estimation, complete data

$$p(\mathcal{D}|\theta) = \prod_{i=1}^N p(\mathbf{x}(i)|\theta) \quad \text{examples independent given } \theta$$
$$= \prod_{i=1}^N \prod_{j=1}^m p(\mathbf{x}_j(i)|pa_j(i), \theta) \quad \text{factorization for BN}$$

# Learning Bayesian Networks



Maximum likelihood estimation, complete data

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^N \prod_{j=1}^m p(\mathbf{x}_j(i) | \text{pa}_j(i), \boldsymbol{\theta}) \quad \text{factorization for BN}$$

$$= \prod_{i=1}^N \prod_{j=1}^m p(\mathbf{x}_j(i) | \text{pa}_j(i), \boldsymbol{\theta}_{X_j | \text{pa}_j}) \quad \text{disjoint CPD parameters}$$

## Maximum likelihood estimation, complete data

- The parameters of each CPD can be estimated independently:

$$\theta_{X_j|Pa_j}^{\max} = \operatorname{argmax}_{\theta_{X_j|Pa_j}} \underbrace{\prod_{i=1}^N \rho(\mathbf{x}_j(i)|pa_j(i), \theta_{X_j|Pa_j})}_{\mathcal{L}(\theta_{X_j|Pa_j}, \mathcal{D})}$$

- A discrete CPD  $P(X|\mathbf{U})$ , can be represented as a table, with:
  - a number of rows equal to the number  $Val(X)$  of configurations for  $X$
  - a number of columns equal to the number  $Val(\mathbf{U})$  of configurations for its parents  $\mathbf{U}$
  - each table entry  $\theta_{x|\mathbf{u}}$  indicating the probability of a specific configuration of  $X = x$  and its parents  $\mathbf{U} = \mathbf{u}$

# Learning graphical models

## Maximum likelihood estimation, complete data

- Replacing  $p(x(i)|pa(i))$  with  $\theta_{x(i)|\mathbf{u}(i)}$ , the local likelihood of a single CPD becomes:

$$\begin{aligned}\mathcal{L}(\theta_{X|Pa}, \mathcal{D}) &= \prod_{i=1}^N p(x(i)|pa(i), \theta_{X|Pa_j}) \\ &= \prod_{i=1}^N \theta_{x(i)|\mathbf{u}(i)} \\ &= \prod_{\mathbf{u} \in \text{Val}(\mathbf{U})} \left[ \prod_{x \in \text{Val}(X)} \theta_{x|\mathbf{u}}^{N_{\mathbf{u},x}} \right]\end{aligned}$$

where  $N_{\mathbf{u},x}$  is the number of times the specific configuration  $X = x$ ,  $\mathbf{U} = \mathbf{u}$  was found in the data

## Maximum likelihood estimation, complete data

- A column in the CPD table contains a multinomial distribution over values of  $X$  for a certain configuration of the parents  $\mathbf{U}$
- Thus each column should sum to one:  $\sum_x \theta_{x|\mathbf{u}} = 1$
- Parameters of different columns can be estimated independently
- For each multinomial distribution, zeroing the gradient of the maximum likelihood and considering the normalization constraint, we obtain:

$$\theta_{x|\mathbf{u}}^{max} = \frac{N_{\mathbf{u},x}}{\sum_x N_{\mathbf{u},x}}$$

- The maximum likelihood parameters are simply the fraction of times in which the specific configuration was observed in the data

## Adding priors

- ML estimation tends to overfit the training set
- Configuration not appearing in the training set will receive zero probability
- A common approach consists of combining ML with a prior probability on the parameters, achieving a maximum-a-posteriori estimate:

$$\theta^{\max} = \operatorname{argmax}_{\theta} p(\mathcal{D}|\theta)p(\theta)$$

## Dirichlet priors

- The conjugate (read natural) prior for a multinomial distribution is a Dirichlet distribution with parameters  $\alpha_{x|\mathbf{u}}$  for each possible value of  $x$
- The resulting maximum-a-posteriori estimate is:

$$\theta_{x|\mathbf{u}}^{\max} = \frac{N_{\mathbf{u},x} + \alpha_{x|\mathbf{u}}}{\sum_x (N_{\mathbf{u},x} + \alpha_{x|\mathbf{u}})}$$

- The prior is like having observed  $\alpha_{x|\mathbf{u}}$  imaginary samples with configuration  $X = x, \mathbf{U} = \mathbf{u}$

## Incomplete data

- With incomplete data, some of the examples miss evidence on some of the variables
- Counts of occurrences of different configurations cannot be computed if not all data are observed
- The full Bayesian approach of integrating over missing variables is often intractable in practice
- We need approximate methods to deal with the problem

## E-M for Bayesian nets in a nutshell

- Sufficient statistics (counts) cannot be computed (missing data)
- Fill-in missing data inferring them using current parameters (solve inference problem to get *expected* counts)
- Compute parameters maximizing likelihood (or posterior) of such expected counts
- Iterate the procedure to improve quality of parameters

## Maximum likelihood estimation

- For general MN, the likelihood function cannot be decomposed into independent pieces for each potential because of the global normalization ( $Z$ )

$$\mathcal{L}(E, \mathcal{D}) = \prod_{i=1}^N \frac{1}{Z} \exp \left( - \sum_C E_c(\mathbf{x}_c(i)) \right)$$

- However the likelihood is concave in  $E_C$ , the energy functionals whose parameters have to be estimated
- The problem is an unconstrained concave maximization problem solved by gradient ascent (or second order methods)

## Maximum likelihood estimation

- For each configuration  $\mathbf{u}_c \in \text{Val}(\mathbf{x}_c)$  we have a parameter  $E_{c,\mathbf{u}_c} \in \mathbb{R}$  (ignoring parameter tying)
- The partial derivative of the log-likelihood wrt  $E_{c,\mathbf{u}_c}$  is:

$$\begin{aligned}\frac{\partial \log \mathcal{L}(E, \mathcal{D})}{\partial E_{c,\mathbf{u}_c}} &= \sum_{i=1}^N [\delta(\mathbf{x}_c(i), \mathbf{u}_c) - p(\mathbf{X}_c = \mathbf{u}_c | E)] \\ &= N_{\mathbf{u}_c} - N p(\mathbf{X}_c = \mathbf{u}_c | E)\end{aligned}$$

- The derivative is zero when the counts of the data correspond to the expected counts predicted by the model
- In order to compute  $p(\mathbf{X}_c = \mathbf{u}_c | E)$ , inference has to be performed on the Markov network, making learning quite expensive

## Approaches

**constraint-based** test conditional independencies on the data and construct a model satisfying them

**score-based** assign a score to each possible structure, define a search procedure looking for the structure maximizing the score

**model-averaging** assign a prior probability to each structure, and average prediction over all possible structures weighted by their probabilities (full Bayesian, intractable)