

Lecture 11

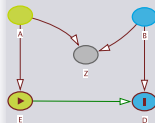
Approximate Inference

Course **Bayesian Networks**, December 17th, 2018

Johannes Textor
Institute for Computing and Information Sciences

**Approximate
Inference**

Johannes Textor



Factor Graphs

Message Passing on
Factor Graphs

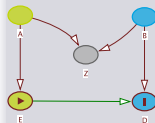
Loopy Message
Passing

Learning Objectives 11

- 1 Know what factor graphs are and how to derive them from Bayesian networks.
- 2 Be able to apply the message passing algorithm to factor graphs.

Approximate
Inference

Johannes Textor



Factor Graphs

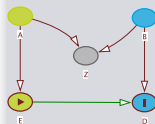
Message Passing on
Factor Graphs

Loopy Message
Passing

1 Factor Graphs

2 Message Passing on Factor Graphs

3 Loopy Message Passing



Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing

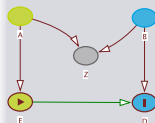
Putting the Fun Into Elimination Orderings

Since the previous lecture, we know about elimination orderings and what they are. But let's be honest – they are no fun.

Today, we will learn about a fun, generic algorithm to find elimination orderings. And this same algorithm can be used to solve a whole host of other problems, too!

Approximate
Inference

Johannes Textor



Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing

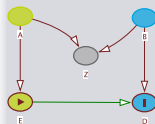
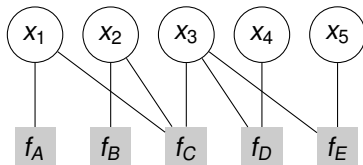
Factor Graphs

Suppose we have a function $g(x_1, \dots, x_n)$ that factorizes as

$$g(x_1, \dots, x_n) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5)$$

The **factor graph** of g contains:

- 1 node for each variable.
- 1 node for each factor.
- Edges between each factor and the variables it contains.



Factor Graphs

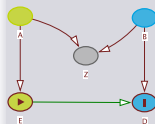
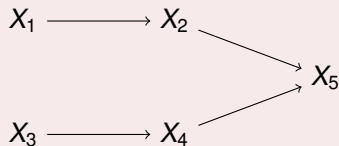
Message Passing on
Factor Graphs

Loopy Message
Passing

5-Minute Exercise

Exercise

Draw a factor graph for a probability density that factorizes according to the following Bayesian network:



Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing

Factor Graphs and Elimination Orderings

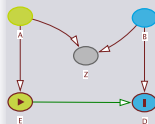
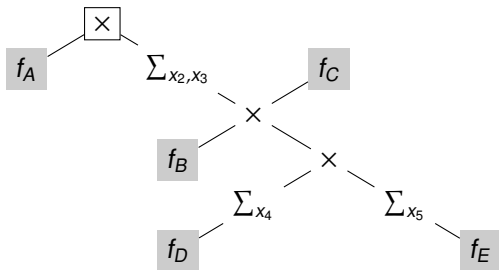
Consider again our function

$$g(x_1, \dots, x_n) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) .$$

For x_1 , this has an elimination ordering

$$g(x_1) = f_A(x_1) \left(\sum_{x_2, x_3} f_B(x_2) f_C(x_1, x_2, x_3) \left(\sum_{x_4} f_D(x_3, x_4) \right) \left(\sum_{x_5} f_E(x_3, x_5) \right) \right)$$

Like any nested arithmetic expression, this can be drawn as a (rooted) **expression tree**:



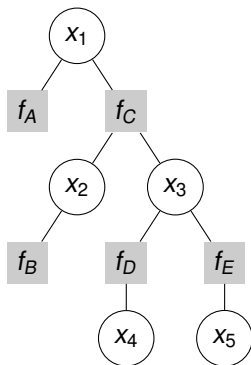
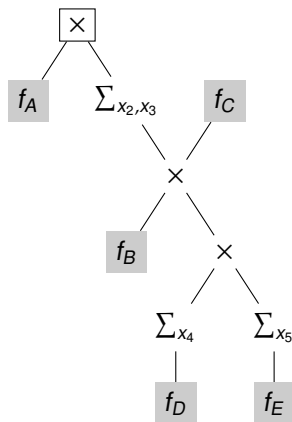
Factor Graphs

Message Passing on
Factor Graphs

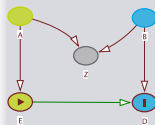
Loopy Message
Passing

Factor Graphs and Expression Trees

If we compare the expression tree of the elimination ordering for $g(x_1)$ and the factor graph rooted at x_1 , we will note topological similarity:



In fact, we can turn every rooted (tree) factor graph into an expression tree!



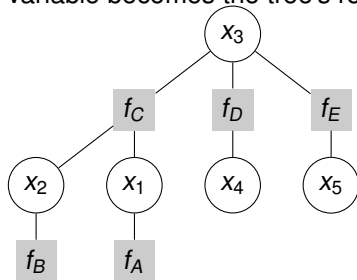
Factor Graphs

Message Passing on
Factor Graphs

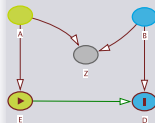
Loopy Message
Passing

Elimination Orderings from Factor Graphs

To begin, we first need to orient the tree such that the desired variable becomes the tree's root node.

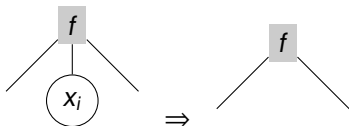


Let's now discuss some **replacement rules** that we can use to turn this tree into an expression tree!

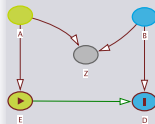
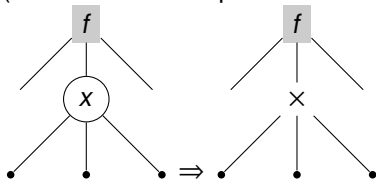


Replacement Rules for Variable Nodes

If x is a leaf variable node with parent f , simply remove it.



Otherwise, substitute x by the product of all its child nodes (which need to be processed before!)



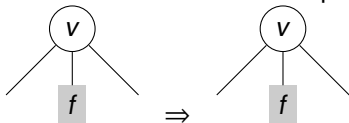
Factor Graphs

Message Passing on
Factor Graphs

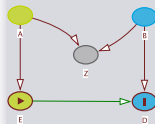
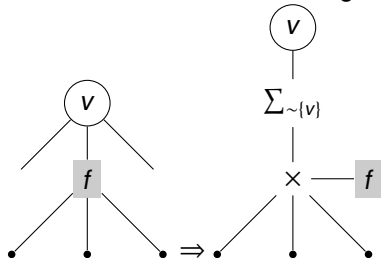
Loopy Message
Passing

Replacement Rules for Factor Nodes

If f is a leaf factor node with parent v , simply keep it as is.



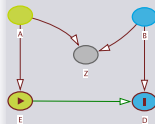
Otherwise, substitute it by a node that first multiplies all children with f , and then marginalizes all variables **except** v .



Factor Graphs

Message Passing on
Factor Graphs

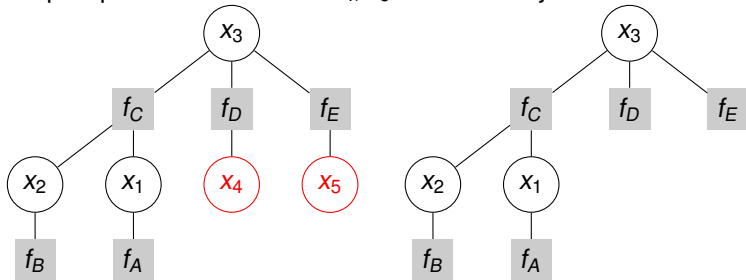
Loopy Message
Passing



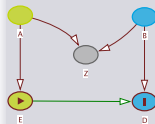
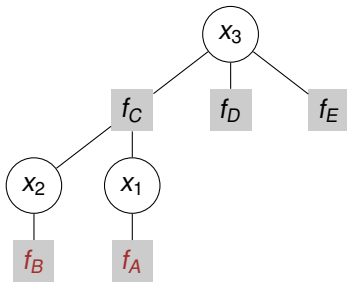
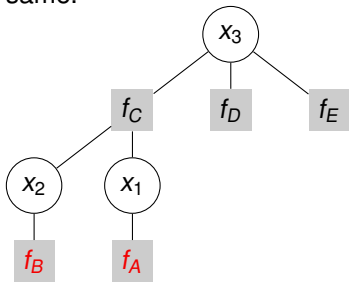
Factor Graphs

Message Passing on
Factor GraphsLoopy Message
Passing

Step 1: process leaf nodes x_4, x_5 . These can just be removed.



Step 2: process leaf factor nodes f_B, f_A . These can just stay the same.

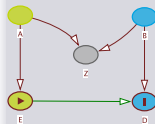
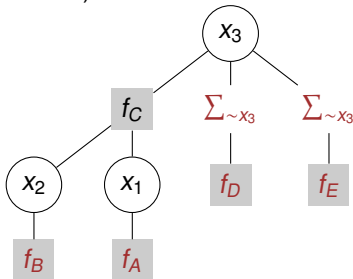
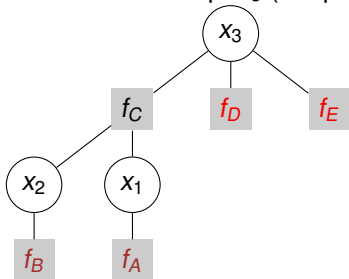


Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing

Step 3: process **interior** (non-leaf) factor nodes f_D, f_E . These have no children, and can be replaced by a sum operator over all variables except x_3 (the parent node).

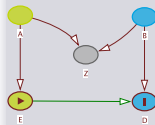
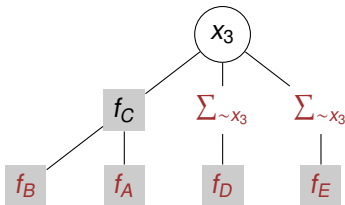
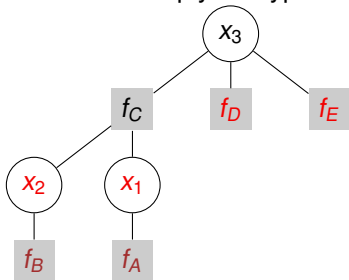


Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing

Step 4: process **interior** variable nodes x_1, x_2 . They should be replaced with a product of all their children. But both have only one child so the product is simply the child. Therefore, the nodes can simply be bypassed.

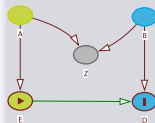
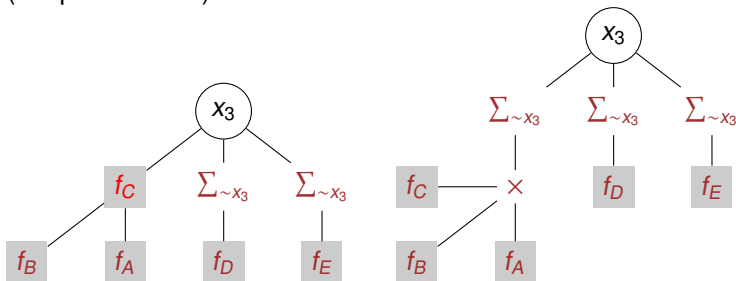


Factor Graphs

Message Passing on Factor Graphs

Loopy Message Passing

Step 5: process factor node f_C . This is replaced by a product of all its children and f_C , summarized over all variables except x_3 (the parent node).

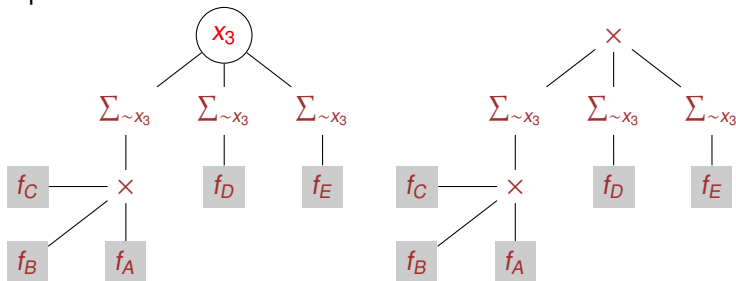


Factor Graphs

Message Passing on
Factor Graphs

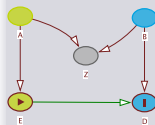
Loopy Message
Passing

Step 6: Process the root variable node x_3 . This is replaced by a product of all its children.



The resulting tree describes an elimination ordering for $g(x_3)$!

$$g(x_3) = \left(\sum_{x_1, x_2} f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) \right) \left(\sum_{x_4} f_D(x_3, x_4) \right) \left(\sum_{x_5} f_E(x_3, x_5) \right)$$

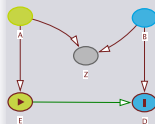


Message Passing

In practice, we will not actually compute the expression trees and the elimination orderings. Instead, we will directly perform computations on the factor graph.

To do this, we imagine that each node in the factor graph is a **processor** with three capabilities:

- Send messages (functions) along edges.
- Receive messages (functions) along edges.
- Process messages by multiplication and addition.



Revisiting the Markov Chain

Consider again our 4-variable Markov chain from the last lecture:

$$X_1 \longrightarrow X_2 \longrightarrow X_3 \longrightarrow X_4$$

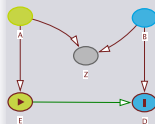
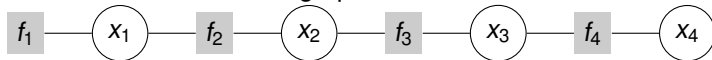
X_1	P
1	0.5

X_1	X_2	P
0	1	0.6
1	1	0.6

X_2	X_3	P
0	1	0.2
1	1	0.4

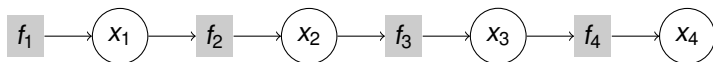
X_3	X_4	P
0	1	0.8
1	1	0.5

The factor graph is also a chain:



Message Passing on the Markov Chain

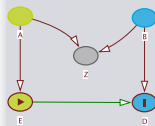
To compute $P(x_4)$, we interpret the factor graph as a tree, and orient its edges towards the root x_4 .



- Messages will be passed along the edge directions.
- Each message is a function of one variable.
- Each edge is associated with one variable.

The Sum-Product Algorithm

The **local function** at a variable node is the unit function **1**. The local function at a factor node is that factor. The **message** on an edge $e = v \rightarrow$ is the product of the local function at v with all messages received on edges $\rightarrow v$, summed over all variables except the one associated with e .



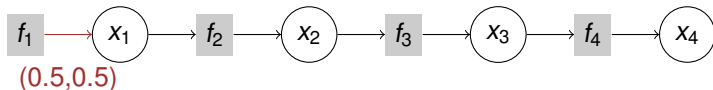
Factor Graphs

Message Passing on
Factor Graphs

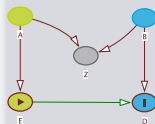
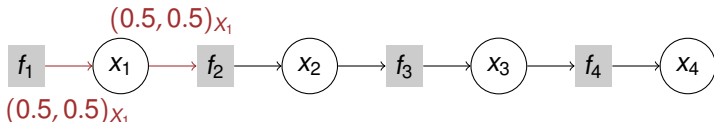
Loopy Message
Passing

Message Passing on the Markov Chain

We begin with the leaf node f_1 . This node simply passes on its local function $f_1(x_1) = P(x_1)$. We write this function as the vector $(P(X_1 = 0), P(X_1 = 1)) = (0.5, 0.5)$.



The variable node x_1 multiplies all incoming messages – in this case, there is only one message, which is therefore simply passed along.



Message Passing on the Markov Chain

The next factor node f_2 , multiplies the incoming message with $f_2(x_1, x_2) = P(x_2 | x_1)$, summarizes the result for x_2 , and passes it on. Let's look at this step in detail.

Incoming message:

$$(P(X_1 = 0), P(X_1 = 1)) = (0.5, 0.5)$$

Local function:

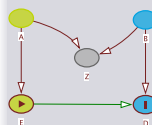
$$\begin{pmatrix} P(X_2 = 0 | X_1 = 0) & P(X_2 = 0 | X_1 = 1) \\ P(X_2 = 1 | X_1 = 0) & P(X_2 = 1 | X_1 = 1) \end{pmatrix} = \begin{pmatrix} 0.4 & 0.4 \\ 0.6 & 0.6 \end{pmatrix}$$

Product of local function and incoming message:

$$\begin{pmatrix} 0.5 \cdot 0.4 & 0.5 \cdot 0.4 \\ 0.5 \cdot 0.6 & 0.5 \cdot 0.6 \end{pmatrix}$$

Summing over X_1 :

$$(P(X_2 = 0), P(X_2 = 1)) = (0.4, 0.6)$$



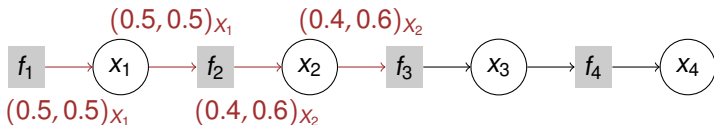
Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing

Message Passing on the Markov Chain

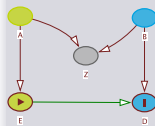
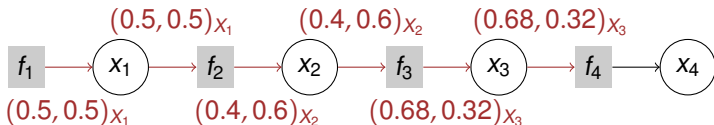
The next variable node, x_2 passes on the message.



At f_3 , we do

$$\begin{pmatrix} 0.8 & 0.6 \\ 0.2 & 0.4 \end{pmatrix} \times \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} = (0.68, 0.32)$$

i.e., we multiply with $f_3(x_2, x_3) = P(x_3 | x_2)$ and pass on a summary for x_3 . At x_3 , we pass the message through.

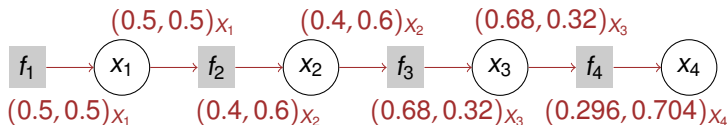


Message Passing on the Markov Chain

Our last step, at f_4 , is

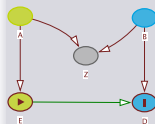
$$\begin{pmatrix} 0.2 & 0.5 \\ 0.8 & 0.5 \end{pmatrix} \times \begin{pmatrix} 0.68 \\ 0.32 \end{pmatrix} = (0.296, 0.704)$$

We arrive at our final outcome: the marginal distribution $P(x_4)$.



Practical Consideration

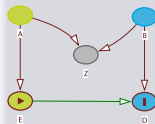
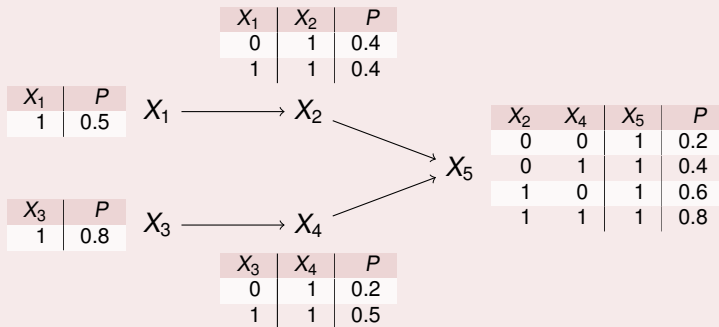
For Bayesian networks, the message transmitted on an edge e will be a marginal distribution of the variable associated with e . This means it will sum up to 1. To avoid numerical issues, we can also pass on an arbitrarily scaled version of that function, such as $(296, 704)$ instead of $(0.296, 0.704)$.



5-Minute Exercise

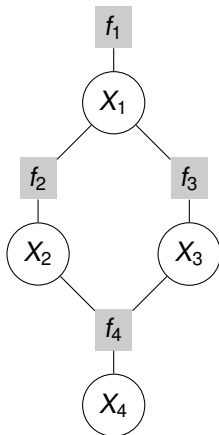
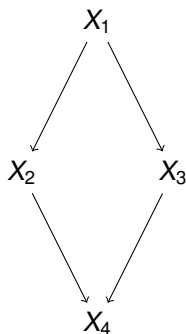
Exercise

Use the message-passing algorithm to compute the marginal probability $P(x_5)$ on the following network:

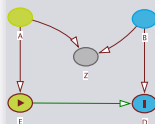


Loopy Factor Graphs

For Bayesian networks that are not polytrees, the factor graph will contain cycles.



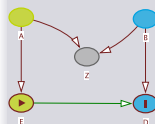
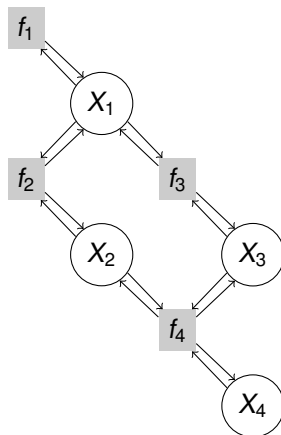
This means that we cannot derive an elimination ordering from the factor graph, since it does not describe an expression tree.



Loopy Message Passing

Loopy message passing is an extension of the message passing algorithm that also works for graphs with loops. It is an **approximation** algorithm, which isn't even guaranteed to converge in most cases, but appears to “work well in practice”.

With loopy message passing, messages are passed in both directions along every edge.



Loopy Message Passing

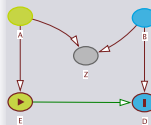
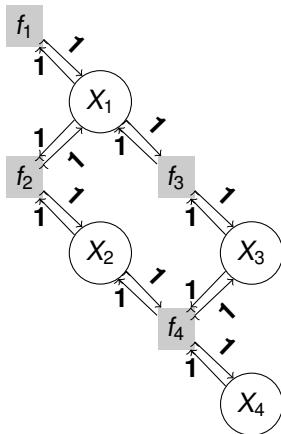
We **initialize** each message to the unit function **1**.

In each step, we compute the message on each edge according to the normal rules of the sum-product algorithm.

At some point, we stop the computation.

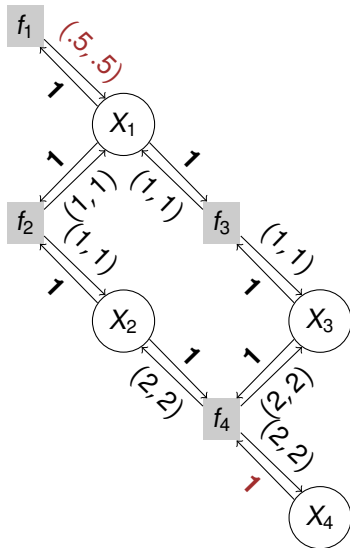
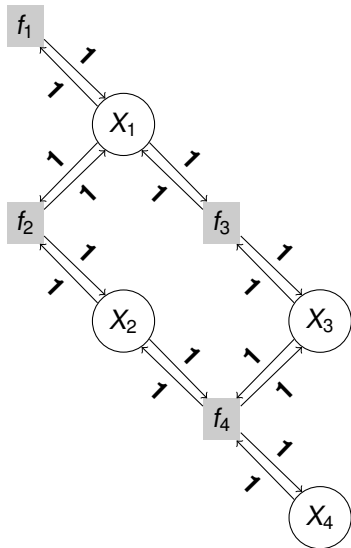
For trees, this procedure converges to the correct solution after a fixed number of steps.

We now give a simple example, in which we assume that all probabilities are 0.5.



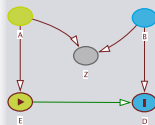
Loopy Message Passing

Let us run the algorithm, assuming that all probabilities are 0.5.



Approximate Inference

Johannes Textor



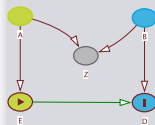
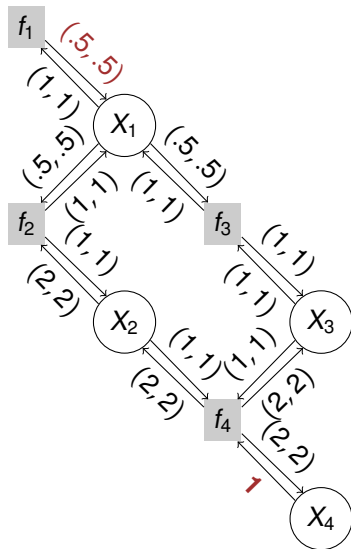
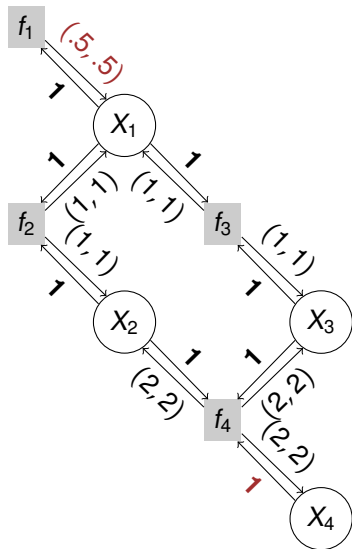
Factor Graphs

Message Passing on Factor Graphs

Loopy Message Passing

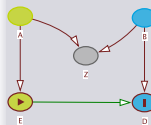
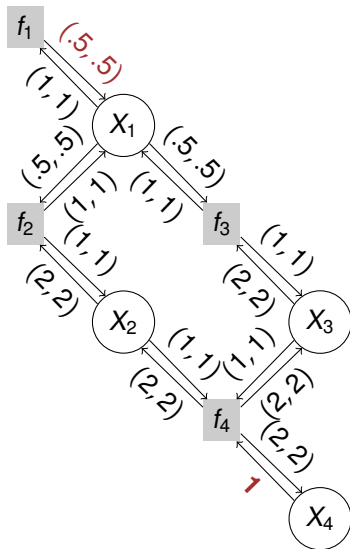
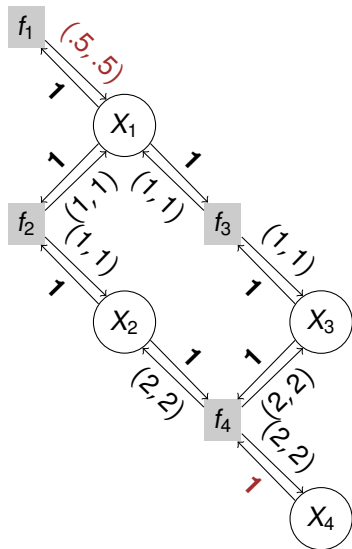
Loopy Message Passing

Let us run the algorithm, assuming that all probabilities are 0.5.



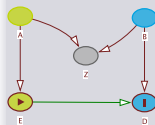
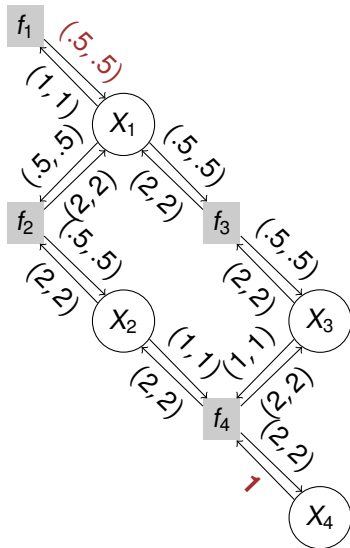
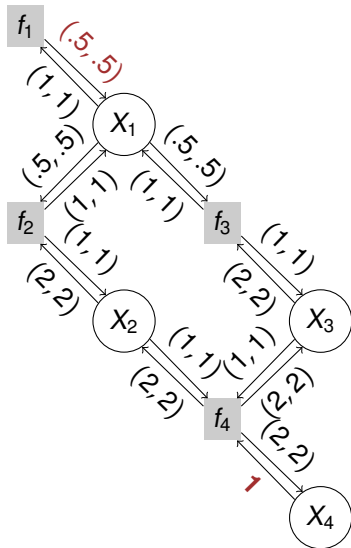
Example: Loopy Message Passing

Let us run the algorithm, assuming that all probabilities are 0.5.



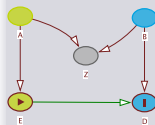
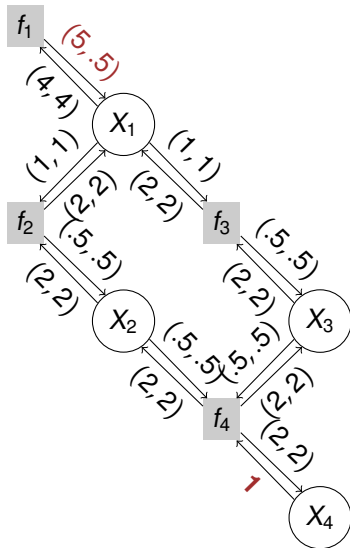
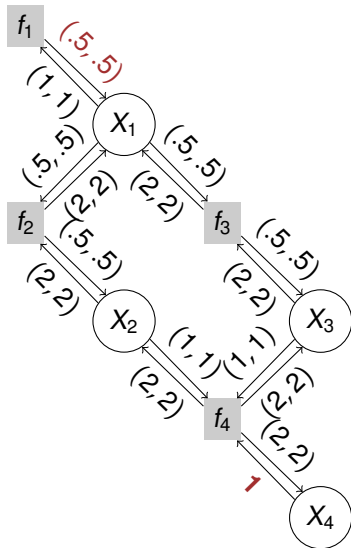
Example: Loopy Message Passing

Let us run the algorithm, assuming that all probabilities are 0.5.




Example: Loopy Message Passing

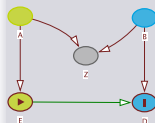
Let us run the algorithm, assuming that all probabilities are 0.5.



Summary

- 1 Factor graphs are a useful representation of Bayesian networks (and many other graphical models).
- 2 For polytree networks, the sum-product algorithm on factor graphs can be used for exact inference.
- 3 The sum-product algorithm is also often useful for loopy networks, in which case it is an iterative approximation algorithm.

 F. R. Kschischang, B. J. Frey, H. -A. Loeliger:
Factor graphs and the sum-product algorithm.
IEEE Transactions on Information Theory 47(2):498–519,
2001.



Factor Graphs

Message Passing on
Factor Graphs

Loopy Message
Passing