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 upper-case 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 in graphical models

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:

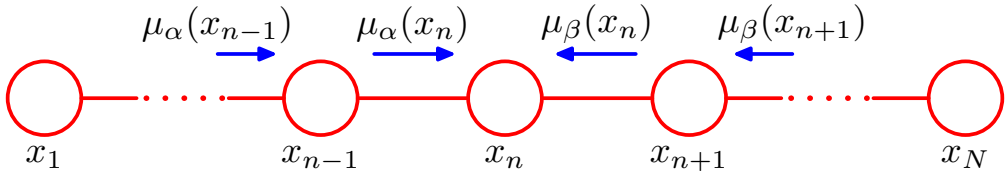
$$\mu_\alpha(X_2) = \sum_{X_1} p(X_1)p(X_2|X_1)$$

  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

Inference in graphical models

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:

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

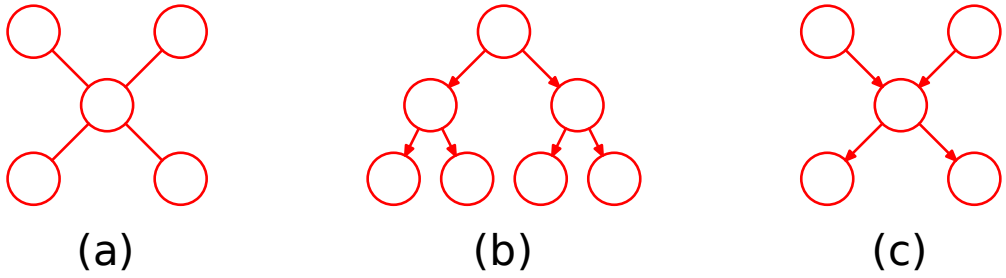
Inference

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



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

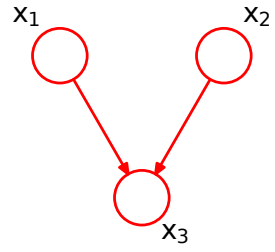
  directed polytrees (c) directed graphs with multiple parents for node and multiple roots, but still a single (undirected) path between each pair of nodes

Factor graphs

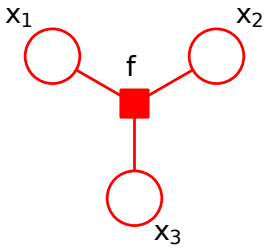
Description

- 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

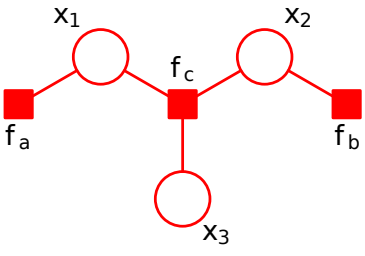
Factor graphs: examples



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)

Inference

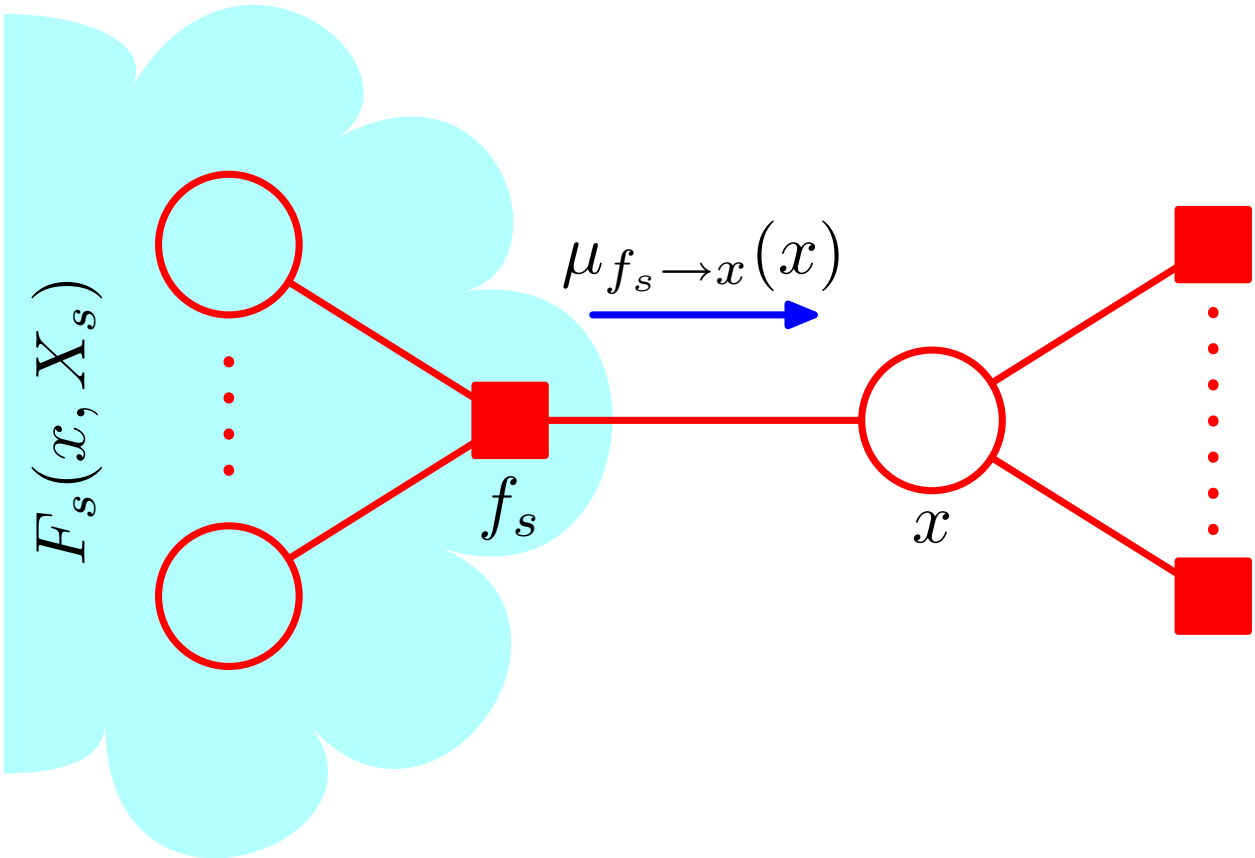
Computing marginals

- We want to compute the marginal probability of X:

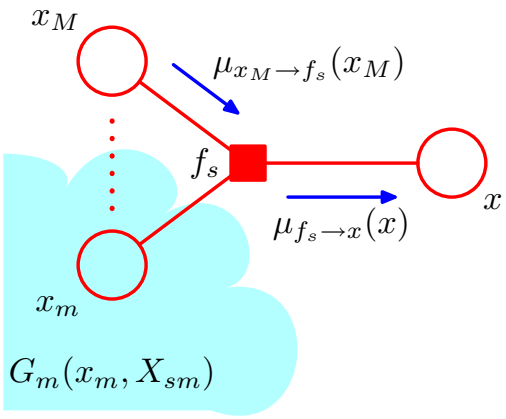
$$p(X) = \sum_{\mathbf{X} \setminus X} p(\mathbf{X})$$

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

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



Inference

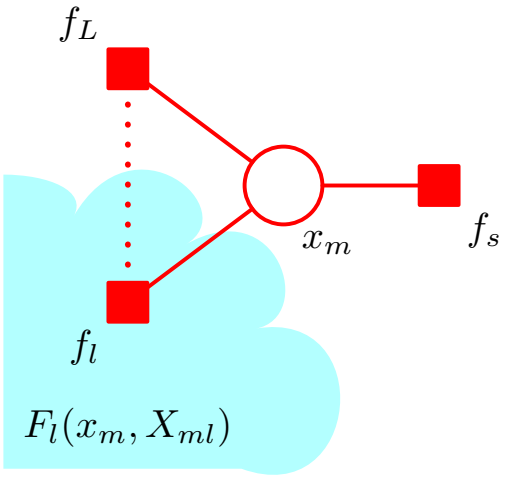


Factor messages

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

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

Inference



Node messages

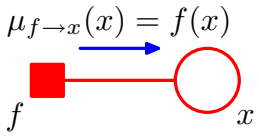
- Each message from node X_m to factor f_s is the product of the factor messages to X_m coming from factors other than f_s:

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

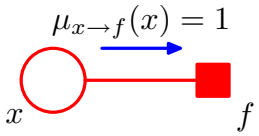
Inference

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



Inference

Message passing scheme

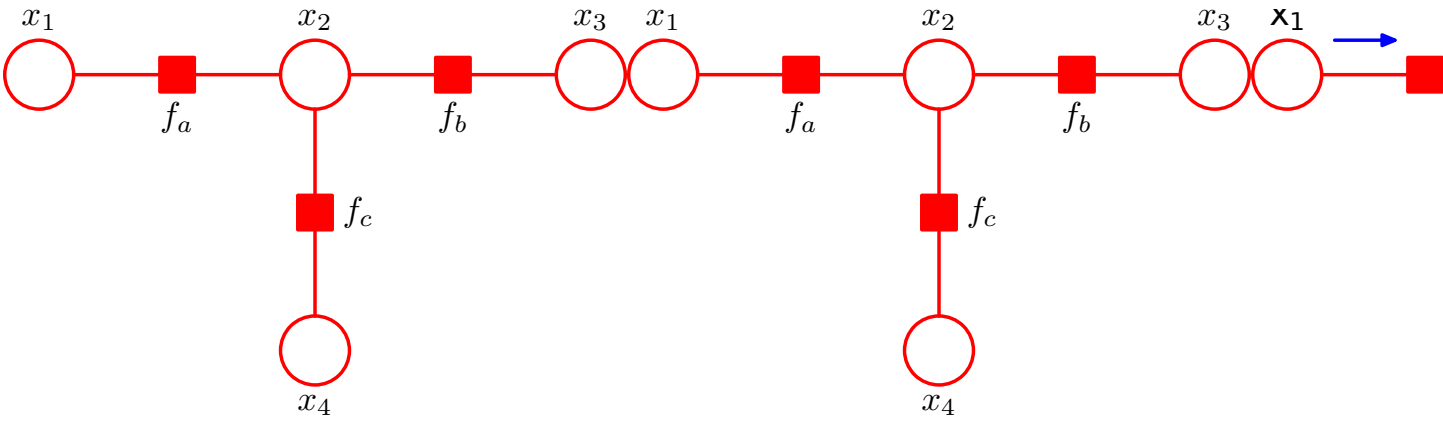
- The node X whose marginal has to be computed is designed as root.
- Messages are sent from all leaves to their neighbours
- Each internal node sends its message towards the root as soon as it received messages from all other neighbours
- Once the root has collected all messages, the marginal can be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

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

Choose X_3 as root

Send initial messages from leaves

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

Send messages from factor nodes to X_2

$$\mu_{f_a \to X_2}(X_2) = \sum_{X_1} f_a(X_1, X_2)$$
$$\mu_{f_c \to X_2}(X_2) = \sum_{X_4} f_c(X_2, X_4)$$

Send message from X_2 to factor node f_b

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

Send message from f_b to X_3

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

Send message from root X_3

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

Send message from f_b to X_2

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

Send messages from X_2 to factor nodes

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

Send messages from factor nodes to leaves

$$\mu_{f_a \to X_1}(X_1) = \sum_{X_2} f_a(X_1, X_2)\mu_{X_2 \to f_a}(X_2)$$
$$\mu_{f_c \to X_4}(X_4) = \sum_{X_2} f_c(X_2, X_4)\mu_{X_2 \to f_c}(X_2)$$

Compute for instance the marginal for X_2

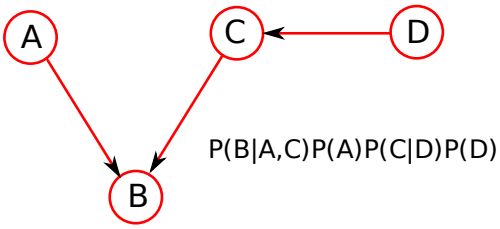
$$\begin{aligned} p(X_2) &= \mu_{f_a \to X_2}(X_2)\mu_{f_b \to X_2}(X_2)\mu_{f_c \to X_2}(X_2) \\ &= \left[\sum_{X_1} f_a(X_1, X_2)\right]\left[\sum_{X_3} f_b(X_2, X_3)\right]\left[\sum_{X_4} f_c(X_2, X_4)\right] \\ &= \sum_{X_1}\sum_{X_3}\sum_{X_4} f_a(X_1, X_2) f_b(X_2, X_3) f_c(X_2, X_4) \\ &= \sum_{X_1}\sum_{X_3}\sum_{X_4}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

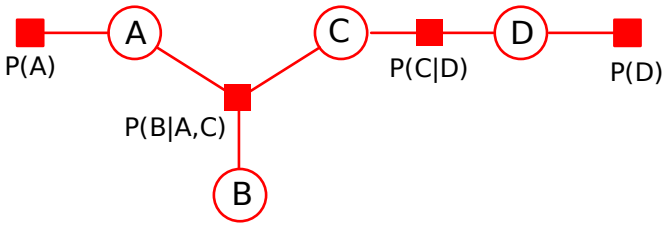
Inference

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

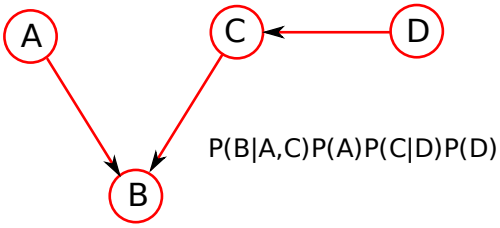


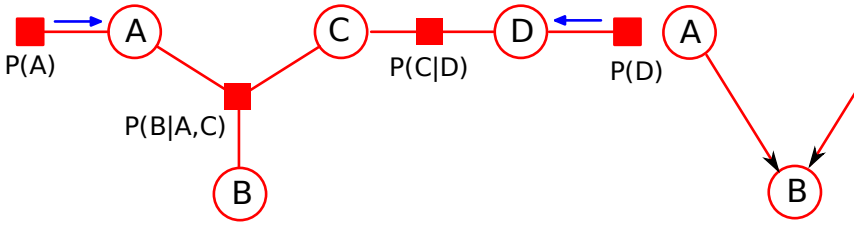


Bayesian network

- 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:

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

- A and D send messages:

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

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

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

- C sends message:

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

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

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

- The desired marginal is obtained:

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

Inference

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

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

Inference

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:

$$\begin{aligned}\mu_{f\to X}(X) &= \max_{X_1,\ldots X_M} \left[f(X, X_1, \ldots, X_M) \prod_{X_m \in ne(f)\setminus X} \mu_{X_m \to f}(X_m) \right] \\ \mu_{X\to f}(X) &= \prod_{f_l \in ne(X)\setminus f} \mu_{f_l \to X}(X)\end{aligned}$$

Inference

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

Inference

Recoving maximal configuration

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

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

- 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:

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

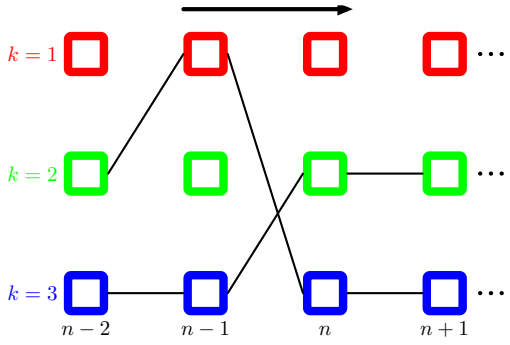
- The procedure can be repeated *back-tracking* to the leaves, retrieving maximal values for all variables

Recoving maximal configuration

Example for linear chain

$$\begin{aligned}
X_N^{\max} &= \underset{X_N}{\operatorname{argmax}}\, \mu_{f_{N-1,N}\to X_N}(X_N) \\
X_{N-1}^{\max} &= \phi_{f_{N-1,N}\to X_N}(X_N^{\max}) \\
X_{N-2}^{\max} &= \phi_{f_{N-2,N-1}\to X_{N-1}}(X_{N-1}^{\max}) \\
&\vdots \\
X_1^{\max} &= \phi_{f_{1,2}\to X_2}(X_2^{\max})
\end{aligned}$$

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.

Inference

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

Inference
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

Approximate inference

Sampling methods

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

Sampling methods

Markov Chain Monte Carlo (MCMC)

- We usually 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

Appendix: Sampling methods

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

Markov chain

Definition

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

$$\mathcal{T}(\mathbf{x} \to \mathbf{x}')$$

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

Markov chain

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

Markov chain

Chain dynamics

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

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

- 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 $\mathbf{x}$

Markov chain

Convergence

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

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

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

Markov chain

Stationary distribution

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

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

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 $Val(\boldsymbol{X})$, *regularity* is a sufficient and necessary condition

Markov chain

Regular chains

A Markov chain is *regular* if there exists a number k such that for all state pairs $\mathbf{x}, \mathbf{x}' \in Val(\boldsymbol{X})$ 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 chain

Markov chains for graphical models

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

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

Markov chain

Transition model

- A simple transition model consists of updating variables in $\hat{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{E}$ one at a time
- This can be seen as made of a set of $k = |\hat{\boldsymbol{X}}|$ *local* transition models $\mathcal{T}_i$, one for each variable $X_i \in \hat{\boldsymbol{X}}$
- Let $\boldsymbol{U}_i = \boldsymbol{X} - X_i$ and $\mathbf{u}_i$ a possible instantiation of $\boldsymbol{U}_i$
- 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

Markov chain

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

Markov chain

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 $\boldsymbol{U}_i$ has non-zero probability
- In this case we can get from any state to any state in at most $k = |\hat{\boldsymbol{X}}|$ local steps
- Gibbs chains have the desired posterior $p(\boldsymbol{X}|\boldsymbol{E} = \mathbf{e})$ 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

Markov chain

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*

Markov chain

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)