Investigations of Low-Density Parity-Check Codes

King Yau

Abstract

LDPC code is a linear error-correcting code firstly implemented by Gallager in 1963 [1]. In the 1990s, the code is rediscovered by David J.C. MacKay which stated that the performance is almost as close to the Shannon limit [2]. For the sparse property, the LDPC code can be encoded and decoded quickly by various algorithms. In this project, a random code with uniform distribution will be encoded by LDGM code and transmitted the message which adding a noise. After that, the message will be decoded by the sum-product algorithm and record the performance. Such a performance would be measured as a probability block error and probability bit error with SNR.

Index Terms - LDPC code, sum-product algorithm, Tanner graphs, factor graphs

1. Introduction

Low density parity-check matrix (LDPC) code is a linear error-correcting code which include sparse nonzero entries of matrix which means most of the entries of matrix are zero. Unlikely other types of error-correcting code, the parity-check matrix is first constructed, and then the generator matrix which called a low-density generator matrix (LDGM) will be constructed along with the LDPC code.

LDPC code has many generations and decoding methods. There are many contributions of LDPC code from people already. However, decoding algorithms and various spheres of LDPC code still not to be studied yet. For instance, the performance and signal-to-noise-ratio (SNR) of the generation of LDPC code by various rules.

In this project, the simulation would try to generate an LDPC code and LDGM. After that, the original uniformly random code M in $GF(2)$ will be generated to multiple the LDGM and adding redundancy as $C = MG$. It will be expressed as $+A$ to "1" and $-A$ to "0". Then the noise Z will be added in the channel to simulate the transmission noise such that the receiving codeword of $\hat{C}$. If $\hat{C}H^T = 0$, there is no error that happened. Otherwise, the decoding algorithm will correct the code and output the corrected result. Maybe they corrected result is not the same as the original message M. That will be counted as a bit error rate P_e and block error rate P_{block} of the codeword and make a summation of the corrupted code together in order to analyze them. Finally, the simulation will compare the different block length LDPC codes and unencoded messages to look at the performance on bit error rate and block error rate, and Signal-to-noise ratio (SNR) by plotting a graph in logarithm form.

2. LDPC and LDGM Generation

Most of LDPC code has followed properties:

a. LDPC code can be further split by regular LDPC code and irregular LDPC code. Regular LDPC code has (w_c, w_r) which is the weight of each column and the weight of each row, irregular does not have this limitation.

b. LDPC code could be expressed by the graph. Mostly is the Tanner graph. For instance, a 5×3 LDPC H the tanner graph form can be expressed by *Fig. 1*. Where

$[x_1, x_{2,} x_3, x_{4,} x_5] \in X$ are the digit nodes with 5 degrees , $[w_{1,} w_{2,} w_3] \in X$ is subcode nodes with 3 degrees . If the edges of $x_{1,2,\dots,n}$ and $w_{1,2,\dots,k}$ is equally with w_c and w_r. That called the regular LDPC code, if not, then called the irregular LDPC code.

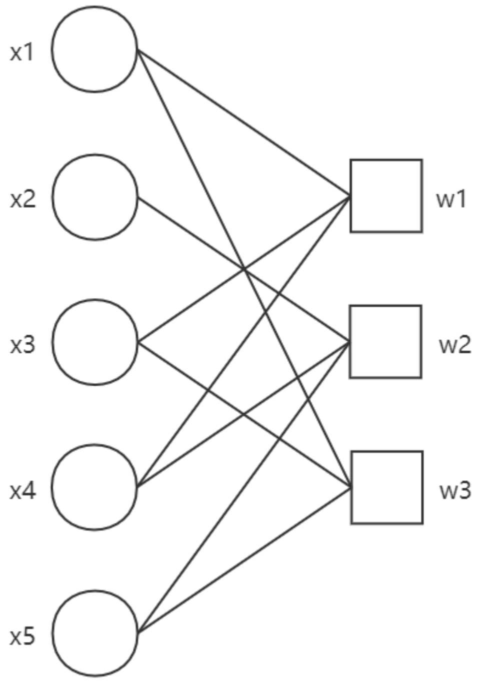

Fig.1: Tanner graph expressed with H

c. When constructing the LDPC code, there are needed to be avoiding the 4-cycle and 6-cycle to compress the time in decoding. If 4-cycle exists, using sum-product-algorithm (SPA) program for each iteration will have significantly impact on the cycle part and the result will be unpredictable.

Fully random construction and algebra construction of the matrix are common ways to generate the LDPC code. That can be taken of both of advantages together and construct an LDPC for the size of $m \times n$ including the avoiding short cycle and mapping the LDGM easily, that is semi-random/semi-algebraic constructions. However, there exist difficulties to generate the LDPC code through those methods, one is to build as a sparse matrix, that the zero elements should have an absolute percentage than the nonzero entries in the matrix. The other consideration is the complication if using fully random construction in long block length, then the weight of column and row would be hard to manipulate.

Many resource and reference can be found for building a LDPC code, the strategy of this simulation is building a Quasi-cyclic (QC) LDPC. QC LDPC is built by shift identity matrix that has already generated by the original generated matrix by semi-random/semi-algebraic constructions.

H (show in *Fig. 2*) is the matrix of generation by algebra, for building a regular matrix, we can let the matrix to become an new matrix $\widehat{H}$ (shown in *Fig. 3*) which each element become an identity-matrix with $\mathrm{k} \times \mathrm{k}$ size. Therefore, the size of $\widehat{H}$ is $5k \times 3k$. However, the weight of row and weight have not changed.

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

$$\widehat{H} = \begin{bmatrix} I_{(0,0)} & 0 & I_{(0,2)} & I_{(0,3)} & 0 \\ 0 & I_{(1,1)} & 0 & I_{(1,3)} & I_{(1,4)} \\ I_{(2,0)} & 0 & I_{(2,2)} & 0 & I_{(2,4)} \end{bmatrix}$$

Fig.2: Matrix represented by matrix

Let $s = 3$, $I_{(x,y)}$ represented as the circulant permutation matrix which is $I_{(x,y)} = row\text{shift}\,(x + y\,mod\,s)$ and

$$I_{(0,0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad I_{(0,1)} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$I_{(1,1)} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad I_{(2,1)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Fig3: The circulant permutation matrix progress

In that case $I_{(2,1)} = I_{(0,0)}$ Therefore, $I_{(x,y)}$ has 3 different types for all $I_{(x,y)}$. $\widehat{H}$ as LDPC code can be established. It also has a sparse property for nonzero entries is greater than others. One potential problem in here is that H and $\widehat{H}$ include 4-cycle represented as $H_{(0,0)}, H_{(0,2)}, H_{(2,0)}, H_{(1,1)}$. It is not a good LDPC code for decoding.

Following the above process, it can get the LDGM code G. Let $G = [\, I_{k\times k} | P \,]$, that can get by subtracted row through LDPC code to be $H = [\, P \mid I_{k\times k}]$ and exchange it.

3. Noise Channel and Decoding for sum-product-algorithm

There are three common noise channel, Binary symmetric channel (BSC), binary erasure channel (BEC) and binary input additive white Gaussian noise (BIAWGN).

For BSC, there are two inputting nodes and two outputting nodes. p is the probability of the sending side which can transmit to the received side. $1-p$ means that it failed to transmit and has been detected as the opposite result.

For BEC, the inputting part is same as BSC. $1-p$ means that it failed to transmit but the received value is uncertain. The generated code of both channels can be instead of the original message directly.

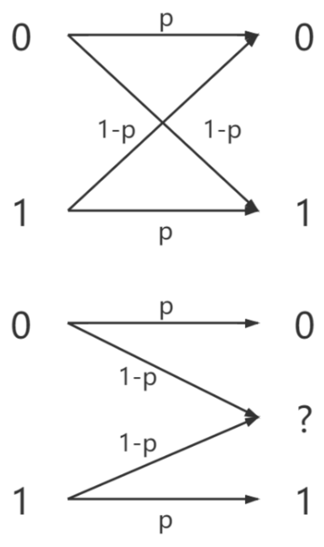

Fig.4 : BSC ,BEC with graph representation

For AWGN channel, that is added the noise to the message by normal distribution. Let noise $Z = N(0,\sigma^2)$ and codewords C . The received message will be $\hat{C} = C + Z$

Therefore, the received message has been set. From the side of decoding algorithm, we can use sum-product-algorithm. SPA is very effectively method to decode the LDPC codes.

For BEC channel with received code $\hat{c}_i$, assuming a given Log-Likelihood Ratio (LLR) λ_i has defined as

$$\lambda_i = \log\left(\frac{\mu_i(0)}{\mu_i(1)}\right) = \begin{cases} \log\left(\frac{1-\delta}{0}\right) = +\infty \,, if\ \hat{c}_i = 0 \\ \log\left(\frac{\delta}{\delta}\right) = 0, \qquad if\ \hat{c}_i = ? \\ \log\left(\frac{0}{1-\delta}\right) = -\infty, if\ \hat{c}_i = 1 \end{cases}$$

$\mu_i(0)$ and $\mu_i(1)$ is expressed as the probability of the message is "0" or "1".

For BIAWGN channel, the LLR λ_i can be defined as

$$\lambda_i = \log\left(\frac{exp\left(-\frac{1}{2\sigma^2}(x + \sqrt{E_b})^2\right)}{exp\left(-\frac{1}{2\sigma^2}(x - \sqrt{E_b})^2\right)}\right) = -\frac{2\sqrt{E_b}}{\sigma^2}\hat{c}_i$$

Then the LLR of $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_n)$ The initially check-node-to-variable-node messages are initialized to be $\lambda_{c_j\to x_i}^{(-0.5)} := 0$

According the work with [3][4], the factor graph of the SPA can be expressed by *fig.5*:

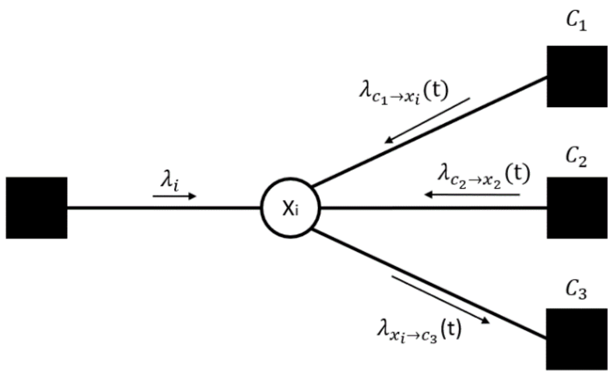
C_1
$\lambda_{c_1\to x_i}(t)$
C_2
λ_i
$\lambda_{c_2\to x_2}(t)$
Xi
C_3
$\lambda_{x_i\to c_3}(t)$

Fig.5 : factor graph represented by part of code in SPA

$$P = \left[\begin{array}{ccccccc|ccccccc} I(1) & 0 & I(2) & I(3) & 0 & 0 & 0 & I(0) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & I(1) & 0 & I(2) & I(3) & 0 & 0 & I(0) & I(0) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I(1) & 0 & I(2) & I(3) & 0 & 0 & I(0) & I(0) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I(1) & 0 & I(2) & I(3) & 0 & 0 & I(0) & I(0) & 0 & 0 & 0 \\ I(3) & 0 & 0 & 0 & I(1) & 0 & I(2) & 0 & 0 & 0 & I(0) & I(0) & 0 & 0 \\ I(2) & I(3) & 0 & 0 & 0 & I(1) & 0 & 0 & 0 & 0 & 0 & I(0) & I(0) & 0 \\ 0 & I(2) & I(3) & 0 & 0 & 0 & I(1) & 0 & 0 & 0 & 0 & 0 & I(0) & I(0) \end{array}\right]$$

Fig.6: The parity-check matrix P

Let iteration k time, for each iteration, the variable-node-to-check-node messages with every t iteration times:

$$\lambda_{X_i \to C_j}^{(t)} = \lambda_i + \sum_{C_{j'} \in N(X_i) \setminus C_j} \lambda_{X_i \to C_j}^{(t-0.5)}$$

$$\lambda_{C_j \to X_i}^{(t+0.5)} = 2 \cdot \operatorname{arctanh}\left(\prod_{X_{i'} \in N(C_j) \setminus x_i} \tanh\left(\lambda_{X_{i'} \to C_j}^{(t)}/2\right) \right)$$

Minding that $\lambda_{C_j \to X_i}^{(t+0.5)}$ is too complicate for calculation, so it can be also rewritten as

$$\lambda_{C_j \to X_i}^{(t+0.5)} = \left(\prod_{X_{i'} \in N(C_j) \setminus x_i} \lambda_{X_{i'} \to C_j}^{(t)} \right)$$

For the final $\lambda_{C_j \to X_i}^{(k)}$, we can define the final message $\widehat{m}$ to be

$$\widehat{m} \triangleq \begin{cases} \text{"0", if } (\mu(0), \mu(1)) \propto (1,0), i.e., \lambda = +\infty \\ \text{"?", if } (\mu(0), \mu(1)) \propto \left(\frac{1}{2}, \frac{1}{2}\right), i.e., \lambda = 0 \\ \text{"1", if } (\mu(0), \mu(1)) \propto (0,1), i.e., \lambda = -\infty \end{cases}$$

It needs to be mentioned the deciding threshold after decoding. The threshold should great than 0 such that the only white noise has not been recognized as a message signal. In this example, we can choose σ as threshold which if $\lambda > \sigma, \lambda < \sigma$, otherwise, the message will not be detected. In sum, $\widehat{m}$ is the result of the code that we get.

4. Methodology and experiment

At first, the code rate should be considered as k/n where n is the total matrix length and k is the message length. Then setting the SNR to get the standard derivation σ . After choosing a noise channel, we can run the code and try to get the experiment result.

The SNR can be expressed as

$$SNR_c \triangleq \frac{E_c}{N_0} = \frac{A^2}{2\sigma^2}$$

E_c is the energy per codeword symbol. N_0 is the one-sided power spectral density of AWGN where $\sigma^2 = \frac{N_0}{2}$. It is more convenient for us to convert the SNR form through dividing code rate $R = \frac{k}{n}$ when compare coding schemes with different type of coding and SNR_c can be formed as

$$SNR_c \triangleq \frac{E_c}{N_0} = \frac{A^2}{2\sigma^2}$$

Because the y-axis of the BER is commonly expressed by logarithm form, SNR_c should be in $[dB]$ form, which is $SNR_c[dB] = 10 \log_{10} SNR_c$.

After one simulation, we can get the final message result. The result can compare with the original message for each element. After that, we can get the probability bit error and probability block error and plotting the SNR on the channels.

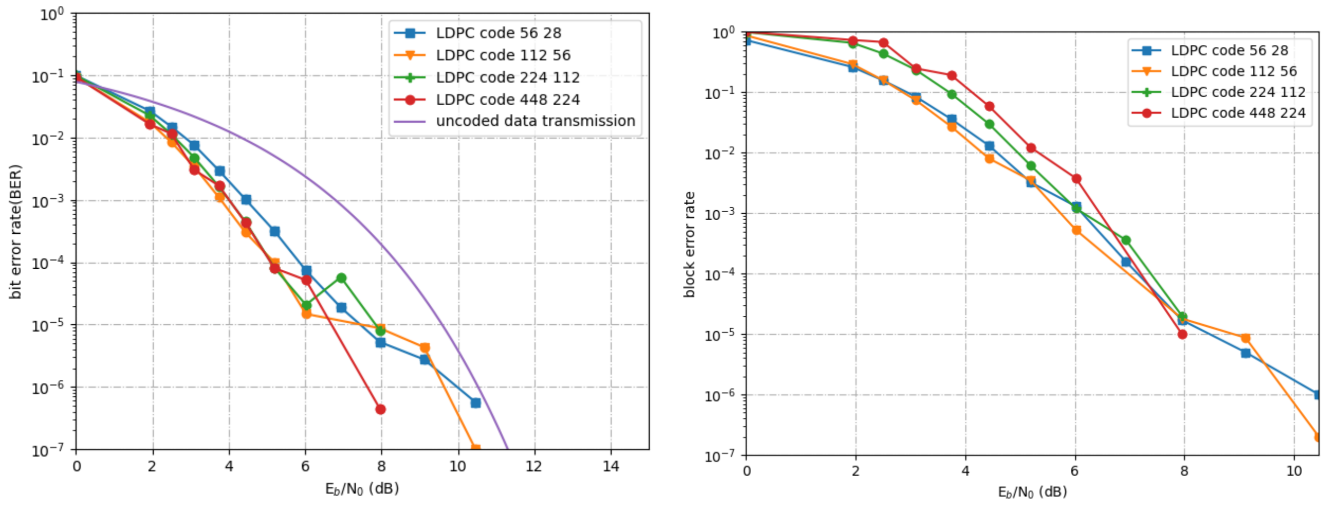

Fig.7 : Performance of the LDPC code

It is seemingly that only one time is not enough, the simulation should change the SNR and make it as a point as $(p_e, SNR_c[dB])$. The message passing programs should be simulated many times. Usually, if we want to find the probability bit error rate $P_e < 10^n$, the message passing time through the channel should be 10^{n+2} time. The passing time can sure that the probability bit error is stable variable.

Given a (n,k,d) LDPC code with parity-check matrix P, generator matrix G and the original message M. The prepared sending codeword $C = MG$ will be mapped as $B(C) = \begin{cases} 0 \to 1 \\ 1 \to -1 \end{cases}$ in BPSK form. Then, the codeword C will be across the AWGN channel, the receiving codeword become $\widehat{C} = C + N$, N is the noise.

For $\widehat{C}$, it should be changed by LLR λ and do the SPA in part3 with iteration r times that can output the final message $\widehat{M}$. $\widehat{M}$ can be compared to the original message M. The bit error probability p_e is $p_e = p(M \neq \widehat{M})$ rate. The BER would be the average of P_e the message passing time m.

Taken a typical LDPC code as an example, the code P be a parity-check matrix in *fig. 6*, it is easy to implement the generator matrix for subtracting matrix row for $(1)-(2),(2)-(3),\ldots,(6)-(7)$. The $I(x)$ mean that the shift matrix with x time. After that, the generator matrix G and parity-check matrix P are established.

Then, we can follow above rule and run the code to record the result of BER by SPA decoding and try to make different length of the identity matrix in *fig. 6* to investigate the difference between those matrices. The result record is given below in appendix.

Fig. 7 show the performance of the LDPC that parity-check matrix in fig.6, the lowest length of fig.6 matrix is (56,28) for the shifted matrix. Each point of the curve is the block error rate and BER with SNR. From the trend of the graph, we can observe in three dimensions including iteration, length, and SNR. If the SNR is increasing, the BER and the block error rate is also decreasing. If the length of the codeword is increasing, the BER will be lower and the block error rate will be larger. For iteration, because the LDPC code has not 4-cycle and the weight of column is 3, if the iteration is greater than the weight of column, there are not seemingly impact on the performance. However, less iteration time significantly change the performance.

5. Conclusion

In this simulation, we simulate the environment to transmit the data and look at the error bit rate and error block rate to consider how to improve its performance and do it again. In future, we will implement the LDPC with Product-code structure to do the parallel codeword decoding to improve decoding speed of LDPC code.

However, this simulation still has a wide space to be improved. In future, it should be less error bit rate and error block rate than nowadays.

6. Reference

[1] MacKay, D. J. C. and Neal, R. M. (1996) ``Near Shannon limit performance of low density parity check codes'', Electronics Letters, vol. 32, pp. 1645-1646.

[2] D.J.C MacKay and R. M. Neal, “Good codes based on very sparse matrices,” in Cryptography and Coding. 5th IMA Conference (Lecture Notes in Computer Science), C. Boyd, Ed. Berlin, Germany: Springer, 1995, vol. 1025, pp. 100-111.

[3] F. R. Kschischang, B. J. Frey and H. -. Loeliger, "Factor graphs and the sum-product algorithm," in IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 498-519, Feb 2001, doi: 10.1109/18.910572.

[4] H.-A. Loeliger, “An introduction to factor graphs,” IEEE Signal Process. Mag., vol. 21, pp. 28–41, Jan. 2004.

7. Appendix1: the record of running LDPC code with iteration 5 times

LDPC_code	Block error rate	SNRbDB	SNRcDB	BER	sending num	time
LDPC code 56 28	0.731	0	-3.0103	0.098909	1000000	7266.9
LDPC code 56 28	0.26332	1.9382	-1.0721	0.02671	100000	438.39
LDPC code 56 28	0.15798	2.4988	-0.5115	0.01491	100000	360.55
LDPC code 56 28	0.08479	3.0983	0.0877	0.007556	100000	286.24
LDPC code 56 28	0.03633	3.7417	0.7314	0.002973	100000	219.57
LDPC code 56 28	0.01311	4.437	1.4267	0.001023	100000	168.29
LDPC code 56 28	0.00326	5.1927	2.1824	0.0003189	100000	132.51
LDPC code 56 28	0.0013	6.0206	3.0103	7.40E-05	1000000	1042.14
LDPC code 56 28	0.000159	6.9357	3.9254	1.88E-05	1000000	773.67
LDPC code 56 28	1.70E-05	7.9588	4.9485	5.25E-06	1000000	518.41
LDPC code 56 28	5.00E-06	9.1186	6.1083	2.75E-06	1000000	332.75
LDPC code 56 28	1.00E-06	10.4576	7.4473	5.71E-07	1000000	248.86
LDPC code 112 56	0.8738	0	-3.0103	0.09424	10000	208.41
LDPC code 112 56	0.2939	1.9382	-1.0721	0.01747	10000	135.7
LDPC code 112 56	0.15935	2.4988	-0.5115	0.008413	100000	1095.37
LDPC code 112 56	0.07422	3.098	0.0877	0.003402	100000	843.98
LDPC code 112 56	0.0273	3.7417	0.7314	0.001103	100000	630.22
LDPC code 112 56	0.008032	4.437	1.4267	0.0003009	1000000	4723.27
LDPC code 112 56	0.0035	5.1927	2.1824	0.0001	100000	361.48
LDPC code 112 56	0.00053	6.0206	3.0103	1.48E-05	100000	287.47
LDPC code 112 56	1.80E-05	7.9588	4.9485	8.73E-06	1000000	1166.74
LDPC code 112 56	8.70E-06	9.1186	6.1083	4.33E-06	10000000	5285.56
LDPC code 112 56	2.00E-07	10.4576	7.4473	1.02E-07	10000000	1657.86
LDPC code 448 224	1	0	-3.0103	0.094147	1000	254.83
LDPC code 448 224	0.736	1.9382	-1.0721	0.01631	1000	233.1
LDPC code 448 224	0.6809	2.4988	-0.5115	0.0115	1000	122.45
LDPC code 448 224	0.2487	3.098	0.0877	0.003033	10000	1567.07
LDPC code 448 224	0.1939	3.7417	0.7314	0.001694	10000	975.83
LDPC code 448 224	0.0588	4.437	1.4267	0.0004353	10000	773.19
LDPC code 448 224	0.0123	5.1927	2.1824	8.08E-05	10000	579.35
LDPC code 448 224	0.0038	6.0206	3.0103	5.18E-05	10000	425.84
LDPC code 448 224	1.00E-05	7.9588	4.9485	4.46E-07	10000	272.97
LDPC code 224 112	0.9233	0	-3.0103	0.099	10000	109.49
LDPC code 224 112	0.9939	0	-3.0103	0.09884	10000	353.47
LDPC code 224 112	0.6592	1.9382	-1.0721	0.02248	10000	335.98
LDPC code 224 112	0.4391	2.4988	-0.5115	0.01127	10000	319.98

LDPC code 224 112	0.2358	3.098	0.0877	0.004784	10000	278.72
LDPC code 224 112	0.0958	3.7417	0.7314	0.001642	100000	2311.15
LDPC code 224 112	0.0299	4.437	1.4267	0.0004458	100000	1798.87
LDPC code 224 112	0.0062	5.1927	2.1824	8.23E-05	100000	1387.71
LDPC code 224 112	0.0012	6.0206	3.0103	2.07E-05	100000	1049.33
LDPC code 224 112	0.00036	6.9357	3.9254	5.77E-05	100000	857.1
LDPC code 224 112	2.00E-05	7.9588	4.9485	8.04E-06	100000	634.1

Weekly Logbook For FYP

Name: Yau King Yuen

SID:1155110900

Date:21-9-2020

Topic: Student Self-Proposed Topic

Log:

(**Week2**)13-9-2020 – 21-9-2020:

Learning RS-code and information theory, prepare to learn LDPC code and decide the topic.

(**Week3**) 22-9-2020 – 28-9-2020:

Learning LDPC code, factor graph, SPA (sum product algorithm) and prepare to learn polar code.

Reading paper https://ieeexplore.ieee.org/document/1267047

(**Week4**) 29-9-2020 – 5-10-2020:

Prepare to construct LDPC encoder and decoder by Python. Learning different LDPC decoder. And find more example for practical application of LDPC.

Reading paper https://ieeexplore.ieee.org/document/910572

(**Week5**) 6-10-2020 – 12-10-2020:

Currently I am working on the decoder of LDPC by Python, I have learnt how to construct a LDPC Parity-check code and building the noisy-channel on this week. And try to construct sum-product algorithm decoding and different decoding algorithm and making a performance for the codewords.

Writing a proposal but because of no certain the specific area of LDPC, so the proposal have not completed fully.

Trying to find other papers and books that can inspire me to think which topic I can try.

(**Week6**) 13-10-2020-19-10-2020:

Currently I rewrite the code in channel part and LDPC to be clearly.

(**Week7**) 20-10-2020-26-10-2020:

Founding a new LDPC code to sure that is no 4-cycle or 6-cycle. That will be affected with the efficiency in decoding. Building a BSE channel and SNR. Hopefully can try other person code for LDPC and compared next week.

(**Week8**) 27-10-2020-2-11-2020:

I plot the SNR graph and write the code to find another LDPC code is good for my project.

(**Week9**) 3-11-2020-9-11-2020:

This week I improve my code for the performance and run it in GPU directly. Prepare to try product code message passing. Also, the LDPC code actually don’t need to change now.

(**Week10**) 10-11-2020-16-11-2020:

This week I try to make unencoded graph and write the code for product code.

(**Week11**) 17-11-2020-23-11-2020:

This week mostly I work is changing the original scaling factor on sum-product algorithm and test the performance.

(**Week12**) 24-11-2020 30-11-2020:

This week rerun the code and implement Mackay code in AWGN channel.