LDPC Codes

Alexios Balatsoukas-Stimming and Athanasios P. Liavas

Technical University of Crete
Dept. of Electronic and Computer Engineering
Telecommunications Laboratory

December 16, 2011

Intracom Telecom, Peania

Outline

LDPC Codes

When?

- Discovered in 1962 by Gallager.
- Rediscovered by MacKay and others in late 1990s.

Why?

- Capacity approaching on many channels.
- Linear decoding complexity using Belief Propagation.
- Linear complexity encoding, under certain conditions.

LDPC Codes

- We can define a linear block code $\mathcal{C}$ as the nullspace of its $m \times n$ parity check matrix:

$$\mathcal{C} = \{\mathbf{c} \in \{0,1\}^n : \mathbf{Hc} = \mathbf{0}\}.$$

- We can also define a linear block code $\mathcal{C}$ through its $k \times n$ generator matrix as follows:

$$\mathcal{C} = \left\{\mathbf{c} = \mathbf{G}^{\mathsf{T}}\mathbf{u} : \mathbf{u} \in \{0,1\}^k\right\}.$$

- LDPC codes are linear block codes with a sparse parity check matrix.

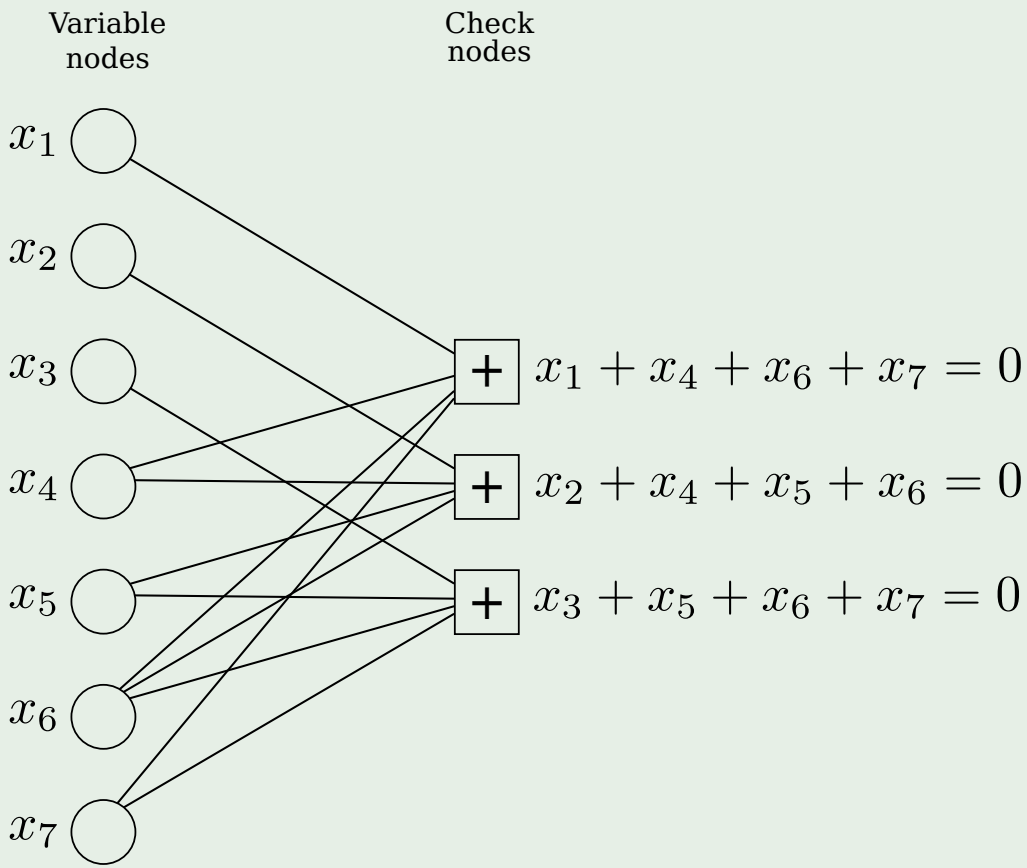
Factor Graph Representation

- An LDPC code can be associated with a bipartite graph, also called a **factor graph** or **Tanner graph**.
- **Variable nodes** correspond to codeword bits and **check nodes** correspond to the parity-check equations enforced by the code.
- Variable node j is connected with check node i if and only if $\mathbf{H}_{ij} = 1$.

Factor Graph Representation and Degree Distributions

Example ((7, 4) Hamming Code)

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



Degree Distributions

- The **variable node degree distribution** from an edge perspective is denoted as:

$$\lambda(x) = \sum_i \lambda_i x^{i-1}$$

- The **check node degree distribution** from an edge perspective is denoted as:

$$\rho(x) = \sum_i \rho_i x^{i-1}$$

- The λ_i's and ρ_i's determine the ratio of edges that are connected to variable and check nodes of degree i, respectively.
- They also determine the code's **design rate**:

$$R = 1 - \frac{\sum_i \rho_i / i}{\sum_i \lambda_i / i},$$

as well as its average performance, as we will see.

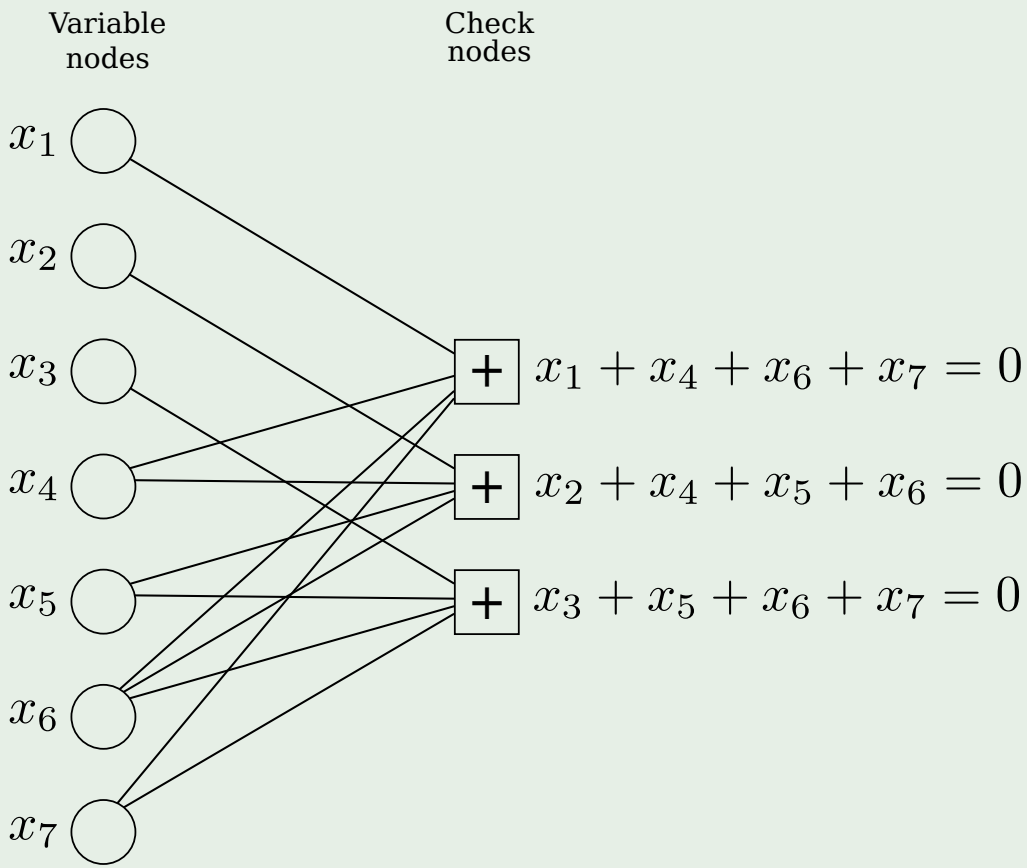
Factor Graph Representation and Degree Distributions

Example ((7, 4) Hamming Code)

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

$$\lambda(x) = \frac{3}{12} + \frac{6}{12}x + \frac{3}{12}x^2$$

$$\rho(x) = x^3$$



1. Introduction to LDPC Codes

2. Iterative Decoding of LDPC Codes
   - Belief Propagation
   - Density Evolution
   - Gaussian Approximation
   - EXIT Charts

3. Efficient Encoding of LDPC and QC-LDPC Codes

4. Construction of QC-LDPC Codes
   - Deterministic
   - Random

5. RA and Structured RA Codes

Sum-Product Marginalization

- Consider a function which can be factorized as follows

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

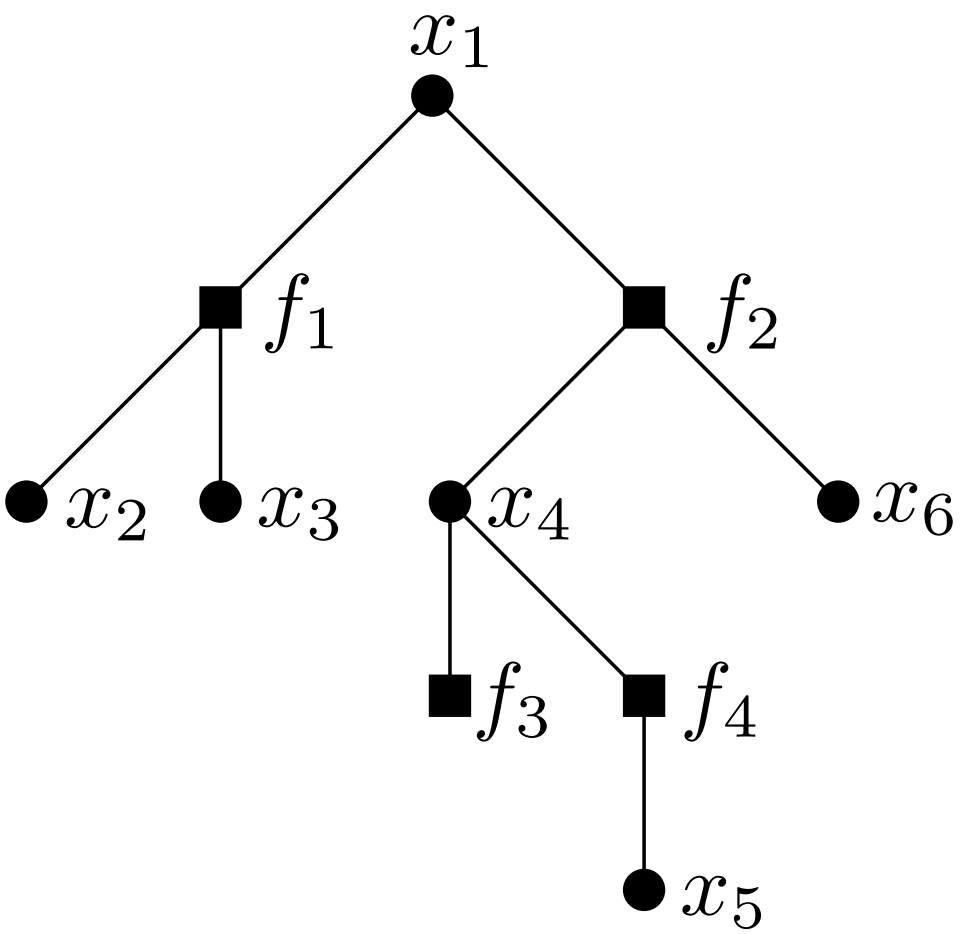
- Brute force calculation of marginal $f(x_1) = \sum_{\sim x_1} f(x_1, \ldots, x_6)$ requires $\mathcal{O}(|\mathcal{X}|^6)$ operations.
- Using the factorization and the distributive law, we can write

$$f(x_1) = \left[\sum_{\sim x_1} f_1(x_1, x_2, x_3) \right] \underbrace{\left[\sum_{\sim x_1} \overbrace{f_2(x_1, x_4, x_6)}^{\text{Kernel}} f_3(x_4) f_4(x_4, x_5) \right]}_{\text{Can be further expanded.}}$$

reducing complexity to $\mathcal{O}(|\mathcal{X}|^4)$.

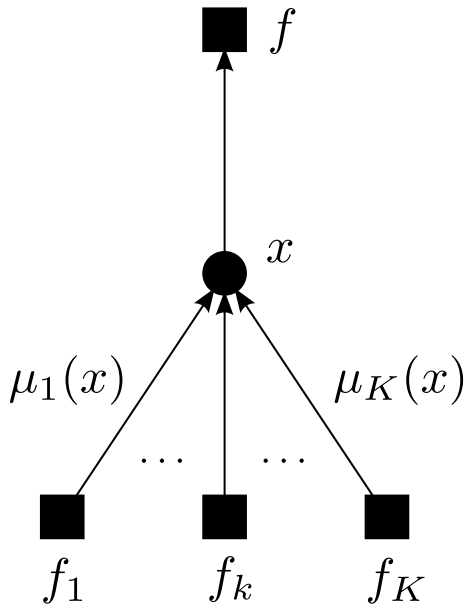
Sum-Product Marginalization

- By further expanding $f(x_1)$ we get the following factor graph:



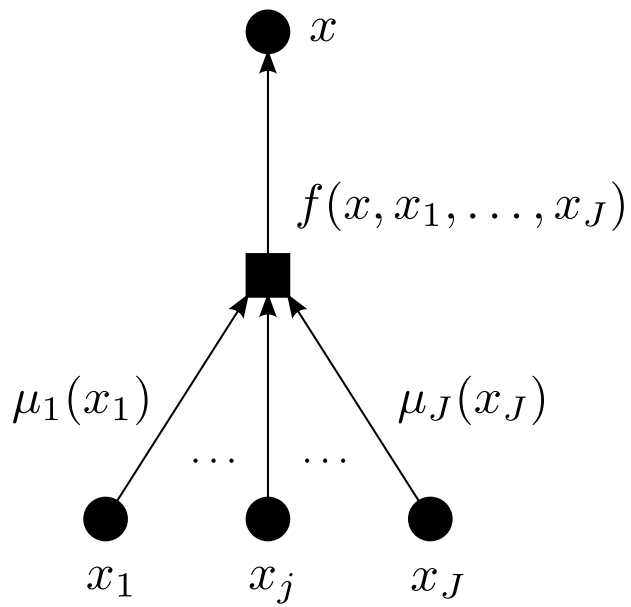
Sum-Product Marginalization Rules

$\mu(x) = \prod_{k=1}^{K} \mu_k(x)$



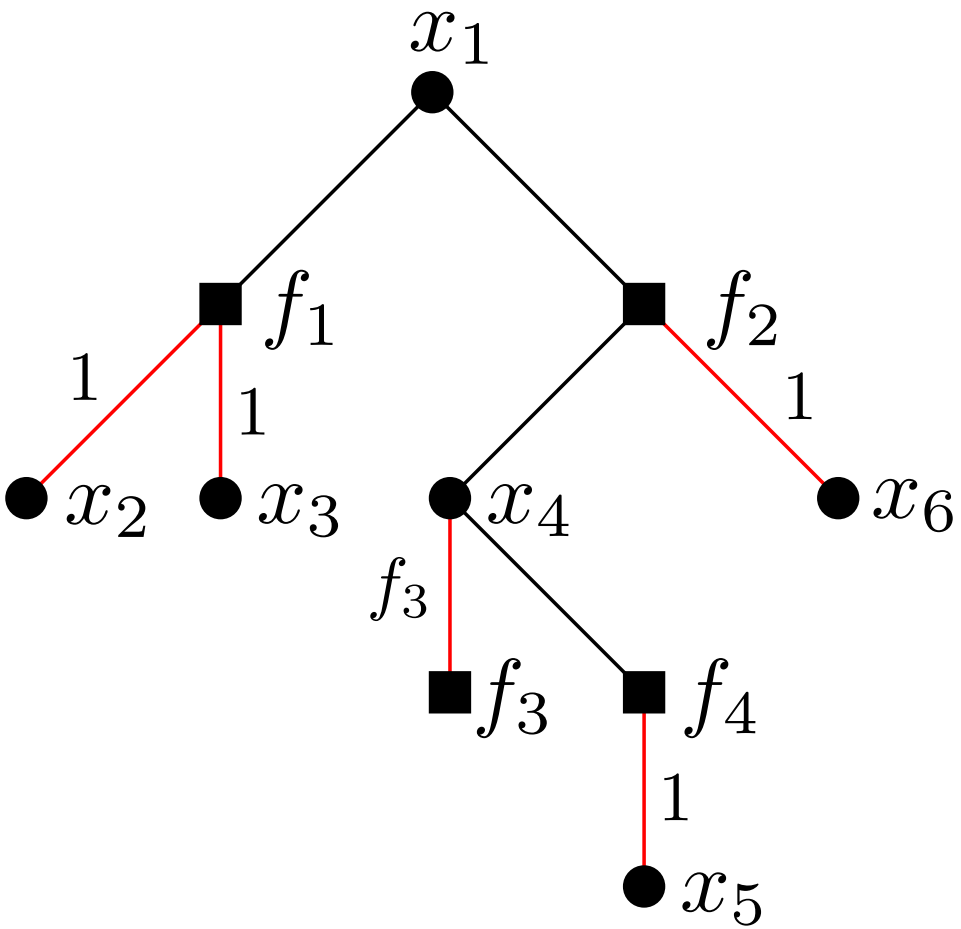
(a) Variable node rules.

$\mu(x) = \sum_{\sim x} f(x, x_1, \ldots, x_J) \prod_{j=1}^{J} \mu_j(x_j)$

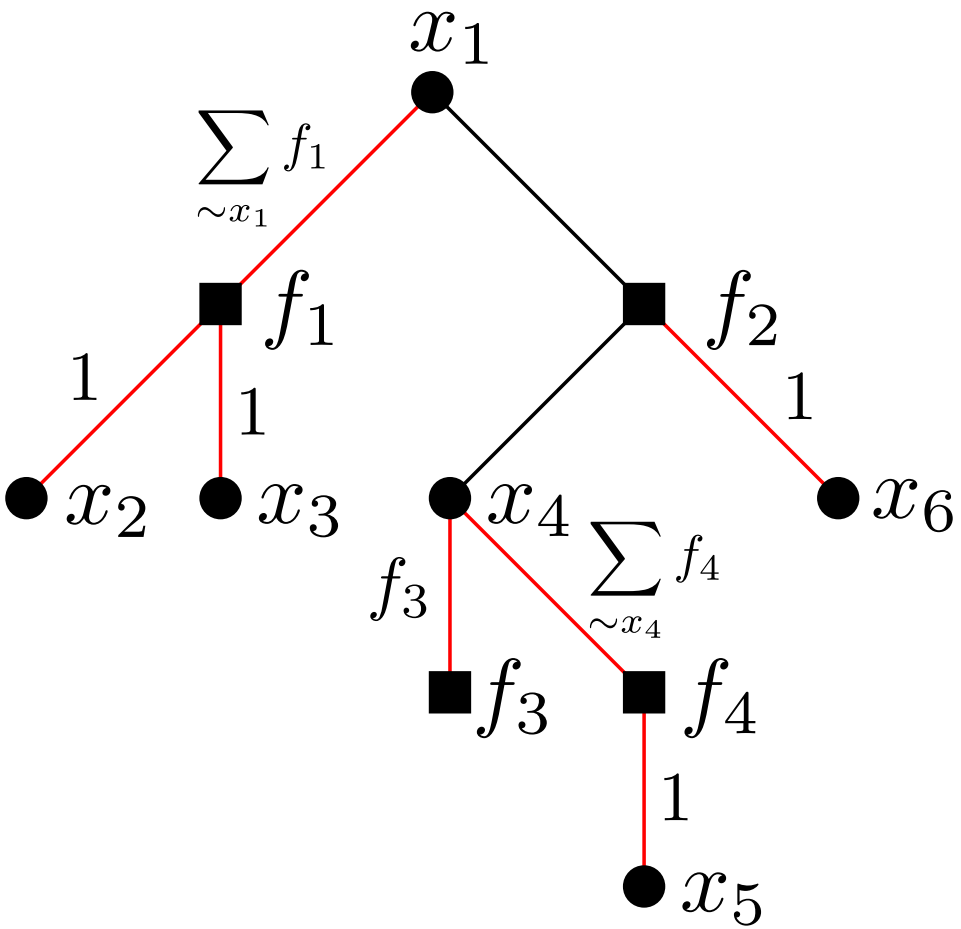


(b) Function node rules.

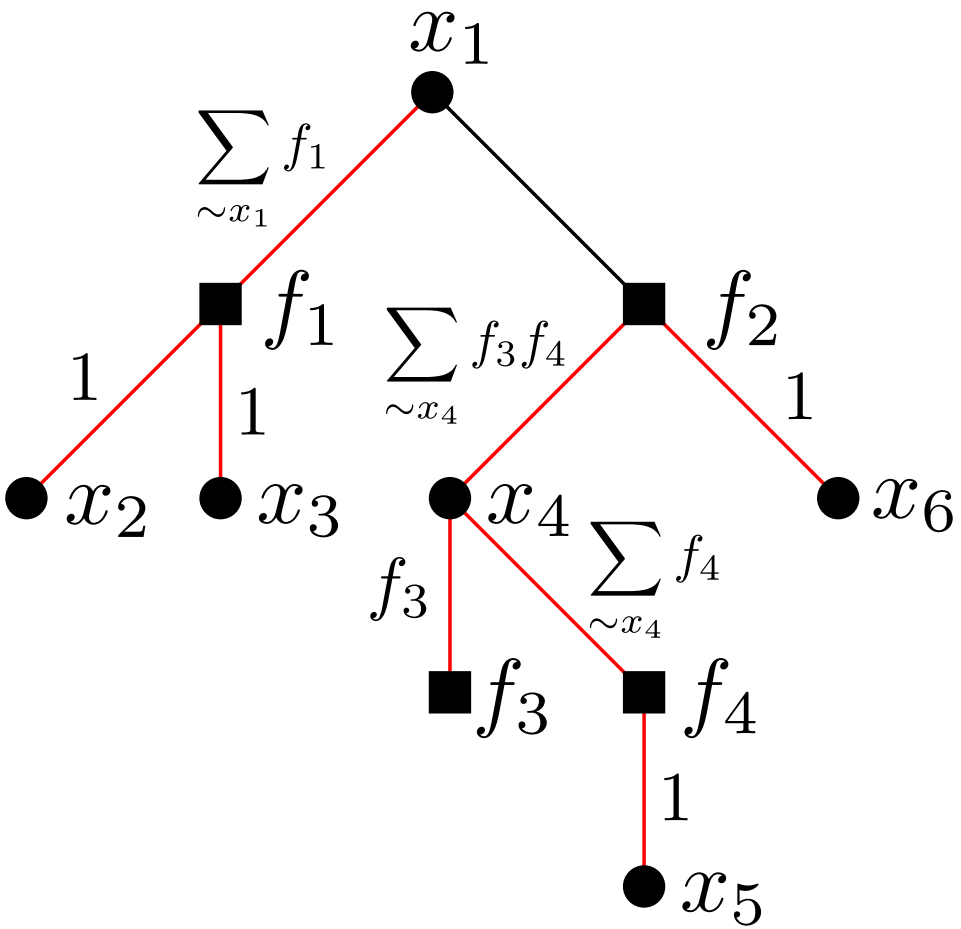
Sum-Product Marginalization - Step 1



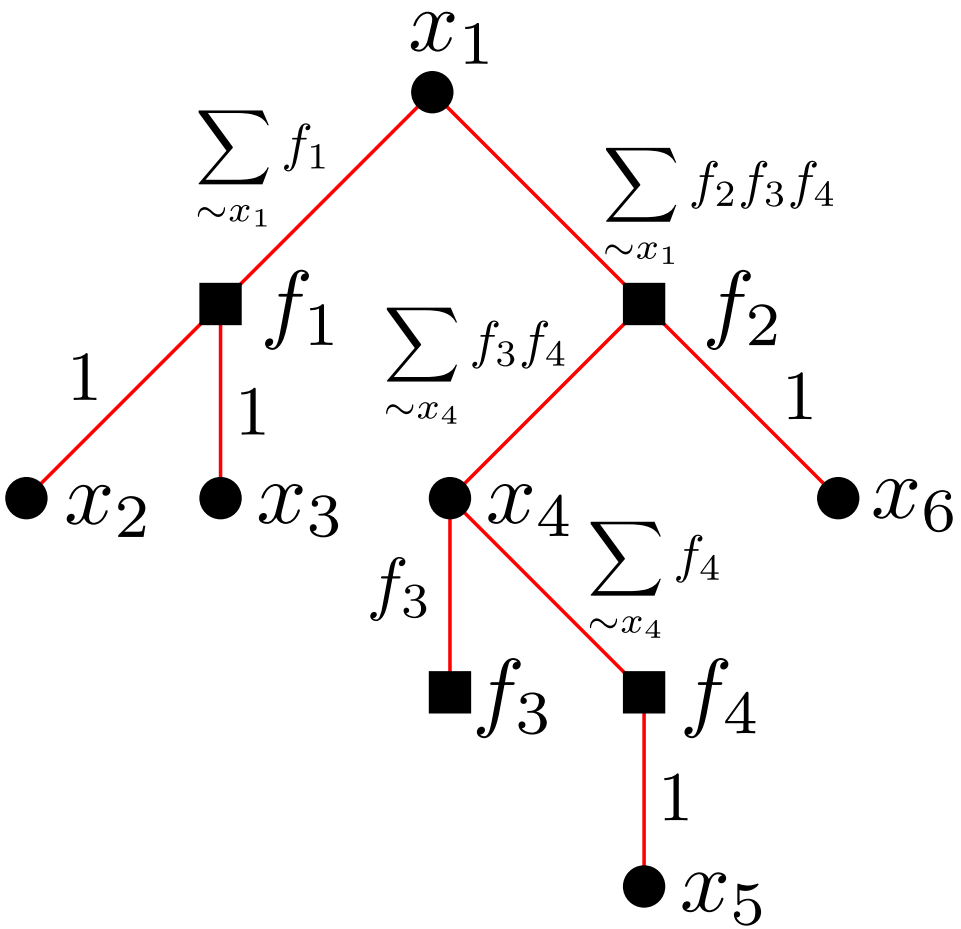
Sum-Product Marginalization - Step 2



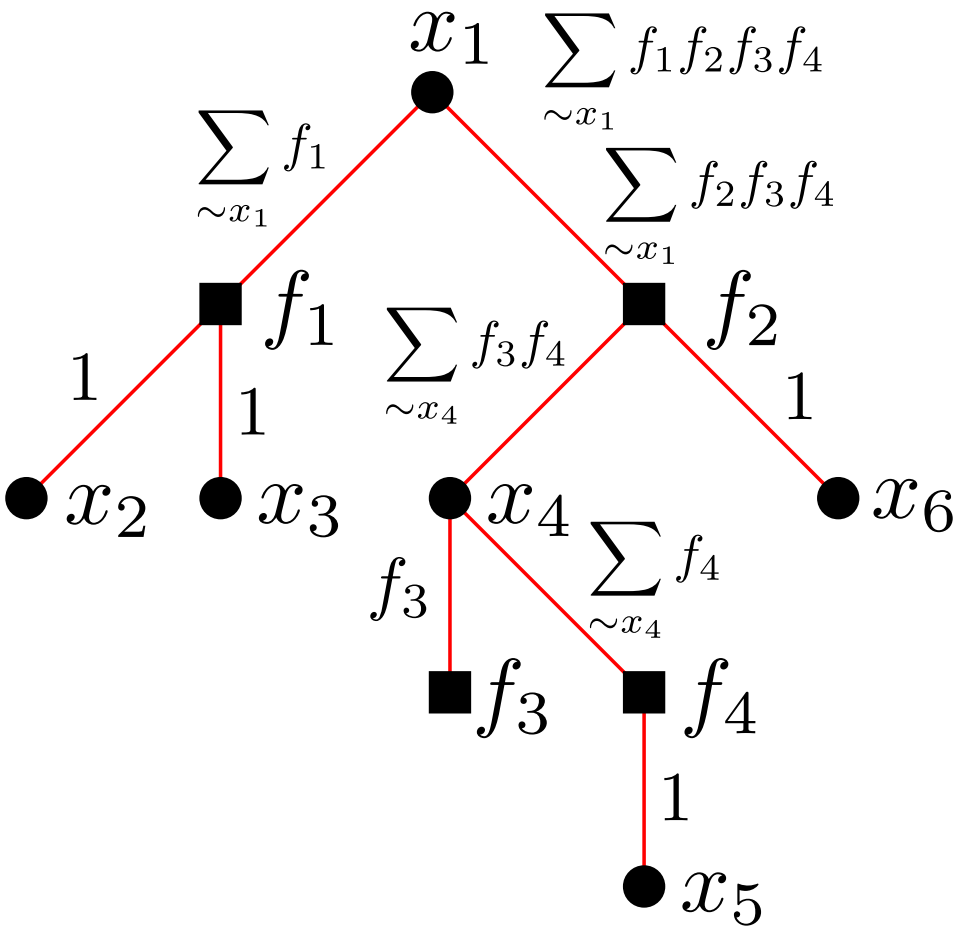
Sum-Product Marginalization - Step 3



Sum-Product Marginalization - Step 4



Sum-Product Marginalization - Final Step



Sum-Product Decoding

- Transmission of a codeword $\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \in \mathcal{C}$ over an AWGN channel

$$y_i = x_i + w_i, \;\; w_i \sim \mathcal{N}(0, \sigma_w^2), \;\; i = 1, 2, \ldots, n.$$

- The MAP decoding rule for x_i reads

$$\begin{aligned}
\hat{x}_i &= \arg \max_{x_i \in \{\pm 1\}} p(x_i | \mathbf{y}) \\
&= \arg \max_{x_i \in \{\pm 1\}} \sum_{\sim x_i} p(\mathbf{x} | \mathbf{y}) \\
&= \cdots \text{(Bayes' rule, conditional independence)} \\
&= \arg \max_{x_i \in \{\pm 1\}} \underbrace{\sum_{\sim x_i} \left(\prod_{j=1}^{n} p(y_j | x_j) \right) \mathbf{1}_{[\mathbf{x} \in \mathcal{C}]}}_{\text{Sum-Product form!}}
\end{aligned}$$

Belief Propagation

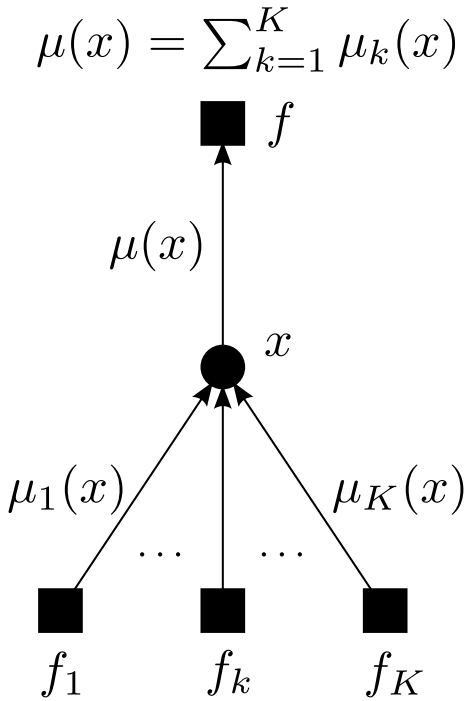
- Log-domain decoding simplifies the sum-product rules and guarantees numerical stability of the calculations.

- LLRs are used as messages, i.e. each variable node aims to calculate

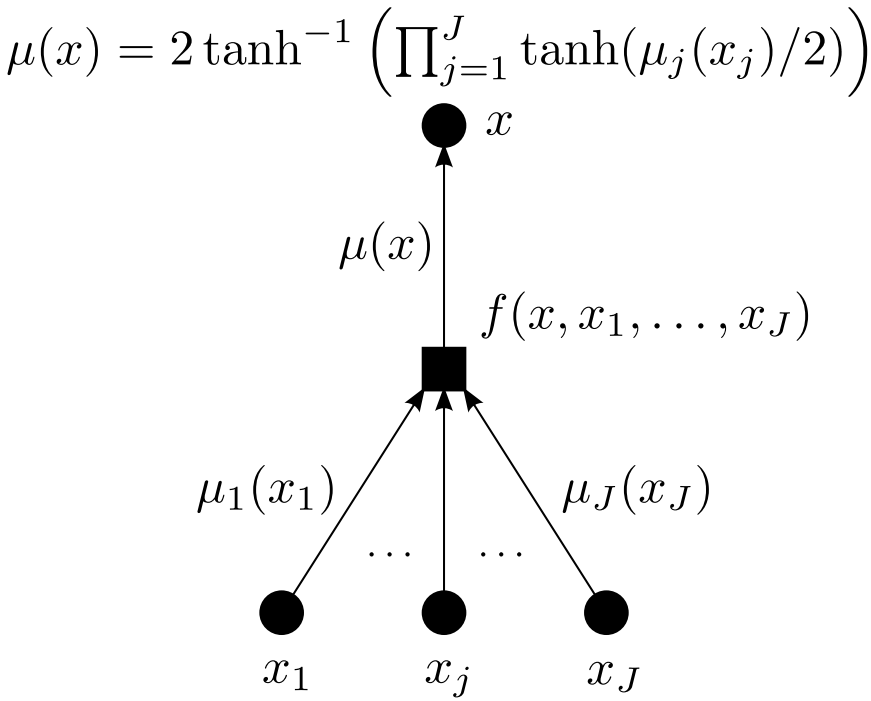
$$L_i = \log\left(\frac{p(x_i = +1|\mathbf{y})}{p(x_i = -1|\mathbf{y})}\right)$$

- If $L_i > 0$, then $\hat{x}_i = +1$, if $L_i < 0$ then $\hat{x}_i = -1$.

Belief Propagation

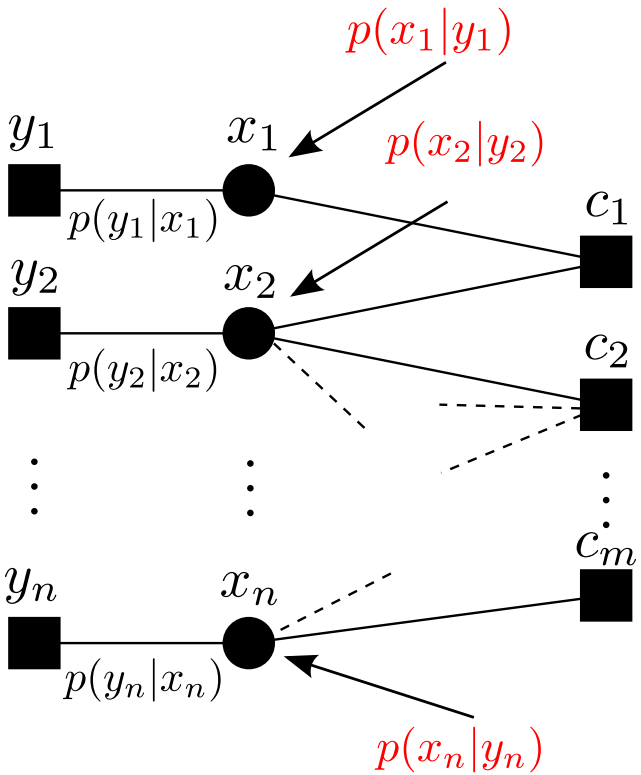


(a) Variable node rules.



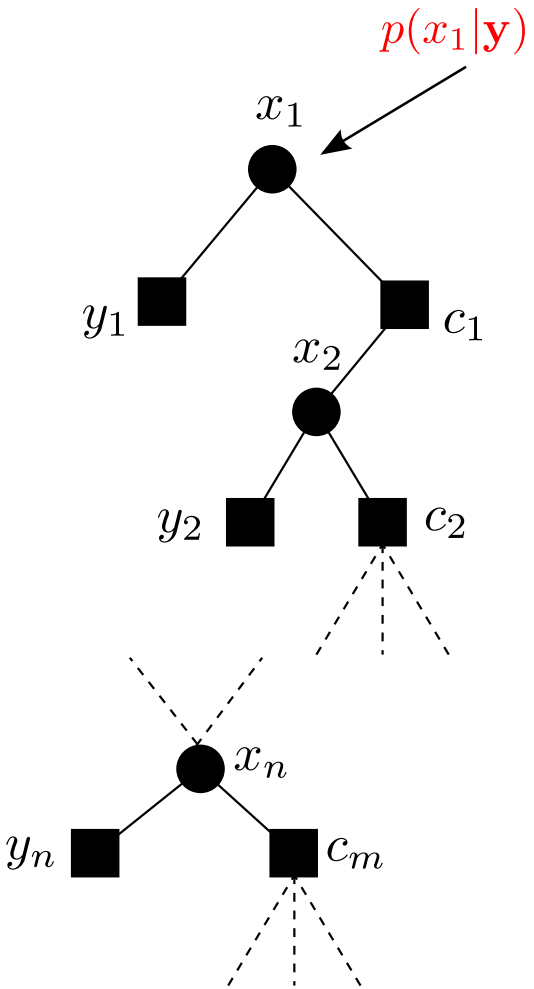
(b) Check node rules.

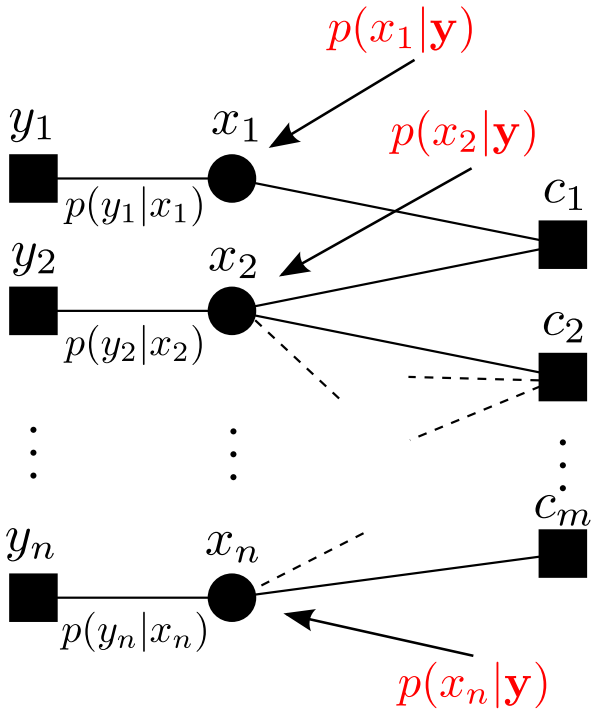
Belief Propagation Decoding - Initialization



- $p(x_i)$ and $p(y_i)$ are known. So, given $p(y_i|x_i)$, we can calculate $p(x_i|y_i)$, i.e. the a posteriori probability of x_i given the channel observation y_i.

Belief Propagation





- If graph is cycle-free, after running BP, we get the marginal $p(x_1|\mathbf{y})$.
- The same holds for all $p(x_i|\mathbf{y})$, $i = 1, 2, \ldots$

Belief Propagation Decoding

- If Tanner graph is cycle-free, we can calculate **all** $p(x_i|\mathbf{y}), i = 1, 2, \ldots,$ **simultaneously**.
- **Bad news about cycle-free codes**: they necessarily contain many low-weight codewords and, hence, have a high probability of error.
- However, BP still performs **very well in practice**, even when cycles are present.
- On graphs with cycles, when $\mathbf{H}\hat{\mathbf{x}} = \mathbf{0}$ or a maximum number of iterations is reached, decoding halts.

Other Iterative Decoding Algorithms

- Wide variety of performance-complexity trade-offs.
- In order of increasing complexity and performance:
  1. Bit-flipping
  2. Majority Logic Decoding
  3. Min-Sum Decoding
  4. Belief Propagation Decoding

Density Evolution

- In the limit of infinite blocklength, the performance of the (λ, ρ)-ensemble of LDPC codes under Belief Propagation decoding can be predicted by **Density Evolution**.

- Due to the concentration theorem, performance of individual codes in the ensemble is close to the ensemble average performance (**exponential convergence** w.r.t. the codeword length).

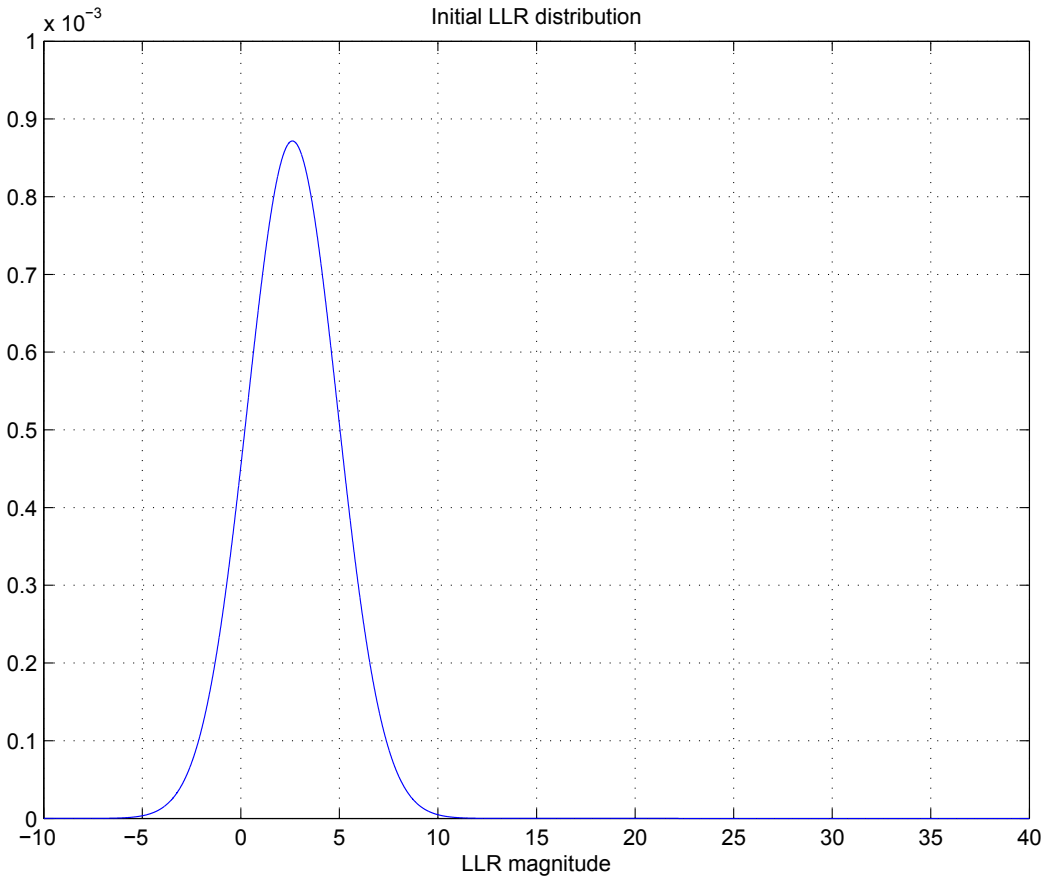
- Density Evolution tracks the evolution of message probability density functions throughout the decoding procedure, under the assumption that the **all-zero codeword is transmitted**.

- For the BI-AWGNC, probability density functions must be tracked since the received LLR messages are continuous random variables (computationally very demanding).

Density Evolution - BI-AWGN Channel

- Initial LLR based only on the channel output is:

$$\mathrm{LLR}(y) = -\frac{2y}{\sigma^2} \sim \mathcal{N}\left(\frac{2}{\sigma^2}, \frac{4}{\sigma^2}\right).$$



Density Evolution - BI-AWGN Channel

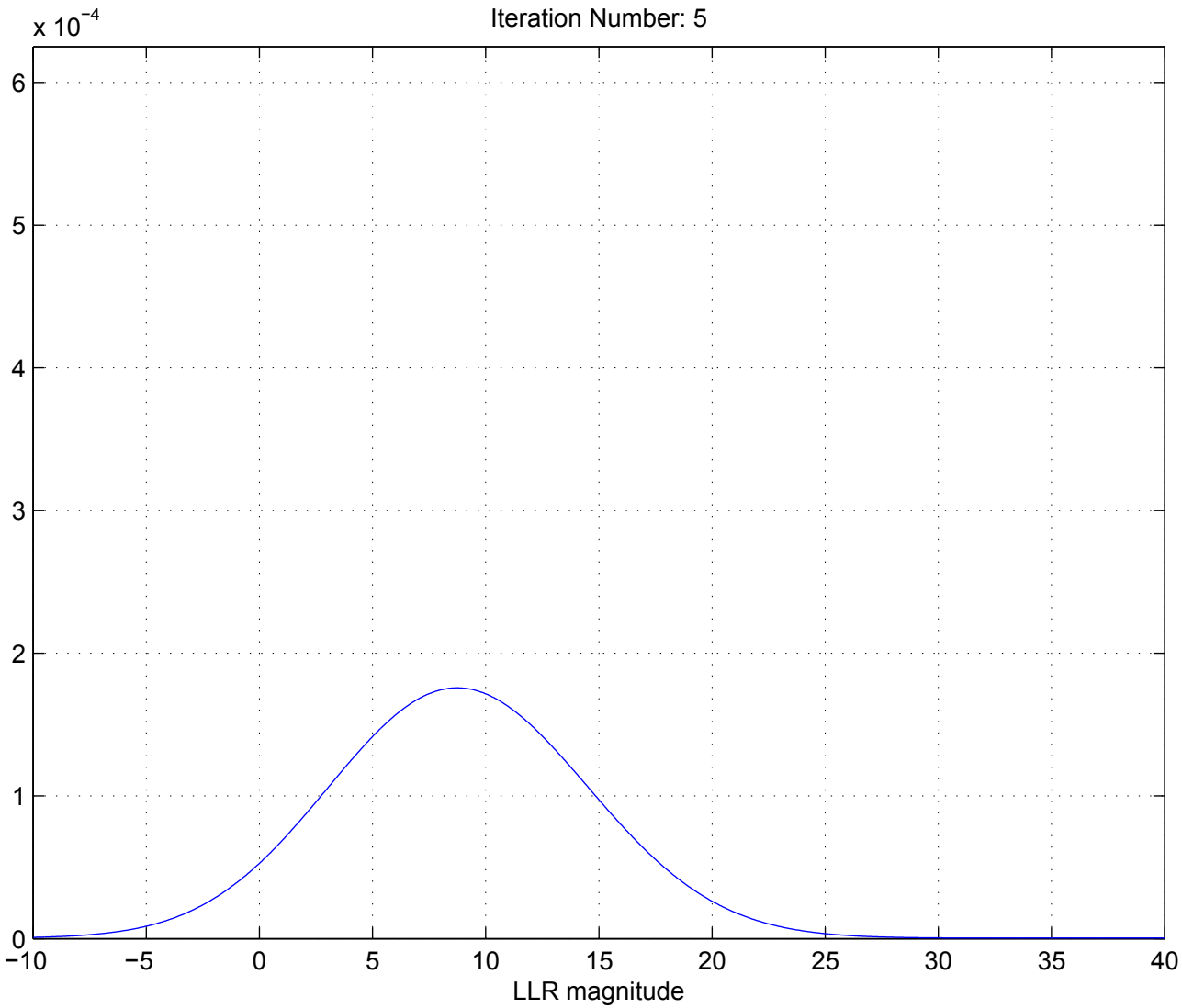


Figure: $(3, 6)$-regular code, $\sigma^2 = 0.7692$.

Density Evolution - BI-AWGN Channel

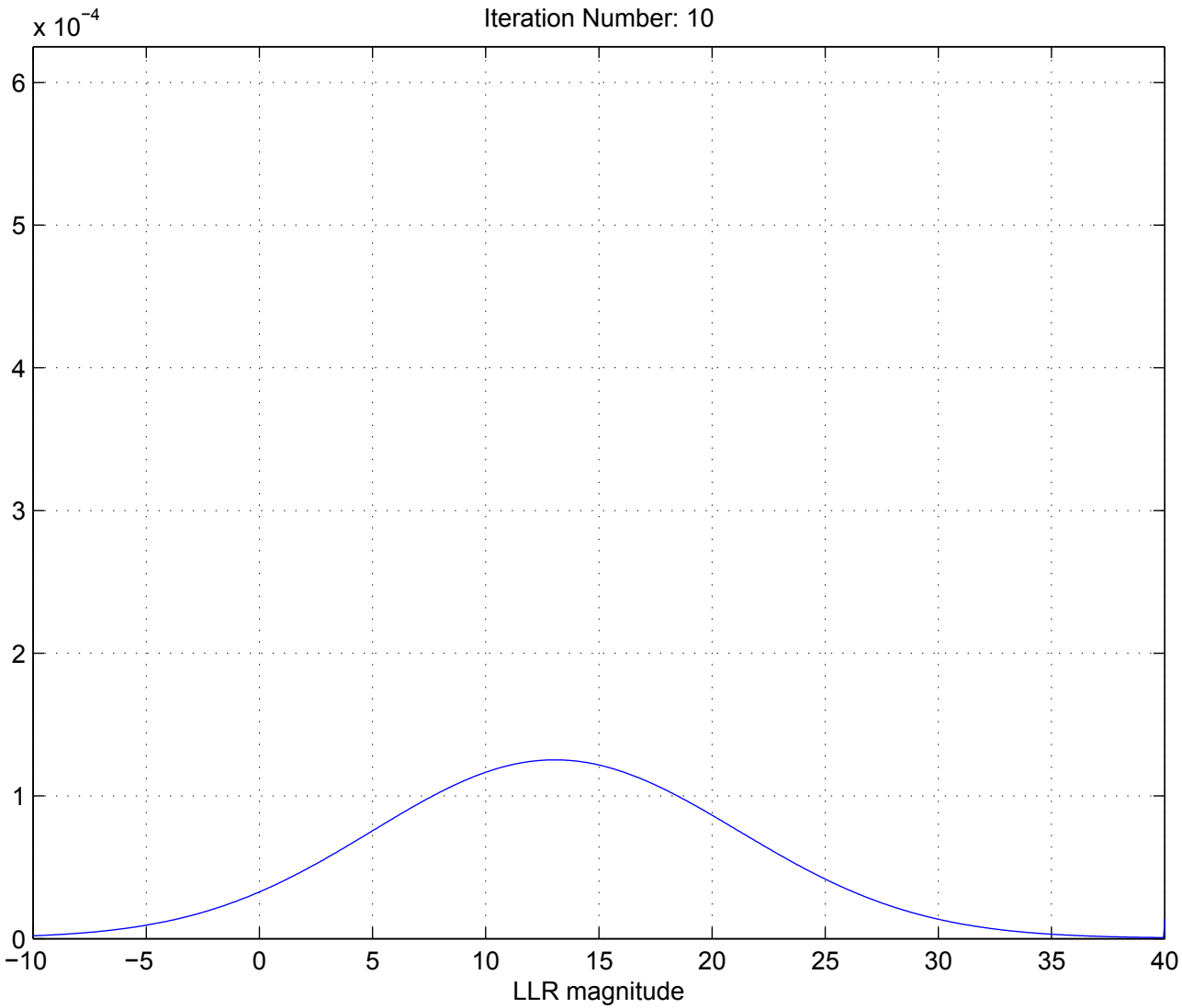


Figure: $(3, 6)$-regular code, $\sigma^2 = 0.7692$.

Density Evolution - BI-AWGN Channel

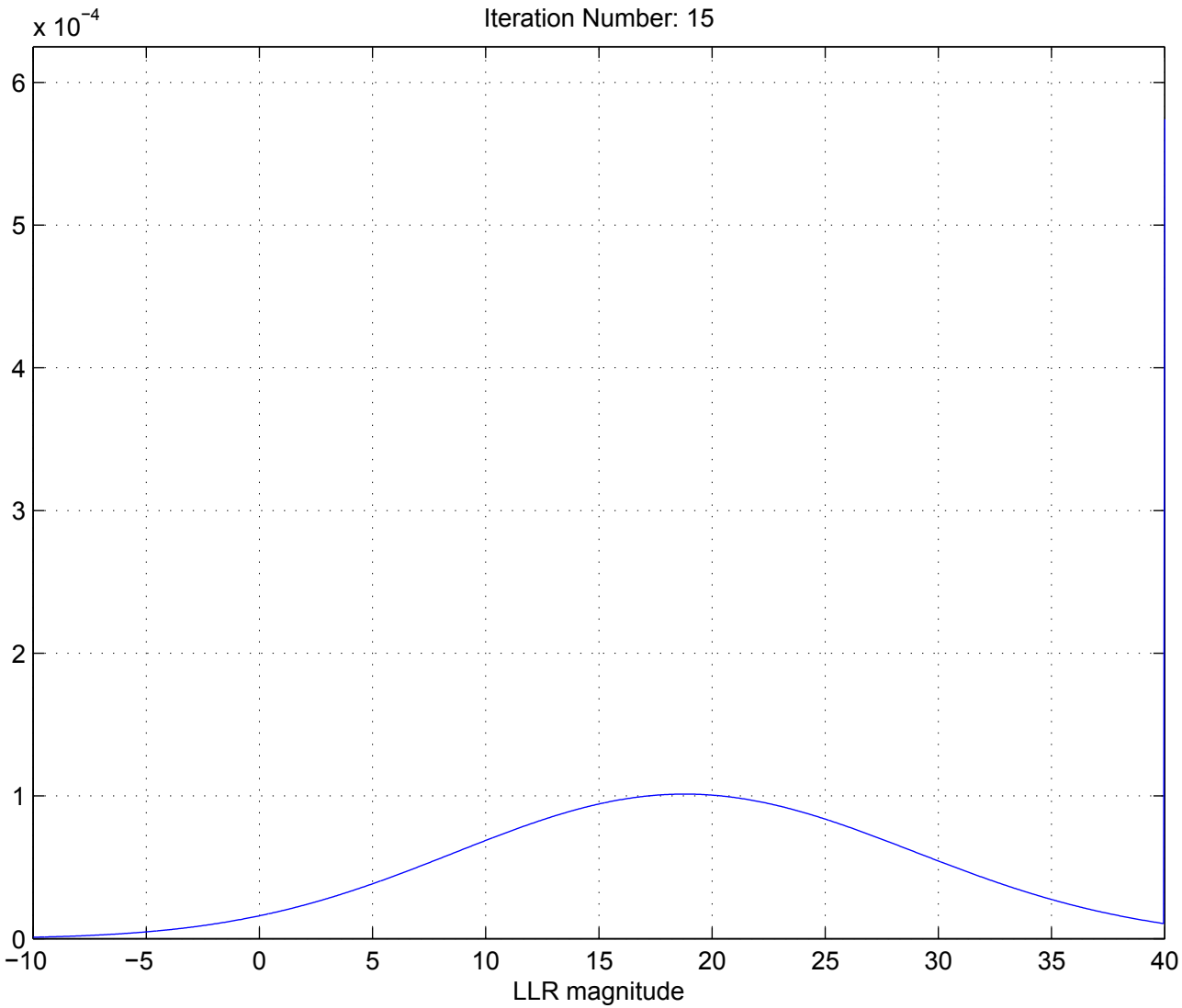
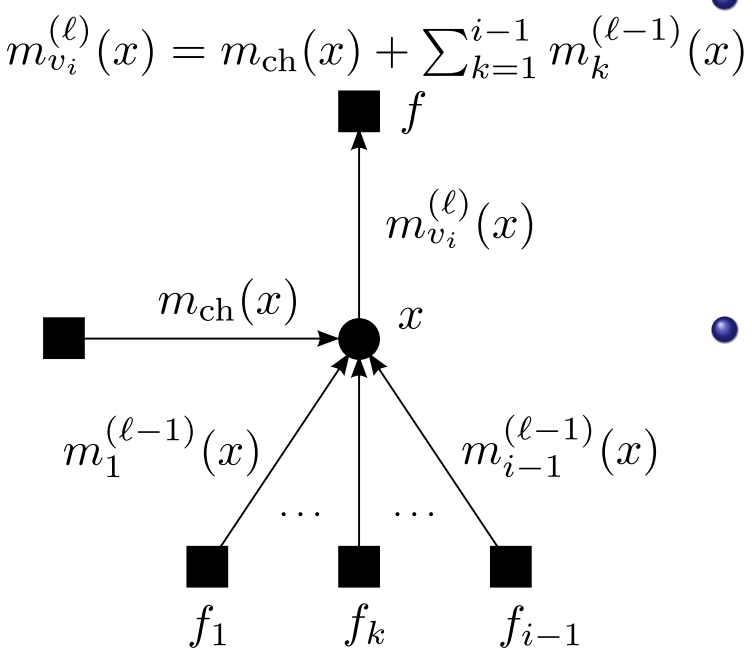


Figure: $(3, 6)$-regular code, $\sigma^2 = 0.7692$.

Density Evolution - Gaussian Approximation

- **Gaussian Approximation (GA)**: we assume the messages are symmetric Gaussian random variables, i.e. variance σ^2 and mean $\sigma^2/2$ (Central Limit Theorem).
- We must track **only the mean**.



- **Variable node** of degree i at iteration ℓ:

$$m_{v_i}^{(\ell)} = m_{\text{ch}} + \sum_k m_k^{(\ell-1)}$$

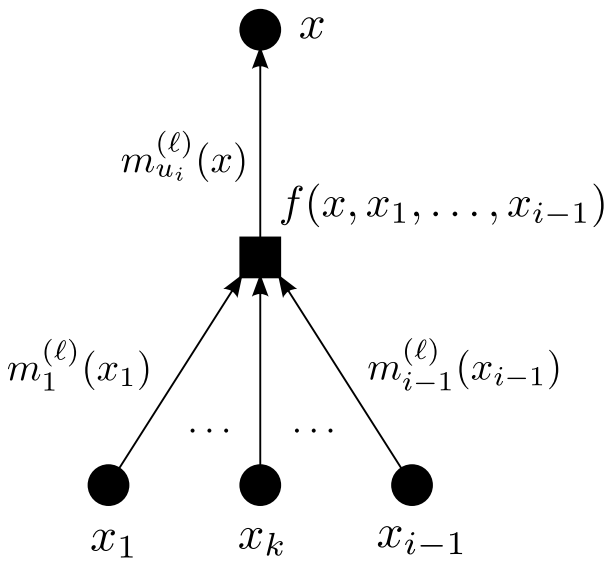
- Average over $\lambda(x)$:

$$m_v^{(\ell)} = \sum_i \lambda_i m_{v_i}^{(\ell)}.$$

Density Evolution - Gaussian Approximation

- **Check node** of degree i at iteration ℓ:

$$m_{u,i}^{(\ell)} = \phi^{-1}\left(1 - \left[1 - \sum_i \lambda_i \phi\left(m_{v_i}^{(\ell)}\right)\right]^{j-1}\right)$$



- Average over $\rho(x)$:

$$m_u^{(\ell)} = \sum_i \rho_i m_{u,i}^{(\ell)}.$$

Gaussian Approximation - Convergence

- Under the assumption that the message densities are symmetric Gaussian and that the all-zero codeword was transmitted, the average probability of bit error is:

$$P_e^{(\ell)} = Q\left(\frac{\sqrt{m_v^{(\ell)}}}{\sqrt{2}}\right) \overset{m_v^{(\ell)} \to \infty}{\longrightarrow} 0.$$

- If

$$\boxed{m_v^{(\ell)} > m_v^{(\ell-1)},\ \forall \ell = 1, 2, \ldots}$$

then, the BP decoder converges to a **vanishingly small probability of error**.

EXIT Charts

- **Message**: extrinsic LLR of codebit i:

$$L_i = \log \frac{p(x_i = +1|\mathbf{y}_{\sim i})}{p(x_i = -1|\mathbf{y}_{\sim i})}.$$

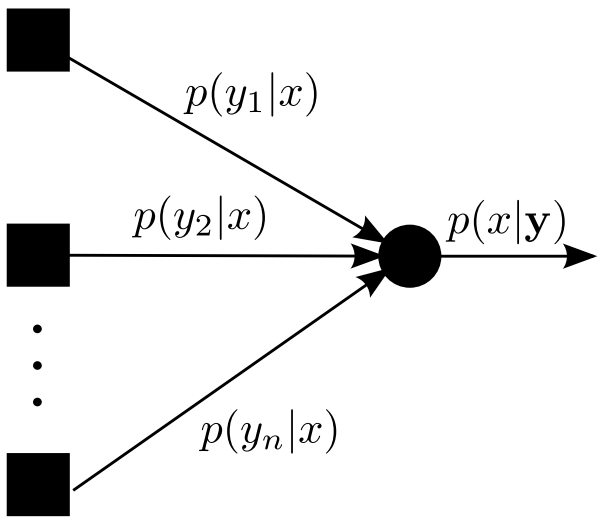
- **Message Density**: density of extrinsic LLR assuming that the all-zero codeword was transmitted.

- BP at a variable node acts as a decoder of a **repetition code**.
- BP at a check node acts as a decoder of a **parity-check code**.

EXIT Charts - Repetition Code

- Consider an $[n, 1, n]$ **repetition code**. Transmit $X = \pm 1$ over AWGNC:

$$y_i = x + w_i, \; w_i \sim \mathcal{N}(0, \sigma_w^2), \; i = 1, \ldots, n.$$



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

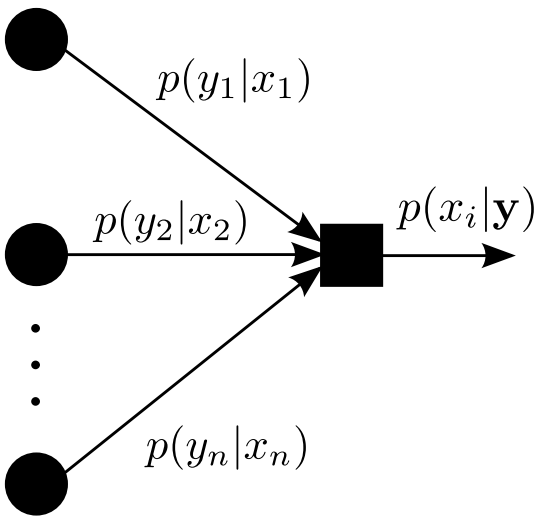
$$\propto p(\mathbf{y}|x) = \underbrace{\prod_{i=1}^{n} p(y_i|x)}_{\text{variable node rule}}.$$

- Before decoding: $p(x|y_i) = a$ and $I(X; Y_i) = 1 - H(a)$.
- After decoding: $I(X; \mathbf{Y}) = 1 - H(a^{*n})$, where $*n$ denotes n-fold convolution.

EXIT Charts - Parity-Check Code

- Consider an $[n, n-1, 2]$ **parity-check code**. Transmit codeword $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$, $x_i = \pm 1$, over AWGNC:

$$y_i = x_i + w_i, \;\; w_i \sim \mathcal{N}(0, \sigma_w^2), \;\; i = 1, \dots, n.$$

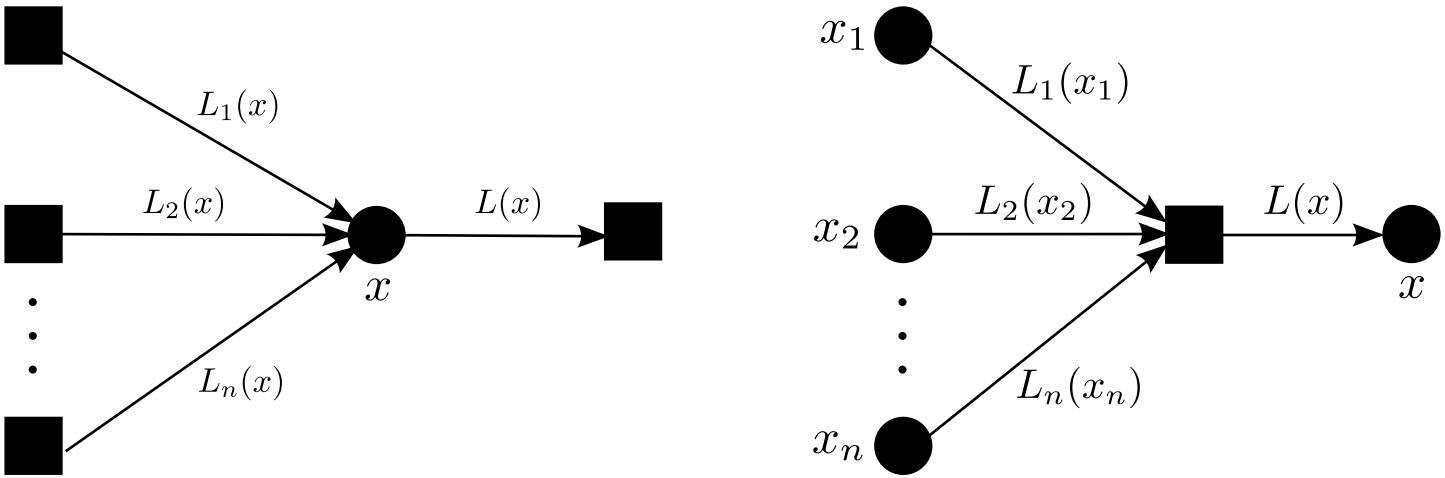


$$p(x_i|\mathbf{y}) = \frac{p(x_i, \mathbf{y})}{p(\mathbf{y})}$$
$$\propto \sum_{\sim x_i} p(\mathbf{x}) p(\mathbf{y}|\mathbf{x})$$
$$= \underbrace{\sum_{\sim x_i} \left(\mathbf{1}_{[x_1 \cdot x_2 \cdot \ldots \cdot x_n = 1]} \prod_{i=1}^{n} p(y_i|x_i) \right)}_{\text{check node rule}}$$

- Before decoding: $p(x_i|y_i) = b$ and $I(X_i; Y_i) = 1 - H(b)$.
- After decoding: $I(X_i; \mathbf{Y}) = 1 - H(b^{\circledast n})$, where $\circledast n$ denotes n-fold convolution in the G-domain.

EXIT Charts

- Now, if we consider the repetition and parity-check codes as being part of a large system that performs message passing decoding, we have:



- For the variable node, if $I(X; L_i(X)) = 1 - H(a)$, then:

$$I(X; L(X)) = 1 - H(a^{*n}).$$

- For the check codes, if the $I(X_i; L_i(X_i)) = 1 - H(b)$, then:

$$I(X; L(X)) = 1 - H(b^{\circledast n}).$$

EXIT Charts

- We define:

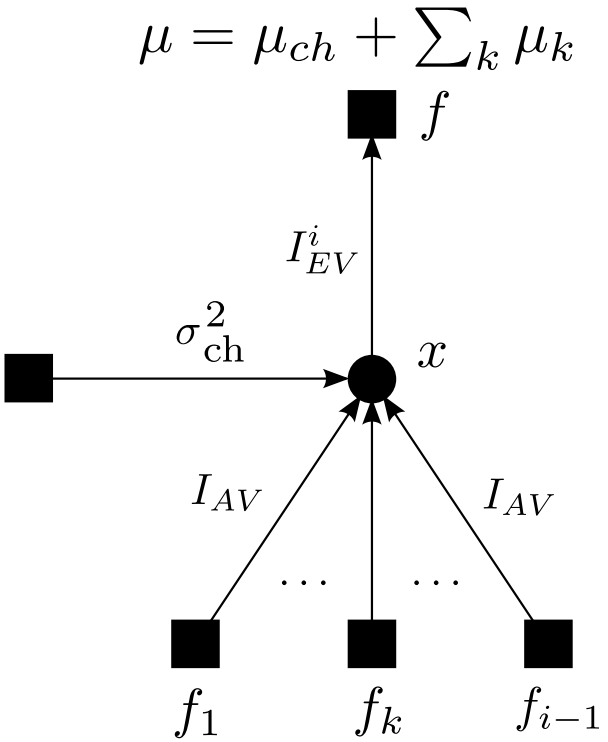
$$J(\sigma) = 1 - \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} e^{-\frac{(x-\sigma^2/2)^2}{2\sigma^2}} \log_2(1 + e^{-x}) dx.$$

- $J(\sigma)$ is the mutual information between a codeword bit and the corresponding message, assuming that the message is a **symmetric Gaussian random variable** with standard deviation σ.

- $J^{-1}(I)$ is the standard deviation of the **symmetric Gaussian distributed** message which has mutual information I with a codeword bit.

EXIT Charts - Variable Nodes

- I_{EV} (resp. I_{AV}) is the average mutual information between the outgoing (resp. incoming) messages of variable nodes and the codeword bits.



- The EXIT chart I_{EV} describing the **variable node** function of degree i:

$$I^i_{EV}(I_{AV}, \sigma^2_{ch}) = J\left(\sqrt{(i-1)J^{-1}(I_{AV})^2 + \sigma^2_{ch}}\right)$$

- Average over $\lambda(x)$:

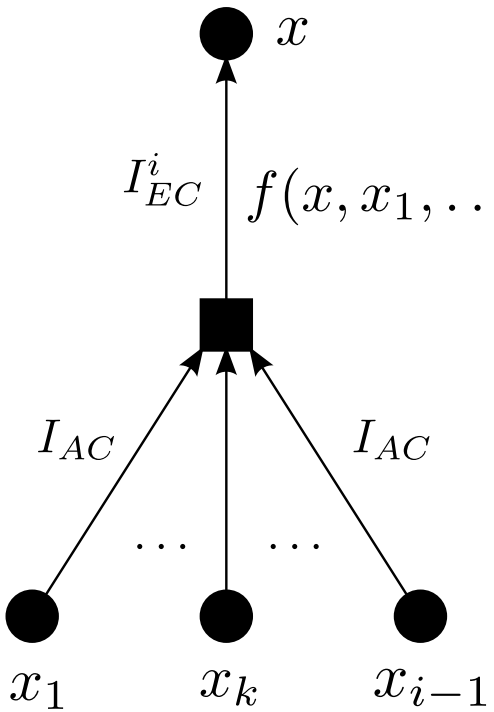
$$I_{EV}(I_{AV}, \sigma^2_{ch}) = \sum_i \lambda_i I^i_{EV}(I_{AV}, \sigma^2_{ch}),$$

with $\sigma^2_{ch} = \frac{4}{\sigma^2_w}$, σ^2_w is the noise variance.

EXIT Charts - Check Nodes

- I_{EC} (resp. I_{AC}) is the average mutual information between the outgoing (resp. incoming) messages of check nodes and the codeword bits.

$\mu = 2\tanh^{-1}\left(\prod_{k=1}^{J}\tanh(\mu_k/2)\right)$



- The EXIT chart I_{EC} describing the **check node** function of degree i:

$$I_{EC}^{i}(I_{AC}) \approx 1 - J\left(\sqrt{(i-1)}J^{-1}(1-I_{AC})\right)$$

- Average over $\rho(x)$:

$$I_{EC}(I_{AC}) \approx \sum_{i} \rho_i I_{EC}^{i}(I_{AC}).$$

EXIT Charts - Code Optimization

- For every possible incoming average mutual information at variable nodes, we want the outgoing average mutual information to be larger:

$$\boxed{I_{EC}\left(I_{EV}\left(I_{AV}\right)\right) > I_{AV}}$$

- If the EXIT chart of the variable nodes lies above the inverse of the EXIT chart for the check nodes, i.e.

$$\boxed{I_{EV}(I_{AV}) > I_{EC}^{-1}(I_{AV})}$$

then the decoding converges to a **vanishingly small probability of error**.

EXIT Charts - Code Rate Optimization

- For given $\lambda(x)$ and $\rho(x)$, the code design rate is:

$$R = 1 - \frac{\sum_i \rho_i / i}{\sum_i \lambda_i / i}.$$

- A common approach is to maximize R over $\lambda(x)$ for fixed $\rho(x)$. This gives rise to a continuous linear program:

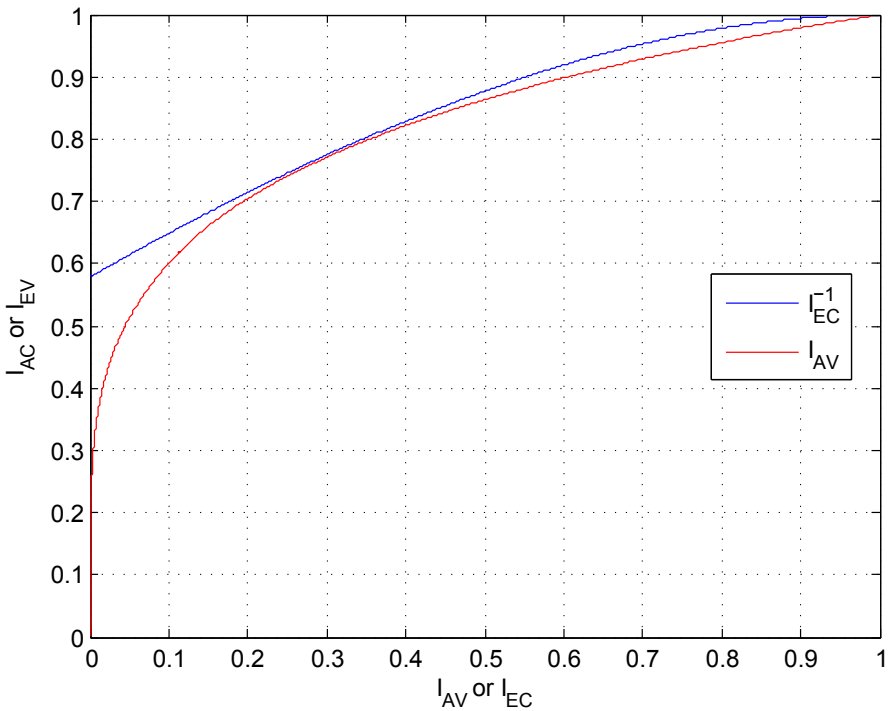
$$\max \sum_{i=2}^{v_{\max}} \lambda_i / i$$

$$\text{s.t. } I_{EC}^{-1}(I_{AV}) < I_{EV}(I_{AV}) = \sum_{i=2}^{v_{\max}} \lambda_i I_{EV}^{i}(I_{AV}), \ \forall I_{AV} \in (0,1)$$

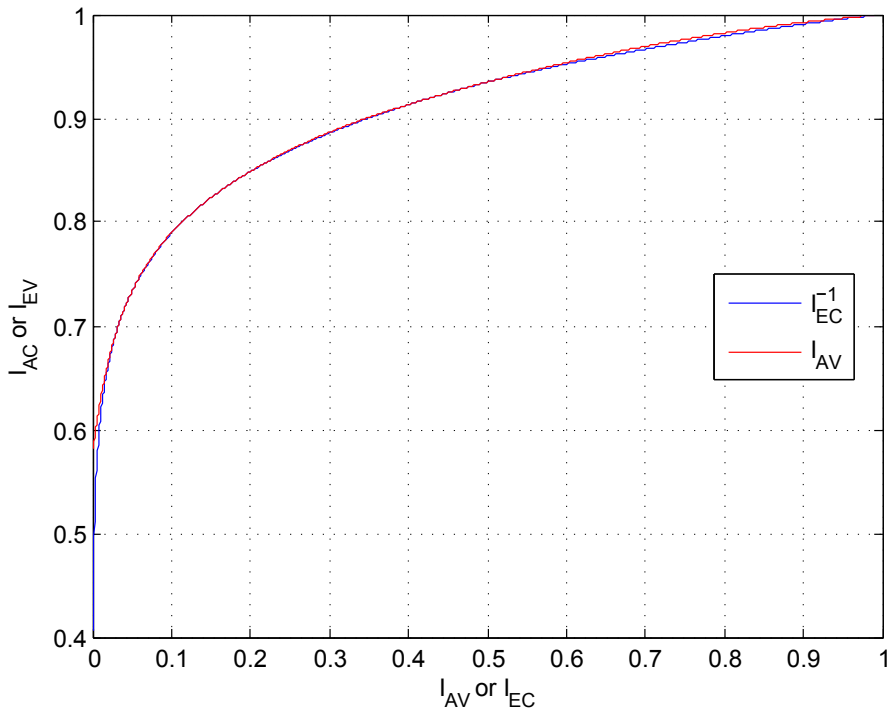
$$\sum_{i=2}^{v_{\max}} \lambda_i = 1, \ \lambda_i \geq 0, \ i = 2, 3, \ldots, v_{\max}.$$

- We solve it by discretizing $I_{AV} \in (0,1)$.

EXIT Charts - Code Optimization



(a) $(3, 6)$-regular ensemble, $r = 0.5$.



(b) Optimized ensemble, $r = 0.58$.

Figure: Noise variance $\sigma^2 = 0.76$.

1. Introduction to LDPC Codes

2. Iterative Decoding of LDPC Codes
   - Belief Propagation
   - Density Evolution
   - Gaussian Approximation
   - EXIT Charts

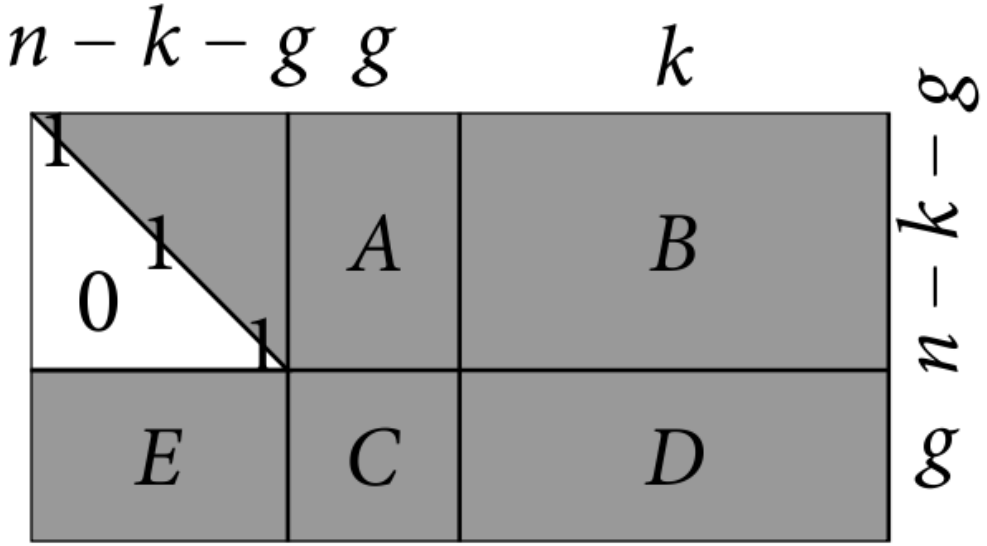
3. Efficient Encoding of LDPC and QC-LDPC Codes

4. Construction of QC-LDPC Codes
   - Deterministic
   - Random

5. RA and Structured RA Codes

Efficient Encoding of LDPC Codes

- In the general case, encoding has complexity $\mathcal{O}(n^2)$.
- If parity-check matrix has an *approximate upper triangular form*, it can be shown that encoding can be achieved with complexity $\mathcal{O}(n + g^2)$.



QC-LDPC Codes - Motivation

- In general, storage of parity check matrix requires $\mathcal{O}(n^2)$ memory.
- In Quasi-Cyclic (QC) LDPC codes, the parity check matrix consists of $q \times q$ sparse circulant submatrices (usually, permutation matrices).
- Each circulant is fully characterized by its first row or column.
- Required memory becomes $\mathcal{O}(n^2/q)$, i.e. **reduction by a factor of q.**

Efficient Encoding of QC-LDPC Codes

- The generator matrix $\mathbf{G}$ can be obtained in systematic form from its $cq \times tq$ parity-check matrix $\mathbf{G} = [\, \mathbf{I} \;\; \mathbf{G}_P \,]$.
- Encoding is done as follows

$$\mathbf{c} = \mathbf{aG} = [\, \mathbf{a} \; \mathbf{p} \,],$$

where

$$\mathbf{a} = \boxed{\mathbf{a}_1 \mid \mathbf{a}_2 \mid \cdots \mid \mathbf{a}_{t-c}}$$

$\mathbf{G}_P$ has the form:

$$\mathbf{G}_P = \begin{bmatrix} \mathbf{G}_{1,1} & \mathbf{G}_{1,2} & \cdots & \mathbf{G}_{1,c} \\ \mathbf{G}_{2,1} & \mathbf{G}_{2,2} & \cdots & \mathbf{G}_{2,c} \\ \vdots & \ddots & \vdots & \\ \mathbf{G}_{t-c,1} & \mathbf{G}_{t-c,2} & \cdots & \mathbf{G}_{t-c,c} \end{bmatrix}.$$

- $\mathbf{G}_{i,j}$'s are circulants, e.g.

$$\mathbf{G}_{i,j} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Efficient Encoding of QC-LDPC Codes

- For $\mathbf{p}$, we have:

$$\mathbf{p} = \boxed{\mathbf{p}_1 \mid \mathbf{p}_2 \mid \cdots \mid \mathbf{p}_c}$$

- The j-th parity q-tuple can then be obtained as follows:

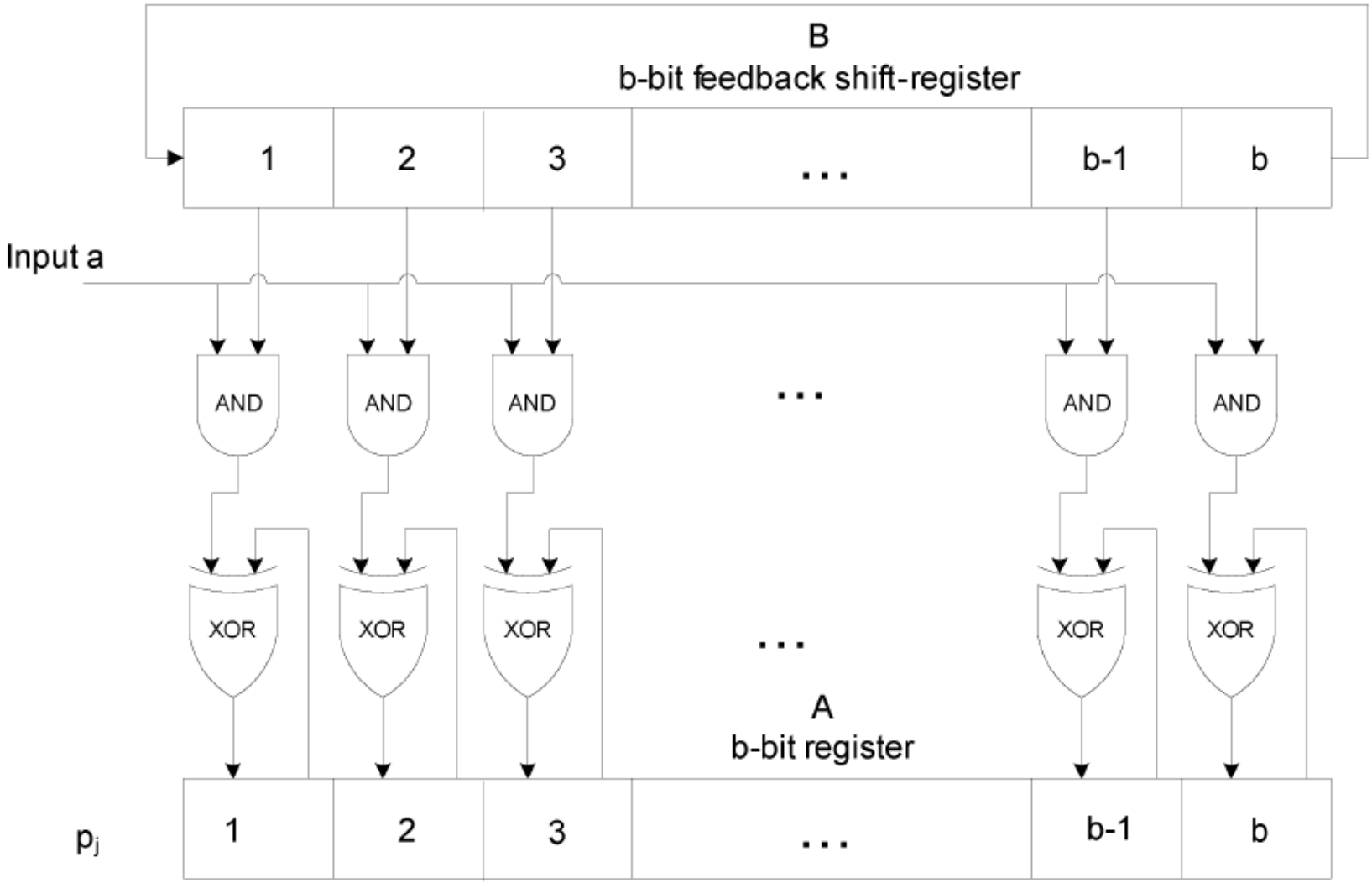
$$\mathbf{p}_j = \mathbf{a}_1\mathbf{G}_{1,j} + \mathbf{a}_2\mathbf{G}_{2,j} + \ldots + \mathbf{a}_{t-c}\mathbf{G}_{t-c,j},$$

  where $\mathbf{a}_j$ is the j-th q-tuple of the input.
- The $\mathbf{G}_{i,j}$'s are also circulants which can be fully described by their first row, $\mathbf{g}_{i,j}$.
- Thus, we can calculate each term of $\mathbf{p}_j$ as follows:

$$\mathbf{a}_i\mathbf{G}_{i,j} = a_{(i-1)b+1}\mathbf{g}_{i,j}^{(0)} + a_{(i-1)b+2}\mathbf{g}_{i,j}^{(1)} + \ldots + a_{ib}\mathbf{g}_{i,j}^{(b-1)}, \qquad (1)$$

  where $\mathbf{g}_{i,j}^{(l)}$ denotes the l-th right cyclic shift of $\mathbf{g}_{i,j}$.

Efficient Encoding of QC-LDPC Codes



Efficient Encoding of QC-LDPC Codes

- Some other schemes have been proposed. Wide variety of throughput-size trade-offs.

Scheme	Encoding speed	FFs	2-input XOR	2-input AND
1	$(t-c)q$	$2cq$	cq	cq
2	cq	$(t-c)q$	$(t-c)q-1$	$(t-c)q$
3	q	tq	$\mathcal{O}(c^2q)$	0

Table: Various encoder architectures and their complexity.

q	Size of circulants
t	Number of horizontally stacked circulants.
c	Number of vertically stacked circulants.

1. Introduction to LDPC Codes

2. Iterative Decoding of LDPC Codes
   - Belief Propagation
   - Density Evolution
   - Gaussian Approximation
   - EXIT Charts

3. Efficient Encoding of LDPC and QC-LDPC Codes

4. Construction of QC-LDPC Codes
   - Deterministic
   - Random

5. RA and Structured RA Codes

Deterministic QC-LDPC Codes (Array Codes)

- For a prime q and positive integer $m \leq q$, we create the matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} & \dots & \mathbf{I} \\ \mathbf{I} & \mathbf{P}^1 & \dots & \mathbf{P}^{(m-1)} & \dots & \mathbf{P}^{(q-1)} \\ \mathbf{I} & \mathbf{P}^2 & \dots & \mathbf{P}^{2(m-1)} & \dots & \mathbf{P}^{2(q-1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{I} & \mathbf{P}^{(m-1)} & \dots & \mathbf{P}^{(m-1)(m-1)} & \dots & \mathbf{P}^{(m-1)(q-1)} \end{bmatrix}.$$

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix}$$

- $\mathbf{P}^i$ is $\mathbf{P}$ with all columns cyclically shifted to the right by i positions.
- By convention, $\mathbf{P}^0 = \mathbf{I}$ and $\mathbf{P}^\infty = \mathbf{0}$.
- $\mathbf{H}$ represents a (j, q) regular QC-LDPC code.

Deterministic QC-LDPC Codes (Modified Array Codes)

- For a prime q and positive integer $m \leq q$, we create the matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} & \dots & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{P}^{1} & \dots & \mathbf{P}^{(m-2)} & \dots & \mathbf{P}^{(q-2)} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \dots & \mathbf{P}^{2(m-3)} & \dots & \mathbf{P}^{2(q-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \dots & \mathbf{P}^{(m-1)(q-m)} \end{bmatrix}.$$

- Efficient encoding possible due to structure. Slightly irregular code.

Random Construction of QC-LDPC Codes (Regular Codes)

- The $mq \times nq$ parity-check matrix is constructed as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P}^{a_{11}} & \mathbf{P}^{a_{12}} & \dots & \mathbf{P}^{a_{1n}} \\ \mathbf{P}^{a_{21}} & \mathbf{P}^{a_{22}} & \dots & \mathbf{P}^{a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}^{a_{m1}} & \mathbf{P}^{a_{m2}} & \dots & \mathbf{P}^{a_{mn}} \end{bmatrix}$$

  where $a_{ij} \in \{0, 1, 2, \dots, q-1\}$.

- If $a_{ij} \in \{0, 1, 2, \dots, L-1\} \cup \infty$, then irregular codes can also be constructed.

Random Construction of QC-LDPC Codes (Irreg. Codes)

- The number of non-zero blocks in each column of the block parity check matrix can be chosen according to a degree distribution.
- Permutation matrices $\mathbf{P}^i$ are used, i is usually chosen at random, and, if some constraints, e.g. cycle girth, check node distribution, are violated, another value for i is chosen.
- Very similar to one of Gallager's constructions of irregular codes.

Other Constructions of QC-LDPC Codes

- Constructions based on Finite Geometries.
- Constructions based on Reed-Solomon codes.
- Constructions based on masking of existing LDPC codes.

1. Introduction to LDPC Codes

2. Iterative Decoding of LDPC Codes
   - Belief Propagation
   - Density Evolution
   - Gaussian Approximation
   - EXIT Charts

3. Efficient Encoding of LDPC and QC-LDPC Codes

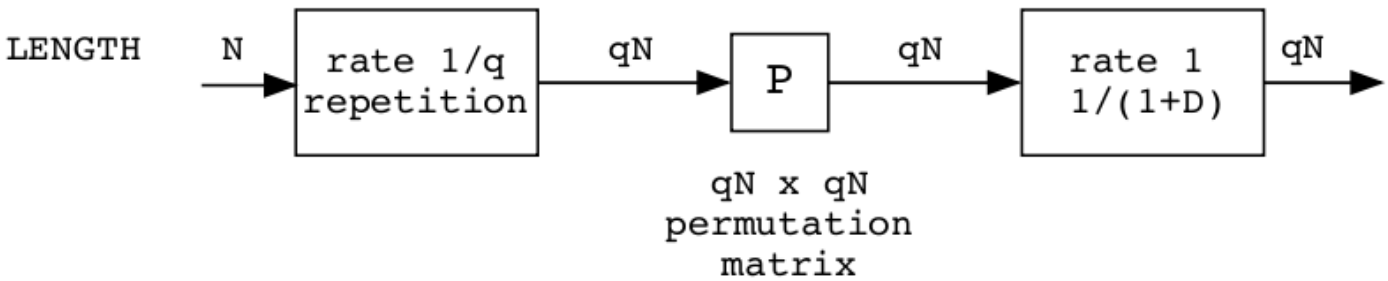
4. Construction of QC-LDPC Codes
   - Deterministic
   - Random

5. RA and Structured RA Codes

Repeat-Accumulate Codes

- RA codes: subclass of LDPC codes.

- Encoding:
  1. A frame of information symbols of length N is repeated q times.
  2. A random (but fixed) permutation is applied to the resulting frame.
  3. The permuted frame is fed to a rate-1 accumulator with transfer function $1/(1+D)$.



- For Irregular RA codes, each information bit is repeated according to a repetition profile.

Structured Repeat-Accumulate Codes

- The parity-check matrix of an RA code can be written as follows:

$$\mathbf{H} = [\mathbf{H}_1 \quad \mathbf{H}_2],$$

where:

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 \end{bmatrix}.$$

- We can make RA codes even simpler by enforcing a QC structure on $\mathbf{H}_1$.

Structured IRA Codes

- We construct the matrix $\mathbf{H}_1$ as follows:

$$\mathbf{H}_1 = \begin{bmatrix} \mathbf{P}^{a_{11}} & \mathbf{P}^{a_{12}} & \dots & \mathbf{P}^{a_{1n}} \\ \mathbf{P}^{a_{21}} & \mathbf{P}^{a_{22}} & \dots & \mathbf{P}^{a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}^{a_{m1}} & \mathbf{P}^{a_{m2}} & \dots & \mathbf{P}^{a_{mn}} \end{bmatrix}$$

where P is a right cyclic shift $q \times q$ permutation matrix and $a_{i,j} \in \{0, 1, \dots, q-1, \infty\}$ are the corresponding exponents.

Structured IRA Codes

- If we use $\mathbf{H}_1$ as is, the resulting code has a poor minimum distance.
- We use a permuted version of $\mathbf{H}_1$ instead, where the permutation is chosen so that the minimum codeword weight for low-weight inputs is increased.
- So, the final parity-check matrix will be:

$$\mathbf{H} = [\,\boldsymbol{\Pi}\mathbf{H}_1 \quad \mathbf{H}_2\,]$$

- The resulting code is regular unless we choose to mask out some entries (by choosing $a^{ij} = \infty$) in the $\mathbf{H}_1$ matrix in accordance with a targeted repetition profile.

Appendix - $\phi(x)$ and $J(\sigma)$

- The function $\phi(x)$ is defined as follows:

$$\phi(x) = \begin{cases} 1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{+\infty} \tanh\left(\frac{u}{2}\right) \exp\left\{-\frac{(u-x)^2}{4x}\right\} du, & x > 0 \\ 1, & x = 0. \end{cases}$$

- The function $J(\sigma)$ is defined as follows:

$$J(\sigma) = 1 - \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} e^{-\frac{(x-\sigma^2/2)^2}{2\sigma^2}} \log_2(1 + e^{-x}) dx.$$