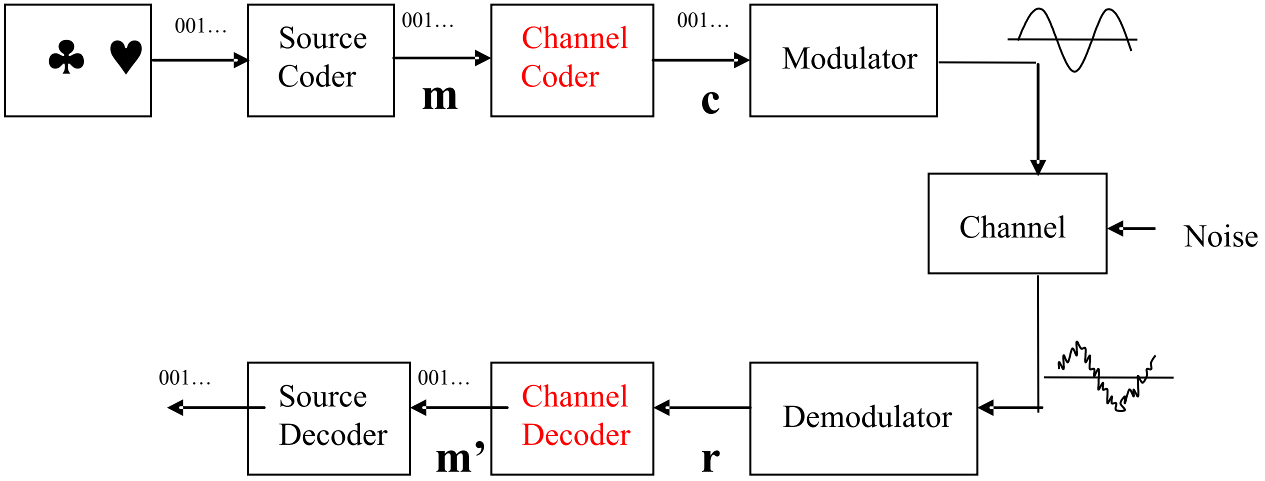
Part-I: A Brief Introduction to Channel Coding

Marc Fossorier
Department of Electrical Engineering
University of Hawaii

Communication Channel:

- (N, K, d_{min}) binary linear block code.



Linear Codes:

- A linear code *C* is totally define by its *KxN* generator matrix *G* or its *(N-K)xN* parity check matrix *H* via:

The picture can't be displayed.

$$\mathrm{m}\,G = \mathrm{c}$$

$$\mathrm{c}\,H^T = 0$$

Example:

$$G_{k\times n} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \Rightarrow H_{(n-k)\times n} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

1. Say you send the message m=[0 1]. Create the codeword c

$$c = mG = [0\ 1] \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = [0\ 1\ 1]$$

2. Say the receiver receive $\hat{c}$=[0 0 1]
3. How can the receiver know the codeword he receive is wrong ?
4. Test the receive codeword with the parity check matrix H

$$\hat{c}H^t = [0\ 0\ 1][1\ 1\ 1]^t = [1] \text{ - error detected}$$

$$\hat{c}H^t = [0\ 1\ 1][1\ 1\ 1]^t = [0] \text{ - good}$$

Maximum Likelihood Decoding:

- Find the most likely codeword **c** based on received sequence.
- For AWGN, **c** minimizes the discrepency metric:

$$L(r,c) = \sum_{\ell : r_\ell^{HD} \neq c_\ell} |r_\ell| \quad (\geq 0)$$

- **"Brute-force" decoding**: Out of 2^K possible solutions, find the most probable (i.e. the codeword with minimum discrepency metric).

Coding/Decoding:

- **Mathematical problem**: design best code (i.e. best performance for given channel).
- **Engineering problem**: design best code that can be implemented.

Majority-logic decoding

Simple and effective way for decoding certain class of block codes, especially cyclic code.

Idea behind Majority logic decoding

Take an (n, k) cyclic code C with parity-check matrix H
Chose a codeword $\mathbf{c}$ in C, and a codeword $\mathbf{h_k}$ in H then

$$\mathbf{c} \cdot \mathbf{h_k} = \mathbf{0}$$

Let $\mathbf{e}$ be an error pattern, and $\mathbf{r}$ a received sequence.

$$\mathbf{r} = \mathbf{c} + \mathbf{e}$$

Majority-logic decoding

Simple and effective way for decoding certain class of block codes, especially cyclic code.

Idea behind Majority logic decoding

Take an (n, k) cyclic code C with parity-check matrix H
Chose a codeword $\mathbf{c}$ in C, and a codeword $\mathbf{h_k}$ in H then

$$\mathbf{c} \cdot \mathbf{h_k} = \mathbf{0}$$

Let $\mathbf{e}$ be an error pattern, and $\mathbf{r}$ a received sequence.

$$\mathbf{r} = \mathbf{c} + \mathbf{e}$$

Imagine $\mathbf{e} = (0, 0, \ldots, 0, e_{14} = 1)$

$$\begin{matrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{matrix} = \begin{matrix} \mathbf{h}_7 \cdot \mathbf{r} \\ \mathbf{h}_{11} \cdot \mathbf{r} \\ \mathbf{h}_{13} \cdot \mathbf{r} \\ \mathbf{h}_{14} \cdot \mathbf{r} \end{matrix} = \begin{matrix} e_7 & +e_8 & +e_{10} & +e_{14} \\ e_3 & +e_{11} & +e_{12} & +e_{14} \\ e_1 & +e_5 & +e_{13} & +e_{14} \\ e_0 & +e_2 & +e_6 & +e_{14} \end{matrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \end{matrix}$$

Check sums A_1, A_2, A_3 and A_4 return a 1 $\Rightarrow$ error detected.
The clear majority has detected the error in e_{14}.

Imagine $\mathbf{e} = (0, 0, ..., 0, e_{13} = 1, e_{14} = 1)$ (correcting e_{14})

$$\begin{matrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{matrix} = \begin{matrix} \mathbf{h}_7 \cdot \mathbf{r} \\ \mathbf{h}_{11} \cdot \mathbf{r} \\ \mathbf{h}_{13} \cdot \mathbf{r} \\ \mathbf{h}_{14} \cdot \mathbf{r} \end{matrix} = \begin{matrix} e_7 & +e_8 & +e_{10} & +e_{14} \\ e_3 & +e_{11} & +e_{12} & +e_{14} \\ e_1 & +e_5 & +e_{13} & +e_{14} \\ e_0 & +e_2 & +e_6 & +e_{14} \end{matrix} = \begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}$$

Imagine $\mathbf{e} = (0, 0, ..., 0, e_{13} = 1, e_{14} = 1)$ (correcting e_{13})

$$\begin{matrix} A'_1 \\ A'_2 \\ A'_3 \\ A'_4 \end{matrix} = \begin{matrix} \mathbf{h}_6 \cdot \mathbf{r} \\ \mathbf{h}_{10} \cdot \mathbf{r} \\ \mathbf{h}_{12} \cdot \mathbf{r} \\ \mathbf{h}_{13} \cdot \mathbf{r} \end{matrix} = \begin{matrix} e_6 & +e_7 & +e_9 & +e_{13} \\ e_2 & +e_{10} & +e_{11} & +e_{13} \\ e_0 & +e_4 & +e_{12} & +e_{13} \\ e_{14} & +e_1 & +e_5 & +e_{13} \end{matrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 0 \end{matrix}$$

Cyclic code helps the decoding process.

Imagine $\mathbf{e} = (0, 0, ..., 0, e_{12} = 1, e_{13} = 1, e_{14} = 1)$ (correcting e_{14})

$$\begin{pmatrix} A_2 \\ A_3 \end{pmatrix} = \begin{pmatrix} \mathbf{h}_{11} \cdot \mathbf{r} \\ \mathbf{h}_{13} \cdot \mathbf{r} \end{pmatrix} = \begin{pmatrix} e_3 & +e_{11} & +e_{12} & +e_{14} \\ e_1 & +e_5 & +e_{13} & +e_{14} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Imagine $\mathbf{e} = (0, 0, ..., 0, e_{12} = 1, e_{13} = 1, e_{14} = 1)$ (correcting e_{13})

$$\begin{pmatrix} A'_3 \\ A'_4 \end{pmatrix} = \begin{pmatrix} \mathbf{h}_{12} \cdot \mathbf{r} \\ \mathbf{h}_{13} \cdot \mathbf{r} \end{pmatrix} = \begin{pmatrix} e_0 & +e_4 & +e_{12} & +e_{13} \\ e_{14} & +e_1 & +e_5 & +e_{13} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Imagine $\mathbf{e} = (0, 0, ..., 0, e_{12} = 1, e_{13} = 1, e_{14} = 1)$ (correcting e_{12})

$$\begin{pmatrix} A''_3 \\ A''_4 \end{pmatrix} = \begin{pmatrix} \mathbf{h}_{11} \cdot \mathbf{r} \\ \mathbf{h}_{12} \cdot \mathbf{r} \end{pmatrix} = \begin{pmatrix} e_{14} & +e_3 & +e_{11} & +e_{12} \\ e_{13} & +e_0 & +e_4 & +e_{12} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Part-II: Introduction to LDPC Codes

Marc Fossorier

Department of Electrical Engineering

University of Hawaii

LDPC Codes:

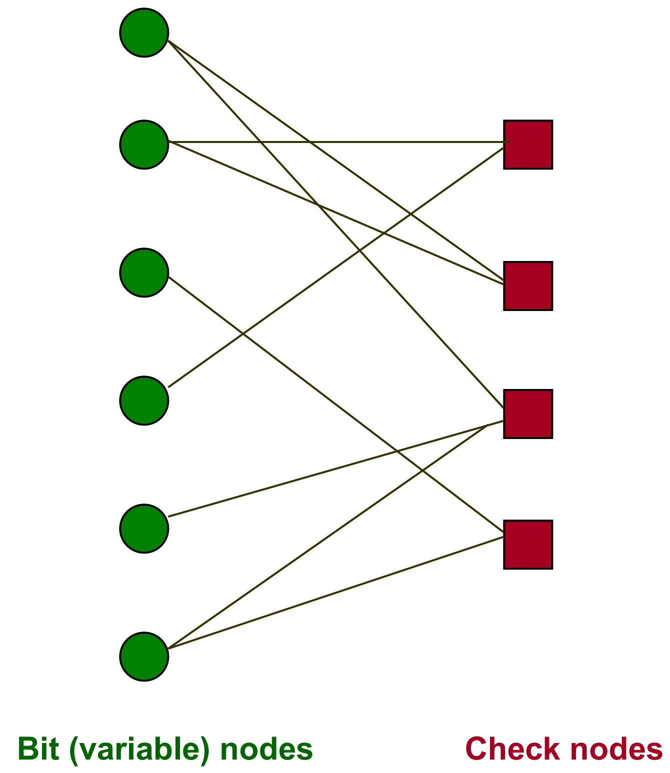
- First proposed by R.G. Gallager in 1960’s, and ressurected recently [Gallager-IRE62, MacKay-IT99] .
- Can achieve near Shannon limit performance with a sophisticated soft decision iterative decoding algorithm called belief propagation (BP) or sum-product algorithm [Luby-Mitzenmacher-Shokrollahi-Spielman:IT01, Richardson-Urbanke-IT01,] .

Representations of LDPC Codes

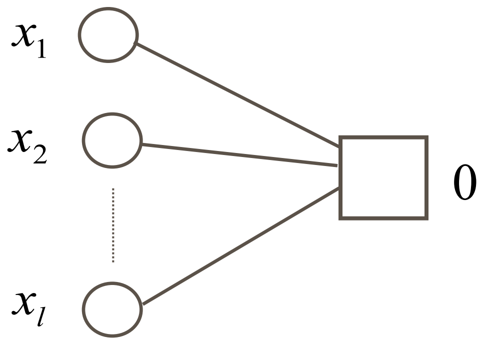
M x N Parity Check Matrix

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

Tanner Graph



Basic idea:



$$x_1 + x_2 + \ldots + x_l = 0$$

- The *l* bits $x_1,\ldots,x_l$ must satisfy a single parity-check constraint.
- If any of the *l* bits $x_1,\ldots,x_l$ is unknown, it can be reconstructed if the others are known.
- A single parity-check (SPC) code can correct at most one erasure.

Regular and Irregular LDPC Codes:

- Few 1's in H.
- An LDPC code is *regular* if its row and column weights are constants (say J and L). Otherwise it is *irregular.*
- *Irregular* LDPC codes have better performance than *regular* LDPC codes (and turbo codes) in general [Richardson-Urbanke-IT01] .

Regular (*J*,*L*) LDPC codes of length *N* and dimension *K*:

- Number of 1's: $J\,N = M\,L$
- Rate:

$$
\begin{aligned}
R &= K/N \\
&= 1-(N-K)/N \\
&\geq 1-M/N \\
&= 1-J/L
\end{aligned}
$$

Irregular (*J*,*L*) LDPC codes of length *N* and dimension *K*:

- Defined by edge degree distributions:

  λ_i : fraction of edges connected to degree-*i* variable (left) nodes.

  ρ_j : fraction of edges connected to *degree-j* check (right) nodes.

$$\sum_i \lambda_i = \sum_j \rho_j = 1$$

- Rate:

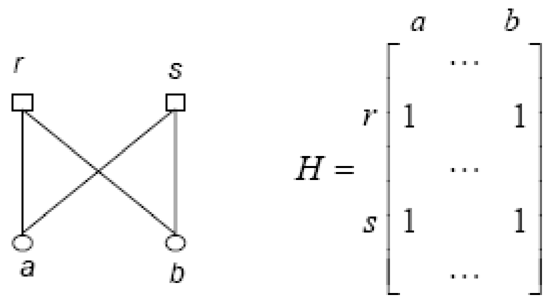
$$R \geq 1 - M / N$$

$$= 1 - \frac{\sum_i \lambda_i / i}{\sum_j \rho_j / j}$$

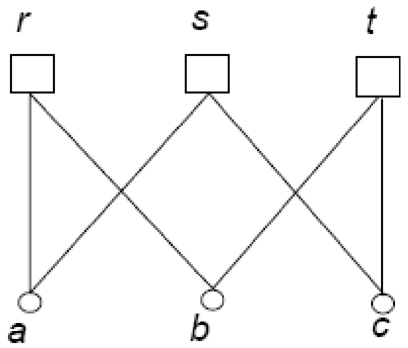
(the number of edges from variable (left) nodes equals the number of edges from check (right) nodes.

Definitions:

- A cycle of length *l* in a Tanner graph is a path comprising *l* edges which closes back on itself.
- The girth of a Tanner graph is the minimum cycle length of the graph.
- The shortest possible cycle in a bipartite graph is of length-4:



- Cycles of length-6 play an important role in iterative decoding:

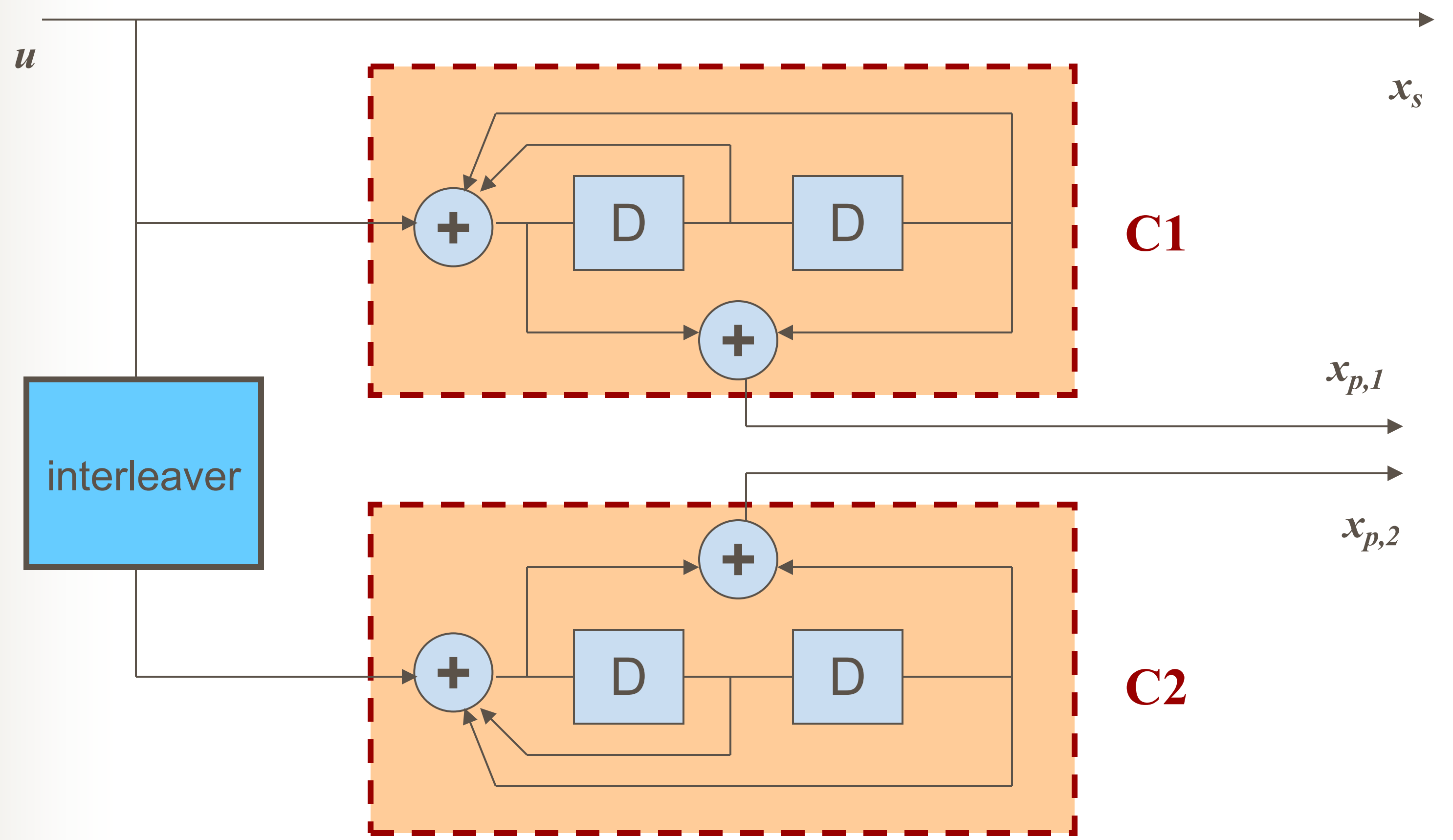


$$H = \begin{array}{c} \\ r \\ s \\ t \end{array}\overset{\begin{array}{ccc} a & b & c \end{array}}{\begin{bmatrix} 1 & 1 & \\ 1 & & 1 \\ & 1 & 1 \end{bmatrix}}$$

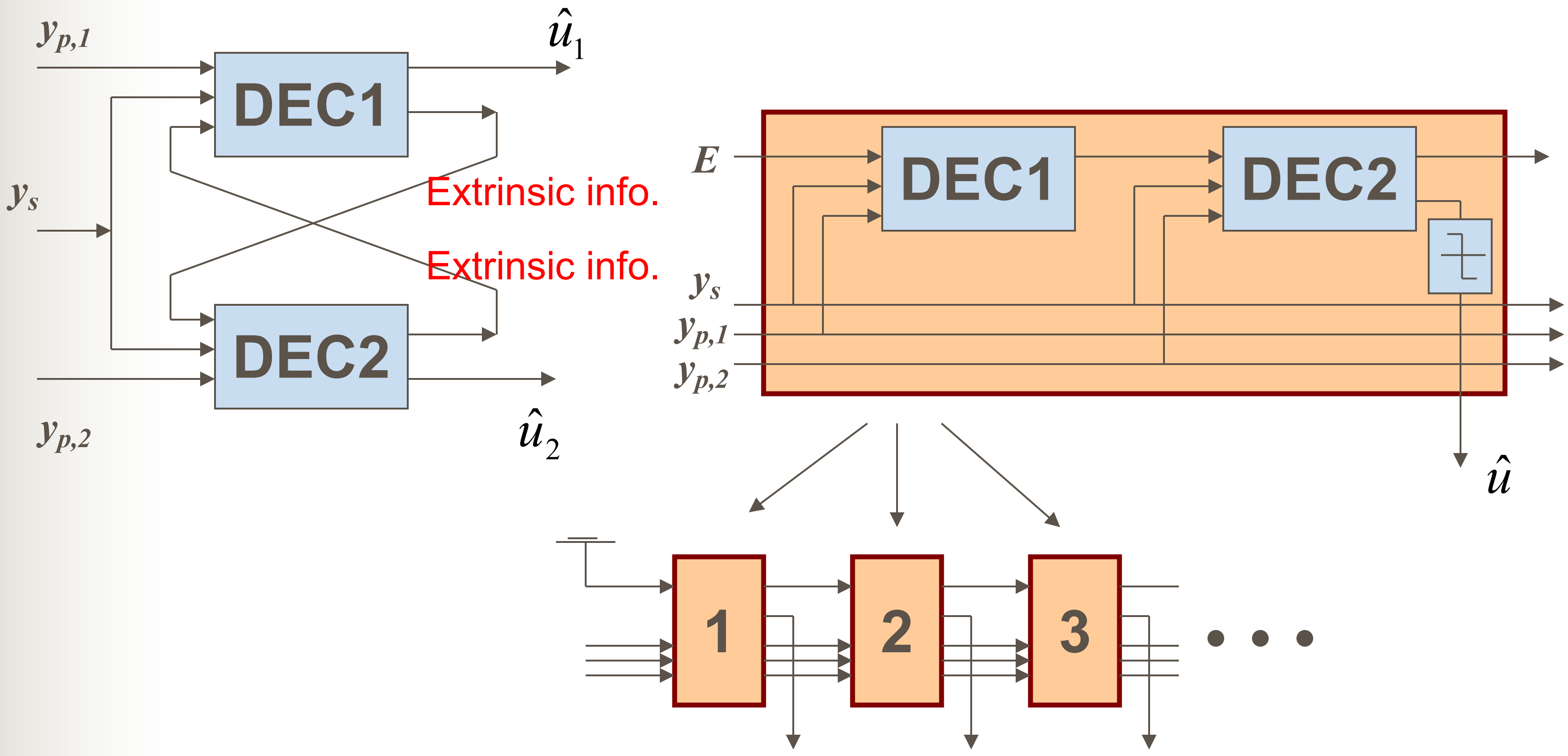
Part-III: Introduction to Turbo Codes

Marc Fossorier
Department of Electrical Engineering
University of Hawaii

- Encoder structure



- Decoder structure



- Decoding Algorithms

❖Soft-inputs soft-outputs (SISO) algorithm

Soft-inputs: component decoders can receive and make use of extrinsic information.

Soft-outputs: component decoders can provide reliability values for each bit, and deliver extrinsic information for further processing.

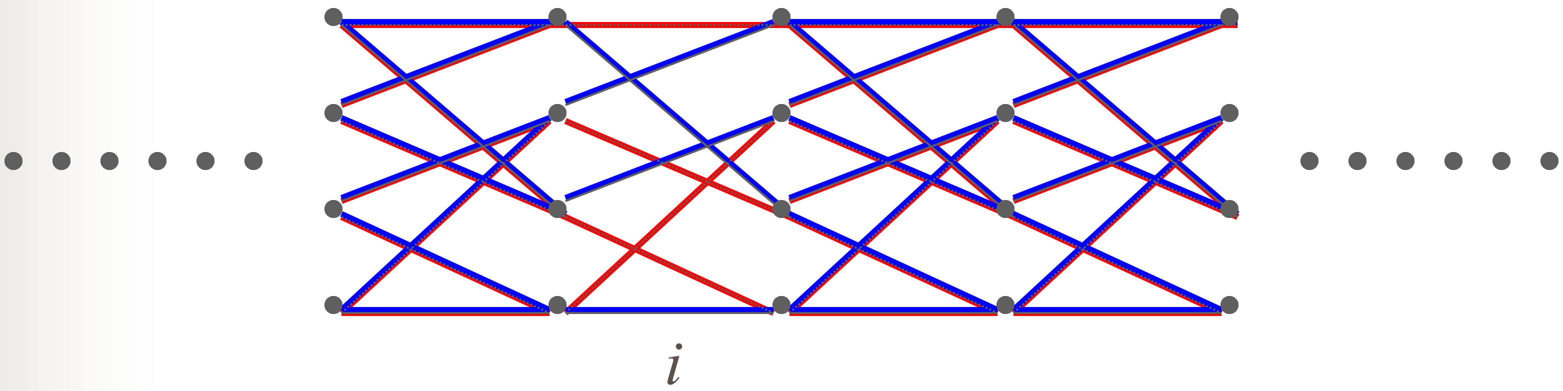
❖Turbo decoding algorithms include :

* Symbol-by-symbol maximum *a posteriori* (MAP),

* Max-Log-MAP,

* soft-outputs Viterbi Algorithm (SOVA).

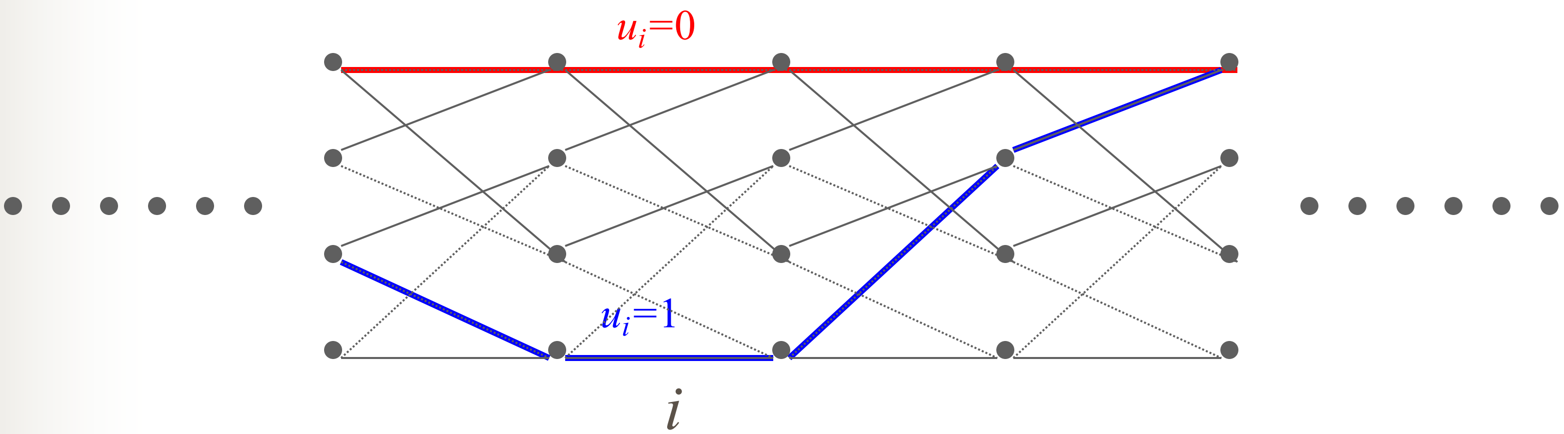
MAP Algorithm



$$\Lambda_i = \ln \frac{\sum_{c \in \Omega_1(i)} P(c \mid y, E)}{\sum_{c \in \Omega_0(i)} P(c \mid y, E)}$$

All paths are considered.

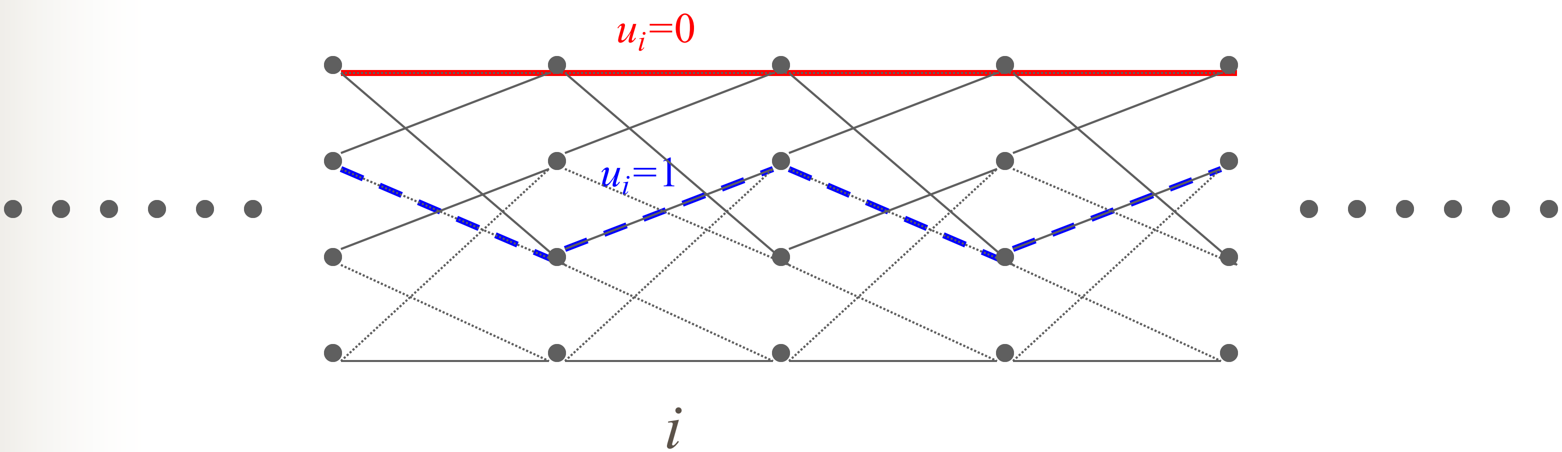
Max-Log-MAP Algorithm



$$\Lambda_i = \ln \frac{\max_{c \in \Omega_1(i)} P(c \mid y, E)}{\max_{c \in \Omega_0(i)} P(c \mid y, E)}$$

Difference of 2 metrics associated with the *best* 2 paths.

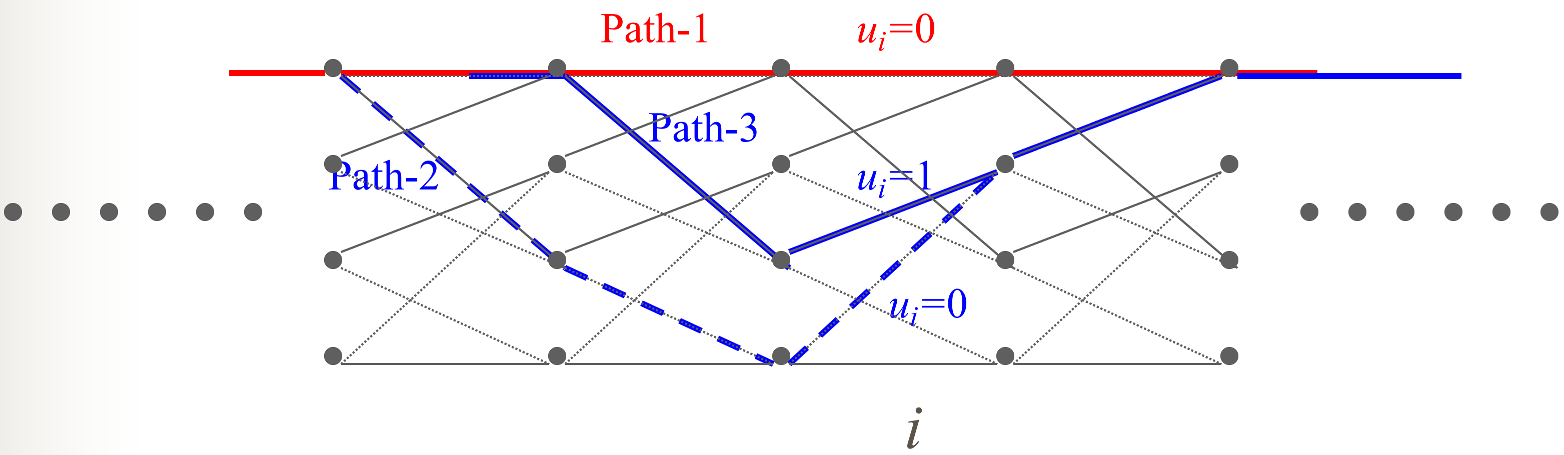
Soft-output Viterbi Algorithm (SOVA)



- Λ_i is the difference between 2 metrics associated with 2 paths.
- No guarantee that both paths are the best,

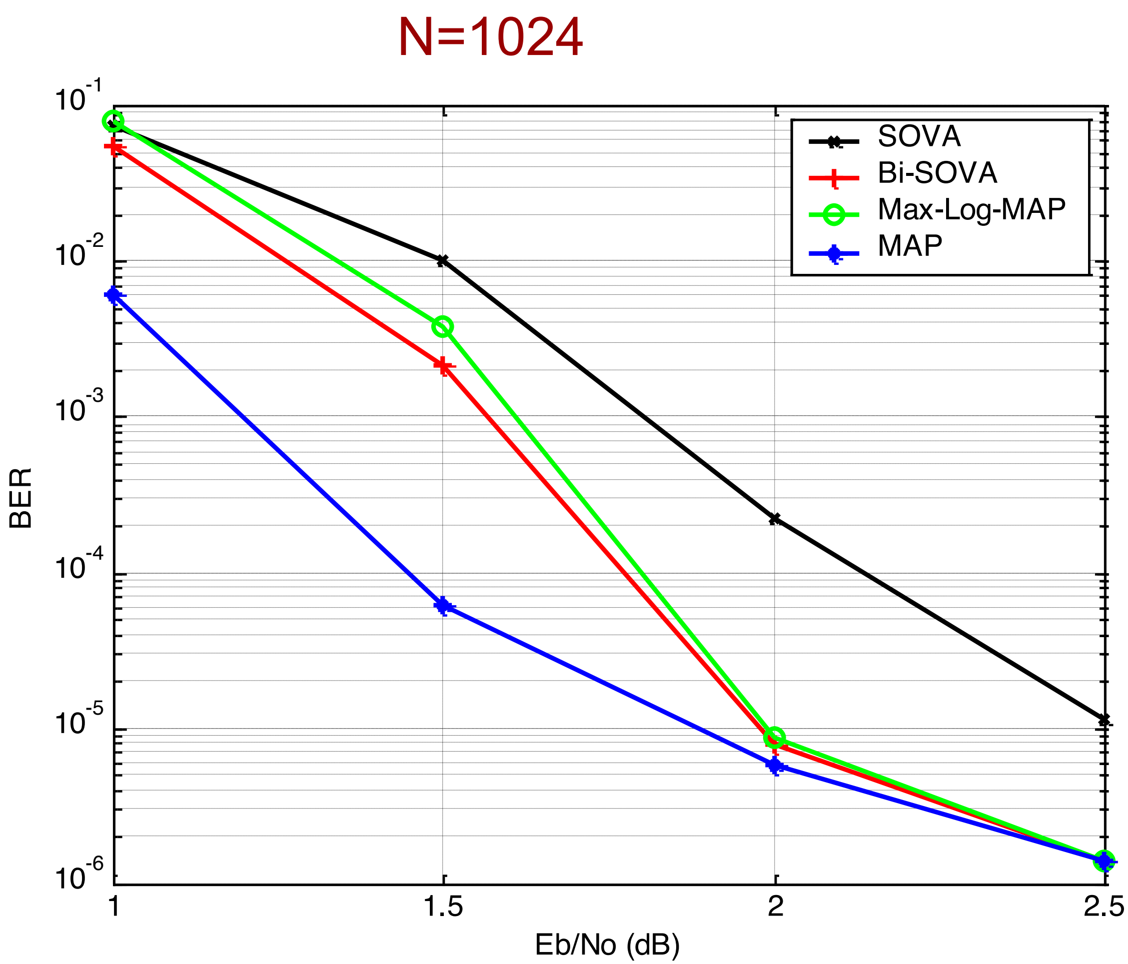
  $\Rightarrow$ Λ_i is often overestimated compared to the Max-Log-MAP.

Possible path selection in SOVA

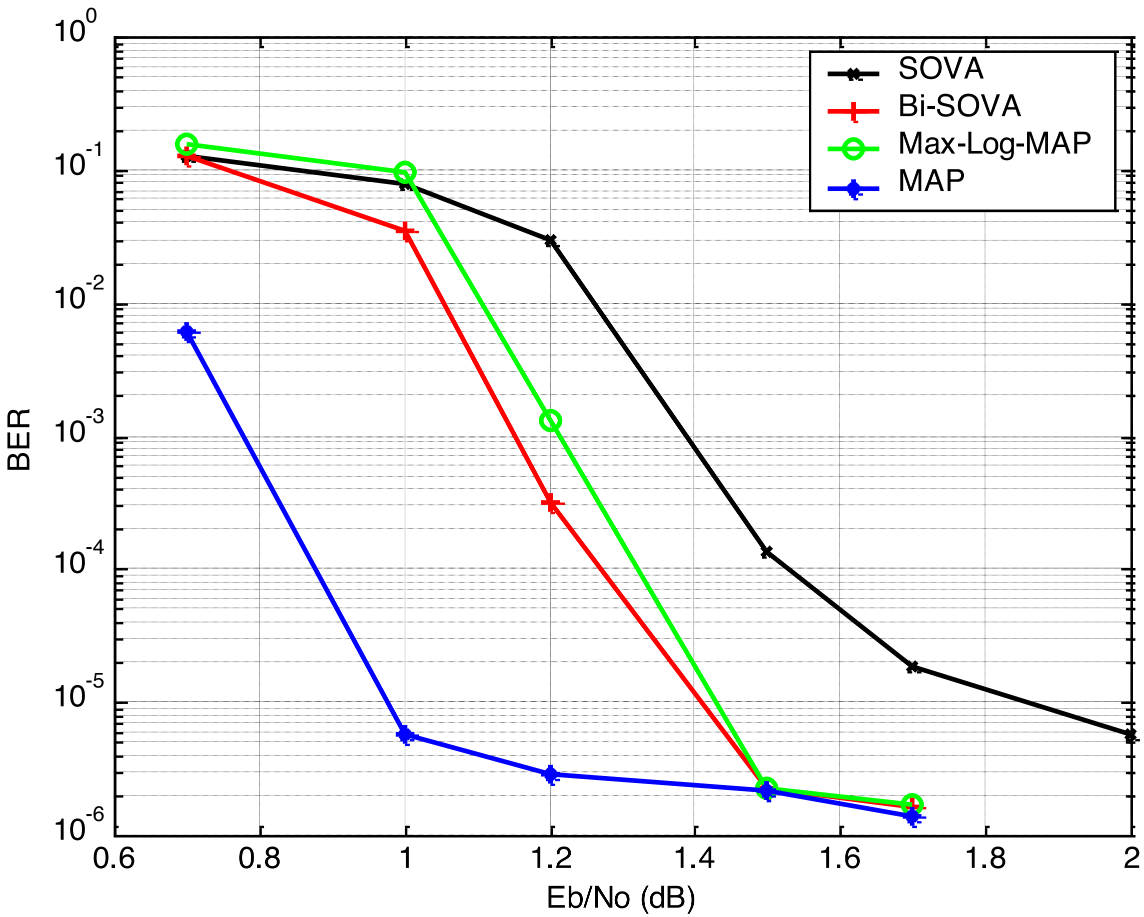


One of the best path may be discarded before remerge the survivor path: suggests bi-directional SOVA.

Decoding performance of Bi-directional SOVA



N=16384



SOVA
Bi-SOVA
Max-Log-MAP
MAP
BER
Eb/No (dB)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Normalized Max-Log-MAP algorithm

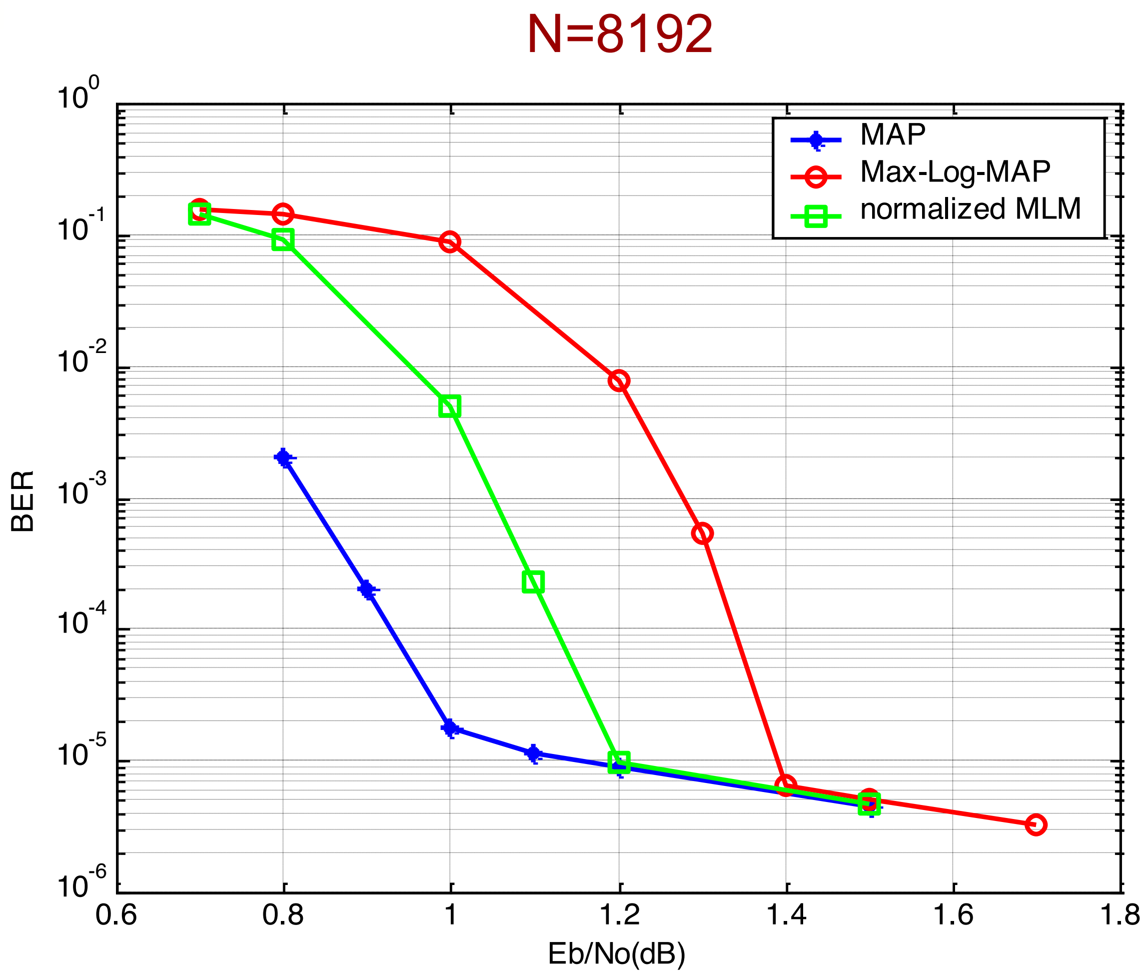
- The outputs of Max-Log-MAP algorithm are generally overestimated compared to those of the MAP algorithm.

Percentages associated with the different cases on the relationship between L_1 and L_2.

E_b/N_o (dB)	$\mathrm{sgn}(L_1) \neq \mathrm{sgn}(L_2)$	$\mathrm{sgn}(L_1) = \mathrm{sgn}(L_2)$ $\lvert L_1 \rvert < \lvert L_2 \rvert$	$\mathrm{sgn}(L_1) = \mathrm{sgn}(L_2)$ $\lvert L_1 \rvert \geq \lvert L_2 \rvert$
0.8	14.6	74.0	11.4
1.0	13.3	74.7	12.0
1.2	11.8	75.7	12.5
1.5	9.7	77.1	13.2
1.7	8.4	78.2	13.4

L_1 – MAP, L_2 – Max-Log-MAP

Performance of Normalized Max-Log-MAP algorithm



Part-IV: Constructions of LDPC Codes

Marc Fossorier
Department of Electrical Engineering
University of Hawaii

Random Constructions of (J,L) codes:

- Generate an all-0 M x N matrix H.
- Randomly assign L 1's per row while ensuring that no more than J 1's are assigned per column.
- Run a post processing subroutine to delete 4 cycles (random swap).

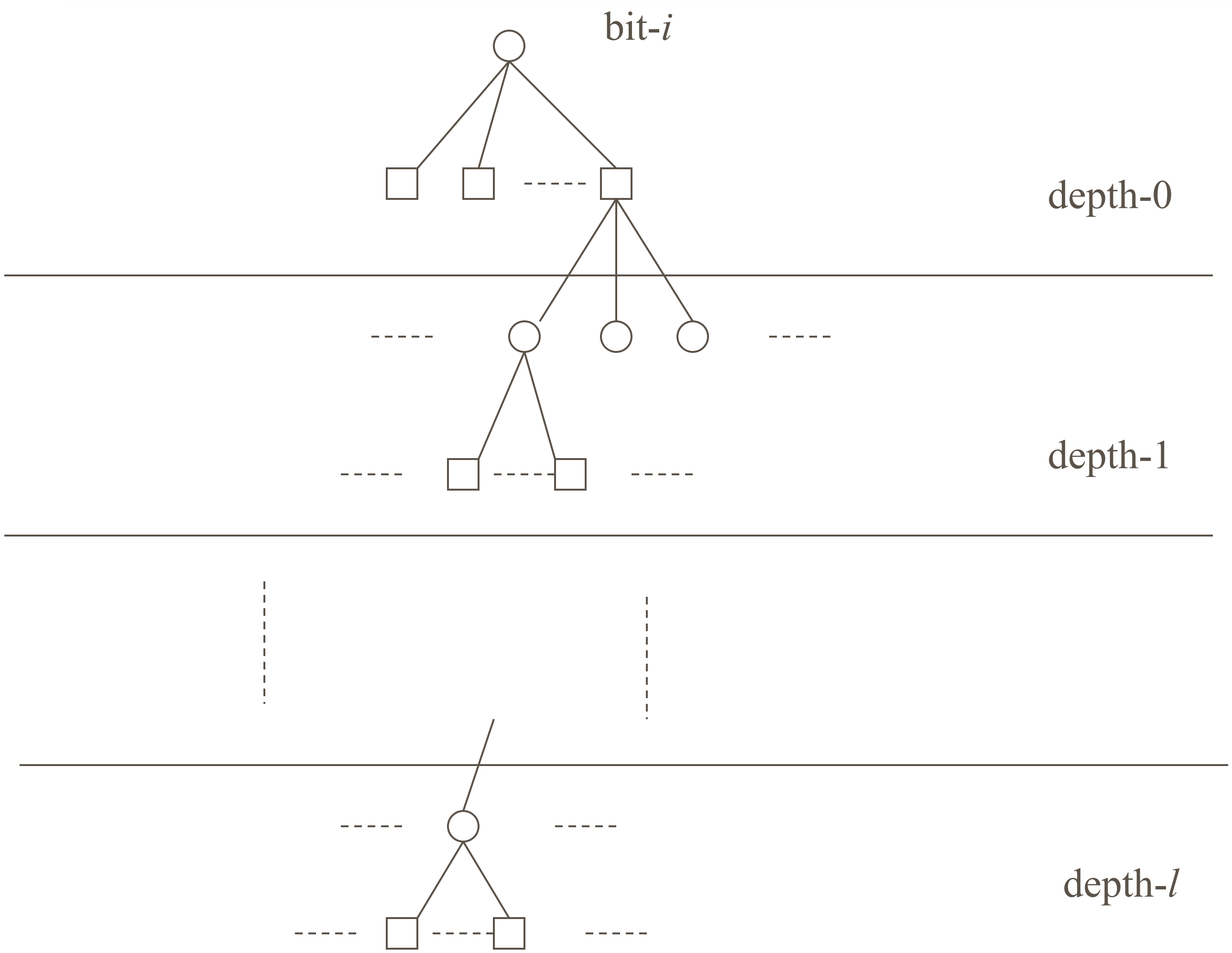
Pseudo-Random Constructions of (J,L) codes:

Progressive edge growth (PEG) algorithm [Hu & al. 02]

- Objective: try to maximize girth $g = 2(l+2)$.
- Edges are assigned one at a time as follows:
- For each bit-i from 1 to N:

  (1) Assign first edge to a check node among those of lowest degree.

  (2) Assign other edges to check nodes which are not among the neighbors of bit-i up to depth-l in the current graph.



bit-i
depth-0
depth-1
depth-l

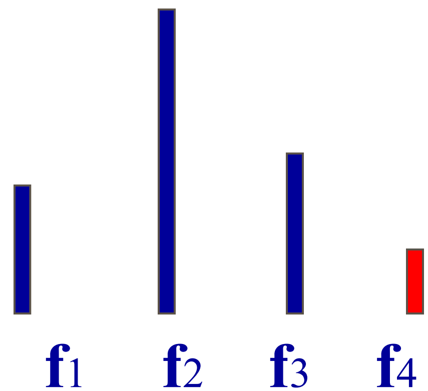
Random or Pseudo-Random Constructions of irregular codes:

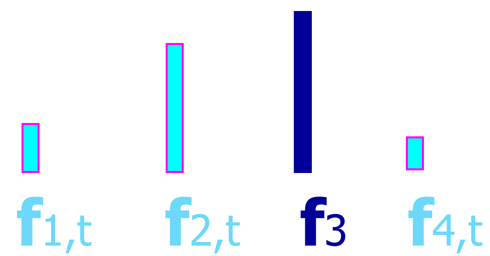
- The same approaches can be applied once degree distribution determined.
- Best degree distribution depends on channel considered as well as decoding algorithm.
- Differential evolution can be applied to determine the best distribution corresponding to a given objective function.

Parallel Differential Optimization:

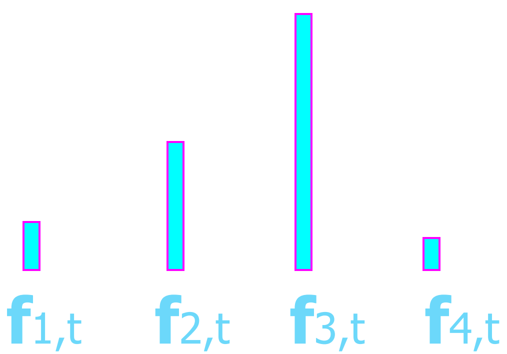
- Step 1: initialization
- Step 2: mutation and test
- Step 3: compare and update.
- Step 4: stopping test

Parallel Differential Optimization





$$\mathbf{f}_{k,t} = \mathbf{f}_4 + r \times (\mathbf{f}_\mathrm{i} + \mathbf{f}_\mathrm{j})$$



next iteration

Algebraic construction of LDPC codes:

- LDPC codes can be constructed based on the **points** and **lines** of finite geometries.
- Let **G** be a finite geometry with n points, $\{\mathrm{p}_1, \mathrm{p}_2, \cdots, \mathrm{p}_n\}$, and J lines, $\{\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_J\}$, which has the following fundamental structural properties:

(1) Each line consists of ρ points.

(2) Any two points are connected by one and only one line.

(3) Each point lies on γ lines, i.e., each point is intersected by γ lines.

(3) Two lines are either **parallel** (i.e., they contain no common point) or **intersect** at one and only one point.

- Let $\mathcal{L}$ be a **line** in **G**. Define a vector based on the points on $\mathcal{L}$ as follows:

$$\mathbf{v}_{\mathcal{L}} = (v_1, v_2, \cdots, v_n)$$

where

$$v_i = \begin{cases} 1, & \text{if } v_i \text{ corresponds to a point on } \mathcal{L}, \\ 0, & \text{otherwise.} \end{cases}$$

This vector $\mathbf{v}_{\mathcal{L}}$ is called the **incidence vector** of $\mathcal{L}$.

- $\mathbf{H}_{\mathbf{G}}^{(1)}$ is a $J \times n$ matrix whose rows are the incidence vectors of the J **lines** in the finite geometry **G** and whose columns correspond to the n **points** in **G**. The matrix $\mathbf{H}_{\mathbf{G}}^{(1)}$ has the following properties:

(1) each row has weight ρ;

(2) each column has weight γ;

(3) any two columns have at most one "1-component" in common, i.e., $\lambda = 0$ or 1;

(4) any two rows have at most one "1" in common.

- The **null space** of $\mathbf{H}_{\mathbf{G}}^{(1)}$ gives a LDPC code which is called a **type-I geometry-G LDPC code**, denote $\mathbf{C}_{\mathbf{G}}^{(1)}$.

- It follows from the structural properties of $\mathbf{H}_{\mathbf{G}}^{(1)}$ that for every code bit position of $\mathbf{C}_{\mathbf{G}}^{(1)}$, there are γ rows in $\mathbf{H}_{\mathbf{G}}^{(1)}$ which are **orthogonal** on it. Therefore, the minimum distance d_{min} of $\mathbf{C}_{\mathbf{G}}^{(1)}$ is at least $\gamma + 1$, i.e.,

$$d_{min} \geq \gamma + 1.$$

- There are two well known families of finite geometries: **Euclidean geometries** over finite fields and **projective geometries** over finite fields.

- Let $\mathrm{EG}(m, 2^s)$ denote the m-dimensional **Euclidean geometry** over $\mathrm{GF}(2^s)$. This geometry consists of

$$2^{ms} \qquad \text{points}$$

and

$$\frac{2^{(m-1)s}(2^{ms}-1)}{2^s-1} \qquad \text{lines.}$$

- Each line consists of

$$2^s \qquad \text{points}$$

- For each point **p** in $\mathrm{EG}(m, 2^s)$, there are

$$\frac{2^{ms}-1}{2^s-1} \qquad \text{lines}$$

that intersect at **p**.

- Let $\mathbf{H}_{EG}^{(1)}$ be a matrix whose rows are the incidence vectors of all the lines in EG$(m, 2^s)$ that do not pass through the origin and the columns correspond to the $2^{ms} - 1$ non-origin points of EG$(m, 2^s)$. Then $\mathbf{H}_{EG}^{(1)}$ consists of $2^{ms} - 1$ columns and $2^{(m-1)s}(2^{ms} - 1)/(2^s - 1)$ rows.

  $\mathbf{H}_{EG}^{(1)}$ has the following properties:

$$
\begin{aligned}
\rho &= 2^s, \\
\gamma &= \frac{2^{ms} - 1}{2^s - 1}, \\
\lambda &= 0 \text{ or } 1,
\end{aligned}
$$

- For $m = 2$, the type-I 2-dimensional EG-LDPC code has the following parameters:

Length	$n = 2^{2s} - 1,$
Number of parity bits	$n - k = 3^s - 1,$
Dimension	$k = 2^{2s} - 3^s,$
Minimum distance	$d_{min} = 2^s + 1,$

- A list of type-I two-dimensional EG-LDPC codes

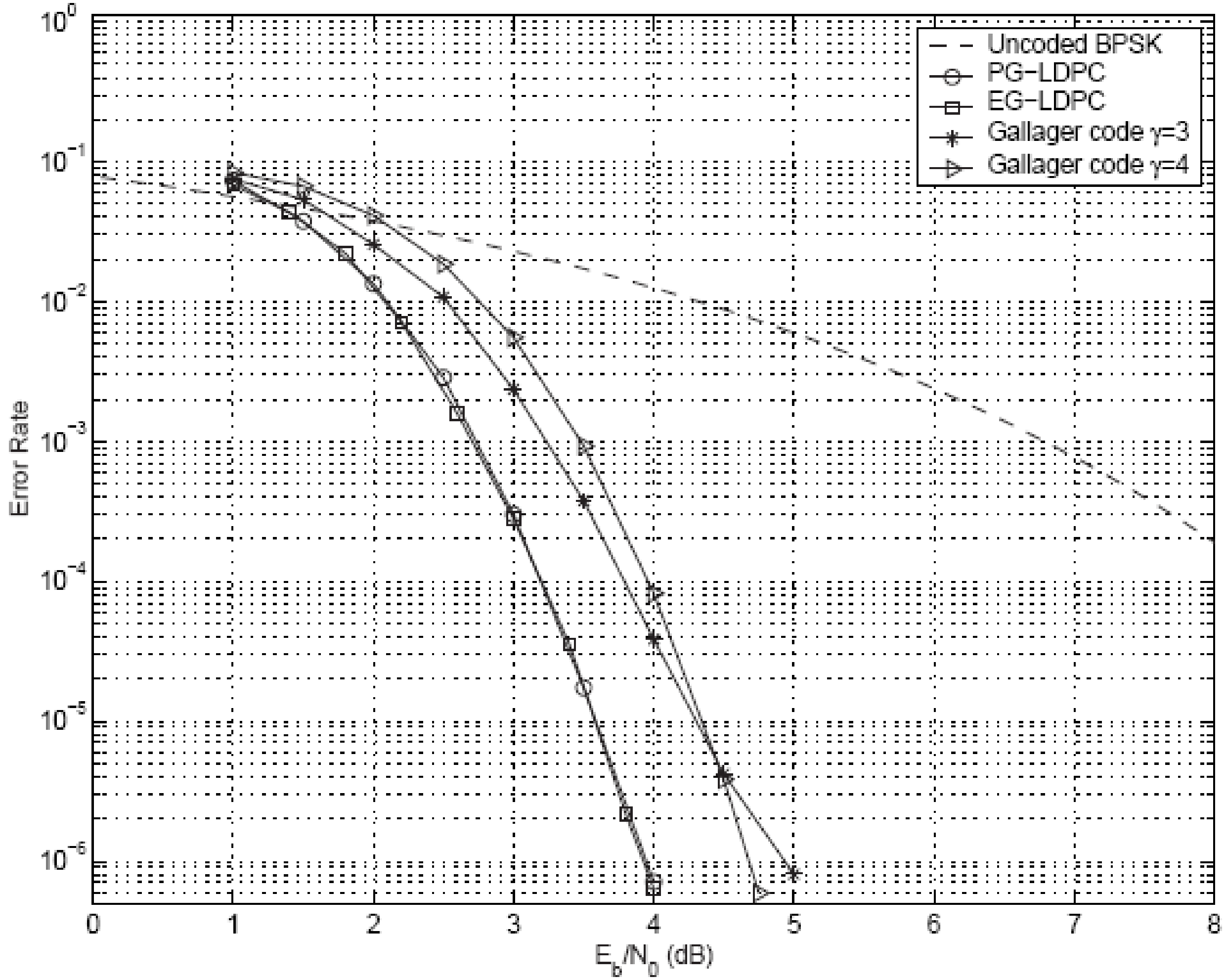
s	n	k	d_{min}	ρ	γ
2	15	7	5	4	4
3	63	37	9	8	8
4	255	175	17	16	16
5	1023	781	33	32	32
6	4095	3367	65	64	64
7	16383	14197	129	128	128

LDPC codes can be constructed based on the **points** and **lines** of the m-dimensional **projective geometry** PG$(m, 2^s)$ over GF(2^s). Type-I PG-LDPC codes are also **cyclic**. For $m = 2$, the type-I 2-dimensional PG-LDPC code has the following parameters:

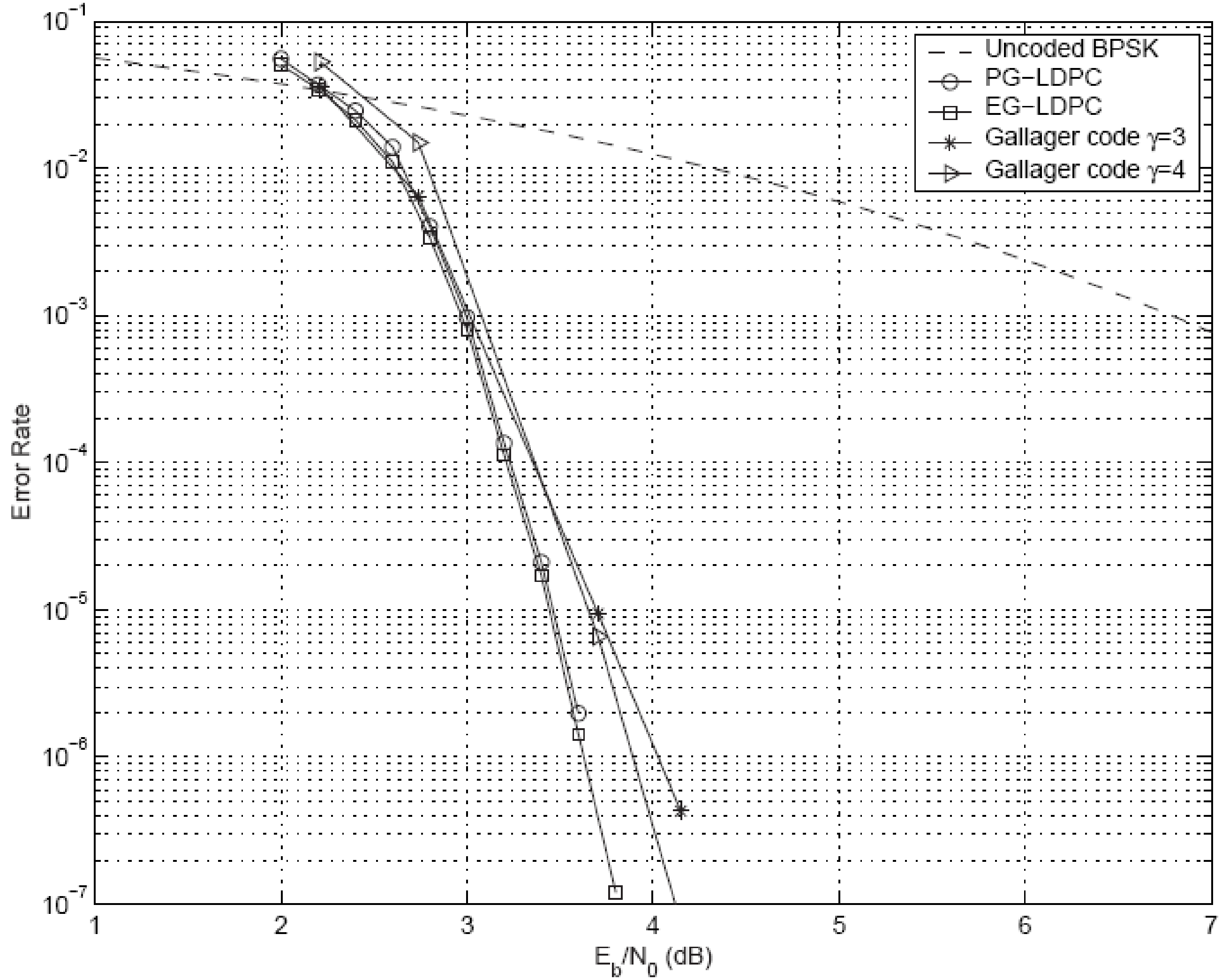
Length	$n = 2^{2s} + 2^s + 1,$
Number of parity bits	$n - k = 3^s + 1,$
Dimension	$k = 2^{2s} + 2^s - 3^s,$
Minimum distance	$d_{min} = 2^s + 2,$

- A list of type-I 2-dimensional PG-LDPC codes

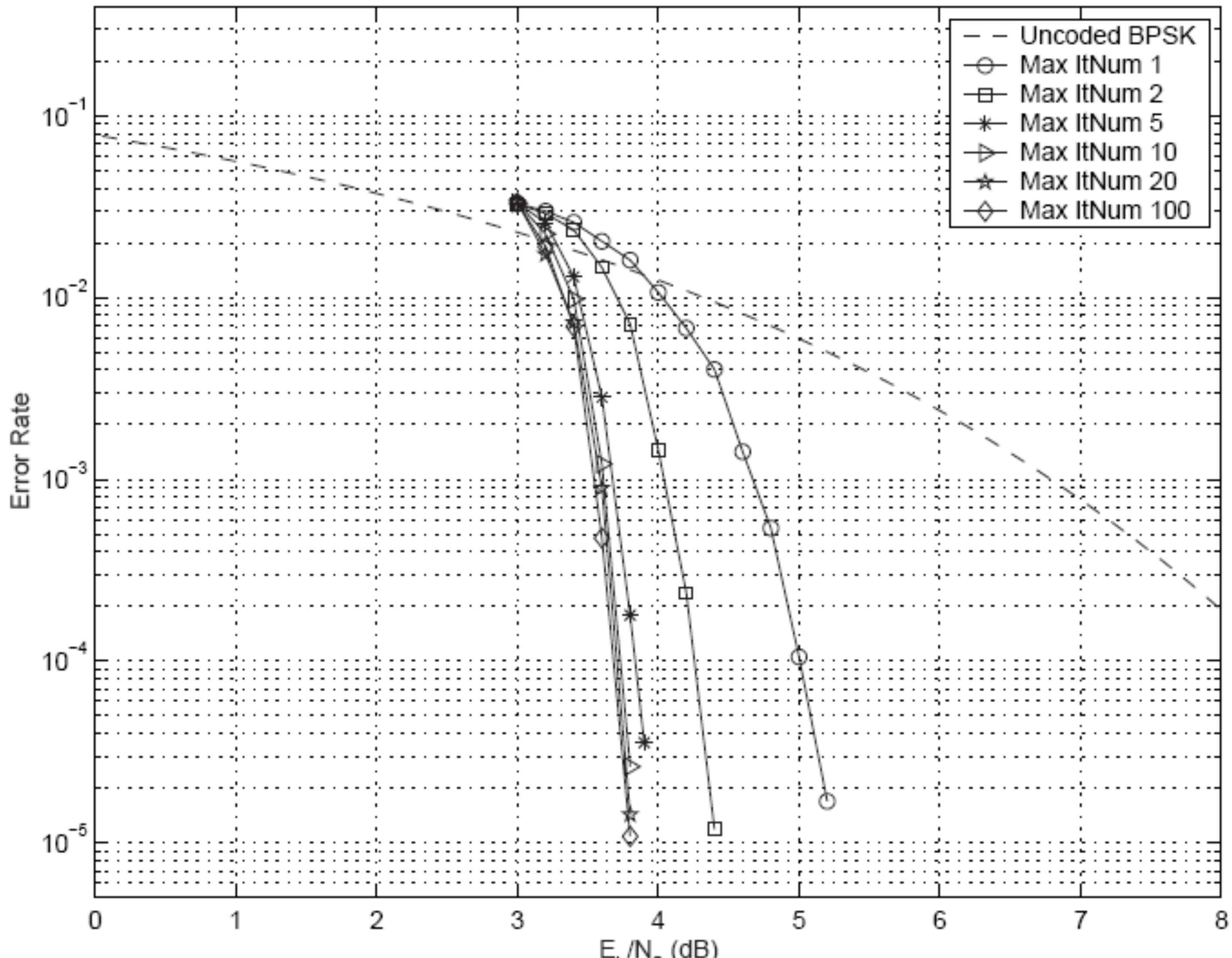
s	n	k	d_{min}	ρ	γ
2	21	11	6	5	5
3	73	45	10	9	9
4	273	191	18	17	17
5	1057	813	34	33	33
6	4161	3431	66	65	65
7	16513	14326	130	129	129



Bit-error probabilities of the (255, 175) EG-LDPC code, (273,191) PG-LDPC code and two computed searched (273,191) Gallager codes with IDBP.



Bit-error probabilities of the (1023, 781) EG-LDPC code, (1057, 813) PG-LDPC code and two computed searched (1057, 813) Gallager codes with IDBP.



Convergence of the IDBP algorithm for the (4095,3367) type-I EG-LDPC code.

- Finite geometry LDPC codes can be shortened to obtain good LDPC codes. This is achieved by deleting properly selected columns from their parity check matrix.

Quasi-Cyclic LDPC codes:

$$H = \begin{bmatrix} I(0) & I(0) & \cdots & I(0) \\ I(0) & I(p_{1,1}) & \cdots & I(p_{1,L-1}) \\ \vdots & & \ddots & \vdots \\ I(0) & I(p_{J-1,1}) & \cdots & I(p_{J-1,L-1}) \end{bmatrix}$$

with $I(p_{j,l})$ pxp circulant permutation matrix with 1 at column-$(r + p_{j,l})$ mod-p for row-r.

(J,L) regular LDPC code of length $N = pL$.

- Example: $J=2; L=3; p=5.$

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

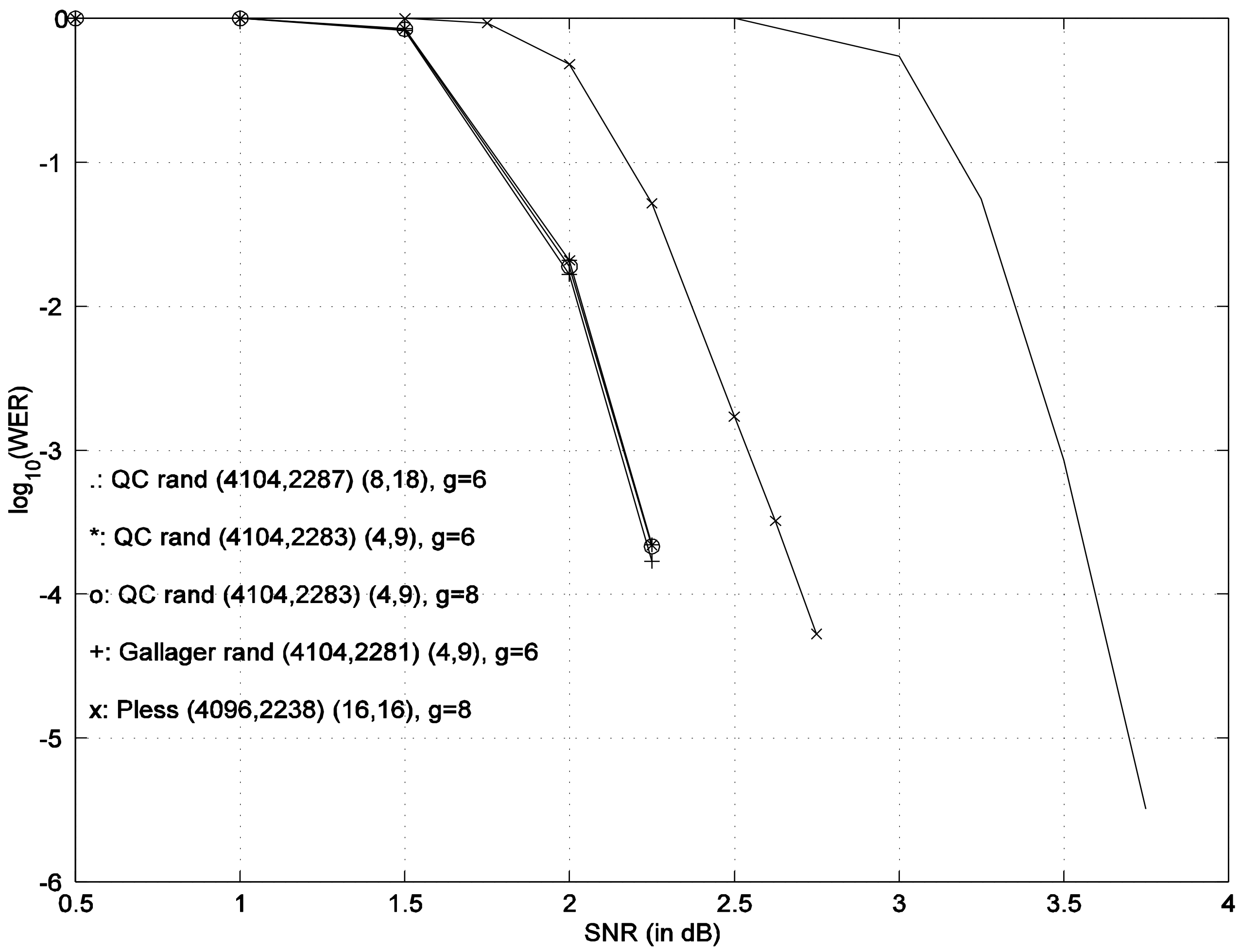
$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 3 \end{bmatrix}$$

- A (J,L) quasi cyclic (QC) LDPC code is totally defined by $(J-1)(L-1)$ integers.
- The quasi cyclic structure allows simple encoding based on shift registers.
- Girth at most 12 and minimum distance at most $(J+1)!$

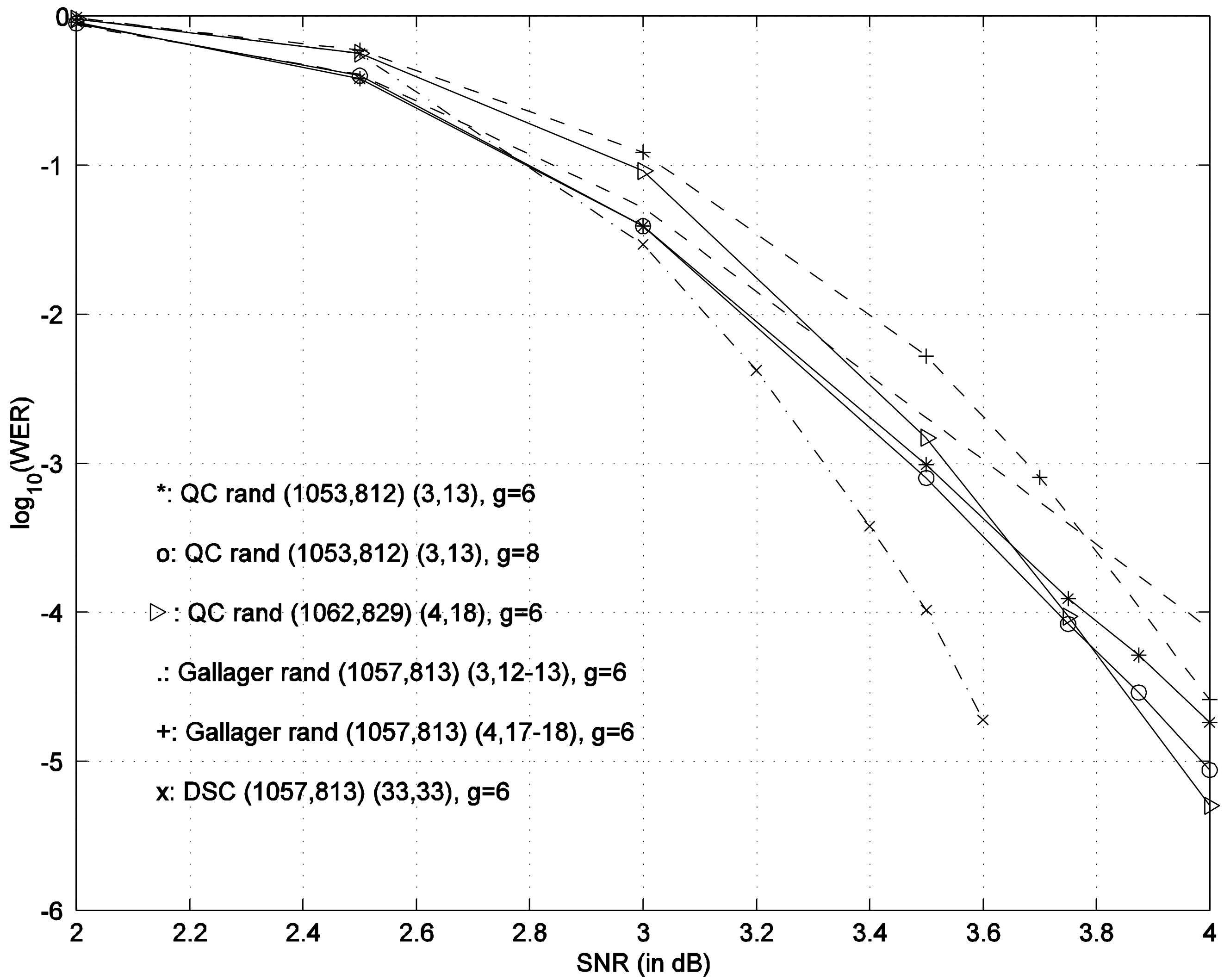
Example:

1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0	0

Rate-0.55 Length-4100 Codes:



Rate-0.77 Length-1050 Codes:



Lifted quasi-cyclic LDPC codes:

- Start with (J,L) M_1 x N_1 small H_b matrix of girth g at least 6.
- Replace every 1 by N_2 x N_2 circulant matrix.
- We obtain a (J,L) LDPC code with:

length $N = N_1 N_2$

co-dimension at most $M_1 N_2$

girth at least g.

RA-type LDPC codes:

$$H = \left[\begin{matrix} \\ \\ \quad H_1 \quad \\ \\ \\ \end{matrix} \quad \underbrace{\begin{matrix} 1 & & & & & \\ 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & 1 & \cdots & & \\ & & & & 1 & \\ & & & & 1 & 1 \end{matrix}}_{\text{linear time encodable.}} \right]$$

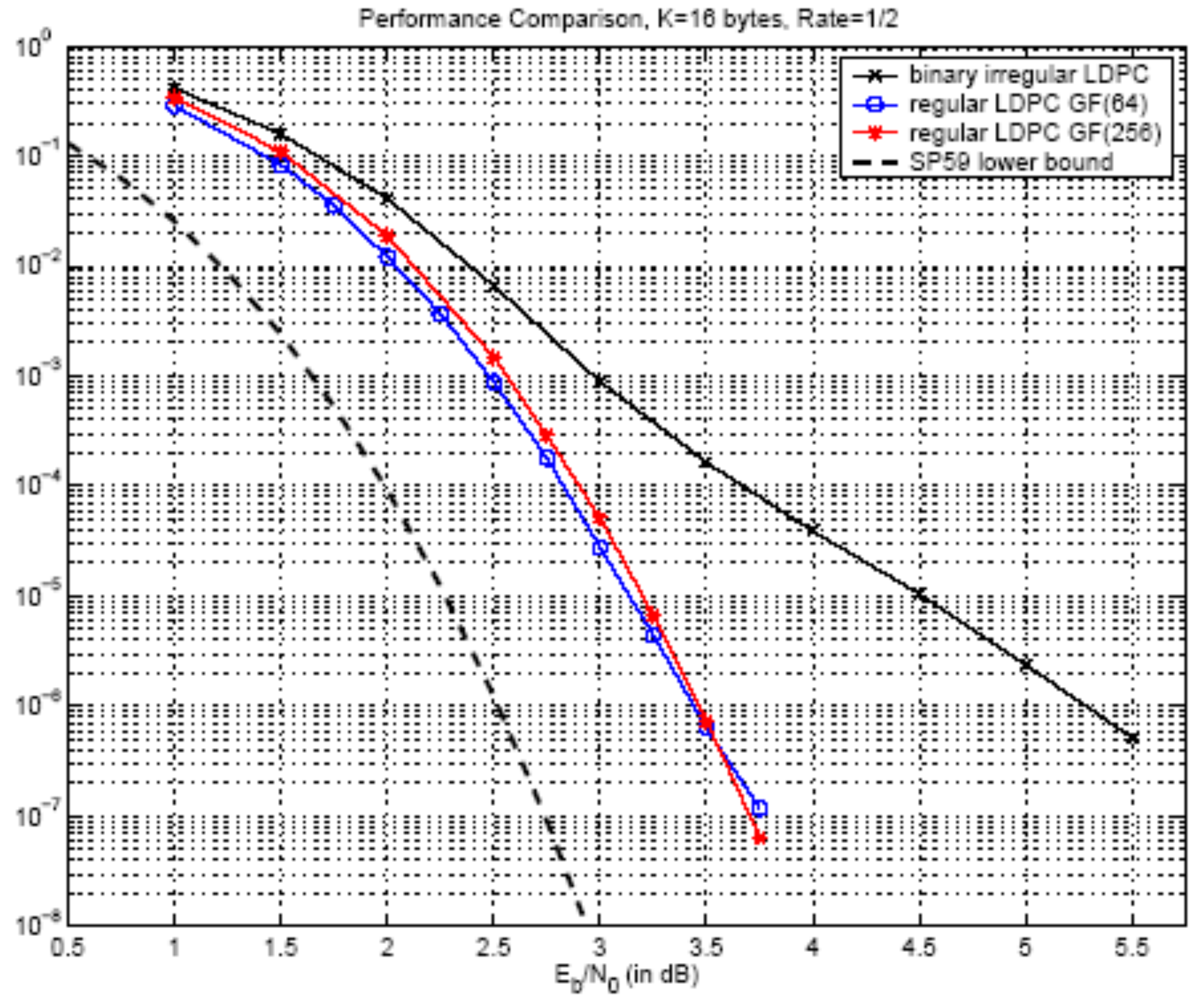
LDPC codes over GF(q):

- In H, $h_{ij} \in \mathrm{GF}(q)$; i.e. each edge is labeled by a symbol of GF(q) - ~ rotation -
- Check sum-i:

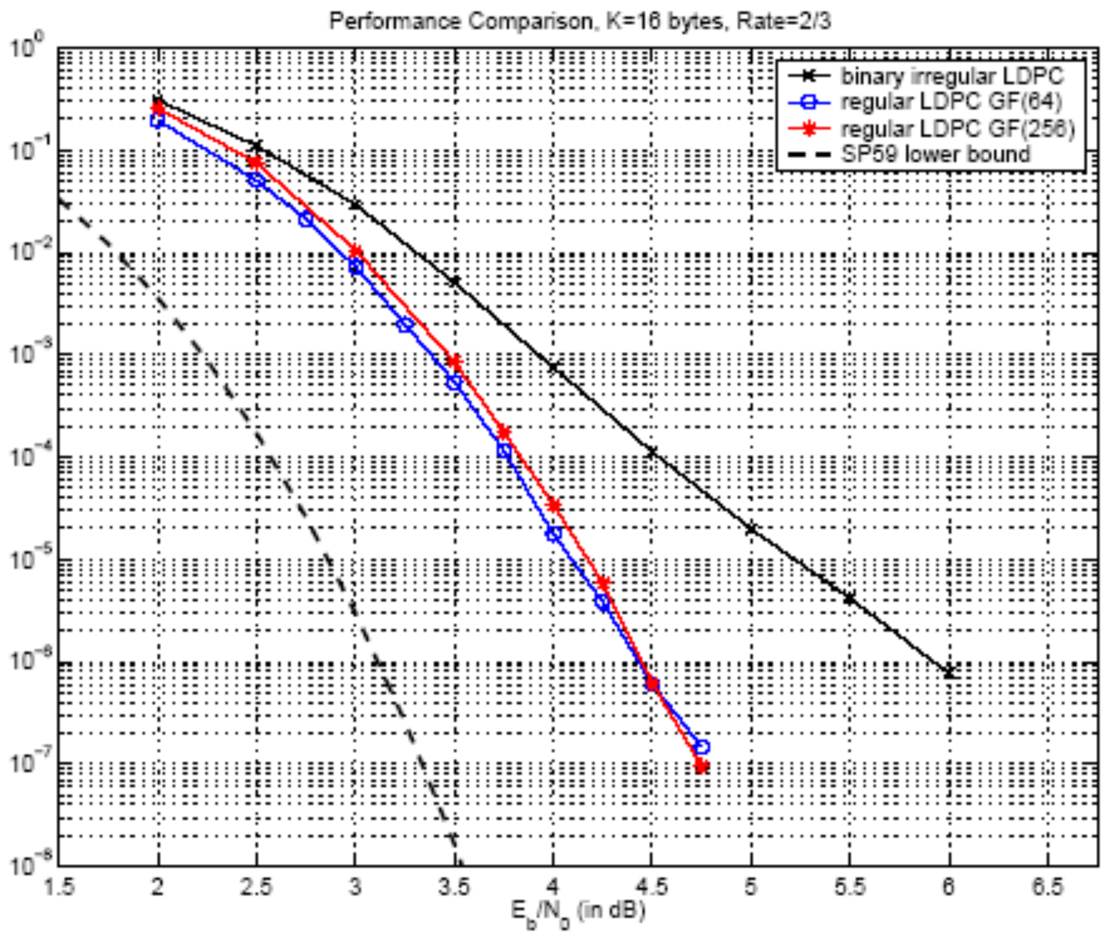
$$\sum_j h_{ij} x_j = 0$$

$$h_{ij} \in \mathrm{GF}(q),\ x_j \in \mathrm{GF}(q)$$

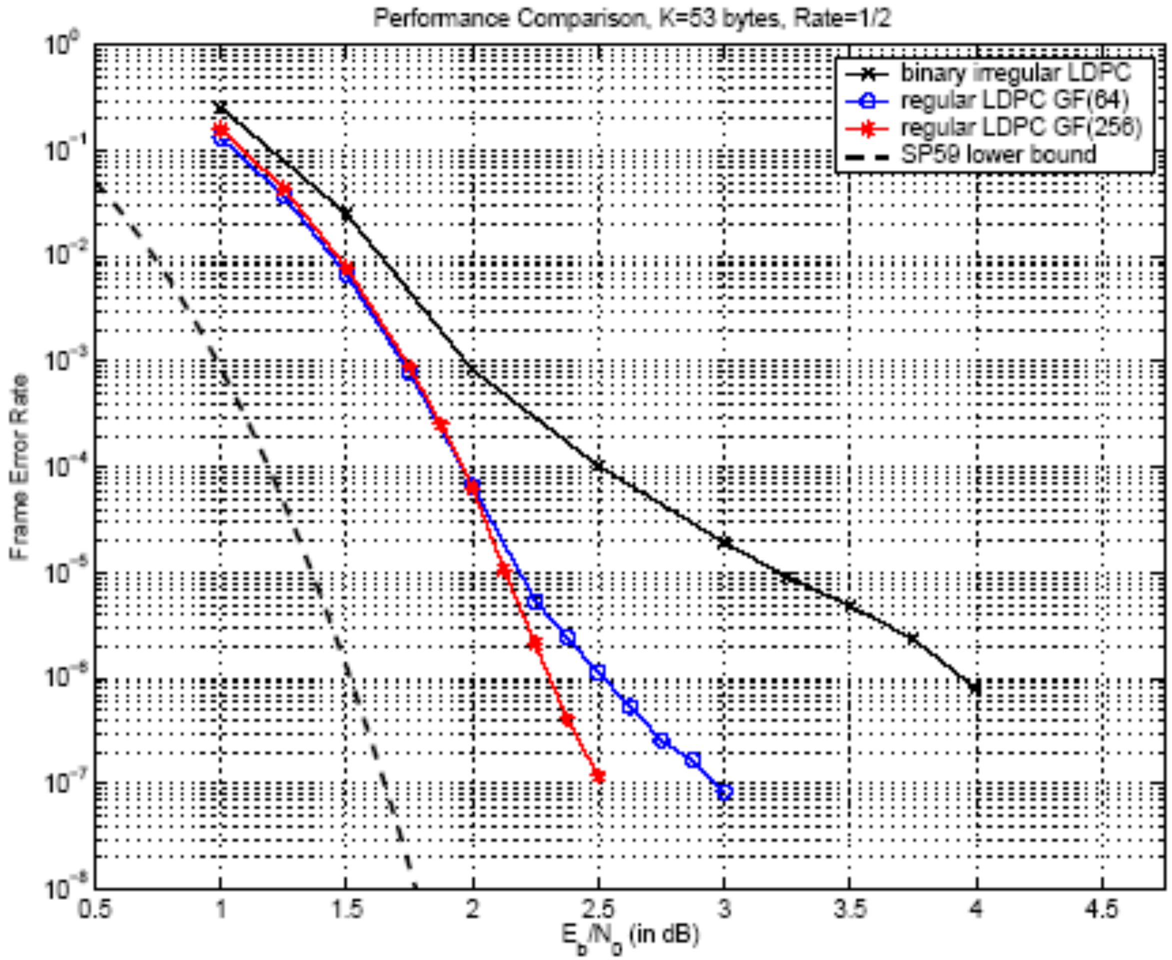
Results for small lengths



Results for small lengths

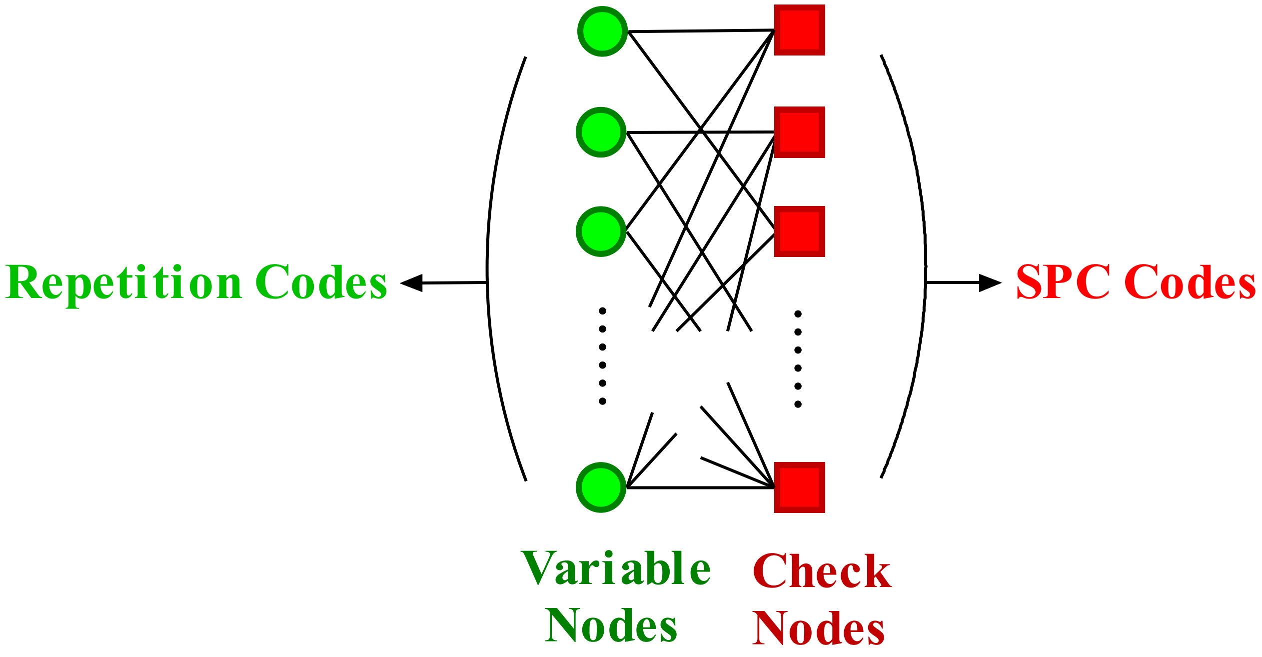


Results for medium lengths

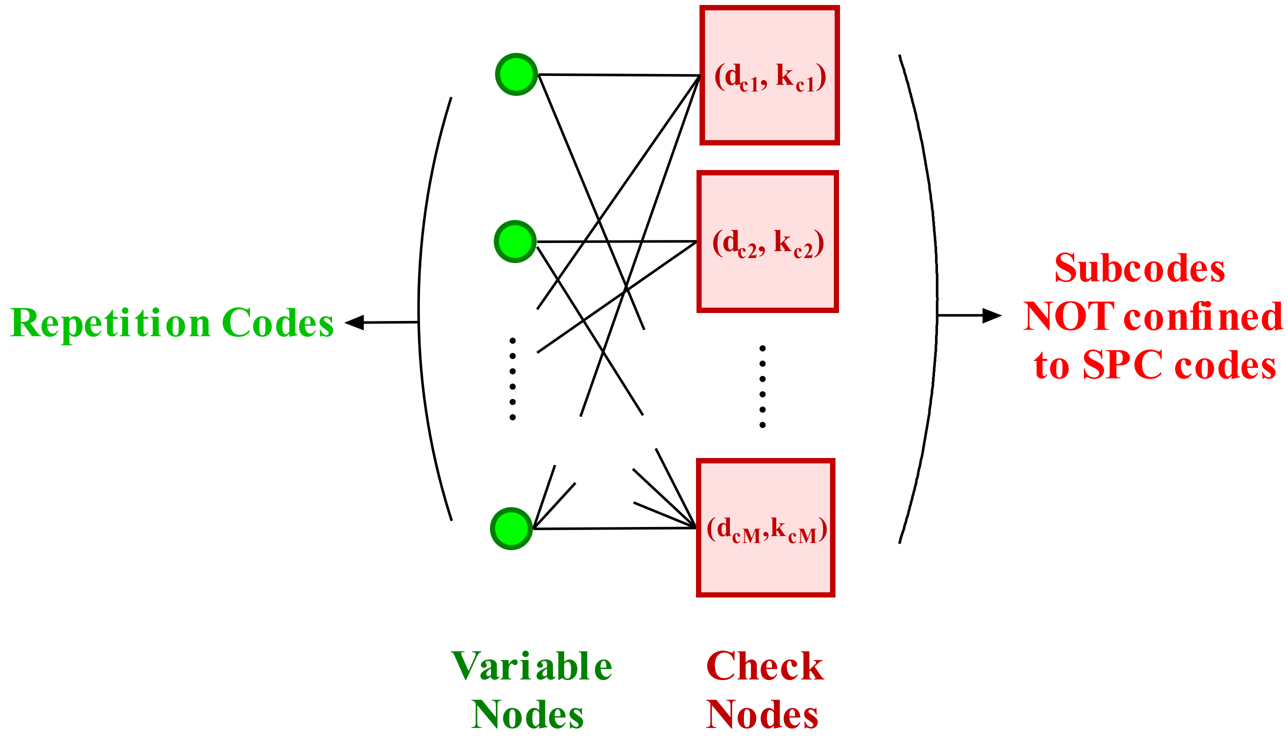


Generalized LDPC codes:

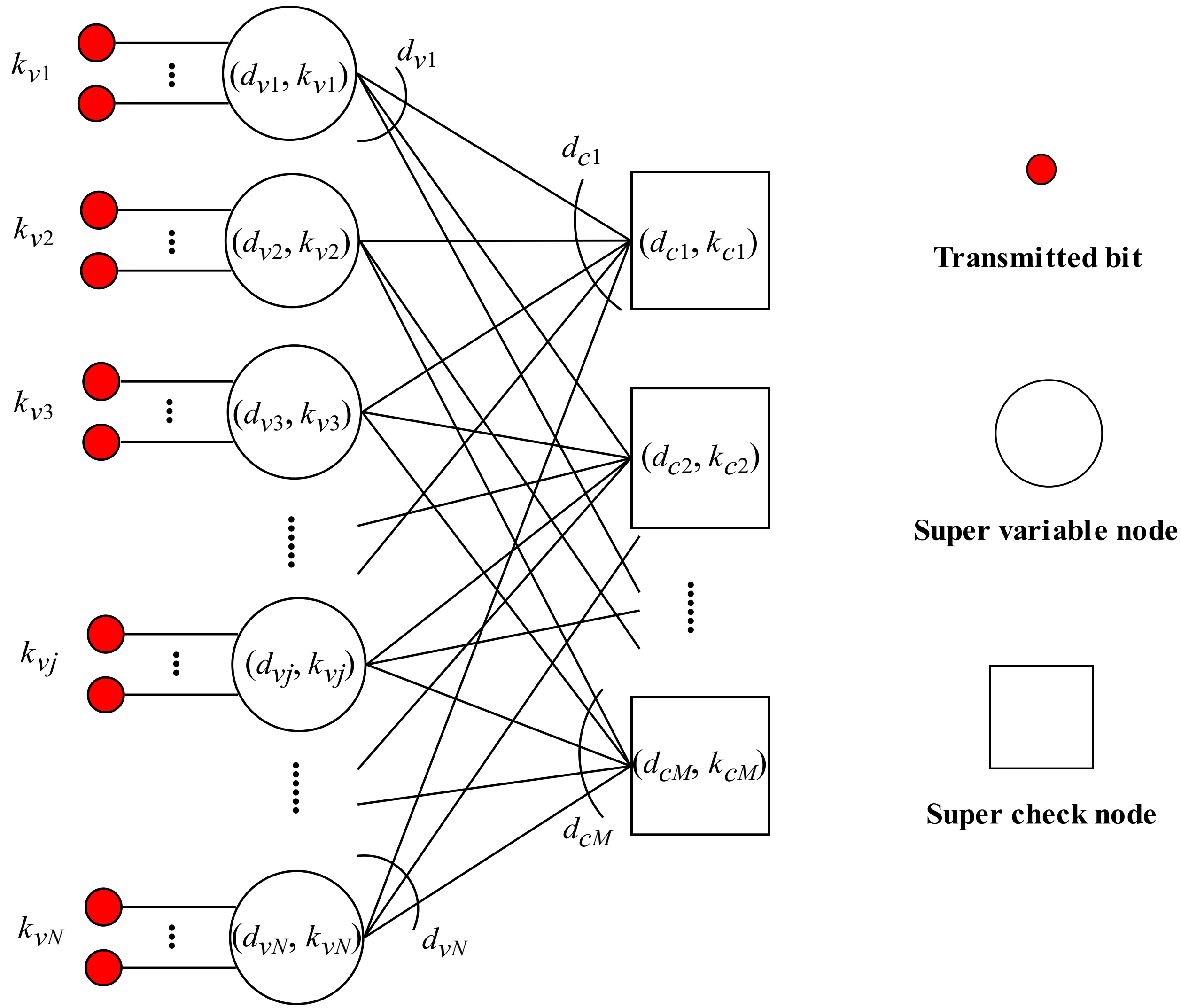
- A standard LDPC code is characterized by the random connection between variable nodes and check nodes.



- **Generalized LDPC codes are obtained by replacing (*dc*, *dc*-1) SPC with other (*dc*, *k*) subcodes.** **[Tanner–IT81]**



doubly-GLDPC codes



Construction steps:

Step 1: row expansion

In every row of parity check matrix, each “1” is replaced with a subcolumn from the subcode parity check matrix of the corresponding super check node based on a one-to-one correspondence and each “0” is replaced with a zero subcolumn.

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

Subcode $\mathbf{H}_{(7,4)\text{Ham}} = \begin{pmatrix} 1&0&0&0&1&1&1 \\ 0&1&0&1&0&1&1 \\ 0&0&1&1&1&0&1 \end{pmatrix}$

$$\begin{matrix} 1&1&1&0&1&0&0&0&0&0&0&0&0&0 \\ 1&0&1&1&0&0&1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&1&1&0&0&0&0&0&0&0 \end{matrix}$$

Construction steps (continued)

Step 2: column expansion

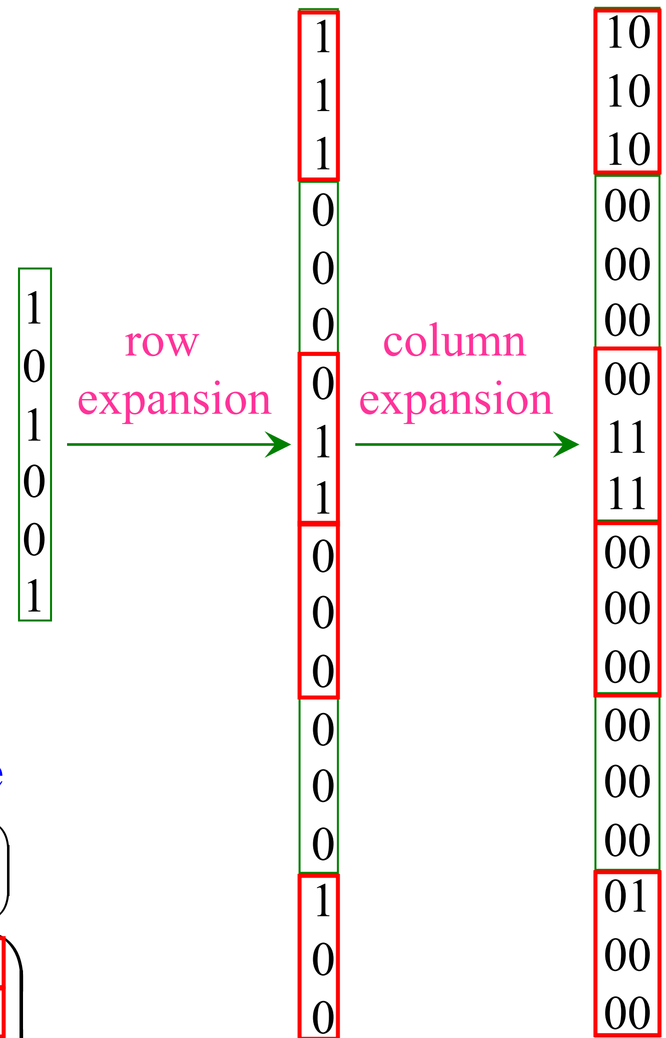
In every column of parity check matrix each “1” in the same subcolumn is replaced with the same subrow in the transposed generator matrix of the corresponding super variable node based on a one-to-one correspondence and each “0” in a subcolumn is replaced with a zero subrow.

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

Subcode

$$\mathbf{G}_1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\mathbf{G}_1^T = \begin{pmatrix} 10 \\ 11 \\ 01 \end{pmatrix}$$

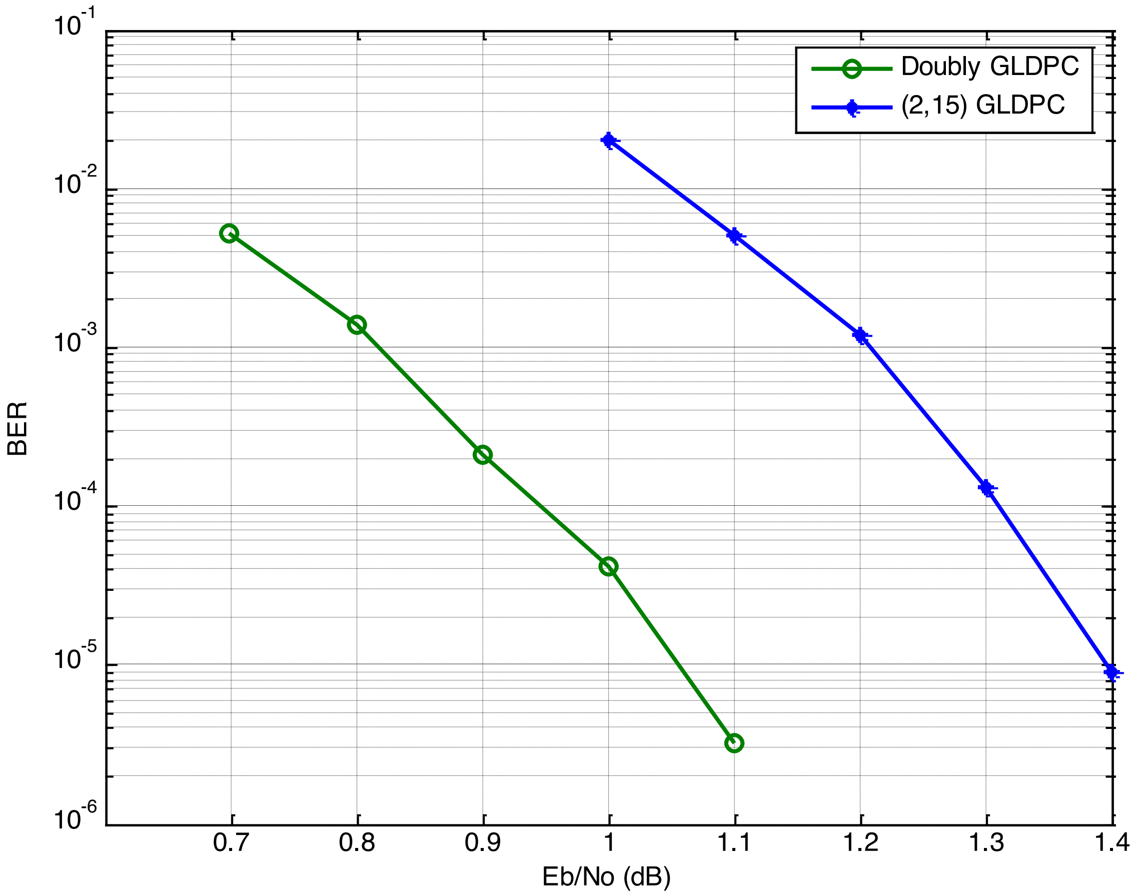


Construction of DGLDPC code *C1*

- Target: obtain good threshold
- C_1 is a rate-7/15 length-7650 code.
- Super variable nodes: (6,1) repetition code, (6,2) code with generator matrix $\begin{pmatrix}111100\\001111\end{pmatrix}$, (6,4) code with generator matrix $\begin{pmatrix}111000\\011100\\001110\\000111\end{pmatrix}$, (6,5) SPC code.
- Super check node: (15,11) Hamming code
- Variable node distribution is $\lambda_1 = 0.425$, $\lambda_2 = 0.075$, $\lambda_3 = 0.075$, and $\lambda_4 = 0.425$.
- Threshold is 0.3dB, only 0.26dB away from capacity.

Simulation Result of *C1*

The (2, 15) GLDPC code, which is used to compare with C_1, has the same kind of check node as C_1, i.e., (15,11) Hamming codes. The simulation result of the (2, 15) GLDPC code is obtained from [Lentmaier *et al*.-CL99].

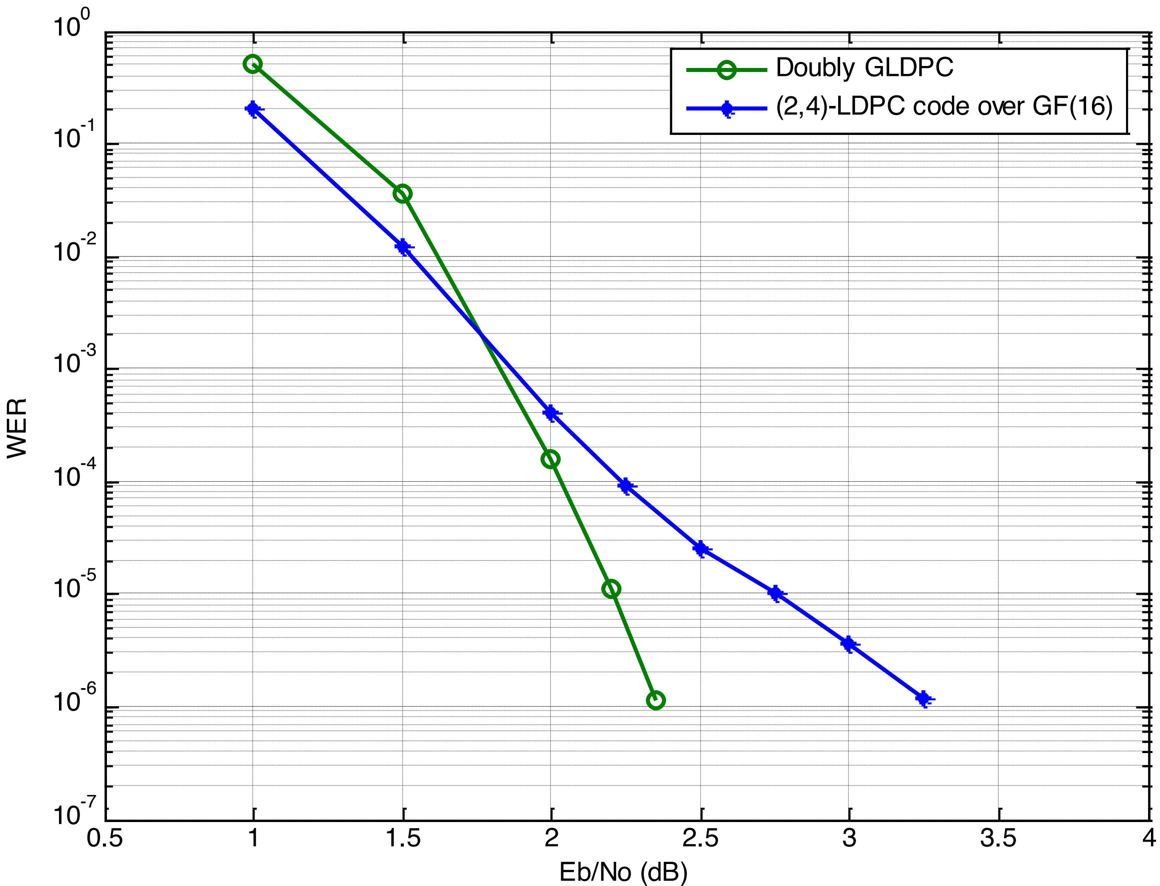


Construction of DGLDPC code *C2*

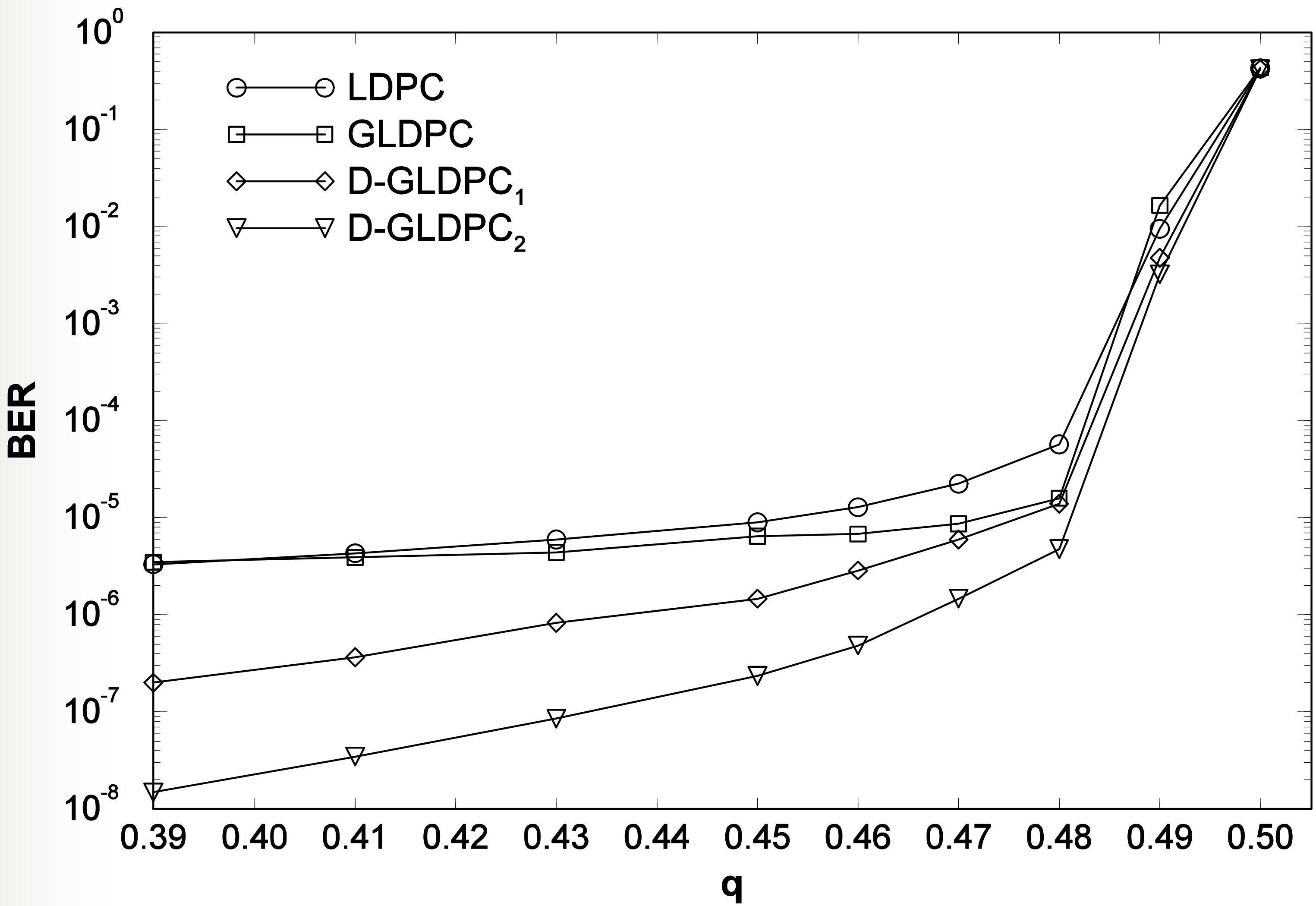
- Target: lower error floor
- C_2 is a rate-1/2 length-1536 code.
- Super variable nodes: the (4,1) repetition code and the (4,3) SPC code.
- Super check node: (15,11) Hamming code
- Threshold is 0.77dB.

Simulation Result of *C2*

The rate-1/2 length-1504 (2,4)-LDPC code over GF(16) is used to compare with C_2. The simulation result of this (2,4)-LDPC code is obtained from [Poulliat *et al*.-ISTC 2006].



Random codes performance comparison on BEC

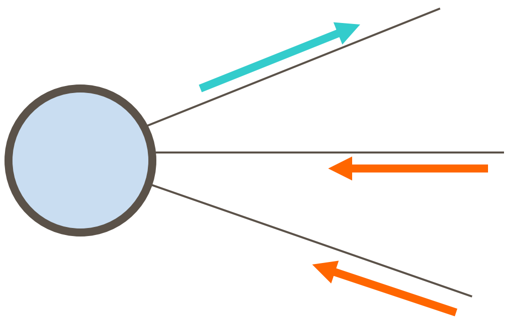


Part-V: Iterative Decoding of LDPC Codes

Marc Fossorier
Department of Electrical Engineering
University of Hawaii

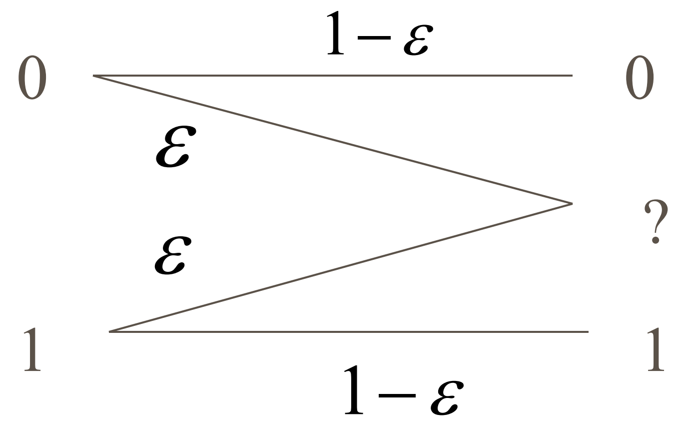
General concept:

- Each bit/check node is a processor, receiving messages from neighbor nodes, and sending back messages after processing.



Main goal: avoid direct correlation assuming incoming messages are independent of each other.

Iterative Decoding on BEC:



- MLD: Find information set (K independent positions) without erasures and perform Gaussian elimination: $O(N^3)$.
- Iterative decoding: Propagate information available at each node.

Processing in bit nodes:

- If node of degree-J, J+1 copies of bit available:

  J estimates from check nodes.

  1 estimate from channel.

- Define $x_l = \text{Prob}(\text{message} = \text{"?" at bit node for it - } l)$

  $y_l = \text{Prob}(\text{message} = \text{"?" at check node for it - } l)$

- Transmitted information still erasure if all other incoming message and initial estimate from channel are erasures:

$$x_{l+1} = \varepsilon y_l^{J-1}$$

Processing in check nodes:

- If node of degree-L, L incoming bits sum to 0.
- Transmitted information still erasure if at least one incoming message is an erasure:

$$y_l = 1 - (1 - x_l)^{L-1}$$

Combining the two equations:

$$x_{l+1} = \varepsilon\,[1-(1-x_l)^{L-1}]^{J-1}$$

- Threshold: largest value of ε such that

  $x_l \longrightarrow 0$ ``with l large enough”.

  ($x_{l+1} \longrightarrow x > 0$ possible)

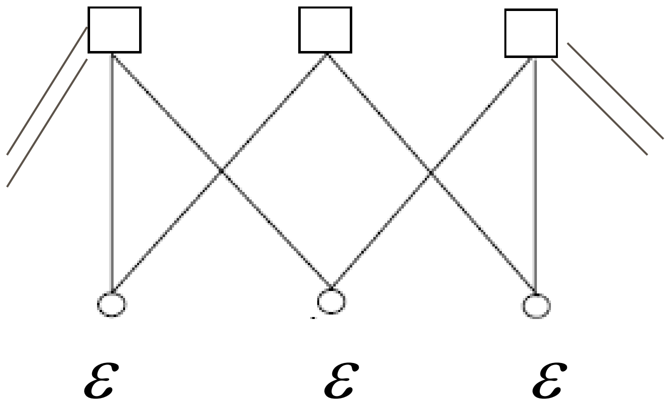
For irregular codes:

$$x_{l+1} = \varepsilon\, \lambda[1 - \rho(1 - x_l)]$$

- Capacity achieving codes of rate R= 1- ε have been found for the BEC (ex: heavy tail Poisson distribution)

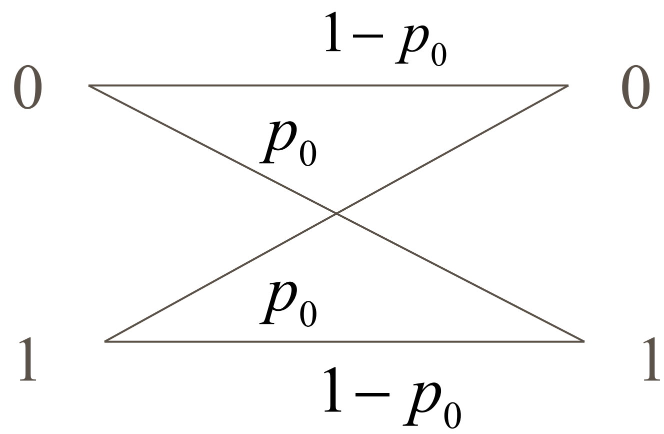
Finite length issues:

Stopping set: subset V of variable nodes such that all neighbors are connected to V at least twice.



These are poor configurations as iterative decoding stuck even if MLD possibly correct

Iterative Decoding on BSC:



- MLD: NP_hard problem.
- Iterative decoding: Propagate information available at each node.

Gallager algorithm-A:

- At iteration-(i+1), send to check node initial value received from channel, unless (J-1) other check values disagree with it.

- Define $$P^{(i)} = \text{Prob(check sum returns an error for it - } i\text{)}$$

- Ways to make an error:

  (1) bit received in error and less than J-1 check sums indicate otherwise.

$$p_0 \; (1-(1-P^{(i)})^{J-1})$$

  (2) bit received correctly and all J-1 check sums indicate otherwise.

$$(1-p_0) \; {P^{(i)}}^{J-1}$$

- It follows:

$$p_{i+1} = p_0 \, (1-(1-P^{(i)})^{J-1}) + (1-p_0) \, {P^{(i)}}^{J-1}$$

- Check sums of weight L indicates an error if L-1 other bits contain odd number of errors:

$$P^{(i)} = \sum_{j \text{ odd}} \binom{L-1}{j} p_i^{\,j} (1-p_i)^{L-1-j}$$

$$= \frac{1-(1-2p_i)^{L-1}}{2}$$

- A necessary and sufficient condition for $p_{i+1} < p_i$:

$$p_0 \left(1-P^{(i)}\right)^{J-1} > (1-p_0)\, {P^{(i)}}^{J-1}$$

- This equation can be used to determine the largest value of p_0 such that $p_{i+1} < p_i$ for i large enough.
- To this end it assumes the incoming messages are independent. On a Tanner graph of girth g, it is true for $\left\lfloor \frac{g-2}{4} \right\rfloor$ iterations.

(~g/2 branches to reach 2 opposite nodes on a cycle and 2 branches per iteration).

Gallager algorithm-B:

- At iteration-(i+1), send to check node initial value received from channel, unless $T(i)$ other check values disagree with it.

  $T(i)$ is a threshold associated with iteration-i.

- Using same reasoning as for alg-A, we obtain:

$$p_{i+1} = p_0 - p_0 \sum_{l=T(i)}^{J-1} \binom{J-1}{l} (1-P^{(i)})^l {P^{(i)}}^{J-1-l}$$

$$+ (1-p_0) \sum_{l=T(i)}^{J-1} \binom{J-1}{l} {P^{(i)}}^l (1-P^{(i)})^{J-1-l}$$

- The optimum theoretical threshold is the smallest T that satisfies:

$$\frac{1-p_0}{p_0} \leq \left(\frac{1-P^{(i)}}{P^{(i)}} \right)^{2T-J+1}$$

- Alg-A is equivalent to Alg-B with $T(i) = J\text{-}1$ (hence alg-B always better).
- In practice, $T(i)$ adjusted from simulation.

Iterative Decoding on AWGN:

- $\mathbf{y} = \mathbf{x} + \mathbf{n}$ with $x_i = (-1)^{c_i}$ and $n_i = N(0, N_0/2)$
- Define: $N(m) = \{n : h_{mn} = 1\}$

$$M(n) = \{m : h_{mn} = 1\}$$

For (J,L) regular code: $|N(m)| = L$; $|M(n)| = J$.

Define:

$$p_0 = P(y_i \mid c_i = 0) = (\pi N_0)^{-1/2} \mathrm{e}^{-(y_i-1)^2/N_0}$$

$$p_1 = P(y_i \mid c_i = 1) = (\pi N_0)^{-1/2} \mathrm{e}^{-(y_i+1)^2/N_0}$$

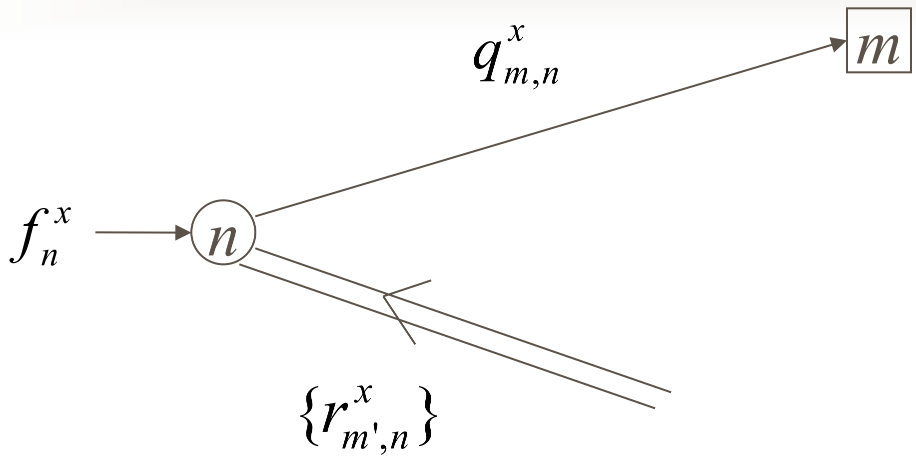
$r_{m,n}^{x}$: Probability that bit - n is x based on other bits n' in $N(m)\setminus n$ which have probabilities $q_{m,n'}^{x}$.

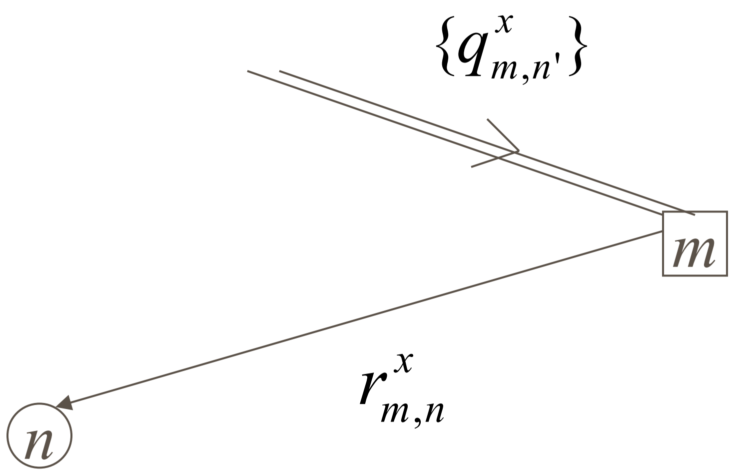
$q_{m,n}^{x}$: Probability that bit - n is x based on f_n^x and the other probabilitiess $r_{m',n}^{x}$ for bit - n in $M(n)\setminus m$.

f_n^x : Probability that bit - n is x based y_n .

$$f_n^0 = p_0/(p_0+p_1); \quad f_n^1 = p_1/(p_0+p_1)$$

q_n^x : Probability that bit - n is x based on f_n^x and the other probabilitiess $r_{m',n}^{x}$ for bit - n in $M(n)$.


$q^x_{m,n}$
m
f^x_n
n
$\{r^x_{m',n}\}$


$\{q^x_{m,n'}\}$
m
$r^x_{m,n}$
n

Belief Propagation (BP) Algorithm:

- BP algorithm is an iterative decoding algorithm [Gallager-IRE62, MacKay-IT99] .

  - Messages can be probabilities, and more conveniently, log-likelihood ratios (LLR's) for binary LDPC codes.

Initialization : $q^0_{m,n} = f^0_n;\ q^1_{m,n} = f^1_n.$

Horizontal step :

$$r^0_{m,n} = 1/2\left(1 + \prod_{n'\in N(m)\backslash n}\left(q^0_{m,n'} - q^1_{m,n'}\right)\right)$$

$$r^1_{m,n} = 1/2\left(1 - \prod_{n'\in N(m)\backslash n}\left(q^0_{m,n'} - q^1_{m,n'}\right)\right)$$

Vertical step :

$$q^0_{m,n} = \alpha_{mn}\ f^0_n \prod_{m'\in M(n)\backslash m} r^0_{m',n}$$

$$q^1_{m,n} = \alpha_{mn}\ f^1_n \prod_{m'\in M(n)\backslash m} r^1_{m',n}$$

$$\alpha_{mn} : q^0_{m,n} + q^1_{m,n} = 1.$$

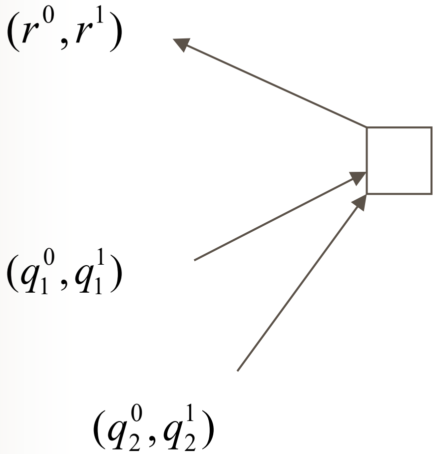
Decision :

$$q_n^0 = q_{m,n}^0 \; r_{m,n}^0$$

$$q_n^1 = q_{m,n}^1 \; r_{m,n}^1$$

Stopping criterion : Stop as soon as hard decision is a codeword.

Decoding in log - domain more stable numerically.



$$r^0 = q_1^0 \; q_2^0 + q_1^1 \; q_2^1 \quad \sim \; (0+0 \;\; \text{or} \;\; 1+1)$$

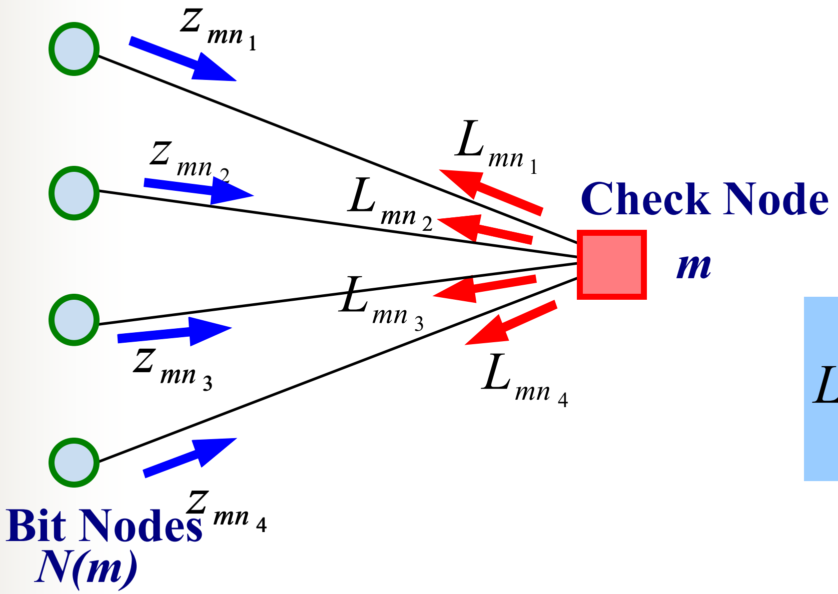
$$r^1 = q_1^0 \; q_2^1 + q_1^1 \; q_2^0 \quad \sim \; (0+1 \;\; \text{or} \;\; 1+0)$$

$$\begin{aligned}
& 1/2\left(1+\left(q_1^0-q_1^1\right)\left(q_2^0-q_2^1\right)\right) \\
&= 1/2\left(1+q_1^0 q_2^0+q_1^1 q_2^1-q_1^0 q_2^1-q_1^1 q_2^0\right) \\
&= 1/2\left(1+q_1^0 q_2^0+q_1^1 q_2^1-q_1^0 \;(1-q_2^0)-q_1^1 \;(1-q_2^1)\right) \\
&= 1/2\left(1+2q_1^0 q_2^0+2q_1^1 q_2^1-q_1^0 \;-q_1^1\right) \\
&= q_1^0 \; q_2^0+q_1^1 \; q_2^1
\end{aligned}$$

Processing in check nodes:

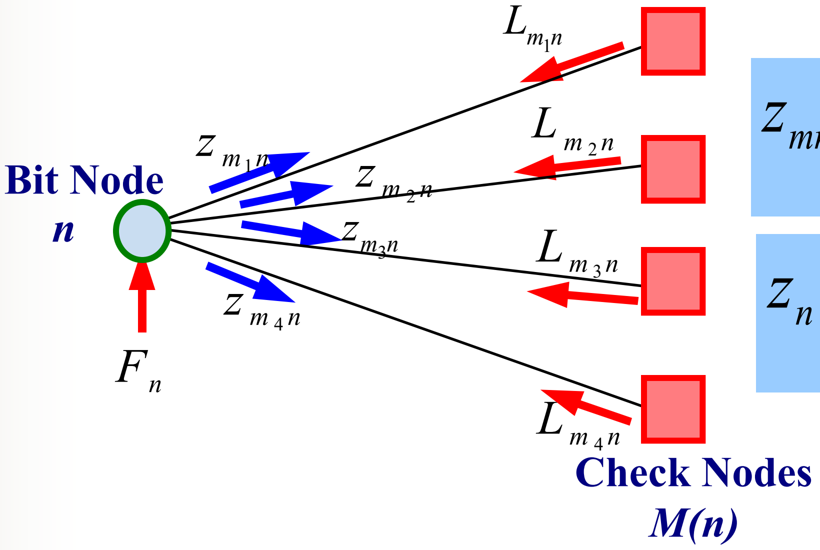
Principles:

incoming messages + constraints ⇒ outgoing messages



$$L_{mn} = 2\tanh^{-1}\left(\prod_{n'\in N(m)\backslash n} \tanh\left(z_{mn'}/2\right)\right)$$

Processing in bit nodes:

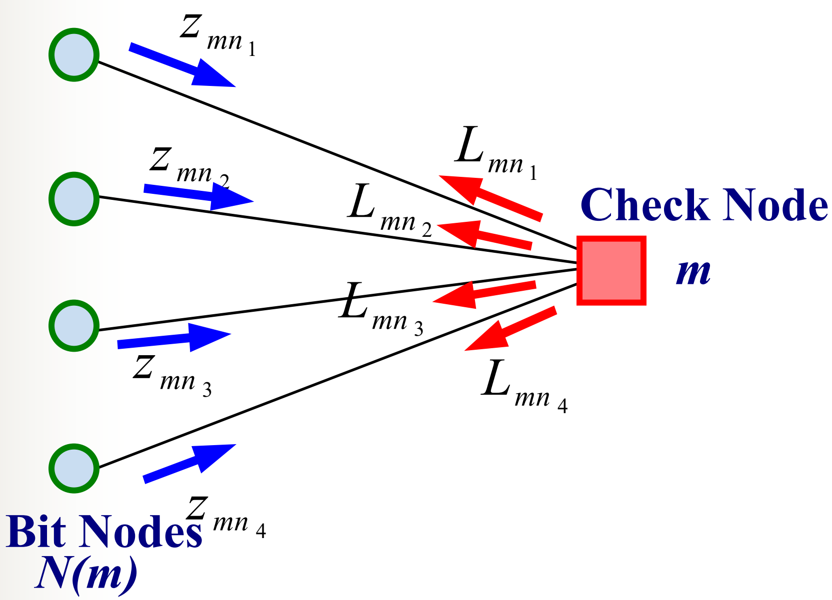


$$z_{mn} = F_n + \sum_{m' \in M(n) \backslash m} L_{m'n}$$

$$z_n = F_n + \sum_{m \in M(n)} L_{mn} \, ,$$ for hard decision

BP-Based Algorithm (min-sum)

—— simplification in check node processing



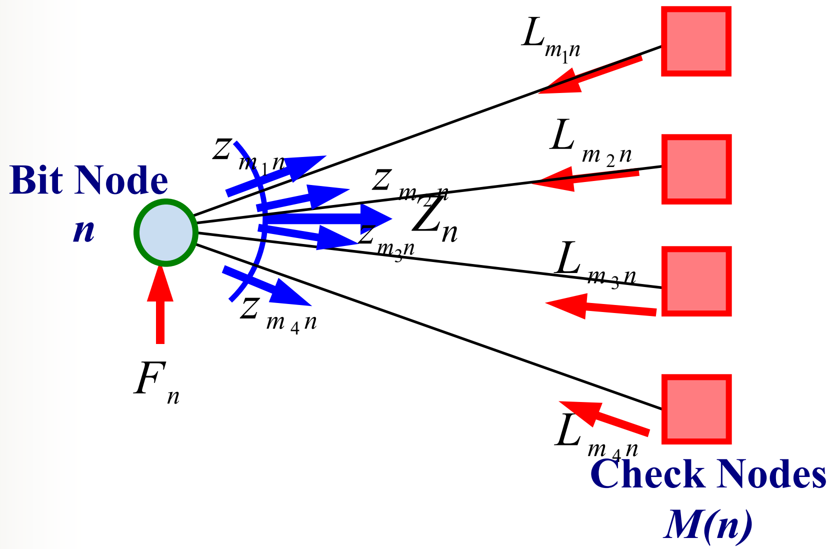
$$L_{mn} = 2\tanh^{-1}\left(\prod_{n' \in N(m)\backslash n} \tanh\left(z_{mn'}/2\right)\right)$$

$$\approx \prod_{n' \in N(m)\backslash n} \operatorname{sgn}(z_{mn'}) \cdot \min_{n' \in N(m)\backslash n} \left|z_{mn'}\right|$$

- Low complexity;
- Independent of channel characteristics for AWGN channels;
- Degradation in performance.

APP Algorithm

—— simplification in bit node processing

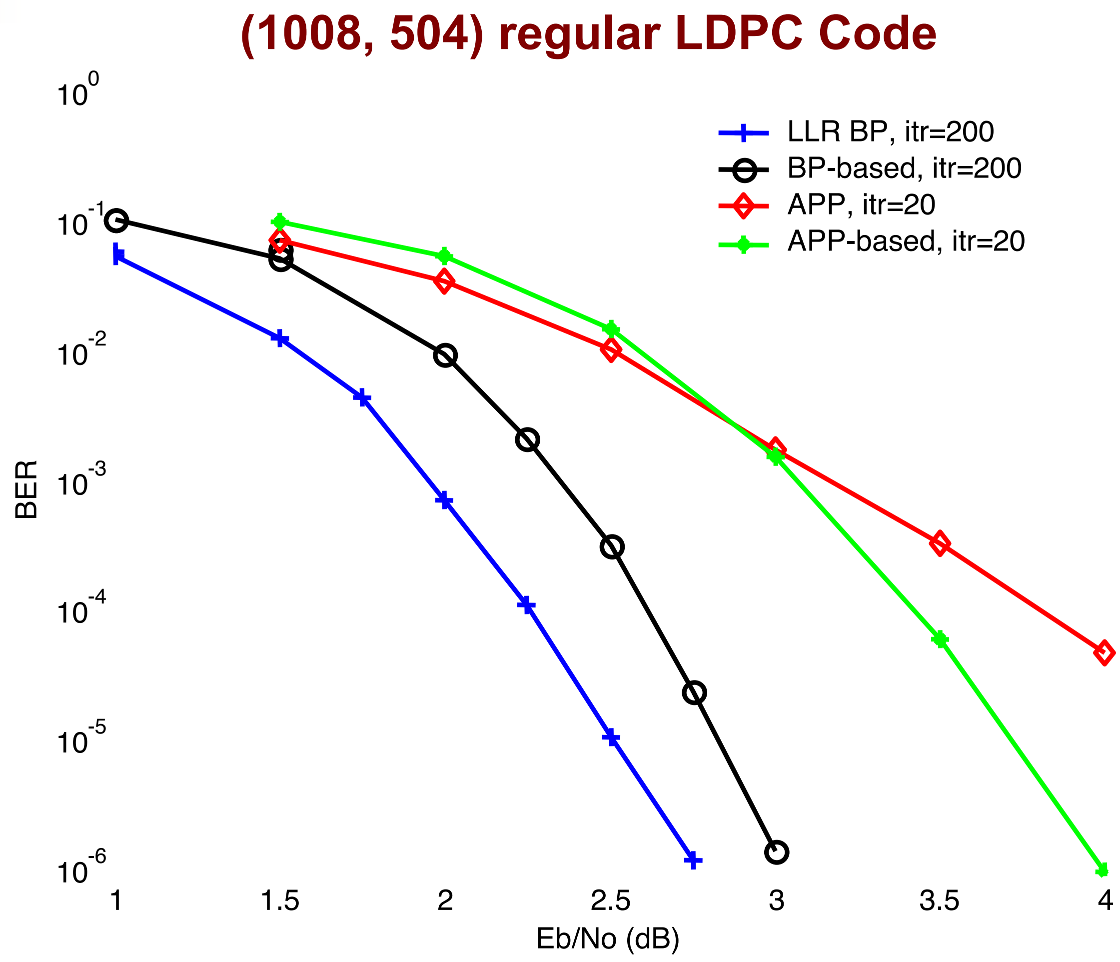


$$Z_n = F_n + \sum_{m \in M(n)} L_{mn}$$

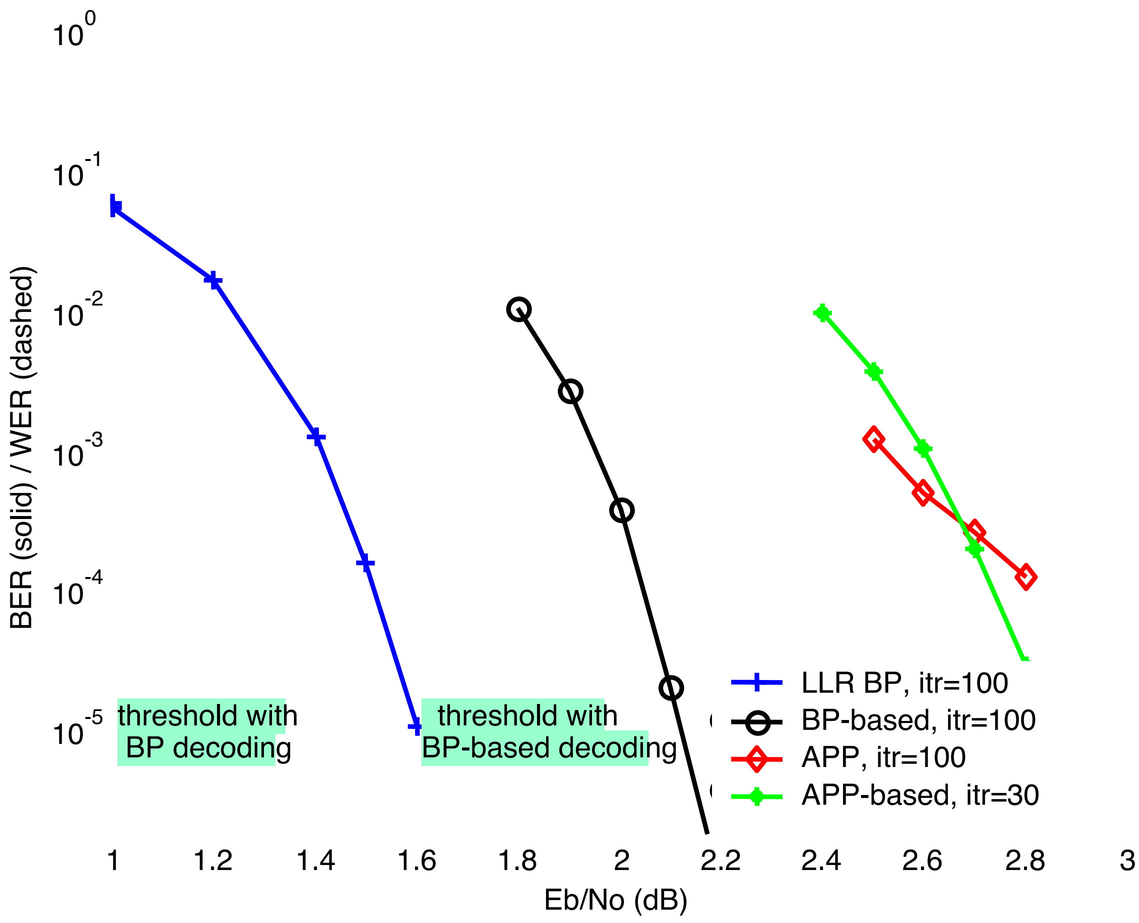
- Z_n is not only for hard decision, but also as a substitution for Z_{mn}.
- Lower computational complexity and storage requirement.
- Introducing correlation in the iterative decoding process.

APP-Based Algorithm —— simplification in both nodes

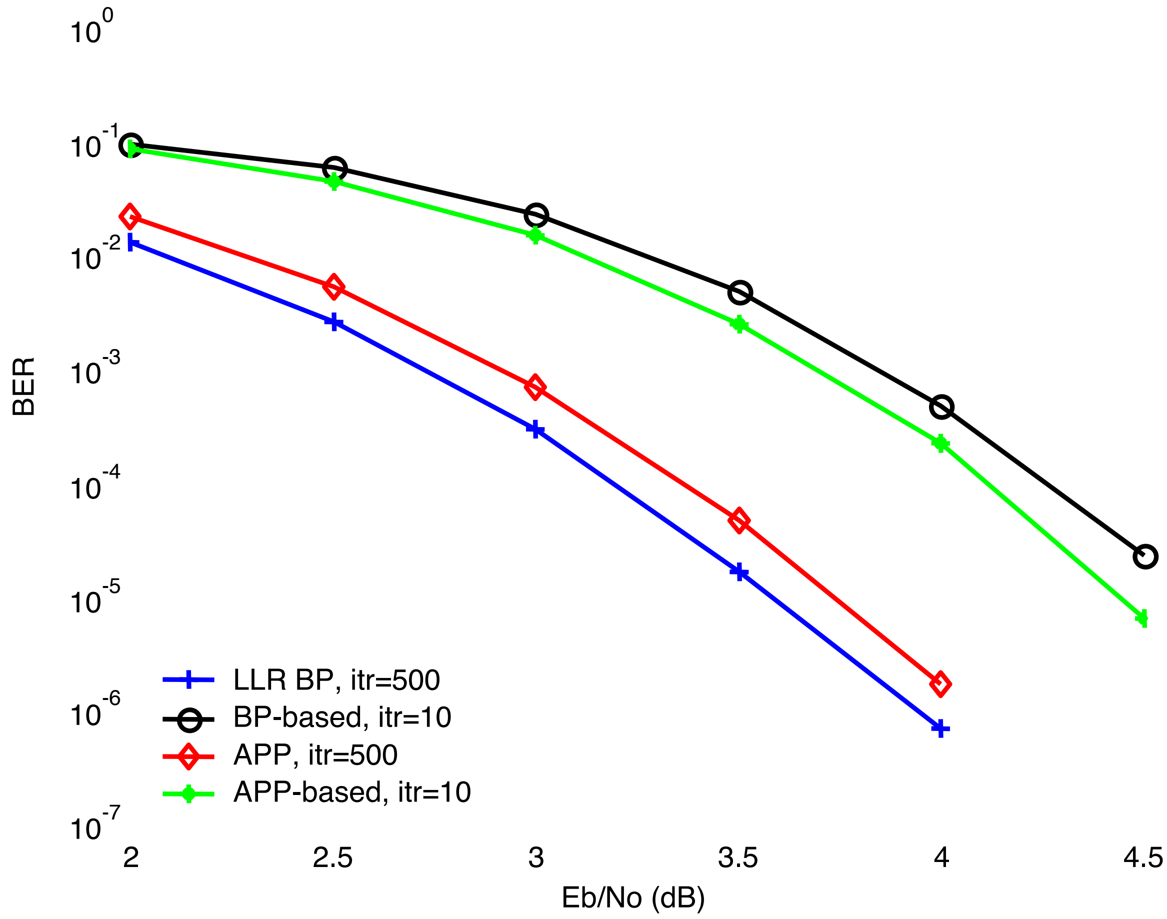
Performance of BP and Its Simplified Versions



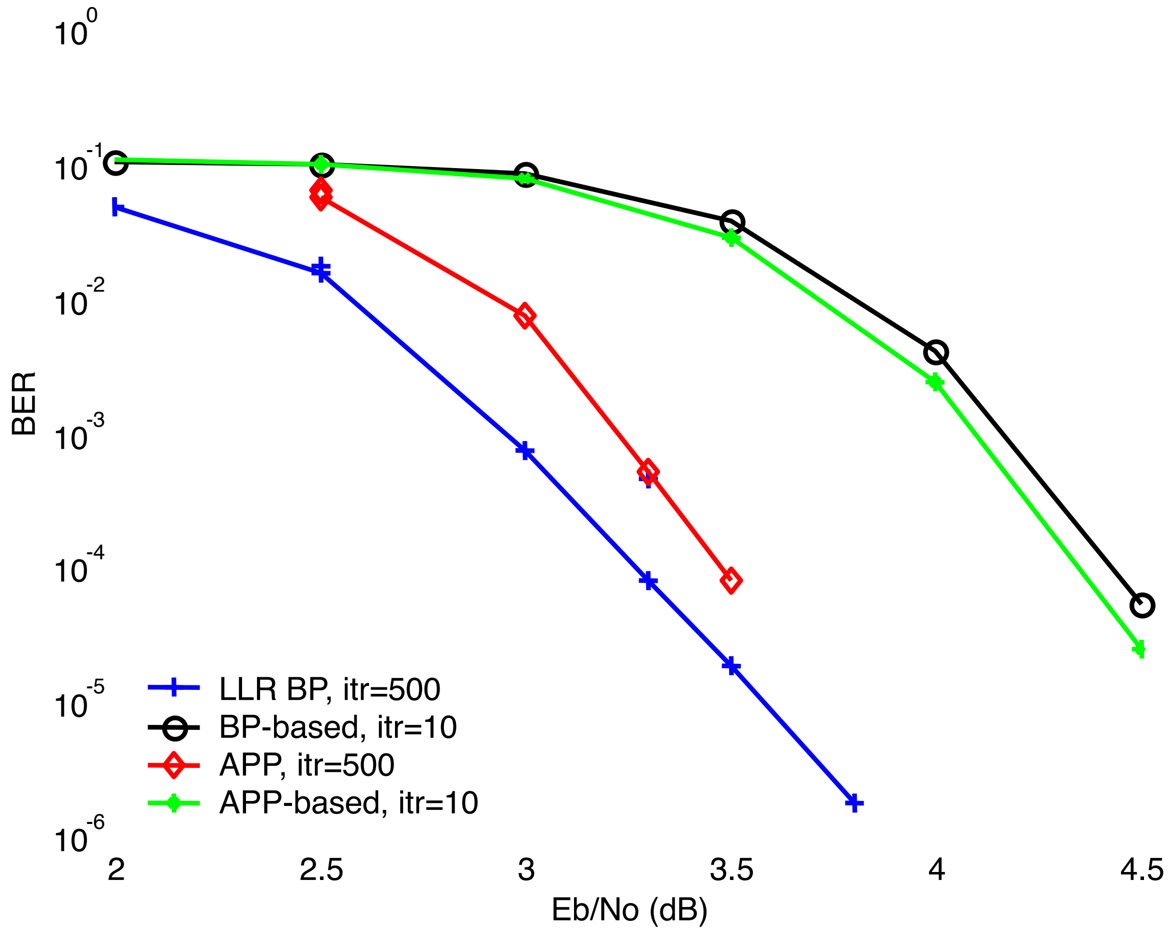
(8000, 4000) Regular LDPC Code



(273, 191) DSC Code

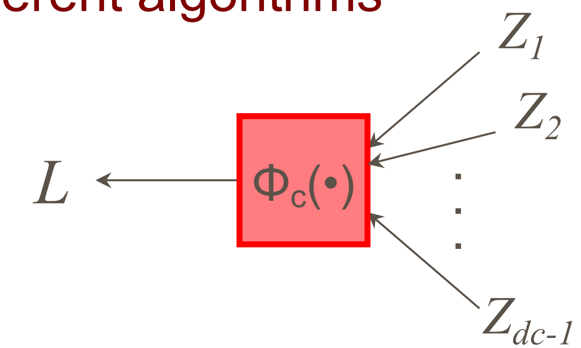


(1057, 813) DSC Code



Improvement of the BP-based algorithm

check node processing in different algorithms



BP:

$$L_1 = 2\tanh^{-1}\left(\prod_i \tanh(Z_i/2)\right)$$

BP-based:

$$L_2 = \prod_i \mathrm{sgn}(z_i) \cdot \min_i |Z_i|$$

Two statements hold:

1. $\mathrm{sgn}(L_1) = \mathrm{sgn}(L_2)$;
2. $|L_1| < |L_2|$.

Two improvements of the check node processing

Normalized BP-based algorithm:

Divide L_2 by a normalization factor α greater than 1,

$$L_2 \leftarrow L_2 / \alpha \quad .$$

Offset BP-based algorithm:

Decreasing $|L_2|$ by a offset value β ,

$$|L_2| \leftarrow \max(|L_2| - \beta, 0) \quad .$$

- Decoder parameters, α's or β's, need to be optimized.

Normalized APP-based algorithm

- APP-based algorithm + normalization in check nodes
  $\Rightarrow$ normalized APP-based algorithm.

Optimizing Parameters by Density Evolution

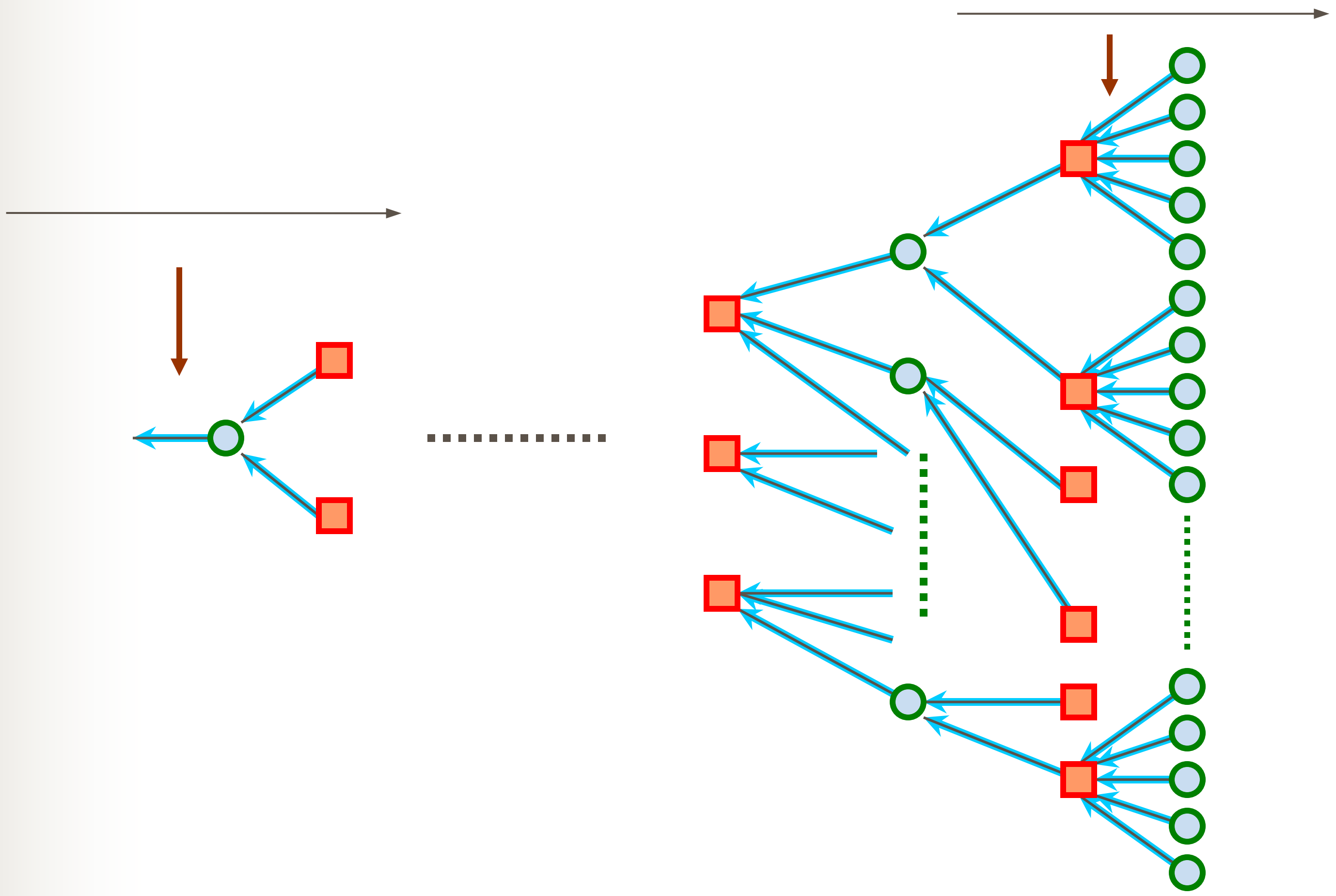
- **Density evolution** (DE) is a powerful tool to analyze message-passing algorithms of LDPC codes [Richardson-IT01].
- Assumptions:

  (1) symmetric channels (BSC, AWGN, ……);

  (2) decoder symmetry;

  (3) all-0 sequences transmitted;

  (4) infinite code length --- loop free.
- Basic idea: numerically derive the probability density functions (pdf) of the messages from one iteration to another, based on decoding algorithms, and then determine the bit error rate.



- Threshold phenomenon: for an ensemble of code, a certain kind of channels and a decoding algorithm, there exits a threshold for a channel parameter, such that the BER approaches to 0 with a channel parameter better than this threshold, and the BER stays away from zero with a worse channel parameter.
- Example:

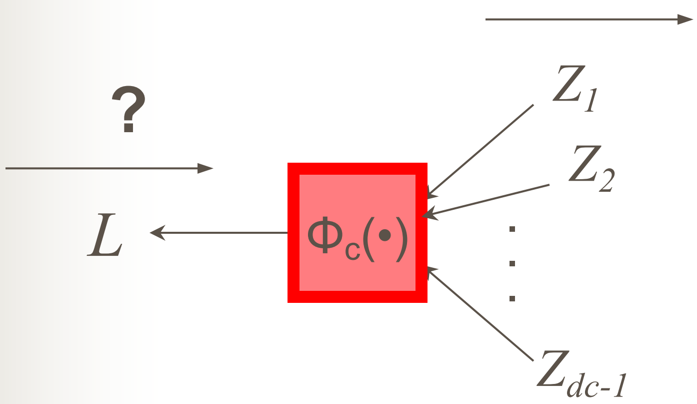
For AWGN channel with variance σ^2, BPSK transmission, BP as decoding algorithm, and $(J,L) = (3, 6)$

$\Rightarrow \sigma_T = 0.880$ (1.11 dB) **[Richardson-Urbanke-IT01]**.

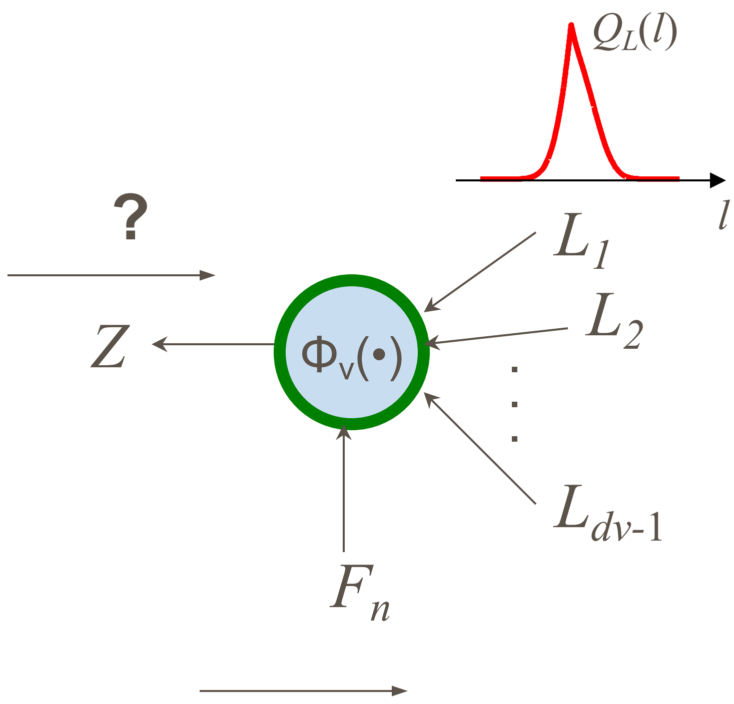
As a comparison, Shannon limit for BPSK is about 0.2 dB.

Density evolution algorithms

Check node processing:



Bit node processing:



Density evolution algorithms for BP and BP-based algorithms

(1) In bit nodes: *SAME*

- Only additions involved in both alogrithms.
- The output pdf is the convolution of the input pdf's.
- Can use FFT to speed up the computation.

(2) In check nodes: *DIFFERENT*

Due to different ways of processing

BP: $$L = 2\tanh^{-1}\left(\prod_i \tanh(Z_i/2)\right)$$

BP-based: $$L = \prod_i \mathrm{sgn}(Z_i) \cdot \min_i |Z_i|$$

DE for normalized and offset BP-based algorithms

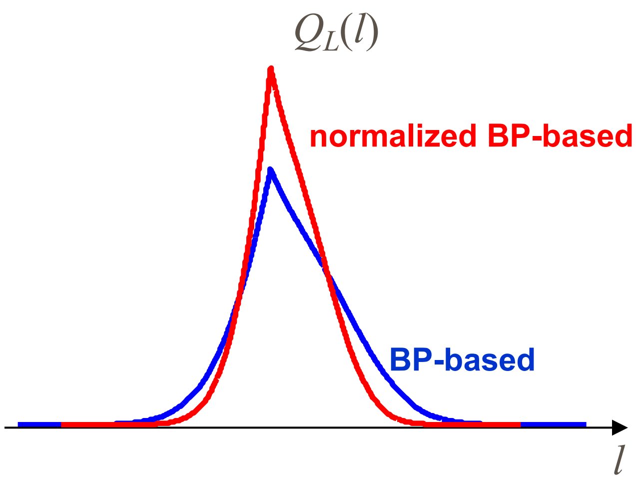
- Slightly modify the DE algorithm of the BP-based algorithm.

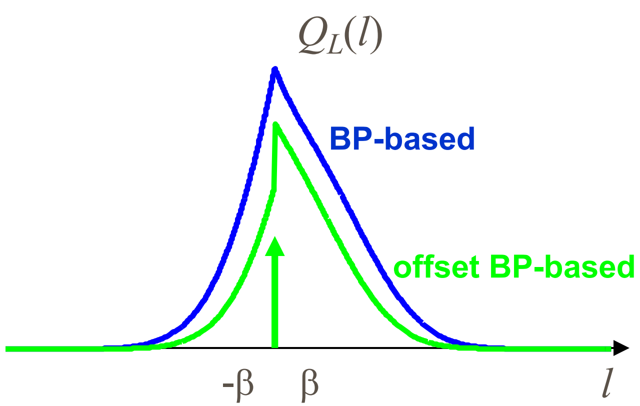
- Normalized BP-based

$$L \leftarrow L/\alpha$$
$$Q_L(l) \leftarrow \alpha\, Q_L(\alpha \cdot l)$$

- Offset BP-based

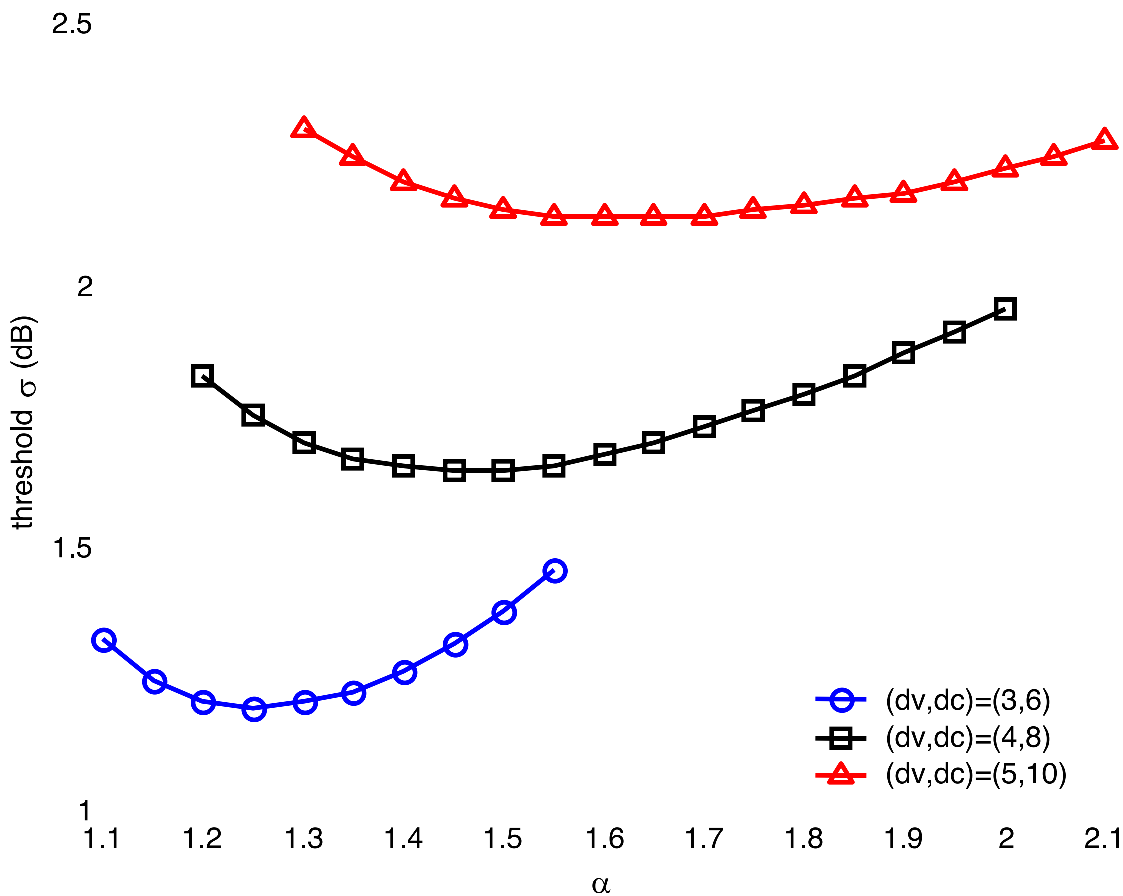
$$|L| \leftarrow \max(|L| - \beta, 0)$$
$$Q_L(l) \leftarrow u(l)\, Q_L(l+\beta) + u(-l)\, Q_L(l-\beta) + \delta(l)\int_{-\beta}^{\beta} Q_L(l)\,dl$$





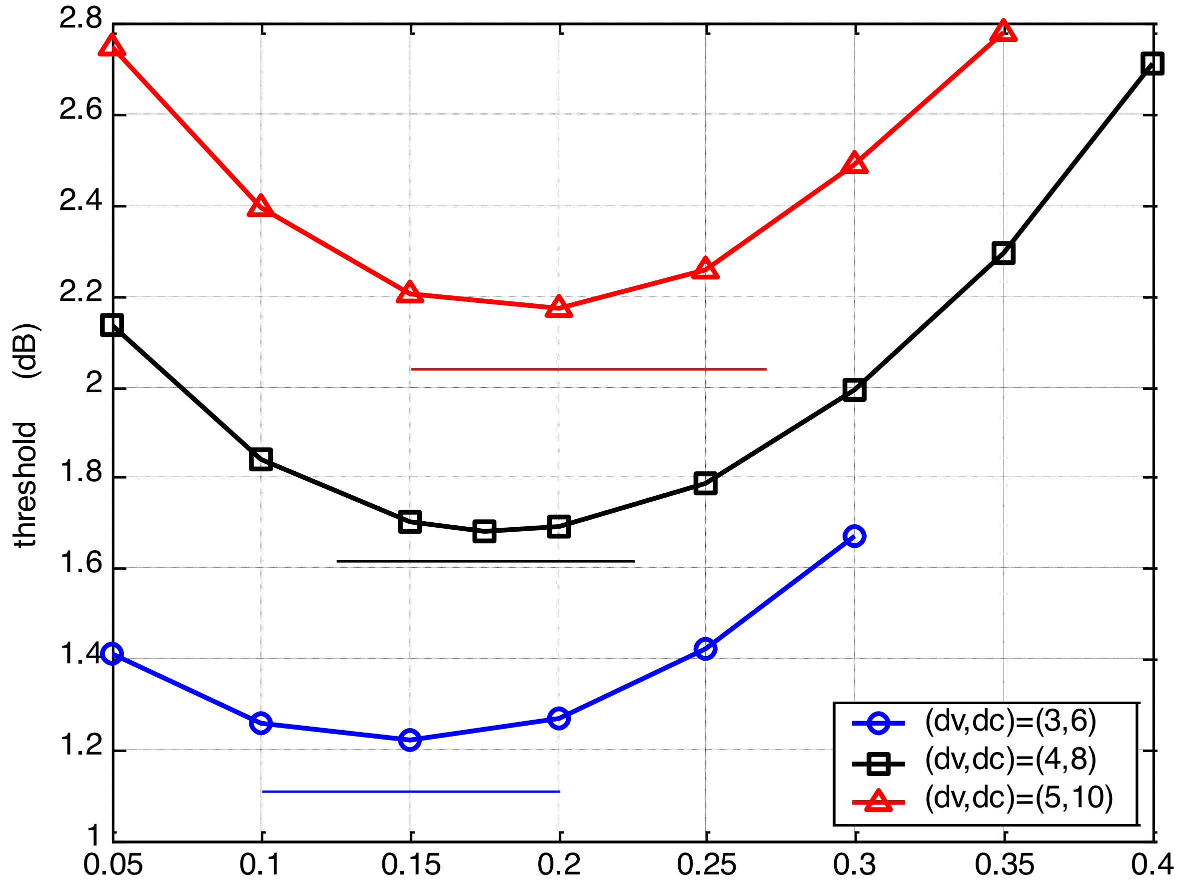
Applying DE to Find Best Decoder Parameters for Improved BP-Based Algorithms

Normalized BP-based



Offset

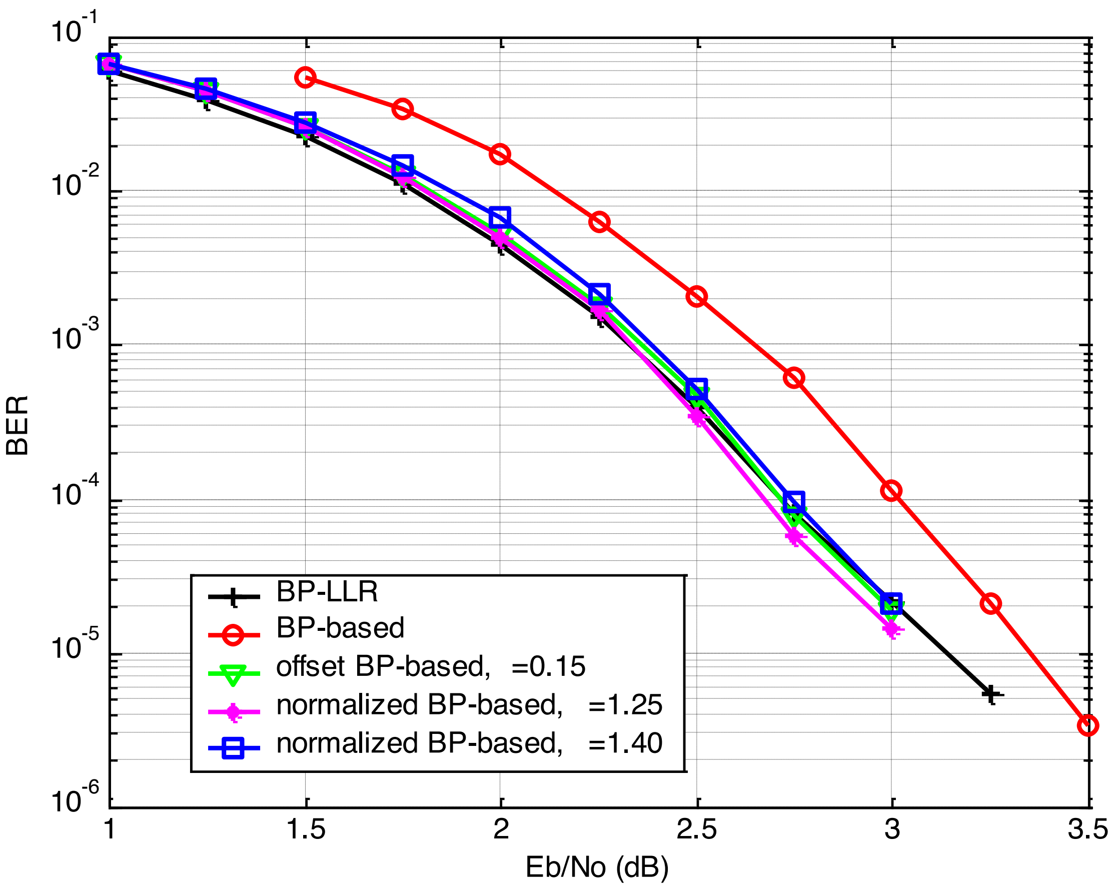
BP-based



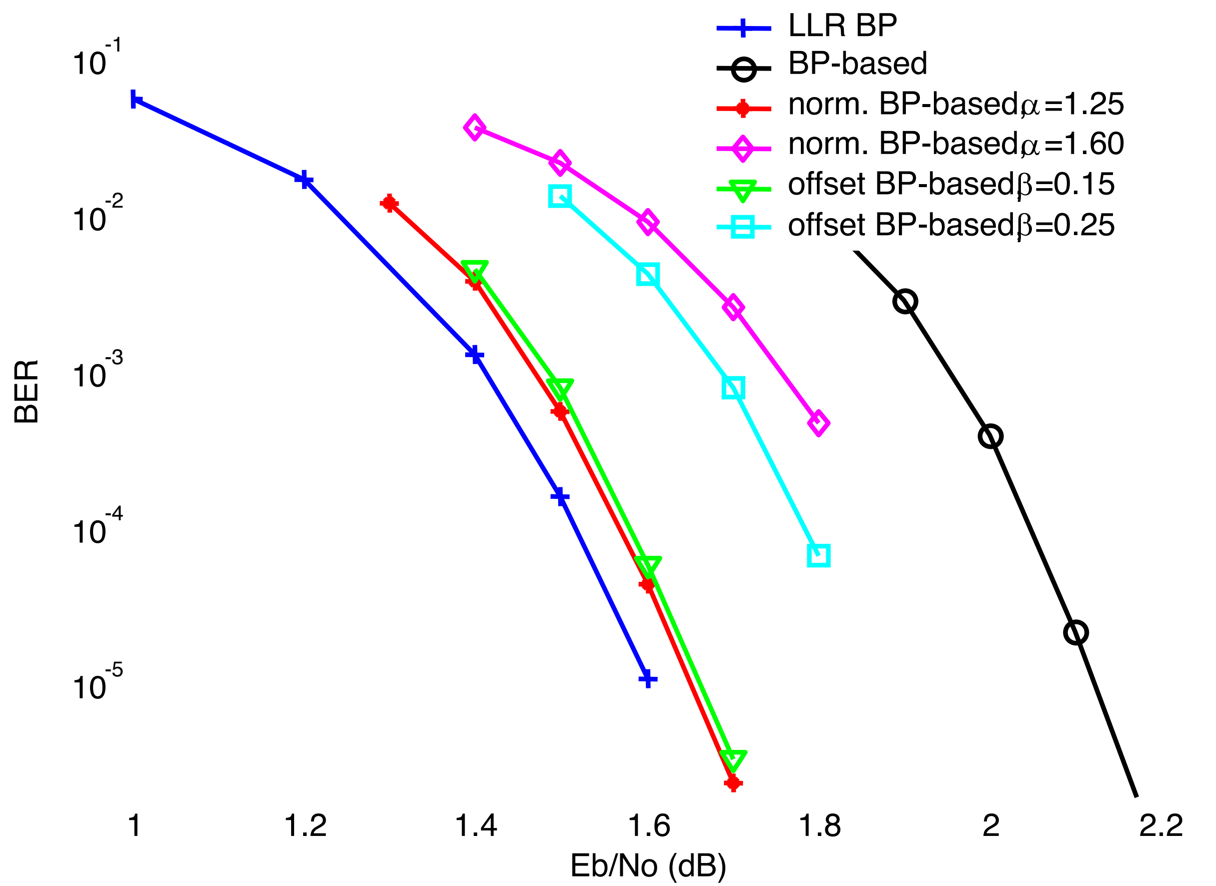
Thresholds (in dB) for various decoding algorithms.

(d_v,d_c)	rate	BP	BP-based	Normalized BP-based		Offset BP-based	
				α	σ	β	σ
(3,6)	0.5	**1.11**	**1.71**	1.25	**1.20**	0.15	**1.22**
(4,8)	0.5	**1.62**	**2.50**	1.50	**1.65**	0.175	**1.70**
(5,10)	0.5	**2.04**	**3.10**	1.65	**2.14**	0.2	**2.17**
(3,5)	0.4	**0.97**	**1.68**	1.25	**1.00**	0.2	**1.03**
(4,6)	1/3	**1.67**	**2.89**	1.45	**1.80**	0.25	**1.84**
(3,4)	0.25	**1.00**	**2.08**	1.25	**1.11**	0.25	**1.13**

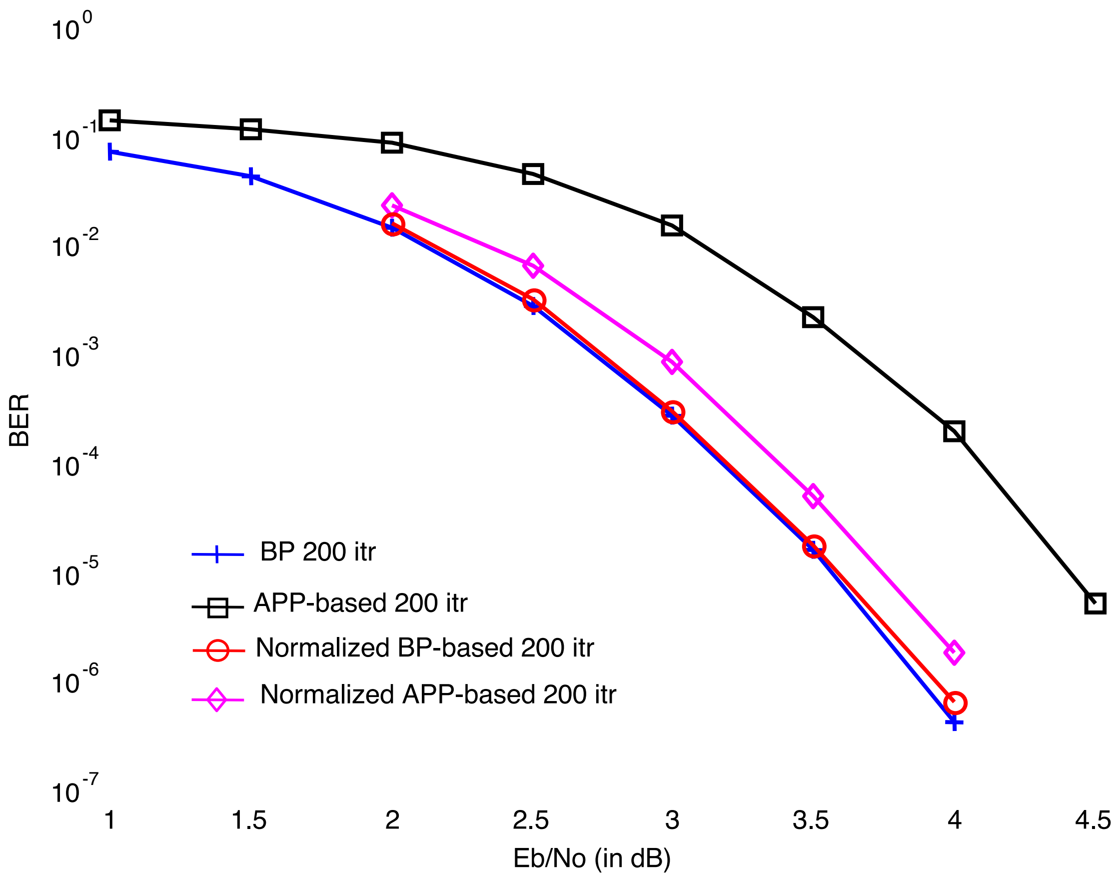
(504,252) LDPC code, (J,L)=(3,6)



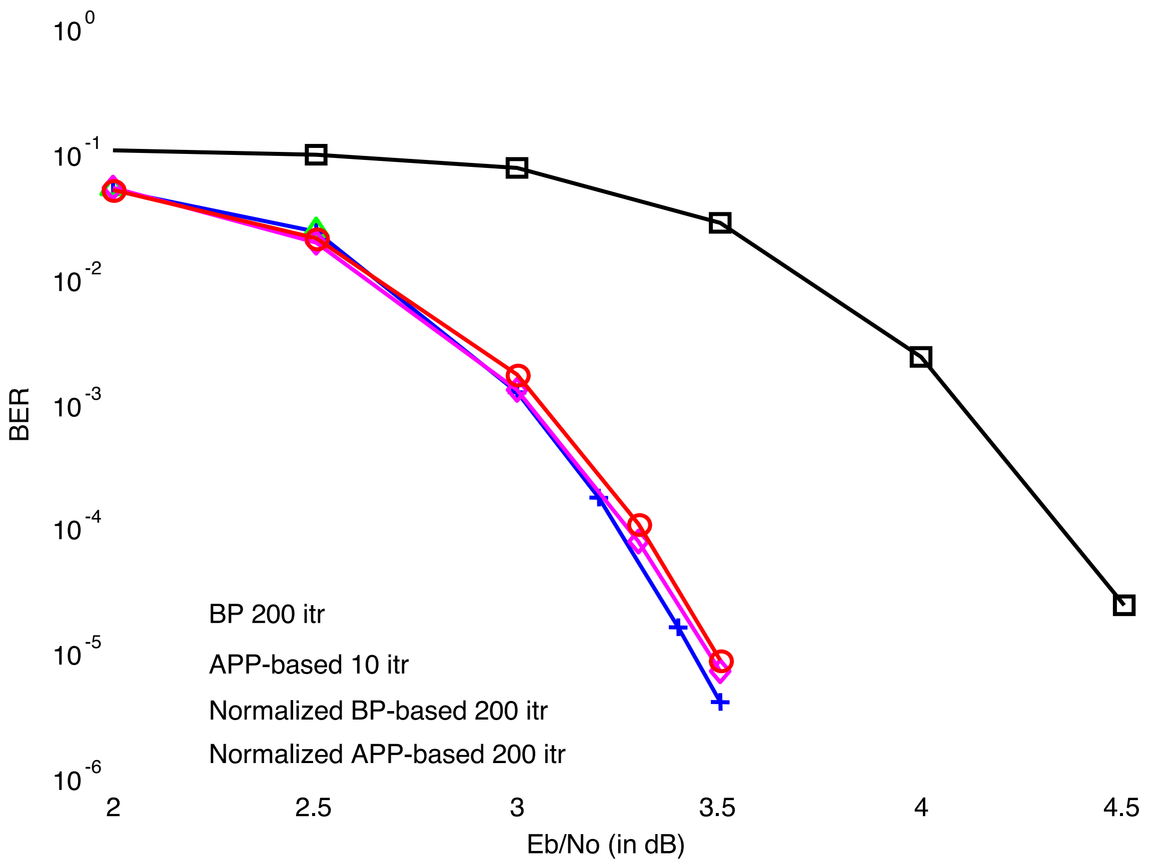
An (8000, 4000) LDPC code, (J,L)=(3,6), 100 iterations.



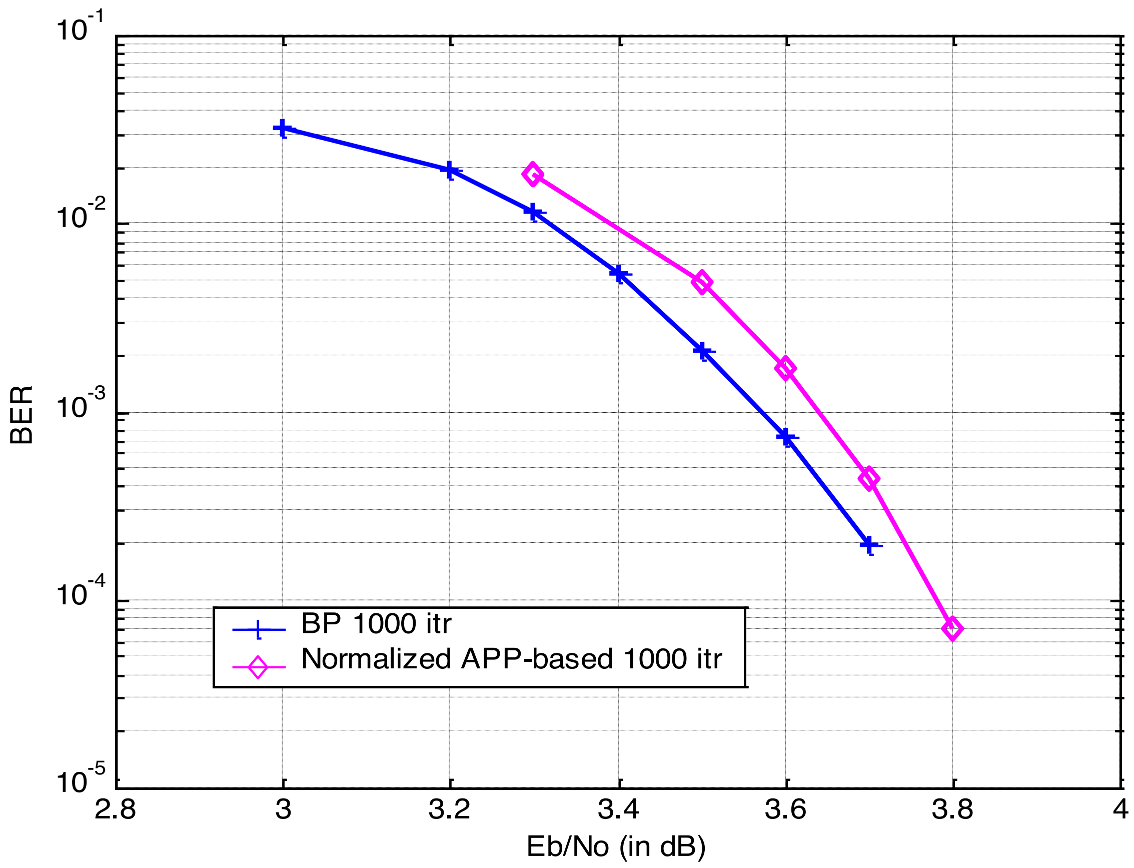
(273, 191) DSC code with BP, APP-based, normalized BP-based and normalized APP-based algorithms, α = 2.0.



(1057, 813) DSC code with BP, APP-based, normalized BP-based and normalized APP-based algorithm, $\alpha = 4.0$.

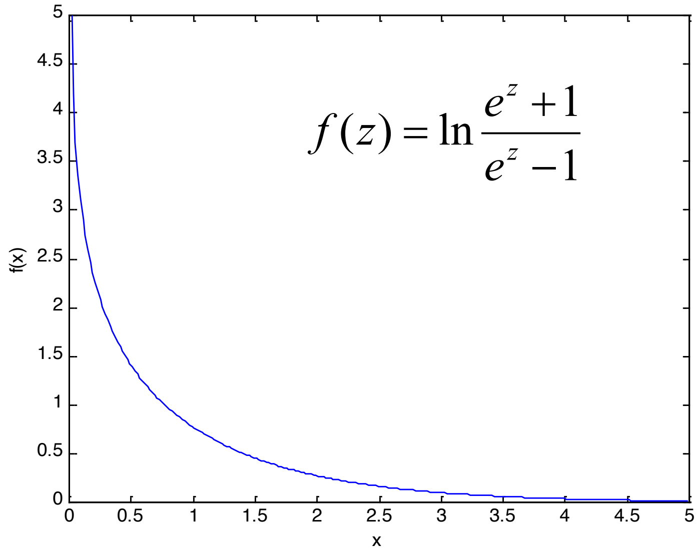


(4161, 3431) DSC code with BP and normalized APP-based algorithm, $\alpha = 8.0$.



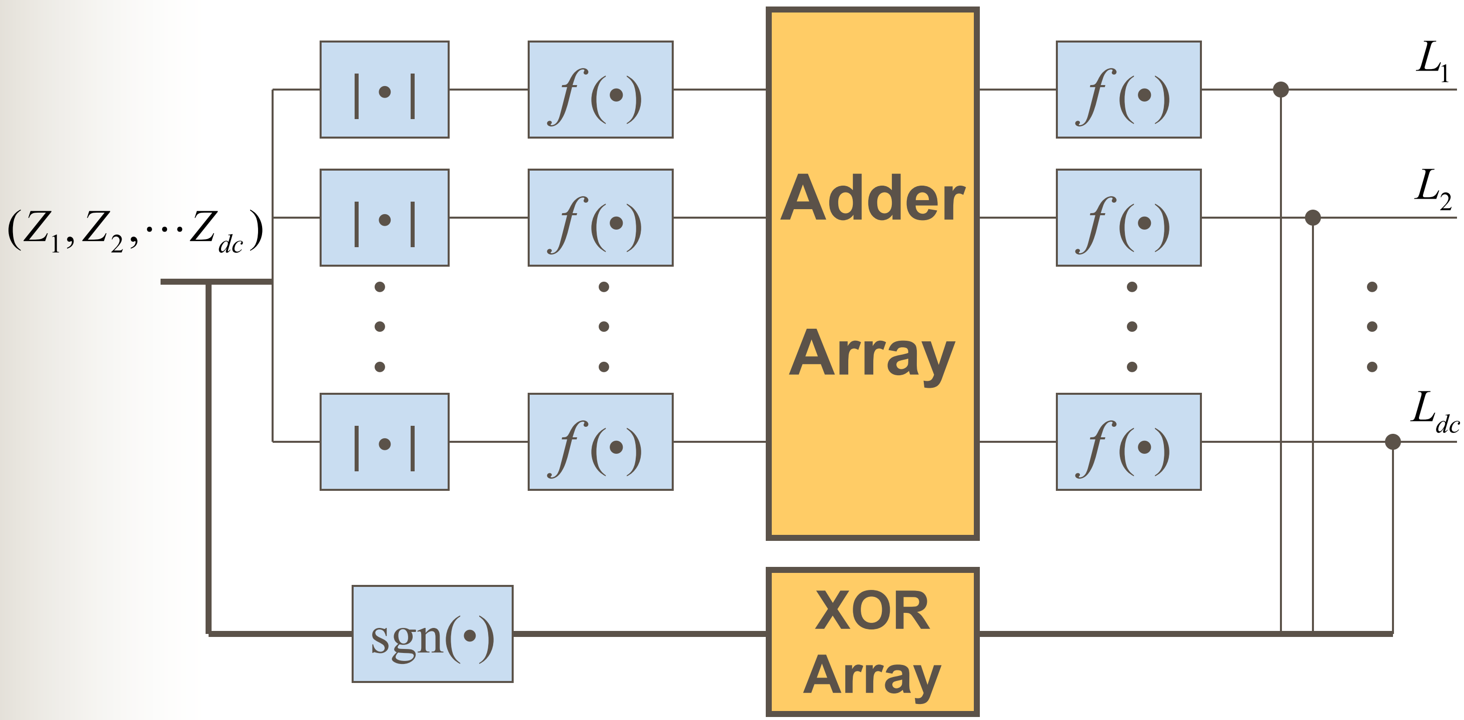
Hardware Implementation of BP Algorithm

$$L = 2\tanh^{-1}\left(\prod_i \tanh(Z_i/2)\right)$$
$$= \prod_i \operatorname{sgn}(Z_i)\cdot f\left(\sum_i f(|Z_i|)\right)$$

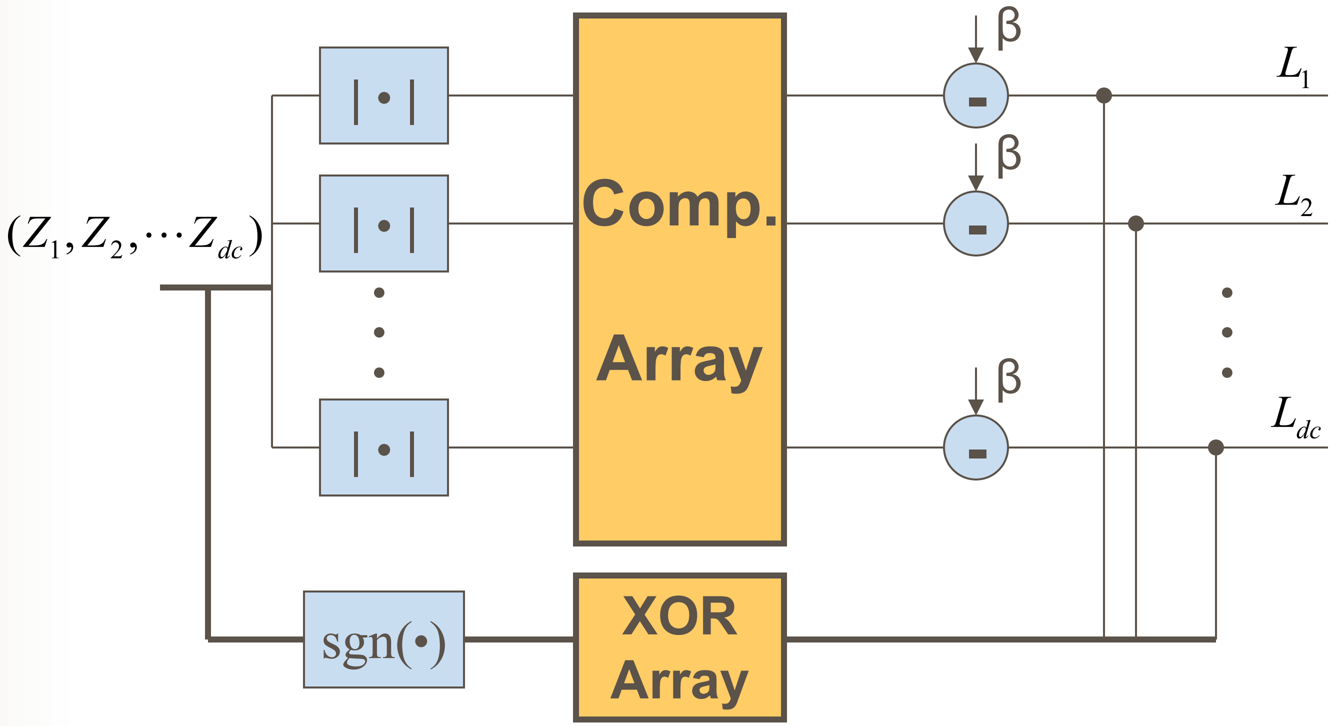


- $f(z)$ can be implemented by look-up table (LUT).
- Only need two kinds of operations: LUT and additions.

Check node implementation of BP algorithm

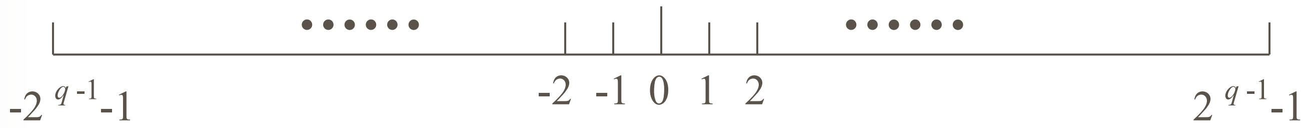


Check node implementation of BP-based algorithm and improved versions



Quantization Effects

q-bit quantization

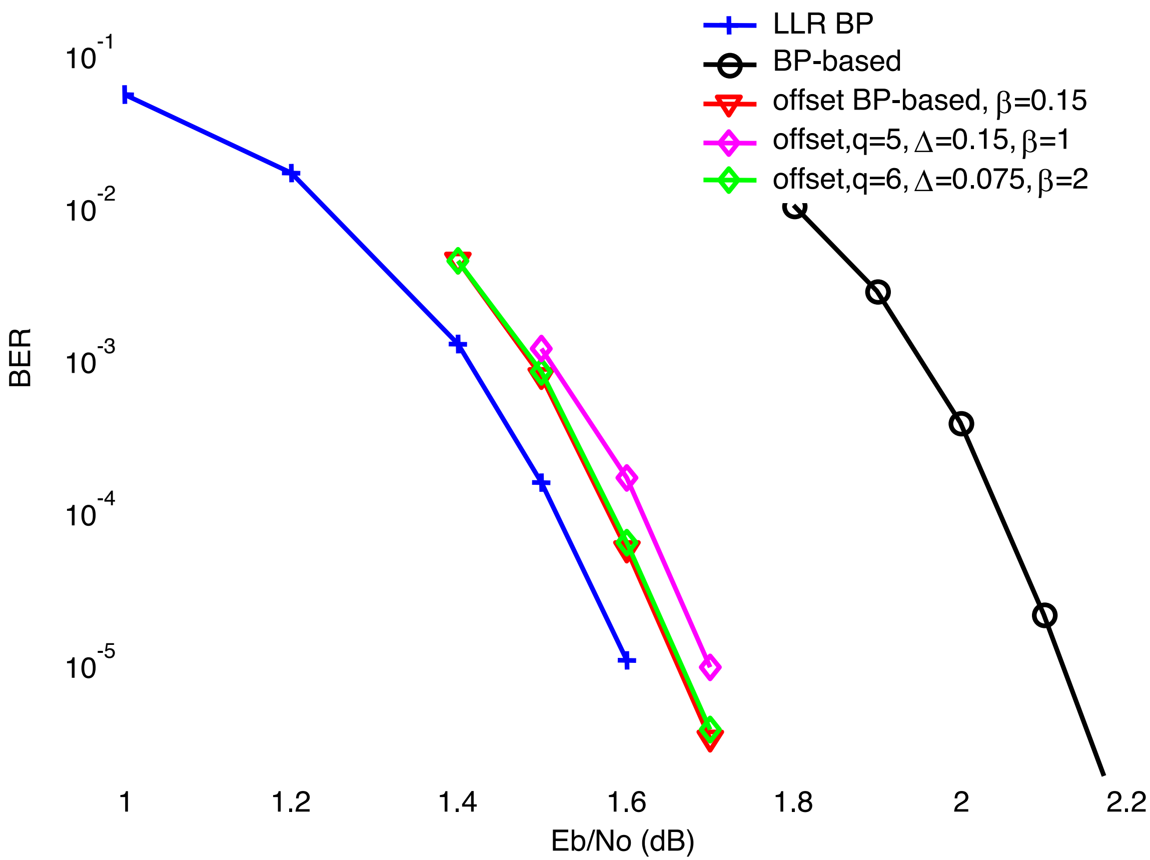


Density evolution algorithms for the BP-based and the normalized BP-based algorithm can be extended to quantized cases.

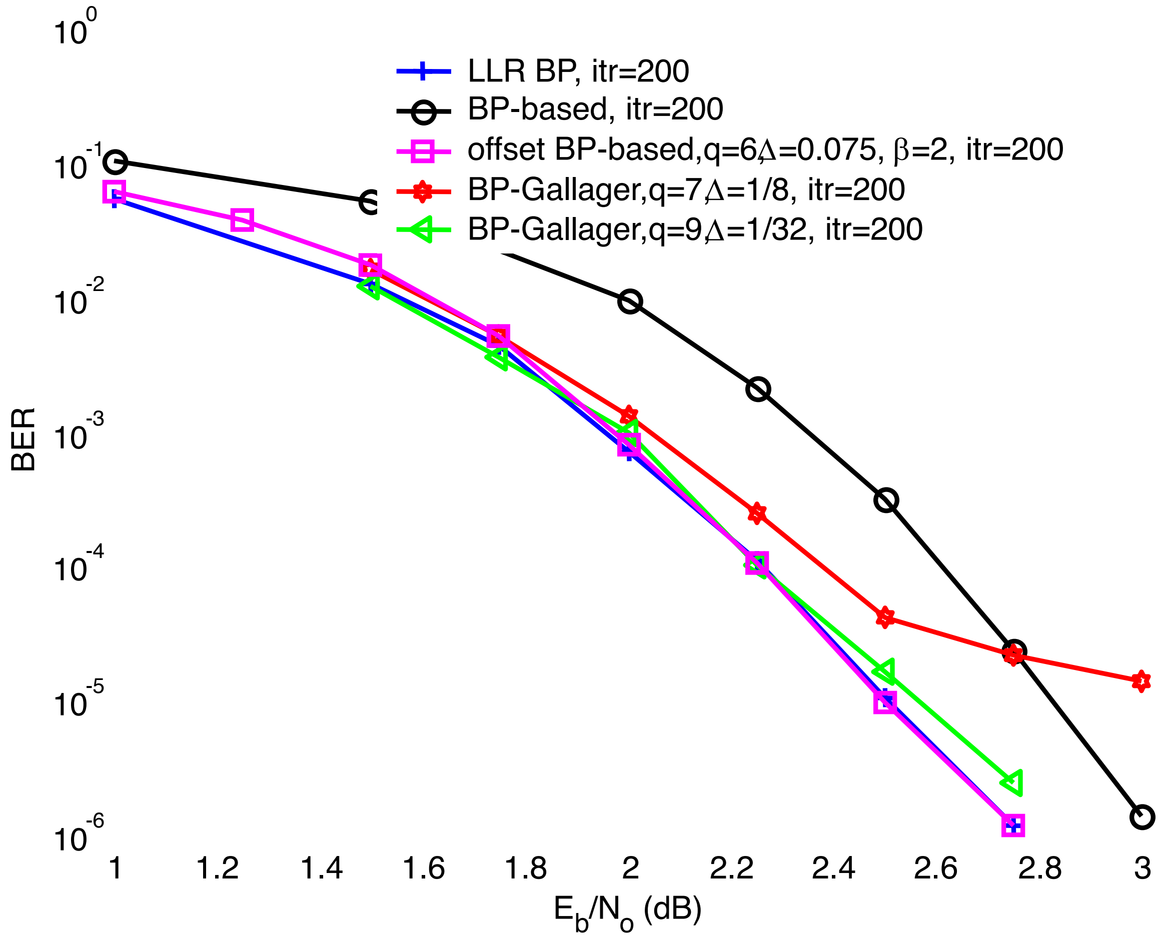
Thresholds for quantized offset BP-based decoding with (dv,dc)=(3,6).

q	Δ	β	thresholds(dB)
5	0.15	1	1.24
5	0.075	2	1.60
6	0.15	1	1.24
6	0.075	2	1.22
7	0.15	1	1.24
7	0.075	2	1.22
7	0.05	3	1.22

An (8000, 4000) , regular LDPC code, (J,L)=(3,6)



(1008, 504) Regular LDPC Code



- BP is sensitive to the error introduced by quantization.

Comparison of various of decoding algorithms

Algorithm	Performance	Complexity	
BP			regular & irregular
Min-sum			
Normalized MS			regular
Normalized MS			irregular
?			irregular

2-D Normalized Min-Sum decoding

- Step 1: (i) Horizontal Step, for $0 \le n \le N-1$ and each $m \in M(n)$:

$$U^{(i)}{}_{mn} = \alpha_{dc(m)} \times \prod_{n' \in N(m) \backslash n} \operatorname{sgn}(V_{mn'}^{(i-1)}) \times \min_{n' \in N(m) \backslash n} |V_{mn'}^{(i-1)}|$$

(ii)Vertical Step, for $0 \le n \le N-1$ and each $m \in M(n)$:

$$V_{mn}^{(i)} = U_{ch,n} + \beta_{dv(n)} \times \sum_{m' \in M(n) \backslash m} U_{m'n}^{(i)}$$

$$V_{n}^{(i)} = U_{ch,n} + \beta_{dv(n)} \times \sum_{m \in M(n)} U_{mn}^{(i)}$$

Density Evolution of 2-D Normalized MS Decoding

- Density evolution for check nodes

$$f_U^{(i)}(u) \leftarrow \sum_{j=1}^{d_{cmax}} \frac{\rho_j}{\alpha_j} \cdot f_U^{(i)}\left(\frac{u}{\alpha_j}\right)$$

- Density evolution for bit nodes

$$f_V^{(i)}(v) \leftarrow \sum_{j=1}^{d_{vmax}} \frac{\lambda_j}{\beta_j} F^{-}\left(F(f_{U_{ch}}) \cdot \left(F(f_U^{(i)})\right)^{j-1}\right)\left(\frac{v}{\beta_j}\right)$$

Optimal Normalization Factors

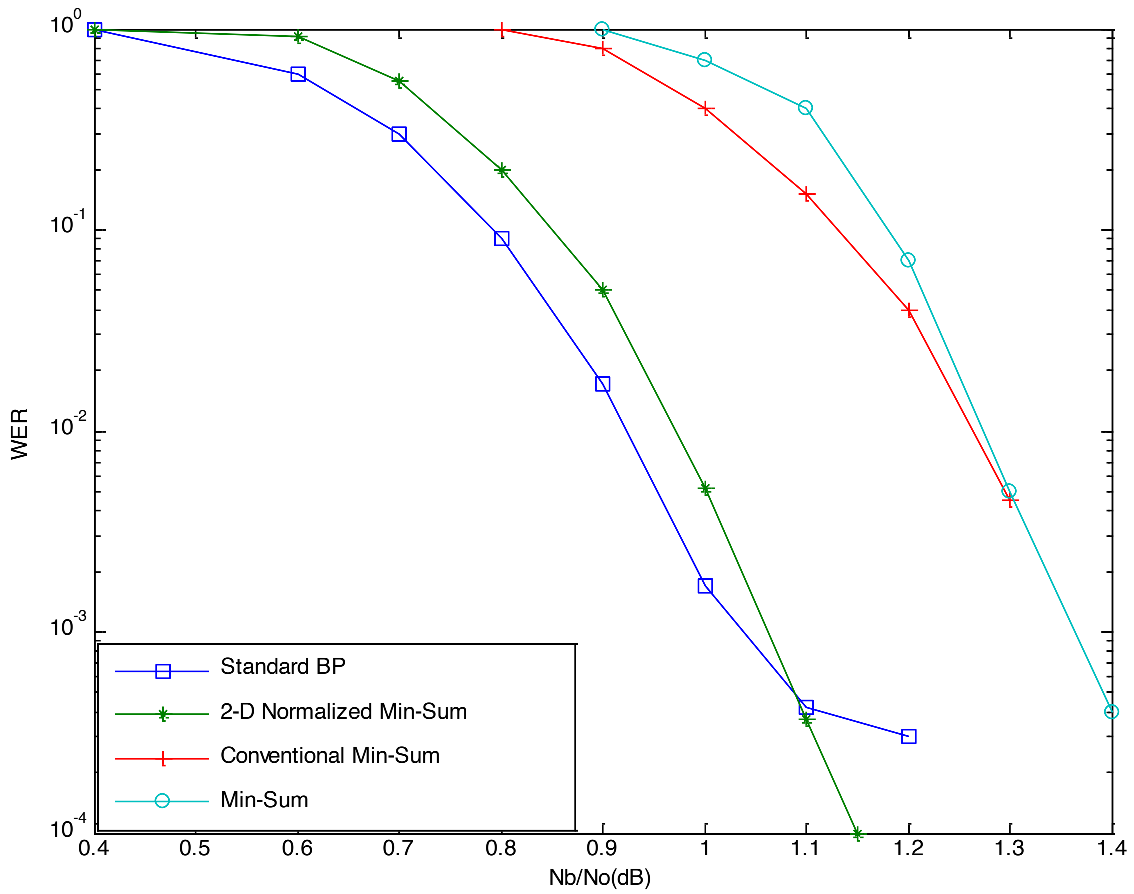
- Normalization factors pair $\mathbf{f} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$

$$\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_{c_{weight}}\}$$

$$\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots, \beta_{v_{weight}}\}$$

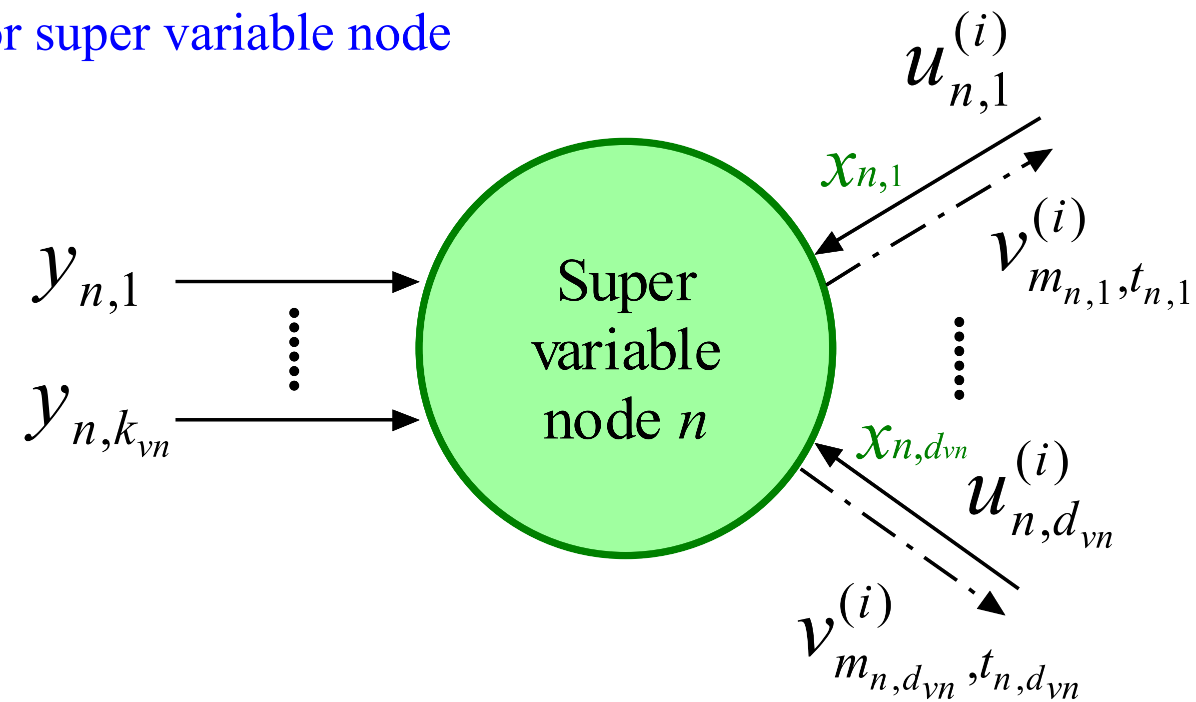
- Intractable when $v_{weight} \times c_{weight}$ is large: use differential evolution.

Simulation Results



Iterative decoding of DG-LDPC codes

For super variable node



$$V^{(i)}_{m_{n,p},t_{n,p}} = \log \frac{P(x_{n,p} = 0 \mid \mathrm{u}^{(i)}_{n[p]}, \mathrm{y}_n)}{P(x_{n,p} = 1 \mid \mathrm{u}^{(i)}_{n[p]}, \mathrm{y}_n)}$$

$$V_{m_{n,p},t_{n,p}}^{(i)} = \log \frac{\mathrm{P}(x_{n,p} = 0 \mid \mathrm{u}_{n[p]}^{(i)}, \mathrm{y}_n)}{\mathrm{P}(x_{n,p} = 1 \mid \mathrm{u}_{n[p]}^{(i)}, \mathrm{y}_n)}$$

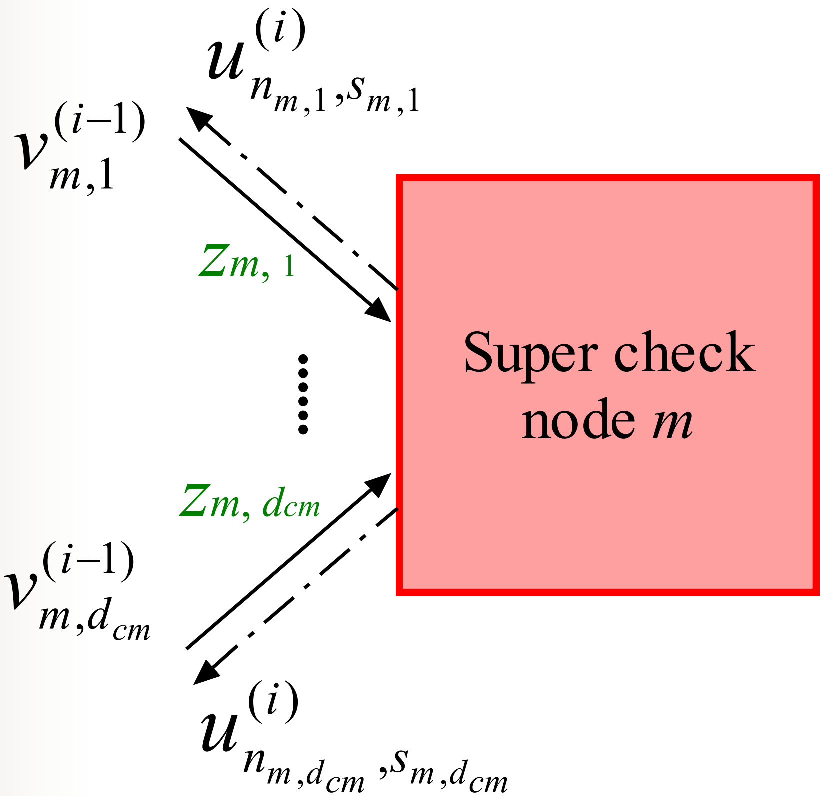
$$V_{m_{n,p},t_{n,p}}^{(i)} = \log \frac{\sum_{\mathbf{b}_n : x_{n,p}=0} \prod_{j=1,\, j\neq p}^{d_{vn}} \mathrm{e}^{-U_{n,j}^{(i)} x_{n,j}} \prod_{j=1}^{k_{vn}} \mathrm{e}^{\frac{2 y_{n,j} c_{n,j}}{N_0}}}{\sum_{\mathbf{b}_n : x_{n,p}=1} \prod_{j=1,\, j\neq p}^{d_{vn}} \mathrm{e}^{-U_{n,j}^{(i)} x_{n,j}} \prod_{j=1}^{k_{vn}} \mathrm{e}^{\frac{2 y_{n,j} c_{n,j}}{N_0}}}$$

$$U_{n_{m,q},s_{m,q}}^{(i)} = \log \frac{\mathrm{P}(z_{m,q} = 0 \mid \mathrm{v}_{m[q]}^{(i-1)})}{\mathrm{P}(z_{m,q} = 1 \mid \mathrm{v}_{m[q]}^{(i-1)})}$$

$$\Downarrow$$

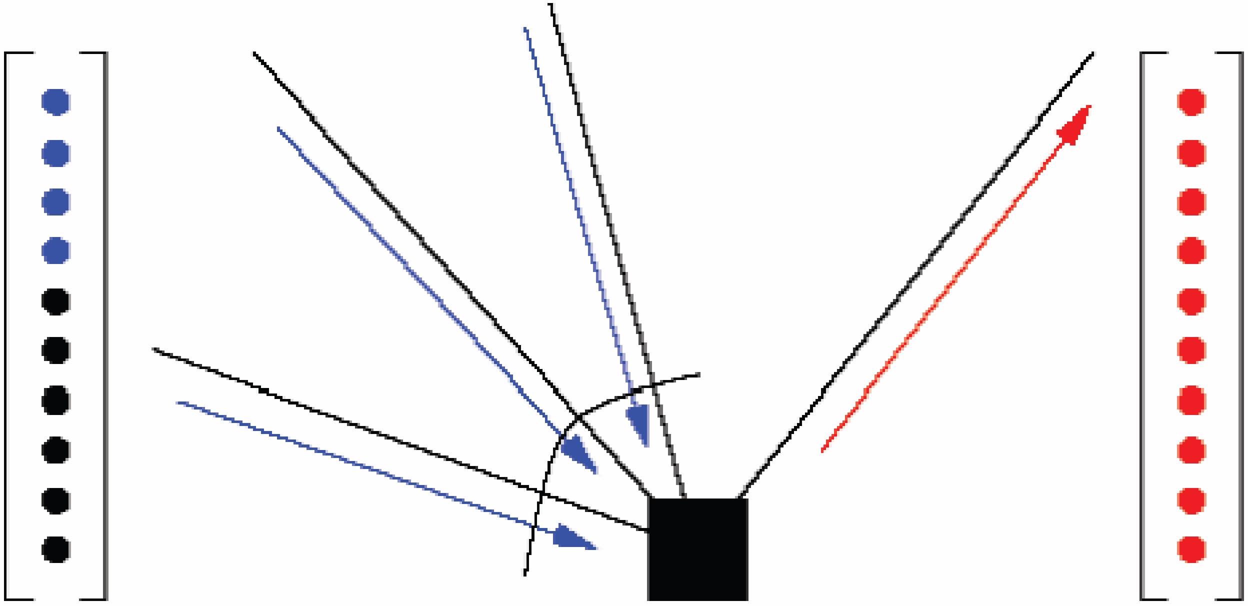
$$U_{n_{m,q},s_{m,q}}^{(i)} = \log \frac{\sum_{\mathbf{z}_m : z_{m,q}=0} \prod_{j=1, j\neq q}^{d_{cm}} \mathrm{e}^{-V_{m,j}^{(i-1)} z_{m,j}}}{\sum_{\mathbf{z}_m : z_{m,q}=1} \prod_{j=1, j\neq q}^{d_{cm}} \mathrm{e}^{-V_{m,j}^{(i-1)} z_{m,j}}}$$

For super check node



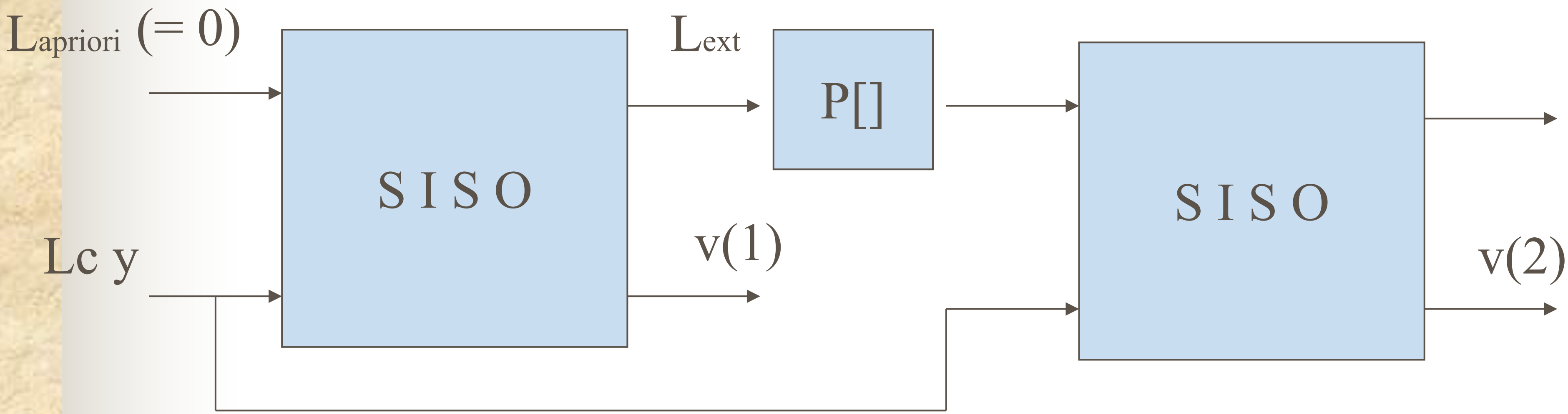
$$U^{(i)}_{n_{m,q}, s_{m,q}} = \log \frac{P(z_{m,q} = 0 \mid v^{(i-1)}_{m[q]})}{P(z_{m,q} = 1 \mid v^{(i-1)}_{m[q]})}$$

Decoding of non binary LDPC codes:

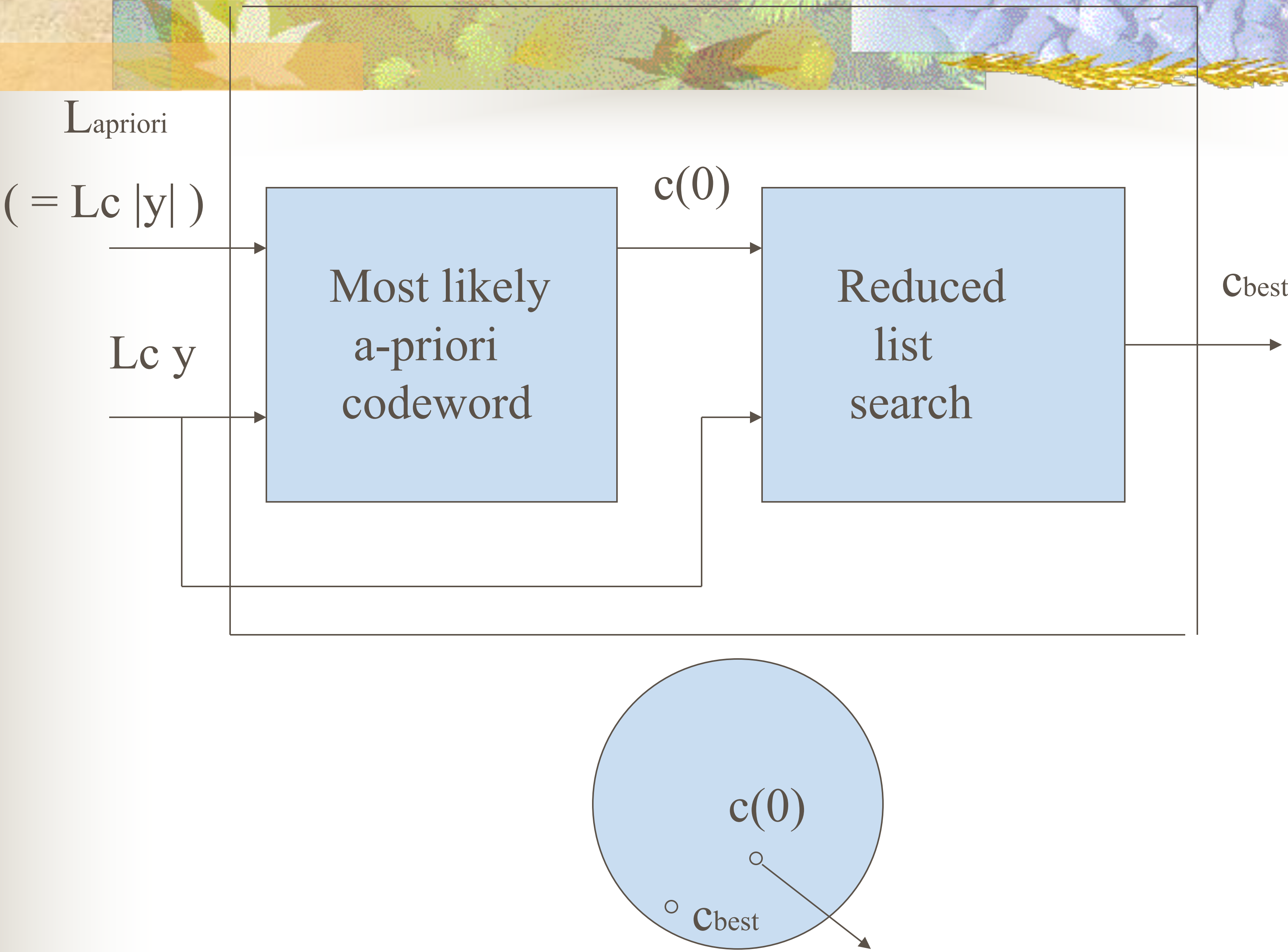


Combined approaches:

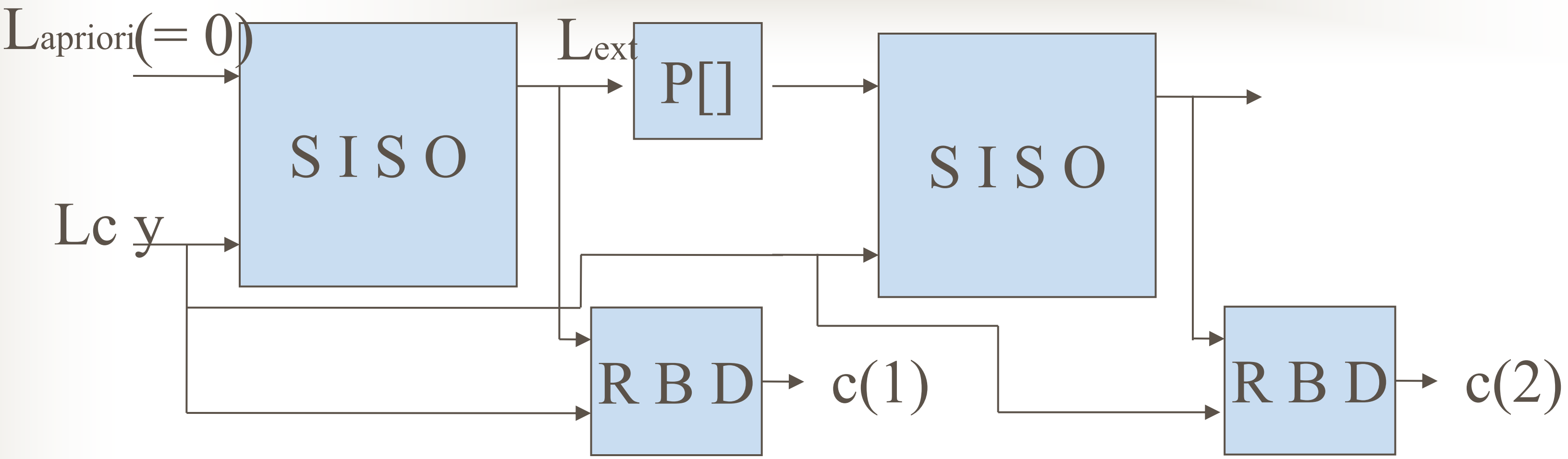
Combined approaches:

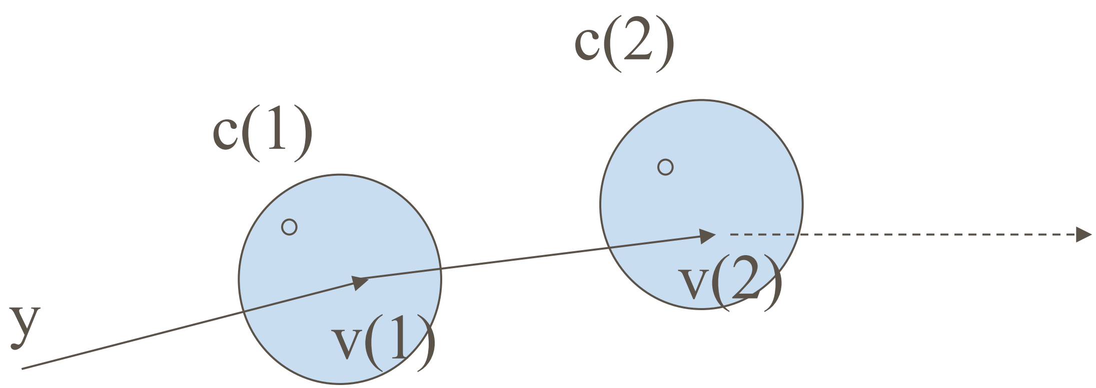


Iterative Decoder



List decoder (RBD)





Combined decoder

Potential Improvement