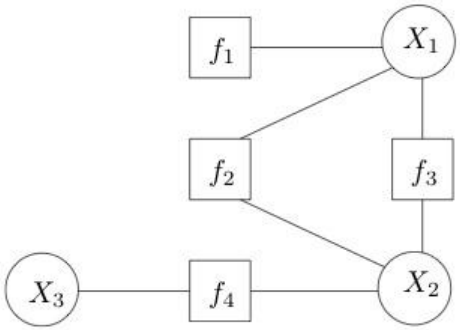
Factor Graphs and Belief Propagation – an Introduction

Contents

- Some terminologies
- Distributive Law (repetition)
- Introduction to Factor Graphs
- Applications of Factor Graphs
- Summary-Product Rule
- Short introduction to Channel Models
- Some examples and Matlab-Demos

Factor Graphs?

- Graphical Model
- Often associated with particular algorithms
  - I. e. Viterbi decoding „on a trellis“
- Support the trend from sequential to iterative processing (Turbo codes)
- A nice way to describe message passing algorithms



Belief Propagation?

- Also called „summary-product algorithm“ or „probability propagation algorithm“
- Operates by passing summaries („messages“) along the edges of a graphical model (→ Factor Graphs!)
- Application in coding, signal processing, artificial intelligence, etc.
- Error correcting codes: invented by Gallager to decode LDPC codes (1963)

Some basics…

- In decoding problems (and much others) calculating marginals is a big issue („eliminating variables“)
- First, we need a quick reminder of the *distributive law*…

$$ab + ac = a(b + c)$$

$$f(a, b, c) = f(a)f(b, c)$$

- Factorization!

And why is this useful?

$$\sum_{i,j} a_i b_j$$

- Distributive law not applied
- Computational complexity:
- i*j Multiplications
- ~i*j Additions

$$\sum_{i} a_i \sum_{j} b_j$$

- With distributive law applied
- Computational complexity:
- 1 Multiplication
- ~i*j Additions

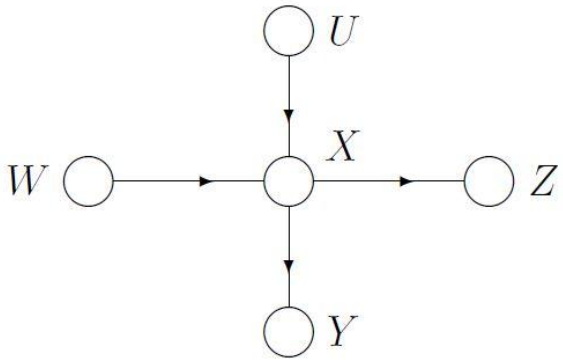
This rule, properly applied, reduces complexity significantly!

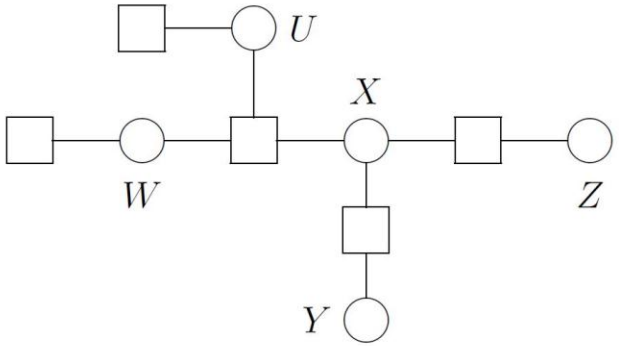
Factor Graphs & Distributive Law

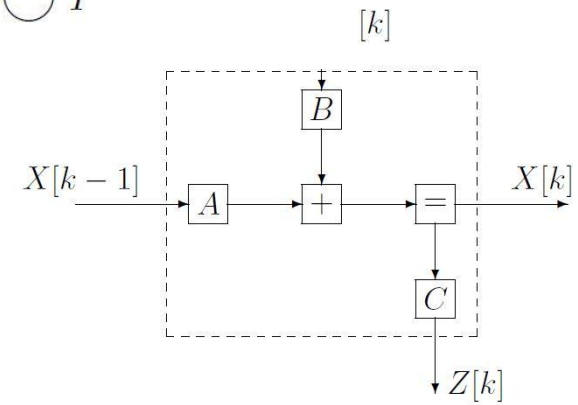
- Factor Graphs allow us to take advantage of the distributive law
- More on this later
- Let us first introduce Factor Graphs…

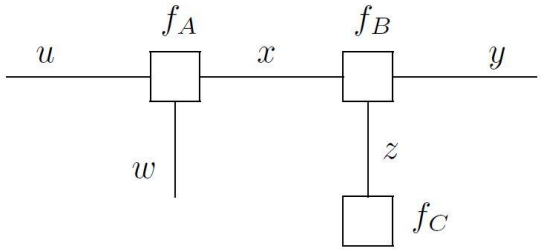
Introduction to Factor Graphs

- (Slightly) differing graphical models available

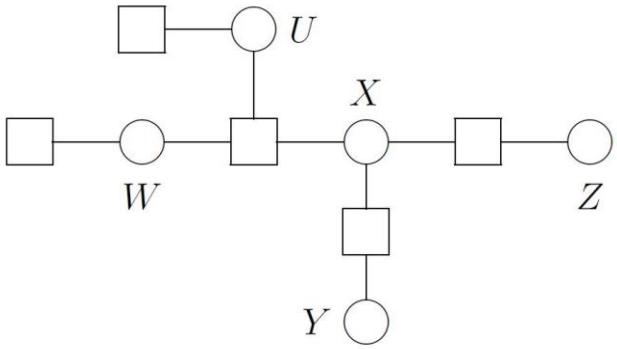






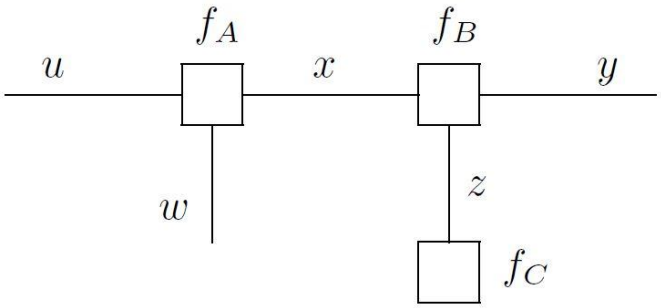


Two types to be used (1)



- Variables represented by circles
- Functions (factors) represented by rectangles

Two types to be used (2)



Forney-style Factor Graph (FFG)

- Edges represent variables
- Rectangles (nodes) represent functions (factors)
- Equality and „EXOR“-node explained later
- How to read these graphs (black board)

Example

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5)$$

A marginal can be computed by

$$f(x_1) = \sum_{\sim x_1} f(x_1, \ldots, x_6)$$

Taking advantage of the distributive law:

$$f(x_1) = \left[\sum_{x_2, x_3} f_1(x_1, x_2, x_3)\right]\left[\sum_{x_4} f_3(x_4)\left(\sum_{x_6} f_2(x_1, x_4, x_6)\right)\left(\sum_{x_5} f_4(x_4, x_5)\right)\right]$$

Application of Factor Graphs

- Graphs of codes
- In this course, we consider Channel Coding
  - Error detection/correction
  - Linear codes (representable by vector/matrix operations)

$$C = \{\boldsymbol{x} \in \boldsymbol{F}^n : H\boldsymbol{x}^T = \boldsymbol{0}\}$$

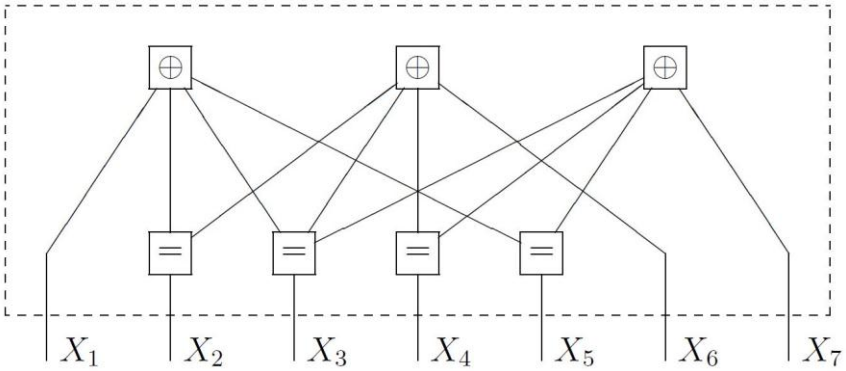
$$C = \{\boldsymbol{x} = \boldsymbol{m}G : \boldsymbol{m} \in \boldsymbol{F}^k\}$$

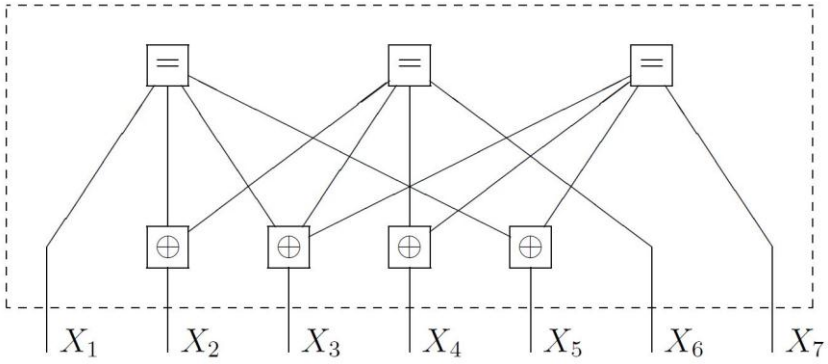
H Parity-Check Matrix

G Generator Matrix

From matrices to FFGs

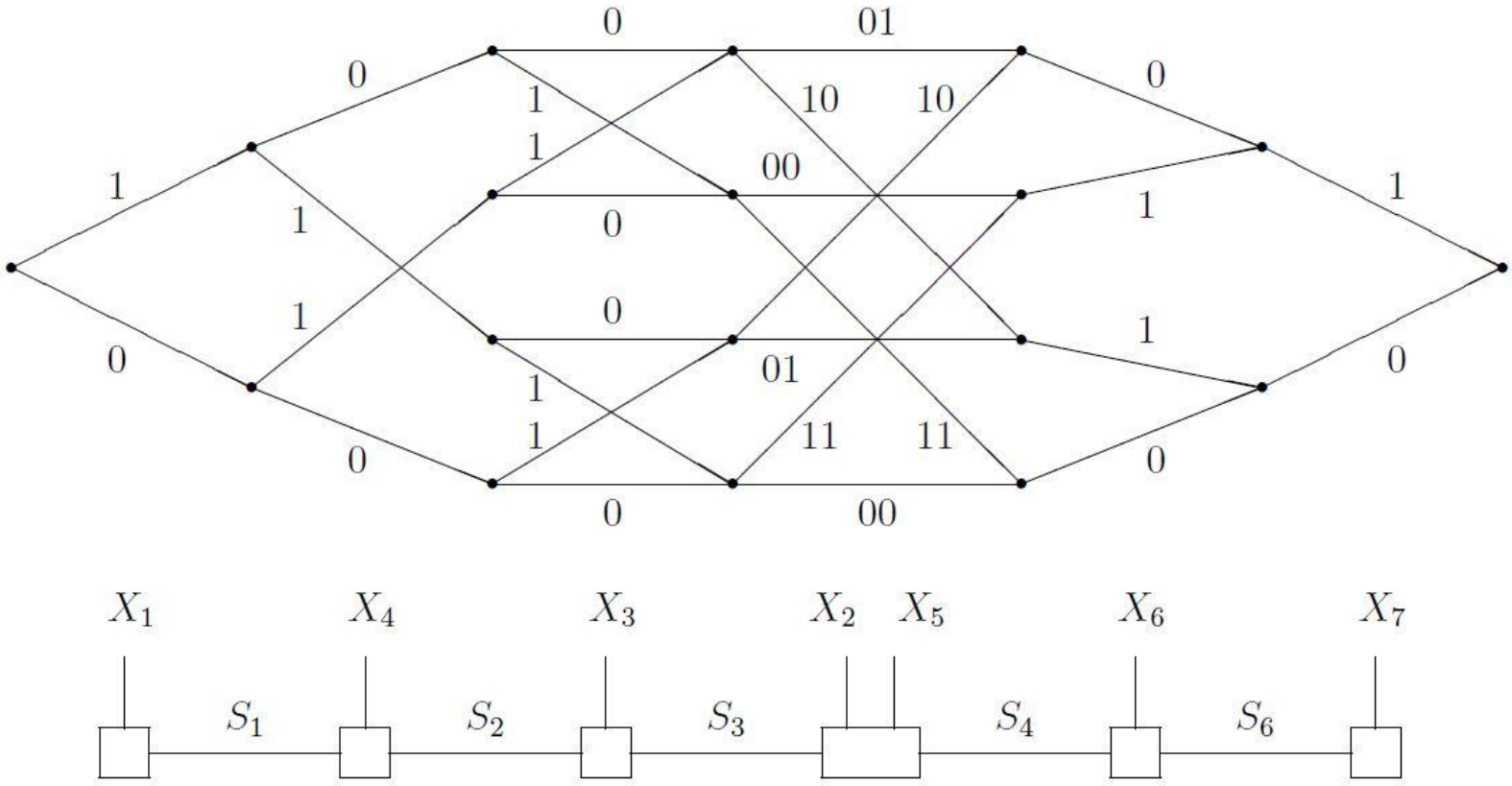
- We can construct a FFG (~Tanner Graph) out of the Parity-Check Matrix (more on this later)
- The dual code can be found easily





- $H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$

Example (Hamming 7,4,3 Code)



Blackboard

- Variables S1 … S6 correspond to the trellis states
- Nodes: {0,1}-valued functions, indicating allowed triples of left state/code symbols/right state.

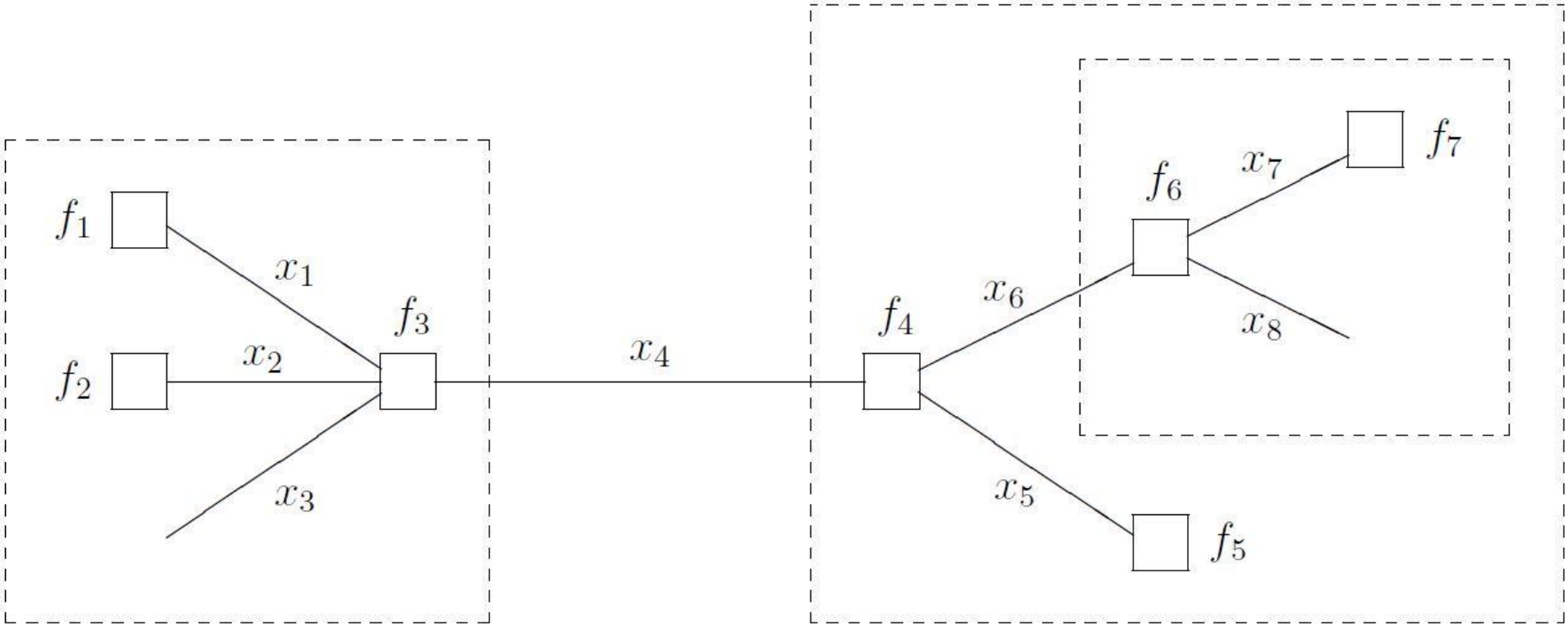
Combined Factor Graphs

- Factor Graphs of the code and channel can be combined
- Channel model: $p(\boldsymbol{y}|\boldsymbol{x})$
- Code: $I_C(\boldsymbol{x})$
- Combined: $p(\boldsymbol{y}|\boldsymbol{x})I_C(\boldsymbol{x})$
- $p(x|y) = \frac{p(y|x)p(x)}{p(y)} \propto p(y|x)I_C(x)$
- The joint code/channel factor graph represents the a posteriori joint probability of the coded symbols

  X1 … Xn

Closing boxes

- Explained best by an example



Example (cont'd)

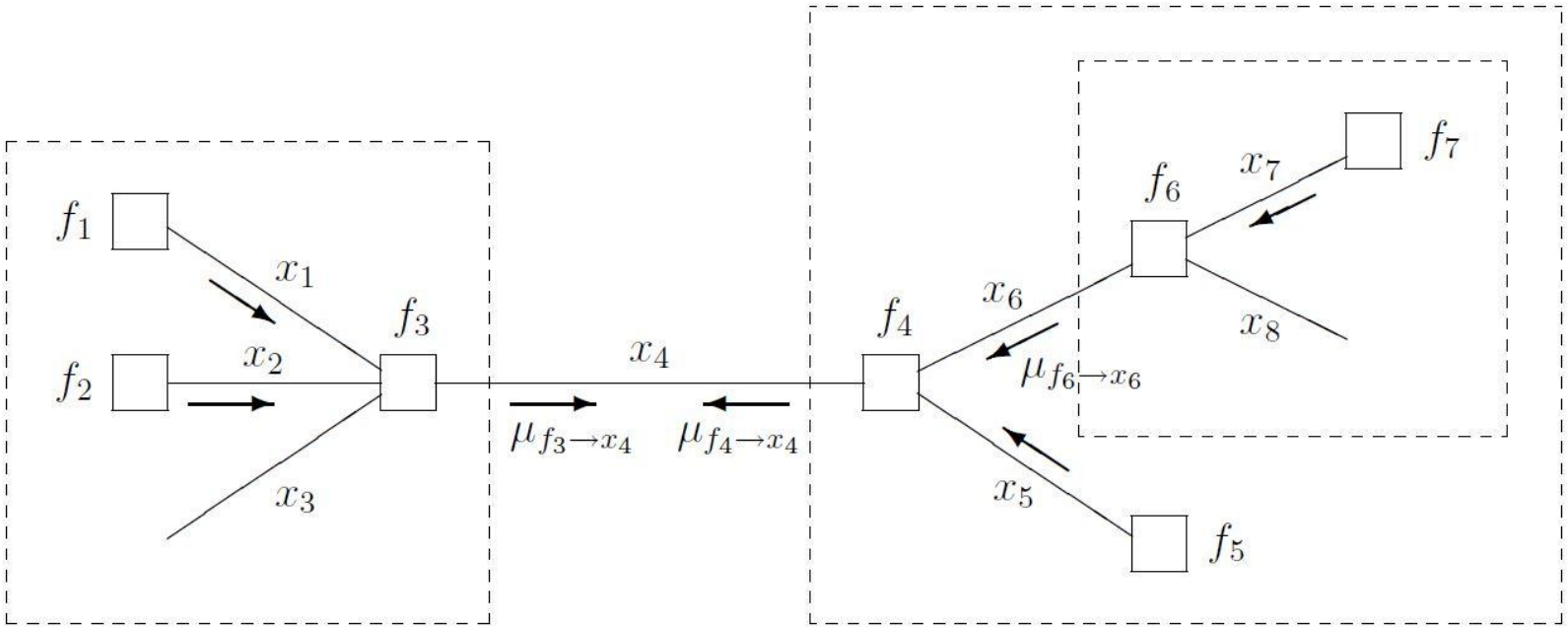
- This factor graph is read as

$$f(x_1,\ldots,x_8) = \Big(f_1(x_1)f_2(x_2)f_3(x_1,x_2,x_3,x_4)\Big)\cdot \Big(f_4(x_4,x_5,x_6)f_5(x_5)\Big(f_6(x_6,x_7,x_8)f_7(x_7)\Big)\Big)$$

- $p(x_4)$ can be determined (distributive law) as

$$p(x_4) = \underbrace{\Big(\sum_{x_1}\sum_{x_2}\sum_{x_3} f_3(x_1,x_2,x_3,x_4)f_1(x_1)f_2(x_2)\Big)}_{\mu_{f_3\to x_4}}\cdot \underbrace{\Big(\sum_{x_5}\sum_{x_6} f_4(x_4,x_5,x_6)f_5(x_5)\underbrace{\Big(\sum_{x_7}\sum_{x_8} f_6(x_6,x_7,x_8)f_7(x_7)\Big)}_{\mu_{f_6\to x_6}}\Big)}_{\mu_{f_4\to x_4}}$$

Example (cont'd)



$$p(x_4) = \mu_{f_3 \to x_4}(x_4) \cdot \mu_{f_4 \to x_4}(x_4)$$

Conclusions

- Summary-Product Rule: The message out of a factor node f(x,…) along the edge x is the product of f(x,…) and all messages towards f along all edges except x, summarized over all variables except x.
- Summaries/Marginals can be computed as the product of two messages
- Such messages are summaries of the subsystem „behind“ them
- All messages are computed from other messages according to the summary-product rule
- Works for graphs without cycles