5G LDPC decoder implementation

Jie Liu hahaliu2001@gmail.com

1 Introduction

LDPC code are chosen for 5G PDSCH and PUSCH channel coding.

Three LDPC decoding algorithm are implemented in this project.

1. Bit Flipping decoding algorithm
   hard-decision, low-complexity, poor performance
2. Belief propagation algorithm
   It is also named as sum-product algorithm.
   Soft-decision, Near-best performance, usually used for simulation only
3. Min-Sum algorithm
   Soft-decision, good performance, low complexity.
   This is so far the only LDPC decoding algorithm that is implemented in commercial chips.
   Four Min-sum algorithms are implemented:
   - Traditional Min-sum
   - Normalized Min-sum
   - Offset Min-sum
   - Mixed Min-sum which has the best performance among these min-sum algorithms

This document is to explain these three algorithms.

The python implementation code is in:

https://github.com/hahaliu2001/python_5gtoolbox.git : py5gphy/ldpc

2 LDPC code

LDPC encoder: $H \times \begin{bmatrix} c \\ w \end{bmatrix} = H \times v_n = 0$

where H is MxN matrix, c is input information bits and w is generated parity bits.

The basic idea of LDPC encoding is to generate parity bit vector from H and information bits

The document to explain 5G LDPC encoder optimization is:

https://github.com/hahaliu2001/python_5gtoolbox.git :
docs/algorithm/LDPC_encoder_optimization.pdf

the basic idea of LDPC decoding is that:

based on received LLR data, estimate v_n vector to make $H \times v_n = 0$

3 Bit-flipping Decoding Algorithm

3.1 Reference

[1] 基于可靠性调度的 LDPC 码比特翻转译码算法

https://www.jsjkx.com/EN/Y2019/V46/I6A/329

[2] Two-Round Selection-Based Bit Flipping Decoding Algorithm for LDPC Codes

https://onlinelibrary.wiley.com/doi/10.1155/2023/6262929

[3] Multi-Stage Bit-Flipping Decoding Algorithms for LDPC Codes

https://www.researchgate.net/publication/333913937_Multi-Stage_Bit-Flipping_Decoding_Algorithms_for_LDPC_Codes

3.2 implementation

This is hard-decision LDPC decoder, simple and poor performance.

for LDPC code (N,K),

where:

N is LDPC code length, K is information bits length, M=N-K is parity bit length

H is M X N parity check matrix

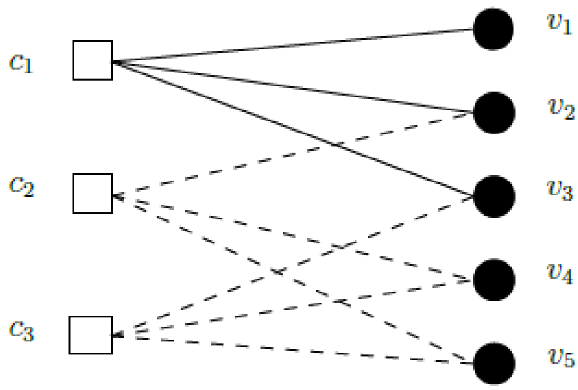
LDPC decoder is to calculate $H \times v_n = s$ and decoding is successful when $s = 0$

Where:

v_n is called “variable nodes” with length=N

s is called “check nodes”, with length=M

Tanner graph is usually used to show the variable nodes and check nodes relations



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

Each line in H matrix shows which variable nodes are used to calculate this check node

Each column in H matrix shows which check nodes are associated with this variable node

For example in above figure,

Line 0 [1,1,1,0,0] means variable nodes[0,1,2] are used for calculate check node 0

Column 1[1,1,0] means check nodes [0,1] are associated with variable node 1

BF decoding procedure:

Input: Receiving N length LLR sequence

Step 1: generate hard-decision sequence C_n from LLR, with C_n=0 if LLR >0 else 1

Step 2: cal $s = HC_n$

Step 3: if s==0 -> decoding success, stop the processing

Step 4: $E_n = (2s-1)H$ is the reliability of C_n.

Higher E_n indicates lower reliability and mean this bit is more possible to be wrong

Step 5: bit flip C_n bit with maximum E_n value, then repeat the processing from step 2

Note 1:

Some paper provides $E_n = sH$, not $E_n = (2s-1)H$.

$E_n = sH$ is used for regular LDPC(number of '1' in each line are the same. Number of '1' in each column are the same)

5G selects irregular LDPC, $E_n = sH$ doesn't work

Note 2:

Many paper shows Weighted Bit-Flipping Algorithm is better than non- Weighted Bit-Flipping Algorithm. But in my test, Weighted Bit-Flipping Algorithm didn't work. BLER is always 100%

Weighted Bit-Flipping Algorithm is:

$$e_k^{(2)} = \sum_{j \in M(k)} \left(2s_j - 1\right) . r_{\min}^j ,$$

Where r_{min}^j is the minimum absolute of LLR value for the bits participating in the jth parity-check equation

Note 3:

BF decoding performance is much lower than soft-decision algorithm.

Python code

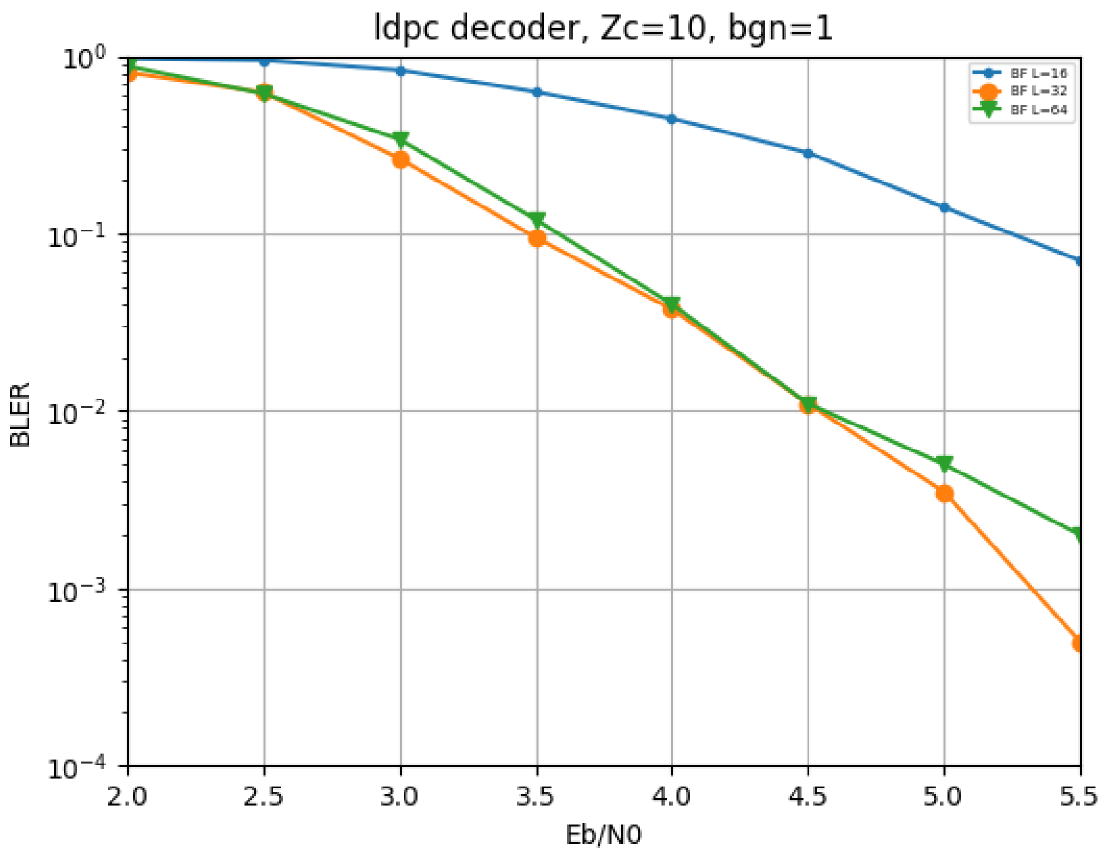
```
#hard coded LLRin to generate ck bit sequence, LLR >0 ->0, LLR<0 -> 1
ck = np.copy(LLRin)
ck[ck > 0] = 0
ck[ck < 0 ] = 1

#main loop
for iter in range(L):
    S = (H @ ck.T) % 2

    if not np.any(S):
        #if S is all zero sequence, decoding success, return Tue
        return ck, True

    #process if S is not all zero
    En = (2*S-1) @ H #this is correct equation
    max_value = np.max(En)
    #bit flip any ck bits with En value==max_value
    ck[En==max_value] = 1- ck[En==max_value]
```

3.3 BF simulation:



test configuration: Zc = 10, bgn = 1, K = Zc*22=220, N=Zc*66=660

test BF with L size 16, 32, 64. It shows that L=32 and 64 have similar performance and both are 1.6dB better than L=16

the simulation script is script/sim_ldpc_decoder_bf.py in

https://github.com/hahaliu2001/python_5gtoolbox.git :

4 sum-product algorithm(or belief propagation in another name)

Note:

"belief propagation" and "sum-product algorithm" are essentially the same thing

4.1 Reference

- A Generalized Adjusted Min-Sum Decoder for 5G LDPC Codes: Algorithm and Implementation Yuqing Ren, Student Member, IEEE, Hassan Harb, Member, IEEE, Yifei Shen, Member, IEEE, Alexios Balatsoukas-Stimming, Member, IEEE, and Andreas Burg, Senior Member, IEEE

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/2310.15801

- https://github.com/PKU-HunterWu/LDPC-Encoder-Decoder/tree/main
- Improved Min-Sum Decoding of LDPC Codes Using 2-Dimensional Normalization
  Juntan Zhang and Marc Fossorier, Daqing Gu and Jinyun Zhang
- **Improved Sum-Min Decoding for Irregular LDPC Codes**
  Gottfried Lechner & Jossy Sayir
- chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://staff.ustc.edu.cn/~wyzhou/chapter8.pdf
- An Introduction to LDPC Codes, William E. Ryan,
  chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://tuk88.free.fr/LDPC/ldpcchap.pdf

4.2 probability knowledge used for LDPC decoder

4.2.1 probability of modulo 2 addition of multiple random variables

The question is:

assume input variables $x1, x2, \ldots . xm$ probability is:

$$P1 = P(x1 = 0), P2 = P(x2 = 0), \ldots Pm = P(xm = 0)$$

Calculate the $P(x1 \oplus . \oplus . \oplus xm)$

<u>First,</u>

calculate $P(x1 \oplus x2)$

$$P(x1 \oplus x2 = 0) = P(x1 = 0)P(x2 = 0) + (P(x1 = 1)(x2 = 1) = P1P2 + (1 - P1)(1 - P2)$$
$$= 1 - (P1 + P2) + 2P1P1$$

Then get: $2P(x1 \oplus x2 = 0) - 1 = 4P1P2 - 2(P1 + P2) + 1 = (2P1 - 1)(2P2 - 1)$ (A1)

<u>Second,</u>

calculate $P(x1 \oplus x2 \oplus x3)$ from $P(x1 \oplus x2)$ based on (A1) equation

$$2P(x1 \oplus x2 \oplus x3 = 0) - 1 = (2P(x1 \oplus x2 = 0) - 1)(2P3 - 1) = (2P1 - 1)(2P2 - 1)(2P3 - 1)$$

<u>By iteration we get the general equation:</u>

$$2P(x1\oplus.\oplus.\oplus xm = 0) - 1 \ = \prod_{n=1}^{m}(2P_n - 1)$$

Then get:

$$P(x1\oplus.\oplus.\oplus xm = 0) \ = \frac{1}{2} + \frac{1}{2}\prod_{n=1}^{m}(2P_n - 1)$$

$$P(x1\oplus.\oplus.\oplus xm = 1) = 1 - P(x1\oplus.\oplus.\oplus xm = 0) \ = \frac{1}{2} - \frac{1}{2}\prod_{n=1}^{m}(2P_n - 1)$$

LLR equation

Usually LLR(log-likelihood ratio) value is used to express the probability.

Definition: $L(x) = LLR(x) = \log \frac{P(x=0)}{P(x=1)}$

Get:

$$P(x = 0) = \frac{e^{L(x)}}{1+e^{L(x)}}, \ \ P(x = 1) = \frac{1}{1+e^{L(x)}}$$

Get:

$$2P(x = 0) - 1 = \frac{e^{L(x)}-1}{e^{L(x)}+1}$$

By $\tanh(t) = \frac{e^{2t}-1}{e^{2t}+1}$ definition,

we get $2P(x = 0) - 1 = \tanh\left(\frac{L(x)}{2}\right) = \frac{e^{L(x)}-1}{e^{L(x)}+1}$

then

$$\prod_{n=1}^{m}(2P_n - 1) = \prod_{n=1}^{m}\tanh\left(L_n(x)/2\right)$$

We get three LLR expressions which are used on different papers

Equation 1:

$$LLR_s = log\frac{P(x1\oplus.\oplus.\oplus xm=0)}{P(x1\oplus.\oplus.\oplus xm=1)} = log\frac{\frac{1}{2}+\frac{1}{2}\prod_{n=1}^{m}\tanh\left(L_n(x)/2\right)}{\frac{1}{2}-\frac{1}{2}\prod_{n=1}^{m}\tanh\left(L_n(x)/2\right)} = log\frac{1+\prod_{n=1}^{m}\tanh\left(L_n(x)/2\right)}{1-\prod_{n=1}^{m}\tanh\left(L_n(x)/2\right)} \qquad \text{(B1)}$$

Equation 2:

With definition: $tanh^{-1}(t) = \frac{1}{2}\log\left(\frac{1+t}{1-t}\right)$

$$LLR_s = 2tanh^{-1}\left(\prod_{n=1}^{m} \tanh(L_n(x)/2)\right) \quad \text{(B2)}$$

<u>Equation 3:</u>

with $\tanh\left(L_n(x)\right) = sign\left(L_n(x)\right)\tanh(|L_n(x)/2|)$

get: $\prod_{n=1}^{m} \tanh\,(L_n(x)/2) = \prod_{n=1}^{m} \text{sign}\,(L_n(x)) \prod_{n=1}^{m} \tanh\,(|L_n(x)/2|) = sA$

where $s = \prod_{n=1}^{m} \text{sign}\,(L_n(x))$, $A = \prod_{n=1}^{m} \tanh\,(|L_n(x)/2|)$

get: $log\,\frac{1+sA}{1-sA} = \begin{cases} log\,\frac{1+A}{1-A} & s = 1 \\ -log\,\frac{1+A}{1-A} & s = -1 \end{cases} = s * log\,\frac{1+A}{1-A}$

LLR equation 3:

$$LLR_s = \prod_{n=1}^{m} \text{sign}\,(L_n(x)) * 2tanh^{-1}\left(\prod_{n=1}^{m} \tanh\,(|L_n(x)/2|)\right) \quad \text{(B3)}$$

4.2.2 Conditional probability given multiple events

From $\frac{P(C|X)}{P(c)} = \frac{P(X|C)}{P(x)}$

Get

$\frac{P(c|x1,x2,\ldots,xm)}{P(c)} = \frac{P(x1,x2,\ldots,xm|c)}{P(x1,x2,\ldots,xm)} = \frac{P(x1|c)}{P(x1)}\frac{P(x2|c)}{P(x2)}\ldots\frac{P(xm|c)}{P(xm)} = \frac{P(c|x1)}{P(c)}\frac{P(c|x2)}{P(c)}\ldots\frac{P(c|xm)}{P(c)}$

Where:

$x1, x2, \ldots, xm$ are independent to each other

Then

$P(c = 0|x1, x2, \ldots, xm) = K\prod_{n=1}^{m} P(c = 0|x_n)$,

$P(c = 1|x1, x2, \ldots, xm) = K\prod_{n=1}^{m} P(c = 1|x_n)$, Where:

$K = \frac{1}{\prod_{n=1}^{m-1} P(c)}$ is constant value

LLR equation:

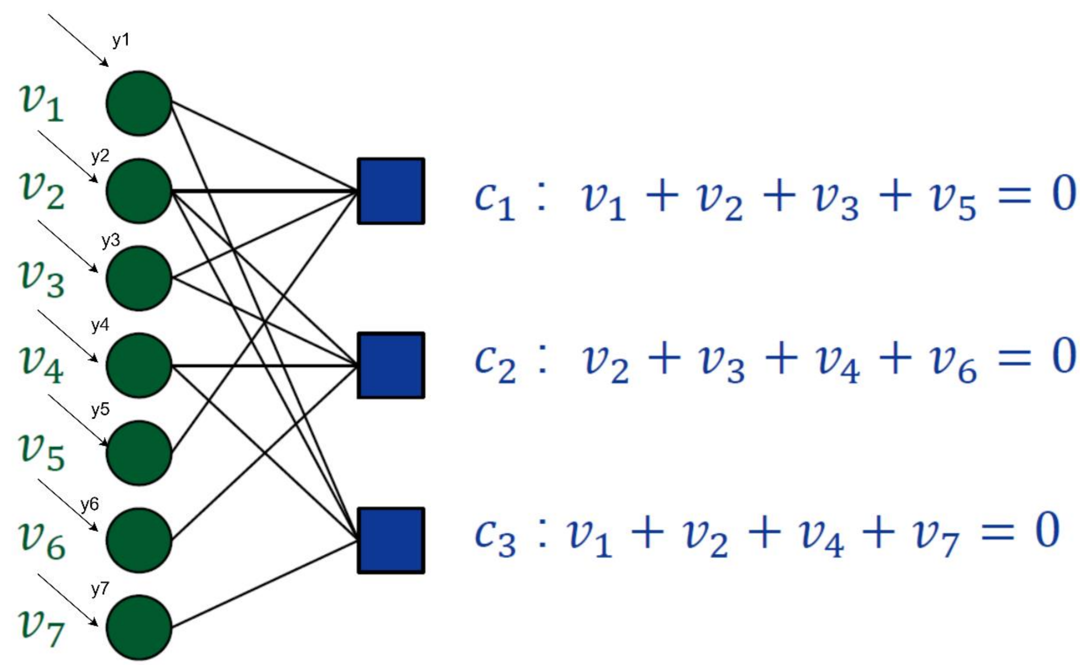
$$LLR(c|x1, x2, \ldots, xm) = log\frac{P(c = 0|x1, x2, \ldots, xm)}{P(c = 1|x1, x2, \ldots, xm)} = \sum_{n=1}^{m} LLR(c|x_n) \qquad \text{(C1)}$$

4.3 Learn SP algorithm from the example

It is better to learn SP algorithm from the example first.

Parity-check matrix:

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



Above is (N=7, M=3) H parity check matrix and Tanner graph.

$v_1, v_2, \ldots v_7$ are variable nodes

c_1, c_2, c_3 are check nodes

$y_1, y_2, \ldots y_7$ are initial values received from external channel equalization for variable nodes

Usually LLR value of $y_1, y_2, \ldots y_7$ are given to BP decoder

Step 1: Check node processing

Each check node value is expected to be zero.

It means that for $c1: v1 \oplus v2 \oplus v3 \oplus v5 = 0$, if $v2 \oplus v3 \oplus v5 = 0 \; or \; 1$, then $v1$= also 0 or 1

$P(v1|c1) = P(v2 \oplus v3 \oplus v5)$ is the probability sent from check node c1 to variable node v1

From equation (B2) we get:

$$LLR(v1|c1) = 2tanh^{-1}(\tanh(LLR(v2)/2)\tanh(LLR(v3)/2)\tanh(LLR(v5)/2))$$

Similar for $c3: v1 \oplus v2 \oplus v4 \oplus v7 = 0$

$P(v1|c3) = P(v2 \oplus v4 \oplus v7)$is the probability sent from check node c3 to variable node v1

$$LLR(v1|c3) = 2tanh^{-1}(\tanh(LLR(v2)/2)\tanh(LLR(v4)/2)\tanh(LLR(v7)/2))$$

Step 2: Variable node processing

$v1$ received three messages:

$P(v1|y1), P(v1|c1), P(v1|c3)$

Where:

$P(v1|y1)$ is the initial message received from external

$P(v1|c1), P(v1|c3)$ are the messages received from check nodes that connect to $v1$

From equation (C1) we get:

$$LLR(v1) = LLR(v1|y1) + LLR(v1|c1) + LLR(v1|c3)$$

In similar way we get $LLR(v2), \ldots, LLR(v7)$

Step 3: Check LDPC decoding result

Generate 7 hard bit sequence c from LLR value: $c_i = \begin{cases} 0 & LLR(v_i) > 0 \\ 1 & else \end{cases}$

LDPC decoding success if $Hc = 0$, the exit LDPC decoding.

If not, run next step.

Step 4: Update LLR messages sent from variable nodes to check nodes if LDPC decoding failed

$LLR(v1)$is calculated using initial LLR value and LLR values from all check nodes that connect to v1

When update LLR messages from v1 to any check node, it need exclude the LLR value from this check node. For example for v1:

$$LLR(v1 \to c1) = LLR(v1|y1) + LLR(v1|c3)$$

$$LLR(v1 \to c3) = LLR(v1|y1) + LLR(v1|c1)$$

After calculate all LLR messages from all variable nodes to all check nodes, go back step 1.

Repeat all the processing until LDPC decode success or reach maximum iteration.

4.4 General SP algorithm

Notation used in the equation

$\mathrm{LQ}_n$is the overall LLR for variable node n

$\mathrm{L}_n$is the initial LLR for variable node n received from channel equalization

$\mathrm{Lq}_{nm}$is the LLR from variable node n to check node m

$\mathrm{Lr}_{mn}$is the LLR from check node m to variable node n

$\mathrm{c}_n$is the decoded bit for variable node n

A(j) is '1' position list for H line j, is all variable nodes connecting to check node j

B(i) is '1' position list for H column I, if all check nodes to connect to variable node i

Processing

Step 1 initialization $\mathrm{Lq}_{nm} = \mathrm{L}_n$

Step 2: calculate Lr_{mn},LLR from check node m to variable node n

$\mathrm{Lr}_{mn} = 2tanh^{-1}(\prod_{i \in A(m), i != n} \tanh{(Lq_{mi}/2)})$

Step 3: calculate overall LLR for variable nodes

$$\mathrm{LQ}_n = \sum_{j \in B(n)} Lr_{jn}$$

Step 4: check LDPC result

Hard -bit decision: $c_n = \begin{cases} 0 & LQ_n > 0 \\ 1 & else \end{cases}$

If $Hc = 0$ or reach maximum iteration:

Terminate LDPC decoding

Else:

Step 5: update Lq_{nm} and go back step 1

$$\mathrm{Lq}_{nm} = \mathrm{LQ}_n - Lr_{mn}$$

5 Min-sum algorithm

Step 2 in BP algorithm $\mathrm{Lr}_{mn} = 2tanh^{-1}(\prod_{i\in A(m),i!=n} \tanh\,(Lq_{mi}\,/2)))$ is not efficient for hardware and software implementation.

Min-sum algorithm is to simplify this calculation.

From equation (B3) we get:

$\mathrm{Lr}_{mn} = \prod_{i\in A(m),i!=n} \mathrm{sign}\,(Lq_{mi})) * 2tanh^{-1}(\prod_{i\in A(m),i!=n} \tanh\,(|Lq_{mi}\,/2|)))$

With approximation:

$$\prod_{i\in A(m),i!=n} \tanh\,(|Lq_{mi}\,/2|) \approx \min_{i\in A(m),i!=n} \tanh\,(\left|\frac{Lq_{mi}}{2}\right|)$$

We get:

$$\mathrm{Lr}_{mn} = \prod_{i\in A(m),i!=n} \mathrm{sign}\,(Lq_{mi})) * \min_{i\in A(m),i!=n} |Lq_{mi}|$$

Replacing BP Lr_{mn} calculation with this new equation, and keep other steps in BP unchangeable, is traditional min-sum algorithm.

Below equation is the optimization of min-sum approximation:

$$\prod_{i\in A(m),i!=n} \tanh\,(\left|\frac{Lq_{mi}}{2}\right|) \approx \alpha * \max\,(\min_{i\in A(m),i!=n} \tanh\,(\left|\frac{Lq_{mi}}{2}\right|) - \beta, 0)$$

- $\alpha = 1, \beta = 0$: traditional min-sum algorithm
- $\alpha\ in\ (0,1), \beta = 0$: normalized min-sum algorithm
- $\alpha = 1, \beta\ in(0,1)$ offset min-sum algorithm
- $\alpha\ in(0,1), \beta\ in(0,1)$ mixed min-sum algorithm

The first three min-sum algorithms are found in the papers, and are supported matlab 5g toolbox. No paper/open-source code mentioned mixed min-sum. But from the equation we can easily think of 'what happen if $\alpha\ in(0,1), \beta\ in(0,1)$'? and in my simulation, if choosing α, β correctly, the mixed min-sum has the best performance.

Further min-sum optimization

It is very possible that α, β value are related to number of '1' in each line of H matrix.

5G LDPC choose irregular LDPC which means each line may have different number of '1'. To further optimize LDPC performance, it may be better to choose different α, β values for different line. Anyone can develop this feature if interesting in it.

5.1 α, β searching result for different 5G LDPC configuration

NMS_ldpc_search_best_alpha.py, OMS_ldpc_search_best_beta.py, mixed_MS_ldpc_search_best_pair.py

Python script are used to search best α for NMS, best β for OMS and best α, β for mixed-MS.

Different 5G LDPC ZC/bgn are used for the test.

The searching result are:

1. $\alpha = 0.7$ is optimized for NMS for all LDPC configuration
2. $\beta\ = 0.5$ is optimized for OMS for all LDPC configuration
3. $\alpha = 0.8,\ \beta = 0.3$ is optimized for mixed-MS for all LDPC configuration
4. The maximum line weight (number of '1' on the line of H matrix) are 19 for all Zc and bgn=1
5. The maximum line weight (number of '1' on the line of H matrix) are 10 for all Zc and bgn=2

Below is the test result

NMS alpha searching for - 0.5dB Eb/N0 and L=32 5G LDPC									
5G LDPC configuration					BLER for different alpha value				
Zc	bgn	N	max line weight	min line weight	0.1	0.3	0.5	0.7	0.9
8	1	544	19	3	1	0.88	0.245	0.16	0.44
8	2	416	10	3	0.995	0.17	0.00375	0.0005	0.005
12	1	816	19	3	1	0.935	0.19	0.085	0.405
12	2	624	10	3	1	0.285	0	0	0.0005
28	1	1904	19	3	1	0.995	0.055	0.0125	0.35
28	2	1456	10	3	1	0.425	0	0	0
40	1	2720	19	3	1	1	0.02	0.0025	0.36
40	2	2080	10	3	1	0.405	0	0	0
72	1	4896	19	3	1	1	0.02	0.0005	0.35

72	2	3744	10	3	1	0.78	0	0	0
176	1	11968	19	3	1	1	0.035	0	0.21
176	2	9152	10	3	1	0.97	0	0	0
208	1	14144	19	3	1	1	0.055	0	0.145
208	2	10816	10	3	1	0.975	0	0	0
384	1	26112	19	3	1	1	0.05	0	0.115

OMS beta searching for -0.5dB Eb/N0 and L=16 5G LDPC									
5G LDPC configuration					BLER for different beta value				
Zc	bgn	N	max line weight	min line weight	0.1	0.3	0.5	0.7	0.9
12	1	816	19	3	0.54	0.245	0.2	0.195	0.44
12	2	624	10	3	0	0.0005	0	0	0
28	1	1904	19	3	0.54	0.14	0.09	0.09	0.295
28	2	1456	10	3	0	0	0	0	0
40	1	2720	19	3	0.505	0.19	0.0225	0.05	0.285
40	2	2080	10	3	0	0	0	0	0
72	1	4896	19	3	0.7	0.095	0.00625	0.02	0.355
72	2	3744	10	3	0	0	0	0	0
176	1	11968	19	3	0.685	0.0175	0.0005	0.0025	0.43
176	2	9152	10	3	0	0	0	0	0
208	1	14144	19	3	0.735	0.02	0	0.002	0.48
208	2	10816	10	3	0	0	0	0	0

mixed MS [alpha,beta] searching for L=16 5G LDPC									
5G LDPC configuration						[alpha, bler] pair and BLER			
Zc	bgn	N	max line weight	min line weight	Eb/N0	[0.7,0.5]	[0.7,0.3]	[0.5,0.5]	[0.8,0.3]
12	1	816	19	3	-1dB	0.55	0.32	0.96	0.26
12	1	816	19	3	-0.5dB	0.12	0.075	0.675	0.035
12	2	624	10	3	-1dB	0	0	0.06	0
12	2	624	10	3	-0.5dB	0	0	0.006	0
28	1	1904	19	3	-1dB	0.385	0.14	0.99	0.095
28	1	1904	19	3	-0.5dB	0.022	0.0025	0.74	0.00625

6 LDPC Simulation and comparison with from MATLAB toolbox LDPC decoder

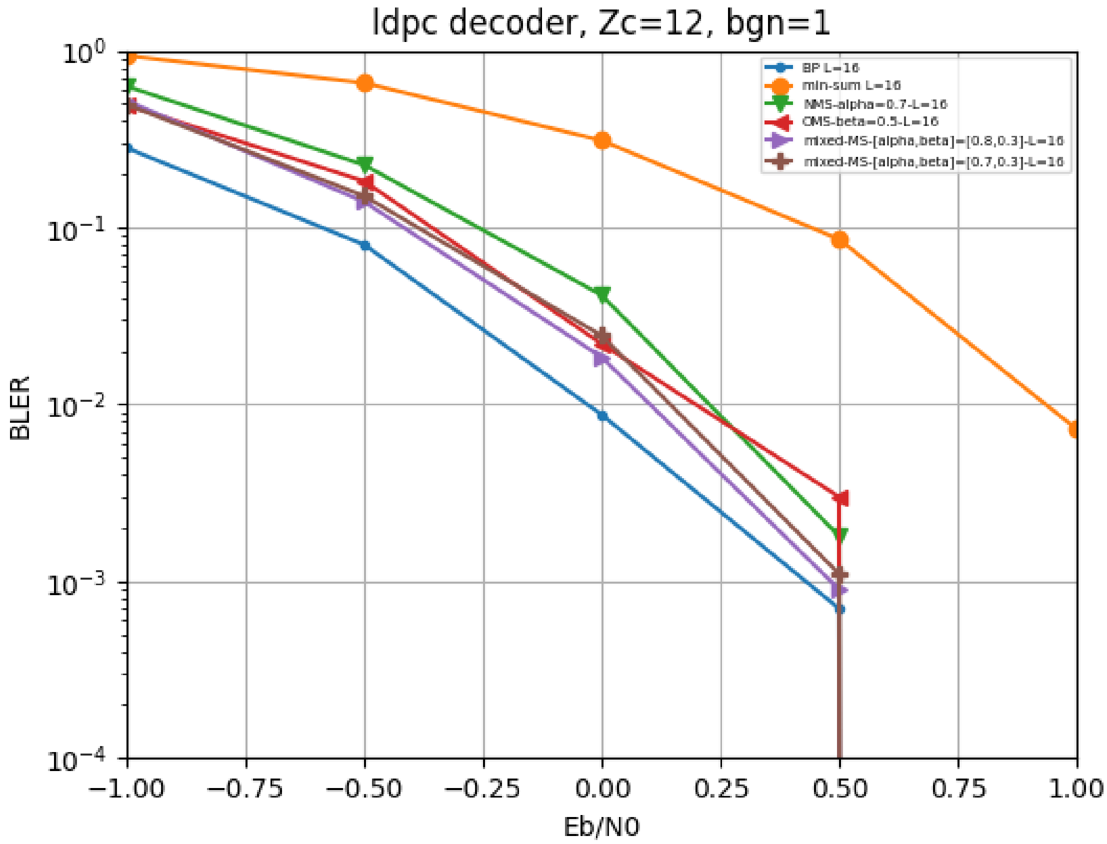
6.1 Summary of LDPC simulation and performance analysis

- $\alpha = 0.7$ is optimized for NMS for all LDPC configuration
- $\beta = 0.5$ is optimized for OMS for all LDPC configuration
- $\alpha = 0.8,\ \beta = 0.3$ is optimized for mixed-MS for all LDPC configuration
- BP has the best performance
- Mixed-MS performance is better than other mix-sum algorithms
  - 0.1dB worse than BP,
  - 0.1dB better than NMS and OMS
  - 0.75dB better than traditional min-sum
- L=32 and L=64 performance is close to each other and is 0.2dB better than L=16
-

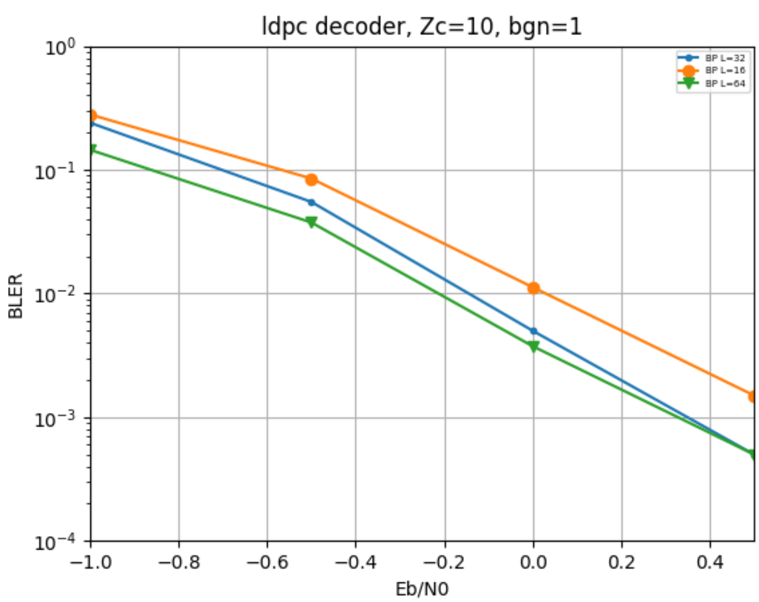
6.2 Different LDPC decoder simulation comparison

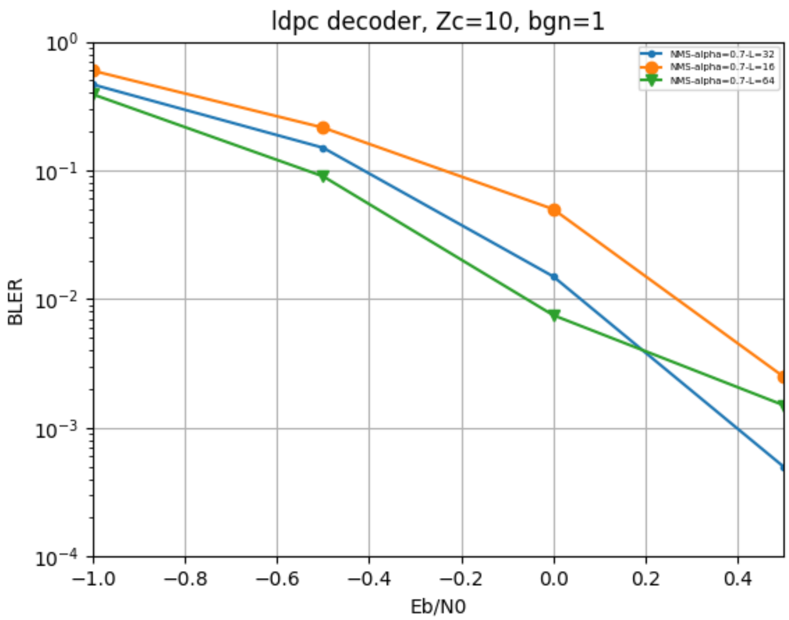
Below test used optimized α, β values.

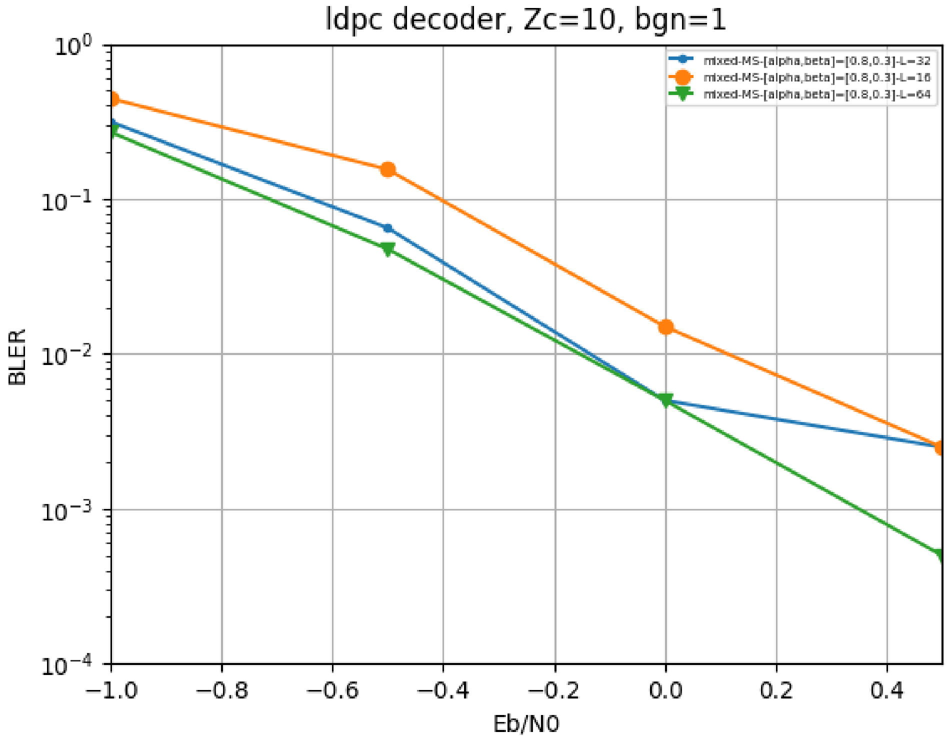
It shows that mixed MS with $\alpha = 0.8,\ \beta = 0.3$ has the best performance among all min-sum algorithm

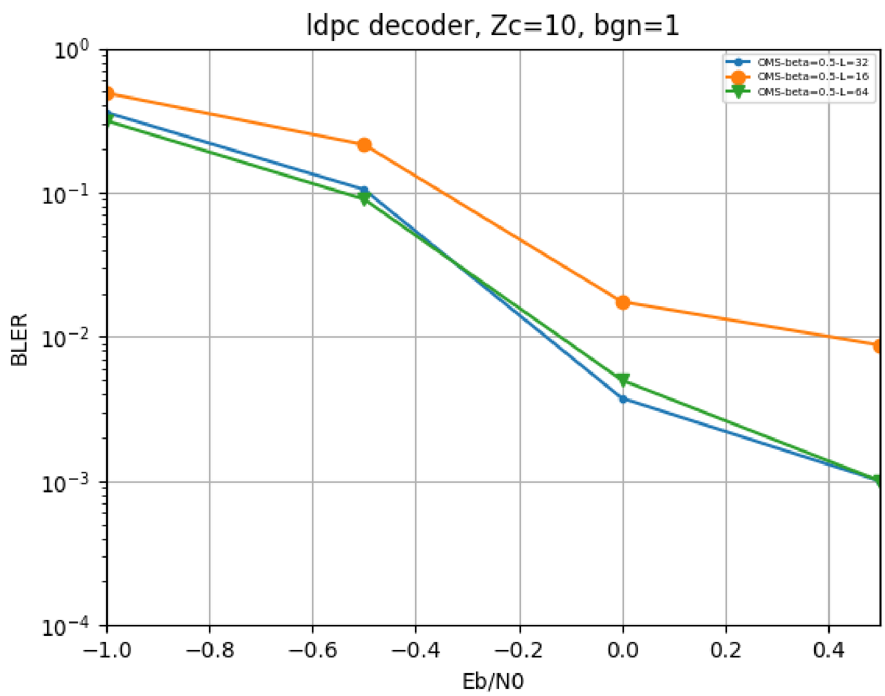


6.3 LDPC decoder performance with different iteration L value

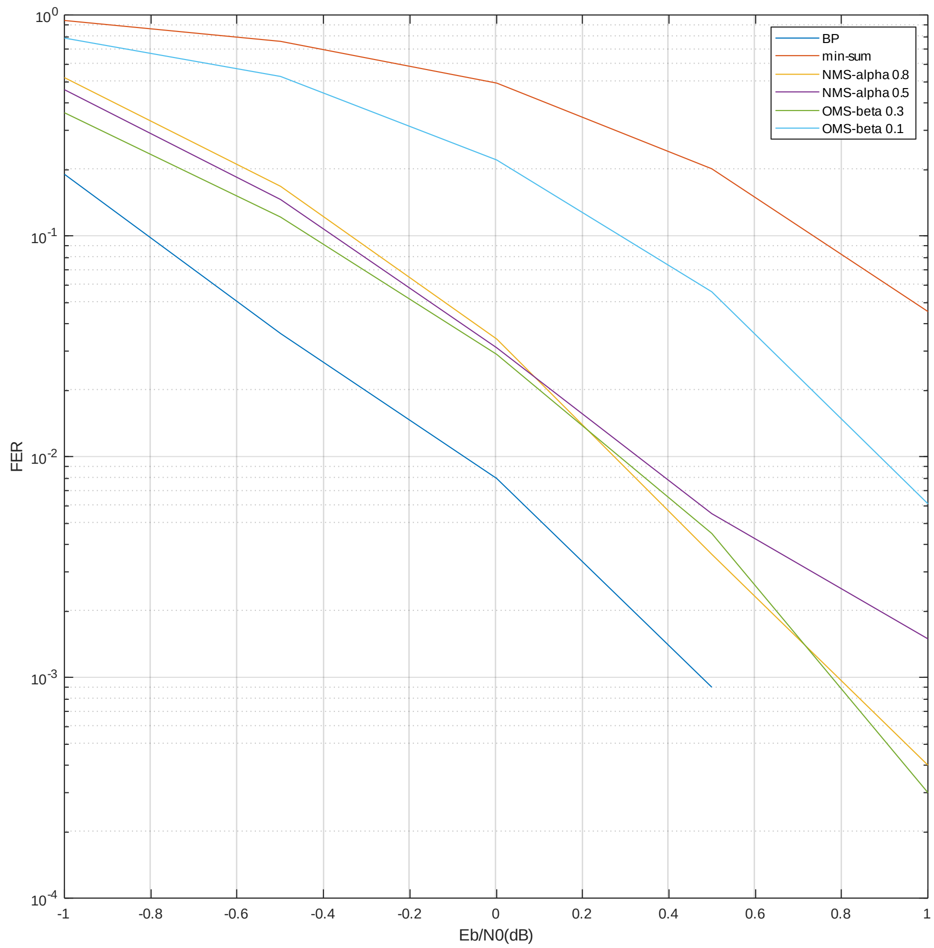


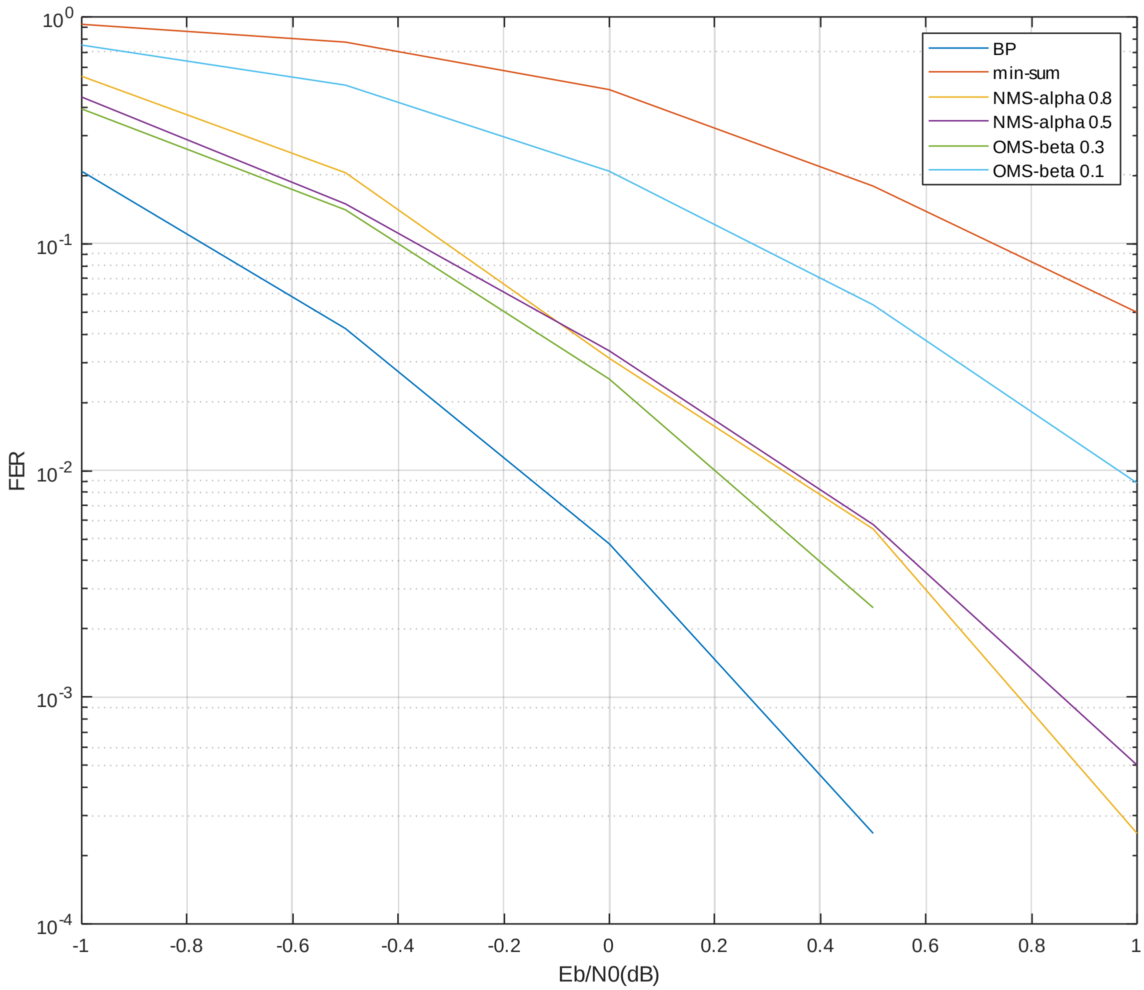

ldpc decoder, Zc=10, bgn=1
NMS-alpha=0.7-L=32
NMS-alpha=0.7-L=16
NMS-alpha=0.7-L=64
BLER
Eb/N0
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4


ldpc decoder, Zc=10, bgn=1
mixed-MS-[alpha,beta]=[0.8,0.3]-L=32
mixed-MS-[alpha,beta]=[0.8,0.3]-L=16
mixed-MS-[alpha,beta]=[0.8,0.3]-L=64
BLER
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
Eb/N0


ldpc decoder, Zc=10, bgn=1
OMS-beta=0.5-L=32
OMS-beta=0.5-L=16
OMS-beta=0.5-L=64
BLER
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
Eb/N0

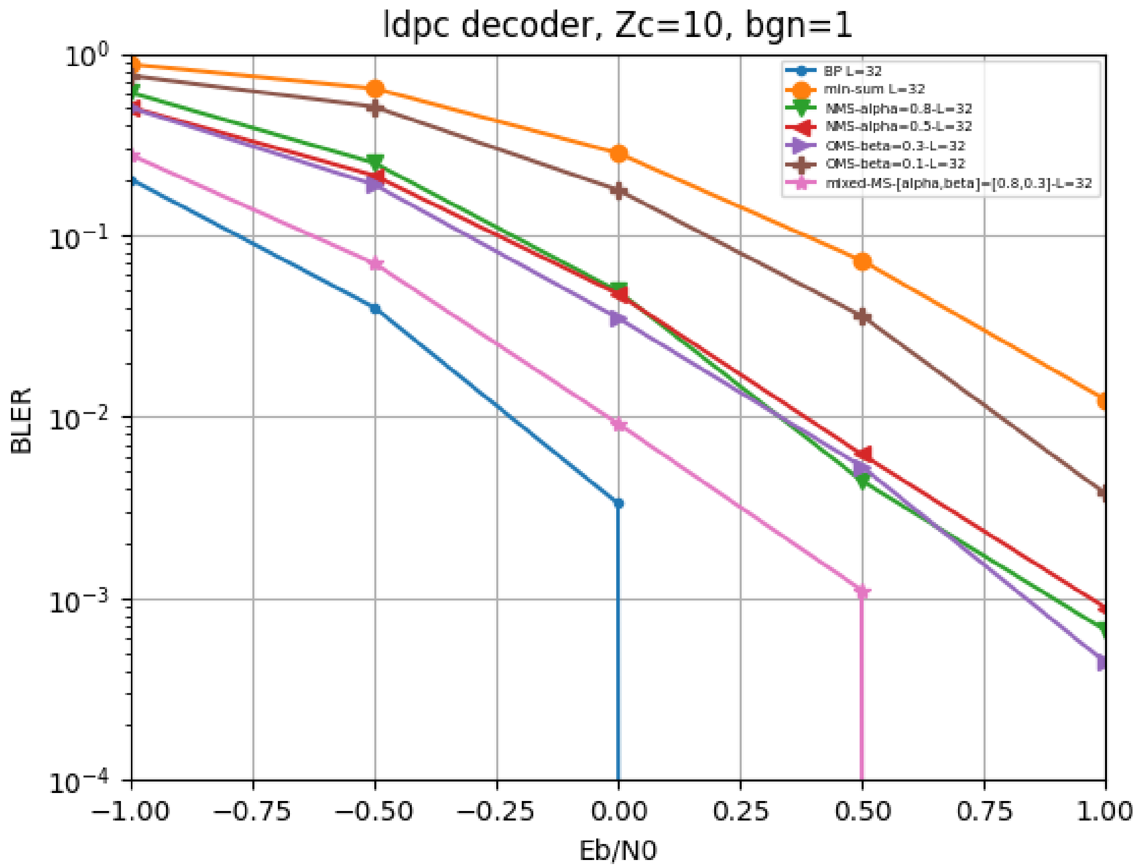
6.4 Matlab toolbox LDPC decoder simulation





BP
min-sum
NMS-alpha 0.8
NMS-alpha 0.5
OMS-beta 0.3
OMS-beta 0.1
FER
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
Eb/N0(dB)

6.5 Py5gphy LDPC decoder simulation



6.6 Test analysis and comparison with Matlab toolbox

Test configuration for both Py5gphy and matlab code

- Zc = 10, bgn = 1 which means K = Zc*22=220, N = Zc*66==660
- Algorithm: BP, min-sim, normalized min-sum, offset min-sum and mixed-min-sum(Py5gphy only)
- Alpha values used for Normalized min-sim: [0.8, 0.5]
- Beta values used for offset min-sum: [0.3, 0.1]
- Alpha and beta pair used for mixed min-sum: [0.8, 0.3]
- LDPC decoder iteration number: 32

BLER result

Eb/N0 db	-1	-0.5	0	0.5	1
Matlab BP	0.21	0.0425	0.0047	0.0003	0
Py5gphy BP	0.203	0.04	0.0033	0	0
Matlab min-sum	0.93	0.78	0.4758	0.1807	0.0495
Py5gphy min-sum	0.87	0.64	0.28	0.073	0.012
Matlab NMS alpha 0.8	0.545	0.205	0.0313	0.0055	0.0003
Py5gphy NMS alpha 0.8	0.61	0.25	0.049	0.0044	0.00067
Matlab NMS alpha 0.5	0.445	0.15	0.0338	0.0057	0.0005
Py5gphy NMS alpha 0.5	0.51	0.21	0.048	0.0062	0.00089
Matlab OMS beta 0.3	0.3925	0.14	0.0253	0.0025	0
Py5gphy OMS beta 0.3	0.503	0.19	0.035	0.0053	0.00044
Matlab OMS beta 0.1	0.75	0.5025	0.2075	0.0535	0.0088
Py5gphy OMS beta 0.1	0.76	0.51	0.178	0.036	0.0038
Py5gphy mixed MS	0.28	0.07	0.0092	0.0011	0

Observation

- Py5gphy BP performance is a little better than Matlab BP

  In another test, above BLER for snr_db[-1,-0.5,0] is [0.13,0.03,0.0025]
- Normalized min-sum and offset min-sum performance is a little worse than matlab code
- Mixed min-sum performance is much better than Normalized min-sum and offset min-sum. It is very close to BP performance