ITERATIVE DECODING OF SPATIALLY-COUPLED PRODUCT CODES USING GUESSING RANDOM ADDITIVE NOISE DECODING ALGORITHMS

by

Ganeshaanand (Rishi) Balasubramanian

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
September 2022

I dedicate this to my relentless parents Balasubramanian and Vidhya Balasubramanian who supported me through and throughout, in-spite of my many failures.

Table of Contents

List of Tables

List of Figures

Abstract

Forward Error Correction Codes (FEC) in Coding Theory, Information Theory and Telecommunication is used to control the errors in information that was transmitted over a noisy channel. Low Density Parity Check (LDPC) codes are linear error correcting codes used to transmit messages over noisy channels. LDPC codes are capacity approaching, meaning that practical constructions exist in which noise threshold can be set very close to Shannon Limit. LDPC codes are defined by a sparse parity check matrix. It is transmitted through Additive White Gaussian Noise channel with Binary Phase Shift keying modulation. The noisy message data is decoded using Sum Product and Min Sum decoding algorithms. Short cycles in LDPC tanner graphs are called girths. They degrade the performance as they affect the independence of extrinsic information exchanged in iterative decoding. In this research, Simulated Annealing[25][24] algorithm is used to construct QC-LDPC codes with Higher girth.

Product Codes are the combination of linear codes. It produces powerful error correcting codes through multiple low error correction capability codes. Since each code symbol appears once in the row element and once in the column element, the encoding of product codes is done in horizontal and vertical manner. Here Zipper Codes are used, a type of Product Codes.

This manuscript will be demonstrating through the research evidence gathered through various experimentation on Error Correction Codes, one of the efficient methods to achieving better Bit Error Rates. Zipper Codes, introduced in 2019, a framework for describing Spatially-Coupled Interleaved Codes with Sliding Window - Iterative Decoding Scheme. Here BCH Codes are used to demonstrate Zipper Codes.

Followed by that is GRAND - Decoder. A new form of decoding strategy where the error patterns are determined rather than determining entire code-words. Together this research intends to demonstrate that the system is able to efficiently decode at rates of 10^{-7} by Sound-Noise-Ratios (SNR) 5dB in the Additive White Gaussian Noise Channel.

Acknowledgements

My sincerest gratitude to my Supervisor Prof. Dr. Dmitry Trukhachev who helped me in many ways possible throughout the journey of my Master's Degree. I will be forever grateful as to have been chosen in random among thousands of applications that applied to Dalhousie University to work with you.

I wish to thank the eminent Thesis Defense Committee members Dr Edward Yao from Engineering and Mathematics Department and Dr Hamed Aly from the Electrical and Computer Department for agreeing to review my thesis and extending their presence to the Defense.

I would like to thank the Department Administrator Tamara Cantrill and Secretary Nicole Smith and Ola Hamada for their timely help with support and defense paperwork.

This research manuscript acknowledges the support of PhD Student Reza Hadavian for his round the clock support with coding the decoders and help in debugging. Personally I'd like to thank you for putting up with my tantrums when results kept failing until we aced it.

I'd like to Specially thank the Author of Zipper Codes Alvin Sukhmadji for his timely support and inputs. This could not have been possible without his help.

Special thanks to Digital Research Alliance of Canada. Their Supercomputers greatly accelerated the speed of the research with round the clock support.

My friends Jenish, Sriram, Naren, roommates Pravesh, Aditya, Yaman and all others who have been there through this journey as pillars of mental support and occasionally brainstorming logic with me, I'm eternally grateful.

Finally I'd like to thank my Parents and brother Rahul Balasubramanian. Their support enabled this all.

This research was part of the Fiber Optic Communication Algorithms Laboratory (FOCAL) group funded by Huawei Technologies Canada Co., Ltd. It comprises of the governing bodies of University of Toronto, The University of British Columbia, Memorial University of Newfoundland and Dalhousie University, Canada.

Chapter 1

Introduction

1.0.1 Background

Error Correction Codes (ECC) in Coding Theory, Information Theory and Telecommunication is used to control the errors in information that was transmitted over a noisy channel. Though technology has developed well, noise and loss of information is inevitable. So the sender/transmitter will send encoded redundant copies of the data to the receiver where a limited number of errors can be corrected without retransmission. Richard Hamming was the first to propose a ECC known as Hamming (7, 4) code[2].

Optical Transport Networks (OTN)[15] is an industry standard protocol that provides effective means of information transmission and receiving through transport, switching and multiplexing services on a high capacity bandwidth over the Optical Network defined in protocols such as G.709 and G.798 by International Telecommunications Union - ITU.

When data transmission rates started increasing exponentially with the advent of cloud services, the OTNs started using Forward Error Correcting codes (FECs). FEC adds redundant bits to the code for transmission, and can correct specific number of errors without retransmission. The First Standards for FEC in OTNs was set in 2000 by the ITU for G.975. It is a $(255, 239)$ Reed Solomon Code with 2.5Gbps throughput.

Low Density Parity Check (LDPC) codes[3] were first invented by Robert Gallager in 1963 and was reinvented by Mackay and Neal in 1995. LDPC codes have been getting much attention as the Bit Error Rate (BER) was close to the Shannon Limit. Since then LDPC has become the Error Correcting Code of Choice. It is employed for use by NASA's Deep Space Network, Satellite Broadcast Standard for Digital Television DVB-S2 and 5th Generation New Radio for the 5G Mobile Network.

LDPC codes are Linear Error Correcting Codes which are defined by a sparse binary parity check matrix[1]. Sparsity of a matrix is defined as when the binary 1's are sparsely populated in the matrix based on LDPC Construction methods. The data to be transmitted is first encoded using this matrix and transmitted on a noisy channel to the receiver where the information is decoded. Many methods of encoding and decoding are used for data transmission depending on their efficiency rates.

For instance assume that a sparse LDPC Parity Check Matrix (PCM) of size 4x6 is created. It can contain six code-words. Each code-word is right if it can satisfy the relation with the PCM. If c is the code-word and H is the PCM then the relation is -

$$H[c]^T = \vec{0} \tag{1.1}$$

Short cycles in parity check matrix also known as girth of the code hamper the code's performance. During decoding the information cycles between the check and variable nodes. But in the presence of short cycles, the wrong information enters a loop within the short cycle and extrinsic information cannot correct the error. So construction of codes with higher girth is considered effective. This research presents Simulated Annealing[23], a method to construct LDPC codes without short cycles. With higher girth per code, the BER increases considerably.

Product Codes, also known as a type of Convolutional codes, differ from the traditional block codes, where instead of linear encoding, the encoding also comprises of data from previous encoding iteration. Product Codes can obtain Powerful Error correction systems using low error correction capacity codes such as BCH.

Zipper codes are a new framework for Spatially Coupled Product like codes, which is iteratively decoded using a sliding window decoder. Along with Guessing Random Additive Noise Decoding algorithm, have achieved better BERs in the range of 10^{-6} for SNR 5dB.

1.0.2 Scope of the Research

This thesis manuscript presents its Novel Results from the Research of Forward Error Correcting Codes. It employs Zipper Codes, with GRAND Decoding strategy to derive a better Bit Error Rate in the order of 10^{-6} for SNR 5dB. This research was conducted with the aim to design an advanced system of Forward Error Correcting Codes[4][30]. It was decided to attempt recreating various classes of codes to demonstrate the research. Beginning with Low Density Parity Check codes which can produce better Bit Error Rate (BER) per Sound to Noise Ratio (SNR dB) than current models. For demonstration purposes the simulation was coded entirely in MATLAB.

First trial was a simulation, a simple LDPC error correction code using the [155, 64, 20] Tanner code[1]. The formation of the binary parity check matrix was formulated based on Quasi Cyclic arrangement. They are algebraically created where binary identity matrices are cyclically shifted to the right and placed in a pattern. This method designed by Tanner has been deemed as the best method in terms of storage, access, encoding, decoding and analysing the performance of the code. They also perform better than its counterpart Random LDPC codes.

Using Sum Product Algorithm (SPA) and Minimum Sum Algorithm (MSA)[6] for decoding, a transmitter was created which creates a random binary data. This binary data is encoded using Modified Approximate Lower Triangular (MALT)[8] Parity check matrix, where the parity matrix is rearranged to the MALT format using row-column operations. This process will encode the binary data and add parity bits to the data with the entire transmittable data being 155 bits long.

This encoded data is modulated using Binary Phase Shift Keying (BPSK) and then passed through the Additive White Gaussian Noise (AWGN)[17] channel[10], which adds random noise to the encoded data for the purpose of the simulation. The receiver will decode the data using SPA and MSA decoders for SNR ratios ranging from 1-5 dB. This produced a better BER rate than random codes.

Post that, the research tackled the problem of short cycles in LDPC codes. After study of various algorithms designed to reduce the short cycles in the Tanner graph of LDPC codes, Simulated Annealing[23], was employed to reduce the number of short cycles in the Tanner Graph which resulted in an improved BER compared to the ones with existing short cycles.

A new class of Spatially Coupled - Product Like Interleaved Codes - Zipper Codes[32] is introduced which shows potential for a better Bit Error Rate. The simulator was redesigned on MATLAB with features for automation with any FEC codes. The system employs Sliding Window Decoder[37], which iteratively decodes a set of code-words, until the window is completed, and the window slides down to the next mark to repeat the process. Then a design simulation using BCH codes on

Zipper with a Staircase Implementation and utilised the Berkelamp Massey [41] to compare results.

Finally this thesis presents a novel method of decoding focusing on Noise instead of code-words[43]. Guessing Random Additive Noise Decoding[40], is a new decoding strategy which tries to determine the noise pattern affecting the transmitted code instead of determining the entire code. Ordered Reliability Bits GRAND[42], is also introduced which has improved BER. With Bound Distance Decoding[13] Combination of both ORBGRAND and GRAND, this thesis achieves its motive of high BER.

1.0.3 Thesis Structure

This thesis manuscript is structured as described. Chapter 2 has the description of Low Density Parity Check Matrix. It discusses about LDPC Code Construction, Quasi Cyclic Codes. The pre-processing required for the Encoding Process and the Decoding Systems is also given followed by the Simulation Results.

Chapter 3 discusses about Girth, short cycles and Tanner Graphs. It demonstrates the use of Simulated Annealing method to reduce short cycles in Tanner Graphs of LDPC Codes. It is followed by its Simulated Results.

The fourth chapter of this manuscript contains information on Zipper Codes, a Spatially Coupled, product like Codes. The first section is about the Zipper Code format and its structure. It discusses about Staircase Codes and the code rate. The next section contains information about the Zipper Simulation Setting done in this thesis. It talks about thee Transmission Setting, Zipper Block Creation, Factor Graph and the Decoding Settings. The final section of this chapter talks about the simulated results of Zipper Hamming Codes and BCH Codes.

Fifth Chapter is about GRAND Decoder and Tiled Diagonal Zipper Codes, used in this research. It contains the algorithm, various other iterations of Grand including ORBGRAND and Bound Distance Decoding. It is finally succeeded by the Performance Evaluation results.

The sixth Chapter is the Conclusion of this thesis research explaining and summarizing this research and the results of it.

The final chapter contains the Bibliography of this research.

Chapter 2

Low Density Parity Check Codes - LDPC Codes

2.1 Error Correction

Error Correction is the process of correcting errors received after transmission which gets distorted due to many factors mainly noise. For instance, consider that system will be transmitting binary messages which consists of 0's and 1's. The basic idea of Forward Error Control is that the actual/original message is augmented or appended with redundant parity bits, which is received and decoded without any errors.

A straightforward and easy demonstration would be the (3, 1) Repetition Code. In this case, the message bits are augmented with parity bits and transmitted. The receiver could get up to eight copies.

Now this enables to user to correct errors using majority. Amongst many error correction techniques, LDPC codes stands alone as the most efficient as they are very efficient and can easily approach the Shannon limit with extreme precision.

2.2 LDPC Codes

Low Density Parity Check Codes (LDPC) are linear block codes with parity check matrix (PCM) H, which is sparsely populated. Sparseness here means that of all

Table 2.1: Simple Error Correction

Received	Error Corrected
000	0 (sans error)
001	0
010	0
100	0
111	1 (sans error)
110	1
101	1
011	1

the binary data in the parity matrix, the 1's are lesser in units compared to the 0's. This means that the decoding complexity and the minimum distance is linearly proportional to the code length. An LDPC Parity Check Matrix has columns with fixed number of non-zero elements d_v and its rows also have fixed number of non-zero elements d_c. This sparse parity check matrix is generally constructed through many ways with algebraic one being the most favored.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Here H is the Parity check matrix that is sparsely populated with $d_v = 2$ and $d_c = 3$, where row-wise or column-wise, the units of 0's supersedes 1's. Such an LDPC code defined by (d_v, d_c) are known as *regular LDPC* codes. The length of the code-word is equivalent to the number of columns present denoted by N. Consider the rows of the Parity Matrix as M where $M < N$. Assuming x as the column vector, the solution of the Parity matrix is given by -

$$H\vec{x} = \vec{0} \tag{2.2}$$

The number of linearly independent columns is characterised by K which is defined by the equation -

$$K = N - M \tag{2.3}$$

This is also is the number of information bits present in a code-word. As the sum of 1's in the rows and columns are equal, So -

$$Nd_v = Md_c \tag{2.4}$$

$$= (N - K)d_c \tag{2.5}$$

$$\frac{K}{N} = 1 - \frac{d_v}{d_c} \tag{2.6}$$

From these equations get the *code-rate* R which is defined by

$$R = \frac{K}{N} \tag{2.7}$$

That is, the average information bits present per code-word symbol. Therefore, a regular (d_v, d_c) LDPC has a code-rate of

$$R = 1 - \frac{d_v}{d_c} \tag{2.8}$$

2.3 LDPC Code Construction

There are many methods to construct an LDPC code. The common methods are random construction techniques which are not reliable, as the random construction's lack of organisation makes it disadvantageous to storing, accessing and code analysis. Adopting an algebraic method of LDPC code construction, bypasses most of the problems related to random code construction. This thesis has adopted Quasi Cyclic LDPC code construction methods for demonstration.

2.4 Quasi Cyclic LDPC Codes

Quasi Cyclic LDPC codes[1] are created algebraically, which has a definitive structure that enables it to produce better Bit Error rates. Basically, their construction has blocks of identity matrices of size m. Each of those identity matrices are shifted cyclically to the left.

For a prime number m, all the integers up to *m-1* conform to the Galois Field (m). The non zero elements of this field forms the cyclic groups. Consider a and b as two non zero elements with multiplicative factors *o(a)* = k and *o(b)* = j. Then the base/parent/mother matrix P is formed which is j *X* k in size, populated with the elements from the Galois Field of m.

$$P = \begin{bmatrix} 1 & a & a^2 & ... & a^{k-1} \\ b & ab & a^2b & ... & a^{k-1}b \\ ... & ... & ... & ... & ... \\ b^{j-1} & ab^{j-1} & a^2b^{j-1} & ... & a^{k-1}b^{j-1} \end{bmatrix} \tag{2.9}$$

The next step towards the process will be expanding the parent matrix/mother matrix or P into its binary counterparts which will form our actual LDPC Parity Check Matrix H.

$$H = \begin{bmatrix} I_1 & I_a & I_{a^2} & ... & I_{a^{k-1}} \\ I_b & I_{ab} & I_{a^2b} & ... & I_{a^{k-1}b} \\ ... & ... & ... & ... & ... \\ I_{b^{j-1}} & I_{ab^{j-1}} & I_{a^2b^{j-1}} & ... & I_{a^{k-1}b^{j-1}} \end{bmatrix} \tag{2.10}$$

So, the parity matrix H is made of a jxk sized circulant identity matrices. Here I_x, is an identity matrix of dimensions *m by m.*

The individual circulant elements of the mother matrix P are obtained by cyclically shifting the elements to the left by their Galois number. Post expansion into the Parity check matrix, will result in a binary matrix which is sparsely populated of 1's and 0's equally distributed across rows and columns in the size of *jm x km.* That is, each row of the parity matrix H will exactly have j 1's and every column will have exactly k 1's. The resultant PCM H is a *(j, k)* regular LDPC code of rate *R >= 1 - (j, k)*[11][6].

This construction method for constructing cyclic algebraic Quasi Cyclic Low Density Parity Check Codes can also be applied to a non - prime m. For any integer m, the set of non-negative integers less than m and relatively prime to m, $\mathbb{Z}_m^*$, forms a multiplicative group. Generally $\mathbb{Z}_m^*$ has order,

$$\phi(m) = m \prod_{p\,|\,m, p\, prime} (1 - 1p) \tag{2.11}$$

This is the Euler phi equation. Some common construction examples and a detailed step by step is also depicted below.

- $[155, 64, 20]$ QC Code with GF(31). With a = 2, b = 5, o(a) = 5 and o(b) = 3, the parity H is

$$H = \begin{bmatrix} I_1 & I_2 & I_4 & I_8 & I_{16} \\ I_5 & I_{10} & I_{20} & I_9 & I_{18} \\ I_{25} & I_{19} & I_7 & I_{14} & I_{28} \end{bmatrix} (93x155) \tag{2.12}$$

- $[21, 8, 6]$ QC LDPC with GF(7). With a = 2, b = 6, o(a) = 3 and o(b) = 2, the parity H is

$$H = \begin{bmatrix} I_1 & I_2 & I_4 \\ I_6 & I_5 & I_3 \end{bmatrix} (14x21) \tag{2.13}$$

- $[104, 20]$ QC LDPC with non prime m = 26. With a = 5, b = 9, o(a) = 4 and o(b) = 3, the parity H is

$$H = \begin{bmatrix} I_1 & I_5 & I_{25} & I_{21} \\ I_9 & I_{19} & I_{17} & I_7 \\ I_3 & I_{15} & I_{23} & I_{11} \end{bmatrix} (78x104) \tag{2.14}$$

- $[5219, 4300]$ QC LDPC with GF(307). With a = 9, b = 17, o(a) = 17 and o(b) = 3, the parity H is given below separately of size (921x5219)
  H =

$$\begin{bmatrix} I_1 & I_9 & I_{81} & I_{155} & I_{114} & I_{105} & I_{24} & I_{216} & I_{102} & I_{304} & I_{280} & I_{64} & I_{269} & I_{272} & I_{299} & I_{235} & I_{273} \\ I_{17} & I_{153} & I_{149} & I_{113} & I_{96} & I_{250} & I_{101} & I_{295} & I_{199} & I_{256} & I_{155} & I_{167} & I_{275} & I_{19} & I_{171} & I_4 & I_{36} \\ I_{289} & I_{145} & I_{77} & I_{79} & I_{97} & I_{259} & I_{182} & I_{103} & I_6 & I_{54} & I_{179} & I_{76} & I_{70} & I_{16} & I_{144} & I_{68} & I_{305} \end{bmatrix} \tag{2.15}$$

For instance, lets try constructing the Tanner code which is a [155, 64, 20] QC code (Galois Field m=31). Initial Elements a=2, b=5 are chosen from GF(31); then o(a) = 5, o(b) = 3 and the Mother matrix P is -

$$H = \begin{bmatrix} I_1 & I_2 & I_4 & I_8 & I_{16} \\ I_5 & I_{10} & I_{20} & I_9 & I_{18} \\ I_{25} & I_{19} & I_7 & I_{14} & I_{28} \end{bmatrix} (93x155) \tag{2.16}$$

The individual circulant permutation matrices above were derived from the methods mentioned above. A breakdown of some of those elements are given below.
Element H(1, 1) $\Rightarrow$ 1
$H(1, 2) \Rightarrow a = 2$
$H(2, 1) \Rightarrow b = 5$
$H(1, 3)...H(1, 5) \Rightarrow$I$_{a^{2+1}}$
H(3, 1) $\Rightarrow$b^2 = 25
H(2, 2) $\Rightarrow$I$_{ab}$ = 5 x 2 = 10

H(2, 3) $\Rightarrow I_{a^2b} = 4 \text{ x } 5 = 20$

H(2, 4) $\Rightarrow I_{a^3b} = 8 \text{ x } 5 = 40 > 31$. So H(2, 4) = 40 % 31 = 9 (remainder)

H(2, 5) $\Rightarrow I_{a^4b} = 16 \text{ x } 5 = 80 > 31$. So H(2, 5) = 80 % 31 = 19

Similarly, the base matrix H is created.

Now populating the base matrix and expanding it, would result in a *jm x km* sized Parity check matrix, which is 93x155 in dimensions.

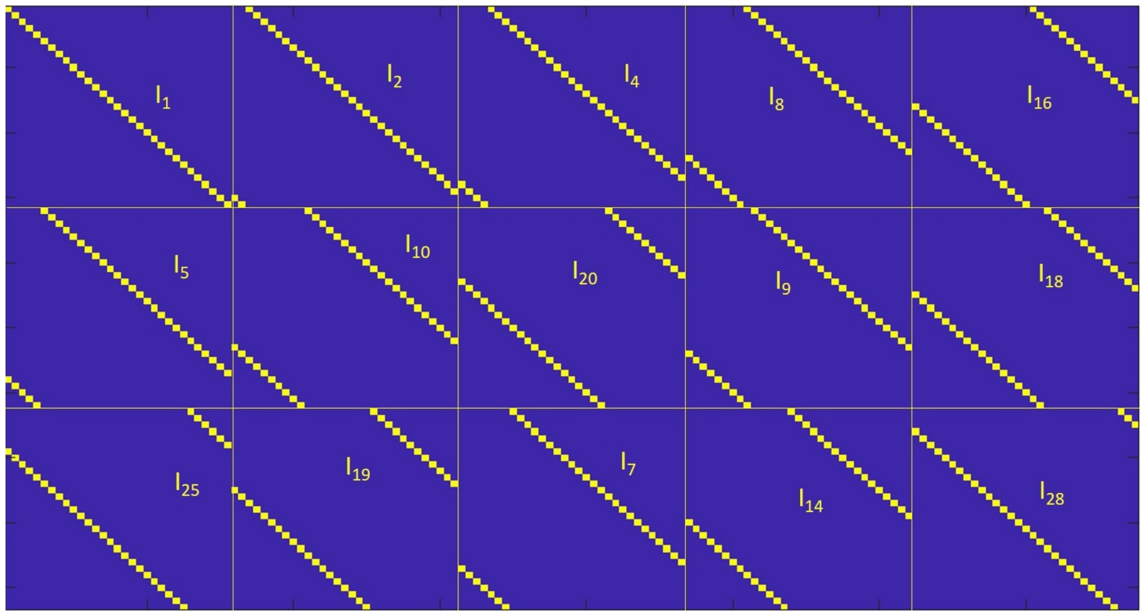

Figure 2.1: [155, 64, 20] QC Tanner code

The above figure was obtained from coding the Quasi Cyclic LDPC parity matrix creation protocols in MATLAB and using the function imagesc(”x”) which gives a sized color-print graph of the matrix. The size is large to perceive directly, so here each yellow square is a binary ’1’ while the rest of the blue are 0’s. The code was generalised to create a Quasi Cyclic regular LDPC code for any matrix settings ranging for any j, k dimension.

2.5 Pre - Processing

Though LDPC codes are superior to most of its counterparts in many ways, one major hurdle in employing LDPC codes is its complications with encoding. It had a high encoding complexity, which made it incredibly tough to even implement with any available technology during Gallager's time period, that it was forgotten and re-discovered by Mackay and Neal in 1996, 27 years later. For instance, Turbo Codes, the closest counterpart of LDPC was linearly encodable with time, but LDPC has complexity quadratic in the block length, which made a straightforward linear encoder unfathomable. In 2001 February, TJ Richardson and RL Urbanke presented their novel idea of encoding which would go on to become the de-facto for LDPC encoding named, 'Richard Urbanke Encoding'. This process involved the rearrangement of the parity check matrix into an Approximate Lower Triangular (ALT)[5] format which then enabled easier implementation of Linear Encoding.

Based on an array of encoding techniques, after research, this thesis will be utilizing Modified Approximate Lower Triangular Rearrangement coupled with linear encoding.

2.5.1 Modified Approximate Lower Triangular (MALT) Format

Modified Approximate Lower Triangular (MALT)[8] format is the result of cascading modifications done collectively by the field of Science over time with Richard Urbanke's Approximate Lower Triangular (ALT) form being the origin. In that method, using row - column permutations the non singular Parity check Matrix is brought to an ALT form -

$$ALT(H) = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \tag{2.17}$$

such that $\phi = -ET^{-1}B + D$ is non singular of dimensions m by n and the resulting matrix is still sparse, meaning the rearrangement hasn't lost data.

Here A, B, C and D are sparse matrices pertaining to data while T is the lower triangular matrix and E is eliminated using Gaussian Eliminations.
Qi and Noertz from the University of Edinburgh proposed an addition to that where they created an algorithm generalised enough to form the lower triangular T for any matrix of any size. Aside from that their, 'Systemic Approximate lower Triangular' (SALT)[7] format had added steps where Gaussian Elimination was added to C and D where D was converted to an identity matrix. After removing all the linearly dependent rows at the bottom, the SALT form was arrived at which promises better

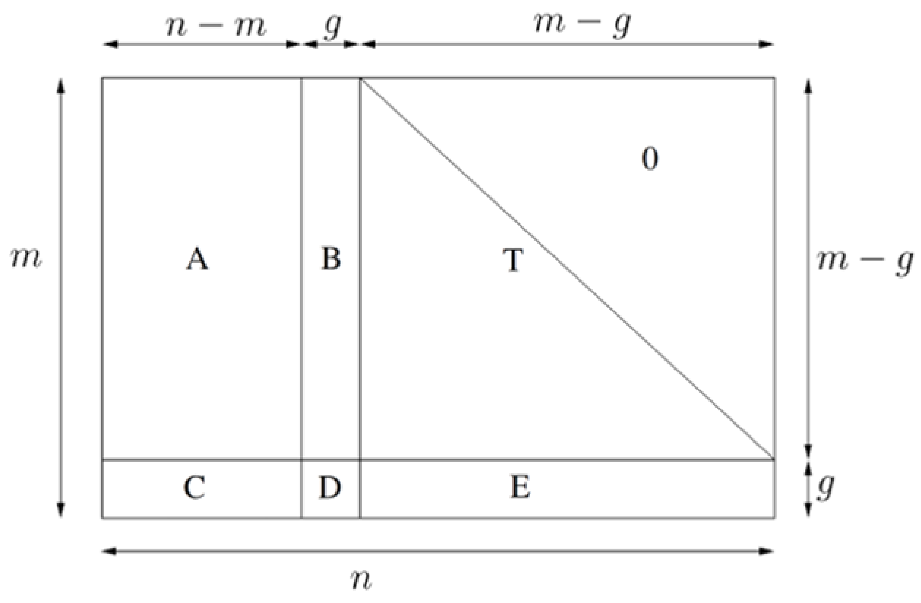

Figure 2.2: Approximate Lower Triangular Format

Bit Error rate compared to the RU method.

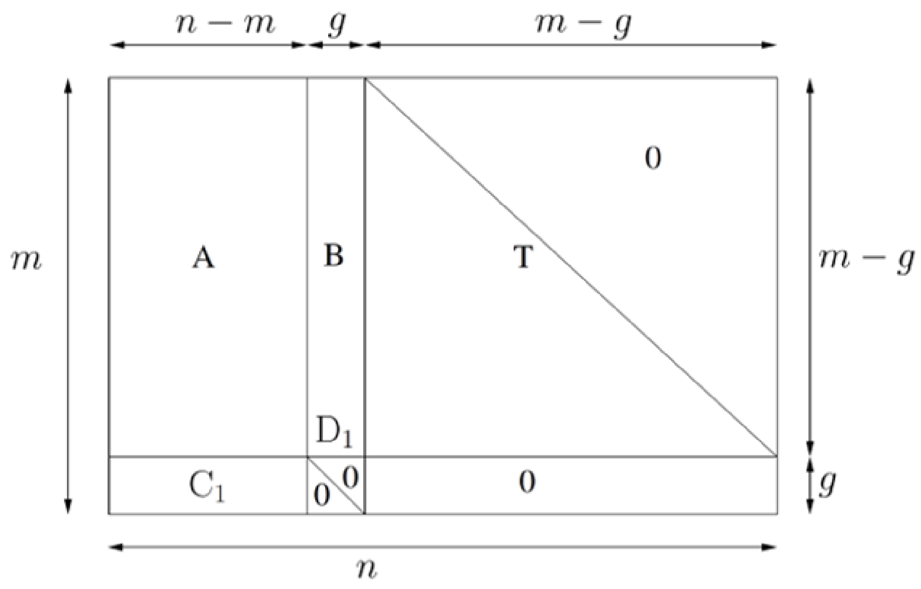

Figure 2.3: Systemic Approximate Lower Triangular Format

As for most matrices, SALT method was sufficient, for some rare cases, the ϕ turned out to be singular meaning its inverse did not exist thereby it couldn't be linearly encoded. Dutta and Pramanik from the Indian Institute of Engineering Science and Technology, Shibpur, India proposed Modified Approximate Lower Triangular form. This was an addition to the SALT format, where Gaussian Elimination was applied to C and D matrices again to obtain an Upper Triangular D matrix. This method promised better BER rates than its predecessors.

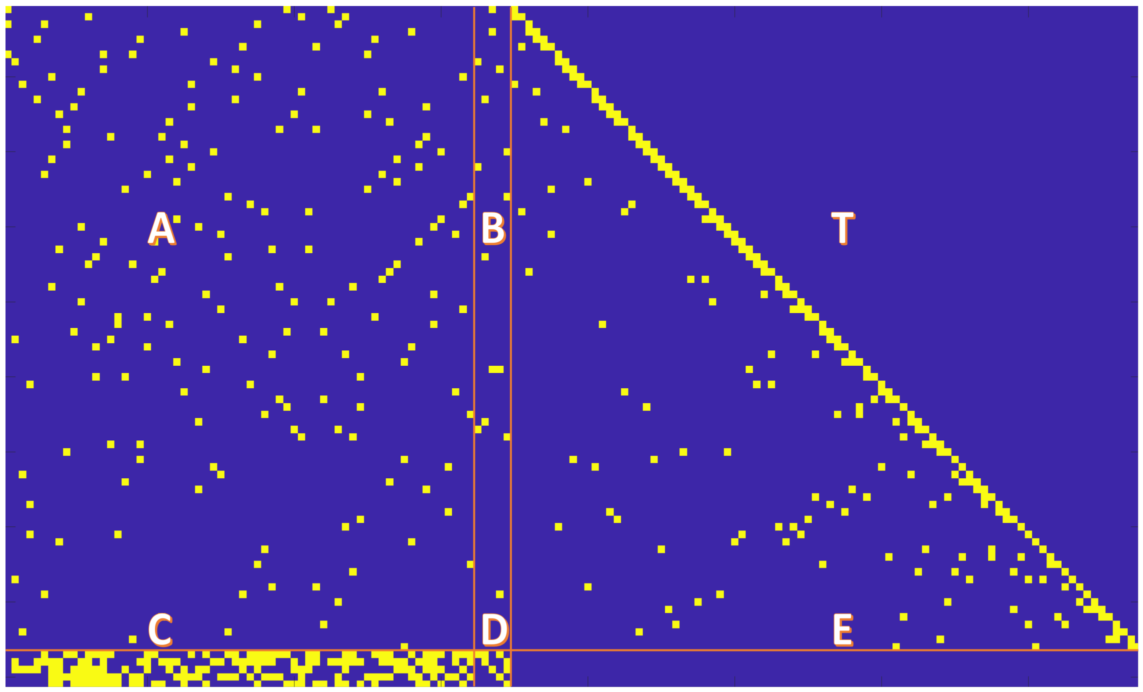

Figure 2.4: Modified Approximate Lower Triangular Format for [155, 64, 20] QC Tanner Code simulated on MATLAB

An algorithm to achieve the MALT format for any sized Quasi Cyclic LDPC Parity Check matrix was developed on MATLAB. Through multiple separate functions performing each task increment wise, performing row - column permutations continuously, this is called the pre-processing and the flowchart of the algorithm is below. This algorithm projects a way to systemically bring the Lower Triangular Part.

The Quasi Cyclic LDPC Parity Check Matrix that was generated for the [155, 64, 20] Tanner code with m = 31, is a 93x155 matrix. As it is a relatively large matrix, directly viewing that seems impossible. Hence here the imagesc(K) function of MATLab is used to create a binary image based on the data of the parity check matrix.The Quasi Cyclic LDPC Parity Check Matrix that was generated for the [155, 64, 20] Tanner code with m = 31, is a 93x155 matrix. As it is a relatively large matrix, directly viewing that seems impossible. Hence here the imagesc(K) function

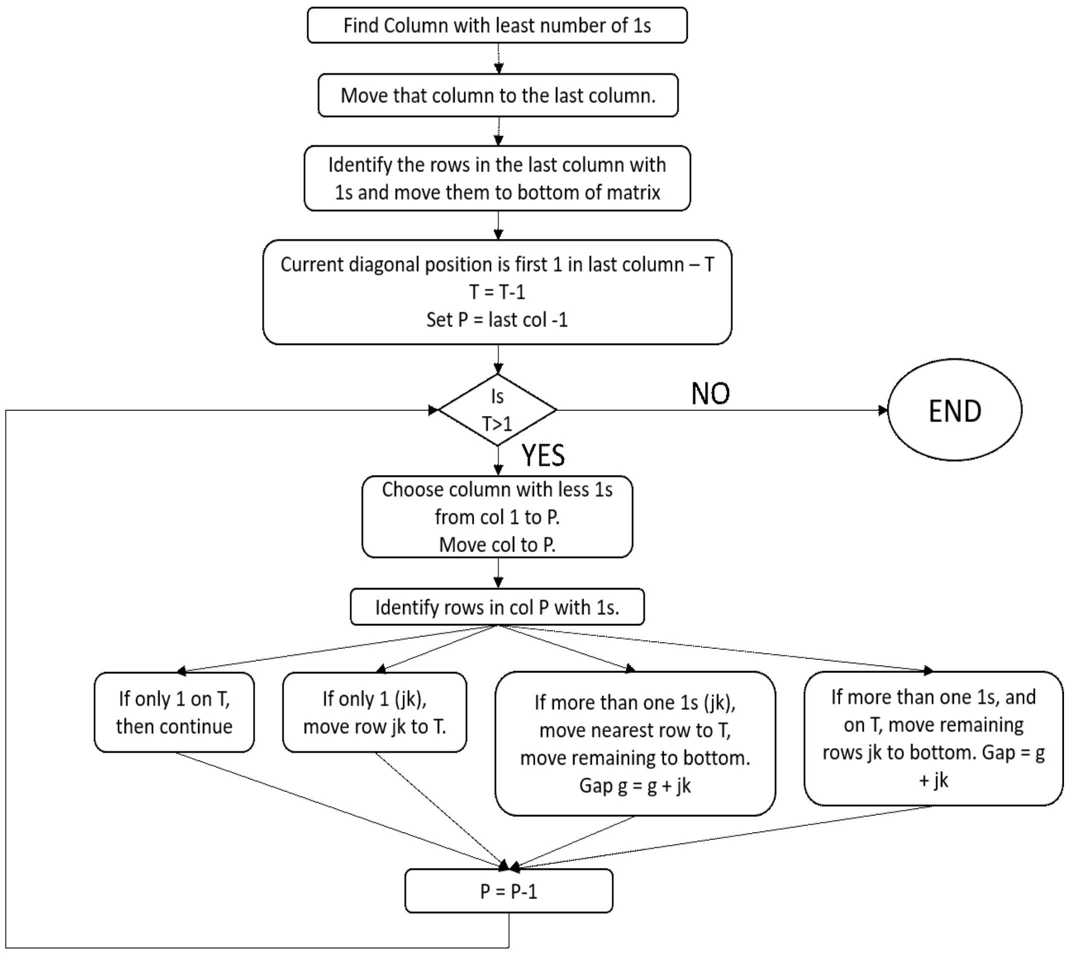
Find Column with least number of 1s
Move that column to the last column.
Identify the rows in the last column with 1s and move them to bottom of matrix
Current diagonal position is first 1 in last column – T
T = T-1
Set P = last col -1
Is T>1
NO
END
YES
Choose column with less 1s from col 1 to P. Move col to P.
Identify rows in col P with 1s.
If only 1 on T, then continue
If only 1 (jk), move row jk to T.
If more than one 1s (jk), move nearest row to T, move remaining to bottom. Gap g = g + jk
If more than one 1s, and on T, move remaining rows jk to bottom. Gap = g + jk
P = P-1

Figure 2.5: Flowchart detailing the method on arriving at the ALT Form[5]

of MATLab is used to create a binary image based on the data of the parity check matrix.

Using the algorithm specified in the flowchart above, an automatic generalized code was developed on MATlab which takes the QC LDPC PCM as input and returns the output. The code was designed such that the rearranged matrix has a smaller gap g.

To begin with the process of rearranging the parity matrix, a scan of the columns $1.../, n$ with the least number of 1's are found. Considering regular LDPC[19] codes, a random initial column can be chosen owing to the reason that the 1's are evenly distributed. That column is cascading at the far right n. Now all the rows in the current column which has entries of 1's are cascaded to the bottom of the matrix. Now the scan is repeated for the columns of $1.../, n-1$ and the same steps are repeated

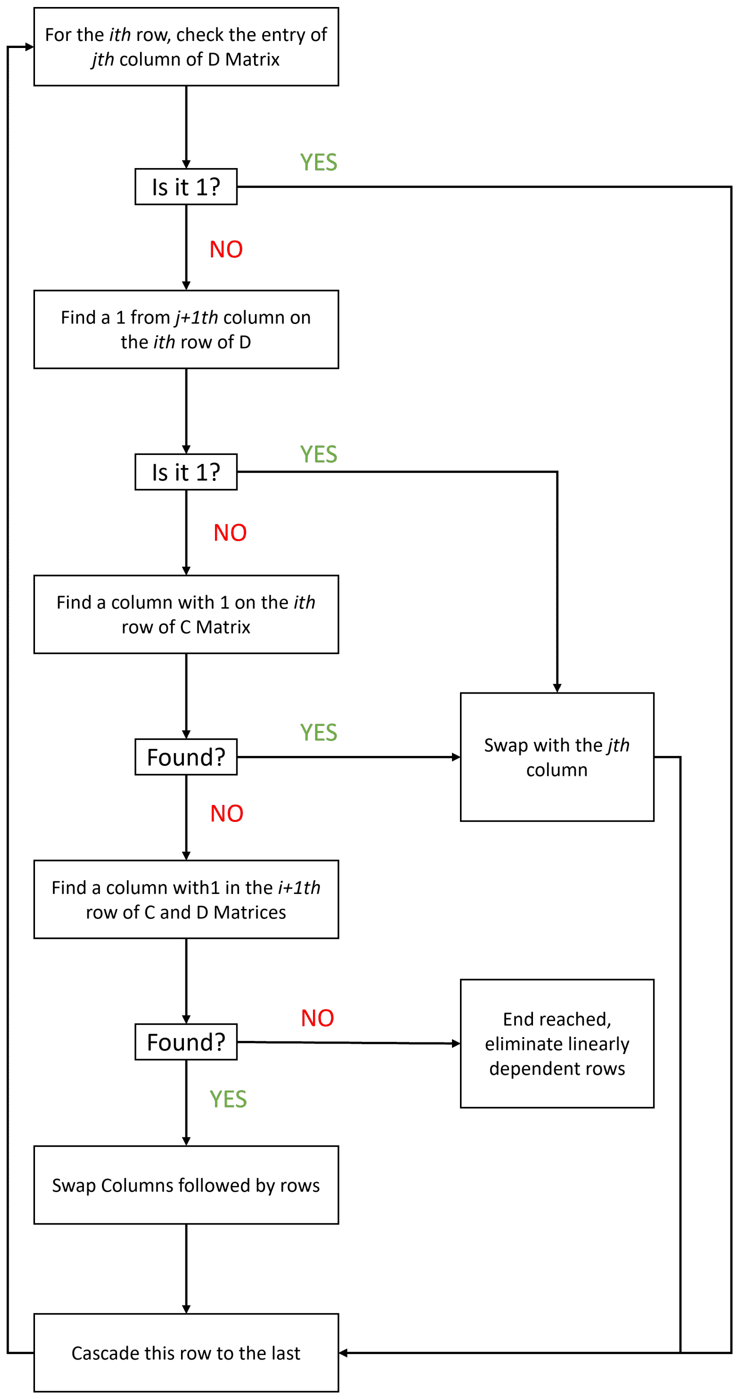
For the ith row, check the entry of jth column of D Matrix
Is it 1?
YES
NO
Find a 1 from j+1th column on the ith row of D
Is it 1?
YES
NO
Find a column with 1 on the ith row of C Matrix
Found?
YES
Swap with the jth column
NO
Find a column with1 in the i+1th row of C and D Matrices
Found?
NO
End reached, eliminate linearly dependent rows
YES
Swap Columns followed by rows
Cascade this row to the last

Figure 2.6: Modified Approximate Lower Triangular Format Algorithm for Lower Triangular D Matrix[8]

until all possible columns are achieved. The rows with 1's in each current column are placed in a diagonal position T. If the column has only one 1, in the current column above the diagonal position, T, then that row is cascaded to that position and moved

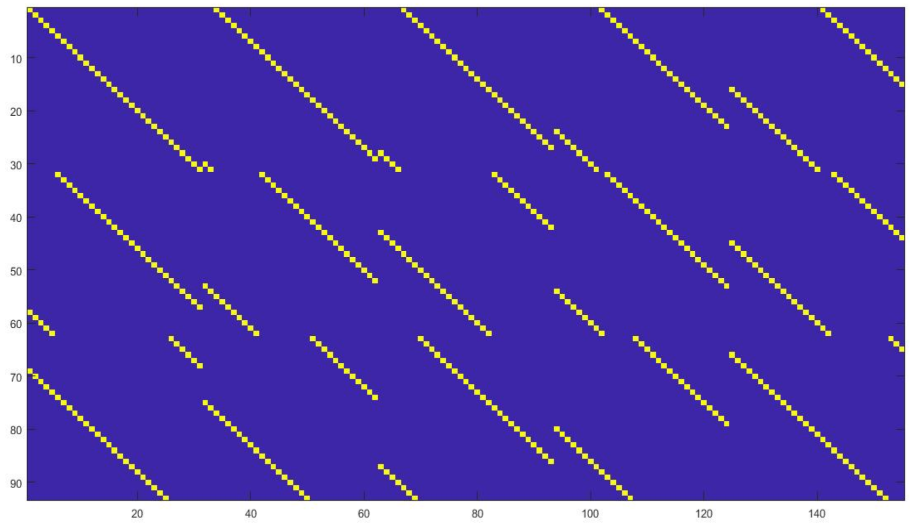

Figure 2.7: [155, 64, 20] Tanner code

on. If there are more entries of rows with 1's, the nearest is cascaded to position T and the rest are cascaded to the bottom of the matrix increasing the gap size g. Now the scan is repeated for the columns of $1.../, n-1$ and the same steps are repeated until all possible columns are achieved. The resultant will be a matrix with a Lower Triangular arrangement similar to **Figure 2.8**.

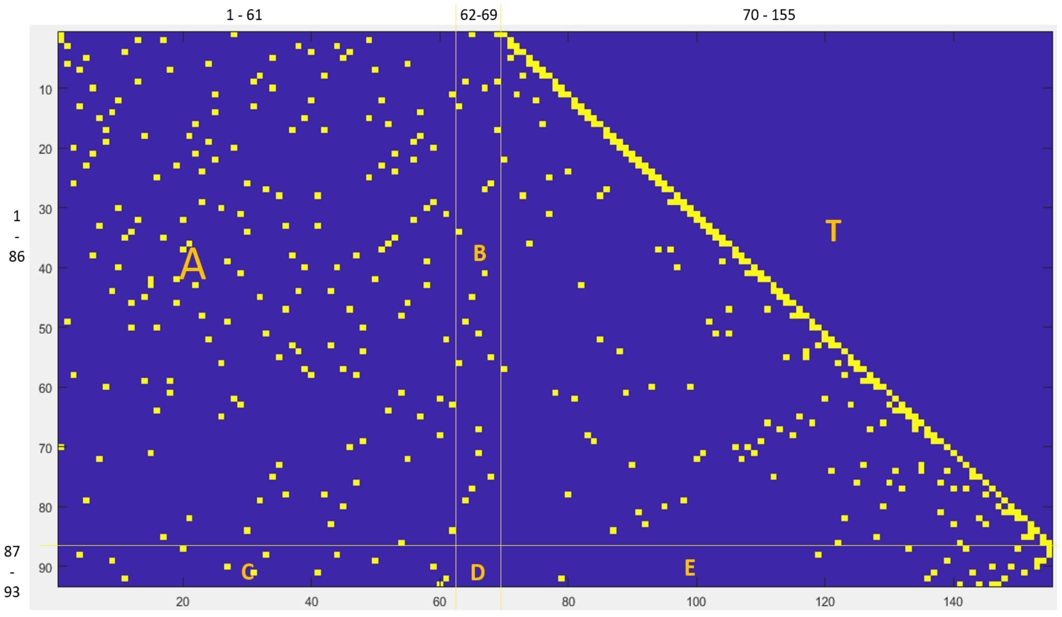

Figure 2.8: ALT form after implementing algorithm

After this process the E matrix is converted to an all zero matrix using Gaussian Jordan Elimination using T matrix. Then the D matrix is converted to an Upper Triangular Matrix without changing the data on matrices A, B and T. In this process C matrix loses its sparsity and some all zero rows occur which are to be eliminated.

To change the D matrix into an Upper Triangular Matrix, the first row and column of the D matrix is checked for 1. If the flag fails, then any column of C and D with 1 in it's first row is cascaded. Else, the remaining 1's below the topmost is XOR'ed and eliminated. Once the process is complete, some all zero rows will be present at the bottom which must be eliminated. Post which the MALT form is achieved similar to **Figure 2.9**.

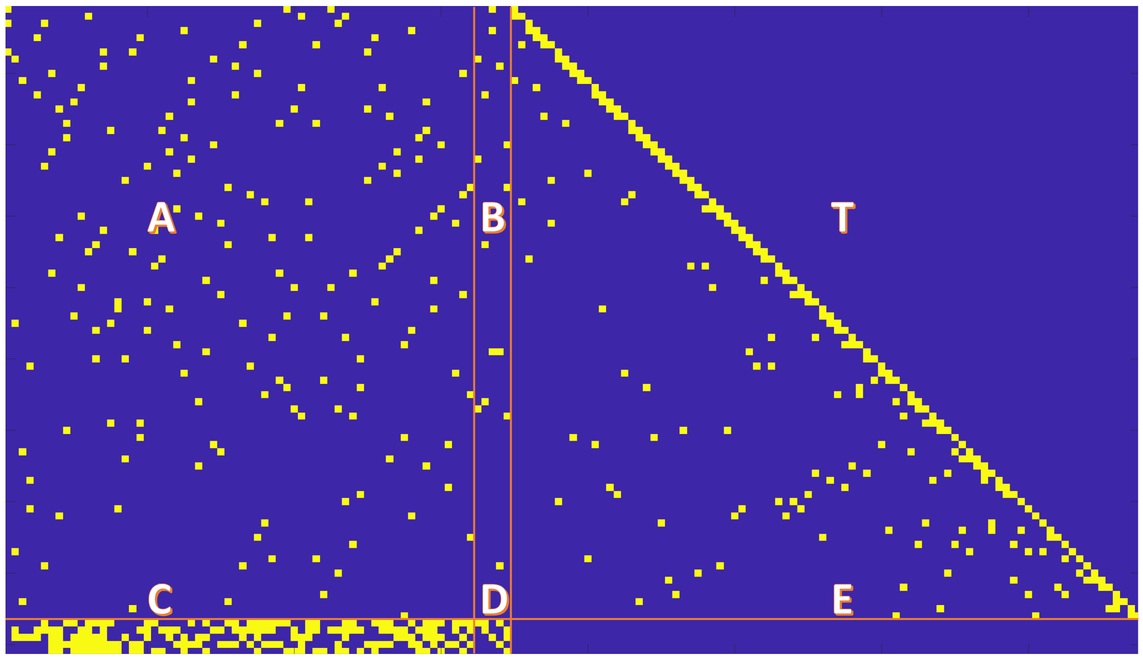

Figure 2.9: Modified Approximate Lower Triangular Format for [155, 64, 20] QC Tanner Code simulated on MATLAB

This format of the Parity Check matrix allows us to apply Linear Encoding technique to this.

2.5.2 Simulated Results

Some of the examples listed above were created and pre-processed to rearrange into MALT format, ready for encoding using a MATLAB script which automates all methods mentioned above. As mentioned in **Section 2.4**, a function will accept 5 inputs in the order of $(o(a), o(b), a, b, m)$ where the first two inputs are the row-column of the base matrix and the next two are the row-column circulant shift for the identity matrices of the parity matrix, finally m, is the size of the identity matrices, which can accept both prime and non prime numbers to create and populate a LDPC Quasi Cyclic Parity Check matrix.

Once the parity matrix is created, the pre-processor function will receive that as input and will being the process. Initially, the matrix is rearranged to the Lower Triangular Matrix format, followed by a function which will Gaussian eliminate the E matrix, finally the last function will use the C and D matrix of the H matrix to iterate D into Upper Triangular matrix which yields the MALT format parity matrix ready for encoding. The examples listed in the **2.12** to **2.15** series were simulated for depiction.

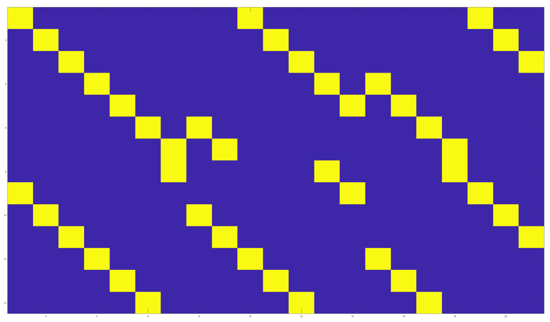

Figure 2.10: $[21, 8, 6]$ QC LDPC with GF(7). With a = 2, b = 6, o(a) = 3 and o(b) = 2.

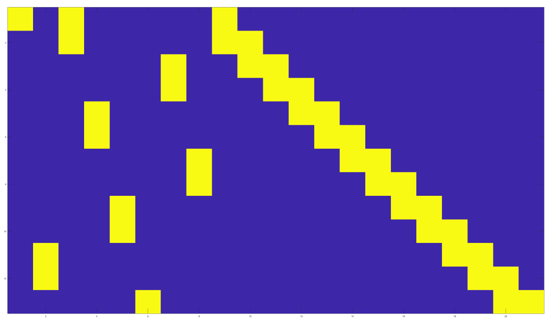

Figure 2.11: $[21, 8, 6]$ QC LDPC with GF(7). After Pre Processing. Note that this resembles a typical parity-generator pair.

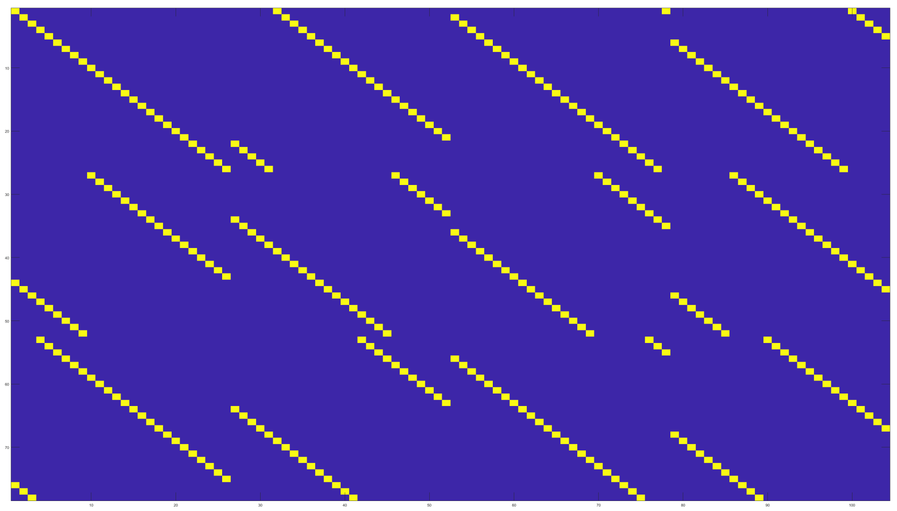

Figure 2.12: $[104, 20]$ QC LDPC with non prime m = 26. With a = 5, b = 9, o(a) = 4 and o(b) = 3.

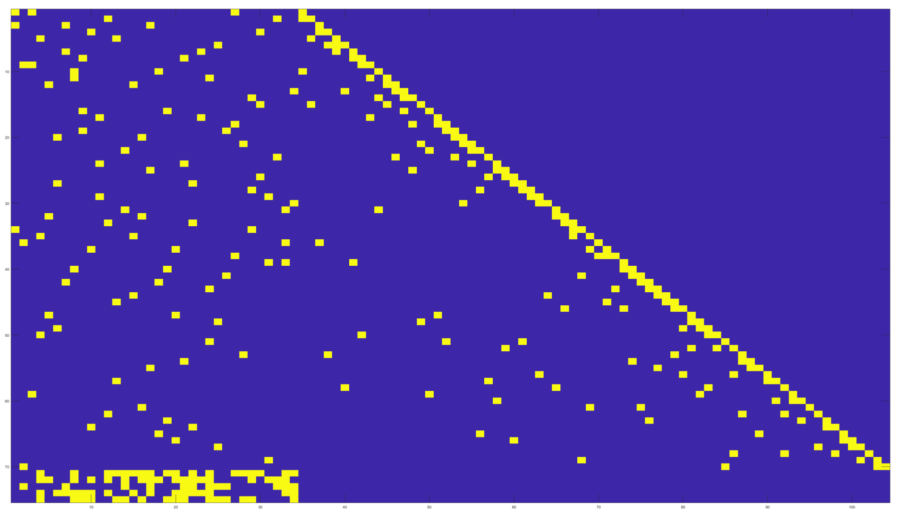

Figure 2.13: $[104, 20]$ QC LDPC with non prime m = 26. After Pre-Processing

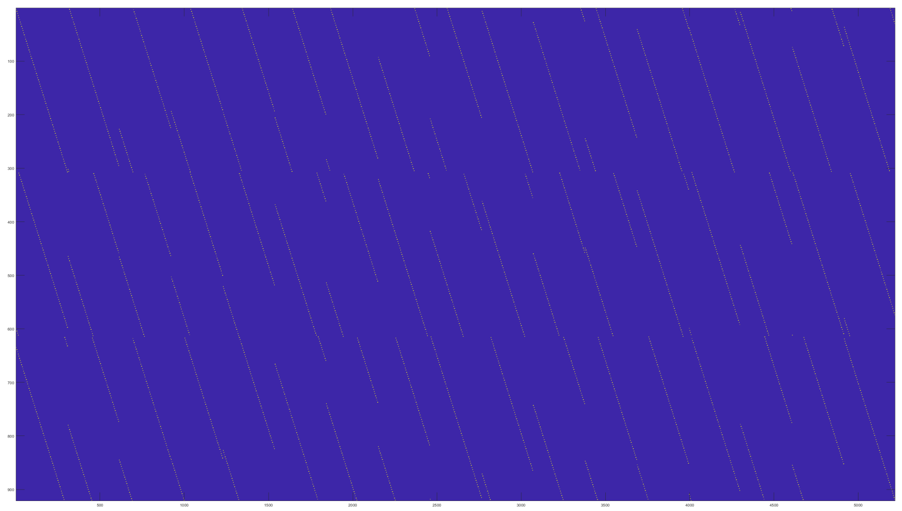

Figure 2.14: [5219, 4300] QC LDPC with GF(307). With a = 9, b = 17, o(a) = 17 and o(b) = 3

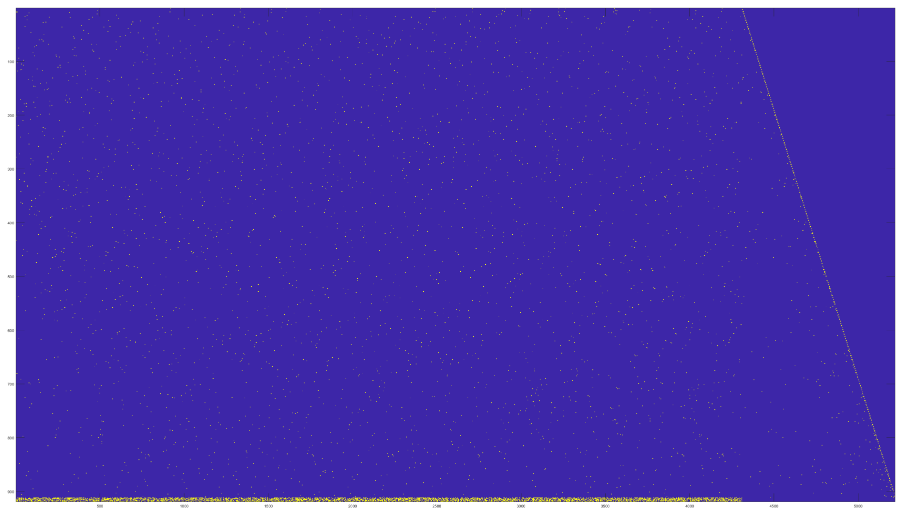

Figure 2.15: [5219, 4300] QC LDPC with GF(307). After Pre-Processing

2.6 Linear Encoding

After matrix manipulation techniques now have the MALT format Parity check matrix. Now apply Linear Encoding. Consider a message of u bits. The resultant code-word c will be turned into 3 parts where $c = [up_1p_2]$ where p_1 and p_2 are parity bits 1 and 2.

The code word c = [u, p1, p2] must satisfy the parity-check equation $cH^T = 0$ and so

$$Au + Bp_1 + Tp_2 = 0 \tag{2.18}$$

$$Cu + Dp_1 + 0p_2 = 0 \tag{2.19}$$

As E is zero, the parity bits in p1 depend only on the message bits, and so can be calculated independently of the parity bits in p2. If D is invertible, p1 can be found from

$$p_1 = D^{-1}Cu. \tag{2.20}$$

If D is not invertible the columns of H can be permuted until it is. By keeping g as small as possible the added complexity burden of the matrix multiplication which is O(g2), is kept low. Once p1 is known p2 can be found from

$$p_2 = -T^{-1}(Au + Bp_1) \tag{2.21}$$

So finding p_2 for all parity bits would be,

$$p_2(l) = \sum_{j=1}^{n-m} A_{l,j}.u_j + \sum_{j=1}^{g} B_{l,j}.p_1(j) + \sum_{j=1}^{l-1} T_{l,j}.p_2(j) \tag{2.22}$$

Form the code vector as

$$C = [u, p1, p2] \tag{2.23}$$

p1 holds the first g parity and p2 contains the remaining parity bits.

2.7 Decoding

The message bits are encoded with parity bits and is transmitted through an Additive White Gaussian Noise (AWGN) Channel after Binary Phase Shift Keying (BPSK)

Modulation. So all the binary bits (0, 1) are now (-1, 1). The receiver will send the received code through decoding to get the corrected data transmitted.
The class of decoders for LDPC[20] Codes are called Message Passing Decoders as the message is passed along the edges of a Tanner Graph. Their are also called iterative decoders as the message volleys back and forth between the bit nodes and check nodes until result is achieved, iteratively. The methods used to decode here are known as Belief Propagation Decoding where the message bits are values of probabilities. They are represented by log-likelihood ratios which are used in Sum Product Algorithm (SPA) and Minimum Sum algorithm (MSA).

2.7.1 Sum Product Algorithm (SPA)

Sum Product Decoding Algorithm (SPA) is a soft - decision message passing algorithm. Each of its message bits are in the values of probabilities which are likelihood values. They accept only probability values as input and hence called soft - decision. As message bits keeps volleying between nodes, the input bits of probability values are called *priori*, meaning that they are known well ahead. The probability values returned back by the decodes is called *posteriori*. All Probability values are in log - likelihood ratio. For a binary x, if p(x=0) is already given, then p(x=1) is just 1-p(x=0). So by storing only either, store the message bit in likelihood ratio as shown.

$$L(x) = log \frac{\left[p(x=0)\right]}{\left[p(x=1)\right]} \tag{2.24}$$

The extrinsic information being passed between the nodes are the probabilities. The check node j and bit node i between which extrinsic information is passed through is denoted by $E_{j,i}$. This gives the probability that the bit c_i will be 1 that satisfies parity j. The probability that an odd number of bits are 1 is given by -

$$P_{j,i}^{ext} = \frac{1}{2} - \frac{1}{2} \prod_{i' \in B_j, i' \neq i} (1 - 2P_{j,i'}) \tag{2.25}$$

And the probability for 0 will be $1 - P_{j,i}^{ext}$. Coming back to eqn (2.18), the sign of *L(x)* gives a hard decision on x and the magnitude will give the reliability. Changing to probabilities,

$$P(x=1) = \frac{e^{-L(x)}}{1+e^{-L(x)}} \tag{2.26}$$

$$P(x=0) = \frac{e^{L(x)}}{1+e^{-L(x)}} \tag{2.27}$$

The extrinsic information can be represented as a Log Likelihood Ratio

$$E_{j,i} = L(P^{ext}_{j,i}) = log\frac{1-P^{ext}_{j,i}}{P^{ext}_{j,i}} \tag{2.28}$$

Now,

$$E_{j,i} = log\frac{1/2 + 1/2\prod_{i\in B_j, i'\neq i}(1-2P_{j,i'})}{1/2 - 1/2\prod_{i\in B_j, i'\neq i}(1-2P_{j,i'})} \tag{2.29}$$

$$E_{j,i} = log\frac{1 + \prod_{i\in B_j, i'\neq i}(1-2\frac{e^{-M_{j,i'}}}{1+e^{-M_{j,i'}}})}{1 - \prod_{i\in B_j, i'\neq i}(1-2\frac{e^{-M_{j,i'}}}{1+e^{-M_{j,i'}}})} \tag{2.30}$$

$$E_{j,i} = log\frac{1 + \prod_{i\in B_j, i'\neq i}(\frac{1-e^{-M_{j,i'}}}{1+e^{-M_{j,i'}}})}{1 - \prod_{i\in B_j, i'\neq i}(\frac{1-e^{-M_{j,i'}}}{1+e^{-M_{j,i'}}})} \tag{2.31}$$

where $M_{j,i'} =^{\triangle} L(P_{j,i'}) = log\frac{1-P_{j,i'}}{P_{j,i'}}$
Using the equation -

$$tanh\frac{1}{2}log\frac{1-p}{p} = 1-2p \tag{2.32}$$

gives us,

$$E_{j,i} = log\frac{1 + \prod_{i\in B_j, i'\neq i} tanh\frac{M_{j,i'}}{2}}{1 - \prod_{i\in B_j, i'\neq i} tanh\frac{M_{j,i'}}{2}} \tag{2.33}$$

Alternatively,

$$2tanh^{-1}p = log\frac{1+p}{1-p} \tag{2.34}$$

Then,

$$E_{j,i} = 2tanh^{-1}\prod_{i\in B_j, i'\neq i} tanh(M_{j,i'}/2) \tag{2.35}$$

The presence of tanh and $tanh^{-1}$ makes the calculations complicated, so following Gallager use the sign and magnitude of M_{ji}.

$$M_{ji} = \alpha_{ji}\beta_{ji} \tag{2.36}$$

$$\alpha_{ji} = sign(M_{ji}) \tag{2.37}$$

$$\beta_{ji} = |M_{ji}| \tag{2.38}$$

So now rewrite as -

$$tanh\frac{M_{ji}}{2} = \prod_{i'} \alpha_{ji'}.\prod_{i\in B_j, i'\neq i} tanh\frac{\beta_{ji'}}{2} \tag{2.39}$$

Following which,

$$E_{j,i} = \prod_{i'} \alpha_{ji'}.2tanh^{-1}\Big(\prod_{i'} tanh\frac{\beta_{ji'}}{2}\Big) \tag{2.40}$$

$$E_{j,i} = \prod_{i'} \alpha_{ji'}.2tanh^{-1}log^{-1}log\Big(\prod_{i'} tanh\frac{\beta_{ji'}}{2}\Big) \tag{2.41}$$

$$E_{j,i} = \prod_{i'} \alpha_{ji'}.2tanh^{-1}log^{-1}\sum_{i'} log\Big(tanh\frac{\beta_{ji'}}{2}\Big) \tag{2.42}$$

Which can be rewritten as -

$$E_{j,i} = \prod_{i'} \alpha_{ji'}.\phi\Big(\sum_{i'} \phi(\beta_{ji'})\Big) \tag{2.43}$$

where $\phi(x)$ is,

$$\phi(x) = -log[tanh(x/2)] = log\frac{e^x+1}{e^x-1} \tag{2.44}$$

$\phi(x) = \phi^{-1}(x)$ when x>0, Each bit node has input probability L_i, and to the probabilities from every check node. The total probabilities for i^{th} bit is its sum,

$$L_i^{Total} = L_i + \sum_{j\in A_i} E_{ji} \tag{2.45}$$

The hard decision is given by the signs of the L_i^{Total}. After checking to see if the parity equation is satisfied, update M_{ji} accordingly,

$$M_{ji} = \sum_{j \in A_i, j' \neq j} E_{j'i} + L_i \tag{2.46}$$

The output received will be the a posteriori bit probabilities of the received bits as Log likelihood ratios. The number of iterations of the decoder is determined by the valid code-word found which satisfies the parity equation or the total iterations specified. The decoder is initialised by setting all variable node messages M_{ji} equal to,

$$L_i = L(c_i|y_i) = log \frac{Pr(c_i = 0|y_i)}{Pr(c_i = 1|y_i)} \tag{2.47}$$

Here y_i is the channel value received and the L_i for different channels can be calculated. This thesis used Additive White Gaussian Noise Channel, where the ith received sample is $y_i = x_i + n_i$ where the n_i are independent and normally distributed as $\eta(0, \sigma^2).\sigma^2 = N_0/2$ where N_0 is the noise density.

$$Pr(x_i = x|y_i) = \frac{1}{1 + exp(-4y_i x/N_0)} \tag{2.48}$$

where $x \in \pm 1$

$$L(c_i|y_i) = \frac{4y_i}{N_0} \tag{2.49}$$

Now these are the steps for iterative SPA decoding

Step 1: Initialization: for all i, initialize L_i for the appropriate channel model. Then, for all i, j for which $h_{i,j} = 1$ set $M_{ji} = L_i$; and $l = 0$. Define B_j to represent the set of bits in the $j^t h$ parity check equation of H and A_i to represent the parity check equations for the $i^t h$ bit of the code.

Step 2: CN update: compute outgoing CN message E_{ji} for each CN

$$M_{ji} = \alpha_{ji}\beta_{ji} \tag{2.50}$$

$$\alpha_{ji} = Sign(M_{ji}) \tag{2.51}$$

$$\beta_{ji} = |M_{ji}| \tag{2.52}$$

$$E_{ji} = \prod_{i'} \alpha_{ji'}.\phi\left(\sum_{i'} \phi(\beta_{ji'})\right) \tag{2.53}$$

$$\phi(x) = -log[tanh(x/2)] \tag{2.54}$$

$$= log\left[\frac{e^x + 1}{e^x - 1}\right] \tag{2.55}$$

Step 3: LLR total: For i = 0; 1; . . .; N - 1 compute total LLR

$$L_i^{Total} = L_i + \sum_{j \in A_i} E_{ji} \tag{2.56}$$

Step 4: Stopping criteria: For i = 0; 1; . . .; N - 1, set

$$\hat{c_i} = \begin{cases} 1 & \text{if } L_i^{Total} < 0, \\ 0 & \text{on } else \end{cases} \tag{2.57}$$

To obtain c. If $cH^T = 0$ or the number of iterations equals the maximum Limit (l = l_max,) stop; else

Step 5: VN update: compute outgoing VN message M_ji for each VN

$$M_{ji} = L_i + \sum_{j' \in A_i, j' \neq j} E_{j'i}.1 = 1 + 1; \tag{2.58}$$

go to step 2

This is the step by step pseudo code used in **Algorithm 1**.

Algorithm 1 Algorithm for Sum Product Decoding[35]

```
1:  procedure DECODE (R)(                                        ▷ )Initialisation
2:      $I = 0$
3:      for $i = 1 : n$ do
4:          for $j = 1 : m$ do
5:              $M_{j,i} = r_i$
6:          end for
7:      end for
8:
9:      repeat
10:         for $j = 1 : m$ do                                   ▷ Step 1: Check Messages
11:             for $i \in B_j$ do
12:                 $E_{j,i} = log\left(\frac{1+\prod_{i' \in B_j, i' \neq i} tanh(M_{j,i'}/2)}{1-\prod_{i' \in B_j, i' \neq i} tanh(M_{j,i'}/2)}\right)$
13:             end for
14:         end for
15:         for $i = 1 : n$ do                                   ▷ Test
16:             $L_i = \sum_{j \in A_i} E_{j,i} + r_i$
17:             $z_i = \begin{cases} 1 & \text{if } L_i \leq 0, \\ 0 & \text{on } L_i > 0 \end{cases}$
18:         end for
19:
20:         if $I = I_{max}$ or $Hz^T = 0$ then
21:             Finished
22:         else
23:             for $i = 1 : n$ do                               ▷ Step 2 : Bit Messages
24:                 for $j \in A_i$ do
25:                     $M_{j,i} = \sum_{j' \in A_i, j' \neq j} E_{j',i} + r_i$
26:                 end for
27:             end for
28:             $I = I + 1$
29:         end if
30:     until Finished
31: end procedure
```

2.7.2 Minimum Sum Algorithm (MSA)

The Min Sum algorithm is not different compared to sum product algorithm. Consider the following relation

$$\phi\left(\sum_{i'}\phi(\beta_{ji'})\right) \cong \phi(\phi(min_{i'}\beta_{ji'})) = min_{i'}\beta_{ji'} \tag{2.59}$$

The min-sum algorithm is simply the log domain SPA with Step 2 replaced by

$$M_{ji} = \alpha_{ji}\beta_{ji} \tag{2.60}$$

$$\alpha_{ji} = Sign(M_{ji}) \tag{2.61}$$

$$\beta_{ji} = |M_{ji}| \tag{2.62}$$

$$E_{ji} = \prod_{i'}\alpha_{ji'}.\min_{i'}\beta_{ji'} \tag{2.63}$$

It can also be shown that, in the AWGN case, the initialization $M_{ji} = 4Y_i/N0$ may be replaced by $M_{ji} = Y_i$ when the simplified log domain sum–product algorithm is employed. The advantage, of course, is that an estimate of the noise power N0 is unnecessary in this case.

This is how the basic structure of the MATLAB Simulation is[22]. Here the SPA and MSA decodes each time for SNR values from 0 to 5 dB for upto length of total message bit each time.

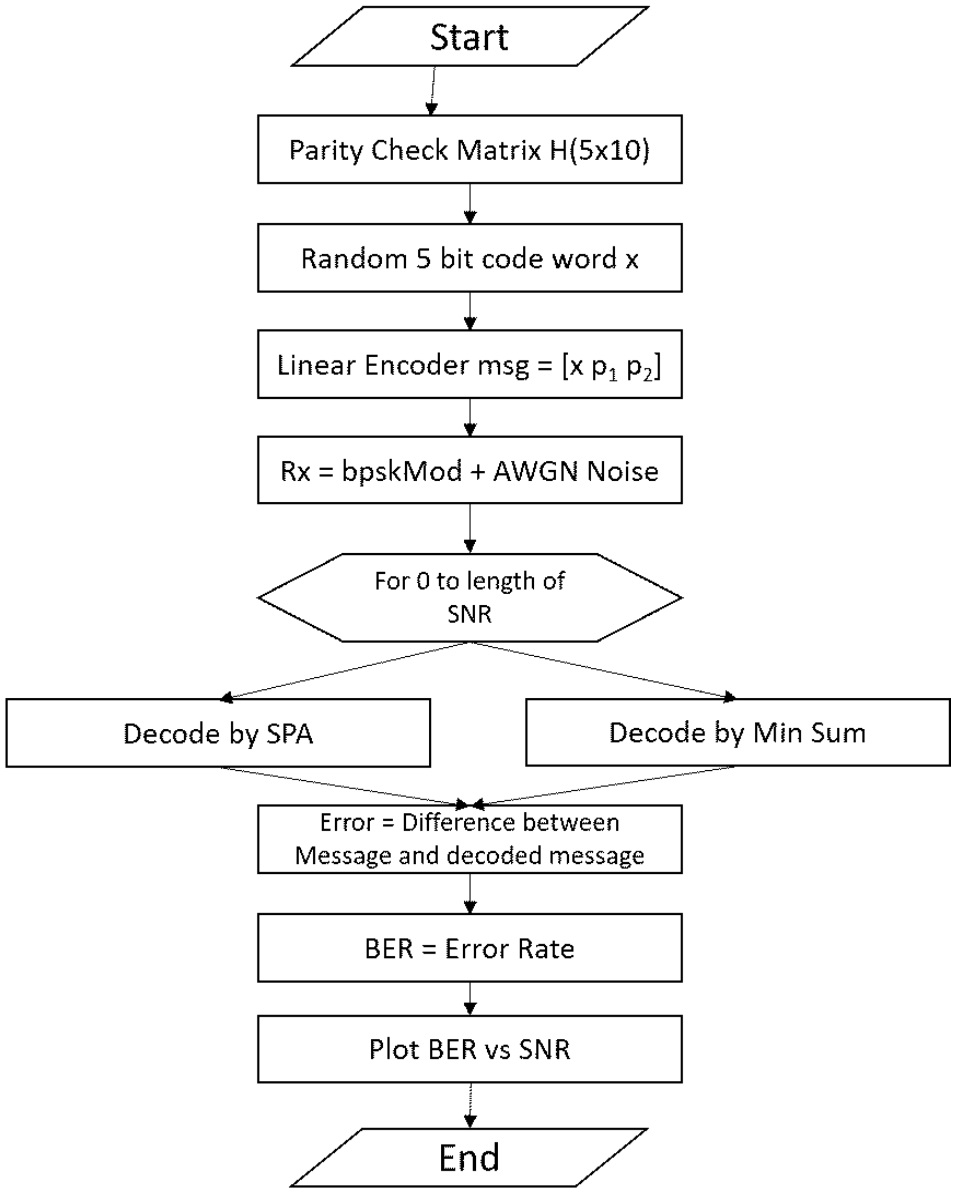
Start
Parity Check Matrix H(5x10)
Random 5 bit code word x
Linear Encoder msg = [x p_1 p_2]
Rx = bpskMod + AWGN Noise
For 0 to length of SNR
Decode by SPA
Decode by Min Sum
Error = Difference between Message and decoded message
BER = Error Rate
Plot BER vs SNR
End

Figure 2.16: Basic Structure of LDPC Simulation - MATLAB

2.8 Simulated Results on MATLAB

For the simulation, a function with input parameters for j, k, m, a and b creates the specified Quasi Cyclic LDPC code. After which that matrix is run on the MALT form rearrangement function which uses row column permutations for each of them iteratively, which will display the graphs of both the original ad rearranged matrices. The SNR values are specified from 0 to 5 dB and the frame is set for N_2 iterations where the PCM is $[N_1, N_2]$. For each iteration of the SNR and Frame the message bit is encoded, BPSK modulated, AWGN passed and decoded by SPA and MSA decoders in parallel, as the Bit Error Rate for each iteration is accumulated. Once the loop is complete, the BER for SPA and MSA are plotted against the SNR values in dB. This method was able to achieve BER in the range of 10^{-6}.
An entire MATLAB Simulation was created to simulate the transmission of data. The software was used on a Windows 10 Intel Dual Core i5 - 7200 of 2.5GHz. Simulations were run on all available cores simultaneously using Parallel Pool. LDPC codes in the range of 100's had a faster pre processing run and overall simulation. Codes of higher length took some processing. For instance, codes above the length of 1000's took anywhere from 4 - 8 hours. During the thesis period, the simulation was run on an Intel Xeon E3 with 3.5GHz, which had a higher speed for the same range, cutting down the time from 6 hours to mere 30 minutes in total. Results are displayed below.

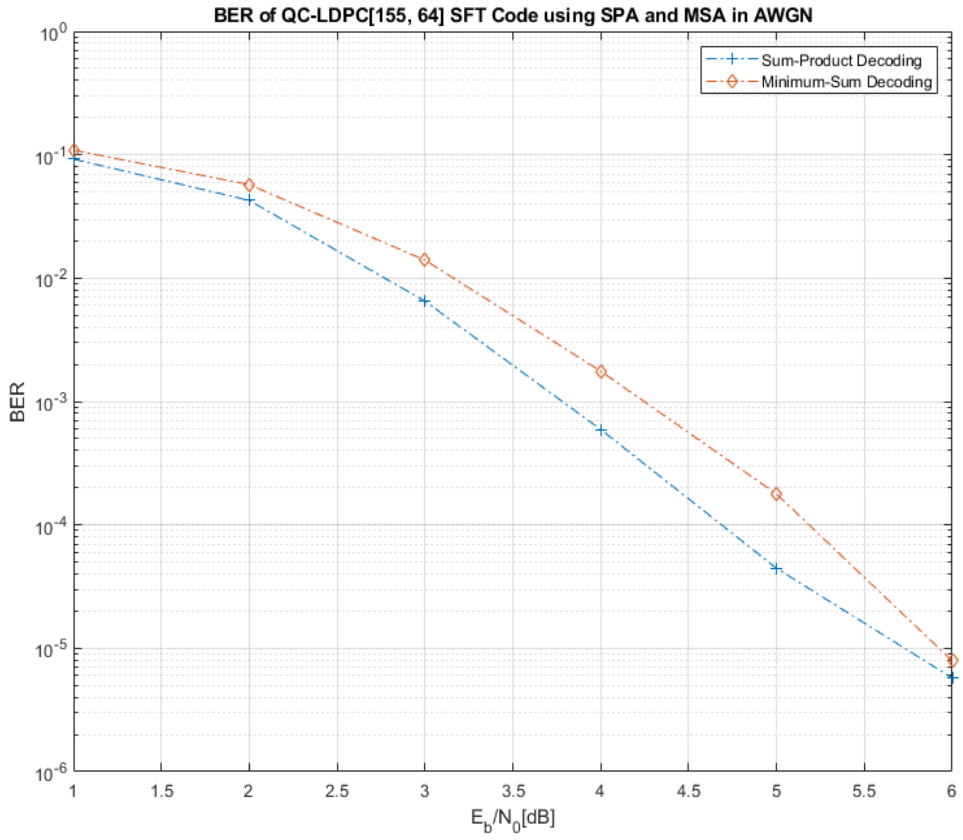

Figure 2.17: [155, 64, 20] Tanner Code Results comparison. SPA is the Blue Line

Chapter 3

Girth

3.1 Short Cycles and Tanner Graph

The scientific community has been studying factor graphs or bipartite or Tanner graphs vastly due to their practical applications for Artificial Intelligence, Computer Science, Communication engineering, statistics, channel coding, wireless systems, acoustics etc. These graphs are decoded through any message passing algorithm.
Quasi Cyclic Low Density Parity Check Codes (QC-LDPC) have shown remarkable performance with respect to approaching the Shannon Limit in a noisy distorted channel. But recent studies have shown that the removal of short cycles in the Tanner Graphs of the code produces a better Bit Error Rate. The presence of short cycles in the Tanner graph increases the probability of errors. These short cycles tend prevent the soft decision decoders from converging. They degrade the performance of the code itself because they hinder with the exchange of intrinsic information.
A cycle is a looped path within the codes defined by the 1's alternating between the check node and variable node, ending at its starting point. During decoding, the alternating between the check and variable nodes tend to cause errors as wrong message bits tend to cycle making it hard for the extrinsic information to update in time. The shorter the cycle, the more errors it tend to produce due to the frequency of the alternating in a short cycle.
The shortest cycle available in a tanner graph of any LDPC code is its girth[26]. So naturally, the best decision is to eliminate them. There are multiple methods to iteratively find a specified length short cycle and eliminate them. But the more preferred way is to eliminate them during construction itself. There are many methods of QC LDPC code construction which can be directly created with a particular girth level. That is, during construction, all short cycles below the girth level are non existent and the short cycles with the girth values are the only ones present. The higher the girth of a code, means better BER. But studies have shown that generally for most

codes, elimination of short cycles of 4, 6 and 8 tends to work well. For larger codes, girth above 12, tends to have a nullified effect in the BER improvement.

Amongst many proposed methods[27][28] to create a QC-LDPC code of particular girth ranging from 4 to 12, Hill Climbing, Progressive Edge Growth (PEG), Modifies (PEG), Girth Cycle Embedding (GCE) and Simulated Annealing (SA) algorithms show promise as one of the top choices for construction. This thesis has employed Simulated Annealing (SA) method for cycle free QC-LDPC code construction presented by Usatyuk and Vorobyev from South-West State University and Institute for Information Transmission Problems, Russia.

3.2 Simulated Annealing

3.2.1 Tanner Graph

Tanner Graph is the graphical representation of a binary parity check matrix. It shows the connection between various check and variable nodes of the matrix. For example consider a random binary LDPC parity matrix as shown below. The rows are the check nodes c and the columns are the variable nodes v. This matrix has 6 variable nodes and 4 check nodes. The tanner graph is constructed by connect all the respective variable nodes to the check nodes which has data, that is, the cells with 1's in it.

$$H_1 = \begin{array}{cccccccc} & V1 & V2 & V3 & V4 & V5 & V6 & \\ \left[\right. & 1 & 1 & 0 & 1 & 0 & 0 & \left.\right] C1 \\ & 0 & 1 & 1 & 0 & 1 & 0 & C2 \\ & 1 & 0 & 0 & 0 & 1 & 1 & C3 \\ & 1 & 0 & 1 & 1 & 1 & 0 & C4 \end{array}$$

Figure 3.1: A random binary PCM

The resultant is the tanner graph formed for the binary matrix.

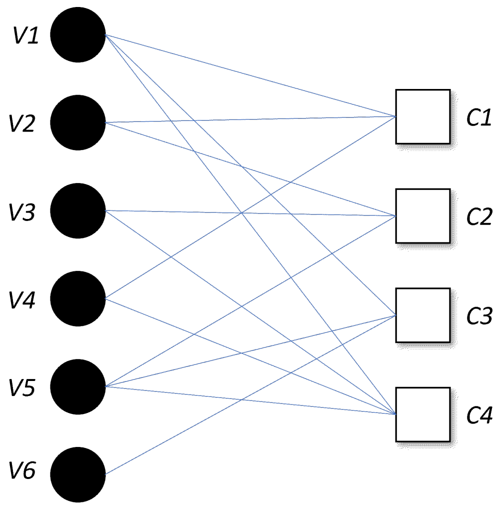

Figure 3.2: The Tanner Graph representation.

3.2.2 Short Cycles from Tanner Graphs

In the Parity check matrix, it is observable that it has short cycles, specifically in the range of 4. Consider one for example. $V_3C_2, V_3C_4, V_5C_2, V_5C_4$. These four cells in the Parity matrix forms a short cycle of girth 4. This short cycle is highlighted in the Tanner graph shown below.

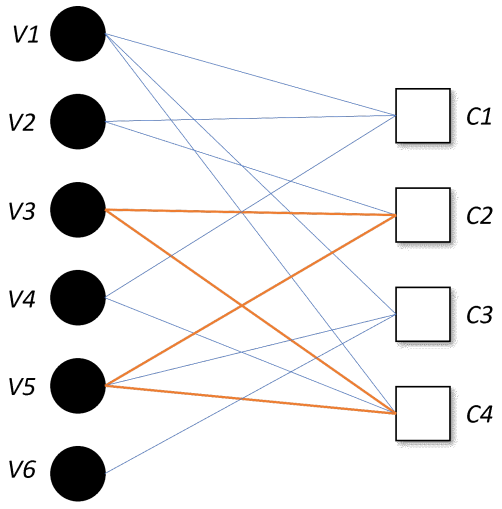

Figure 3.3: Short Cycle 4 in the Tanner Graph representation.

With all the other short cycles included, they tend to affect the bit Error Rate marginally. Identifying all the short cycles of range 4, Apply row column permutations to rearrange the matrix without those short cycles. Matrix H_2 has all its short cycles of range four removed.

$$H_2 = \begin{array}{cccccc|c} V1 & V2 & V3 & V4 & V5 & V6 & \\ \hline 1 & 1 & 0 & 1 & 0 & 0 & C1 \\ 0 & 1 & 1 & 0 & 1 & 0 & C2 \\ 1 & 0 & 0 & 0 & 1 & 1 & C3 \\ 0 & 0 & 1 & 1 & 0 & 1 & C4 \end{array}$$

Figure 3.4: Short Cycles of range 4 eliminated.

Notice that the matrix is sparse and has exactly the least required amount of changes necessary to remove the short cycles of range four from the parity check matrix H_1. A tanner graph representation has an implied effect where one nodes of each of the short cycles are eliminated which breaks the loop, thereby nullifying that particular short cycle.

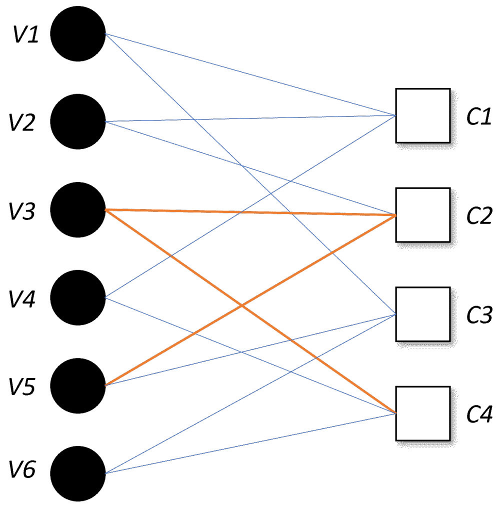

Figure 3.5: Short Cycles of range 4 removed in the Tanner Graph representation.

A simulation of the Bit Error Rate of both the matrices, where one has short cycles of 4 and the other was removed, the Bit Error Rate shows that the improvement in BER is significant in the absence of short cycles.

From the construction of Quasi Cyclic LDPC Codes from Chapter 2, the expanded parity matrix H, is the identity matrices that populated the parity check matrix cyclically shifted defined by their GF number.

The message passing decoder will fail to converge if the erased bits include a set of code bits, S, which are a stopping set/ *Trapping Set*. A stopping set, S, is a set of

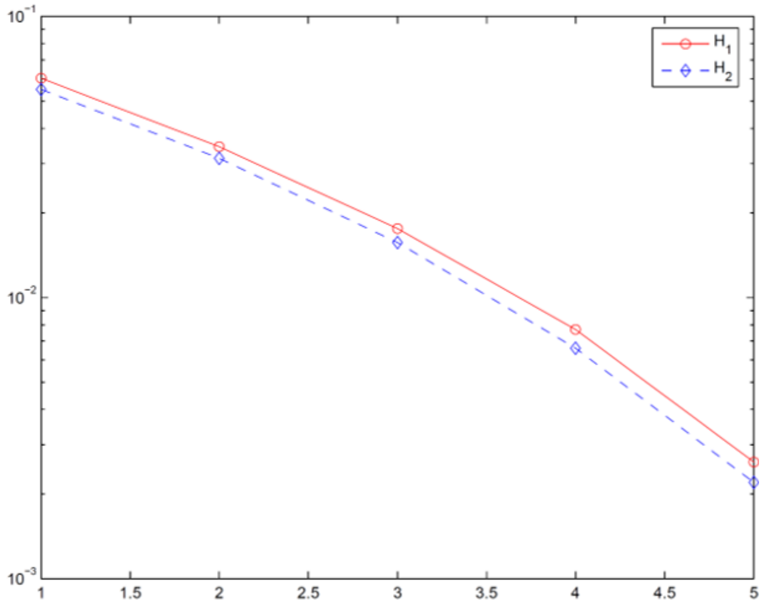

Figure 3.6: BER Results of the Short Cycles of range 4 removed. H2 is the matrix sans short cycles.

code bits with the property that every parity-check equation which checks on a bit in S checks on at least two bits in S. The size of a stopping set is the number of bits it includes. The message-passing decoder cannot correct a set of erased bits S which are a stopping set. Since every parity-check node connected to S includes at least two erased bits there will never be a parity-check equation available to correct a bit in S, regardless of the number of iterations employed. In a sense the decoder has converged to the stopping set. The stopping set distribution of an LDPC parity-check matrix determines the erasure patterns for which the message-passing decoding algorithm will fail in the same way that the codeword distribution of a code determines the error patterns for which the ML decoder will fail. The minimum stopping set size determines the minimum number of erased bits which can cause a decoding failure.

$$H = \begin{bmatrix} I_1 & I_a & I_{a^2} & \dots & I_{a^{k-1}} \\ I_b & I_{ab} & I_{a^2b} & \dots & I_{a^{k-1}b} \\ \dots & \dots & \dots & \dots & \dots \\ I_{b^{j-1}} & I_{ab^{j-1}} & I_{a^2b^{j-1}} & \dots & I_{a^{k-1}b^{j-1}} \end{bmatrix} \tag{3.1}$$

A short cycle in a matrix is called the block-cycle and is defined by $2l$, its length. The sequence of integers determining the cyclical shift of each circulant identity matrix (i x j) defined by its GF(L) is called the exponent chain sequence. The method to identify short cycles in a Tanner Graph is given by

Theorem 1 *An exponent chain forms a cycle in the Tanner graph of H if the following condition holds*

$$\sum_{i=1}^{2l}(-1)^i a_i \equiv 0 \ mod \ L \tag{3.2}$$

Extrinsic Message Degree (EMD) is the number of check nodes connected to a variable node which constitutes the short cycle in a Tanner Graph. As each short cycle pose a threat of message distortion through errors, EMD becomes a vital variable in determining the amount of short cycles. The EMD can be calculated by the approximate cycle EMD (ACE).

$$ACE(C) = \sum_{v \in E(V_c)} (d(v) - 2) \tag{3.3}$$

Here C is the cycles, V_c is the variable nodes and *d(v)* is the degree of the variable node in forming a cycle.

3.2.3 Simulated Annealing Algorithm For QC-LDPC Code Construction

Consider a QC-LDPC matrix of *m by n* dimensions and the circulant identity matrix size L. The Mother Matrix will be *M(H)* and the expanded *exponent* matrix is $E_{ij}(H)$. Code length will be *nL*. Simulated Annealing can be applied on the basis of -

1. girth g
2. minimal EMD
3. minimum code word weight

All these basis ensure the construction of high girth QC-LDPC codes bereft of short cycles. Higher Girth requires search and identification of short cycles in any construction. Generally greedy algorithms are entrusted with this task, but the Simulated annealing method promises high girth LDPC codes with minimal circulant identity matrix size compared to other methods, making this efficient.

By increasing the minimum EMD value, the trapping sets with equal probabilities gets eliminated. The trapping set (a, b) error probability depends on its ratio

Algorithm 2 Simulated Annealing algorithm to construct QC-LDPC codes with higher girth[23]

```
Require: M(H) - base/mother matrix, L - circulant identity matrix size, g - girth,
    EMD - minimal value, Iter - Max Value, seed - Pseudo random generator, Temp
    - Initial value
 1: $Nstep = 0$
 2: $i, j = rnd(seed)$
 3: for $it = 0; it \leq Iter; it++$ do
 4:     while $M_{ij}(H) = 0$ do
 5:         $i, j = rnd(seed)$
 6:     end while
 7:     for $k = 0; k \leq L-1; k++$ do
 8:         $\Theta_k = enumcircycles(i, j, k, g, EMD)$
 9:         $P(k) = w(k) / \sum_{m=0}^{L-1} w(m)$
10:         $w(k) = \epsilon^{-\frac{\Theta_k}{Temp}}$
    where $\Theta_k$ - number of cycles through $E_{ij}$, (H) - circulant identity matrix with
    shift $k$, P($k$) - probability of k shift circulant matrix choice, w($k$) - weight;
11:     end for
12:     $\Phi = enumcycles(E(H), g, EMD)$
    where $\Phi$ is total cycles in binary exponent matrix E(H);
13:     $E_{ij}(H) = rndshift(P, Temp)$
14:     $NStep++$
15:     $Temp = \eta \frac{\Phi}{NStep^2}$
    where $\eta$ is a constant
16: end for
    return E(H)
```

b/a. Unfortunately probability of Trapping Set errors depend on the parameters of the decoder: scale and offset values, scale for message quantization, input bits and message bit, considering errors only in information bits, puncturing, shortening and etc.

An initial exponent matrix can be generated by taking sub optimal random values of shifts for non-empty circulant identity matrices. Construction of exponent parity matrix of QC-LDPC codes is done by greedy choice of circulant permutation matrices value (shift value). This exponent matrix usually has low girth and EMD value due sub-optimality of circulant permutation matrices shift choice. To improve cycles properties of initial exponent matrix, simulated annealing optimization method helps. The process is looped iteratively to improve the initial parity check matrix by repeating the procedure defined by the pseudo code.

The proposed simulated annealing method progresses as described here.

- The counter *Nstep* is initialized to zero.
- A random element present in the base matrix is chosen $h_{j,i}$
- All possible cycles determined by girth value g is determined in this circulant matrix $h_{j,i}$
- Enumerate all the cycles found Θ for all possible cyclic shift in $a_{j,l} \in 0 ... L-1$
- Randomly one value of $a_{j,l}$ is chosen based on the probability depending on the number of cycles and *Temp*. Shift selection probability circulant $a_{j,l}$ is given by,

$$P(a_{j,l}) = \frac{w(a_{j,l})}{\sum_{i=0}^{L-1} w(i)} \tag{3.4}$$

  where the probability density function $w(a_{j,l})$ increases with decreasing cycles Θ and decreases with decreasing *Temp*.

$$w(a_{j,l}) = e^{\frac{-\Theta(a_{j,l})}{Temp}} \tag{3.5}$$

- Increment the counter variable *NStep* and calculate the new *Temp* value,

$$Temp = \eta \frac{\Phi}{NStep^2} \tag{3.6}$$

  where η is a constant multiplier, Φ is the number of cycles in the expanded parity check matrix $\boldsymbol{H}$.

- The loop is terminated when after *NStep* iterations, the number of available cycles Φ has not decremented, else restart the loop.

3.3 Simulated Results of Simulated Annealing on MATLAB

A similar simulation environment as of the previous was employed to simulate for this instance. Simulated Annealing method was used to create Parity Matrices of certain girth, which was imported into MATLAB which then populated the parity check matrix based on the circulant matrices produced by simulated annealing.

This was then encoded using Modified Approximate Lower Triangular (MALT), linearly encoded and parity bits added, and the message was transmitted in an Additive White Gaussian Noise (AWGN) channel and BPSK modulated. The receiver decoded the higher girth matrices using Sum Product and Min Sum Decoding Algorithms (SPA and MSA), and the Bit Error Rates (BER) proved the validity of the Simulated Annealing Algorithm. The simulation was run on a Quad Core Intel Xeon E2 with 16 GB RAM. Various parity check matrices of various lengths of code were created and run on Parallel Pooling, various results of those simulations are presented below.

A crucial observation made by Sarah J Johnson[12] of the University of Newcastle, was validated by the simulations. For a larger LDPC codes, elimination of short cycles above 4, 6, 8 tend to have diminished improvements in BER rate over increasing block length.

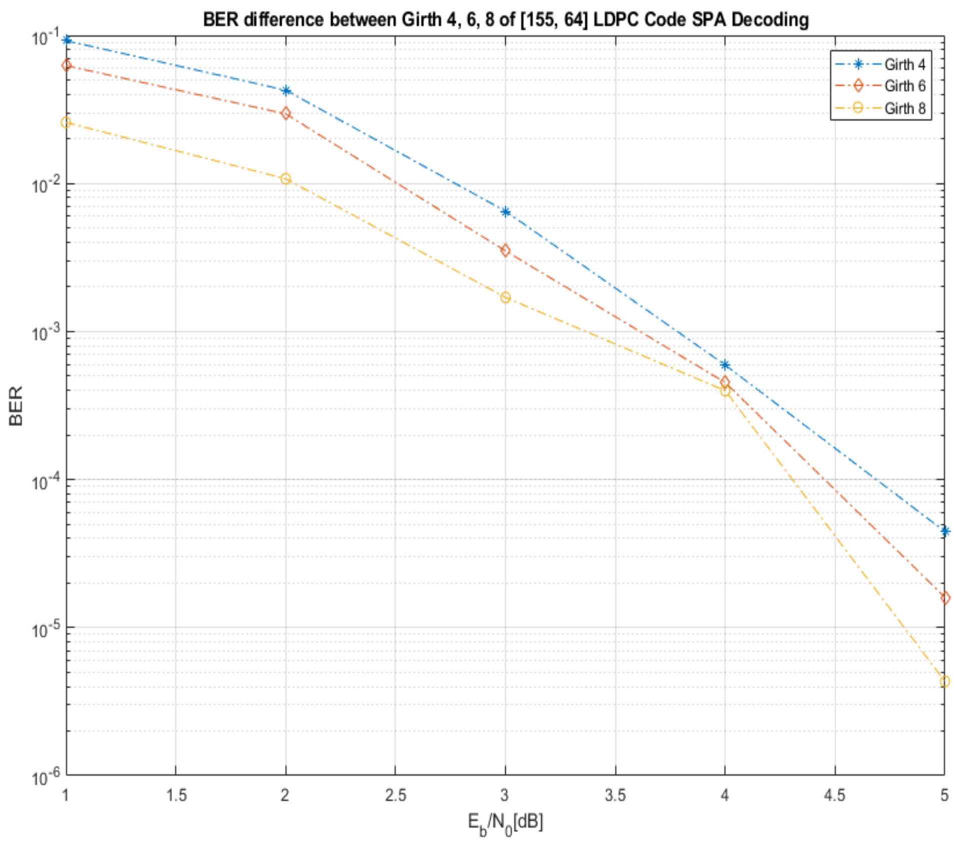

Figure 3.7: BER difference between Girth 4, 6, 8 of [155, 64] LDPC Code SPA Decoding

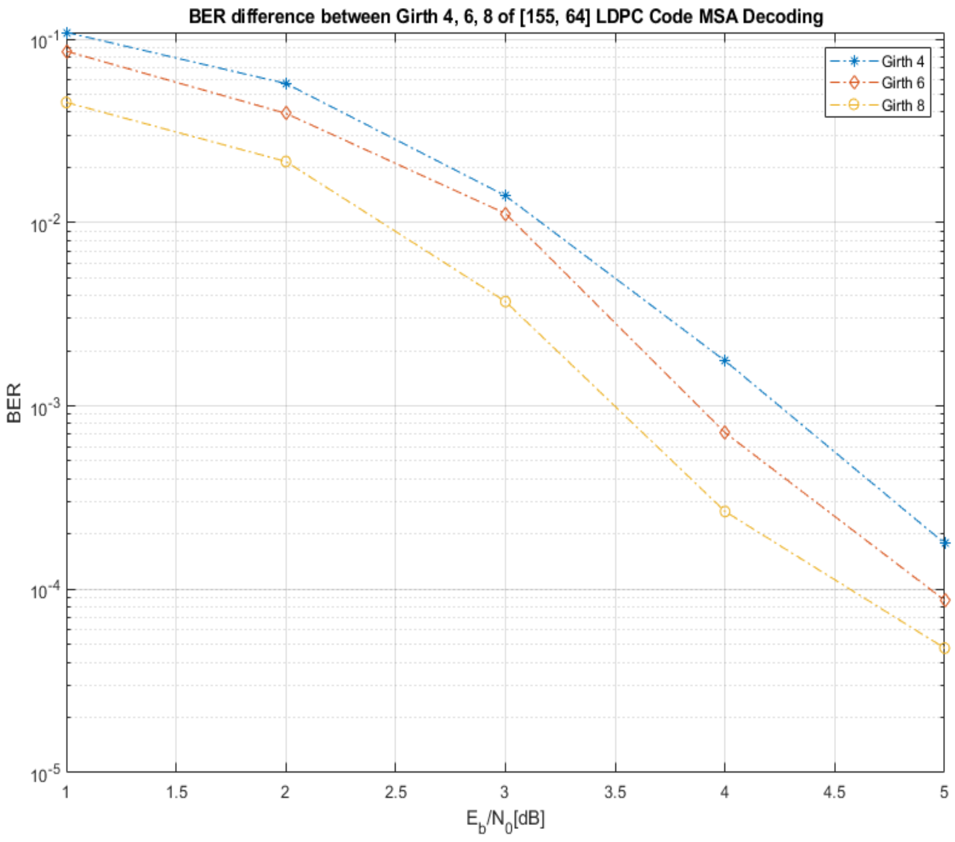

Figure 3.8: BER difference between Girth 4, 6, 8 of [155, 64] LDPC Code MSA Decoding

Chapter 4

Zipper Codes

Zipper Codes are the newest addition to the class of Spatially - Coupled Product - Like Codes. It is a form of Generalised LDPC Codes with each variable node having a degree of two. Zipper Codes can be applied to Staircase Codes, Braided Block Codes, Swizzle Codes, Tiled Diagonal and Delayed Diagonal Codes. This thesis uses Zipper Codes on Staircase Codes for demonstration purposes.

4.1 Zipper Code - Format

The Zipper Code has an arrangement similar to a zipping pair and hence, the name. It is an efficiently interleaved arrangement of codes where in the interleaving format resembles the interleaved structure of a zipping line.

4.1.1 Zipper Code Formation based on Staircase Codes

Zipper code, as mentioned earlier is an arrangement of a set of code-words.

$$c_0, c_1, c_2, c_3, ... \tag{4.1}$$

where each of them is a code-word of a constituent code or inner code which could be any Forward Error Correction Code $\boldsymbol{C}(n, k, d)$. Where n is the length, k is the dimension of message, d the minimum hamming distance. The Zipper Code is considered in two halves, its real and virtual pair. For staircase codes, it is formed of a series of blocks of data of dimensions *m x m*. The are in the form of $[B_i, B_{i+1}^T]$ where its length is *2m*.

From the figure (4.1), each block is the size of m by m, and n is *2m*, k is the dimension of the information bits, and p for the parity. At the beginning of the transmission, the transmitter will relay a known message (all - zero) to begin transmitting chunks of data. The information bits are randomly filled up in the B_1^T. Then each $[B_0, B_1^T]$ is encoded using the constituent code's encoder and the parity bits are

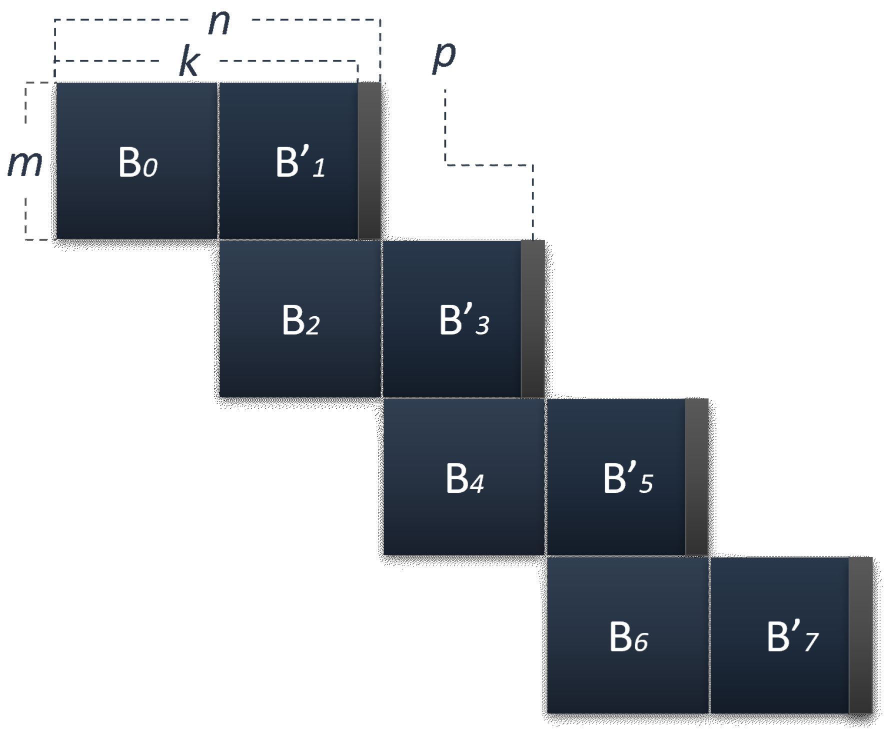

Figure 4.1: Staircase Arrangement with n = 2m

calculated and the periods of data are produced. For the next period, the previous block is transposed and appended as B_2 and the process is reiterated as necessarily defined by the period.

To re construct the Staircase into a Zipper Code, as previously mentioned, the data is split into virtual and real pairs. To begin transmission, for instance for the period $[B_1, B_2^T]$, the previous block B_1^T is transposed to serve as B_1, followed by information bits being filled up in B_2^T. Now each $[B_1, B_2^T]$ is encoded by the constituent's encoder and parity bits are calculated. The width of both the real and virtual pairs are m, and together the width is $2m = n$.

Let m_i for i ranging from 0, 1, 2, ... such that

$$0 \leq m_i \leq k \tag{4.2}$$

Every code-word is denoted as the $(i, j)^{th}$ entry of the code is the j^{th} entry of the i^{th} code-word. Now let A and B be,

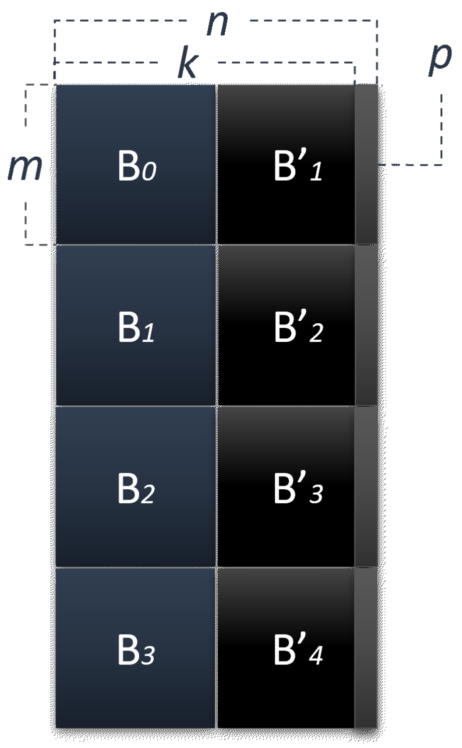

Figure 4.2: Zipper Arrangement of a Staircase Code with n = 2m

$$A_i = (i, j) : j \in [m_i] \tag{4.3}$$

$$B_i = (i, j) : j \in [n]\backslash[m_i] \tag{4.4}$$

and

$$A = \bigcup_i A_i, \tag{4.5}$$

$$B = \bigcup_i B_i \tag{4.6}$$

A is the virtual set and B is the real set and together they form the zipping line.

4.1.2 Interleaving

The interleaving function ϕ is defined as,

$$\phi : A \Rightarrow B \tag{4.7}$$

which is the description of the co-ordinates of the bits in the virtual set mapped to the real set. Every bit in the virtual set must be a direct copy of the previous real set. The interleaving function ϕ for Staircase codes for period m and parity r is defined by,

$$\phi(mi + r, j) = (m(i-1) + j, m + r) \qquad where\, r \in [m]. \tag{4.8}$$

For encoding, each k bit information bit is of the form $[B_i, B_i^T - r]$. So the encoder memory size is $m^2 + m$.

4.1.3 Code Rate

Code Rate of a Zipper code is given by the ratio of information bits in the real buffer for one period v.

$$R = \frac{\sum_{i=0}^{v-1}(n - m_i - r_i)}{\sum_{i=0}^{v-1}(n - m_i)} \tag{4.9}$$

$$= 1 - \frac{\overline{r}}{n - \overline{m}} \tag{4.10}$$

$$where \qquad \overline{r} = \frac{1}{v}\sum_{i=0}^{v-1} r_i \tag{4.11}$$

$$and \qquad \overline{m} = \frac{1}{v}\sum_{i=0}^{v-1} m_i \tag{4.12}$$

Interleaving is assumed to be bijective meaning that the virtual and real pairs have same dimensions. So,

$$\sum_{i=0}^{v-1} m_i = \sum_{i=0}^{v-1}(n - m_i) \tag{4.13}$$

Now the rate equation is,

$$R = 1 - \frac{2\overline{r}}{n} \tag{4.14}$$

$$= 1 - \frac{\overline{r}}{\overline{m}} \tag{4.15}$$

The number of symbols in a buffer for a period is given by vn.

$$vn = 2\sum_{i=0}^{v-1} m_i \tag{4.16}$$

which implies that vn must be even.

4.2 Zipper Simulation Setting

A Zipper Simulator for any length of constituent code $\boldsymbol{C}(n, k)$ comprising of LDPC, BCH and Extended Hamming[16] Codes, for any period length p, for any buffer length, for any Sound to Noise Ratio was entirely developed on MATLAB. The simulator uses *parfor*, MATLAB's parallel processing function to utilise all available cores of the running CPU simultaneously to cut down the longer processing times.

Multiple different functions pertaining to encoding, multiple decoders, Pre-processing, Zipper Buffer generator all work in sync to simulate the transmission and decoding over various SNR values. The simulator BPSK Modulates the transmission signal then passes it through an Additive White Gaussian Noise (AWGN) channel to simulate pseudo - random errors which the hard or soft decoder decodes with each iteration to produce the right code-word post distortion. It calculates the average Bit Error Rate (BER) for every iteration over each SNR value to produce the $EbN0\,vs\,BER$ plot which has produced BER's up to 10^{-10}. Some parameters to be defined to understand the simulation scenario are as follows.

4.2.1 Transmission Setting

As the virtual pair is a copy of the previous real pair, technically Zipper Code transmits real information bits. They are linearly encoded and linearly transmitted similar to a stacked operation. Chunks or stacks of data together could be transmitted as an alternative way. Through the use of nestled loops, for each SNR value, for each buffer, a zipper buffer is created and encoded up-to to the length of the period defined, which is decoded bit-by-bit as required by the decoder's iteration limit whenever present.

4.2.2 Zipper Block Creation

A specific function with inputs of the constituent code, n, k and the period length to produce the zipper frames of multiple periods. As initialization, the first block B_0 is an all-zero matrix of dimensions m *by* m. This half of the code is the virtual pair. The information bits are generated by a pseudo random binary generator and filled from $m+1$ to k length of the information bit of the first period defined by dimensions *[m, n-k]*. The parity bits p is generated after encoding the first set of $[B_0,\ B_1^T - p]$

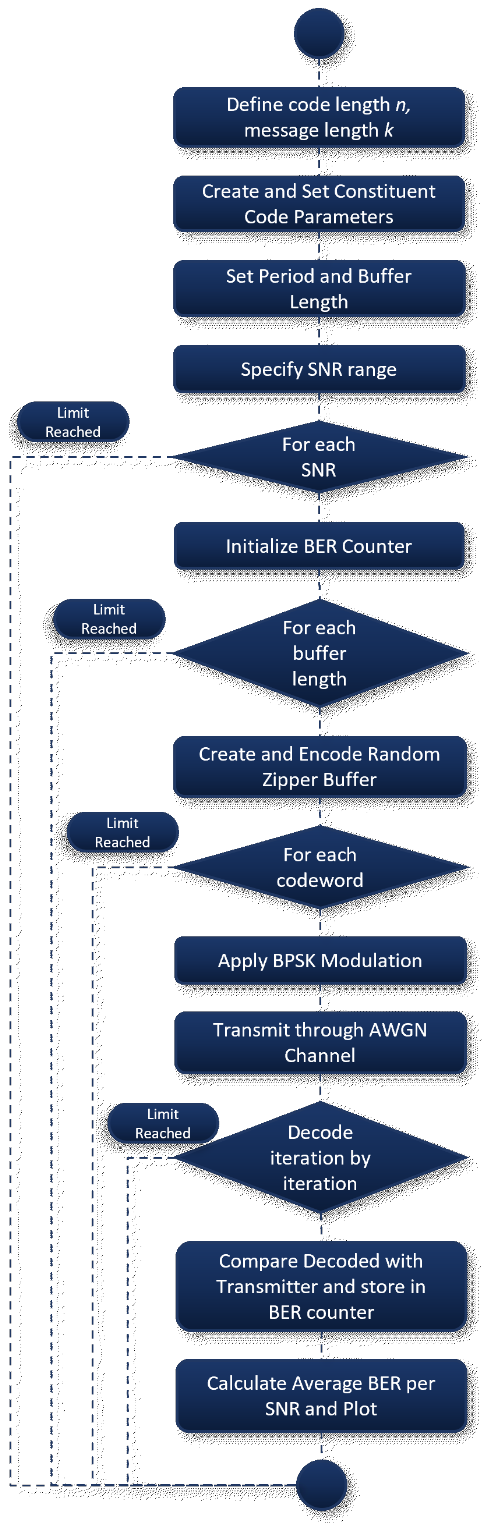
Define code length *n*, message length *k*
Create and Set Constituent Code Parameters
Set Period and Buffer Length
Specify SNR range
Limit Reached
For each SNR
Initialize BER Counter
Limit Reached
For each buffer length
Create and Encode Random Zipper Buffer
Limit Reached
For each codeword
Apply BPSK Modulation
Transmit through AWGN Channel
Limit Reached
Decode iteration by iteration
Compare Decoded with Transmitter and store in BER counter
Calculate Average BER per SNR and Plot

Figure 4.3: Zipper Simulator Flowchart

and appended as the row for that particular period. The dimensions described by *[m, m+1 : 2m]* is the real pair.

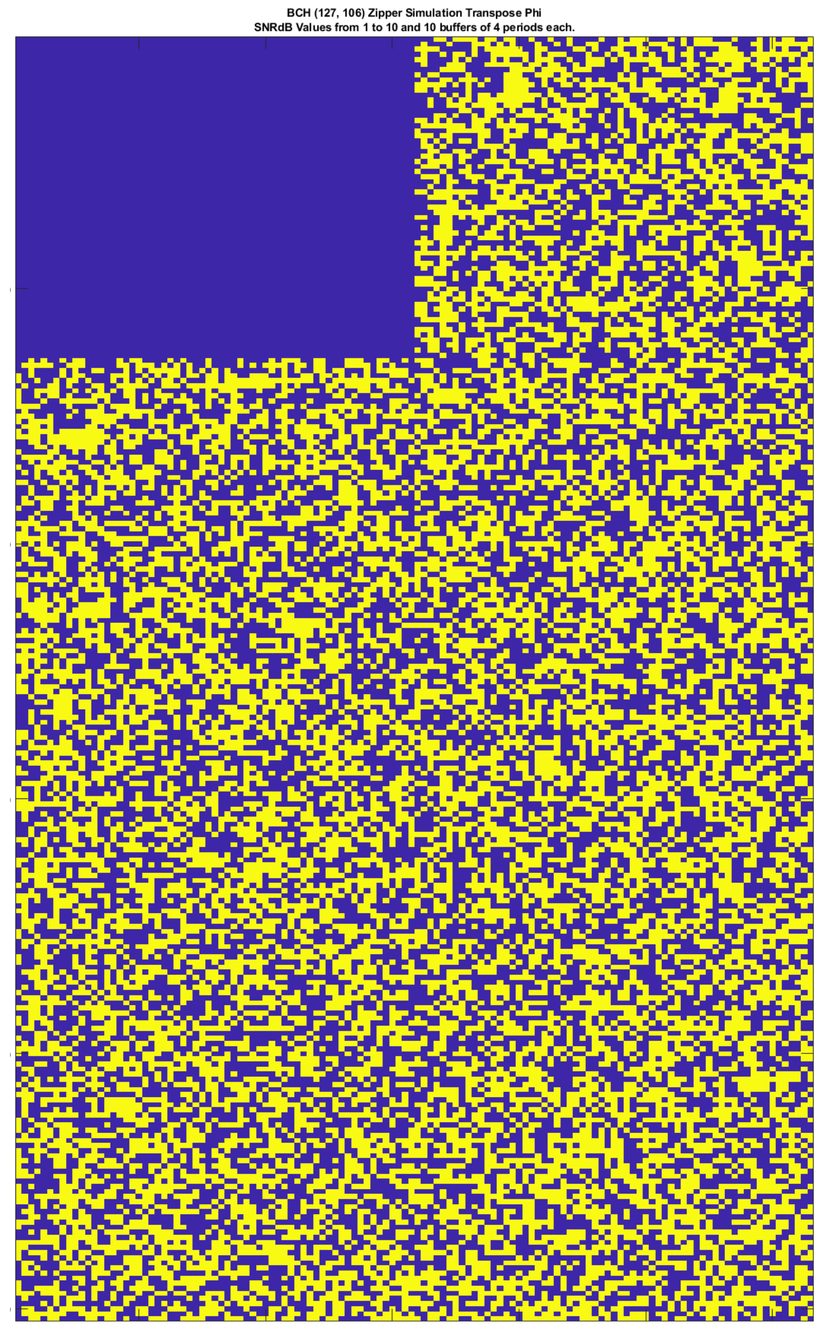

Figure 4.4: BCH Zipper Buffer Visualisation

Post generating the first period, the algorithm begins to run a recursive function until the periods from 2 to p which is the length of the period. In this loop, the previous real pair B_1^T is transposed and filled at block B_1 as the new periods virtual pair. The transposition is the result of the ϕ function as pertaining to staircase code formation. The information bits are now filled from *m+1 to n-k* then encoded in a similar fashion. This process is looped until the number of periods are created and the zipper block buffer has been formed.

In traditional Zipper codes, the Phi function describes the location of the current bit and its proportional transposed bit addresses, which is used to flip the erroneous bits during decoding, twice at the same time. This phi function can be used to mathematically generate a note of addresses of each bit of each code-word arranged in a zipper pair along with each address of its transposed copy for one whole period known as the Inter-leaver map. A period is defined by a $[B_x,\ B_{x+1}^T]$. This required an additional function that generated the inter-leaver map according to the phi function and size of the code, inputting for each run.

This research developed a new method using deduction formulas, where the address of the bit and its respective $copy^T$, for within one period starting from addresses (1, 1), that enabled mathematical reverse tracking for each and every bit throughout 100 Million bit transmissions. In essence, these formulas derived became the equivalent of a phi function in traditional Zipper, which is termed as the Bit Address Formula (BAF), a generalised formula applicable to any kind of of Zipper Codes.

The BAF for Zipper codes always has two variants pertaining to the Virtual and the real pairs of the Zipper Array. For staircase codes, the BAF for an error present in a current real pair, meaning it also has same error in its immediate future virtual pair is given.

For each error location position *epos*, with the current decoded row indicator *rowid*, the beginning and end of the current block with the current *rowid*, d_{start} and d_{end} respectively, m is half the code-word length. Then

$$\begin{aligned} \mathrm{row} &= \mathrm{rowid} - \mathrm{d_{start}} + 1 \\ \mathrm{col} &= \mathrm{epos} - \mathrm{m} \end{aligned} \tag{4.17}$$

where *row* and *col* are the indices of the bit within the current block. Then

$$\begin{aligned} \mathrm{vrow} &= \mathrm{col} + \mathrm{d_{end}} \\ \mathrm{vcol} &= \mathrm{row} \end{aligned} \tag{4.18}$$

where *vrow* and *vcol* are the future virtual indices of the transposed bit. Then the data of the current erroneous bit can be copied over the existing future virtual addresses data for both the parallel arrays.

Similarly the BAF for the condition when the erroneous bit is located in the current virtual pair of the zipper code,

$$\begin{aligned} \mathrm{row} &= \mathrm{rowid} - \mathrm{d_{start}} + 1 \\ \mathrm{col} &= \mathrm{epos} \end{aligned} \tag{4.19}$$

$$\begin{aligned} \mathrm{rrow} &= \mathrm{col} + \mathrm{d_{start}} - 1 - \mathrm{m} \\ \mathrm{rcol} &= \mathrm{row} + \mathrm{m} \end{aligned} \tag{4.20}$$

where *rrow* and *rcol* are the corresponding indices of the past real pair bit.

Many random Zipper buffers are produced throughout the simulation as defined by factors such as *n*, *k*, *p* period length and buffer length. A BCH Zipper buffer of dimensions (127, 106) was created for demonstration purposes in (Fig. 4.4). The yellow blocks are the binary 1's and notice the difference between the previous real pair and the subsequent virtual pair. A detailed algorithm on how this function works is detailed.

4.2.3 Factor Graph

The Zipper Code can be represented as a Tanner Factor Graph as shown in **Fig. 4.x**. Each variable node has connections extending to two check nodes, so every information bit in variable node can be a real or a virtual bit. One of the check nodes pertain to the constituent code $\boldsymbol{C}$ and the $=$ represents the ϕ function. Every constituent code $\boldsymbol{C}$ is a check node that connects to the n real and virtual pair of that code-word and each ϕ function is a check node connecting the virtual to the real pair variable node. Depending on the length of the period, the length of the factor graph increases linearly. The factor graph can also be converted into a periodic graph by replacing with edges for the variable and '=' nodes.

Algorithm 3 Zipper Block Creation Algorithm

```
Require: Constituent Code Parity $H$, Generator $G$, $n$, $k$, $p$
 1: $m = n/2$                                   ▷ To Split Virtual and Real buffers
 2: $zm$ = period $*$ $m$                       ▷ Initialize the entire buffer with zeros
 3: Zipper Buffer =zeros($zm, n$)
 4: Initial Period =zeros($m, n$)               ▷ For the first period $B_0$, $B_1^T$

 5: for ($i = 1 : m$) do
 6:     Initial Period($i$, $m+1$ : $k$) = randi([0, 1], 1, length($m+1$ : $k$))
    ————————————————————>                       ▷ Fill information for m+1 to k

 7:     parity = Encode(Initial Period($i$, $1 : k$))      ▷ Now encode
 8:     Initial Period($i$, :) = parity
 9: end for

10: Previous Period = Initial Period                        ▷ Set as previous period
11: Zipper Buffer($1 : m$, :) = Initial Period             ▷ Fuse Initial Period with Buffer
12: for ($p = 2 : m$) do                                    ▷ From 2nd Period on-wards
13:     Current Period = zeros(m, n)                        ▷ Initialize
14:     for ($i = 1 : m$) do
15:         Current Period(:, $i$) = Previous Period($i$, $m+1$ : $n$).$'$    ▷ The $\phi$
16:         Current Period($i$, $m+1$ : $k$) = randi([0, 1], 1, length($m+1$ : $k$))
17:         parity = Encode(Current Period($i$, $1 : k$))
18:         Current Period($i$, :) = parity
19:     end for

20:     Previous Period = Current Period
21:     Zipper Buffer((($m$ $*$ ($p-1$)) + 1) : ($m$ $*$ $p$), :) = Current Period
    ————————————————————>                       ▷ Fuse Current Period to Zipper Buffer
22: end for
```

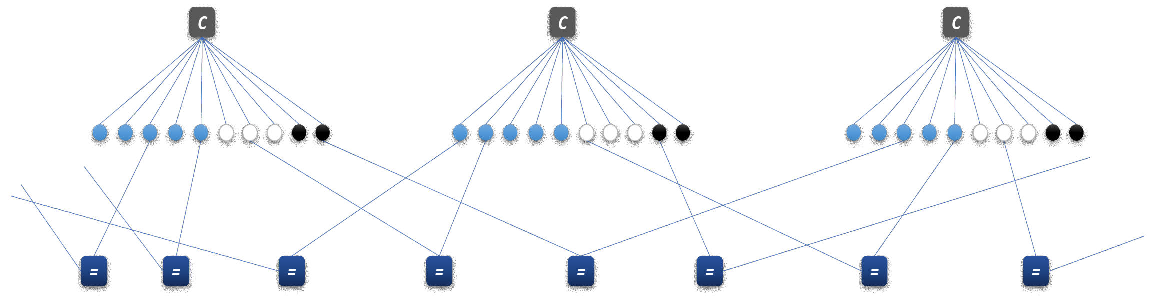

Figure 4.5: A Tanner Graph Visualisation of a Zipper Code

4.2.4 Decoding Setting

For Decoding Setting of Zipper Code, a sliding-window decoder is utilised which can decode groups of code-words M. Depending on the type of constituent code selected, the simulator uses its respective decoders. Considering the decoding procedure for a random sliding window of code-words, the decoding is linearized, where each code-word grouped is decoded in a linear fashion to prevent race conditions and collisions when using *parfor* which splits the batch of code-words to be simultaneously decoded using the available number of cores.

This decoding method also pre-computes the next up-coming rows of data, as the setup requires the virtual pair to be a copy of the real pair. So for the potentially next set of oncoming data, the simulator only updates for the current row with previous half. The simulator looks back at the previous code-word for the current set of new symbols making the decoding more precise. **Fig. 4.5** is the structure of decoding.

4.3 Simulated Results

Multiple Zipper Simulators were developed on MATLAB with constituent codes comprising of Bose–Chaudhuri–Hocquenghem Codes, Extended Hamming Codes, and Low Density Parity Check Codes. As depicted in the previous parts of this thesis manuscript, a LDPC Simulator was developed which created QC-LDPC Codes, used Modified Approximate Lower Triangular Transformation (MALT) pre-processing based Linear LDPC Encoder, Sum-Product and Minimum Sum Decoders to simulate the BER over various SNR values. Using that as primary model, simulators for BCH Codes and Extended Hamming Codes were developed, which aided in the creation of

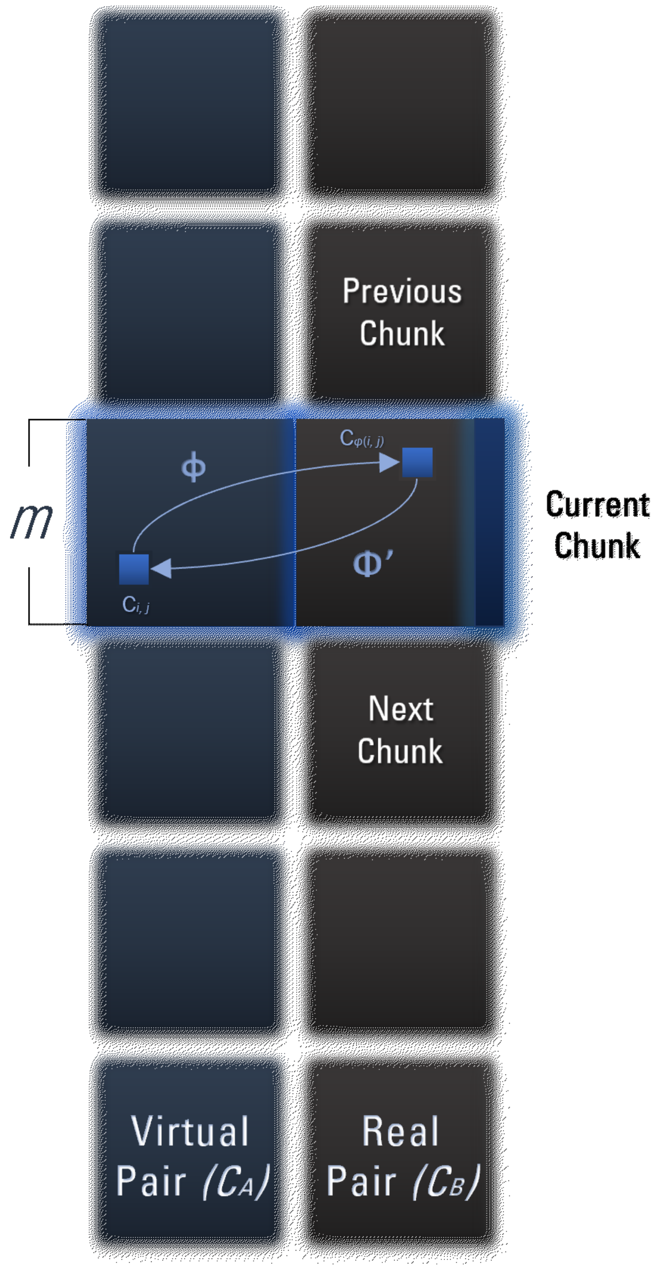

Figure 4.6: Zipper Decoding Visualisation

Multiple Zipper Simulators pertaining to each class of codes. All Zipper Simulators used a Staircase Implementation of the above mentioned class of codes to simulate and decode.

4.3.1 Bose–Chaudhuri–Hocquenghem (BCH) Codes

BCH Codes are powerful multiple error correcting codes over the finite-Galois Field. It belongs to the class of cyclic codes and became the common error - correction codes for digital communication systems such as Satellite Comms, CD and DVD Players, disk drives, SSD's and more. BCH codes have easy applicability for a hardware implementation, aiding its rise in popularity. The Zipper Simulator for BCH codes uses various codes of the staircase format.

Let a be the elements in GF(2^m) where m is any positive integer with value $m \geq 3$. BCH codes can be constructed using generator polynomial *g(x)* which is the lowest degree polynomial over GF(2). Then the roots of the polynomial are $a,\ a^2, ...\ a^{2t}$. Let $p_i(x)$ be minimal polynomial of α^i where i is a positive integer. The g(x) can be found by getting the Least Common Multiple of the series $p_1(x),\ p_2(x),\ ...\ p_{2t}(x)$. If m is the message with length k, it can be represented using polynomial $m(x) = m_0 + m_1x + m_2x^2 + ... + m_{k-1}x^{k-1}$. Then the code-word is

$$c(x) = x^{n-k}m(x) + x^{n-k}m(x)(mod\ g(x)) \tag{4.21}$$

BCH Codes of dimensions (127, 106), (255, 231), (511, 484), (1023, 993) were simulated for Zipper Codes of multiple periods and buffers over Sound to Noise Ratio values ranging from 1 to 10. The multiple nestled loops sends in one code-word at a time for decoding to avoid collisions. The channel is AWGN with BPSK modulation, and the Bit Error Rate was simulated. The **Fig. 4.7** is the BER vs SNR semilog plot of the BCH-Staircase Error only and Error-Erasure Decoding Simulation by Alvin Yonathan Sukmadji for their thesis from the University of Toronto in C++. This research managed to validate and modify it for various other class of codes. **Fig 4.8** is the BER vs SNR for BCH Codes developed on MATLAB with transposed ϕ.

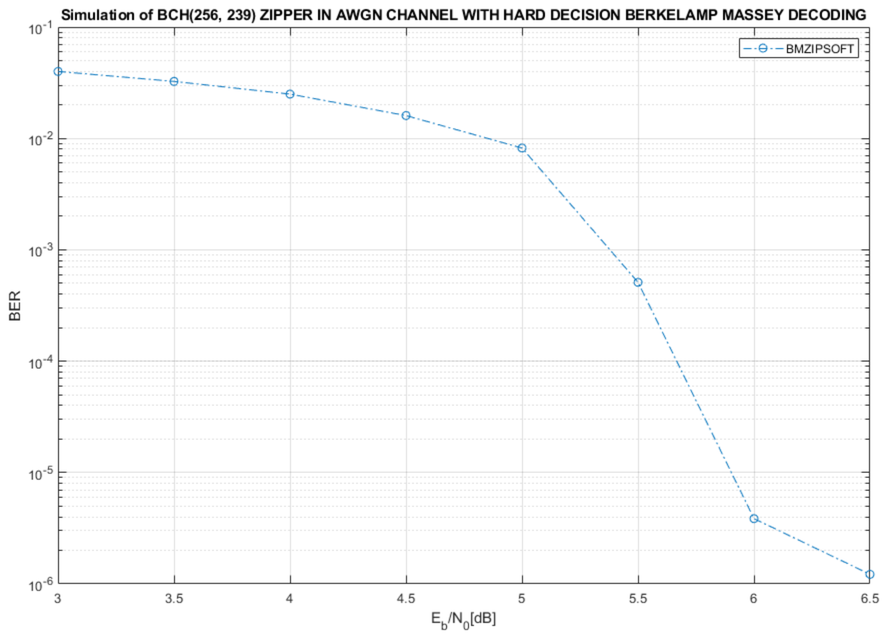

Figure 4.7: BER versus SNR for staircase codes Zipper with inner BCH(256, 239) decoded by Berkelamp Massey

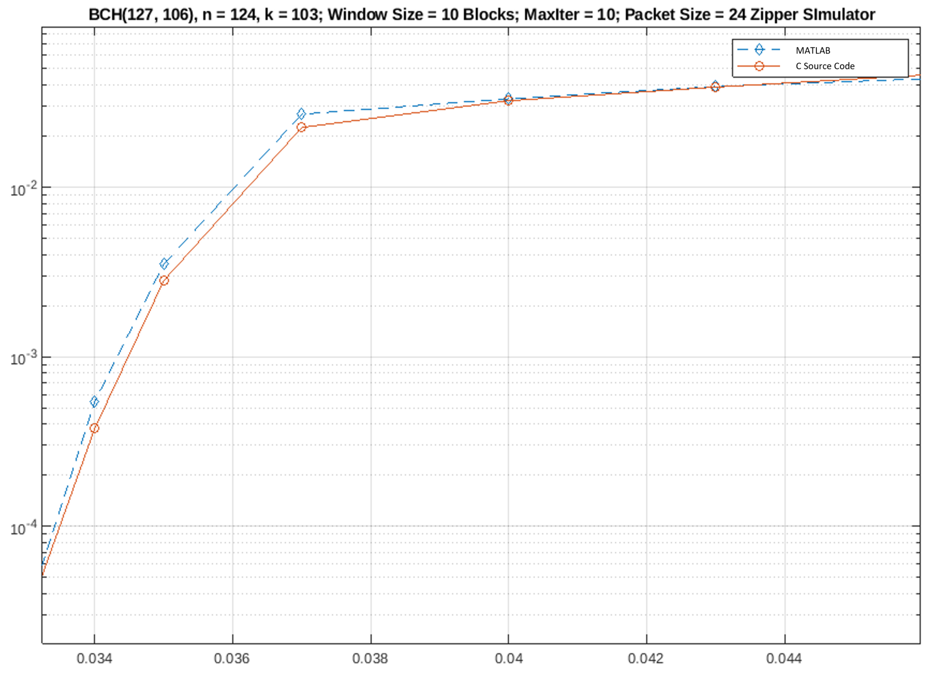

Figure 4.8: Comparison of the BERs of the same code with same parameters over BSC channel between this research's software vs Alvin Sukhmadji's[31] on C Language

Chapter 5

Stall Patterns and Simulated Annealing based Random Zipper Codes

5.1 Stall Patterns

Sometimes, during iterative decoding using Sliding Window decoding scheme, there might be some error patterns, that cannot be corrected. These are termed as Stall Patterns and they affect the performance of the code. For a constituent code with error correcting capacity, t, if the number of errors present is higher than t, then the decoder will attempt to perform a miscorrection. This will result in additional errors due to decoding failure. So error correction must only happen when the number of errors present in a code is equal or less than t.

5.1.1 Stall Pattern Terminologies

From Zipper Codes structure, the code-words are split into Virtual A and Real B pairs along the midpoint of the code-word length N. Recall that during transmission, only the real pair B is transmitted. Consider S to be the stall pattern present in B. Then

$$S^* = S \cup \phi^{-1} \tag{5.1}$$

is the Stall pattern for the corresponding virtual pair A. Then $\pi_1(S^*)$ is the number of rows in the block with that Stall Pattern and the row $i \in \pi_1(S^*)$ is correctable if,

$$|S^* \cap (A_i \cup B_i)| \leq t \tag{5.2}$$

Then the correctable rows are

$$K(S) = i \in \pi_1(S^*) : |S^* \cap (A_i \cup B_i)| \leq t \tag{5.3}$$

The decoding function is given by

$$D : P(B) \Rightarrow P(B) \tag{5.4}$$

$$S \mapsto \begin{cases} S & if K(S) = \phi \\ S \backslash (B_{k^*} \cup \phi(A_{k^*}) & otherwise, \end{cases} \tag{5.5}$$

where P(B) is the power set of B and $K^* = \min K(S)$. So that only the rows with lowest affected index along with its respective transposed copies will be removed.

For a Stall Pattern S, $|D(S)| \leq |S|$, so the correct-ability can be found by applying D, $|\pi_1(S^*)|$ times. S is correctable when,

$$D^{|\pi_1(S^*)|} = D(D(D(...D(D(S))...))) = \phi \tag{5.6}$$

A stall pattern S is minimal if for all nonempty $T \subseteq S, T$ is correctable.
A stall pattern S is always minimum sized if for all patterns T have $|T| \geq |S|$.
A minimum sized stall pattern is always minimal but converse necessarily don't. In staircase codes with $t = 2$, minimum sized stall patterns are of size $(t + 1)^2 = 9$. but there can also be a minimal, but not a minimum sized pattern of size 12.

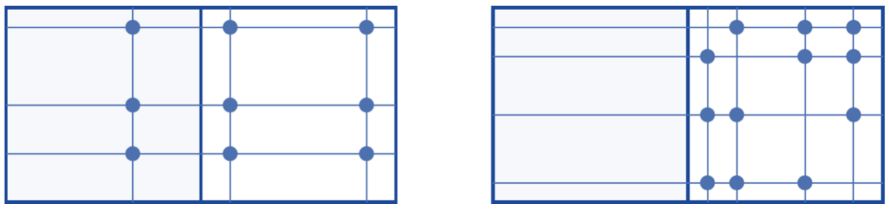

Figure 5.1: Minimum-sized and minimal, not minimum sized stall patterns of a staircase code with double-error-correcting constituent code.

5.2 Experiments on reducing Stall Patterns by limiting Girth

As mentioned in Chapter 3, A short cycle in a parity check matrix is defined as an even number set of node coordinates, starting and ending in the same node as the initial, alternating between variable and check nodes bound by binary 1^s, also known as girth. A code of girth X means that code does not have short cycles of length X up-to X short cycles. The presence of short cycles in a code is harmful, as wrong information starts looping within the cycle disabling the transfer of extrinsic information, there by decoding failure happens.

5.2.1 Manipulating the Phi Function of the Zipper Code

The phi function of the zipper code is the mathematical relation for each bit between its real and virtual pair. In zipper codes, for any form of arrangement such as Staircase, Braided Block, Tiled Diagonal, the underlying principle, is that the message part is in the real pair, which is transmitted and copied in some format into its corresponding virtual pair. The generalised form of phi for any arrangement of Zipper Code is,

$$\phi : A \Rightarrow B \tag{5.7}$$

For instance in Staircase codes, each block of the real pair, is transposed to the next virtual pair. Then the mathematical relation of the Staircase Zipper Phi function is,

$$\phi(mi + r, j) = (m(i - 1) + j, m + r) \tag{5.8}$$

where r is parity, i, j are the coordinates of each bit.

The phi function for Tiled Diagonal Zipper Codes, discussed later in this thesis, is given by,

$$\phi(wq + i, ws + j) = (w(q - s - 1) + j, w(L + s) + i) \tag{5.9}$$

where w is the size of the square tile, and i, j are the bit coordinates. L is the number of tiles, q, s is the coordinates of the tile position.

As the phi function, determines the structure of the Zipper Code, crafting a proper phi function can lead to avoidance of stall patterns.

During the research,various methods in correcting the performance of the Zipper Code were tried.

For experimentation, a Staircase Zipper code with with inner constituent code as Extended Hamming (16, 11) with it's constituent decoder, the results of transpose phi, and random row permutation was compared.

5.2.2 Simulated Annealing Zipper Codes

As discussed in Section 3.2, Simulated Annealing[23] is an effective method to create Low Density Parity Check Matrices without short cycles. In this research investigators

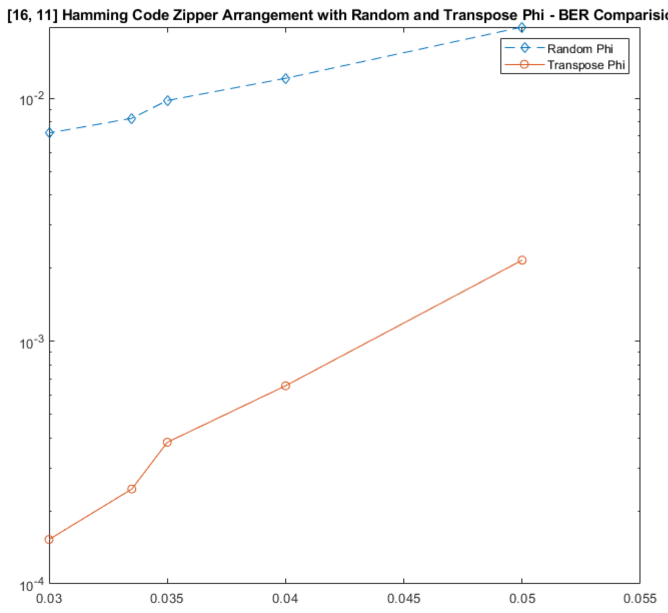

Figure 5.2: Comparison of Transpose Phi and Random Row Permutation Phi over various crossover for BSC Channel.

experimented with eliminating the short cycles of Zipper Codes by constructing a short-cycle free constituent code deriving from a specifically designed Low Density parity Check Matrix without short cycles using Simulated Annealing. The presence of short cycles prevent the exchange of extrinsic information during error correction, and codes without short cycles have shown better performance compared to otherwise.

To begin with,a special binary matrix with specific parameters defined, from which the Phi function for the Zipper Codes were to be derived was created.

Staircase Encoding Scheme and factor graphs

The encoding process of Staircase codes is given in the picture. Staircase codes are characterised by their block arrangement, in the structure of staircases and hence, its name. Blocks are encoded alternatively in a horizontal, followed by a vertical fashion. So, each bit is encoded twice and their parity bits are interleaved with horizontal and its vertical components.

The factor graph, or the tanner graph of the bits can be realised by splitting into its corresponding check nodes and variable nodes. Each bit is alternating between

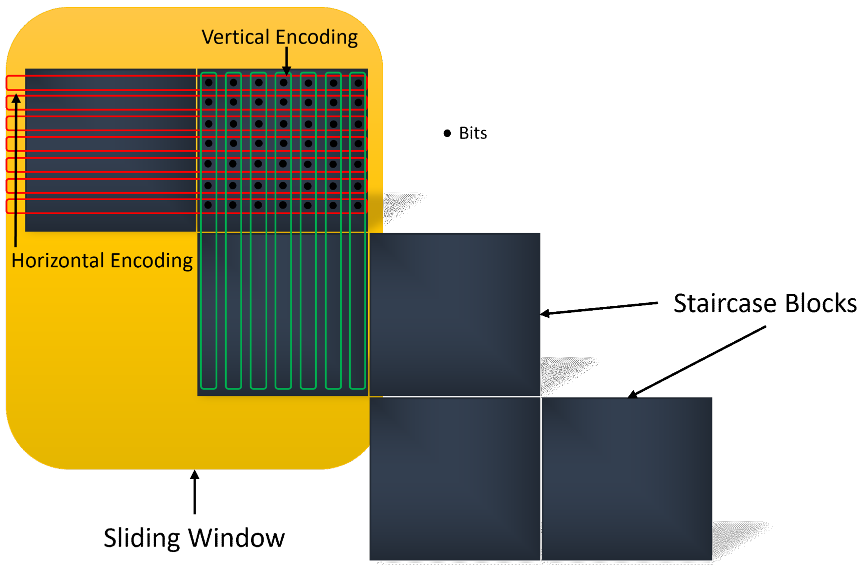

Figure 5.3: Visualization of Staircase Codes Encoding Scheme

one check node and one variable node, denoting the horizontal and vertical encoding pattern of Staircase Codes. From the picture displayed, see the factor graph of girth 8 from the Staircase Codes.

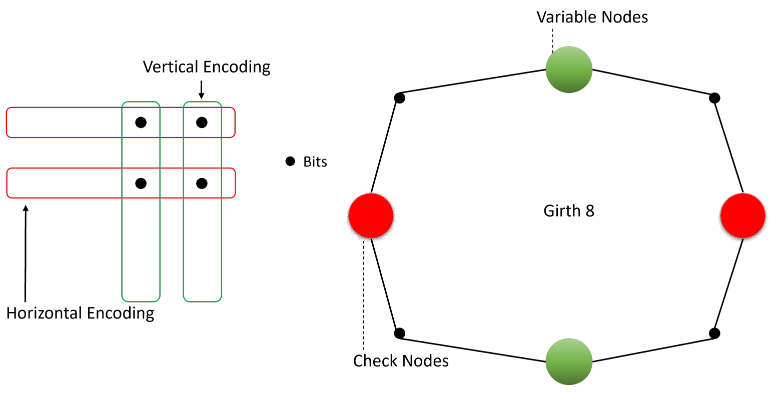

Figure 5.4: Staircase Encoding and the Factor graph indicating a Girth 8

It is also possible to derive the girth of a code from its factor graph. For instance, the Staircase code[21][39] of each block containing 3 rows and 3 columns with bits present, being encoded horizontally and vertically is given.

As every short cycle is bound by binary 1^s, starting and ending at the same node, alternating between a check node and a variable node, see the girth of a code from its factor graph.

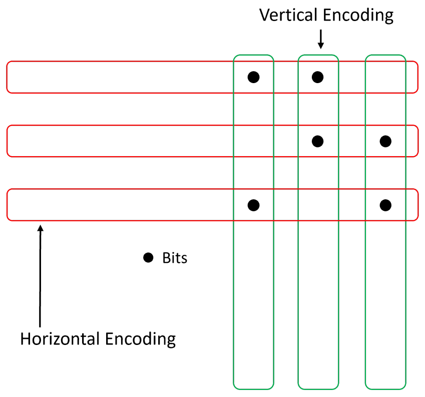

Figure 5.5: Encoding of a Staircase code with Block size 3 X 3

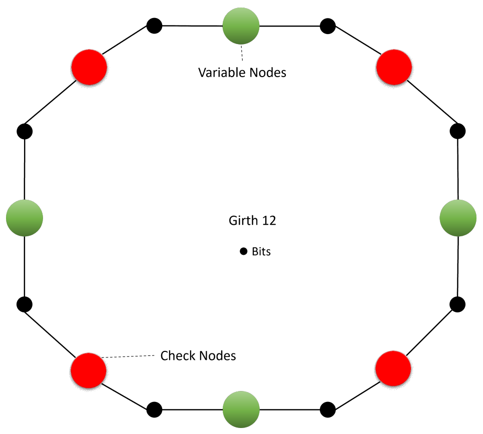

Figure 5.6: Factor Graph of a Staircase code of square block size 3, with its girth

Creation of Specific Binary Matrix

A binary Quasi Cyclic Parent Matrix of size 2 by 16 was created. Each element of the parent matrix is an identity matrix of size 8. The white squares in the figure denotes binary 1. The Quasi Cycling matrix is split into its four quadrants going anti-clock wise from the top right quadrant starting from A, B, C, D.

Quadrants A, B and C contain the same elements. Every element of those matrices of those quadrants are identity matrices of size 8. For quadrant D, an LDPC Parity Check matrix without short cycles of size 1 by 8 is created using Simulated Annealing. This matrix is Quasi Cyclic, with each of its element, cyclically shifted by the number of the element. The matrix is joined to the quadrant and this forms the Specific binary Matrix.

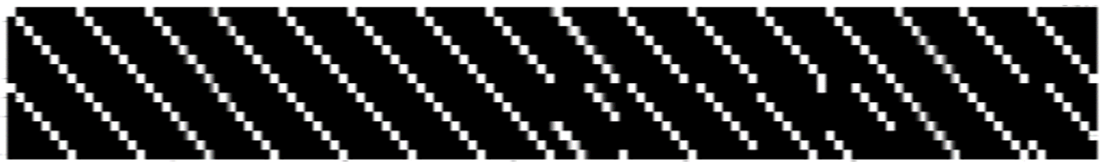

Figure 5.7: Specific Binary Matrix formed using Simulated annealing

With the specific binary matrix, check nodes and variable nodes denotes the position of each bit in the matrix. On the left quadrants, each identity matrix has variable nodes from $U_1, U_2...U_n$ followed by $U_{n+1}, ...U_{2n}$, $U_{2n+1}, ..., U_{3n}$ and so on upto $U_{7n+1}, ..., U_{8n}$.

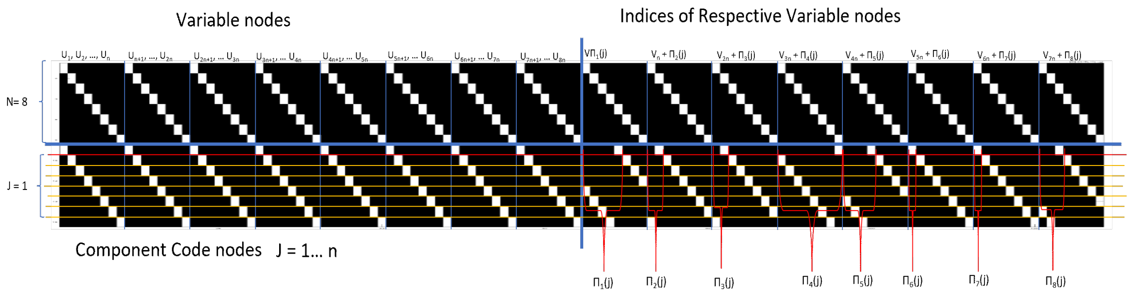

Figure 5.8: Specific Binary Matrix with variable nodes denoting Phi function

The right side of the quadrant shows the indices of the various respective variable nodes denoted by π. From $V\pi_1(j)$, to, $V_n + \pi_2(j)$, $V_{2n} + \pi_3(j)$ to $V_{7n} + \pi_8(j)$. The composition of the matrix of individual variable and check nodes is shown.

Table 5.1: The Variable and Check nodes for the Specific Binary Matrix

V1	Vn+1	**V2n+1**	**V3n+1**	**V4n+1**	**V5n+1**	**V6n+1**	**V7n+1**
V2	Vn+2	V2n+2	V3n+2	V4n+2	V5n+2	V6n+2	V7n+2
V3	Vn+3	V2n+3	V3n+3	V4n+3	V5n+3	V6n+3	V7n+3
V4	Vn+4	V2n+4	V3n+4	V4n+4	V5n+4	V6n+4	V7n+4
V5	Vn+5	V2n+5	V3n+5	V4n+5	V5n+5	V6n+5	V7n+5
V6	Vn+6	V2n+6	V3n+6	V4n+6	V5n+6	V6n+6	V7n+6
V7	Vn+7	V2n+7	V3n+7	V4n+7	V5n+7	V6n+7	V7n+7
Vn	V2n	V3n	V4n	V5n	V6n	V7n	V8n

Now from the picture, the phi function for the zipper code can be derived. For each of the Cyclic shift matrix, for each check node j, the respective index is the index

Table 5.2: Minimum required value of circulant matrix size for a parent matrix with row m = 3, column n, and Girth 10[23]

Column number	Simulated annealing	Hill Climbing [18]	Improved PEG [20]	Lower Bound [33]
4	37	39	37	37
5	61	63	61	61
6	91	103	91	91
7	155	160	155	127
8	215	233	227	168
9	304	329	323	217
10	412	439	429	271
11	545	577	571	331
12	709	758	-	-

of the bit within that check node corresponding to its variable node. Its is given by $\pi_n(j)$ where n is the number of variable nodes and j is the j^{th} check node.

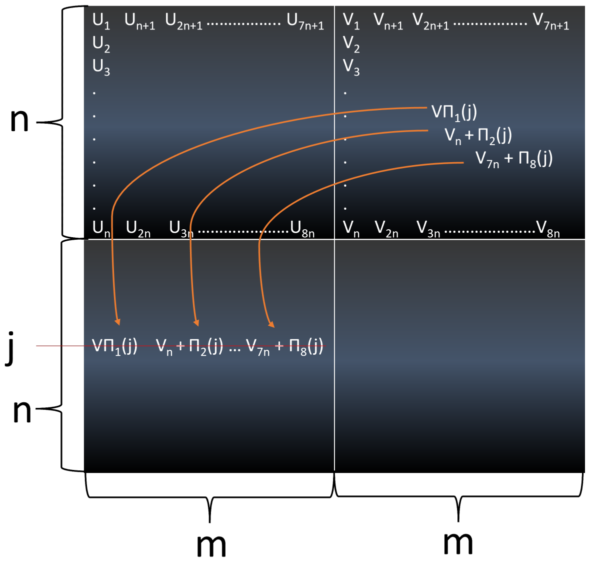
U_1 U_{n+1} U_{2n+1} .... U_{7n+1}
U_2
U_3
U_n U_{2n} U_{3n} ....U_{8n}
V_1 V_{n+1} V_{2n+1} .... V_{7n+1}
V_2
V_3
$V\Pi_1(j)$
$V_n + \Pi_2(j)$
$V_{7n} + \Pi_8(j)$
V_n V_{2n} V_{3n} ....V_{8n}
$V\Pi_1(j)$ $V_n + \Pi_2(j)$ ... $V_{7n} + \Pi_8(j)$
n
j
n
m
m

Figure 5.9: Phi function mapping for current real block to future virtual block from Specific Binary Matrix created by Simulated Annealing

The Phi function derived from the Specific Binary Matrix created by Simulated annealing is given by,

$$\Phi(j, k) = (k - 1).n + \pi_k(j) \tag{5.10}$$

where k is the variable node.

Table 5.3: Minimum required value of circulant matrix size for a parent matrix with row m = 3, column n, and Girth 12[23]

Column number	Simulated annealing	Improved PEG	Table V, [12]
4	73	73	97
5	160	163	239
6	320	369	479
7	614	679	881
8	1060	1291	1493
9	1745	1963	2087
10	2734	-	-
11	4083	-	-
12	5964	-	-

When creating Parity Matrices without specific number of short cycles, the constraint on the number of rows and columns, increase linearly with increase in the girth. Meaning, to create a Parity Check Matrix with short cycles above X, then the parent parity matrix rows and columns will be increased by a factor of X.

So, it is not possible to apply this method to a short sized code. The table for required circulant matrix size for specific girth and specific rows and columns is given.

As seen in the tables, for various girth, the demand for the size of the binary matrix increases greatly, with increase in girth. So a Zipper Code with a large constituent code is necessary to remove short cycles. It is not possible to apply for constituent codes of smaller dimensions.

Chapter 6

Improving Zipper Codes using Guessing Random Additive Noise Decoding - GRAND

6.1 GRAND Decoder

6.1.1 Hard Detection GRAND

Guessing Random Additive Noise Decoders (GRAND) [40][48] have recently been proposed as an alternative to code-specific decoders to perform ML decoding or near-ML decoding with limited complexity. Rather than using code structure, GRAND generates noise sequences based on channel statistical information, sorts them from most likely to least likely breaking ties arbitrarily, subtracts noise sequence from hard decision received sequence and queries whether the result does exist in the code-book[44][45]. The first code-book member is selected as the originally sent code-word. GRAND is a universal decoder, i.e., it can be used to decode any code. Pseudo code for GRAND is shown below.

In general, all GRAND versions are comprised of two steps. First step is to sort noise sequences in descending likelihood order, and second step is to check that a given code-word is a member of the code-book. For block codes, a simple membership check function is given by the parity check matrix of the code. While the second step is the same for all versions of GRAND algorithms, the first step can vary for each one.

GRAND algorithm is designed to generate a permutation-combination of all possible error patterns which is repeatedly tested against the received code-word to see if the decoded code-word belongs in a Code-Word-List, a list of all possible encodings which both the transmitter and receiver will posses throughout the transmission. For linear codes like Hamming Code (7,4), BCH Codes, LDPC codes, the Code-List is its parity check matrix H. This research will be demonstrated using linear (n, k) BCH codes. The type of ranking/ordering schema of the GRAND algorithm is the distinctive feature between the many GRAND variants of some, which will be demonstrated here.

6.1.2 The GRAND Algorithm

Let X^n and Y^n be the input and the output of a discrete channel made of blocks of n symbols from a finite set of characters $\mathbb{A}$ of size $|\mathbb{A}|$. The channel is a memory-less random noise N^n, independent of input and set of $\mathbb{A}^n$. Then -

$$Y^n = X^n \oplus N^n \tag{6.1}$$

which is also invertible meaning,

$$X^n = Y^n \ominus N^n \tag{6.2}$$

For decoding, both the transmitter and receiver must share the code-word list $C_n = c_n, ..., c_{n,M_n}$ having M^n elements of $\mathbb{A}^n$. Then the conditional probability of each of the received code-words are given by -

$$p(y^n|c^{n,i}) = P(y^n = c^{n,i} \oplus N^n)\; for\; i \in 1, ..., M_n \tag{6.3}$$

The decoded code-word by GRAND is then an element of the code-list with the highest probability of transmission.

$$c^{n,*} = \arg\max \left\{p(y^n|c^{n,i}) : c^{n,i} \in C_n\right\} \tag{6.4}$$

$$= \arg\max \left\{P(N^n = y^n \ominus c^{n,i}) : c^{n,i} \in C_n\right\} \tag{6.5}$$

Code-Lists are large for storage rising exponentially with block length n. The normalised rate of that code-word list is

$$R = \lim_{n} \frac{1}{n} log(M_n) \tag{6.6}$$

This is why Maximum Likelihood Decoding is infeasible as it would have to -

- Calculate $\mathbb{A}^{nR}$ conditional probabilities of equation (5.3), ordering with weighted ranks for every signal.

  or

- Calculate $|\mathbb{A}|^{n(R+1)}$ conditional probabilities in advance for every $(c^{n,i}, y^n)$ pair, storing the resulting $|\mathbb{A}|^n$ decoding for every received signal.

To combat that. GRAND offers a different approach. One where the receiver, weighted-orders the noise sequences in terms of likeliness, and sequentially calculate the received sequence by eliminating each noise sequence for every signal to find the resulting code-word which is part of the code-list.

Let the ordered list of noise sequence in terms of likeliness be,

$$G : \mathbb{A}^n \mapsto \{1, ..., |\mathbb{A}|^n\} \tag{6.7}$$

$$G(z^{n,i}) \leq G(z^{n,j})\ if\ and\ only\ if \tag{6.8}$$

$$P(N^n = z^{n,i}) \geq P(N^n = z^{n,j}) \tag{6.9}$$

For each received signal, the GRAND Decoder employs the following algorithm

Algorithm 4 GRAND[40] Algorithm

```
Require: Input Signal $X^n$, Output $Y^n$, Noise $Z^n$, Code-List $C_n$.
1: Initialize $i = 1$.
2: Form the noise pattern $Z^n$ such that $G(Z^n) = i$.
3: while $(X^n = Y^n \ominus Z^n \notin C_n)$ do
4:     $i++$
5:     Set next noise sequence $Z^n$
6:     Return $X^n$
7: end while
```

6.1.3 Tiled Diagonal Zipper Code

As the name implies, Tiled Diagonal Zipper codes[32] are made of tiles formulated into a zipping pair. Tiled Diagonal Zipper is a generalised form of Staircase Zipper codes, when the number of Tile is 1, then Tiled Diagonal becomes a Staircase Zipper Code.

From the structure above, the number of tiles is L, and the size of a tile is ω. For a inner constituent $BCH(N, K)$ code, the $m = N/2 = \omega L$. The size of a tile must be a square and the product of tile size with number of tiles must be equal to half

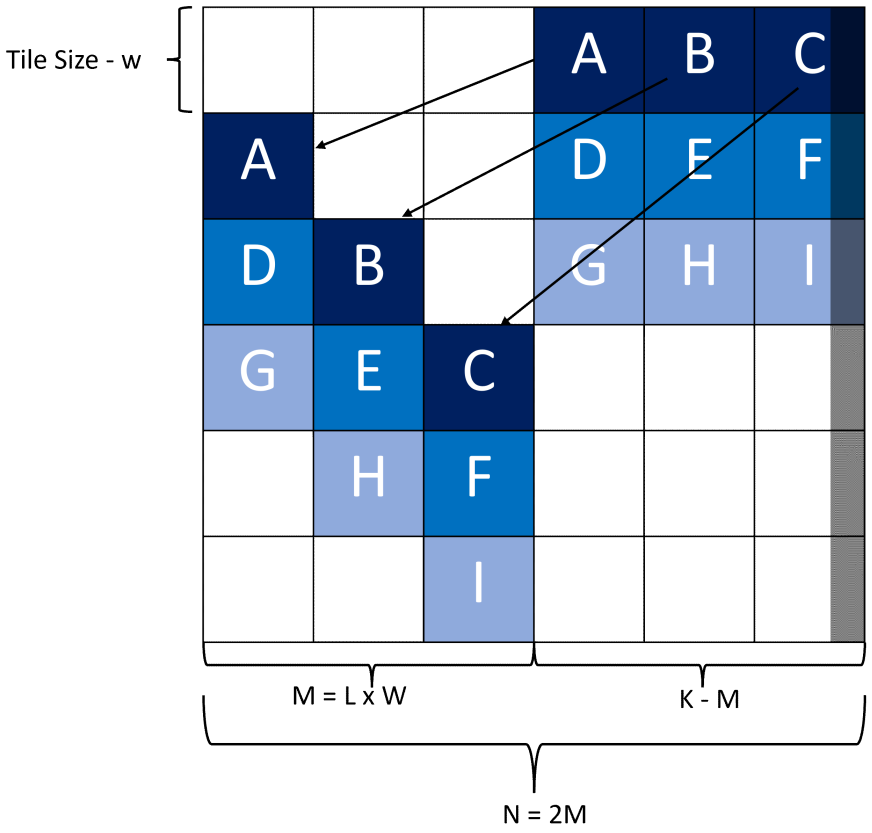

Figure 6.1: Encoding Structure of a Tiled Diagonal Zipper code with number of Tiles, L = 3.

the encoded message size. Now the row and column coordinates of each bit in a tile is given by $\omega q + i$ and $\omega s + j$, for $q \in \mathbb{Z}$, $s \in 2L$ and $i, j \in [\omega]$. Then the sub-matrix $T_{q,s}$ is given by,

$$T_{q,s} = \begin{bmatrix} C_{0,0}^{(q,s)} & \dots & C_{0,\omega-1}^{(q,s)} \\ \dots & \dots & \dots \\ C_{\omega-1,0}^{(q,s)} & \dots & C_{\omega-1,\omega-1}^{(q,s)} \end{bmatrix} \tag{6.10}$$

where $C_{i,j}^{(q,s)}$ is a notation of the bit $\omega q + i$, $\omega s + j$. Like Zipper codes, if the Tile in row a, column b is in the virtual side where $s < m$. And if $s > m$, it is on the real side.

The encoding happens in the following fashion[33].

- The Zipper pair of $BCH(N, K)$ size is initialised for Sliding window capacity in all Zeros.

- Tile size ω and number of tiles L is chosen with relation $\omega L = m = N/2$.
- Random data is added in rows $= \omega$, column $= K - m$.
- Linear Encoding and Parity bits are added similar to Staircase Zipper Codes.
- The tiles are transposed to upcoming virtual side. In increasing ranges of Tile number, each corresponding block is placed at forthcoming rows displaced by tile number.
- Repeat from above.

The rate of the Tiled Diagonal Zipper Code is given by $r = 1 - \frac{2(n-k)}{n}$. As a result, the encoder must store,

$$\omega^2 + 2\omega^2 + 3\omega^2 + ... + L\omega^2 = \frac{L(L+1)\omega^2}{2} \tag{6.11}$$

The Bit Address Formula (BAF) of Tiled Diagonal Zipper Codes is presented. For each error location position *epos*, with the current decoded row indicator *rowid*, the beginning and end of the current block with the current *rowid*, d_{start} and d_{end} respectively, m is half the code-word length. Then the BAF for error in current real and future virtual is,

$$\begin{aligned} \mathrm{row} &= \mathrm{rowid} - \mathrm{d_{start}} + 1 \\ \mathrm{col} &= \mathrm{epos} - \mathrm{m} \end{aligned} \tag{6.12}$$

For tiled diagonal zipper codes, more variables are introduced. Firstly $Tcol$ which calculates the bit beginning from the current tile. $Tile$ is the count of the current tile. W is the size of the tile, then the $Tcol$ can be found by,

$$\mathrm{tcol} = \mathrm{col} - ((\mathrm{tile} - 1) * \mathrm{w})$$

Then the future indices of every bit from every tile of both binary code-word and LLR array can be found by

$$\begin{aligned} \mathrm{vrow} &= \mathrm{tcol} + \mathrm{d_{end}} + (\mathrm{tile} - 1) * \mathrm{w} \\ \mathrm{vcol} &= \mathrm{row} + (\mathrm{tile} - 1) * \mathrm{w} \end{aligned} \tag{6.13}$$

Similarly the BAF for error in current virtual and past real can also be found by

$$\text{row} = \text{rowid} - \text{d}_{\text{start}} + 1 \tag{6.14}$$
$$\text{col} = \text{epos}$$

$$\text{tcol} = \text{col} - ((\text{tile} - 1) * \text{w})$$

$$\text{rrow} = \text{d}_{\text{start}} - (\text{tile} * \text{w}) + \text{tcol} - 1 \tag{6.15}$$
$$\text{rcol} = \text{row} + (\text{tile} - 1) * \text{w} + \text{m}$$

Decoding of Tiled Diagonal Zipper codes is straightforward. A sliding window decoder of tile capacity 5, maximum sliding window iterations 5. Simulation is done in BPSK modulation and AWGN channel for SNRs 5 to 7.5. The tile size $\omega = 32$ and tiles $L = 4$ for BCH(256, 239) constituent code. GRAND Decoder was used to derive results. With Tiled Diagonal Zipper, the window size set is $\omega * 5 = 160$ code-words per window, whereas for Staircase Zipper, the window size was $m * 5 = 640$, as the window capacity affects results, in-spite of which Tiled Diagonal had better than expected performance using GRAND Decoder, the results of which are presented.

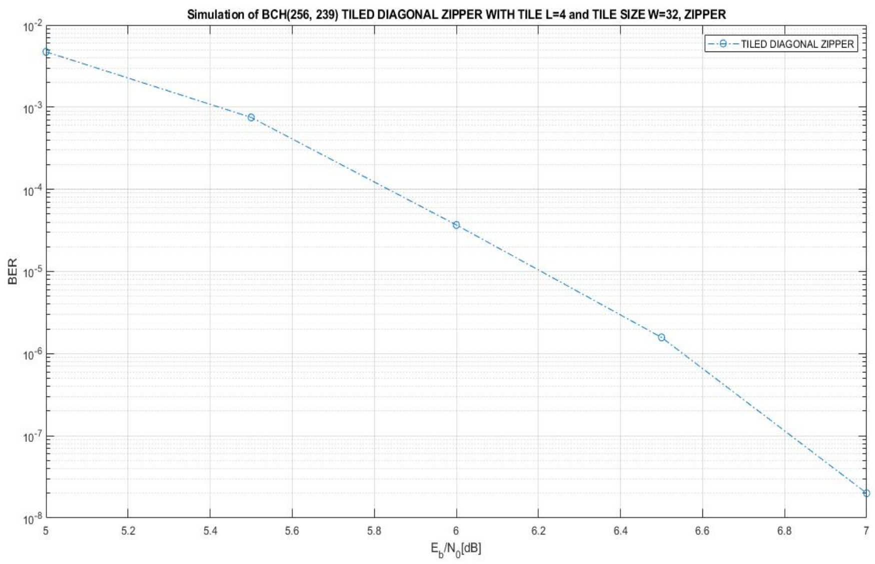

Figure 6.2: BER vs SNR of a Tiled Diagonal Zipper code with number of Tiles, L = 4. tile size $\omega = 32$.

6.1.4 ORBGRAND

Using soft information is known to significantly improve accuracy of the decoder. To this end, Ordered Reliability Bits GRAND has recently been introduced to alter the noise generation component of GRAND algorithm in order to use soft information effectively. Consider $x^n \in \{0,1\}^n$ as the soft output of the channel that is subjected to an additive white Gaussian noise resulting in a random received signal. Let y^n denote the hard decision demodulation. ORBGRAND decodes y^n using x^n to record reliability order of the bits in a vector π^n that contains permutation of $(1, 2, \cdot, n)$. The algorithm generates $e^{n,1}, e^{n,2}, ...$ assuming that the first bit is the least reliable bit, the second bit is the second least reliable bit and so forth. To this end, *Logistic Weight* is defined as

$$\omega_L(e^n) = \sum_{k=1}^{n} k \tag{6.16}$$

For pattern generation, a logistic weight will be assigned to each sequence. Initializing with with $\omega_L(e^n) = 0$, the first noise sequence will be all zeros. The next query for $\omega_L(e^n) = 1$ corresponds to the least reliable bit being flipped. For the next noise sequence only the second least reliable bit is flipped. For the next logistic weight $\omega_L(e^n) = 3$ either the third bit is flipped or both least and second least reliable bits are flipped with the tie broken arbitrarily. This procedure will be continued until $\omega_L(e^n) = \frac{n(n+1)}{2}$. This process defines ORBGRAND algorithm. Therefore, for its operation, ORBGRAND only requires the order of bit reliability of each received sequence that are stored in vector π^n.

The algorithm will begin by sorting all the soft valued likelihood values of the received signal in ascending order and their location is recorded by the system. Now the algorithm will apply logistic weights (LW) to the LLR values which is the sum of the non zero element's indices. The error pattern is generated which will be run against the received signal and passed through the Parity check matrix H to determine if the decoded code-word is part of it. In the event the decoding fails to return a code-word part of H, the system will return the erroneous received signal. The ORBGRAND Algorithm is given below.

Algorithm 5 ORBGRAND[42] Algorithm

```
Require: Input Signal $X^n$, Output $Y^n$, Noise $Z^n$, Parity Matrix $H$
 1: Sort the LLR values of input $X^n$
 2: Record the indices $ind$
 3: Calculate the Logistic Weights $LW$
 4: for $i = 0 : LW_{max}$ do
 5:     Form the error pattern $Z^n$
 6:     Check $H(X^n \oplus Z^n) == 0$
 7:     if Yes then
 8:         Return Output $Y^n$
 9:     end if
10: end for
```

6.1.5 Applying Bounded Distance Decoding

A source of decoding failure of product codes is miscorrection. Consider a component code with error correction capacity t. If there exist more than t errors in a component code, decoder attempts to correct them but more errors are introduced instead. One solution to this problem is to apply bounded distance to decoder. It means that error correction happen only if no more than t errors exist in the component code. Another advantage of applying BDD in GRAND is that it reduces the decoder's complexity by limiting the number of noise sequences.

6.2 Performance Evaluation

The following configuration is considered for our simulations. $BCH(256, 239)$ is set to be the constituent code of Zipper product code, and period of the sliding window is set to 5 block lengths. For each sliding window, 5 iterations are performed. Additive White Gaussian Noise (AWGN) Channel is employed with a BPSK modulation for error simulation. Sound-Noise-Ratio (SNR) values of 1 to 6 were simulated in a series of parallel executions on High Performance Computing Supercomputers of Alliance Canada formerly known as Compute Canada.

All scenarios were simulated above 100,000 errors, 100,000,000 bits until BER stability. First, principal investigator simulated GRAND and ORBGRAND without limitation on number of corrected bits. For making a trade-off between accuracy and computational complexity, This research also proposed a hybrid ORBGRAND-GRAND method which uses ORBGRAND in first 3 iterations and uses GRAND in the second 2 iterations. Then applied bounded distance to GRAND algorithm. To this end, rather than applying a fixed bounded distance, initialized with $\phi = 1$ and rose it in the next iterations. To be more specific, this research used $\phi = 1$ for the first 2 iterations, then incremented it for for the second 1 iterations, and further increased it to $\phi = 3$ in the remaining 1 iterations. Also applied bounded distance to ORBGRAND in the same manner. Finally, applied bounded distance to our hybrid ORBGRAND-GRAND method. To this end, $\phi = 1$ is applied to ORBGRAND in the first 2 iterations. Then, bounded distance is increased to $\phi = 2$ for the next 1 iterations. For the remaining 1 iterations, $\phi = 3$ is applied to the GRAND algorithm. The results of BCH(256, 239) Zipper Code simulated with various decoders in BPSK-AWGN Channel for SNRs 1 to 6 is given.

The sliding window decoder, operates as follows. Initially it will begin with block 1 to block 5. The window capacity defined as 5 blocks can handle $5 * m$ number of rows as specified by the constituent code. When decoding, begins, the code-words in the sliding window are decoded one by one from starting to the last code-word of the window.

When there is an error present in the code-word that can be connected, the algorithm first checks if the erroneous bit is in the real or virtual side. If the error is in real, then the index of that bit along with its corresponding future transposed bits are flipped to correct the error. Similarly, if the error is in the virtual side of the array, then the index of the virtual and the corresponding past real indices are flipped to correct the error.

So error correction happens at two different places at any given time during decoding. Only after the entire window is completed, the sliding window slides from block 1 to block 2, now comprising blocks from 2 to 6, and so on.

The system design of this research is a Hybrid of Soft Input and Hard Output (SIHO) unlike that of a Hard Input Hard Output (HIHO) or a Soft Input Soft Output (SISO)[34][50]. The simulator was designed with two parallel array of code-word data. After applying AWGN to an encoded code-word, the simulator will convert the Log

Likelihood Ratios (LLRs) into binary and one array consists of Binary Code-words in Zipper Arrangement. Another array will have the LLR values of every corresponding binary bit. As ORBGRAND uses LLRs to compute, during iterative decoding, when error correction happens, bits are flipped in current line of code-word and its proportional transposed copy. Likewise, negating the LLR value mathematically achieves the equivalency of a binary bit flip, negating both current and corresponding transpose copy of the LLRs, hence making this a hybrid SIHO system.

6.2.1 BER of Zipper - GRAND

Zipper GRAND was the first combination to be simulated which resulted in a higher performance showing the efficiency of the combination of Product codes with GRAND Sliding Window iterative decoding. Upto SNR 4, see that the Bit Error rate is poor. This is due to simulation criteria, where with lower SNR's the frequency of error injection above the constituent code's correction capacity t, is high. So the decoder is programmed to avoid decoding for such code-words which is known as Burst Errors. At lower SNR's Burst Error injections happen very frequently, hence the Bit error rates tend to be in the ranges of 10^{-1} to 10^{-2}.

Once the SNR crosses 4.5 dB, begin to see improvements in the Bit Error Rates. Obbserved the BER to be $1.083558e-02$ for SNR 4.5dB. Post which see an immediate drop in the rates, which denotes the waterfall of the curve. For SNR's 5.5 and 6 dB, see the error rates fall to $2.312142e - 04$ and $2.009797e - 06$ respectively.

6.2.2 Zipper ORBGRAND

Ordered Reliability Bits Grand Zipper immediately proves to be a powerful competitor to Grand-Zipper. From the results, it is evident that at SNR 5.5dB there is approximately 1.7 dB gain in BER in comparison with Grand Zipper.

6.2.3 Zipper GRAND Bound Distance Decoding

Bound Distance Decoding for GRAND Algorithm applied to Zipper Codes resulted in considerable gain in the bit error rates when compared to Grand-Zipper. There is approximately, a gain of 0.25 dB in comparison, but this fails short of ORBGRAND Zipper's BER.

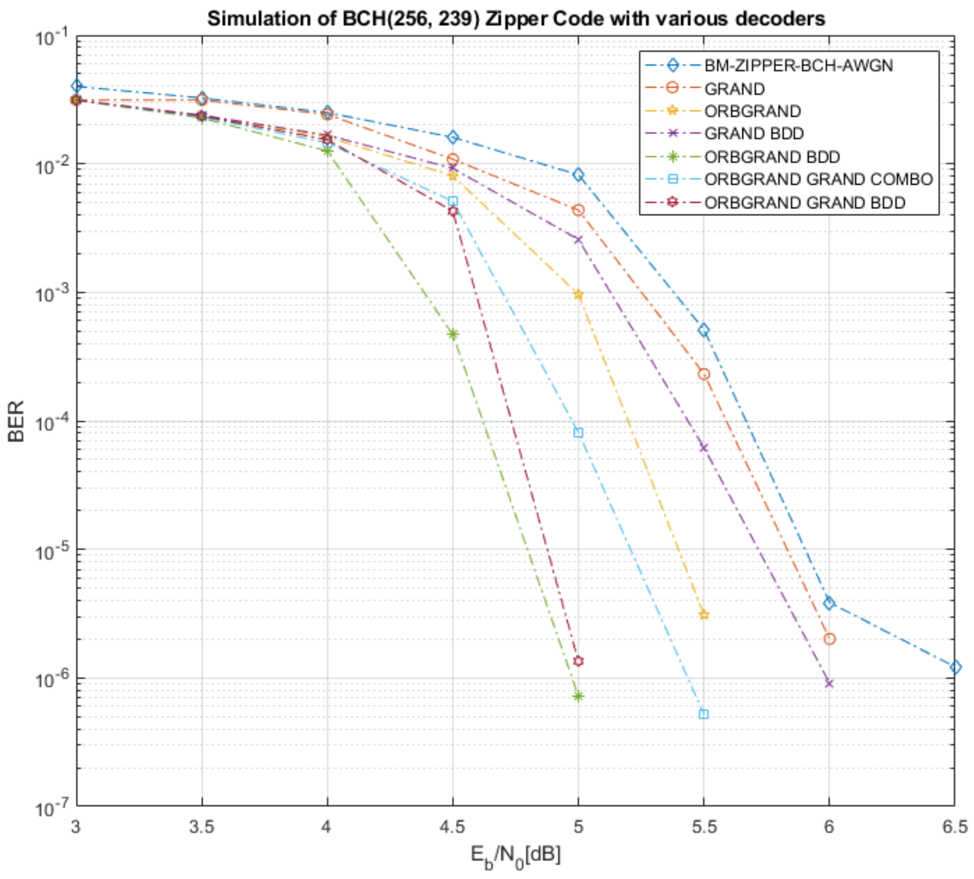

Figure 6.3: BCH(256, 239) Zipper Code with various Decoders

6.2.4 Zipper Dual Decoders - Grand and ORBGRAND

This research proposes a hybrid dual decoding scheme where the Grand Decoder and the ORBGRAND decoder combination is applied to Zipper Codes. This results in a considerable BER boost in comparison to its individual counterparts. In comparison with Grand Zipper, it has approximately 2.5 dB gain at SNR 5 dB. It has above 1.5 dB gain in comparison with GRAND-BDD-Zipper and roughly 0.4dB gain in comparison with ORBGRAND.

6.2.5 Zipper ORBGRAND with Bound Distance Decoding

This particular variant of Bound Distance OrbGrand[49] Decoding strategy proved to result in the most highest BER among its other candidates. At SNR 5, ORBGRAND-BDD-Zipper easily trumps over Grand-Zipper with a staggering 4 dB gain in Bit Error Rates. It went to surpass every combination tested, and proved out to be the most efficient decoding strategy for Zipper Codes. Even in comparison with its own non-BDD counterpart, it easily has 3 dB gain.

6.2.6 Zipper Dual Decoder - BDD-Grand-ORBGRAND

This variant of Zipper code has results close to BDD-ORBGRAND-Zipper. It lags behind the latter by a loss of 0.1-0.2 dB. This can be considered as an alternative to ORBGRAND-BDD when cost of computing becomes critical.

Chapter 7

Conclusion

Communication has been a primary need for human communities since the beginning. Every time in history, there was a new revolutionary means of communication discovered, those communities have made great strides across time. In this digital era of communication transmission systems, unfortunately, we are pressed to transmit digital data through erroneous channels. Since we encode our data and transmit as electric signals, the channel naturally comes with many random error factors such as thermal noise, shot noise, coupling noise through nearby electrical interference human errors etc. Thereby error correction is a crucial field in supporting higher powered transmission systems with high throughput and low latency rates.

Through this research we have demonstrated how different classes of Forward Error Correction Systems work. FEC's are essential to achieve error correction without the need for redundant transmissions. We have demonstrated how Low Density Parity Check codes can achieve an improved Bit Error Rate through the Additive White Gaussian Noise Channel with lesser short cycles preventing the lapse in exchange of intrinsic data across variable nodes thereby increasing its Bit Error Rate.

Product codes, a system with each message symbol being an encoded part of the horizontal and vertical component has better Bit Error rate with increased complexity. Through the use of Staircase Codes implementation, this thesis has demonstrated that Zipper Codes, a form of Spatially-Coupled, Product Like Interleaved Codes has better BER over linear codes. Using the powerful Bose–Chadhuri–Hocquenghem Codes, we have demonstrated that Zipper Codes has the ability to produce a improved Bit Error Rate. The Sliding Window Iterative Decoding Strategy repeatedly[29] decodes the Product Like Codes for extended improvement on the error correction.

We experimented with various methods to eliminate short cycles in the constituent code of Zipper Codes. Simulated Annealing was used to create a specific design matrix, for an efficient mapping of Phi function to improve the performance of Zipper codes. It became evident that with more eliminations of short cycles in the code, the size of the code increases by the factor of the girth. This is applicable for constituent codes of greater size, with a greater memory usage.

Finally this thesis has demonstrated the improvement in decoding rates when focused on the noise patterns of the signal instead of determining the entire code using the Guessing Random Additive Noise Decoding Algorithm. Compared to many FEC systems for Zipper Codes, GRAND decoder has better potential in delivering a better Bit Error Rate with lesser complexity resulting in faster decoding of received code-word. Using the Ordered Reliability Bits strategy for GRAND, the system was able to achieve improved BER compared to GRAND. As the minimum hamming distance between two code-words denoting the minimum change in bits required to distinguish two code-words also determines the error correcting capability of the code, sometimes when the noise is beyond the error correcting capability, the decoding is bound to result in miscorrection. By applying Bound Distance Decoding, the system was proved to improve the BER beyond ORBGRAND. When coupled with Bound Distanced Decoding ORBGRAND and BDD-GRAND, the Zipper System was able to achieve the Highest Bit Error Rate with lesser decoding complexity.

7.0.1 Future Work

This research has some potential places to develop and extend into future work. Some of them are

- LDPC inner constituent code based Zipper Codes. With LDPC's advanced encoding schema and the Sum Product algorithm, the results must be promising.
- Stall Patterns can be minimised. Stall patterns are those errors which failed correction in Sliding window decoding. Those can be minimised by correcting the girth of the code.
- Soft Input Soft Output AWGN Zipper Codes may prove to have better Bit Error Rate[47].
- A hardware based implementation of Zipper Codes has the potential to outperform software simulations with lower memory requirement with BER's potentially going up-to $1e^{-15}$[46][51].

- This research set serialised decoding to avoid code-word conflict when computing syndromes. A parallel set up sliding window decoder might be able to surpass standard serialised setup[52].

Bibliography

[1] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," in IEEE Transactions on Information Theory, vol. 50, no. 12, pp. 2966-2984, Dec. 2004, doi: 10.1109/TIT.2004.838370.

[2] Hamming, Richard W. (1950). "Error detecting and error correcting codes" (PDF). Bell System Technical Journal. 29 (2): 147–160.

[3] Gallager, R. G. (1963) Low Density Parity Check Codes, Cambridge, MA: MIT Press.

[4] T. Mizuochi, "Recent progress in forward error correction and its interplay with transmission impairments," IEEE J. Sel. Topics Quantum Electron., vol. 12, no. 4, pp. 544-554, Jul. 2006.

[5] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," in IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 638-656, Feb 2001, doi: 10.1109/18.910579.

[6] C. A. Kelley and J. L. Walker, "LDPC codes from voltage graphs," 2008 IEEE International Symposium on Information Theory, 2008, pp. 792-796, doi: 10.1109/ISIT.2008.4595095.

[7] Qi, H., Goertz, N. Low-Complexity Encoding of LDPC Codes : A New Algorithm and its Performance.

[8] A. Dutta and A. Pramanik, "Modified approximate lower triangular encoding of LDPC codes," 2015 International Conference on Advances in Computer Engineering and Applications, 2015, pp. 364-369, doi: 10.1109/ICACEA.2015.7164731.

[9] T. Richardson and R. Urbanke, Modern Coding Theory. New York, NY, USA: Cambridge University Press, 2008.

[10] S. Lin and D. J. Costello, Error Control Coding, Second Edition. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2004.

[11] S. Timakul and S. Choomchuay, "Construction of quasi-cyclic LDPC codes form SFT structure and cyclic shift," 2011 International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), 2011, pp. 1-4, doi: 10.1109/ISPACS.2011.6146084.

[12] Johnson, Sarah. (2010). Introducing Low-Density Parity-Check Codes.

[13] L. M. Zhang, D. Truhachev and F. R. Kschischang, "Spatially-coupled split-component codes with bounded-distance component decoding," 2015 IEEE International Symposium on Information Theory (ISIT), 2015, pp. 56-60, doi: 10.1109/ISIT.2015.7282416.

[14] Zeng, Zhibin Feng, Yuan Sun, Xiangran. (2012). An efficient LDPC encoder for CMMB using RU method. Procedia Engineering. 29. 1851-1855. 10.1016/j.proeng.2012.01.225.

[15] J. Justesen, K. J. Larsen and L. A. Pedersen, "Error correcting coding for OTN," in IEEE Communications Magazine, vol. 48, no. 9, pp. 70-75, Sept. 2010, doi: 10.1109/MCOM.2010.5560589.

[16] Hillier, Caleb Balyan, V.. (2019). Error Detection and Correction On-Board Nanosatellites Using Hamming Codes. Journal of Electrical and Computer Engineering. 2019. 1-15. 10.1155/2019/3905094.

[17] Monar, Wilson. [Mathuranathan Viswanathan] SIMULATION OF DIGITAL(BookZZ.Org). www.academia.edu

[18] Kumar, V. Anand, and V. Nandalal. 'A Detailed Study on LDPC Encoding Techniques'. Wireless Communication Technology, vol. 2, no. 2, July 2020. www.ipindexing.com, https://www.ipindexing.com/article/24543.

[19] Nguyen, Tram Thi Bao, et al. 'Efficient QC-LDPC Encoder for 5G New Radio'. Electronics, vol. 8, no. 6, June 2019, p. 668. DOI.org (Crossref), https://doi.org/10.3390/electronics8060668.

[20] Pusane, A. E., et al. 'Deriving Good LDPC Convolutional Codes from LDPC Block Codes'. IEEE Transactions on Information Theory, vol. 57, no. 2, Feb. 2011, pp. 835–57. DOI.org (Crossref), https://doi.org/10.1109/TIT.2010.2095211.

[21] Kukieattikool, Pratana, and Norbert Goertz. 'Variable-Rate Staircase Codes with RS Component Codes for Optical Wireless Transmission: P. Kukieattikool and N. Goertz'. Transactions on Emerging Telecommunications Technologies, vol. 28, no. 4, Apr. 2017, p. e3045. DOI.org (Crossref), https://doi.org/10.1002/ett.3045.

[22] Zhang, Zhengya, et al. 'Design of LDPC Decoders for Improved Low Error Rate Performance: Quantization and Algorithm Choices'. IEEE Transactions on Communications, vol. 57, no. 11, Nov. 2009, pp. 3258–68. DOI.org (Crossref), https://doi.org/10.1109/TCOMM.2009.11.080105.

[23] Usatyuk, Vasiliy, and Ilya Vorobyev. Simulated Annealing Method for Construction of High-Girth QC-LDPC Codes. IEEE, 2018, pp. 1–5. DOI.org (Crossref), https://doi.org/10.1109/TSP.2018.8441303.

[24] Martinez-Mateo, Jesus, et al. 'Improved Construction of Irregular Progressive Edge-Growth Tanner Graphs'. IEEE Communications Letters, vol. 14, no. 12, Dec. 2010, pp. 1155–57. DOI.org (Crossref), https://doi.org/10.1109/LCOMM.2010.101810.101384.

[25] Yige Wang, et al. Construction of High-Girth QC-LDPC Codes. IEEE, 2008, pp. 180–85. DOI.org (Crossref), https://doi.org/10.1109/TURBOCODING.2008.4658694.

[26] Gao, Chaohui, et al. 'Constructing LDPC Codes with Any Desired Girth'. Sensors, vol. 21, no. 6, Mar. 2021, p. 2012. DOI.org (Crossref), https://doi.org/10.3390/s21062012.

[27] Fan, Jun, et al. 'Design LDPC Codes without Cycles of Length 4 and 6'. Research Letters in Communications, vol. 2008, 2008, pp. 1–5. DOI.org (Crossref), https://doi.org/10.1155/2008/354137.

[28] Gu, Jing Fan, Yuzi Zhang, Guohua. (2017). Explicit construction of QC-LDPC codes with girth at least ten from cycle classification. Application Research of Computers. 35.

[29] Y. Jian, H. D. Pfister and K. R. Narayanan, "Approaching capacity at high rates with iterative hard-decision decoding," 2012 IEEE International Symposium on Information Theory Proceedings, 2012, pp. 2696-2700, doi: 10.1109/ISIT.2012.6284009.

[30] Vincent Roca,Christoph Neumann,David Furodet. Low Density Parity Check(LDPC) Staircase and Triangle Forward Error Correction(FEC) Schemes. 2006. inria-00000147v4

[31] A. Y. Sukmadji, U. Martínez-Peñas and F. R. Kschischang, "Zipper Codes: Spatially-Coupled Product-Like Codes with Iterative Algebraic Decoding," 2019 16th Canadian Workshop on Information Theory (CWIT), 2019, pp. 1-6, doi: 10.1109/CWIT.2019.8929906.

[32] A. Y. Sukmadji, U. Martínez-Peñas and F. R. Kschischang, "Zipper Codes," in Journal of Lightwave Technology, 2022, doi: 10.1109/JLT.2022.3193635.

[33] X. Zhao, S. Zhao and X. Ma, "A Class of Tiled Diagonal Zipper Codes With Multiple Chains," in IEEE Transactions on Communications, vol. 70, no. 8, pp. 5004-5017, Aug. 2022, doi: 10.1109/TCOMM.2022.3185065.

[34] Condo, C. (2022, July 25). Iterative soft-input soft-output decoding with ordered reliability bits grand. arXiv.org. Retrieved September 5, 2022, from https://arxiv.org/abs/2207.06691v2

[35] Rao, K.. (2019). Channel Coding Techniques for Wireless Communications. 10.1007/978-981-15-0561-4.

[36] Li, Shizhong El-Sankary, Kamal Karami, Alireza Truhachev, Dmitri. (2019). Area- and Power-Efficient Staircase Encoder Implementation for High-Throughput Fiber-Optical Communications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. PP. 1-5. 10.1109/TVLSI.2019.2950129.

[37] Smith, Benjamin Farhood, Arash Hunt, Andrew Kschischang, Frank Lodge, John. (2012). Staircase codes: FEC for 100 Gb/s OTN. Lightwave Technology, Journal of. 30. 110 - 117. 10.1109/JLT.2011.2175479.

[38] Zhang, Lei Kschischang, Frank. (2014). Staircase Codes With 6% to 33% Overhead. Journal of Lightwave Technology. 32. 1-1. 10.1109/JLT.2014.2316732.

[39] Hurtic, E., Lillmaa, H. (2016). Hard-decision Staircase Decoder in 28-nm Fully-Depleted Silicon-on-Insulator.

[40] K. R. Duffy, J. Li and M. Médard, "Capacity-Achieving Guessing Random Additive Noise Decoding," in IEEE Transactions on Information Theory, vol. 65, no. 7, pp. 4023-4040, July 2019, doi: 10.1109/TIT.2019.2896110.

[41] E. R. Berlekamp, "The enumeration of information symbols in BCH codes", Bell Syst. Tech. J., vol. 46, no. 8, pp. 1861-1880, Oct. 1967.

[42] K. R. Duffy, "Ordered Reliability Bits Guessing Random Additive Noise Decoding," ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 8268-8272, doi: 10.1109/ICASSP39728.2021.9414615.

[43] K. R. Duffy, J. Li and M. Médard, "Guessing noise, not code-words," 2018 IEEE International Symposium on Information Theory (ISIT), 2018, pp. 671-675, doi: 10.1109/ISIT.2018.8437648.

[44] W. An, M. Médard and K. R. Duffy, "Keep the bursts and ditch the interleavers," GLOBECOM 2020 - 2020 IEEE Global Communications Conference, 2020, pp. 1-6, doi: 10.1109/GLOBECOM42002.2020.9322303.

[45] A. Solomon, K. R. Duffy and M. Médard, "Soft Maximum Likelihood Decoding using GRAND," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1-6, doi: 10.1109/ICC40277.2020.9149208.

[46] S. M. Abbas, T. Tonnellier, F. Ercan, M. Jalaleddine and W. J. Gross, "High-Throughput and Energy-Efficient VLSI Architecture for Ordered Reliability Bits GRAND," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 6, pp. 681-693, June 2022, doi: 10.1109/TVLSI.2022.3153605.

[47] K. R. Duffy and M. Médard, "Guessing random additive noise decoding with soft detection symbol reliability information - SGRAND," 2019 IEEE International Symposium on Information Theory (ISIT), 2019, pp. 480-484, doi: 10.1109/ISIT.2019.8849297.

[48] K. Galligan, A. Solomon, A. Riaz, M. Médard, R. T. Yazicigil and K. R. Duffy, "IGRAND: decode any product code," 2021 IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1-6, doi: 10.1109/GLOBECOM46510.2021.9685645.

[49] K. R. Duffy, W. An and M. Médard, "ORDERED RELIABILITY BITS GUESSING RANDOM ADDITIVE NOISE DECODING," in IEEE Transactions on Signal Processing, 2022, doi: 10.1109/TSP.2022.3203251.

[50] C. Condo, V. Bioglio and I. Land, "High-performance low-complexity error pattern generation for ORBGRAND decoding," 2021 IEEE Globecom Workshops (GC Wkshps), 2021, pp. 1-6, doi: 10.1109/GCWkshps52748.2021.9682165.

[51] S. M. Abbas, T. Tonnellier, F. Ercan and W. J. Gross, "High-Throughput VLSI Architecture for GRAND," 2020 IEEE Workshop on Signal Processing Systems (SiPS), 2020, pp. 1-6, doi: 10.1109/SiPS50750.2020.9195254.

[52] A. Solomon, K. R. Duffy and M. Médard, "Managing Noise and Interference Separately - Multiple Access Channel Decoding using Soft GRAND," 2021 IEEE International Symposium on Information Theory (ISIT), 2021, pp. 2602-2607, doi: 10.1109/ISIT45174.2021.9517890.