Probabilistic Graphical Models

Spring 2024

Instructor: Shandian Zhe
zhe@cs.utah.edu
School of Computing

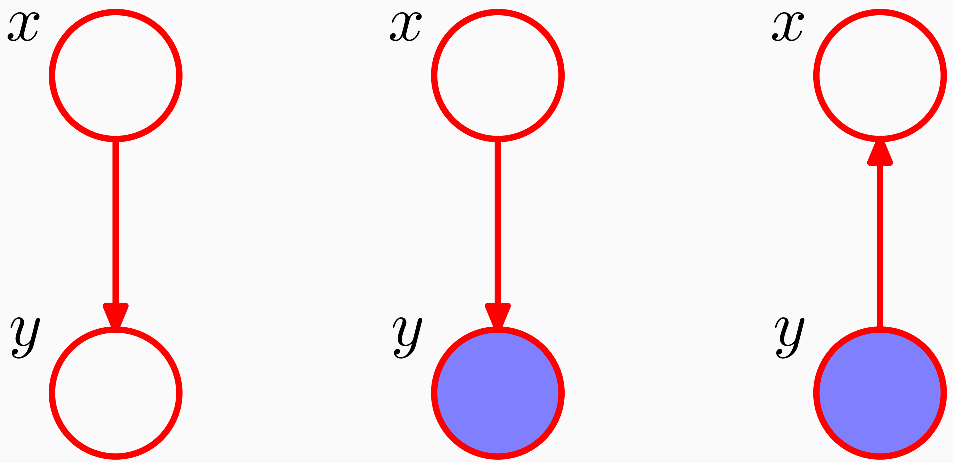
THE UNIVERSITY OF UTAH

Outline

- Bayesian networks
- Markov random fields
- Inference

The task

- Compute the posterior of one or more subsets of nodes given the observed nodes

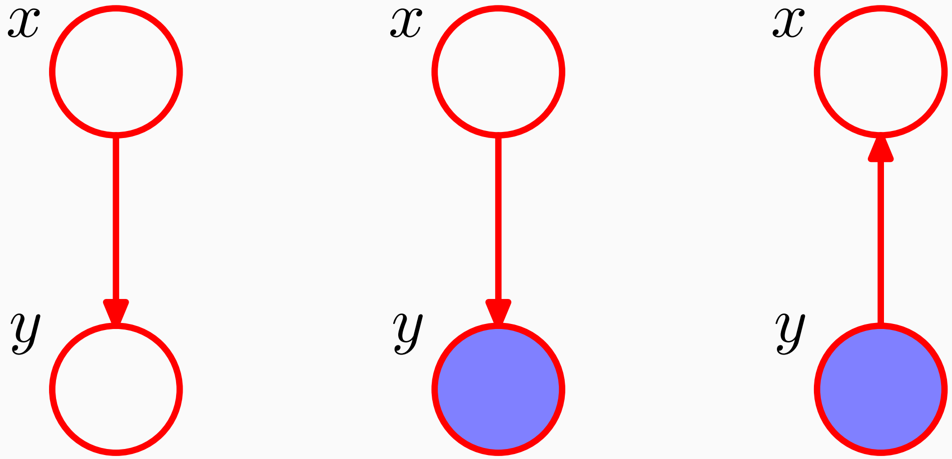


$$p(y) = \sum_{x'} p(y|x')p(x')$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

The task

- Compute the posterior of one or more subsets of nodes given the observed nodes



$$p(y) = \sum_{x'} p(y|x')p(x') \qquad p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Key: compute the marginal of one or a subset of nodes!

Let us start with a chain

Let us start with a chain

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

Let us start with a chain

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

Each node takes *K* states and so each potential is a *K x K* table

Let us start with a chain

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

Each node takes K states and so each potential is a K x K table

Let us consider to infer the marginal of a node x_n

Let us start with a chain

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

Each node takes *K* states and so each potential is a *K x K* table

Let us consider to infer the marginal of a node x_n

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

Let us start with a chain

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

Each node takes *K* states and so each potential is a *K* x *K* table

Let us consider to infer the marginal of a node x_n

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

How much cost?

Let us start with a chain

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

Each node takes *K* states and so each potential is a *K x K* table

Let us consider to infer the marginal of a node x_n

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

How much cost?

$O(K^*K^{N-1})$

Let us start with a chain

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

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

How to reduce the cost?

Key observations: many terms are repeated in the calculation, so we can use the distributive law to save products and sums

$$a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2 = a_1(b_1 + b_2) + a_2(b_1 + b_2) = (a_1 + a_2)(b_1 + b_2)$$

Let us start with a chain

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

$$\underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, \boxed{x_n}) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2)\right]\right] \cdots\right]}_{\mu_\alpha(x_n)}$$

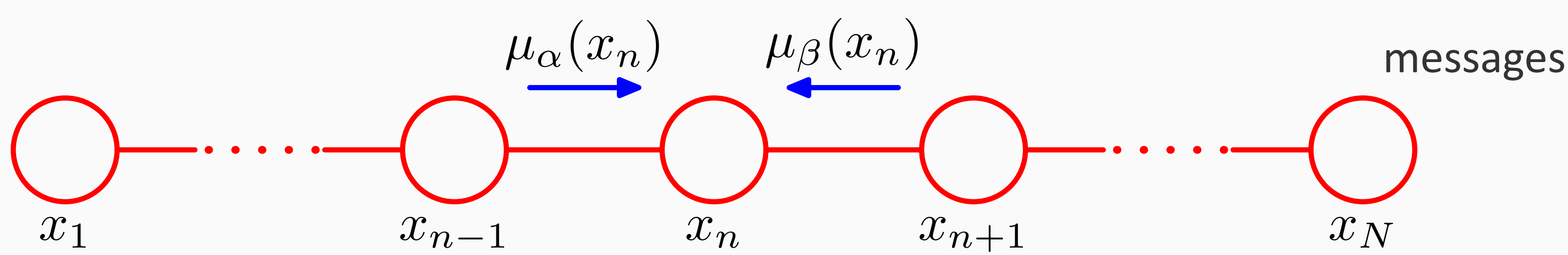
$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(\boxed{x_n}, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots\right]}_{\mu_\beta(x_n)}$$

Let us start with a chain

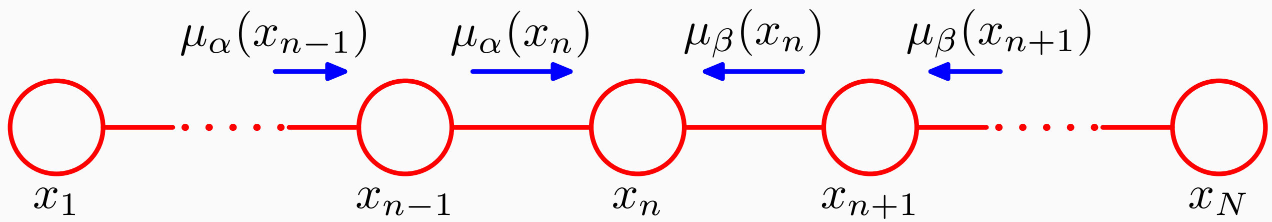
$$p(x_n) = \frac{1}{Z}$$

$$\underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, \boxed{x_n}) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2)\right]\right] \cdots \right]}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(\boxed{x_n}, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots \right]}_{\mu_\beta(x_n)}$$



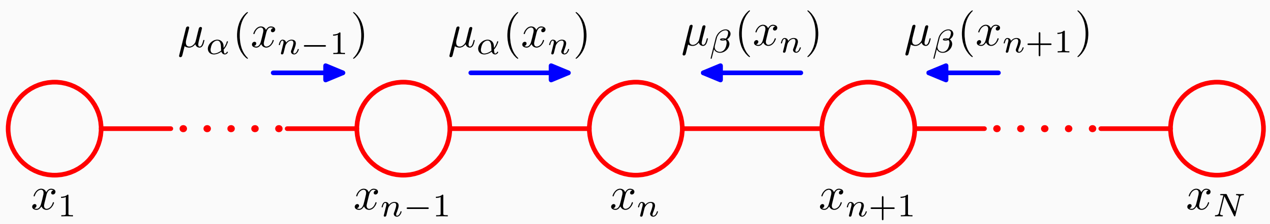
Recursively,



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

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

Initial message



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

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

Question: What is Z? $\quad Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$

Summary: inference on a chain

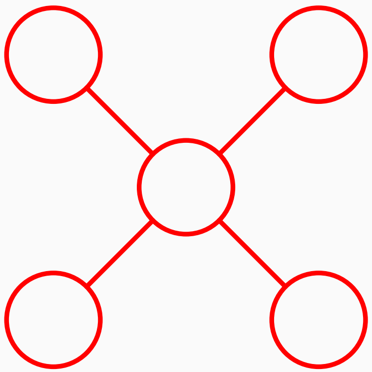
- To compute local marginals:
  - Compute and store all forward messages, $\mu_\alpha(x_n)$
  - Compute and store all backward messages, $\mu_\beta(x_n)$
  - Compute Z at any node x_m
  - Compute
$$p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$$
  for all variables required

What is the cost? *$O(NK^2)$*

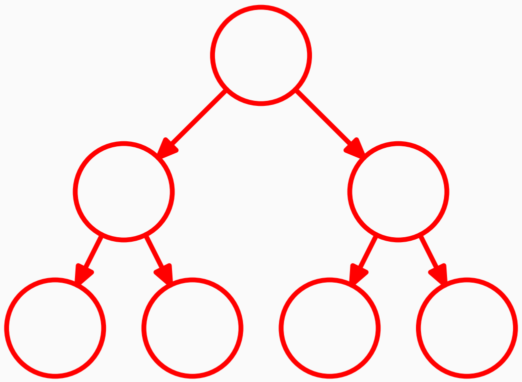
Question: how to infer the marginal of two neighboring variables?

Let us generalize the idea to trees

Tree-structured MRF

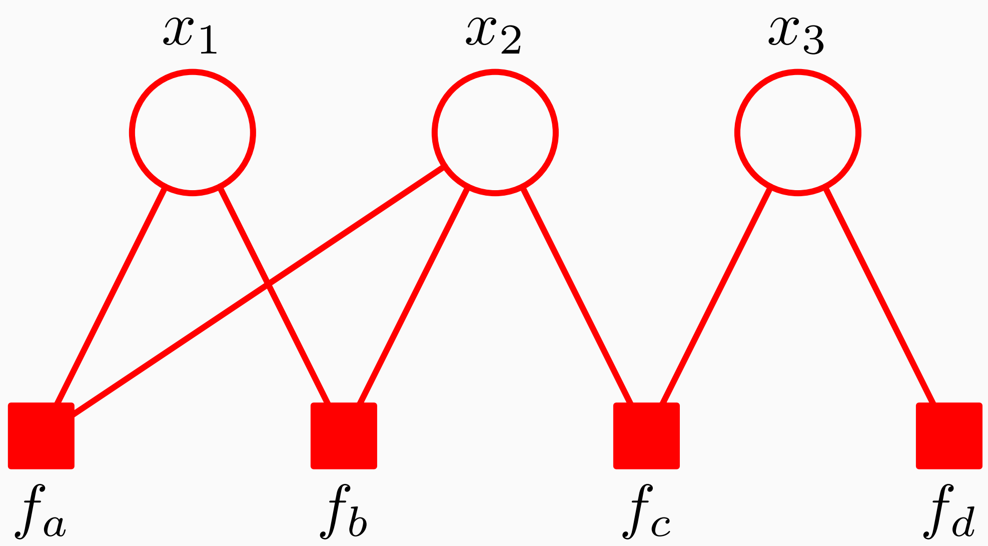


Tree-structured Bayesian network



Why trees: tree structures can guarantee exact inference (we will see it later)

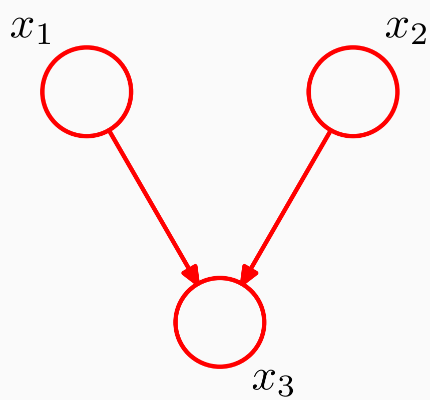
Factor graphs – bipartite graphs

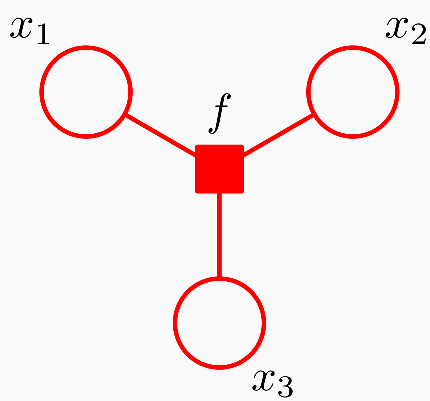


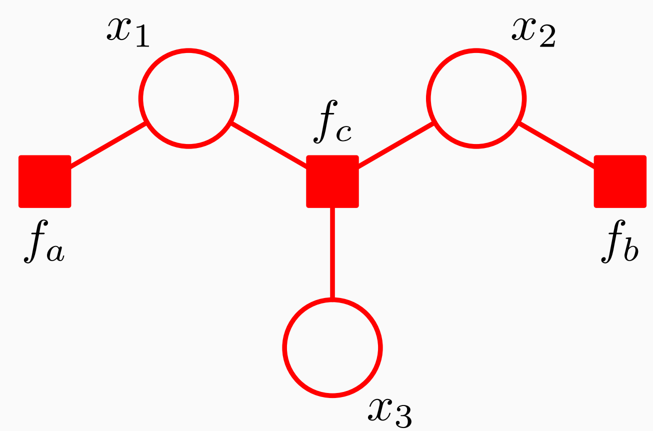
$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

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

Factor graphs – multiple choices







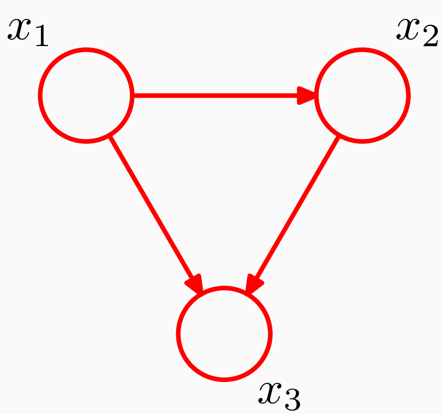
$$\begin{aligned} p(\mathbf{x}) &= p(x_1)p(x_2) \\ &\quad p(x_3|x_1, x_2) \end{aligned}$$

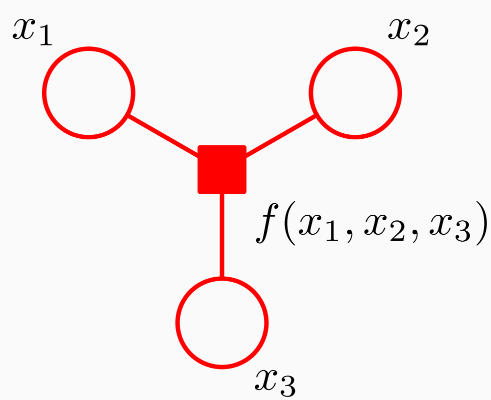
$$\begin{aligned} &f(x_1, x_2, x_3) = \\ &\quad p(x_1)p(x_2)p(_3|x_1, x_2) \end{aligned}$$

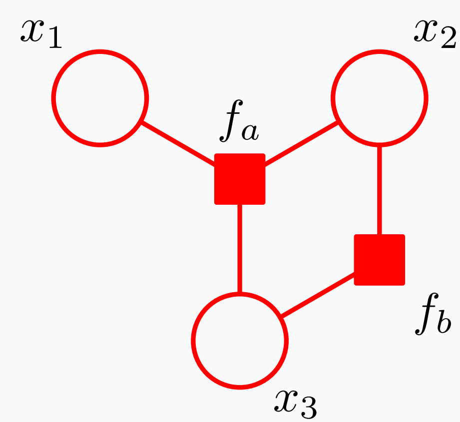
$$\begin{aligned} f_a(x_1) &= p(x_1) \\ f_b(x_2) &= p(x_2) \end{aligned}$$

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

Factor graphs for undirected graphs







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

$$\begin{aligned} f(x_1, x_2, x_3) \\ = \quad \psi(x_1, x_2, x_3) \end{aligned}$$

$$\begin{aligned} f_a(x_1, x_2, x_3) f_b(x_2, x_3) \\ = \quad \psi(x_1, x_2, x_3) \end{aligned}$$

Overview: The Sum-Product algorithm

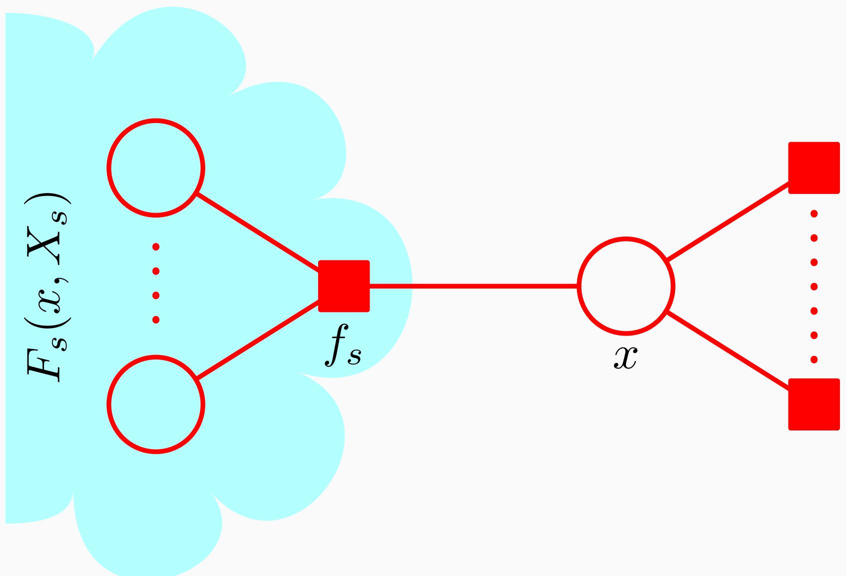
- Objective:
  - efficient, exact inference to find marginals
  - When several marginals are required, allow computations to be shared

Key idea: Distributive Law

$$a_1 b_1 + a_1 b_2 + a_2 b_1 + a_2 b_2 = a_1(b_1 + b_2) + a_2(b_1 + b_2) = (a_1 + a_2)(b_1 + b_2)$$

The Sum-Product algorithm

Given a tree-structured graphical model



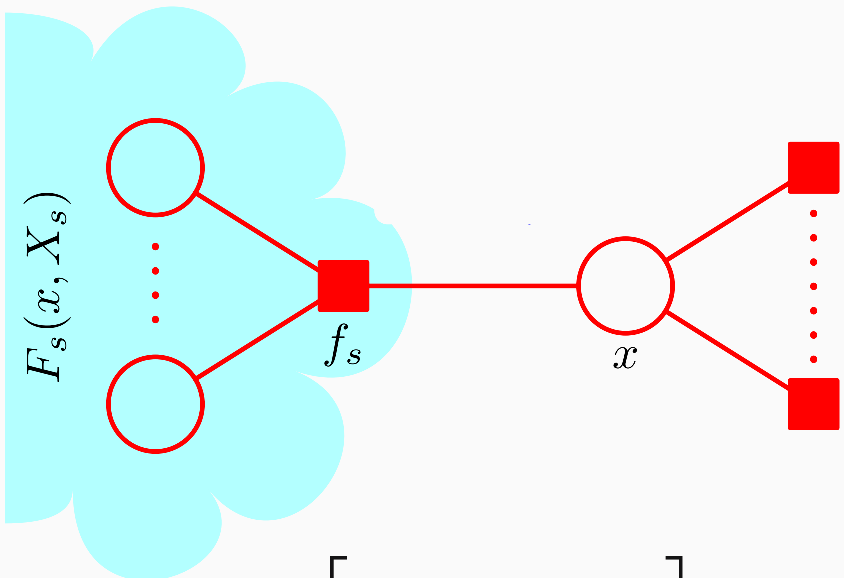
$$p(x) = \sum_{\mathbf{x} \backslash x} p(\mathbf{x})$$

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

ne(x): factor nodes that are neighbors of *x*

X_s: variables in the subtree that connect to *x* via the factor node f_s

The Sum-Product algorithm

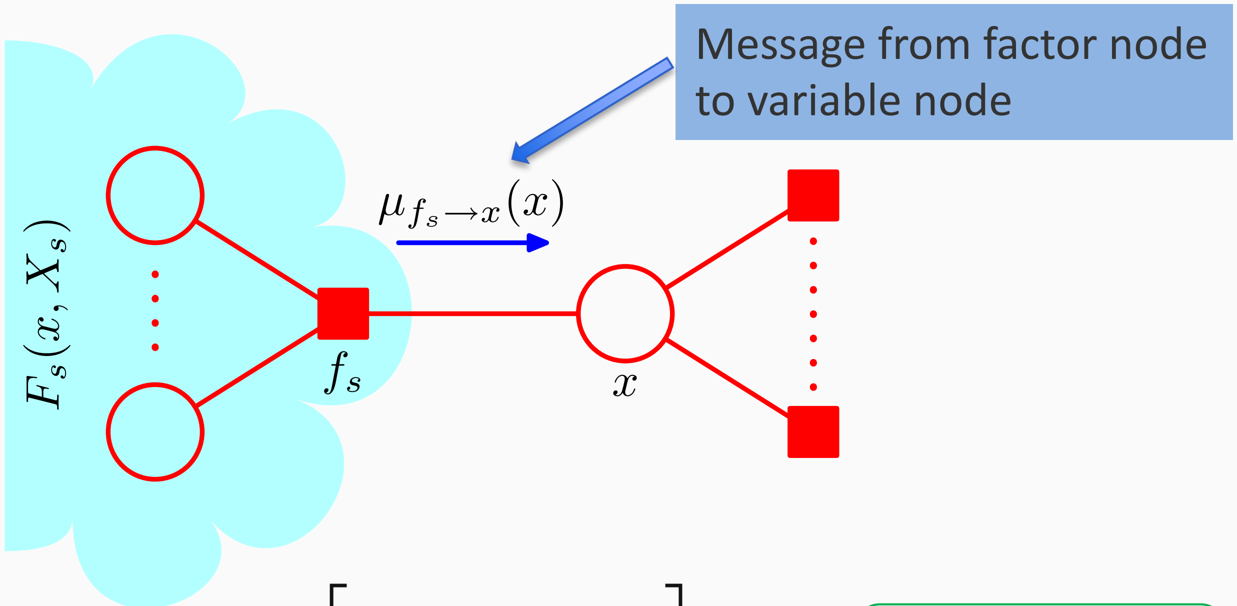


$$p(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right]$$
$$= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x).$$

Why this is true?

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

The Sum-Product algorithm



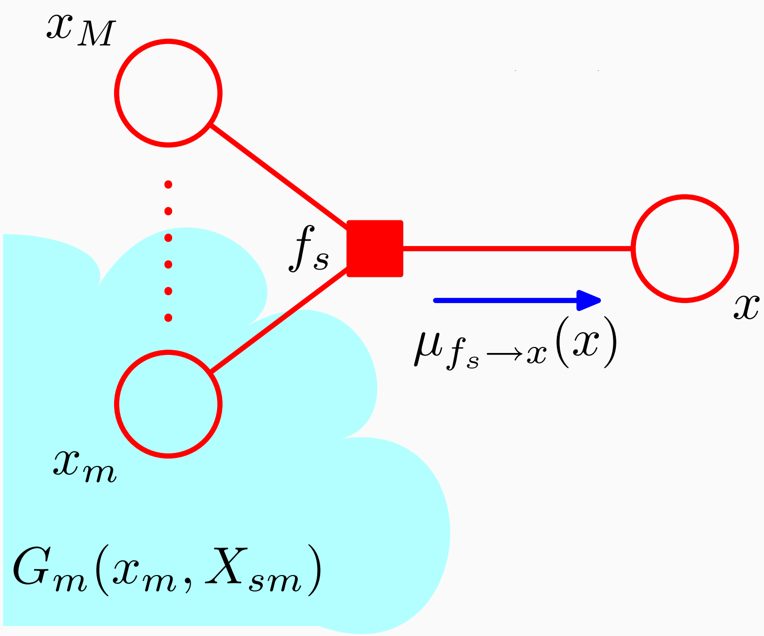
$$p(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right]$$

Why this is true?

$$= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x).$$

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

The Sum-Product algorithm



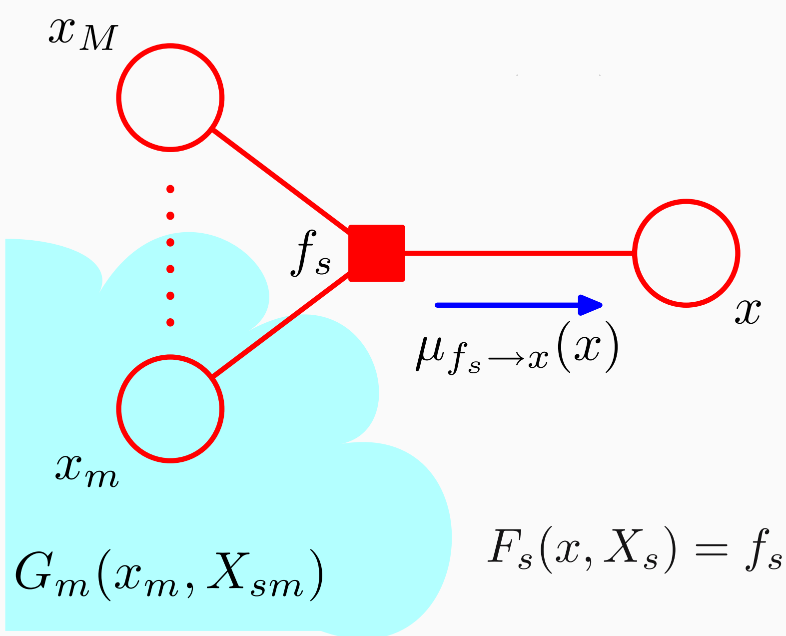
$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \ldots G_M(x_M, X_{sM})$$

X_{sm} : variables in the subtree that connect to x *first* through x_m and *then* through the factor node f_s

Different X_{sm} do not overlap. Why?

The Sum-Product algorithm



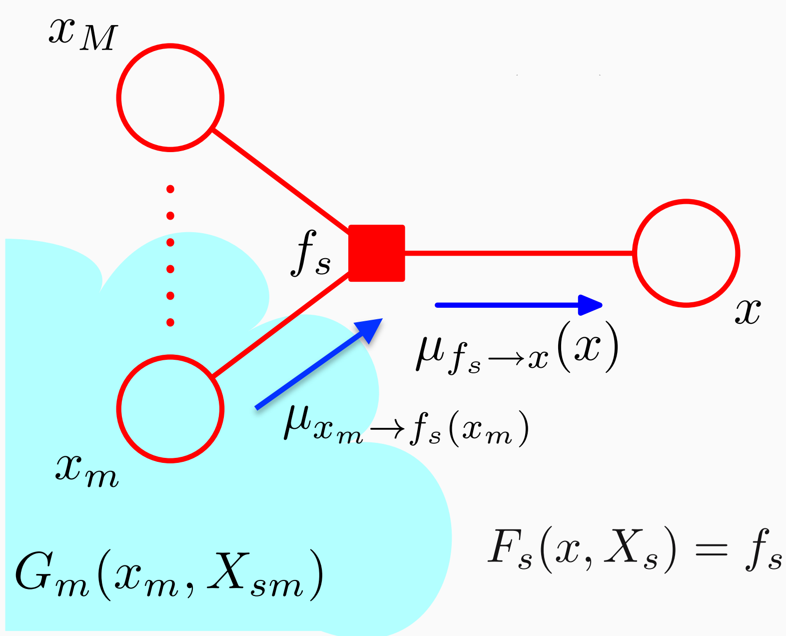
$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \ldots G_M(x_M, X_{sM})$$

distribute the summation

$$\begin{aligned}
\mu_{f_s \to x}(x) &= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{xm}} G_m(x_m, X_{sm}) \right] \\
&= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \boxed{\mu_{x_m \to f_s}(x_m)}
\end{aligned}$$

The Sum-Product algorithm



$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

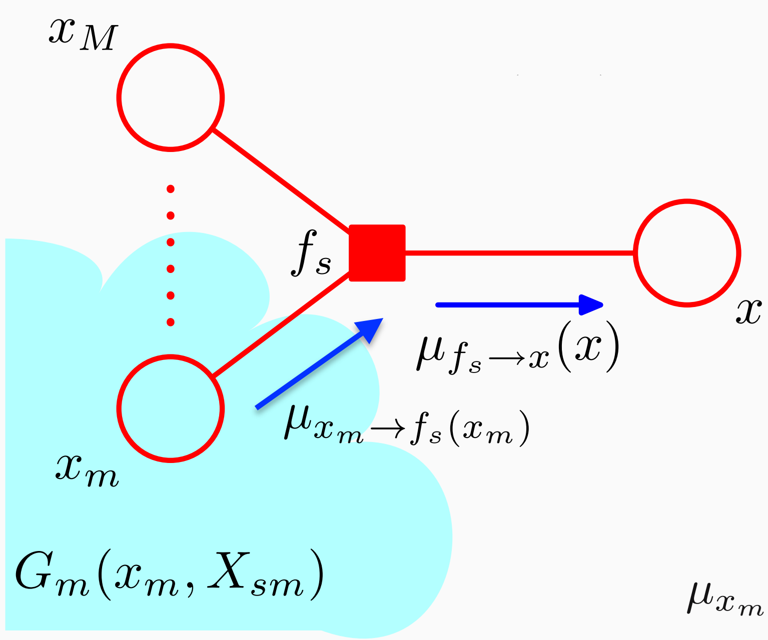
$$F_s(x, X_s) = f_s(x, x_1, \dots, x_M) G_1(x_1, X_{s1}) \dots G_M(x_M, X_{sM})$$

distribute the summation

$$\begin{aligned} \mu_{f_s \to x}(x) &= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{xm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \dots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \boxed{\mu_{x_m \to f_s}(x_m)} \end{aligned}$$

Message from a variable node to a factor node

The Sum-Product algorithm

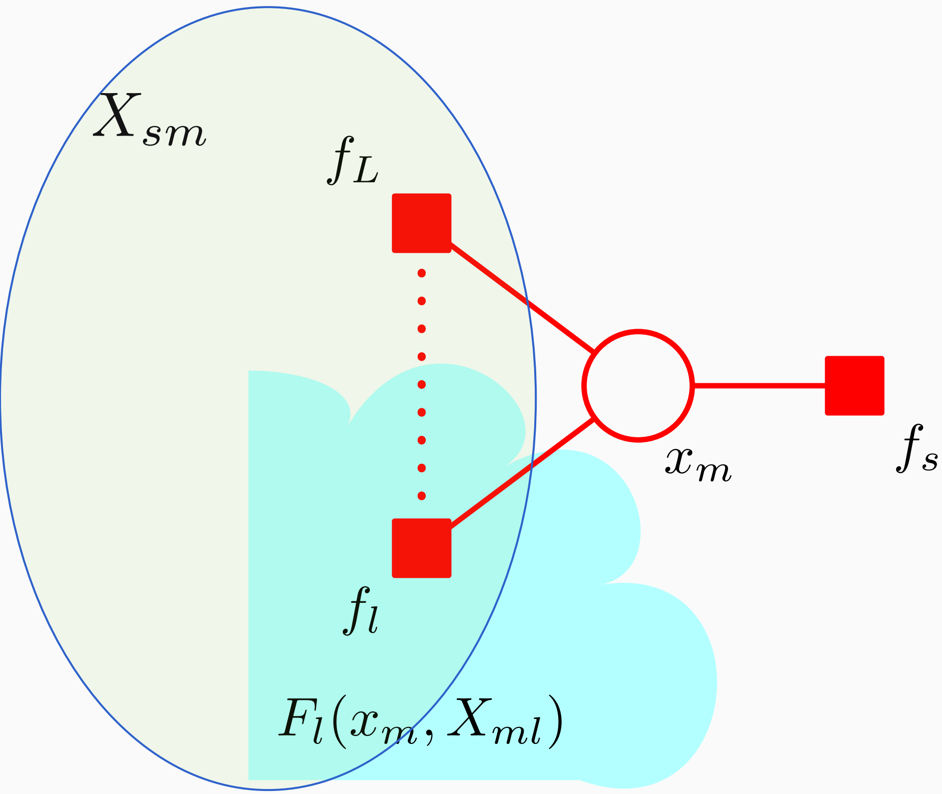


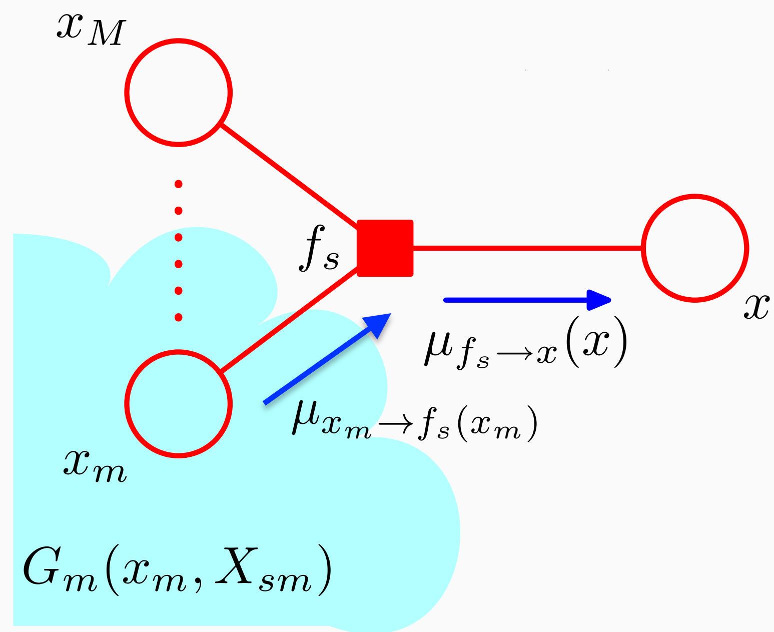
How to compute the message from a variable to a factor?

$$\mu_{x_m \to f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm})$$

Very similar to how we compute p(x), but with a small difference

The Sum-Product algorithm





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

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

The Sum-Product algorithm

- Now we have two message passing rules
  - ❑ From a factor node to a variable node

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

  - ❑ From a variable node to a factor node

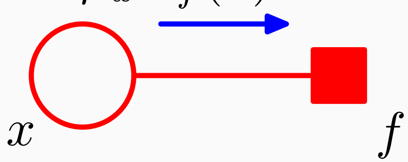
$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

Alternately pass messages!

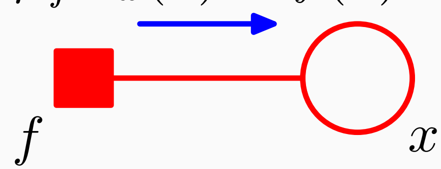
The Sum-Product algorithm

- Initial messages on the leaves

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



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



The Sum-Product algorithm

- How to conduct the order of message passing?

1. Pick an arbitrary node as the root

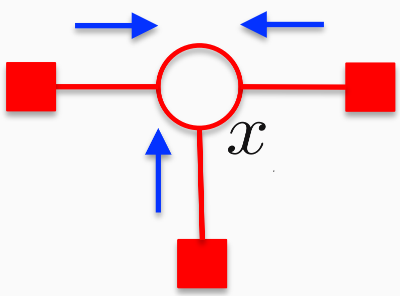
2. Compute and propagate messages *from the leaf nodes to the root*, and store received messages at every node

3. Compute and propagate messages *from the root the leaf nodes*, storing the messages at every node

The Sum-Product algorithm

- After the message passing done, how to compute the marginals?

$$
\begin{aligned}
p(x) &= \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_s(x, X_s) \right] \\
&= \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x).
\end{aligned}
$$



Just multiple the received messages of the variable, and normalize as necessary!

The Sum-Product algorithm

- Why do we need normalization
  - Undirected graphical models (MRF)

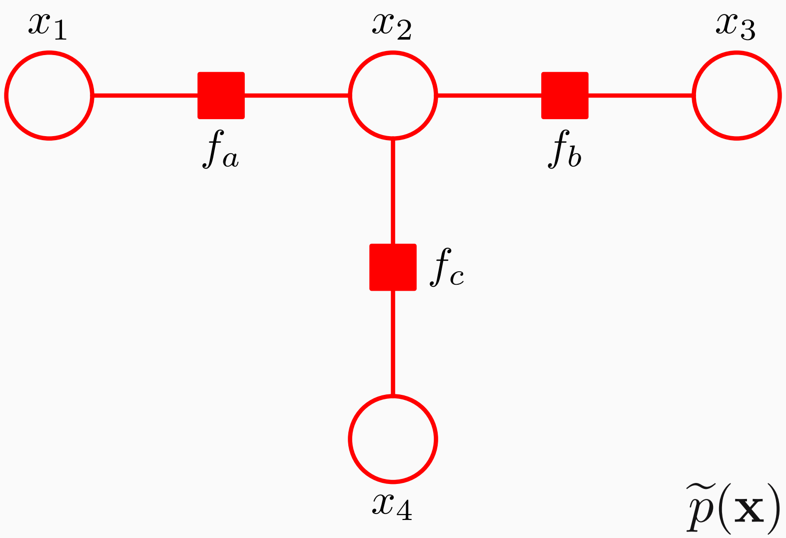
$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C)$$

The potentials are not normalization

  - Some nodes have been observed

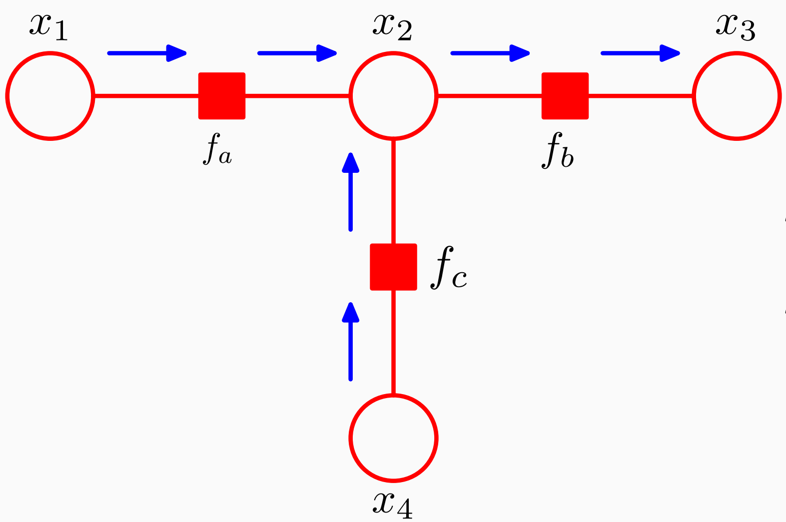
We actually fix the value of the observed nodes in message computation

The Sum-Product algorithm: example



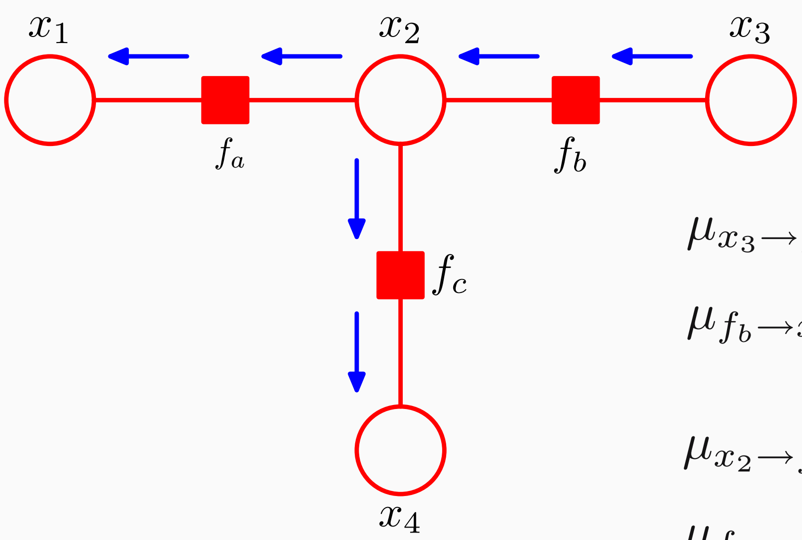
$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

The sum-product algorithm: example



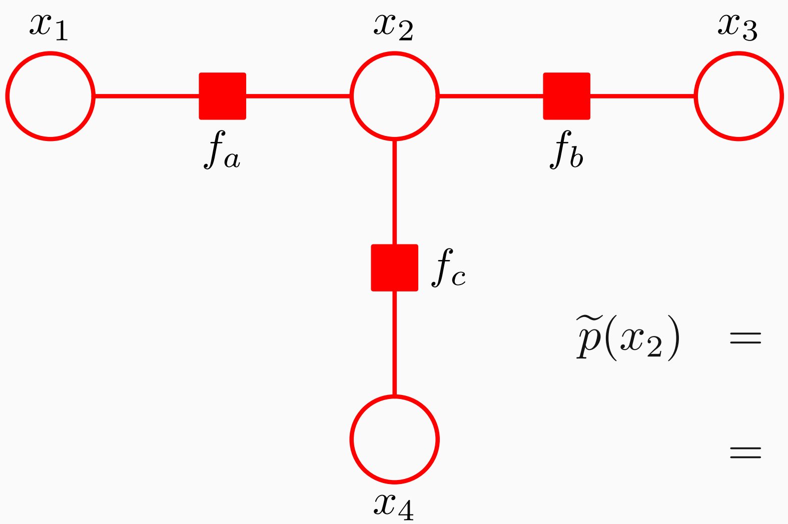
$$
\begin{aligned}
\mu_{x_1 \to f_a}(x_1) &= 1 \\
\mu_{f_a \to x_2}(x_2) &= \sum_{x_1} f_a(x_1, x_2) \\
\mu_{x_4 \to f_c}(x_4) &= 1 \\
\mu_{f_c \to x_2}(x_2) &= \sum_{x_4} f_c(x_2, x_4) \\
\mu_{x_2 \to f_b}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_b \to x_3}(x_3) &= \sum_{x_2} f_b(x_2, x_3)\mu_{x_2 \to f_b}.
\end{aligned}
$$

The sum-product algorithm: example



$$
\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2) \\
\mu_{f_c \to x_4}(x_4) &= \sum_{x_2} f_c(x_2, x_4)\mu_{x_2 \to f_c}(x_2).
\end{aligned}
$$

The sum-product algorithm: example



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

The sum-product algorithm: implementation

- Step 1. Pick a root node x and arrange the graph into a tree

- Step 2.

  For each child factor *f* of *x*

  $\mu_{f \to x}(x)$ = Collect(*f*, *x*)

- Step 3.

  For each child factor *f* of *x*

  Distribute (*x*, *f*)

The sum-product algorithm: implementation

Collect (*x*, *f*)

  if *x* is a leaf, return 1

  for each child factor f_j of *x* *(note: not including f)*

    $\mu_{f_j \to x}(x)$ = Collect(f_j, *x*)

  return $\prod_j \mu_{f_j \to x}(x)$

Collect (*f*, *x*)

  if *f* is a leaf, return *f(x)*

  for each child variable x_j of *f* *(note: not including x)*

    $\mu_{x_j \to f}(x)$ = Collect(x_j, *f*)

  return $\sum_{x_1,\ldots,x_M} f(x, x_1, \ldots, x_M) \prod_j \mu_{x_j \to f}(x)$

The sum-product algorithm: implementation

Distribute (x, f)
    compute and store $\mu_{x \to f}(x)$ directly
    if f is a leaf, return
    for each child variable x_j of f *(note: not including x)*
        Distribute (f, x_j)

Distribute (f, x)
    compute and store $\mu_{f \to x}(x)$ directly
    if x is a leaf, return
    for each child factor f_j of x *(note: not including f)*
        Distribute(x, f_j)

What about general graphs?

- In general graphs that contain cycles, sum-product *cannot guarantee exact inference*
- The exact inference on general graphs is called Junction tree algorithm
  - It first merges factors and turns the initial graph into a junction tree and then run a sum-product-like algorithm
  - Intractable on graphs with large factors

Loopy belief propagation (LBP)

- We can still apply sum-product on general graphs as an *approximate* inference algorithm
- First initialize all the messages with 1 (or random)
- Run sum-product (with any message passing order) repeatedly until convergence (not guaranteed!)
- Often works really well, sometimes totally fail
- Striking connections between LBP and decoding (turbo codes) in information theory

The max-sum algorithm

- A simple variant of the sum-product algorithm
- Objective: an efficient algorithm to find
  - The value $\boldsymbol{x}_{\mathrm{max}}$ that maximizes $p(\boldsymbol{x})$
  - The value of $p(\boldsymbol{x}_{\mathrm{max}})$
- Very important in many tasks, e.g., structure prediction, decision, ….

The max-sum algorithm

- In general, maximum marginals $\neq$ joint maximum

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

$$\arg\max_x p(x, y) = 1 \qquad \arg\max_x p(x) = 0$$

Maximizing over a chain

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

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

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

Observation

- We still have the distributive law

$$\max(ab, ac) = a \max(b, c)$$

So we can simply replace sum by max in the sum-product algorithm!

Observation

- Generalizes to tree-structured factor graph

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_n} \prod_{f_s \in \mathrm{ne}(x_n)} \max_{X_s} f_s(x_n, X_s)$$

Observation

- To enhance numerical stability, we take log

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

The distributive law still holds

$$\max(a+b, a+c) = a + \max(b, c)$$

So we only need to replace sum by max, product by sum in the sum-product algorithm

The max-sum algorithm

Initialization message (leaf nodes)

$$\mu_{x \to f}(x) = 0 \qquad \mu_{f \to x}(x) = \ln f(x)$$

Message passing (recursively)

$$\mu_{f \to x}(x) = \max_{x_1, \ldots, x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

$$\mu_{x \to f}(x) = \sum_{l \in \mathrm{ne}(x) \setminus f} \mu_{f_l \to x}(x)$$

The max-sum algorithm

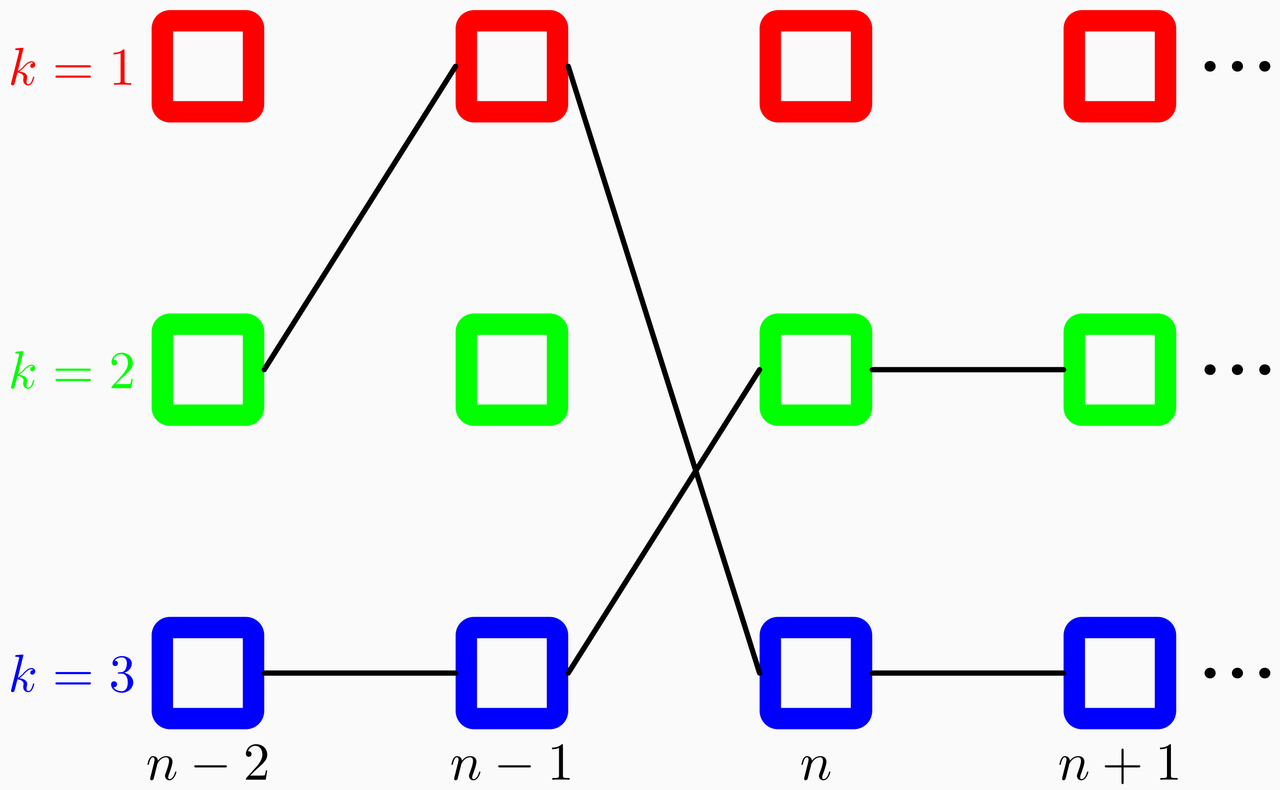
- First pass from leaves to the root and the second pass from the root to leaves
- Termination

$$
\begin{aligned}
p^{\max} &= \max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right] \\
x^{\max} &= \arg\max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]
\end{aligned}
$$

The max-sum algorithm

- How to find the global configuration $\boldsymbol{x}_{\max}$ that gives the maximum probability?
- We need to store a quantity to tell us how to trace back to the variable value that maximizes the previous sub-problem (back-tracking)
- So each message can contain two component: (1) the max-sum value (2) the variable value that gives the max-sum (i.e., argmax)

The max-sum algorithm



The max-sum algorithm

- This is essentially dynamic programming
- For hidden Markov models, this is known as Viterbi algorithm

What you need to know

- Factor graph definition
- Sum-product algorithm
- Message-passing
- Accurate for tree-structured graphs, not guaranteed to be accurate for graphs with cycles
- Loopy belief propagation
- Max-product algorithm, max-sum
- Be able to implement the algorithms!