# Chap. 4 Factor Graphs and the Sum-Product Algorithm

**Reference**: F. R. Kschischang, F. J. Frey and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. on Information Theory*, Feb. 2001.

## □ Tanner Graph of a Code

- Tanner (1981)
  - introduced *bipartite graphs* (called *"Tanner graph"*) to describe families of codes which are generalizations of LDPC codes; and
  - also described the sum-product algorithm in this setting.

- A Tanner graph is a bipartite graph whose nodes are partitioned into two disjoint classes and whose edges may only connect one node of one class to a node of the other class, but there are no edges connecting nodes of the same class.
  - *Bit nodes* or *variable nodes*: "visible"
  - *check nodes* or *function nodes*

**Remark:**

1) Wiberg et al. introduced "*hidden (latent) state variables*" and also suggested applications beyond coding.
2) *Factor graphs* apply graph-theoretic models to functions.
   A Tanner graph for a code represents a particular factorization of the characteristic (indicator) function of the code.

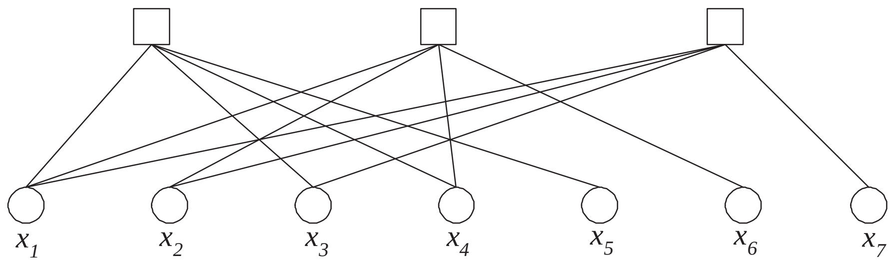
**Example:** $[7,4,3]$ Hamming code $\mathcal{C}$ whose parity-check matrix is given by

$$\begin{array}{cc} & \downarrow \text{ variable nodes} \\ H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} & \leftarrow \text{ check nodes} \end{array}$$

Note that $\mathbf{x} = (x_1, x_2, \ldots, x_7)$ is a codeword of $\mathcal{C}$ iff

$$\begin{cases} x_1 + x_3 + x_4 + x_5 = 0, \\ x_1 + x_2 + x_4 + x_6 = 0, \\ x_1 + x_2 + x_3 + x_7 = 0. \end{cases}$$

The corresponding Tanner graph (or factor graph) can be represented as




special

Define the indication function $[P]$ of the statement $P$ as

$$[P] = \begin{cases} 1, & \text{if } P \text{ is true} \\ 0, & \text{otherwise.} \end{cases}$$

The characteristic (or indicator) function $\chi$ for $\mathcal{C}$ is given by

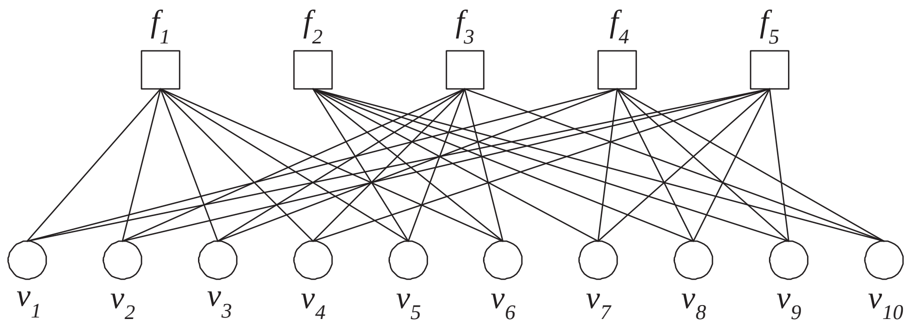
$$\begin{aligned} \chi(x_1, x_2, \cdots, x_7) &= [\ (x_1, x_2, \cdots, x_7) \in C\ ] \\ &= [\ x_1 + x_3 + x_4 + x_5 = 0\ ] \\ &\quad\ [\ x_1 + x_2 + x_4 + x_6 = 0\ ] \\ &\quad\ [\ x_1 + x_2 + x_3 + x_7 = 0\ ] \end{aligned}$$

Note that *a global function can be expressed as a product of local functions.* □

- A Tanner graph for a code is defined by a parity-check matrix $H = (h_{il})$:

$$h_{il} = 1 \quad \Leftrightarrow \quad \text{check node } i \text{ is connected to bit node } l$$

**Example:** (10, 3, 6) regular LDPC code




Let $f_1 : \; v_1 + v_2 + v_3 + v_4 + v_5 + v_6 = 0$, etc. Then the corresponding indicator function is given by

$$\chi_C(v_1, v_2, \cdots, v_{10}) = [f_1][f_2][f_3][f_4][f_5]$$

- **Degree of a node**:

Degree of bit node $l$: $\sum_i h_{il}$ (integer)

Degree of check node $i$: $\sum_l h_{il}$ (integer)

**Example:** $(N, j, k)$ regular LDPC code:

- degree of bit node $l$: $j$ for all $l$
- degree of check node $i$: $k$ for all $i$

- **A cycle of length** $l$ in a Tanner graph is a closed path of $l$ edges.

  **Note:** $l =$ even

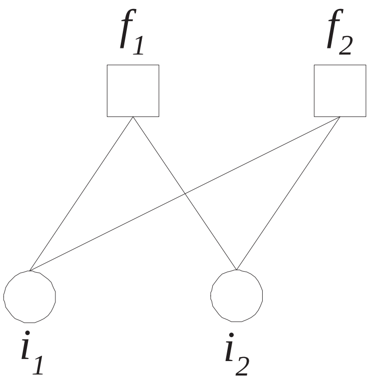
  **Example**: In the previous example, there exist a cycle of length 4.

- **The girth of a Tanner graph** is the minimum cycle length of the graph.

**Remark:**

1) The shortest possible cycle in a bipartite graph is clearly a length-4 cycle.

<u>Length-4 cycle</u>

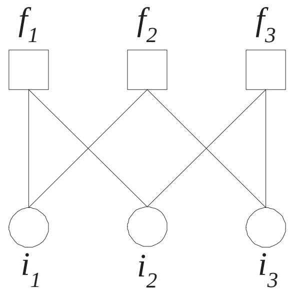



$$\begin{array}{cc} & \begin{array}{cc} i_1 & i_2 \end{array} \\ H = & \left[\begin{array}{cc} 1 & 1 \\ 1 & 1 \end{array}\right] \begin{array}{c} f_1 \\ f_2 \end{array} \end{array}$$

**Example:**

$$H = \left[\begin{array}{cccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

<u>Length-6 cycle</u>




$$\begin{array}{ccc} & \begin{array}{ccc} i_1 & i_2 & i_3 \end{array} & \\ H = & \left[\begin{array}{ccc} 1 & 1 & \\ 1 & & 1 \\ & 1 & 1 \end{array}\right] & \begin{array}{c} f_1 \\ f_2 \\ f_3 \end{array} \end{array}$$

2) Length-4 cycles cause a bad performance for LDPC codes.

3) When the number of iterations increases, dependencies between bit nodes for short-length cycles appear rapidly.

**Remark:**

1) The GDL (Generalized Distributive Law) by Aji and McEliece solves the MPF (marginalize product-of-functions) problem using a "junction-tree" representation of the global function

2) Junction tree / GDL $\longleftrightarrow$ Factor graph / sum-product
Example: iterative decoding of turbo codes and LDPC codes

3) Factor graphs $\longleftrightarrow$ Graphical models for multidimensional probability distributions
   – Markov random fields
   – Bayesian (belief) networks

"sum-product" "belief propagation"

- **Marginal Functions**

  – Global function $g(x_1, x_2, \cdots, x_n) : \underbrace{A_1 \times A_2 \times \cdots \times A_n}_{\triangleq\ S} \longrightarrow \underbrace{R}_{\text{any semiring}}$

  · $S$ = configuration space

  · $(a_1, a_2, \cdots, a_n) \in S$: a configuration of the variables

  – Marginal functions: $g_i(x_i), \quad i = 1, 2, \cdots, n.$

  – Summary operation for $x_2$ (or "the not-sum $x_2$"):

$$\sum_{\sim\{x_2\}} h(x_1, x_2, x_3) \triangleq \sum_{x_1 \in A_1} \sum_{x_3 \in A_3} h(x_1, x_2, x_3)$$

  – Then each marginal function $g_i(x_i)$ can be expressed as

$$g_i(x_i) = \sum_{\sim\{x_i\}} g(x_1, \cdots, x_n).$$

- **Factorization of a global function**:

  Suppose $g(x_1, \cdots, x_n)$ factors into a product of several local functions, each having some subset of $\{x_1, \cdots, x_n\}$ as arguments, i.e.,

$$g(x_1, \cdots, x_n) = \prod_{j \in J} f_j(X_j)$$

  where

  $J$: a discrete index set

  $X_j \subset \{x_1, \cdots, x_n\}$

  $f_j(X_j)$: a function having the elements of $X_j$ as arguments called "*local function*"

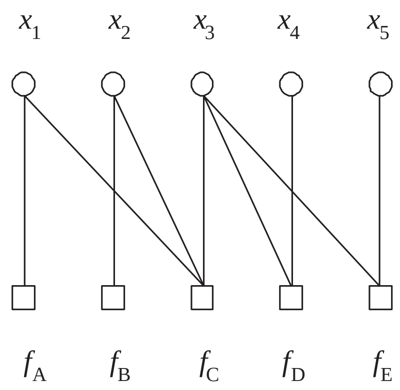
- **Components of a factor graph**

A factor graph is a bipartite graph that *expresses the structure of the factorization*:

1) variable node for each variable $x_i$;
2) factor node for each local function $f_j$; and
3) an edge connects variable node $x_i$ to factor node $f_j$ if and only if $x_i$ is an argument of $f_j$.

**Example:** Consider

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_c(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5).$$



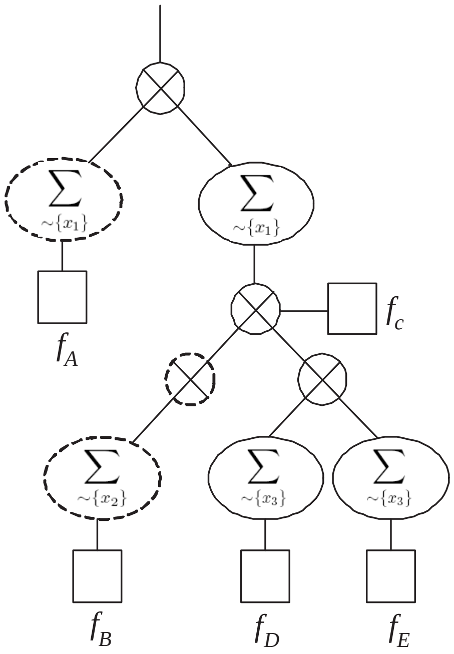

The marginal function $g_1(x_1)$ can be expressed as

$$\begin{aligned} g_1(x_1) &= f_A(x_1) \left( \sum_{x_2} f_B(x_2) \left( \sum_{x_3} f_C(x_1, x_2, x_3) \right.\right. \\ &\qquad \left.\left. \cdot \left( \sum_{x_4} f_D(x_3, x_4) \right) \left( \sum_{x_5} f_E(x_3, x_5) \right) \right) \right) \end{aligned}$$

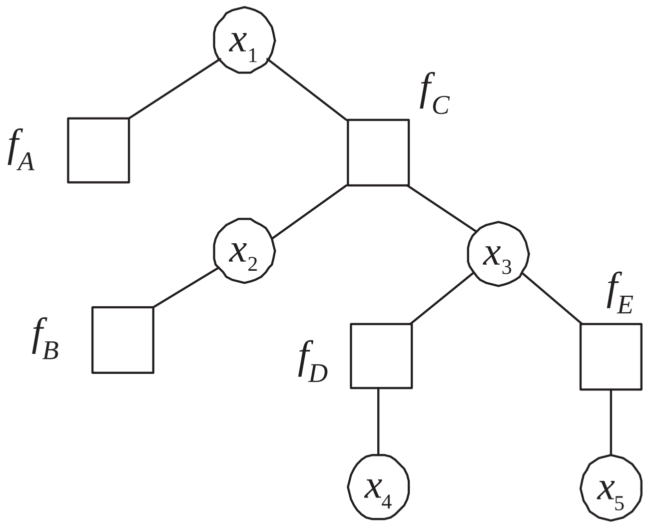
or

$$\begin{aligned} g_1(x_1) &= f_A(x_1) \times \sum_{\sim\{x_1\}} f_B(x_2) f_C(x_1, x_2, x_3) \\ &\qquad \cdot \left( \sum_{\sim\{x_3\}} f_D(x_3, x_4) \right) \left( \sum_{\sim\{x_3\}} f_E(x_3, x_5) \right) \end{aligned}$$

tree-representation




factor graph

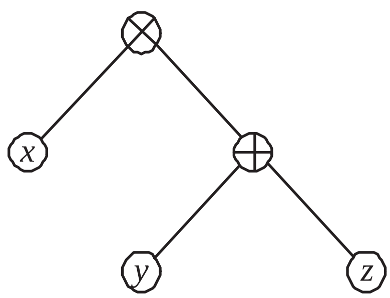



- **Expression trees**
  - Internal vertices: vertices with descendants

    $\rightarrow$ arithmetic operators (eg. addition, multiplication, negation, etc.).
  - leaf vertices: vertices without descendants

    $\rightarrow$ variables or constants

**Example:** Expression tree for $x(y+z)$



**Note:**

1) *Every expression tree represents an algorithm for computing the corresponding expression.*

2) To compute marginal functions,
   - bottom-up procedure
   - top-down procedure

- **When a factor graph is cycle-free, the factor graph encodes**

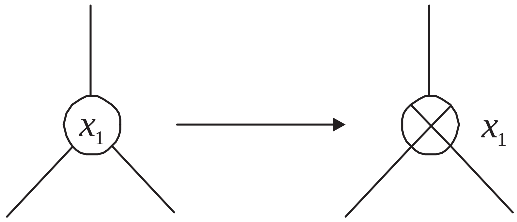
1) the factorization of the global function in its structure; and

2) the arithmetic expressions by which the marginal functions associated with the global function may be computed.

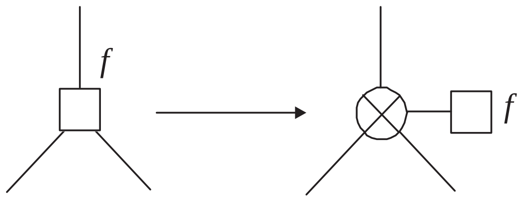
$\Rightarrow$ *The sum-product algorithm computes marginal functions.*

- **Transformation from a Factor Graph to an Expression Tree for $g_i(x_i)$**

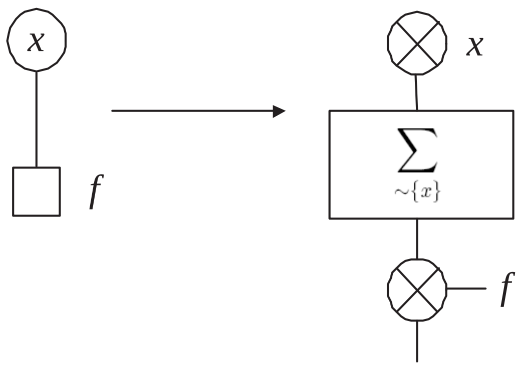
1) Variable node $x_1$ $\longrightarrow$ product operator




2) Factor node $f$ $\longrightarrow$ a form product and multiply by $f$ operator




3) Edge between a factor node $f$ and its parent $x$ $\longrightarrow$ summary operator $\sum_{\sim\{x\}}$

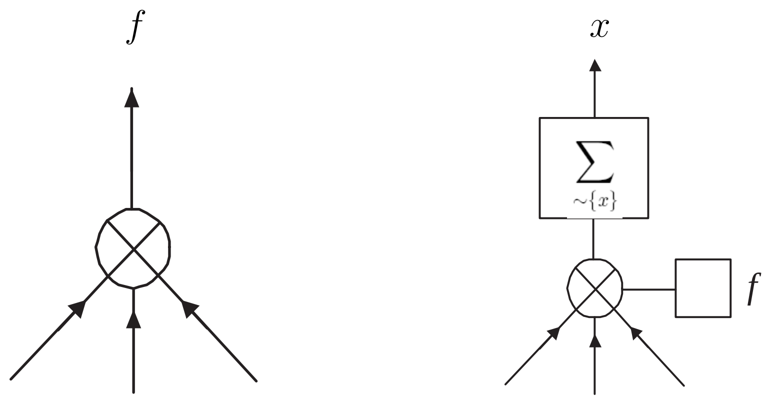



cf. trivial nodes

- **Message Passing Algorithm to Compute** $g_i(x_i)$

  (in a rooted cycle-free factor graph, with $x_i$ taken as a root vertex)

1) variable node 2) factor node




**Remark:**

1) A message passed on the edge $\{x, f\}$, either from variable $x$ to factor $f$, or vice versa, is *a single-argument function of* $x$, the variable associated with the given edge.
2) At every factor node, summary operations are always performed for the variable associated with the edge on which the message is passed.
3) At a variable node, all messages are functions of that variable, and so is any product of these messages.
4) These local transformation rules always work under the assumptions:
   - the distributive law holds, i.e.,

$$x(y+z) = xy + yz, \quad \forall x, y, z \in R;$$

   - the graph is cycle-free.

- **The Sum-Product Algorithm can be described very naturally as a Message-Passing Algorithm**:
  - Vertices are "*processors*".
  - Edges are "*channels*" between processors.
  - "*Messages*" sent over channels are simply appropriate descriptions of local functions
  - Any particular processor can "*fire*" once it has received messages from its children
  - Start from the leaves and send messages "*up*" towards the root
  - The marginal function $g_i(x_i)$ is the "*final*" message; namely, the product of all messages sent to $x_i$

**Remark:**

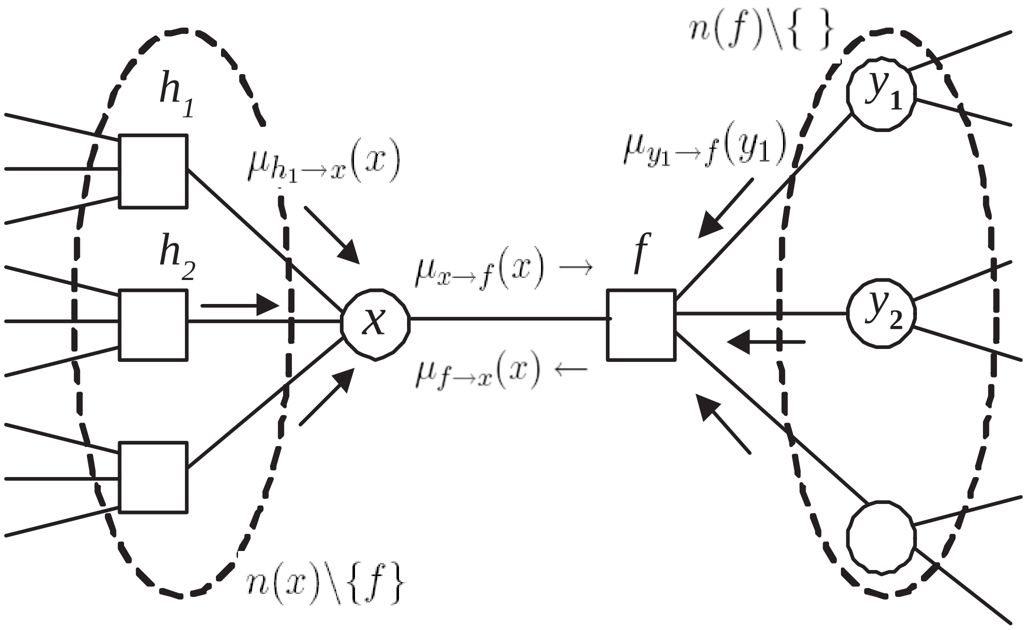
1) Messages are descriptions of functions, say, a list of values, a parametrization, etc.

2) Prob. mass function for Bernoulli random variables:

$$(P_0, P_1), \quad P_0 - P_1, \quad P_0/P_1, \quad \ln(P_0/P_1)$$

(since $P_0 + P_1 = 1$, the Gaussian pdf is given by the parameter pair $(m, \sigma^2)$.)

- **Sum-Product Algorithm**




1) variable to local function (*product rule*)

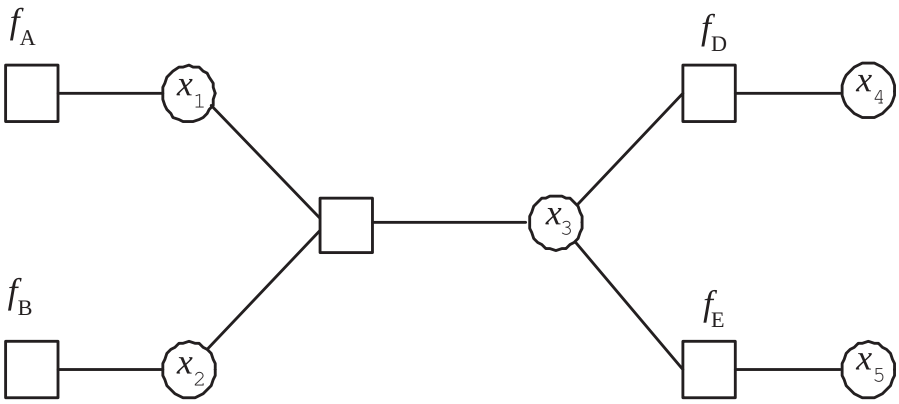
$$\mu_{x\to f}(x) = \prod_{h\in n(x)\backslash\{f\}} \mu_{h\to x}(x)$$

2) local function to variable (*sum-product rule*)

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} f(X) \prod_{y\in n(f)\backslash\{x\}} \mu_{y\to f}(y)$$

where $X = n(f)$ is the set of arguments of the function $f$

**Example:**




Initiation:

$$
\begin{aligned}
\mu_{f_A \to x_1}(x_1) &= \sum_{\sim\{x_1\}} f_A(x_1) = f_A(x_1), \\
\mu_{f_B \to x_2}(x_2) &= \sum_{\sim\{x_2\}} f_B(x_2) = f_B(x_2), \\
\mu_{x_4 \to f_D}(x_4) &= 1, \\
\mu_{x_5 \to f_E}(x_5) &= 1.
\end{aligned}
$$

Termination:

$$
\begin{aligned}
g_1(x_1) &= \mu_{f_A \to x_1}(x_1)\ \mu_{f_C \to x_1}(x_1), \\
g_2(x_2) &= \mu_{f_B \to x_2}(x_2)\ \mu_{f_C \to x_2}(x_2), \\
g_3(x_3) &= \mu_{f_C \to x_3}(x_3)\ \mu_{f_D \to x_3}(x_3)\ \mu_{f_E \to x_3}(x_3), \\
g_4(x_4) &= \mu_{f_D \to x_4}(x_4), \\
g_5(x_5) &= \mu_{f_E \to x_5}(x_5).
\end{aligned}
$$

**Note:**

1) $g_i(x_i)$ = product of all messages directed toward $x_i$

2) $g_i(x_i)$ = product of the two messages that were passed (in opposite directions) over any single edge incident on $x_i$,

since the message passed on any given edge is equal to the product of all but one of these messages.

**Example:**

$$\begin{aligned} g_3(x_3) &= \mu_{f_C \to x_3}(x_3)\ \mu_{x_3 \to f_C}(x_3) \\ &= \mu_{f_D \to x_3}(x_3)\ \mu_{x_3 \to f_D}(x_3) \\ &= \mu_{f_E \to x_3}(x_3)\ \mu_{x_3 \to f_E}(x_3) \end{aligned}$$

## □ Modeling Systems with Factor Graphs

- **A system is a collection of interacting variables**:

  1) *Probabilistic modeling*: system behavior is specified in probabilistic terms.
     - joint probability mass function of the variables.
     - factorization $\Rightarrow$ statistical dependencies among these variables.

  2) *Behavioral modeling*: system behavior is specified in set-theoretic terms.
     - characteristic (i.e., set-indicator) function
     - factorization $\Rightarrow$ structural information about the model

  3) *Combined modeling*: Probabilistic modeling $+$ Behavioral modeling

     Example: In channel coding,
     - valid behavior: set of codewords
     - a posteriori joint probability mass function

- **Iverson's convention**

  1) If $P$ is a Boolean proposition, then

  $$[P] = \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

  2) If $P = P_1 \wedge P_2 \wedge \cdots \wedge P_n$, then

  $$[P] = [P_1][P_2]\cdots[P_n] \;=\; \prod_{i=1}^{n}[P_i].$$

  3) If $P$ is a logical conjunction of predicates, then $[P]$ can be factored and hence represented using a factor graph.

# □ Behavioral Modeling

- **Behavioral Modeling of a Linear Code**

1) A behavior $C \subset S = \mathcal{A}^n$ is called a block code of length $n$ over $\mathcal{A}$, and the valid configurations are called codewords.

2) The characteristic (or set membership indicator) function for a behavior $B$ is degined as

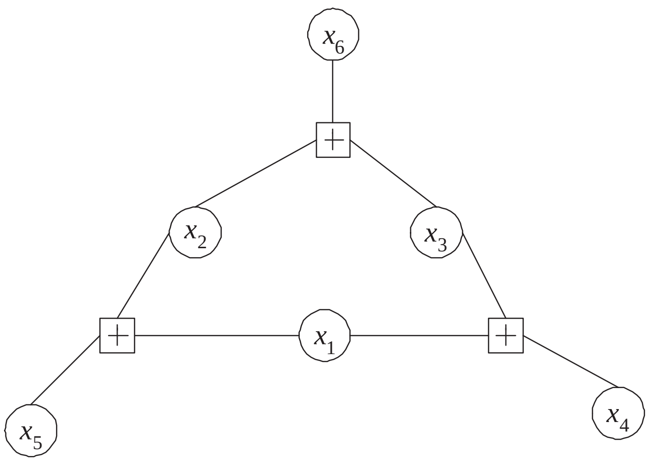
$$\mathcal{X}_B(x_1, \cdots, x_n) \triangleq [(x_1, \cdots, x_n) \in B]$$

**Example:** Tanner Graph for the Linear Code $\mathcal{C} = \{\mathbf{x} \in F_2 | H\mathbf{x} = 0\}$ with

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

Note that

$$\begin{aligned} \mathcal{X}_C(x_1, \cdots, x_6) &= [(x_1, \cdots, x_6) \in \mathcal{C}] \\ &= [x_1 + x_2 + x_5 = 0][x_2 + x_3 + x_6 = 0][x_1 + x_3 + x_4 = 0] \end{aligned}$$



- **Hidden Variables**

1) Introducing *hidden (auxiliary, latent, or state) variables* in a model can give the designer more freedom in modeling the system.

2) *Non-hidden variables are visible.*

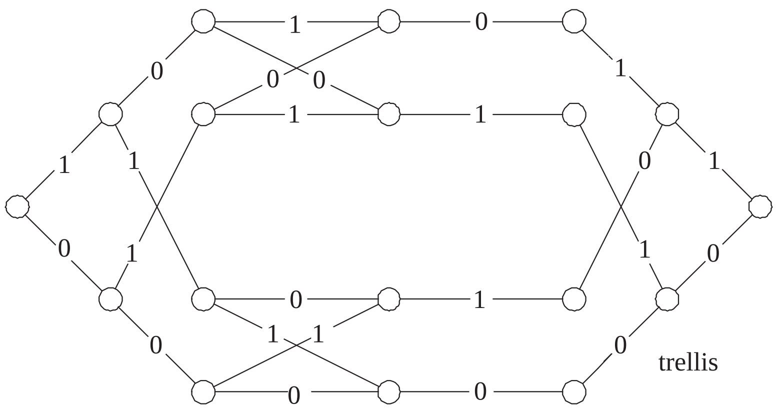
3) A behavior $\mathcal{B}$ (with both hidden and visible variables) represents a given (visible) behavior $\mathcal{C}$ if the projection of the elements of $\mathcal{B}$ on their visible coordinates is equal to $\mathcal{C}$.

4) A factor graph for $\mathcal{B}$ is then considered to be a factor graph for $\mathcal{C}$.

5) An important class of models with hidden variables are the *trellis representations*.

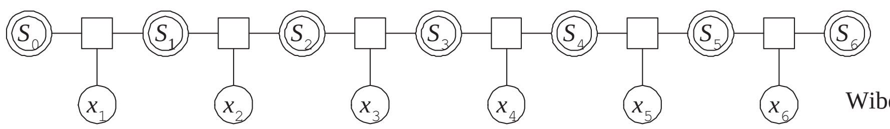
**Example:**

– Trellis



trellis

– Wiberg-type graph




Wibey-type graph

6) A trellis divides naturally into $n$ sections, where the $i$th section $T_i$ is the subgraph of the trellis induced by the vertices at depth $i-1$ and depth $i$.

   $\Rightarrow$ $T_i$: local behavior on $s_{i-1}, x_i, s_i$

**Example:** In the previous example,

– variable nodes: $x_1, x_2, \cdots, x_6$

– hidden (state) variable nodes: $s_0, s_1, \cdots, s_6$

– factor nodes: $T_1, T_2, \cdots, T_6$

  For example,

$$T_2 = \{(0,0,0), (0,1,2), (1,1,1), (1,0,3)\}$$

  where $s_1 \in \{0,1\}$, $s_2 \in \{0,1,2,3\}$

– The corresponding factor node in the Wiberg-type graph is the indicator fuction

$$f(s_1, x_2, s_2) = [(s_1, x_2, s_2) \in T_2].$$

**Remark:**

1) A factor graph corresponding to a trellis is cycle-free.

2) Every code can be represented by a cycle-free factor.

3) The state-space sizes (the sizes of domains of the state variables) can easily become too large to be practical.

- **State-Space Models**

A system can be described as

$$\begin{aligned} \mathbf{x}(j+1) &= A\,\mathbf{x}(j) + B\,\mathbf{u}(j) \quad \text{(over F)} \\ \mathbf{y}(j) &= C\,\mathbf{x}(j) + D\,\mathbf{u}(j) \end{aligned}$$

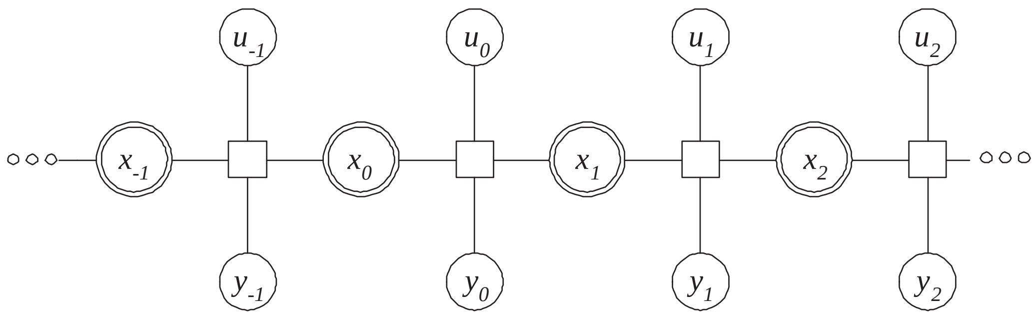
where $j \in \mathbb{Z}$ is the the discrete time index,

$$\begin{aligned} \mathbf{u}(j) &= (u_1(j), \cdots, u_k(j))^t, && \text{the time-j input vector;} \\ \mathbf{y}(j) &= (y_1(j), \cdots, y_n(j))^t, && \text{the time-j output vector;} \\ \mathbf{x}(j) &= (x_1(j), \cdots, x_m(j))^t, && \text{the time-j state vector.} \end{aligned}$$

The time-$j$ check function $f : F^m \times F^k \times F^n \times F^m \rightarrow \{0,1\}$ is given by

$$\begin{aligned} f(\mathbf{x}(j), \mathbf{u}(j), \mathbf{y}(j), \mathbf{x}(j+1)) &= [\mathbf{x}(j+1) = A\,\mathbf{x}(j) + B\,\mathbf{u}(j)] \\ &\quad \cdot [\mathbf{y}(j) = C\,\mathbf{x}(j) + D\,\mathbf{u}(j)]. \end{aligned}$$

The factor graph for a state-space model of a time-invariant or time-varying system can be described as follows:



## □ Probabilistic Modeling

Let $x = (x_1, x_2, \cdots, x_n) \in \mathcal{C}$ be a transmitted codeword and assume that $y = (y_1, y_2, \cdots, y_n)$ is received through a memoryless channel.

A priori distribution:

$$p(x) = \mathcal{X}_{\mathcal{C}}(x)/|\mathcal{C}|$$

where

$\mathcal{X}_{\mathcal{C}}(x)$: characteristic function for $\mathcal{C}$,

$|\mathcal{C}|$ : number of codewords in $\mathcal{C}$.

For each fixed observation $y$, the joint a posteriori probability (APP) distribution $P(x|y)$ can be determined by

$$P(x|y) \;\sim\; g(x) \;\triangleq\; f(y|x)\,p(x)$$

where

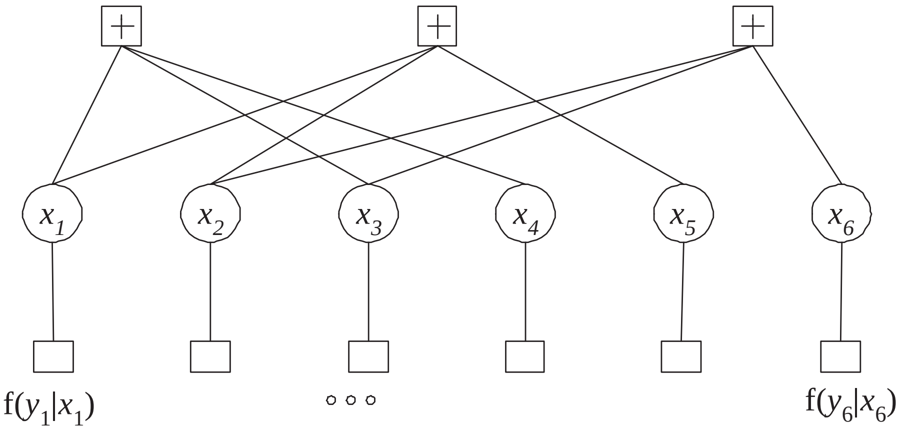
$$f(y|x) \;=\; \prod_{i=1}^{n} f(y_i|x_i)$$

is the conditional pdf for $y$ when $x$ is transmitted.

Note that

$$g(x_1, \cdots, x_n) \;=\; \frac{1}{|\mathcal{C}|}\mathcal{X}_{\mathcal{C}}(x_1, \cdots, x_n)\prod_{i=1}^{n} f(y_i|x_i).$$

**Example:** Factor graph with conditional pdf:

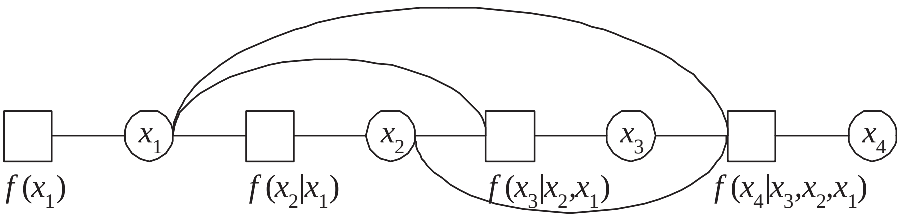



- **Chain Rule of Conditional Probability**

$$f(x_1, \cdots, x_n) = \prod_{i=1}^{n} f(x_i | x_1, \cdots, x_{i-1})$$

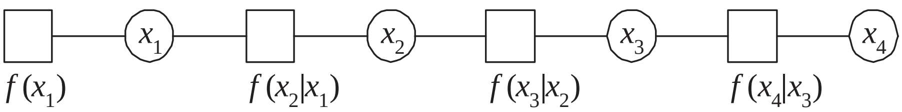
Example: $n = 4$

$$f(x_1, \cdots, x_4) = f(x_1) f(x_2|x_1) f(x_3|x_1, x_2) f(x_4|x_1, x_2, x_3)$$




- **Markov Chain**

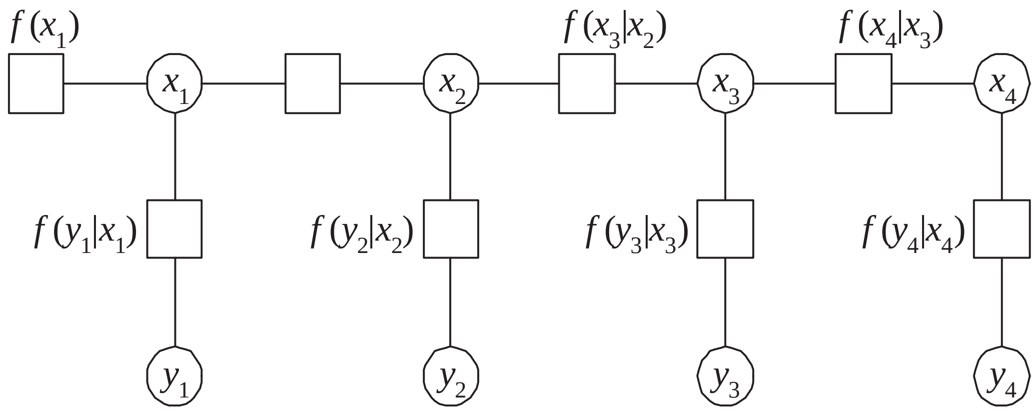
$$f(x_1, \cdots, x_n) = \prod_{i=1}^{n} f(x_i | x_{i-1})$$



- **Hidden Markov Models**
  - *Cannot observe each $X_i$ directly;*
  - Can observe only $Y_i$
    (where $Y_i$ is the output of a memoryless channel with $X_i$ as input)
  - PDF in the hidden Markov models:

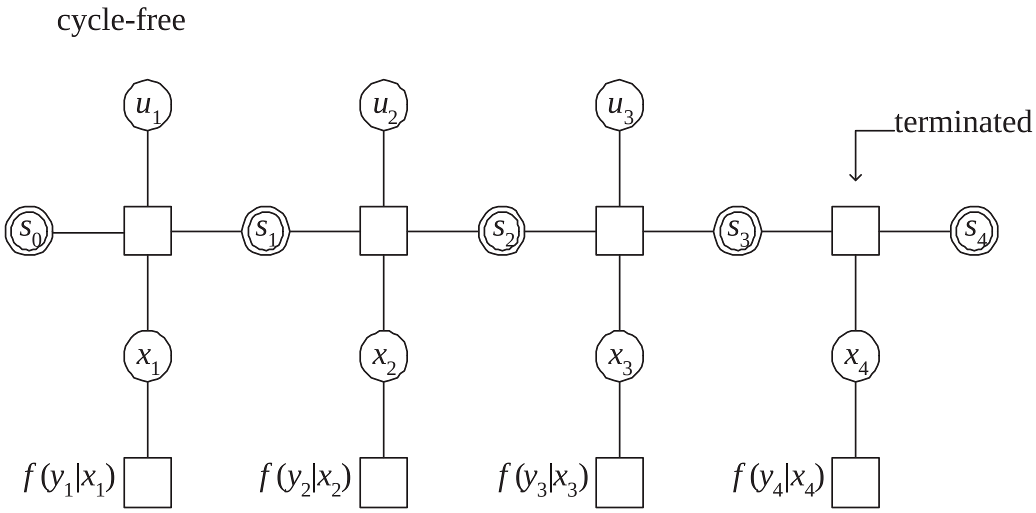
$$f(x_1, \cdots, x_n, y_1, \cdots, y_n) = \prod_{i=1}^{n} f(x_i|x_{i-1}) f(y_i|x_i)$$

  - Factor graph of a hidden Markov model:



## ☐ The Forward / Backward Algorithm

- BCJR, APP, or MAP Algorithm
- **Factor graph (of a "hidden" Markov model)**




- Given the observation $y$, the a posteriori joint probability mass function for $u$, $s$ and $x$ can be determined by

$$g_y(u,s,x) \triangleq \prod_{i=1}^{n} T_i(s_{i-1},x_i,u_i,s_i) \cdot \prod_{i=1}^{n} f(y_i|x_i)$$

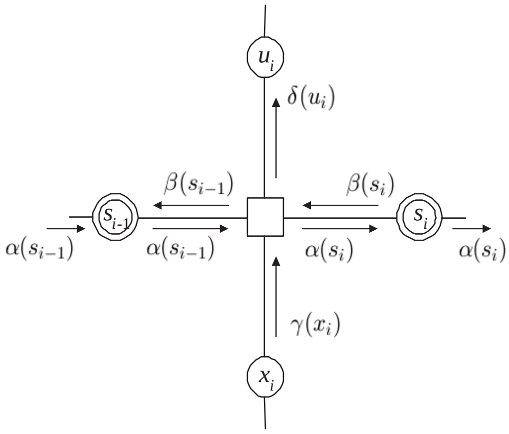
where

$$\begin{aligned} u &= (u_1,u_2,\cdots,u_k) : \text{input variables} \\ x &= (x_1,x_2,\cdots,x_n) : \text{output variables} \\ s &= (s_0,s_1,\cdots,s_n) : \text{state variables} \\ &T_i(s_{i-1},x_i,u_i,s_i) : \text{local check functions} \end{aligned}$$

- **Goal**: Given $y$, compute the APPs $p(u_i|y)$ for all $i$.

Note that

$$p(u_i|y) \propto \sum_{\sim\{u_i\}} g_y(u,s,x) \qquad (\Rightarrow \text{ sum-product algorithm})$$




$$\begin{aligned}
\mu_{x_i \to T_i}(x_i) &= \gamma(x_i) \triangleq \Pr(s_i, y_i \,|\, s_{i-1}) \\
\mu_{s_i \to T_{i+1}}(s_i) &= \alpha(s_i) \triangleq \Pr(s_i, \mathbf{y}_1^i) \\
&\sim p(s_i | y_1, \cdots, y_i) \\
\mu_{s_i \to T_i}(s_i) &= \beta(s_i) \triangleq \Pr(\mathbf{y}_{i+1}^N \,|\, s_i) \\
&\sim p(s_i | y_{i+1}, \cdots, y_n) \\
\mu_{T_i \to u_i}(u_i) &= \delta(u_i)
\end{aligned}$$

- **The Forward/Backward Recursions**

$$\begin{aligned}
\alpha(s_i) &= \sum_{\sim\{s_i\}} T_i(s_{i-1}, u_i, x_i, s_i)\, \alpha(s_{i-1})\, \gamma(x_i) \\
\beta(s_{i-1}) &= \sum_{\sim\{s_{i-1}\}} T_i(s_{i-1}, u_i, x_i, s_i)\, \beta(s_i)\, \gamma(x_i)
\end{aligned}$$

- **Termination**

$$\delta(u_i) = \sum_{\sim\{u_i\}} T_i(s_{i-1}, u_i, x_i, s_i)\, \alpha(s_{i-1})\, \beta(s_{i+1})\, \gamma(x_i)$$

- More specially,

$e = (s_{i-1}, u_i, x_i, s_i)$ such that $T_i(e) = 1$

$\alpha(e) \triangleq \alpha(s_{i-1}), \quad \beta(e) \triangleq \beta(s_i), \quad \gamma(e) \triangleq \gamma(x_i)$

$E_i(s) \triangleq$ set of edges incident on a state $s$ in the $i$th trellis section.

Then

$$\alpha(s_i) = \sum_{e \in E_i(s_i)} \alpha(e)\, \gamma(e),$$

$$\beta(s_{i-1}) = \sum_{e \in E_i(s_{i-1})} \beta(e)\, \gamma(e).$$

## □ The Min-Sum and Max-Product Semirings and the Viterbi Algorithm

- **Two Basic Problems in Coding Theory**
  - Determine the APPs for the individual symbols.
  - Determine which valid configuration has largest APP.
    $\Rightarrow$ MLSD (maximum likelihood sequence detection)

- **Commutative semiring**: $(K, +, \cdot)$

1) $(K, +)$ is a commutative monoid:
   $+$ is associative and commutative; and
   there is an additive identity element $0$ such that $k + 0 = k \quad \forall k \in K$.

2) $(K, \cdot)$ is a commutative monoid:
   $\cdot$ is associative and commutative; and
   there is a multiplicative identity element 1 such that $1 \cdot k = k \quad \forall k \in K$.

3) The distributive law holds i.e.,

$$(a \cdot b) + (a \cdot c) = a \cdot (b + c), \quad \forall a, b, c \in K$$

- The fact that the structure of a cycle-free factor graph encodes expressions (i.e., algorithms) for the computation of marginal functions follows from the distributive law.
  $\Rightarrow$ *Generalized distributive law* (by Aji and McEliece, IT 2000)

- **"Max-product" semiring**: $K = [0, \infty)$ with

$$+ \;\rightarrow\; \text{max} \;; \qquad 0 \rightarrow 0$$
$$\cdot \;\rightarrow\; \text{product} \;; \quad 1 \rightarrow 1$$

$$x(\max(y, z)) = \max(xy, xz)$$

For a nonnegative real-valued function $g(x_1, \cdots, x_n)$,

$$\begin{aligned} \max g(x_1, \cdots, x_n) &= \max_{x_1}(\max_{x_2}(\cdots(\max_{x_n} g(x_1, \cdots, x_n))\cdots)) \\ &\triangleq \sum_{\sim\{\}} g(x_1, \cdots, x_n). \end{aligned}$$

Example: MLSD problem in coding theory.

- **Min-sum semiring**: $K = (-\infty, \infty)$ with

$$+ \;\rightarrow\; \text{min} \;; \qquad 0 \rightarrow \infty$$
$$\cdot \;\rightarrow\; \text{sum}(\text{"+"}) \;; \quad 1 \rightarrow 0$$

$$x + \min(y, z) = \min(x + y, x + z)$$

Example: $-\ln P$ in MLSD.

- **Iverson's convention in the general semiring case**:

$$[P] = \begin{cases} u & \text{if } P \text{ is true} \\ z & \text{otherwise} \end{cases}$$

where $u$ = multiplicative identity

$z$ = additive identity.

- **Sum-product algorithm** $\longrightarrow$ **Min-sum algorithm**.

$$\begin{array}{lcl} \text{sum} & \longmapsto & \text{min} \\ \text{product} & \longmapsto & \text{sum} \end{array}$$

Example:

$$p(x,s|y) \longmapsto f(x,s|y) = -a \ln p(x,s|y) + b \qquad (a > 0)$$

$$\alpha(s_i) = \sum_{e \in E_i(s_i)} \alpha(e)\gamma(e) \longrightarrow \alpha(s_i) = \min_{e \in E_i(s_i)} (\alpha(e) + \gamma(e))$$

forward / backward algorithm $\longrightarrow$ "bidirectional" Viterbi algorithm.

- The operation of the sum-product algorithm in any cycle-free factor graph in which all distributions (factors) are Gaussian can be regarded as *a generalized Kalman filter*, and in a graph with cycles as an iterative approximation to the Kalman filter.

## □ Iterative Processing: The Sum-Product Algorithm in Factor Graphs with Cycles

- **Factor graphs with cycles**

  1) The results of the sum-product algorithm operating in a factor graph with cycles can not in general be interpreted as exact function summaries.

  2) Examples of factor graphs with cycles:

     – Factor graphs of turbo codes, LDPC codes, RA codes etc.

     – However, they achieve near Shannon limit.

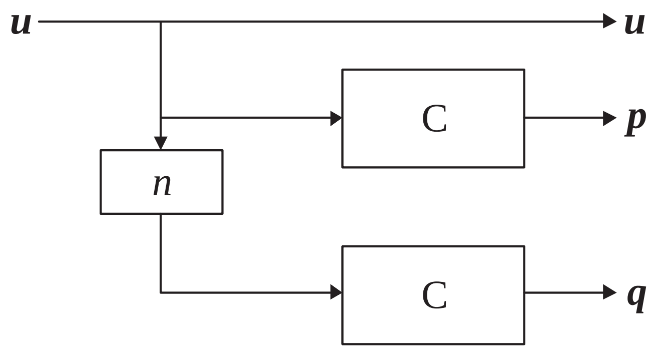
- **Message-Passing Schedules**:
  specifications of messages to be passed during each clock tick.

  1) *Flooding schedule*: A message passes in each direction over each edge at each clock tick.

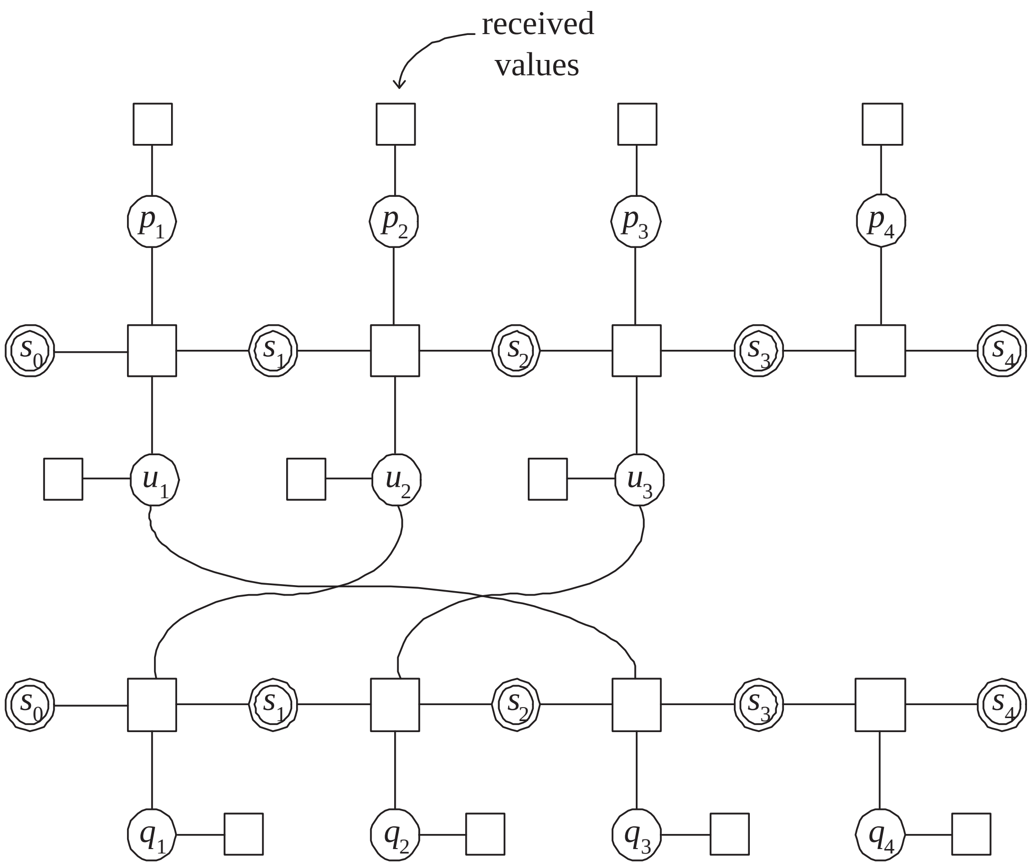
  2) *Serial schedule*: At most one passage is passed anywhere in the graph at each clock tick.

- **Iterative decoding of turbo codes**

1) Encoder

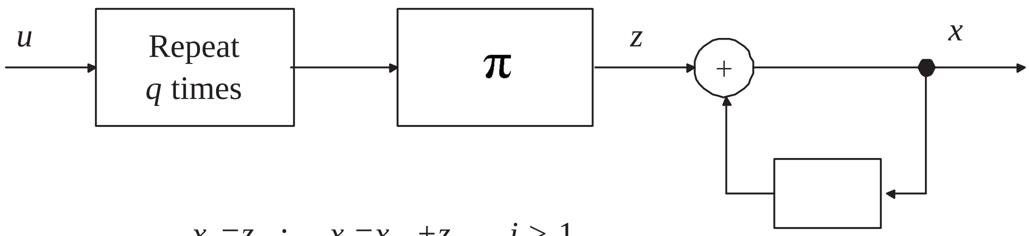



2) Factor graph (with received values)



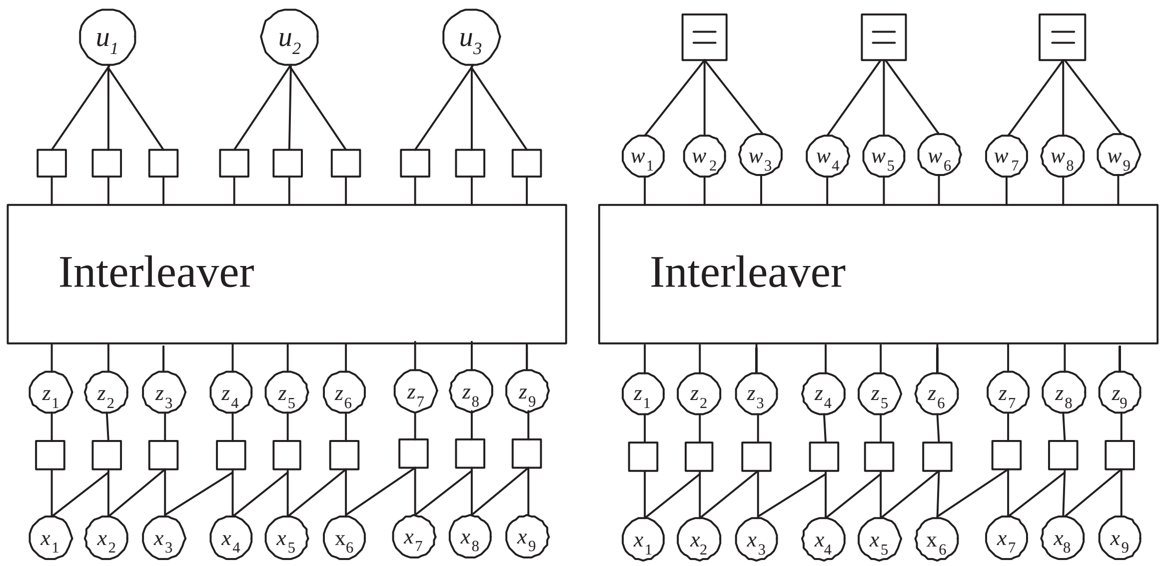
- **Repeat-Accumulate (RA) codes**
  - The ensemble weight distributions are relatively easy to derive;
  - A special low-complexity class of turbo codes;
  - Introduced by Divsalar, McEliece, and Jin.
  - *Repeater + Interleaver + Accumulator*

1) Encoder

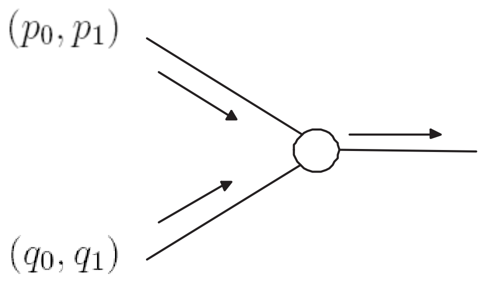



$$x_1 = z_1 \;; \quad x_i = x_{i-1} + z_i \,, \quad i > 1$$

2) Factor graph

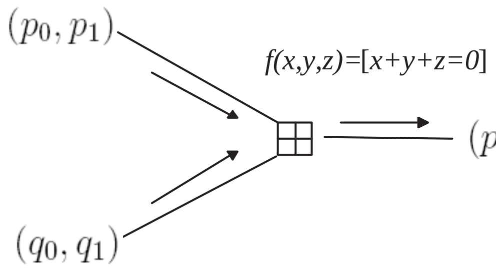


- **Updating Rules for Binary Variables and Parity Checks**



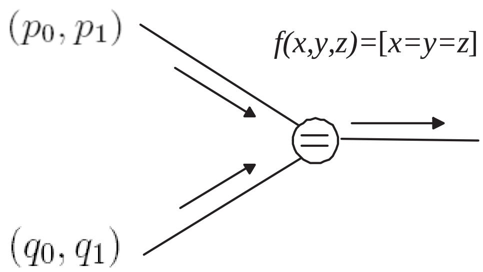

$$\left(\frac{p_0q_0}{p_0q_0+p_1q_1}, \frac{p_1q_1}{p_0q_0+p_1q_1}\right) = \mathsf{VAR}(p_0, p_1; q_0, q_1)$$

< variable node >




$$(p_0q_0 + p_1q_1, p_0q_1 + p_1q_0) = \mathsf{CHK}(p_0, p_1; q_0, q_1)$$

< check node >




$$\mathsf{REP}(p_0, p_1; q_0, q_1) = \mathsf{VAR}(p_0, p_1; q_0, q_1)$$

< repetition constraints >

- **Parametrizations** (using that $p_0 + p_1 = 1$)

1) *Likelihood ratio (LR)*: $\lambda(p_0, p_1) \triangleq p_0/p_1$

$$\begin{aligned} \mathsf{VAR}(\lambda_1, \lambda_2) &= \lambda_1\lambda_2 \\ \mathsf{CHK}(\lambda_1, \lambda_2) &= \frac{1+\lambda_1\lambda_2}{\lambda_1+\lambda_2} \end{aligned}$$

2) *Log-likelihood ratio (LLR)*: $\Lambda(p_0, p_1) = \ln(p_0/p_1)$

$$\begin{aligned} \mathsf{VAR}(\Lambda_1, \Lambda_2) &= \Lambda_1 + \Lambda_2 \\ \mathsf{CHK}(\Lambda_1, \Lambda_2) &= \ln\cosh\left(\frac{\Lambda_1+\Lambda_2}{2}\right) - \ln\cosh\left(\frac{\Lambda_1-\Lambda_2}{2}\right) \\ &= 2\tanh^{-1}\left[\tanh\left(\frac{\Lambda_1}{2}\right)\tanh\left(\frac{\Lambda_2}{2}\right)\right] \end{aligned}$$

3) *Likelihood Difference (LD)*: $\delta(p_0, p_1) = p_0 - p_1$

$$\begin{aligned} \mathsf{VAR}(\delta_1, \delta_2) &= \frac{\delta_1+\delta_2}{1+\delta_1\delta_2} \\ \mathsf{CHK}(\delta_1, \delta_2) &= \delta_1\delta_2 \end{aligned}$$

4) *Signed Log-likelihood Difference (SLLD)*: $\Delta(p_0, p_1) = \mathsf{sgn}(p_1 - p_0)\ln|p_1 - p_0|$

$$\mathsf{VAR}(\Delta_1, \Delta_2) = \begin{cases} s \cdot \ln\left(\frac{\cosh((|\Delta_1|+|\Delta_2|)/2)}{\cosh((|\Delta_1|-|\Delta_2|)/2)}\right) & \text{if } \mathsf{sgn}(\Delta_1) = \mathsf{sgn}(\Delta_2) = s \\ s \cdot \mathsf{sgn}(|\Delta_1| - |\Delta_2|) \cdot \ln\frac{\sinh\left(\frac{|\Delta_1|+|\Delta_2|}{2}\right)}{\sinh\left(\frac{|\Delta_1|-|\Delta_2|}{2}\right)} & \\ & \text{if } \mathsf{sgn}(\Delta_1) = -\mathsf{sgn}(\Delta_2) = -s \end{cases}$$

$$\mathsf{CHK}(\Lambda_1, \Lambda_2) = \mathsf{sgn}(\Delta_1)\mathsf{sgn}(\Delta_2)(|\Delta_1| + |\Delta_2|)$$

**Exercise**: Show the validity of the results in 1), 2), 3) and 4) hold.

**Note**:

1) For $x \gg 1$, the function $\ln(\cosh(x))$ can be approximated as

$$\ln(\cosh(x)) \approx |x| - \ln 2.$$

Then the updating rule at the check node can be reduced to

$$\begin{aligned} \mathsf{CHK}(\Lambda_1, \Lambda_2) &\approx \left|\frac{\Lambda_1 + \Lambda_2}{2}\right| - \left|\frac{\Lambda_1 - \Lambda_2}{2}\right| \\ &= \mathsf{sgn}(\Lambda_1)\mathsf{sgn}(\Lambda_2)\min(|\Lambda_1|, |\Lambda_2|), \end{aligned}$$

which is called the *min-sum update rule*.

2) Updating rule at nodes of higher degree:

$$\begin{aligned} \mathsf{VAR}(x_1, x_2, \cdots, x_n) &= \mathsf{VAR}(x_1, \mathsf{VAR}(x_2, \cdots, x_n)) \\ \mathsf{CHK}(x_1, x_2, \cdots, x_n) &= \mathsf{CHK}(x_1, \mathsf{CHK}(x_2, \cdots, x_n)) \end{aligned}$$

## □ Factor-Graph Transformations

*Modify a factor graph with an inconvenient structure into a more convenient form.*

Example: Factor graph with cycles → cycle-free factor graph

(at the expense of increasing the complexity of the local functions and/or the domains of the variables)

- **Clustering**: $v, w \quad \rightarrow \quad (v, w)$

1) $v, w$: variables with domains $A_v$ and $A_w \quad \rightarrow \quad (v, w) : A_v \times A_w$

   ⇒ a substantial cost increase in computational complexity of the SPA

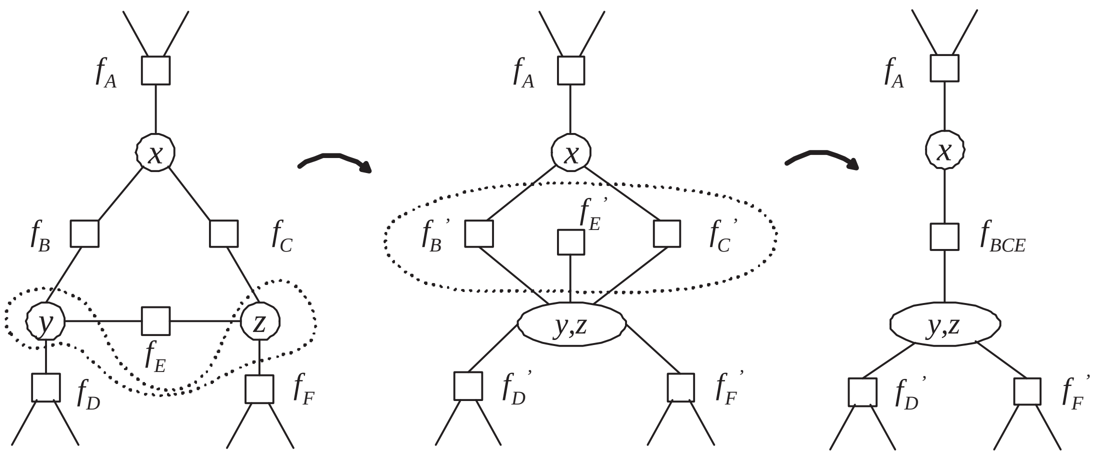
2) $v, w$: local functions $\quad \rightarrow \quad (v, w) =$ product of the local functions.

   ⇒ a substantial cost increase in computational complexity of the SPA

But, clustering functions does not increase the complexity of the variables.

Example: $g(\cdots, x, y, z, \cdots)$

$$
\begin{aligned}
&= \cdots f_A(\cdots, x) f_B(x, y) f_C(x, z) f_D(\cdots, y) f_E(y, z) f_F(z, \cdots) \\
&= \cdots f_A(\cdots, x) f_B'(x, y, z) f_C'(x, y, z) f_D'(\cdots, y, z) f_E'(y, z) f_F'(y, z, \cdots) \\
&= \cdots f_A(\cdots, x) f_{BCE}(x, y, z) f_D'(\cdots, y, z) f_F'(y, z, \cdots)
\end{aligned}
$$

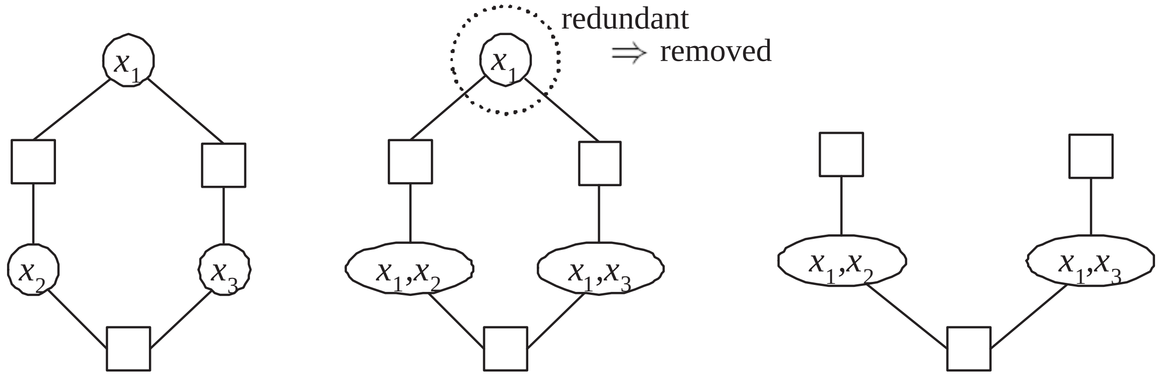


- **Stretching Variable Nodes**

$x$: a variable node in a factor graph $F$.

$n_2(x) \triangleq$ set of nodes that can be reached from $x$ by a path of length two in $F$.

*Stretching transformation*: $y \longrightarrow (x, y) \qquad \forall\ y \in n_2(x)$




$x_1$ is streched to $x_2$ & $x_3$ "a cycle is broken"

- **Spanning Trees**

A *spanning tree* $T$ for a connected graph $G$ is a connected cycle-free subgraph of $G$ having the same vertex set as $G$.

- **Fast Fourier Transform (FFT)**

  – For a $N$-tuple vector $\mathbf{w} = (w_0, w_1, \cdots, w_{N-1}) \in \mathbb{C}^N$, the *Discrete Fourier Transform (DFT)* $\mathbf{W} = (W_0, \cdots, W_{N-1})$ of $\mathbf{w}$ is given by

$$W_k = \sum_{n=0}^{N-1} w_n \, \Omega^{-nk}$$

  where $\Omega = e^{j2\pi/N}$ and $j = \sqrt{-1}$.

  – *$N = 8$ case*

  1) Indexing

$$\begin{aligned} n &= 4x_2 + 2x_1 + x_0 \leftrightarrow (x_0, x_1, x_2) \\ k &= 4y_2 + 2y_1 + y_0 \leftrightarrow (y_0, y_1, y_2) \end{aligned}$$

  2) Let $g(x_0, x_1, x_2, y_0, y_1, y_2) \triangleq w_n \, \Omega^{-nk}$. Then $g$ can be expressed as

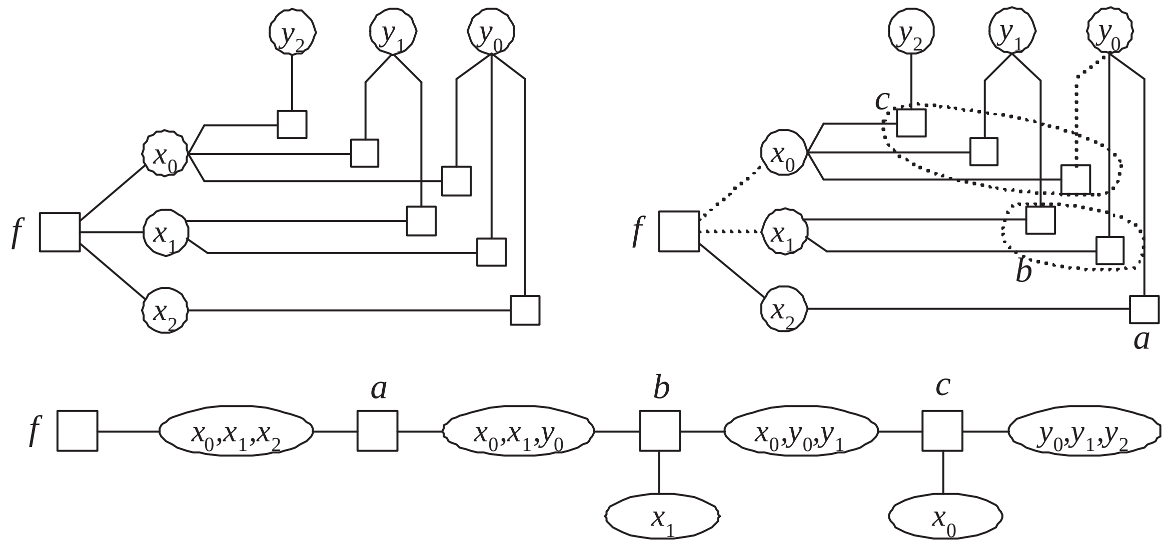
$$\begin{aligned} g(x_0, x_1, x_2, y_0, y_1, y_2) &= w_{4x_2+2x_1+x_0} \Omega^{-(4x_2+2x_1+x_0)(4y_2+2y_1+y_0)} \\ &= f(x_0, x_1, x_2)(-1)^{x_2 y_0}(-1)^{x_1 y_1}(-1)^{x_0 y_2} \\ &\quad \cdot j^{-x_0 y_1} j^{-x_1 y_0} \, \Omega^{-x_0 y_0} \end{aligned}$$

  where $f(x_0, x_1, x_2) = w_{4x_2+2x_1+x_0}$.

  3) Therefore, $W_k$ can be computed as

$$W_k \triangleq W_{4y_2+2y_1+y_0} = \sum_{x_0} \sum_{x_1} \sum_{x_2} g(x_0, x_1, x_2, y_0, y_1, y_2)$$

  That is, the DFT can be viewed as a marginal function, much like a probability mass function.




$$a(x_2, y_0) = (-1)^{x_2 y_0}$$
$$b(x_1, y_0, y_1) = (-1)^{x_1 y_1}\, j^{-x_1 y_0}$$
$$c(x_0, y_0, y_1, y_2) = (-1)^{x_0 y_2}\, j^{-\,x_0 y_1}\ \Omega^{-x_0 y_0}$$

Figure 1: A factor graph for DFT

Figure 2: A particular spanning tree

Figure 3: Spanning tree after clustering and stretching transformation

Conclusion: An FFT can be regraded as as an instance of the sum-product algorithm.

**Note:**

1) Various fast transform algorithms may be developed using a graph-based approach.

2) The SPA can be used to compute marginal functions exactly in any spanning tree $T$ of $F$, provided that each variable $x$ is stretched along all variable nodes appearing in each path from $x$ to a local function having $x$ as an argument.