Inference in Bayesian Networks

Andrea Passerini
passerini@disi.unitn.it

Machine Learning

Inference in graphical models

Description

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

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

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.

Inference in graphical models

Efficiency

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

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

- 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)

$$p(\mathbf{X}) = p(X_1)p(X_2|X_1)p(X_3|X_2)\cdots p(X_N|X_{N-1})$$

- 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 $p(X_N|X_{N-1})$ is involved in the last summation which can be computed first, giving a function of X_{N-1}:

$$\mu_\beta(X_{N-1}) = \sum_{X_N} p(X_N|X_{N-1})$$

Inference on a chain (2)

- the marginalization can be iterated as:

$$\mu_\beta(X_{N-2}) = \sum_{X_{N-1}} p(X_{N-1}|X_{N-2})\mu_\beta(X_{N-1})$$

down to the desired variable X_n, giving:

$$\mu_\beta(X_n) = \sum_{X_{n+1}} p(X_{n+1}|X_n)\mu_\beta(X_{n+1})$$

Inference in graphical models

Inference on a chain (3)

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

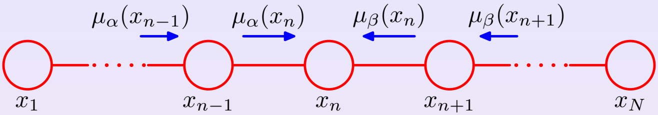
$$\mu_\alpha(X_2) = \sum_{X_1} p(X_1)p(X_2|X_1)$$

  up to $\mu_\alpha(X_n)$

- The marginal probability is now computed as the product of the contributions coming from both ends:

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

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

$$\mu_\alpha(X_n) = \sum_{X_{n-1}} p(X_n|X_{n-1})\mu_\alpha(X_{n-1})$$

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

$$\mu_\beta(X_n) = \sum_{X_{n+1}} p(X_{n+1}|X_n)\mu_\beta(X_{n+1})$$

- Each outgoing message is obtained multiplying the incoming message by the “local” probability, 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

Example

$$p(\mathbf{X}) = p(X_1)p(X_2|X_1)p(X_3|X_2)p(X_4|X_3)$$

- The marginal probability of X_2 and observations $X_1 = x_{e_1}$ and $X_3 = x_{e_3}$ is:

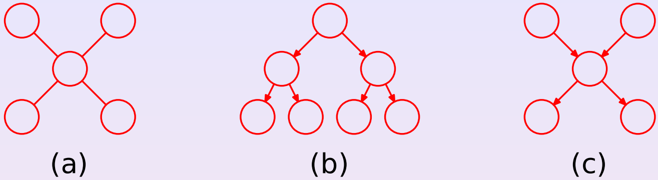
$$p(X_2, X_1 = x_{e_1}, X_3 = x_{e_3}) = p(X_1 = x_{e_1})p(X_2|X_1 = x_{e_1}) \cdot \\ \cdot p(X_3 = x_{e_3}|X_2) \sum_{X_4} p(X_4|X_3 = x_{e_3})$$

Inference

Computing conditional probability given evidence

- 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 *given* the evidence:

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



Inference on trees

- Efficient inference can be computed for the broaded 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

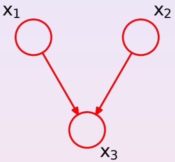
  directed polytrees (c) directed graphs with multiple parents for node and multiple roots, but still a single (undirected) path between each pair of nodes

Factor graphs

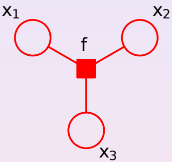
Description

- Efficient inference algorithms can be better explained using an alternative graphical representation called *factor graph*
- A *factor graph* is a graphical representation of a graphical model highlighting its factorization (i.e. conditional probabilities)
- 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

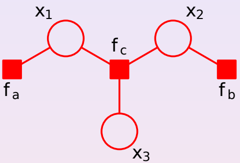
Factor graphs: examples



$p(x3|x1,x2)p(x1)p(x2)$



$f(x1,x2,x3)=p(x3|x1,x2)p(x1)p(x2)$



$fc(x1,x2,x3)=p(x3|x1,x2)$

$fa(x1)=p(x1)$

$fb(x2)=p(x2)$

Inference

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, assuming a tree-structured graph giving rise to a factor graph which is a tree
- The algorithm will be applicable to undirected models (i.e. Markov Networks) as well as directed ones (i.e. Bayesian Networks)

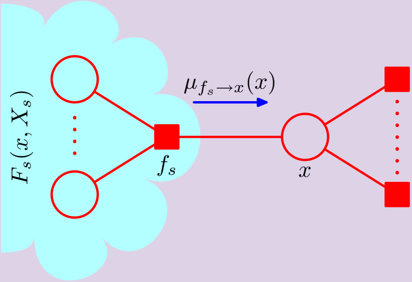
Computing marginals

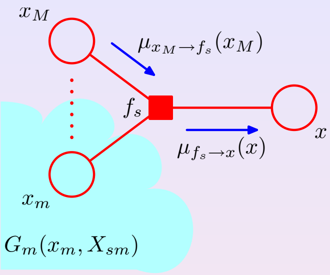
- We want to compute the marginal probability of X:

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

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

$$p(X) = \prod_{f_s \in \mathrm{ne}(X)} \mu_{f_s \to X}(X)$$



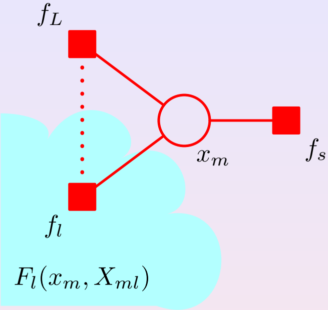


Factor messages

- 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 $(X_1, \ldots, X_M)$:

$$\mu_{f_s \to X}(X) = \sum_{X_1} \cdots \sum_{X_M} f_s(X, X_1, \ldots, X_M) \prod_{X_m \in \mathrm{ne}(f_s) \setminus X} \mu_{X_m \to f_s}(X_m)$$

Inference



Node messages

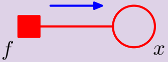
- Each message from node X_m to factor f_s is the product of the factor messages to X_m coming from factors other than f_s:

$$\mu_{X_m \to f_s}(X_m) = \prod_{f_l \in \mathrm{ne}(X_m) \backslash f_s} \mu_{f_l \to X_m}(X_m)$$

Initialization

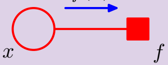
- Message passing start from leaves, either factors or nodes
- Messages from leaf factors are initialized to the factor itself (there will be no X_m different from the destination on which to sum over)

$$\mu_{f \to x}(x) = f(x)$$



- Messages from leaf nodes are initialized to 1

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



Message passing scheme

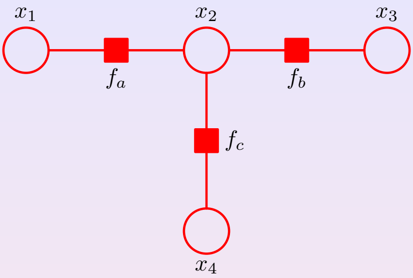
- The node X whose marginal has to be computed is designed as root.
- Messages are sent from all leaves to their neighbours
- Each internal node sends its message towards the root as soon as it received messages from all other neighbours
- Once the root has collected all messages, the marginal can be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

Inference

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

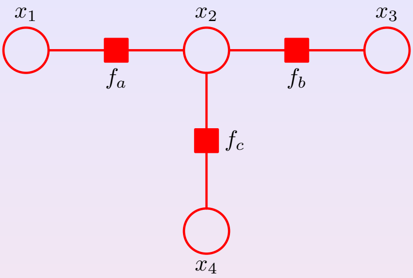
Inference example



Consider the joint distribution as product of factors

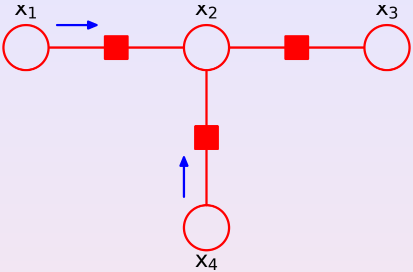
$$p(\mathbf{X}) = f_a(X_1, X_2) f_b(X_2, X_3) f_c(X_2, X_4)$$

Inference example



Choose X_3 as root

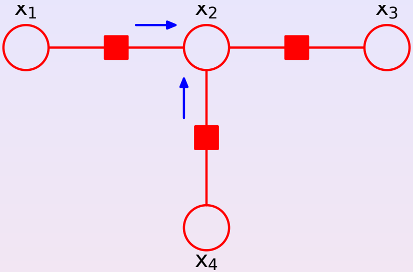
Inference example



Send initial messages from leaves

$$\mu_{X_1 \to f_a}(X_1) = 1$$
$$\mu_{X_4 \to f_c}(X_4) = 1$$

Inference example

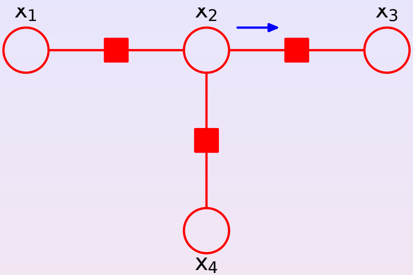


Send messages from factor nodes to X_2

$$\mu_{f_a \to X_2}(X_2) = \sum_{X_1} f_a(X_1, X_2)$$

$$\mu_{f_c \to X_2}(X_2) = \sum_{X_4} f_c(X_2, X_4)$$

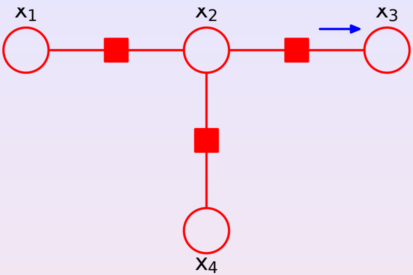
Inference example



Send message from X_2 to factor node f_b

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

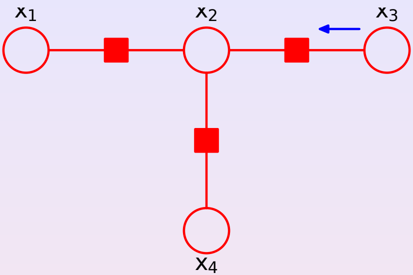
Inference example



Send message from f_b to X_3

$$\mu_{f_b \to X_3}(X_3) = \sum_{X_2} f_b(X_2, X_3)\mu_{X_2 \to f_b}(X_2)$$

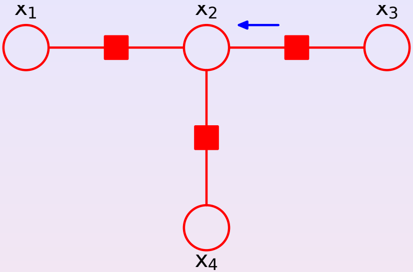
Inference example



Send message from root X_3

$$\mu_{X_3 \to f_b}(X_3) = 1$$

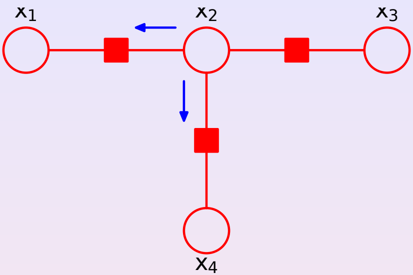
Inference example



Send message from f_b to X_2

$$\mu_{f_b \to X_2}(X_2) = \sum_{X_3} f_b(X_2, X_3)$$

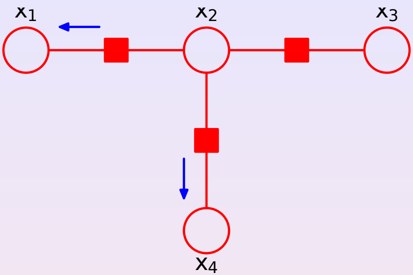
Inference example



Send messages from X_2 to factor nodes

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

Inference example

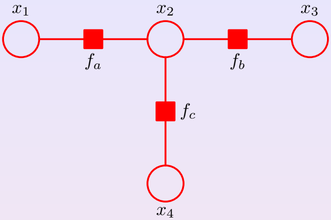


Send messages from factor nodes to leaves

$$\mu_{f_a \to X_1}(X_1) = \sum_{X_2} f_a(X_1, X_2)\mu_{X_2 \to f_a}(X_2)$$

$$\mu_{f_c \to X_4}(X_4) = \sum_{X_2} f_c(X_2, X_4)\mu_{X_2 \to f_c}(X_2)$$

Inference example



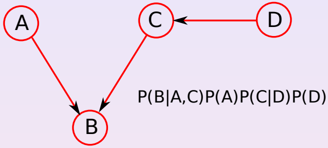
Compute for instance the marginal for X_2

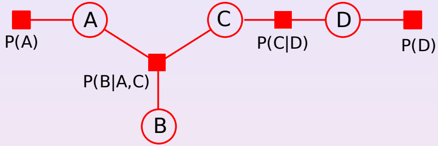
$$
\begin{aligned}
p(X_2) &= \mu_{f_a \to X_2}(X_2)\mu_{f_b \to X_2}(X_2)\mu_{f_c \to X_2}(X_2) \\
&= \left[\sum_{X_1} f_a(X_1, X_2)\right]\left[\sum_{X_3} f_b(X_2, X_3)\right]\left[\sum_{X_4} f_c(X_2, X_4)\right] \\
&= \sum_{X_1}\sum_{X_3}\sum_{X_4} f_a(X_1, X_2) f_b(X_2, X_3) f_c(X_2, X_4) \\
&= \sum_{X_1}\sum_{X_3}\sum_{X_4} p(\mathbf{X})
\end{aligned}
$$

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, this gives the conditional probability given the evidence

Inference example

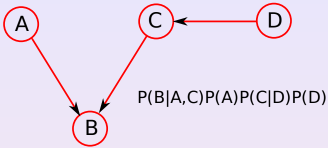


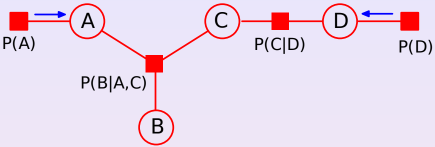


Bayesian network

- Take a Bayesian network
- Build a factor graph representing it
- Compute the marginal for a variable (e.g. B)

Inference example



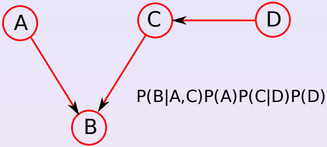


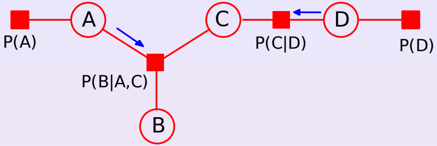
Compute the marginal for B

- Leaf factor nodes send messages:

$$\mu_{f_A \to A} = P(A)$$
$$\mu_{f_D \to D} = P(D)$$

Inference example



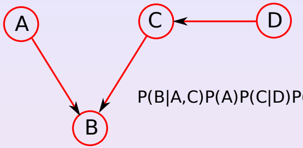


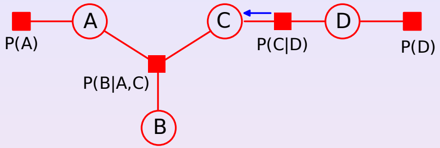
Compute the marginal for *B*

- *A* and *D* send messages:

$$\mu_{A \to f_{A,B,C}}(A) = \mu_{f_A \to A} = P(A)$$
$$\mu_{D \to f_{C,D}}(D) = \mu_{f_D \to D} = P(D)$$

Inference example



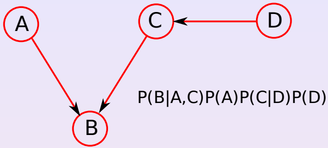


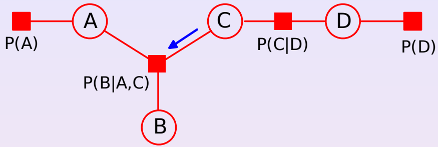
Compute the marginal for B

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

$$\mu_{f_{C,D}\to C}(C) = \sum_{D} P(C|D)\mu_{f_D\to D} = \sum_{D} P(C|D)P(D)$$

Inference example



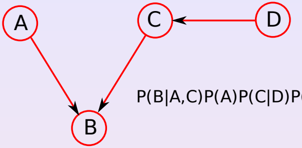


Compute the marginal for B

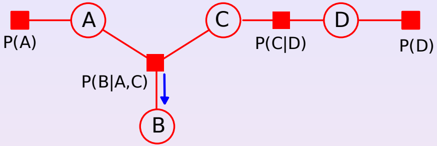
- C sends message:

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

Inference example



P(B|A,C)P(A)P(C|D)P(D)

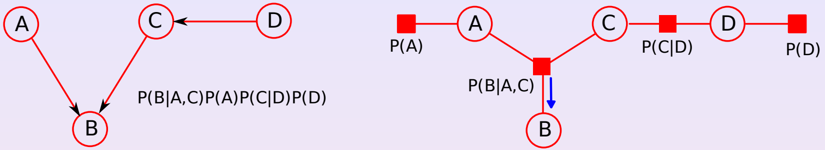


Compute the marginal for B

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

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

Inference example



Compute the marginal for B

- The desired marginal is obtained:

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

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

Inference

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

$$\max_{\mathbf{X}} p(\mathbf{X}) = \max_{X_1} \cdots \max_{X_N} [p(X_1)p(X_2|X_1)\cdots p(X_N|X_{N-1})]$$
$$= \max_{X_1} \left[p(X_1)p(X_2|X_1) \left[\cdots \max_{X_N} p(X_N|X_{N-1}) \right] \right]$$

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:

$$\mu_{f\to X}(X) = \max_{X_1,\dots,X_M}\left[f(X, X_1, \dots, X_M) \prod_{X_m \in ne(f)\setminus X} \mu_{X_m\to f}(X_m)\right]$$

$$\mu_{X\to f}(X) = \prod_{f_l \in ne(X)\setminus f} \mu_{f_l\to X}(X)$$

Recoving maximal configuration

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

$$p(\mathbf{X}^{\max}) = \max_{X_r} \left[\prod_{f_l \in ne(X_r)} \mu_{f_l \to X_r}(X_r) \right]$$

- The maximal configuration for the root is obtained as:

$$X_r^{\max} = \operatorname*{argmax}_{X_r} \left[\prod_{f_l \in ne(X_r)} \mu_{f_l \to X_r}(X_r) \right]$$

- We need to recover maximal configuration for the other variables

Recoving maximal configuration

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

$$\phi_{f \to X}(X) = \underset{X_1,\ldots,X_M}{\text{argmax}} \left[f(X, X_1, \ldots, X_M) \prod_{X_m \in ne(f) \setminus X} \mu_{X_m \to f}(X_m) \right]$$

- 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:

$$X_1^{\text{max}}, \ldots, X_M^{\text{max}} = \phi_{f \to X_r}(X_r^{\text{max}})$$

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

Recoving maximal configuration

Example for linear chain

$$X_N^{\max} = \operatorname*{argmax}_{X_N} \mu_{f_{N-1,N}\to X_N}(X_N)$$

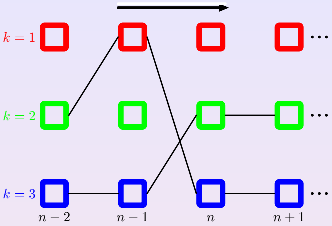
$$X_{N-1}^{\max} = \phi_{f_{N-1,N}\to X_N}(X_N^{\max})$$

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

$$\vdots$$

$$X_1^{\max} = \phi_{f_{1,2}\to X_2}(X_2^{\max})$$

Recoving 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, $\phi_{f_{n-1,n} \to X_n}(X_n)$ 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

Inference

Exact inference on general graphs

- The sum-product and max-product algorithms can be applied to tree-structured graphs
- Many applications require graphs with (undirected) 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.

Inference

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