# ABSTRACT

Title of Thesis: OPTIMIZATION OF PERMUTATION KEY FOR $\pi$-ROTATION LDPC CODES

Nasim Vakili Pourtaklo, Master of Science, 2006

Dissertation directed by: Associate Professor Steven Tretter
Department of Electrical and Computer Engineering

The original low-density parity-check (LDPC) codes were developed by Robert Gallager in early 1960 and are based on a random parity-check matrix construction. In the mid 1990's it was discovered that LDPC codes could be modified slightly to provide the more powerful error correction. These newer LDPC codes, based on an irregular column weight in the underlying check matrix, were still defined with random construction techniques. The $\pi$-rotation LDPC codes discovered by Echard are a family of LDPC codes completely defined by a small set of integers and have several symmetrical features that are exploited to build efficient encoding and decoding designs. The $\pi$-rotation codes can be extended to include irregular matrix patterns to obtain the highest performance. In this dissertation we develop a heuristic algorithm to find the best parity-check matrix for $\pi$-rotation LDPC codes.

# OPTIMIZATION OF PERMUTATION KEY FOR $\pi$-ROTATION LDPC CODE

by

NASIM VAKILI POURTAKLO

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2005

Advisory Commmittee:

Associate Professor Steven Tretter/Advisor
Assistant Professor Adrian Papamarcou
Assistant Professor Nuno Martins

# ACKNOWLEDGMENTS

I would like to express my most sincere appreciation to my advisior, Dr. Steven Tretter. For this research, he not only helped me to shape the whole research but also shared a lot of his findings with me. He has always been a great support for me during the harsh times that I had. I feel very fortunate to have had him as my advisor.

I would like to thank my committee member Dr. Adrian Papamarcou and Dr. Nuno Martins. I benefited a lot from courses that I took with Dr. Papamarcou and he is one of the best teachers I have ever had.

I feel lucky to come to the University of Maryland. A lot of friends and classmates made my life unforgettable and fruitful.

Finally, I want to thank my brothers, Ramin, Amin and Armin; and my parents who always put my interests ahead of theirs; my husband, Behnam, whose love has been unwavering and who has always been the biggest support for me. I love and thank you all!

TABLE OF CONTENTS

LIST OF FIGURES

# Chapter 1

# Introduction

## 1.1 Coding for Digital Transmission

The need for efficient and reliable digital data communication systems has been rising rapidly in recent years. This need has been brought on for a variety of reasons, among them are the increase in automatic data processing equipment and the increased need for long range communication. Communication systems rely on data transmission through channels that link the data source and the data receiver. Examples of channels include the wireless link between mobile radio systems, satellite communication channels, and cable and fiber networks.

Error correcting coding is a key component for efficient and reliable communication. Coding for error correction is the process by which errors introduced during information transmission are corrected by employing prearranged symbol constructions. The symbol construction enables the receiver to correctly interpret the intended transmission from the corrupted version.

In 1948, Claude Shannon [11] introduced the fundamental theorems on communication systems. In order to understand the meaning of this theorem consider Figure 1.1. The source produces binary digits, at some fixed rate R. The encoder is a device that performs data processing, modulation, and anything else that might be necessary to prepare the data for transmission over the channel. We shall assume

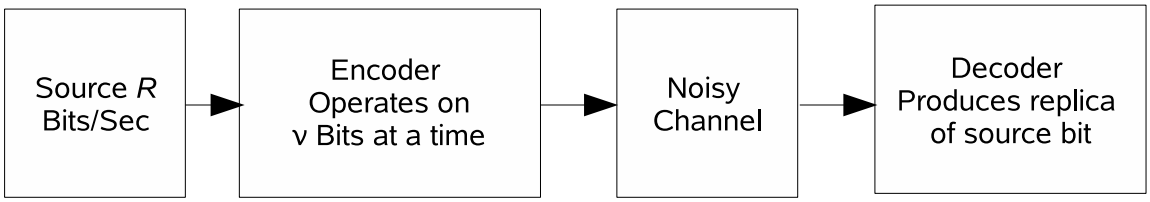



Figure 1.1: Block diagram of communication system.

that encoder separates the source sequence into blocks of $\nu$ bits and operates on only one block at a time. The encoder output is then transmitted over the channel and changed by some sort of random disturbance or noise. The decoder processes the channel output and produces a delayed replica of the source bits. The coding theorem states that for a large variety of channel models, encoders and decoders exist such that the probability of the decoder reproducing a source bit in error $P_e$ is bounded by

$$e^{-\nu[E_L(R_t)+O(\nu)]} \leq P_e \leq e^{-\nu E(R_t)} \tag{1.1}$$

The functions $E(R_t)$ and $E_L(R_t)$ depend upon the channel but not upon $\nu$; they are positive when $R_t = 0$, and decrease with $R_t$ until they become 0 at some rate $C_t$ known as the channel capacity [12]. The exact nature of these functions and the particular class of channels for which this theorem has been proven need not concern us here. The important result is that the coding constraint length $\nu$ is a fundamental parameter of a communication system. If a channel is to be used efficiently, that is with $R_t$ close to $C_t$ then $\nu$ must be made correspondingly large to

achieve a satisfactory error probability.

The standard approach to creating an LDPC code is by randomly generating a very large matrix. While it is well known that for large size the random method will produce very good performance, a method to create the matrix through a more deterministic approach will provide many benefits. A deterministic approach will enable designers to simplify the generation and storing of LDPC codes and allow a less complex description to define the code. The benefit of reduced complexity is the ability to incorporate these high performance code structures into practical products. This will enable their use in systems with smaller amounts of memory, less circuit complexity and more flexibility for dynamic reconfiguration of code structure[3].

## 1.2 History of Error Correcting Codes

After the discovery by Shannon, research in error correction encoding/decoding systems led to the discovery of iterated decoding algorithms. Elias used an iterative approach to decode product codes[6]. Gallager, a student of Elias, introduced Low Density Parity Check codes based on a parity check matrix with a low density of $1's$ and used an iterative method for decoding. In addition, Gallager showed that the LDPC code construction creates a code with minimum distance to length ratio that approaches a nonzero constant with increasing length[8]. In 1967, Viterbi invented convolutional codes. Forney presented concatenated encoders and a method to decode them to reduce the susceptibility of convolutional codes to burst errors[14]. In the early 1990, Berrou and Glavieux discovered turbo-codes[1]. The turbo-coding

system used both the early iterative techniques and concatenated convolutional coding ideas. This was the first practical system that would approach the limiting capacity and was presented by Berrou, Glavieux and Thitimajshima[2]. These new codes retain the original random features in the encoding process but incorporate an iterative decoder in place of an exponential search. MacKay and Neal[9] discovered that turbo-codes can be represented as a type of LDPC code. The top performing codes today are LDPC codes based on a random construction with certain constraints to produce irregular column weight in the parity check matrix.

A deterministic construction to create irregular parity check codes was introduced by Echard[4]. The encoder can create codes of various lengths and rates allowing the system to adapt dynamically to changing conditions in the communication channel and message content. The memory requirement of the encoder is dramatically reduced as the entire code is defined with a handful of integers. These codes are called $\pi$-rotation LDPC codes.

The $\pi$-rotation code is defined around a single permutation matrix. By rotating and repeating the permutation pattern, a larger matrix pattern that becomes a portion of the parity check matrix is formed. The remaining part of the parity check matrix is a dual diagonal matrix.

## 1.3 Outline of Dissertation

In Chapter 2, we will explain the the basics of error correcting codes. The Gallager Low Density Parity Check codes (LDPC) which are based on random

parity check matrices will be explained. These codes are called uniform LDPC codes because there are a fixed number of ones per row and column in the parity check matrix. We will go over message passing decoding or iterative decoding in detail and explain the girth of the matrix and problems which it causes for the decoding procedure.

In Chapter 3, a deterministic method for constructing a non-uniform parity check matrix which was introduced by Echard is explained. The complexity of the code and required memory storage are reduced dramatically. A method to find matrix girth of four is introduced.

To find a permutation matrix or permutation vector with better performance, a heuristic method which is based on search among different key vectors is explained. In this method, all of the permutation vectors or the whole search domain can not be scanned, but a local maximum performance can be obtained.

In the final chapter we will conclude our work and suggest future work.

# Chapter 2

# Error Correcting Codes

When we transmiting the information bits, over a noisy channel, we will observe a corrupted signal at the receiver. The probability of error can be reduced either by increasing the signal power or using coding techniques.

Coding for error control is perhaps described as a mapping between a set of messages created by information source to a set of representations that will replace those messages. The representations are transmitted instead of original message to get lower probability of error. To compare the performance of coding system we will always measure signal as energy expanded per bit of information retrieved and balance this against the noise power present in one hertz of bandwidth. $E_b/N_0$ will represent the signal to noise ratio.

In this chapter, we explain the basic background required to understand, design, encode and decode of LDPC codes based on the random construction technique which was invented by Gallager in the early 1960's[8].

## 2.1 Generator and Parity Check Matrix

The information bits to be encoded are described by a vector of binary digits called the information vector. The encoded message is also a vector of binary digits and has a longer length than the information vector and is called a code-

word. The matrix **G**, called the generator matrix, maps the information vector $\mathbf{u}=[u_1, u_2, ..., u_k]^{'}$ to codeword $\mathbf{v}=[v_1, v_2, ..., v_n]^{'}$. The set of codewords is called the *code* and represents all possible information vectors. There are $2^k$ codeword vectors. The rate of code is defined as the ratio $k/n$. The code, $C$, is defined as the set of all codeword vectors, $\mathbf{v}$, such that $\mathbf{0}=\mathbf{Hv}$ where $\mathbf{H}$ is a $n \times (n-k)$ full rank matrix, called the *parity check matrix*. The rows of $\mathbf{H}$ represent a series of check relations that must be satisfied in order for $\mathbf{v}$ to be a valid code word. For example, let $\mathbf{v}=[v_1, v_2, v_3, v_4, v_5, v_6]^{'}$ and

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

The parity check relations require that $v_1 + v_4 + v_6 = 0$, $v_2 + v_4 + v_5 = 0$ and $v_3 + v_5 + v_6 = 0$. Since $\mathbf{H}$ is full rank the size of the code is $2^3 = 8$.

### 2.1.1 Converting **H** Matrix to **G**

To create LDPC codes from the $\mathbf{H}$ matrix, we must first obtain the generator matrix, $\mathbf{G}$. The code words can be generated by $\mathbf{v}^{'} = \mathbf{u}^{'}\mathbf{G}$ . The following procedure explain this conversion:

- **Step 1** Use Guassian elimination to obtain the identity matrix in the left side of $\mathbf{H}$ matrix :

  $$\mathbf{H} \rightarrow [\mathbf{I}|\mathbf{P}]$$

  the binary matrix $\mathbf{P}$ has dimension $(n-k) \times k$

- **Step 2** The relation between $\mathbf{G}$ and $\mathbf{P}$ is

$$\mathbf{G} = [\mathbf{P}^{'}|\mathbf{I}]$$

where $\mathbf{P}^{'}$ is the transpose of $\mathbf{P}$.

### 2.1.2 Minimum Distance for Linear Codes

The distance between two codewords $\mathbf{x}$ and $\mathbf{y}$, $d(\mathbf{x}, \mathbf{y})$, is defined as the number of places in which $\mathbf{x}$ and $\mathbf{y}$ are different. The weight of vector $\mathbf{x}$, $w(\mathbf{x})$, is the number of non-zero components of $\mathbf{x}$. Therefore we can say, the distance between two binary vector is the weight of binary sum of two vectors.

$$d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} + \mathbf{y}) = \sum_{i=1}^{n}(x_i \bigoplus y_i)$$

The minimum distance for linear code $C$ is the minimum weight of all code words excluding codeword zero[13].

## 2.2 Low Density Parity Check Codes

Low-density parity-check codes are codes specified by a parity check matrix containing mostly $0^{'}$s and relatively few $1^{'}$s. An $(n, j, k)$ low-density code is a code of block length $n$, with a random generated parity check matrix where each column has $j$ $1^{'}$s and each row has $k$ $1^{'}$s , with $k$ and $j$ small relative to the dimensions of $\mathbf{H}$.

### 2.2.1 Decoding LDPC codes

LDPC codes usually have long block lengths. Thus, maximum likelihood decoding can not be used because of the extensive computation required. Gallager introduced an iterative decoding algorithm with complexity of $O(n^2)$ that can perform close to optimum. This algorithm is equivalent to the message passing algorithm[7]. In the following section we explain the decoding procedure in detail.

An LDPC codes can be represented by a Tanner graph. The graph consists of "bit nodes", which are nodes that represent the bits of the codeword, and "check nodes" that express the parity-check relationships. The parity check matrix **H** provides the structure for the graph and decoding algorithm. As shown in Figure 2.1 there is an edge in the graph connecting the bit and check nodes exactly when there is a "1" in the parity check matrix, so that an association can be made between edge-variables and the non zero elements of the parity check matrix. The edge-variables in the same row of the parity-check matrix **H** are connected to the same check node, and so satisfy a parity-check constraint. Meanwhile, the edge-variables in the same column of the parity check matrix **H** are connected to the same bit node.

The graph completely describes all the relations of the code, and can be used for decoding using the message-passing algorithm. Starting with the input consisting of intrinsic [1] probabilities for the bits, the message-passing algorithm uses the parity

[1] For a variable $x$, there are several type of probabilities that can be associate with the variable. For the event $x = a_k$, suppose that $E$ is an event whose effect on variable $x$ is under question. The *prior* (or *a priori* probability refers to the probability $P(x = a_k)$ that the variable $x$ takes the value $a_k$. In the iterative algorithms that we consider, however, the prior probability that is used for variable $x$ may depend on the choice of the event $E$, so that it is more appropriate to speak speak

check relationships amongst the bits to iteratively pass messages between the bit nodes and check nodes to obtain extrinsic probabilities for the bits. The intrinsic and extrinsic probability can then be combined to give posterior probabilities for the codeword bits that incorporate the knowledge gained from the LDPC code[7].

Note, there can be many equivalent parity-check matrices for the same code, and it is preferable to use a parity-check matrix with low density so as to avoid short cycles in the graph. In particular, cycles of length 4, in which two bits are both connected by edges to the same two checks, should especially be avoided.

For a parity-check matrix $\mathbf{H}$ of size $M \times N$, suppose that the bit nodes are $B_i$ for $i = 1, 2, ..., N$, and check nodes are $C_j$ for $j = 1, 2, ..., M$. If an edge connects between the nodes $B_i$ and $C_j$, then that edge is labeled by the variable $e_{ij}$. These are the internal edges of the graph for the LDPC code. In addition, there is also

of the prior probability for *x with respect to E*. To avoid confusion with true prior probability, we substitute the term "*intrinsic*" for "prior" to describe this probability. The *intrinsic probability* for *x with respect to E* is denoted by

$$P^{int}(x = a) = P(x = a).$$

On the other hand, the *posterior* (or *a posteriori*) probability is the conditional probability based on knowledge of the event $E$. The *posterior probability* for *x with respect to E* is denoted

$$P^{post}(x = a) = P(x = a|E).$$

The intrinsic and posterior probabilities represent the probability *before* and *after* taking account the event $E$. The posterior probability can be written using Bayes' theorem as

$$P(x = a|E) = \frac{1}{P(E)} P(E|x = a)P(x = a).$$

The complementary term $P(E|x = a)$ is proportional to the "extrinsic" probability, which is probability that describes the new information for $x$ that has been obtained from the event $E$. The *extrinsic probability for x with respect to E* is defined by

$$P^{ext}(x = a) = \frac{1}{\sum_{a \in A} P(E|x = a)} P(E|x = a).$$

an edge for each bit node $B_i$ that is associated with the variable $v_i$. These edges correspond to the external edges of the graph for the LDPC code. In addition, it will be convenient to introduce a node $N_i$ that is connected to $B_i$ via the edge-variable $v_i$. With this node $N_i$ the intrinsic probability with respect to the LDPC decoder can be denoted as the message $\mu_{N_i \to B_i}(v_i) = P^{int}(v_i)$.

The message from bit node $B_i$ to check node $C_j$ is given by the expressions,

$$\mu_{B_i \to C_i}(e_{ij} = 0) = c_{ji}\mu_{N_i \to B_i}(v_i = 0) \prod_{j' \in M(i) \setminus j} \mu_{C_{j'} \to B_i}(e_{ij} = 0) \tag{2.2}$$

$$\mu_{B_i \to C_i}(e_{ij} = 1) = c_{ji}\mu_{N_i \to B_i}(v_i = 1) \prod_{j' \in M(i) \setminus j} \mu_{C_{j'} \to B_i}(e_{ij} = 1) \tag{2.3}$$

where $M(i)$ is the set of parity checks in which bit $v_i$ is involved. This is also the set of row locations in the $i$-th column of the parity check matrix that contain 1. The set $M(i) \setminus j$ means the set $M(i)$ with the element $j$ omitted. The normalization constant $c_{ij}$ is necessary to make the message into a probability.

The messages from check node $C_j$ to bit node $B_i$ are given by the expressions,

$$\mu_{C_j \to B_i}(x_0 = 0) = \frac{1}{2}\left(1 + \prod_{i' \in L(j) \setminus i} \Big(1 - 2\mu_{B_{i'} \to C_j}(x_{i'} = 1)\Big)\right) \tag{2.4}$$

$$\mu_{C_j \to B_i}(x_0 = 1) = \frac{1}{2}\left(1 - \prod_{i' \in L(j) \setminus i} \Big(1 - 2\mu_{B_{i'} \to C_j}(x_{i'} = 1)\Big)\right) \tag{2.5}$$

where $L(j)$ is the set of bit nodes connected to the $j$-th parity-check, or the set of columns in the $j - th$ row that contain a 1.

These equations describe the message passing algorithm for the low-density parity check codes. We will use the notation used in [9]. Let

$$p_i^b = \mu_{N_i \to B_i}(v_i = b) \tag{2.6}$$

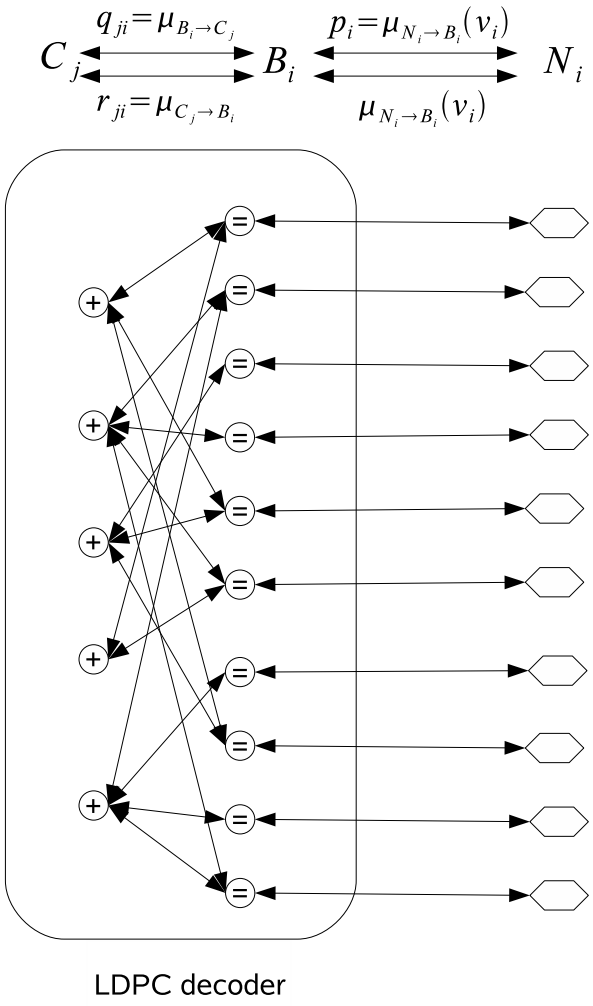



Figure 2.1: Graph for an LDPC decoder

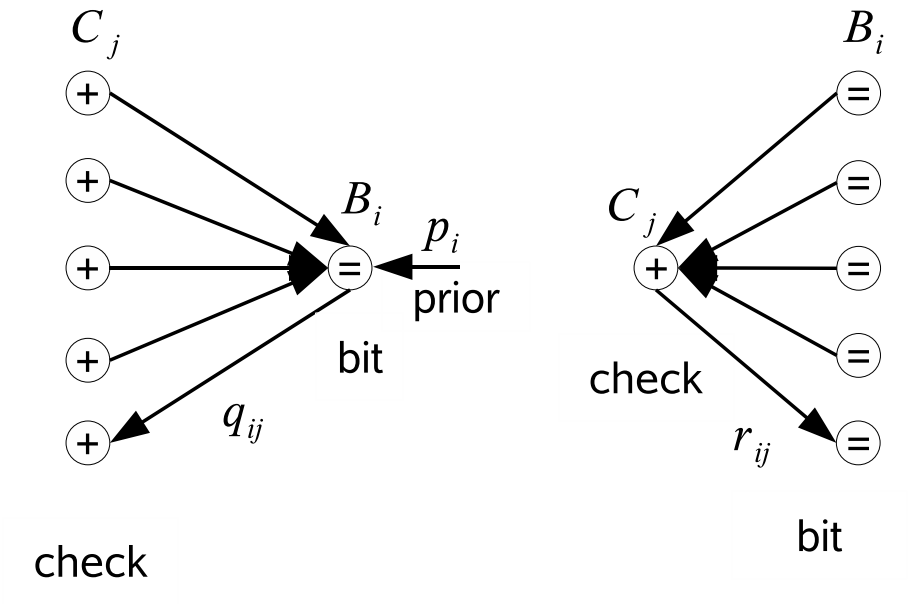



Figure 2.2: Message-passing from bit node to check node, and vice versa

$$q_{ji}^{b} = \mu_{B_i \to C_j}(e_{ji} = b) \tag{2.7}$$

$$r_{ji}^{b} = \mu_{C_j \to B_i}(e_{ji} = b) \tag{2.8}$$

for $b = 0, 1$, as shown in Figure 2.1. The the update equations for the messages from bit node $B_i$ to the check node $C_j$ are

$$q_{ji}^{0} = c_{ji}^{'}.p_i^0 \prod_{j^{'} \in M(i) \backslash j} r_{j'i}^{0} \tag{2.9}$$

$$q_{ji}^{1} = c_{ji}^{'}.p_i^1 \prod_{j^{'} \in M(i) \backslash j} r_{j'i}^{1} \tag{2.10}$$

The message sent from check node $C_j$ to bit node $B_i$ are

$$r_{ji}^{0} = \frac{1}{2}\left(1 + \prod_{j^{'} \in M(i) \backslash j} \delta q_{ji}\right) \tag{2.11}$$

$$r_{ji}^{1} = \frac{1}{2}\left(1 - \prod_{j^{'} \in M(i) \backslash j} \delta q_{ji}\right) \tag{2.12}$$

where $\delta q_{ji} = q_{ji}^{0} - q_{ji}^{1} = 1 - 2q_{ji}^{1}$. The decoding algorithm starts with the intrinsic probabilities $(P_i^0, P_i^1)$, and uniform distribution for $r_{ji}$. Then the relations (2.9)

and (2.10) yield the messages $q_{ji}$ from bits to checks. The messages $q_{ji}$ are then used in (2.11) and (2.12) to calculate the messages $r_{ji}$ from checks to bits.

These two steps comprise a single iteration of the message passing algorithm. The extrinsic proability w.r.t the LDPC decoder $P^{ext}(v_i)$ maybe found by computing the outgoing message for each bit node $v_i$,

$$\mu_{B_i \to N_i}(v_i = 0) = c_i^{'}. \prod_{j \in M(i)} r_{ji}^0 \tag{2.13}$$

$$\mu_{B_i \to N_i}(v_i = 1) = c_i^{'}. \prod_{j \in M(i)} r_{ji}^1 \tag{2.14}$$

where $c_i^{'}$ is a normalizing constant. Finally, the posterior probabilities are

$$q_i^0 = c_i.p_i^0 \prod_{j \in M(i)} r_{ji}^0 \tag{2.15}$$

$$q_i^1 = c_i.p_i^1 \prod_{j \in M(i)} r_{ji}^1 \tag{2.16}$$

It is hoped that after a number of iterations, these estimates of the posterior probabilities $(q_i^0, q_i^1)$ converge to the actual probabilities.

## Decoding Algorithm for LDPC Codes Using LLRs

Since the variables are binary, these computation can also be describe in terms of the log-likelihood ratios (LLRs). The following steps explain the procedure for the LDPC decoder:

- **Step 0, Initialize.** The input to the decoding algorithm is a message vector $(p_i^0, p_i^1)$ for each bit $v_i$, giving the intrinsic probability with respect to the decoder for each bit. As a log-liklihood ratio, this intrinsic information can be

represented as

$$\mathrm{LLR}(p_i) = \mathrm{LLR}^{int}(v_i) = \log \frac{P^{int}(v_i = 1)}{P^{int}(v_i = 0)} \tag{2.17}$$

In addition, the check to bit message vectors start off set to a uniform distribution, so that $(r_{ji}^0, r_{ji}^1) = (\frac{1}{2}, \frac{1}{2})$, and

$$\mathrm{LLR}^0(r_{ji}) = \log \frac{r_{ji}^1}{r_{ji}^1} = 0. \tag{2.18}$$

The iteration number $k$ starts at 1.

- **Step 1, Bit-to-Check messages.** The messages from bit nodes to check nodes are

$$\mathrm{LLR}^{(k)}(q_{ji}) = \sum_{j' \in M(i) \backslash j} \mathrm{LLR}^{(k-1)}(r_{j'i}) + \mathrm{LLR}(p_i). \tag{2.19}$$

- **Step 2. Check-to-Bit messages.** The messages from check nodes to bit nodes are

$$\mathrm{LLR}^k(r_{ji}) = (-1)^{|L(j)|} \left( \prod_{i' \in L(j) \backslash i} \mathrm{sgn}\left(\mathrm{LLR}^{(k)}(q_{ji'})\right) \right) . \Psi \left( \sum_{i' \in L(j) \backslash i} \Psi\left(|\mathrm{LLR}^{(k)}(q_{ji'})\right) \right), \tag{2.20}$$

where $\Psi(x) = -\log\left(\tanh(\frac{x}{2})\right)$

- **Step 3, Compute output.** The output message from the decoder is the extrinsic information,

$$\mathrm{LLR}^{ext}(v_i) = \sum_{j' \in M(i)} \mathrm{LLR}^{(k)}(r_{j'i}), \tag{2.21}$$

and the estimate of the posterior information $\mathrm{LLR}^{post}(v_i)$ is given by

$$\mathrm{LLR}^{(k)}(q_i) = \sum_{j' \in M(i)} \mathrm{LLR}^{(k)}(r_{j'i}) + \mathrm{LLR}(p_i), \tag{2.22}$$

This is used to compute the bit-by-bit estimate of the codeword:

$$\hat{v_i}^{(k)} = \begin{cases} 1 & \text{if } \mathrm{LLR}^{(k)}(q_i) > 0 \\ 0 & \text{if } \mathrm{LLR}^{(k)}(q_i) < 0 \end{cases} \tag{2.23}$$

- **Step 4, Repeat until done.** Check if the stopping condition has been reached (e.g. a fixed number of iterations $k_{max}$ has been reached, or the decision word $\hat{v_i}^{(k)}$satisfies the parity-check matrix $\mathbf{H}$). If not, then increment $k \leftarrow k + 1$, and repeat step 1,2 ,and 3.

For reference the intrinsic LLRs can be found as follows for some common channel models:

- for a binary symmetric channel with crossover probability $p$ and received bit $y_i$,

$$\mathrm{LLR}^{int}(v_i) = \begin{cases} \log \frac{1-p}{p} & \text{if } y_i = 1 \\ \log \frac{p}{1-p} & \text{if } y_i = 0 \end{cases} \tag{2.24}$$

- for a AWGN channel with noise variance $\sigma^2$ and received signal $\mathrm{y}_i$,

$$\mathrm{LLR}^{int}(v_i) = \frac{2}{\sigma^2}\mathbf{y}_i. \tag{2.25}$$

To have a better sense of iterative decoding, we provide a numerical example in the following section [7].

### 2.2.2 Numerical Example of Message Passing Decoder

Consider the following parity check matrix (This example is from [7]):

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

Suppose that the input message is $\mathbf{u} = [\ 1\ 1\ 0\ ]$. Then taking three bits as the systematic message bits, the corresponding encoded codeword is

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

we use 1 and -1 volt to send bit 0 and 1 respectively so the transmitted word is then

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 \end{bmatrix}$$

Suppose that due to noise on an AWGN channel, the received vector is

$$\mathbf{y} = \begin{bmatrix} -\frac{1}{2} & 1 & -1 & -1 & 1 & 0 \end{bmatrix}.$$

The intrinsic probability w.r.t the LDPC decoder is,

$$\mathrm{LLR}^{int} = \frac{2}{\sigma^2}\mathbf{y}_i.$$

Suppose the AWGN channel is known to have a noise parameter of $\sigma^2 = 1$. for this received word $\mathbf{y}$, the input to the LDPC decoder has a log-likelihood ratio of

$$\mathrm{LLR} = \begin{bmatrix} -1 & 2 & -2 & -2 & 2 & 0 \end{bmatrix}$$

corresponding to the probabilities

$$p_i = \begin{bmatrix} 0.268 & 0.881 & 0.119 & 0.119 & 0.881 & 0.5 \end{bmatrix}.$$

the first bit has been corrupted, and the last bit has been erased.

Using the matrix $\mathbf{H}$ and the intrinsic information $\mathrm{LLR}(p_i)$, we can apply the log-domain version of the message-passing algorithm. Since $\mathrm{LLR}^0(r_{ji})$ is initially set to zero, applying (2.19) gives the bit-to-check messages $\mathrm{LLR}^1(q_{ji})$ as $\mathrm{LLR}(p_i)$.

$$\mathrm{LLR}^{(1)}(q_{ji}) = \begin{bmatrix} -1 & 2 & & -2 & & \\ & 2 & -2 & & 2 & \\ -1 & & -2 & & & 0 \end{bmatrix}$$

Notice that the array for $\mathrm{LLR}(q_{ji})$ is left blank when the corresponding entry of matrix $\mathbf{H}$ is zero. The dependencies of the entries are shown in Figure 2.3.

The first check-to-bit message $\mathrm{LLR}^1(r_{ji})$ are computed using (2.20), where each message $\mathrm{LLR}(r_{ji})$ is a function of all the other messages $\mathrm{LLR}(q_{ji'})$ in the same row. For example, the first entry in the first row is given by

$$\begin{aligned} \mathrm{LLR}(r_{11}) &= (-1)2\tanh^{-1}\left(\tanh\left(\frac{1}{2}.2\right)\tanh\left(\frac{1}{2}.-2\right)\right) \\ &= 2\tanh^{-1}(0.762^2) = 1.325. \end{aligned}$$

Similarly, it is possible to compute the rest of the messages from checks to bits.

$$\mathrm{LLR}^{(1)}(r_{ji}) = \begin{bmatrix} -1.325 & -0.735 & & 0.735 & & \\ & 1.325 & -1.325 & & 1.325 & \\ 0 & & -0 & & & -0.735 \end{bmatrix}$$

Then summing up the columns gives the extrinsic output for the first iteration of the decoder, and adding the intrinsic inputs $\mathrm{LLR}(p_i)$ as in (2.22) gives the posterior information as

$$\mathrm{LLR}^{(1)}(q_i) = \begin{bmatrix} 0.325 & 2.590 & -3.325 & -1.265 & 3.325 & -0.735 \end{bmatrix}$$

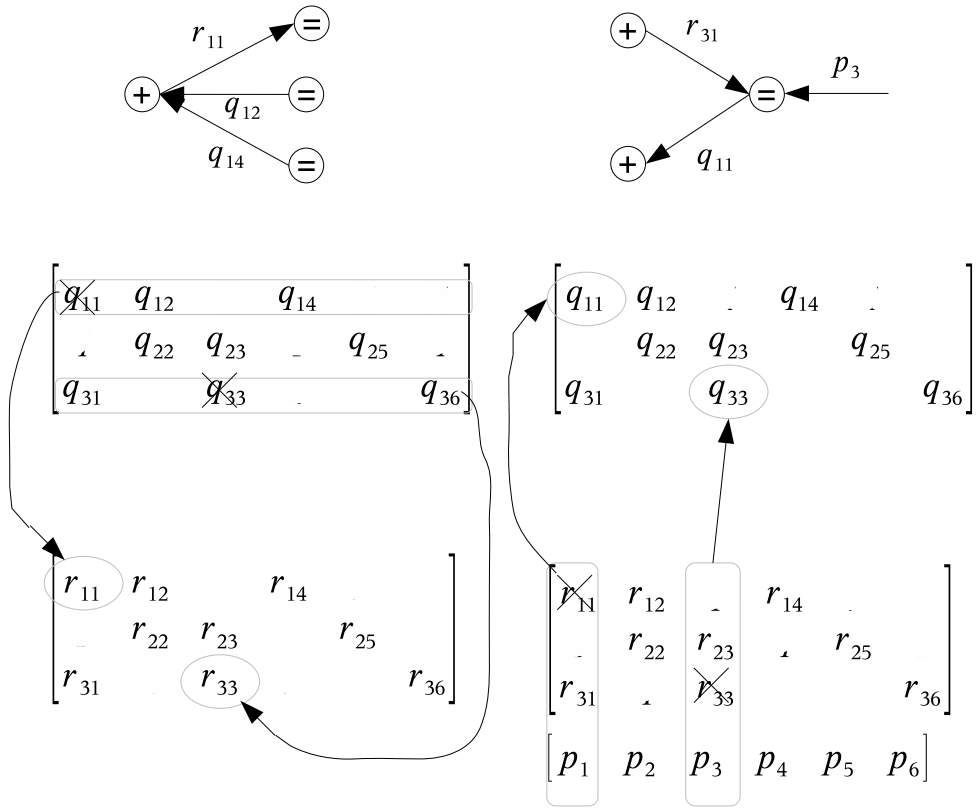



Figure 2.3: Iterating probabilities in the LDPC decoding algorithm (start at the upper left matrix and proceed counter-clockwise

Making hard-decision based on the signs of these LLR as follows,

$$\hat{v_i} = \begin{cases} 1 & \text{if } \mathrm{LLR}(q_i) > 0 \\ 0 & \text{if } \mathrm{LLR}(q_i) < 0 \end{cases}$$

gives an initial estimate of the transmitted codeword. After one iteration of message-passing the result is:

$$\hat{\mathbf{v}}(1) = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Comparing with the original codeword $\mathbf{v} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$, we see that the first bit has been corrected, but the last bit is still in error.

Proceeding with another iteration of the message-passing algorithm, (2.19) computes the message $\mathrm{LLR}^{(2)}(q_{ji})$ by summing all the messages $\mathrm{LLR}^{(1)}(r_{j'i})$ in the same column (except for the one in the j-th row) and also adding the intrinsic information $\mathrm{LLR}(p_i)$. As an example, the first entry $\mathrm{LLR}^{(2)}(q_{11})$ can be found by summing over all $\mathrm{LLR}^{(1)}(r_{j'1})$ in the first column, except for $\mathrm{LLR}^{(1)}(r_{11})$ (since the incoming intrinsic message must be excludes when computing the extrinsic message).

$$\begin{aligned} \mathrm{LLR}^{(2)}(q_{11}) &= \sum_{j' \neq 1} \mathrm{LLR}^{(1)}(r_{j'1}) + \mathrm{LLR}(p_1) \\ &= \mathrm{LLR}^{(1)}(r_{31}) + \mathrm{LLR}(p_1) \\ &= 0 + (-1) = -1 \end{aligned}$$

Continuing this computation, all the messages from bits to checks can be updated as follows:

$$\mathrm{LLR}^{(2)}(q_{ji}) = \begin{bmatrix} -1 & 3.325 & & -2 & \\ & 1.325 & -2 & & 2 \\ 0.325 & & -3.325 & & 0 \end{bmatrix}$$

Note that if the posterior information $\mathrm{LLR}^{(1)}(q_i)$ has already been calculated, then the bit-to-check messages can also be found as

$$\mathrm{LLR}^{(2)}(q_{ji}) = \mathrm{LLR}^{(1)}(q_i) - \mathrm{LLR}^{(1)}(r_{ji})$$

Next, applying (2.20) gives an updated set of check-to-bit messages:

$$\mathrm{LLR}^{(2)}(r_{ji}) = \begin{bmatrix} 1.769 & -0.735 & & 0.920 & & \\ & 1.325 & -0.911 & & 0.911 & \\ 0 & & 0 & & & 0.302 \end{bmatrix}$$

Applying (2.22) for the second time yields

$$\mathrm{LLR}^{(2)}(q_i) = \begin{bmatrix} 0.769 & 2.590 & -2.911 & -1.080 & 2.911 & 0.302 \end{bmatrix}$$

taking hard decisions then gives a correct estimate of the transmitted codeword,

$$\hat{\mathbf{v}}^{(2)} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

This short parity-check code has corrected two error using two iterations of the message-passing algorithm.

## Matrix Girth

The girth of a graph is the number of edges in the shortest cycle. The message-passing algorithm can be shown as a bipartite graph . For each 1 in the parity check matrix , there is an edge between the associated check and variable node. We say that a node has a cycle of four when it is included in a group of nodes that can be connected by two vertical and horizontal paths. In this case, the message passing

algorithm will share the information after two iterations. Similarly, if three vertical and three horizental paths connect a group of nodes we say that the nodes have a six cycle or a girth of six.

In the message-passing decoding, if the messages are statistically independent then the algorithm is known to converge to the maximum likelihood estimate of each symbol. Since at each iteration, messages flow vertically and horizontally, the girth of the node in the matrix determines independency. For example, if the minimum girth of the matrix is 30 then a decoder with the maximum of 15 iterations would pass accurate probabilities. We have to choose the check matrix pattern to avoid the short cycles.

# Chapter 3

## Construction of Low Density Parity Check Codes

In this chapter we present the construction of LDPC codes based on the $\pi$-rotation technique. The parity check matrix is composed of two sub-matrices. One of the sub-matrices is the *dual-diagonal* matrix which makes the structure very adaptable to a simplified encoding scheme and contains enough symmetry to estimate the matrix girth. The other half is a sub-matrix which can be obtained from a random permutation of the identity matrix. In this chapter we explain how to construct a composite parity-check matrix and a method to remove some short cycles.

## 3.1 Composite Parity Check Matrix

The parity check matrix is composed of two sub-matrices $\mathbf{H}^p$ and $\mathbf{H}^d$

$$\mathbf{H} = [\mathbf{H}^p|\mathbf{H}^d] \tag{3.1}$$

The dimensions of $\mathbf{H}^p$ and $\mathbf{H}^d$ are $(n-k) \times (n-k)$ and $(n-k) \times k$ respectively. For a codeword $\mathbf{x}$, the parity check constraint requires $\mathbf{Hx} = \mathbf{0}$. We can decompose $\mathbf{x}$ into sub-vectors $[\mathbf{x}^p, \mathbf{x}^d]$ where $\mathbf{x}^p$ is the parity vector and $\mathbf{x}^d$ is the information vector. So we have:

$$\mathbf{H}^p\mathbf{x}^p = \mathbf{H}^d\mathbf{x}^d = \mathbf{v} \tag{3.2}$$

We call $\mathbf{v}$ the *projection vector* and $\mathbf{H}^p$ is a *dual-diagonal* matrix defined by

$$\mathbf{H}^p = \mathbf{I} + \mathbf{D} \tag{3.3}$$

where $\mathbf{I}$ is an identity matrix and $\mathbf{D}$ is obtained from the identity matrix by removing the first row and appending a bottom row of zeros. Here is an example of the $\mathbf{H}^p$ matrix:

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

The inverse of $\mathbf{H}^p$ is an upper triangular matrix, $\mathbf{U}^p$, and thus:

$$\mathbf{x}^p = \mathbf{U}^p \mathbf{v} \tag{3.4}$$

So we can simply generate codewords from information vectors using the following two steps:

- **Step 1** Compute $\mathbf{v}$ from $\mathbf{x}^p$ by (3.2).
- **Step 2** Using (3.2), we start from the bottom of column vector $\mathbf{v}$, the bottom bit of $\mathbf{v}$ is equal to the last bit of $\mathbf{x}^p$ and we can calculate each bit of $\mathbf{x}^p$ as

$$\mathbf{x}^p(n) = \mathbf{v}(n) + \mathbf{x}^p(n-1) \tag{3.5}$$

We can easily obtain the parity vector from projection vector by passing it through a shift register as shown in Figure 3.1.

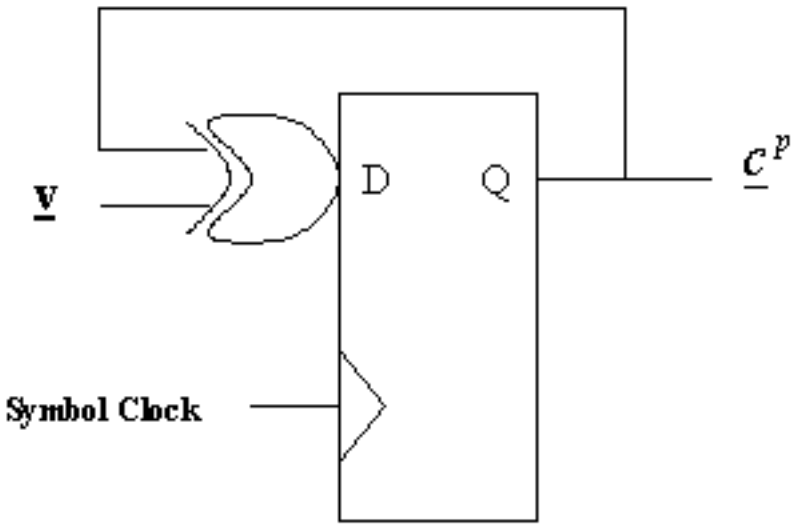



Figure 3.1: The circuit to produce parity bits from projected vector.

## 3.2 The $\mathbf{H}^d$ Sub-Matrix

The basic idea for constructing the $\mathbf{H}^d$ matrix is to use $m \times m$ random permutation matrices. A random permutation matrix is simply the identity matrix with randomly permuted rows. If $\mathbf{H}^d$ is a $q \times t$ array of random permutation matrices, then we have $t$ ones per row and $q$ ones per column and size of $\mathbf{H}$ would be $qm \times (t+q)m$ and the rate of code would be $q/(q+t)$. For example, a code with the following $\mathbf{H}^d$ matrix has rate of $\frac{2}{5}$. $\mathbf{H}^d$ has three ones per column and two ones per row [5].

$$\mathbf{H}^d = \begin{bmatrix} \pi_1 & \pi_2 \\ \pi_3 & \pi_4 \\ \pi_5 & \pi_6 \end{bmatrix} \tag{3.6}$$

### 3.2.1 Method of Constructing $\pi$-Rotation Matrix

We can start with one permutation matrix, $\pi_A$, and find the other permutation matrices by rotating $\pi_A$ and construct the $\mathbf{H}^d$ matrix as explained in Figure 3.2.

### 3.2.2 The $\pi$-Rotation Vector

Now, we will show how to create a $\mathbf{H}^d$ sub-matrix just from a single permutation vector. This vector indicates the positions of the non-zero elements in each column, counting from the bottom. For example, for the permutation vector of [ 1 3 2 ] we obtain the following four $\pi$-rotations:

$$\pi_A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \qquad \pi_D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\pi_B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \pi_C = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The resulting parity check matrix is shown below:

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

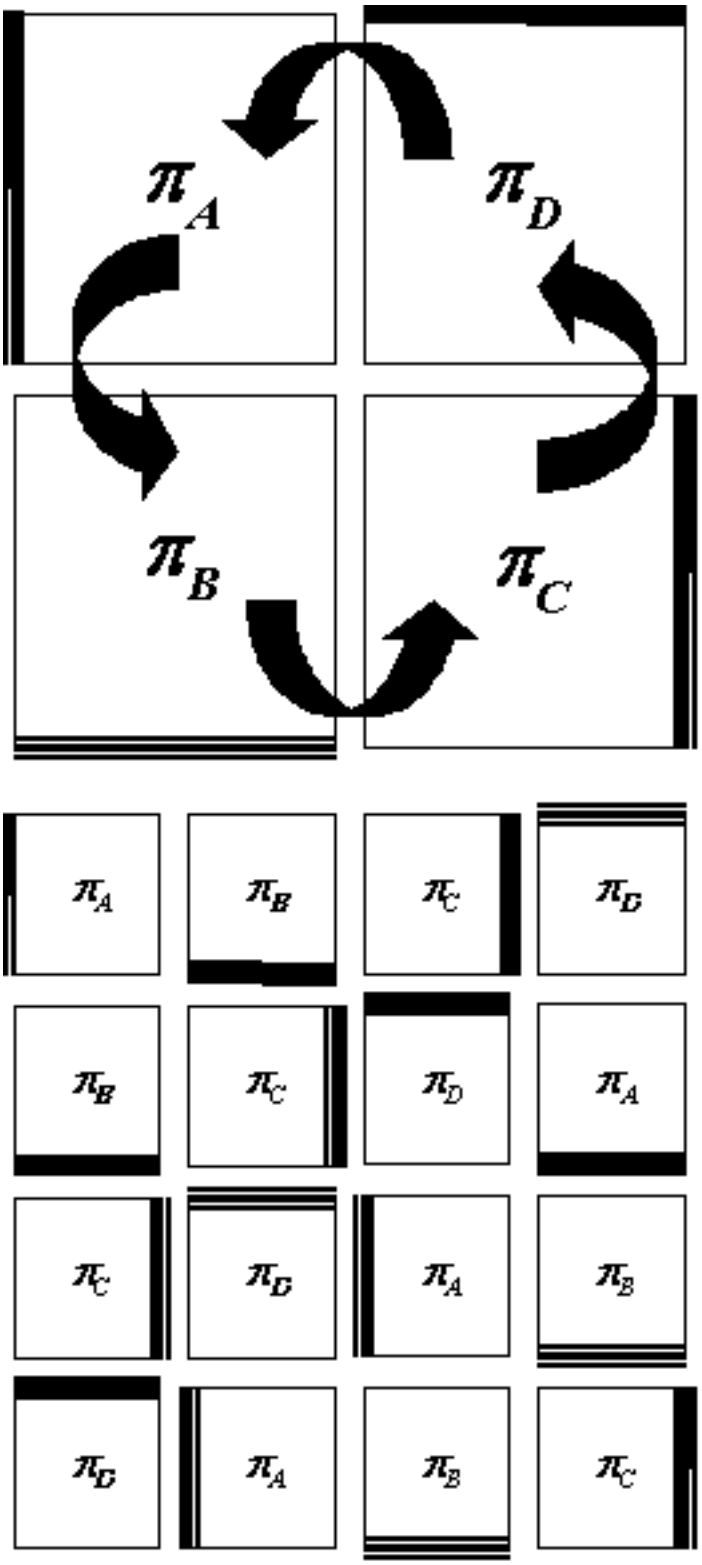



Figure 3.2: The upper figure shows the four $\pi$-rotation matrices and the lower figure shows the arrangement of four $\pi$-rotation permutation matrices to create the $\mathbf{H}^d$ matrix for a rate $\frac{1}{2}$ code.

In general, we can create the rate $\frac{1}{2}$ parity check matrix from any permutation vector as follow:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & \cdots & 0 & \pi_A & \pi_B & \pi_C & \pi_D \\ 0 & 1 & \cdots & 0 & \pi_B & \pi_A & \pi_B & \pi_A \\ \vdots & \ddots & \cdots & \vdots & \pi_C & \pi_C & \pi_A & \pi_B \\ 0 & \ldots & 1 & 1 & \pi_D & \pi_D & \pi_D & \pi_C \end{bmatrix} \tag{3.7}$$

## 3.3 Finding a Good Permutation

In this section we study how to select a good permutation based on distance and girth properties. Under the general conditions the search would take a great deal of processing time. In the next sub-sections we use the symmetries of matrix (3.7) to facilitate search techniques.

### 3.3.1 Counting The Short Loops

Figure 3.3 shows a simple representation of the $\mathbf{H}^d$ matrix. We search for 4-cycle loops. Particularly, short loops exist between quad arrangements of permutations. In Figure 3.4, there are 36 quads for a 4 by 4 pattern. Consider first the top-left and bottom-right quad. We can see they have the same permutation pattern **A-B-C-B**. However, the quad represented by **B-C-B-A** has the same number of short loops (cycle of length 4) as **A-B-C-B**.

Figure 3.5 shows the the list of quads and where this quad and one of its

| | | | |
|---|---|---|---|
| A | B | C | D |
| B | C | D | A |
| C | D | A | B |
| D | A | B | C |

Figure 3.3: The basic configuration of $\mathbf{H}^d$ matrix.

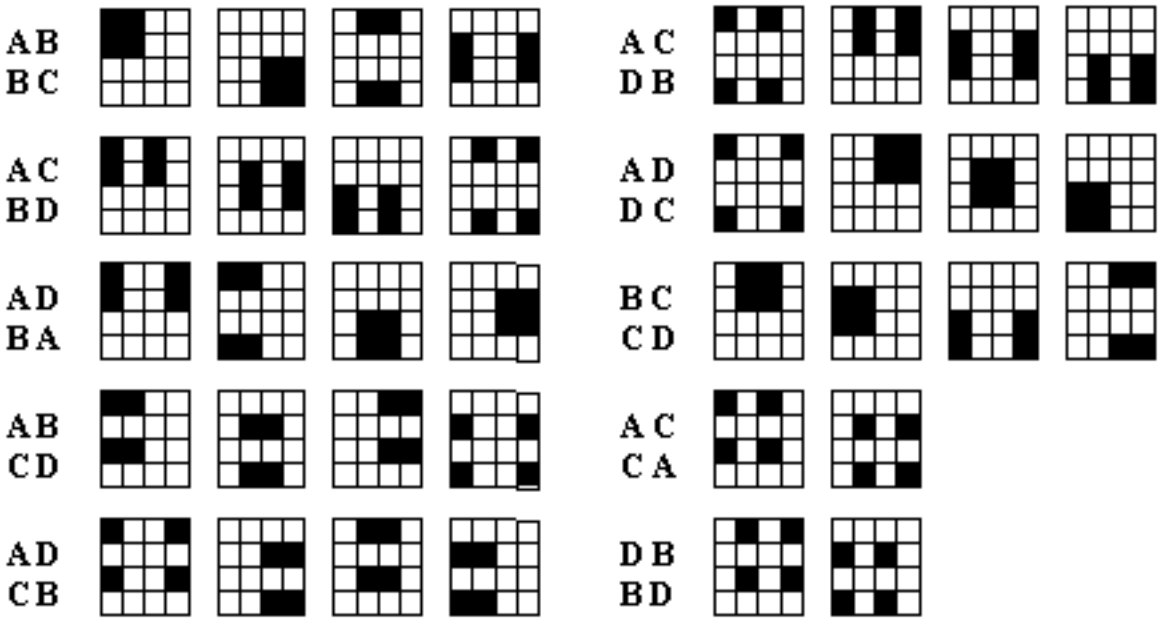



Figure 3.4: There are ten quads which cover all possible short loops within the $\mathbf{H}^d$ matrix.

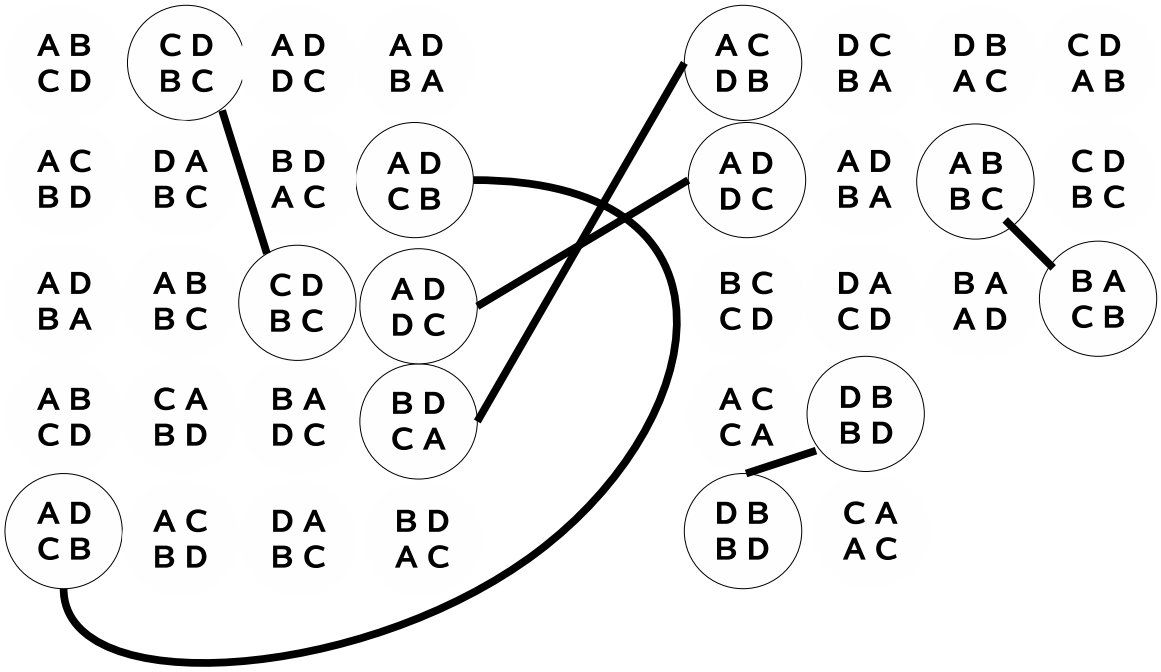



Figure 3.5: The quads that are invariant through rotation. The number of independent quads is reduced to four

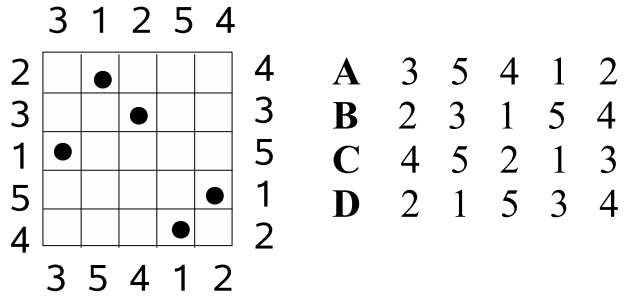
invariants reside in the permutation grid. Moreover, additional invariants are observed when the matrices are rotations of one another as they are in the $\pi$-rotation matrix, for example, with rotating **A-B-C-B** once counterclockwise to obtain the **C-D-C-B** quad which has the same number of short loops. In Figure 3.4, the four possible quad rotation for each quad are shown [5].

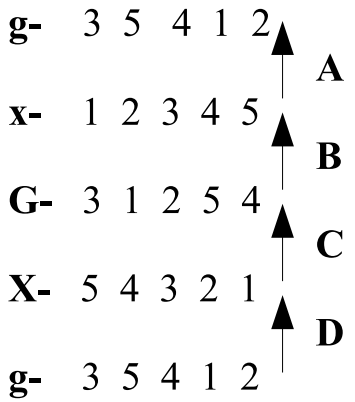
There are cases where rotating the quad produces a quad from another group, as we can see in Figure 3.5. The four base quads are chosen to be **A-B-C-B**, **A-C-D-B**, **A-B-D-C** and **A-C-A-C**. We can count the repetition of each quad type and find the multiplicities of 16, 8, 8, and 4 respectively. Thus the total number of short loops is given by:

$$\textbf{LoopCount} = 16Q_{ABCB} + 8Q_{ACDB} + 8Q_{ABDC} + 4Q_{ACAC} \tag{3.8}$$

where $Q$ is the short loop count in the associated quad.

Now, we would like to express the short loop count of a quad directly from the premutation vector $\pi_A$ . For this purpose define $\mathbf{x}$ as the $m$ length integer list starting from one. Three other lists, $\mathbf{g}$, $\mathbf{X}$ and $\mathbf{G}$ are derived from the $\pi_A$ permutation. The $\mathbf{g}$ list is the $\pi_A$ permutation mapping itself, the $\mathbf{X}$ list is derived from $\mathbf{x}$, where $\mathbf{x}$ is a vector with $x_i = i$, by transformation $\mathbf{X} = m + 1 - \mathbf{x}$ and in a similar way, $\mathbf{G}$ can be obtained from $\mathbf{g}$ with the same transformation. Thus the $\pi_A$ vector is obtained by mapping $\mathbf{x}$ to $\mathbf{g}$, the $\pi_B$ vector is obtained by mapping $\mathbf{G}$ to $\mathbf{x}$, the $\pi_C$ vector is obtained from by mapping from $\mathbf{X}$ to $\mathbf{G}$ and the $\pi_D$ vector is obtained from $\mathbf{g}$ to $\mathbf{X}$. Searching for short loops in a particular quad is simply a matter of tracking these various mappings and identifying any matches [5].




**A B**
**B C**

Begin with a 'x' value, find g(x)

Find G(g(x))

The values x and G(g(x)) are the locations in the 'A' and 'B' vectors that are equal.

If x=G(G(g(g(x)))) then there is a short loop in the **A-B-C-B** quad.

Figure 3.6: The algebraic relation which is developed to find a short loop in the **A-B-C-B** quad.

Figure 3.6 explains how we find the relations for short loops in the **A-B-C-B** quad. The mapping $\mathbf{g(x)}$ determines the nonzero locations in permutation $\pi_A$. $\mathbf{G(g(x))}$ gives the horizontal location of the nonzero values in $\pi_B$. Thus $\mathbf{x}$ and $\mathbf{G(g(x))}$ are the column locations having the same nonzero row numbers. If matrices $\mathbf{B}$ and $\mathbf{C}$ have identical nonzero row numbers then we have a short loop. Refering to our relationships, we find that for the $\mathbf{B}$ rotation the row is identified with $\mathbf{G^{-1}(x)}$ and for $\mathbf{C}$, $\mathbf{G(X^{-1}(G(g(x))))}$ delivers the row number (counting from the bottom). We note that $\mathbf{X^{-1}G = g}$ to simplify the relation as follows:

$$x = \mathbf{G(G(g(g(x))))} \tag{3.9}$$

To find number of short loops in the **A-B-C-B** quad, we have to try equation (3.9) for each value of $x$, from 1 to $m$ , and if $x$ satisfies (3.9) we count one short loop in **A-B-C-B**. The relation for each of the basis quads are as follows:

$\mathbf{A\text{-}B\text{-}C\text{-}B \ldots = G(G(g(g(x))))}$

$\mathbf{A\text{-}C\text{-}D\text{-}B \ldots = G(X(G^{-1}(G^{-1}(g(x)))))}$

$\mathbf{A\text{-}B\text{-}D\text{-}C \ldots = X(g^{-1}(g^{-1}(G(g(x)))))}$

$\mathbf{A\text{-}C\text{-}A\text{-}C \ldots = X^{-1}(G(g(X(G^{-1}(g(x))))))}$

We can simply search for each value of $x$ in the above equations. The number of solutions obtained for each quad is placed in the appropriate $Q$ variable in (3.8) to determine the number of short loops in the $\mathbf{H}^d$ sub-matrix. The problem with this method is that it does not identify ALL possible short loops in the parity check matrix [5].

# Chapter 4

# A Heuristic Method to Find The Best Permutation Vector

As explained in Chapter 3, there are some permutation vectors which cause short cycles in the parity check matrix. To avoid this problem, Echard proposed a method to find short cycles of length four. This method does not cover all possible matrix girths of length four. Moreover, the extension of this method to find bigger matrix girths, say six or eight, is very difficult.

Suppose the size of the $\mathbf{H}^d$ sub-matrix is $4m \times 4m$ where $m$ is the size of permutation vector then there are $m!$ possible permutation vectors. Finding the best permutation vector among $m!$ possible vectors is a NP complete problem. When m increases, the size of search domain increases exponentially. Thus we have to use a heuristic algorithm to find a local optimum permutation vector.

## 4.1 Performance of Different Permutation Vectors

In this section, the difference in performance among permutation vectors will be discussed. We considered AWGN channel, in which additive white Guassian noise is added to the transmitted signal. The received signal $\mathbf{y}_i$ is given by:

$$\mathbf{y}_i = \mathbf{x}_i + \mathbf{n}_i \tag{4.1}$$

which consists of the transmitted signal $\mathbf{x}_i$ with additive noise $\mathbf{n}_i$ chosen from a zero-mean Gaussian distribution:

$$P_n(a) = \frac{1}{\sqrt{2\pi\sigma}} exp(-\frac{a^2}{2\sigma^2}) \tag{4.2}$$

where $\sigma$ is the standard deviation of the noise. The transmitted signal $\mathbf{x}_i$ is often restricted to a discrete signaling constellation, such as pulse amplitude modulation (PAM). The encoded bits $v_i \in (0, 1)$ are mapped to the constellation points $\mathbf{x}_i \in (-1, 1)$ for transmission, using the mapping $\mathbf{x}_i = 2v_i - 1$. The quality of channel is measured in terms of the signal to noise ratio (SNR), which is defined here as

$$\frac{E_b}{N_0} = \frac{P}{2R\sigma^2} \tag{4.3}$$

which represents the energy normalized per used bit. The power is assumed to be $P = 1$, the code rate $R$, and the noise variance $\sigma^2$. The signal to noise ratio is usually measured in decibels (dB), so that the SNR in dB is equal to $10log_{10}(\frac{E_b}{N_0})$ dB. In the simulation program, with given SNR and $R$ we can find $\sigma$, then we produce two random variable for information bits and noise. The received signal will be passed through a message passing-decoder. We run the simulation for almost $10^9$ bits.

We start with a small codeword length, for example, for codewords of length 24 and 32 , so there are $3! = 6$ and $4! = 24$ different possible parity check matrices. The performance of different permutation vectors can be seen in Figure 4.1.

The permutation vectors of [ 1 2 3 ] and [ 3 2 1 ] have higher BER compared to the other permutation vectors. Due to symmetry, the number of short cycles in these permutations is higher. The best performance among permutation vectors of size four belongs to [ 4 2 1 3 ], [ 4 1 3 2 ], [ 3 2 4 1 ], [ 3 1 2 4 ],[ 2 4 3 1 ], [ 2 3 1 4 ],

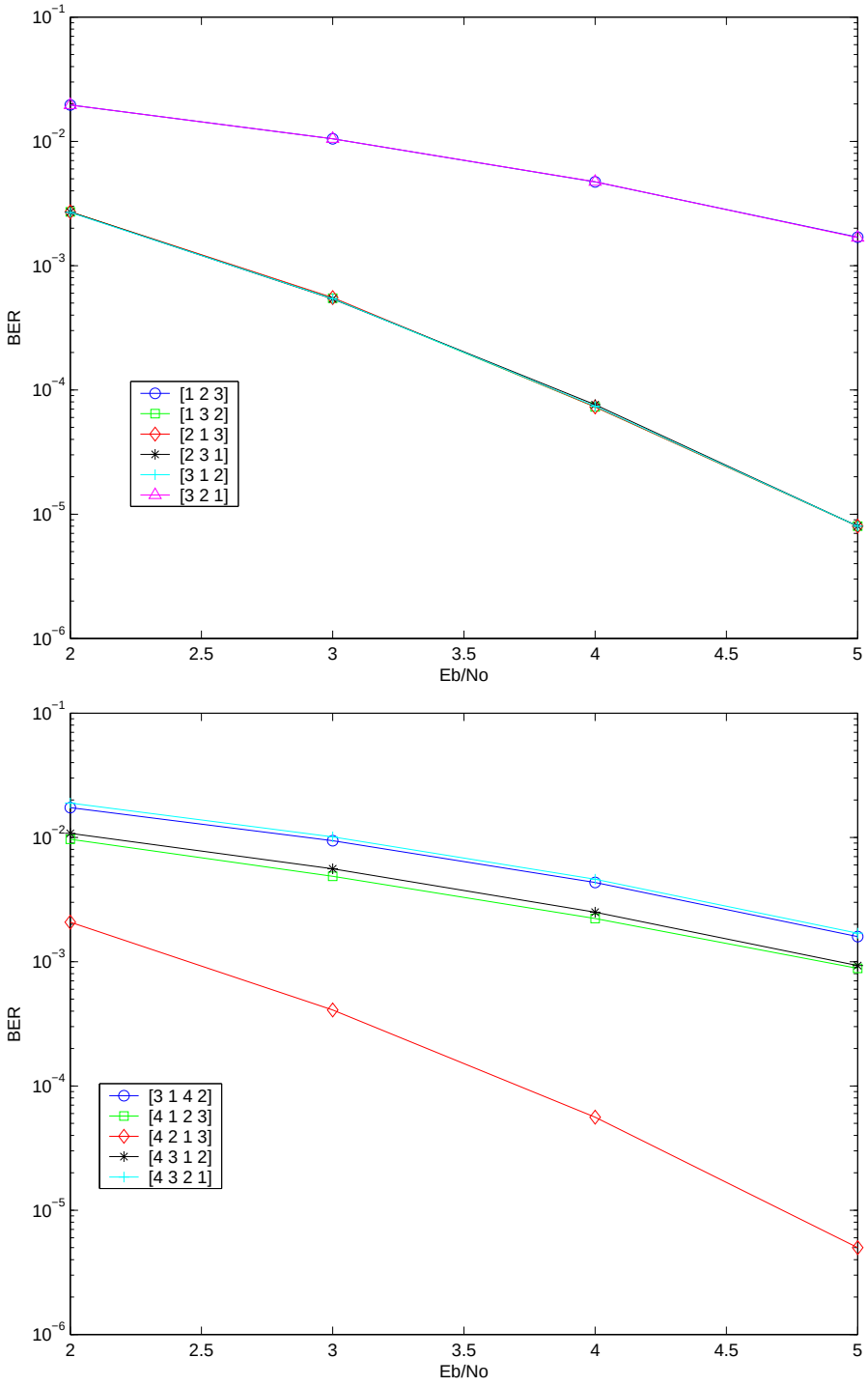



Figure 4.1: The BER for different permutation vectors of size 3 and 4.

[ 1 4 2 3 ] and [ 1 3 4 2 ] ( In Figure 4.1, just 5 different permutation vectors from the 24 permutations are shown.) When the size of the matrix increases we can not run the simulation for all different permutations, so we need another algorithm to find the best local one.

## 4.2 Searching for a Good Permutation Vector

As discussed before, we can not examine the performance of each permutation vector, because our search domain contains $m!$ different vectors. In this section, we explain a suboptimal algorithm in detail and give some examples to show how it works.

**Definition 1** *Mapping $g(i,j)$ on vector $\underline{\mathbf{a}} = [a_1, a_2, \ldots, a_n]$ for $0 \leq i < j < n$ is :*

$$\underline{\mathbf{a}} = [a_1, \ldots, {}^{i}a_{i+1}, \ldots, {}^{j}a_{j+1}, \ldots, a_n] \longrightarrow \underline{\mathbf{a}} = [a_{j+1}, \ldots, a_n, a_{i+1}, \ldots, a_j, a_1, \ldots, a_i]$$

For example, if $i = 3$ and $j = 7$ mapping $g(3,7)$ on vector $\underline{\mathbf{a}}$=[ 1 2 3 4 5 6 7 8 9 ] is [ 8 9 4 5 6 7 1 2 3 ] and for $i = 0$ and $j = 7$ is [ 8 9 1 2 3 4 5 6 7 ].

**Definition 2** *The neighborhood of a vector $\underline{\mathbf{a}} = [a_1, a_2, \ldots, a_n]$ is the set of vectors which can be obtained from $\underline{\mathbf{a}}$ by repositioning the elements of $\underline{\mathbf{a}}$ using mapping $g(i,j)$ for $\forall \quad 0 \leq i < j < n$.*

For example, the neighborhood of vector [ 1 2 3 4 ] is:

$i = 0, j = 1 \qquad [2 \quad 3 \quad 4 \quad 1] \qquad i = 1, j = 2 \qquad [3 \quad 4 \quad 2 \quad 1]$

$i = 0, j = 2 \qquad [3 \quad 4 \quad 1 \quad 2] \qquad i = 1, j = 3 \qquad [4 \quad 2 \quad 3 \quad 1]$

$$i = 0, j = 3 \qquad [4 \quad 1 \quad 2 \quad 3] \qquad i = 2, j = 3 \qquad [4 \quad 3 \quad 1 \quad 2]$$

Thus, each vector has 6 other vectors in its neighborhood. In general, a vector of length $n$ has $C_n^k = \frac{n(n-1)}{2}$ vectors in its neighborhood.

**Proposition 1** *If* $\underline{\boldsymbol{a}}$ *is in the neighborhood of* $\underline{\boldsymbol{b}}$ *then* $\underline{\boldsymbol{b}}$ *is in* $\underline{\boldsymbol{a}}$*'s neighborhood as well.*

We can show this as a connected graph. Each node represents a permutation vector and each edge connects two nodes that can be obtained from each other. Figure 4.2 shows a connected graph for a permutation vector of size four. this graph has 24 nodes and 72 edges.

**Proposition 2** *All permutation vectors can be obtained from each permutation with positive probability.*

We propose the following procedure to find locally the best performance among all possible permutation vectors.

- **Step 0** Generate a random permutation vector, $v$. Compute $\text{BER}_{int}$ and choose $state0 = v$ and $state1 = v$.
- **Step 1** Generate random numbers $0 \leq i < j < n$ and use mapping $g(i, j)$ on $v$ to obtain another vector and compute $\text{BER}_1$
- **Step 2** If $\text{BER}_1 < \text{BER}_{int}$ then

$$state0 = v \qquad state1 = v \qquad \text{BER}_{int} = \text{BER}_1$$

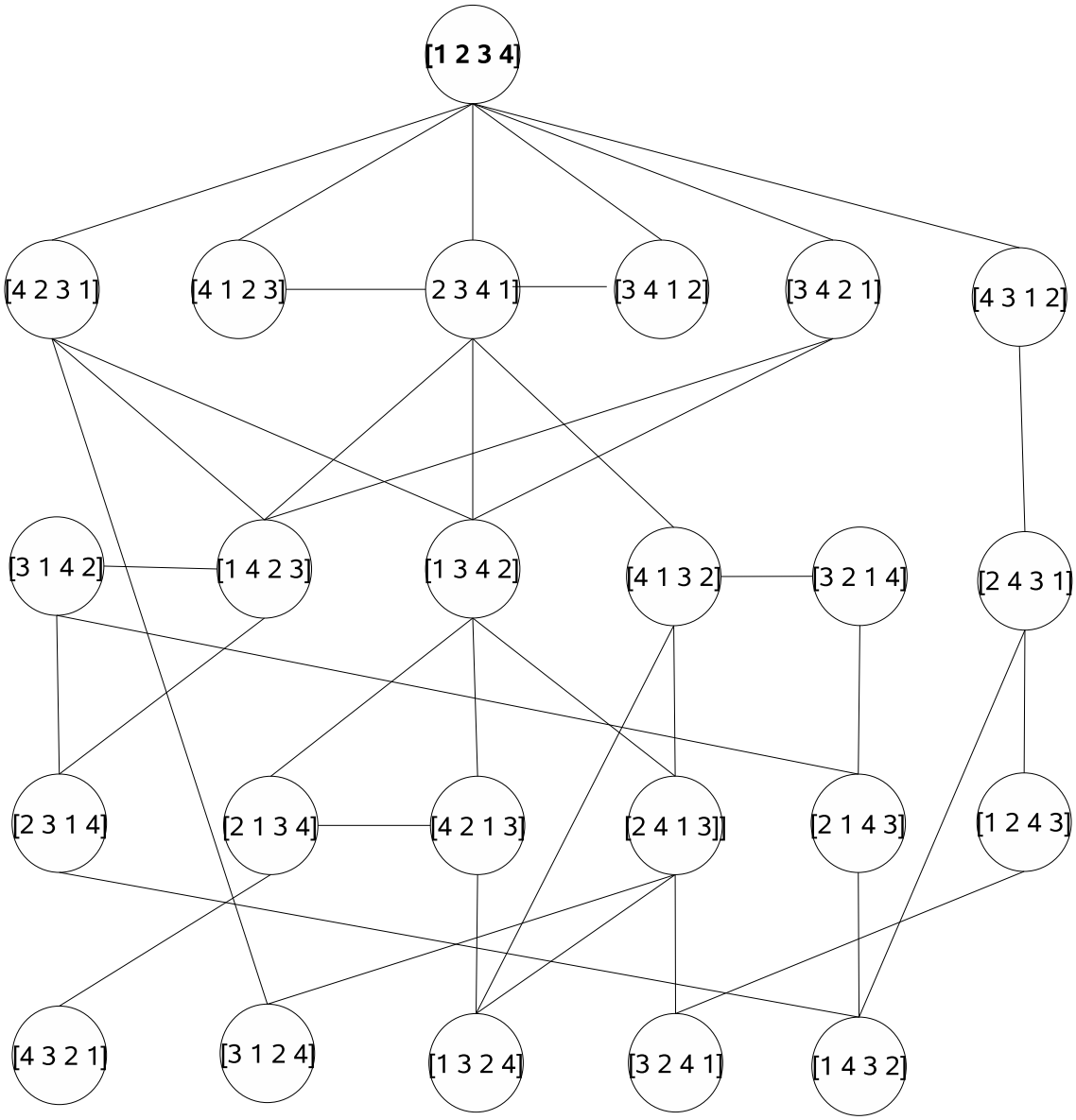



Figure 4.2: This graph shows the transformation between two permutation vectors. Each node present one permutation vector and each edge connects two nodes which are neighbors. Thus, each node has degree of 6. (To avoid complexity, all edges are not shown.)

else

$$state1 = v$$

- **Step 3** Go to step 1 until we reach the maximum number of iterations.
- **Step 4** Get $state0$ as permutation vector.

Figure 4.3 shows the flowchart of this procedure. At the begining, we choose one of the permutation vectors as the initial vector. Then, the next permutation vector is obtained from the first one by mapping $g(i, j)$ . The best permutation vector is stored in $state0$, and $state1$ is a new permutation vector which will be generated. Do this procedure for maximum number of iterations. A schematic of this procedure is shown in Figure 4.4.

### 4.2.1 The Modified Procedure

In the original procedure we could go from one state to another by using mapping $g(i, j)$. We make a modification on the procedure such that in step 2, if the difference between the lowest BER and next generated BER is greater than a threshold, $\delta$, then we do not move to that state and we generate another state from the current state. In the other words, we go to the next state if its BER is within a $\delta$ from lowest BER. Thus, the procedure becomes:

- **Step 0** Generate a random permutation vector, $v$, compute $\text{BER}_{int}$ choose $state0 = v$ and $state1 = v$.

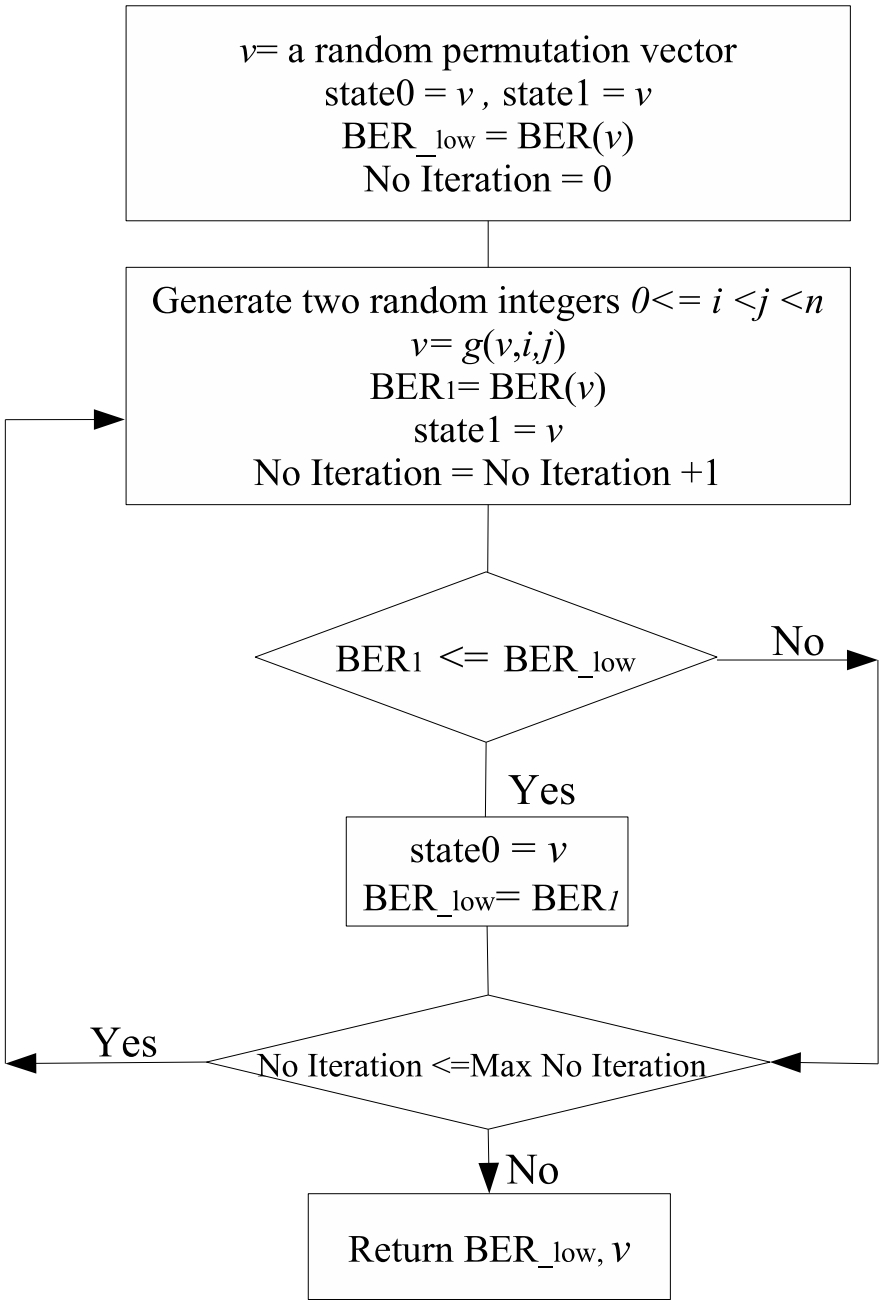
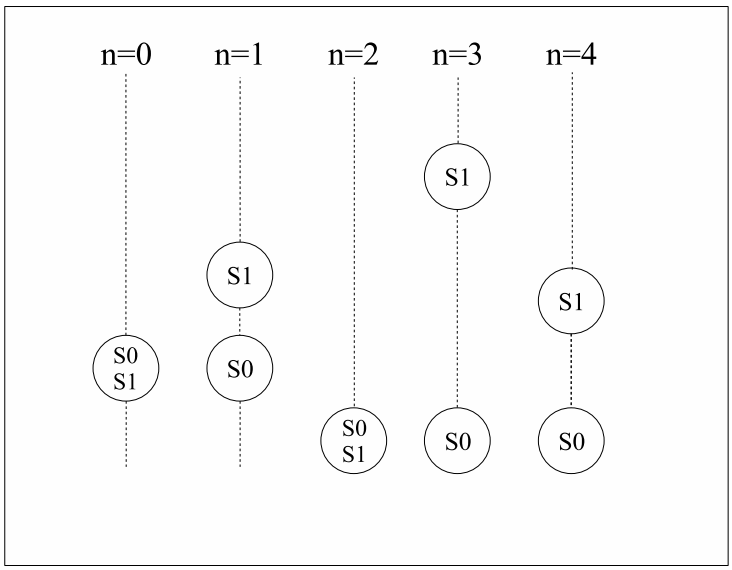
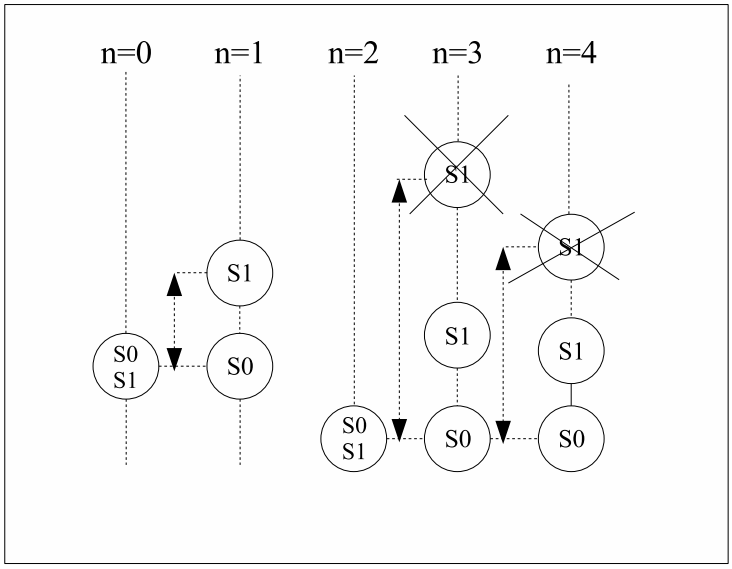



Figure 4.3: The flow chart of proposed heuristic




(a)




(b)

Figure 4.4: Schematic for changing states during procedure, The upper states have higher BER. (a) Changing states without any restriction (b) changing states restricted to threshold $\delta$

- **Step 1** Generate random numbers $0 \leq i < j < n$ and use mapping $g(i, j)$ on $v$ to obtain another vector and compute $\mathrm{BER}_1$

- **Step 2** If $\mathrm{BER}_1 < \mathrm{BER}_{low}$ then

$$state0 = v \qquad state1 = v \qquad \mathrm{BER}_{low} = \mathrm{BER}_1$$

  else if $\mathrm{BER}_1 - \mathrm{BER}_{low} < \delta$

$$state1 = v$$

- **Step 3** Go to step 1 until we reach the maximum number of iterations.

- **Step 4** Get $state0$ as permutation vector.

Figure 4.4b shows the schematic for the modified heuristic. The problem is how we to choose the threshold $\delta$.

## 4.3 Simulation Results

In this section, the original procedure and modified one are simulated. At the begining, a permutation vector of size 5 or a codeword of size 40 is used to explain the procedures in detail. Then results of simulations for permutation vectors of size 50, 100 and 200, which correspond to codewords of size 400, 800 and 1600 are shown. The channel is AWGN channel and signal to noise ratio is chosen 3.

For a permutation vector of length 5, there are $5! = 120$ different parity check matrices. Thus we have chosen 12 steps, one-tenth of the total number of possibilities, to run the simulation. Figure 4.5 shows the state and BER at each iteration.

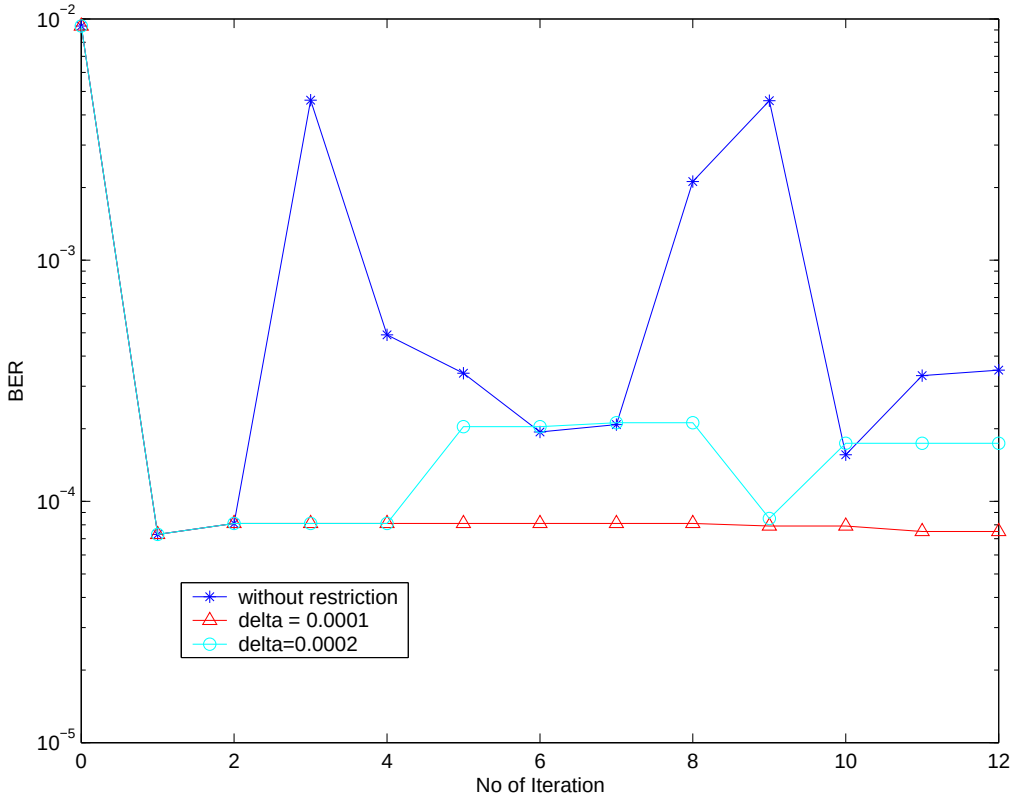



Figure 4.5: The BER of permutation vector of size 5 at each iteration.

Now, the role of $\delta$ in choosing the next step is investigated. If $\delta$ is too big, the modified and original procedure will have the same result. If $\delta$ is too small, then the algorithm may get stuck in one state neighborhood.

As we can see in Figure 4.5 when $\delta$ is chosen very small, 0.0001 , the neighbors of one node are scanned. In general, if we stay in one node more than $\frac{n(n-1)}{2}$ times, we can not do better than the BER of the current state. If $\delta > 0.01$ , the movement among states is like not having any threshold. Figure 4.6 shows the permutation vectors exchange for each iteration. As we can see, when there is no restriction, we can go from each state to another one using mapping $g(i, j)$. When $\delta$ is small, 0.0001, after one move we get stuck in the state of [ 5 3 1 4 2 ]. This shows that $\delta = 0.0001$ is small for a vector of size 5. Increasing $\delta$ from 0.0001 to 0.0002 gives us more flexibility to change the states. Nevertheless, in all cases [ 4 2 5 3 1 ] emerged as the best permutation vector.

Increasing the size of the parity check matrix makes the problem more interesting. For the first step, we increase the size of the permutation vector to the 50. Figure 4.7 shows the results of the program for 50 steps and for three different cases: without any restriction, with $\delta = 0.00001$, and $\delta = 0.00005$. The number of iterations is 50 which is very small compared to our search domain which has size of 50!. These two $\delta$ are almost reasonable, they are not so big that there is no difference from the no restriction case and are not so small such that we stay in one state forever.

Figure 4.8 and Figure 4.9 show the BER results for codewords of length 800 and 1600. As we can see, when the size of permutation matrix is 100, if $\delta = 0.000005$

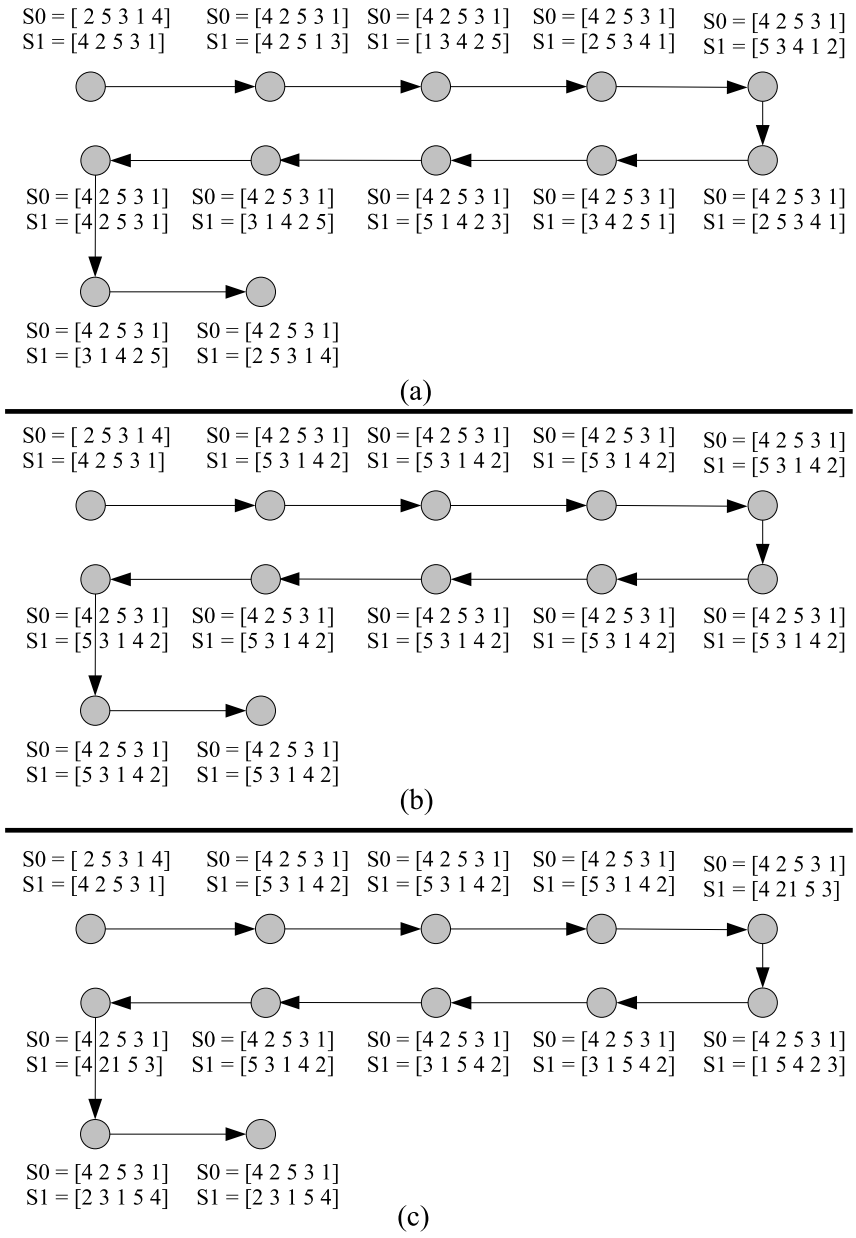



Figure 4.6: The permutation vectors of size 5 at each state (a) without $\delta$, (b) with $\delta = 0.0001$, and (c) $\delta = 0.0002$.

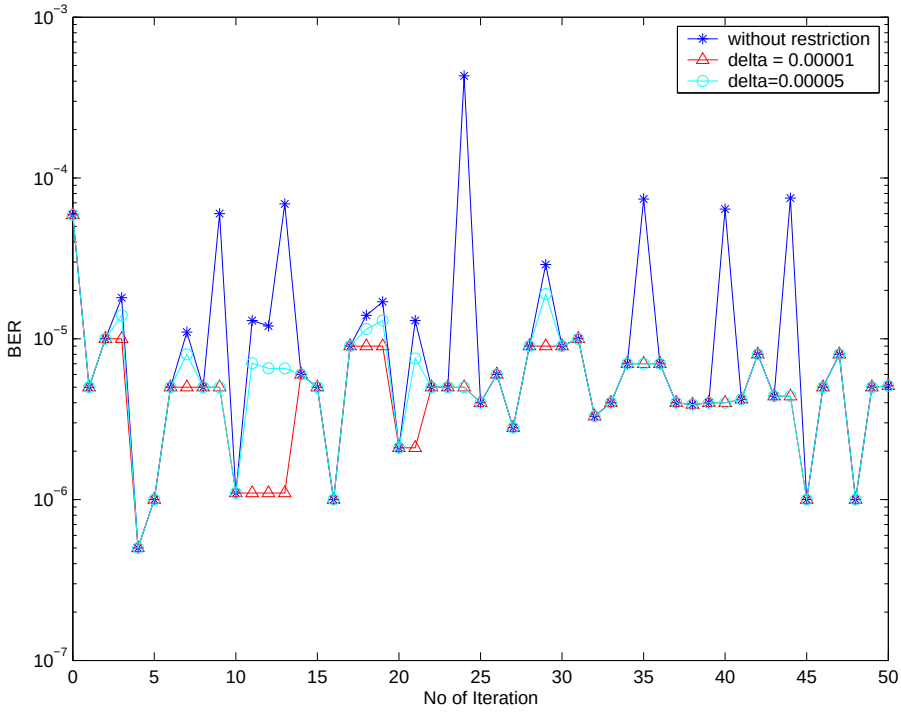



Figure 4.7: The BER for permutation vector of size 50 at each iteration for three different cases.

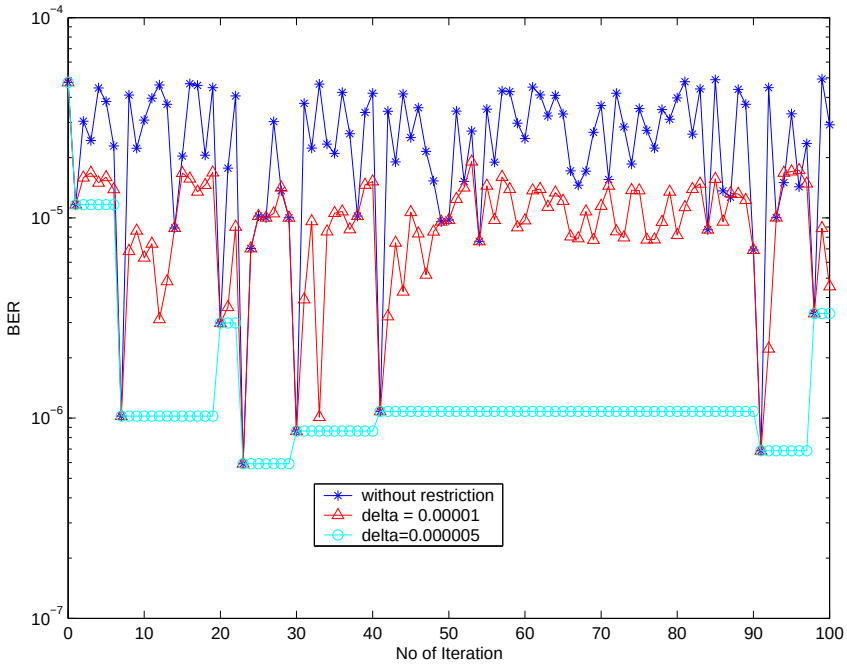



Figure 4.8: The BER of permutation vector of size 100 at each iteration for three different cases.

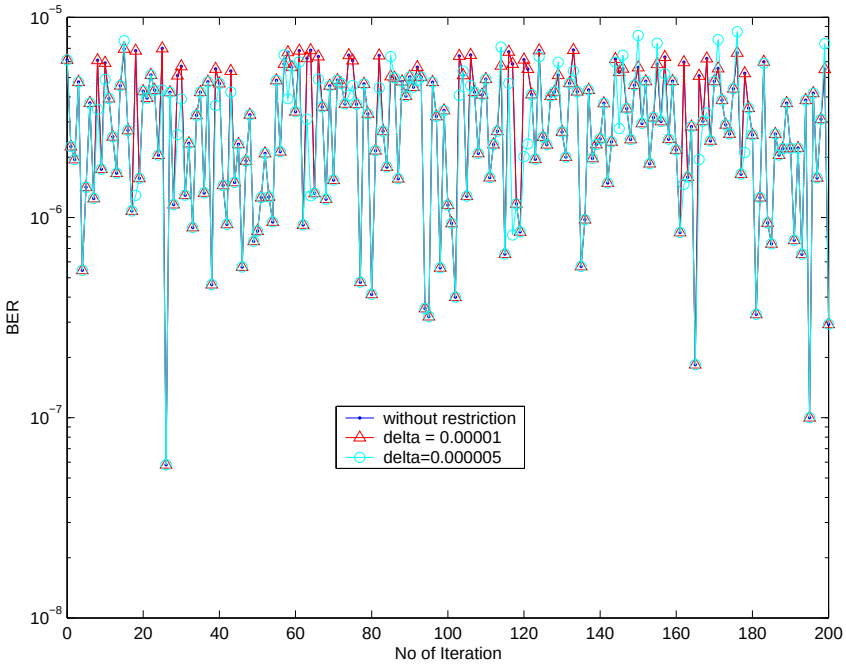



Figure 4.9: The BER of permutation vector of size 200 at each iteration for three different cases.

most of the time we stay in one state which means that this $\delta$ is small for this permutation vector size. When we increase the length of permutation vector to 200, it works better. In general, for each size of permutation vector choosing a good $\delta$ is an important problem.

Another important issue is how many times we need to run the program. The more we increase the number of iterations, the more states we scan . So, there a trade off between running time and having better BER. We can put limits on our performance. For example, we can run the program until we get the BER $< \epsilon$, then we stop.

The other method is run the program for at most $n$ times. If an acceptable BER is reached , we are done. Otherwise we delete all those states which have been

already visited and run the program one more time for the rest of states. In general, we can not set a specific number of iterations for each permutation vector because we are doing a random search in a graph with too many nodes.

# Chapter 5

# Conclusions

Error correcting codes introduce redundancy into a sequence of information bits to reduce the BER and increase the reliability of telecommunications systems. Shannon proved that to get a rate $R$ close to the channel capacity, $C$, we need to choose the block length $\nu$ sufficiently large. Many code books were introduced to approach the Shannon limit. Gallager in his Ph.D thesis [8] introduced the Low-Density Parity Check (LDPC) codes which perform well with large code length. His parity check matrix has a random and sparse structure. The number of ones in each row and column are fixed and small compared to the size of the matrix. He used an iterative decoding method in receiver to eestimate the information bits. To have a good decoder, the girth of the matrix must be big enough. Other researchers have shown that if the parity check matrix in LDPC codes is chosen irregular instead instead of regular, the performance can be better. The main disadvantages of a random parity check matrix is that it needs a large amount of storage and circuit implementation is complex.

A deterministic parity check matrix is another option. Echard proposed an irregular deterministic parity check matrix which is based on a $\pi$-rotation permutation matrix. This parity check matrix can perform as well as the random matrix and its implementation and storage are easy. To find a good permutation matrix,

Echard suggested counting the number of cycles of length four in the sub-matrix, which does not include all the loops with girth four.

We introduced a heuristic to search for the best permutation vector. Each permutation vector is considered as a node in a graph and two nodes can be connected together if they can be obtained from each other by using a defined mapping. Thus we have a big graph with too many nodes where the transition probability between each node and other nodes is $p$ if they are connected and otherwise is zero. Therefore we have a Markov chain, and our search is like a random walk in the chain. In general, there is no fixed rule for this search. You can start from any state and end up to any state in the graph. The procedure does not require the number of short loops in the matrix to be counted, which is an exhaustive job when your codeword length is large. Transition restrictions from one state to another can be introduced, which avoids going to those states which have poor performance compared to the current state.

For further research, it would be good to analyse the worst case performance of the algorithm and somehow merge it with Echard's counting loop procedure to reach the local optimum sooner.

# BIBLIOGRAPHY


[1] Berrou, C.;Glavieux, A.,"Near optimum error correcting coding and decoding: turbo codes," *Communication, IEEE Transaction on*, vol.44, no.10, pp. 1261-1271, Oct 1996

[2] Berrou, C.;Glavieux, A.; Thitimajshima, P.,"Near shannon limit error correcting coding and decoding: Turbo-codes.1," *Communication,1993.ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol.2, pp. 1064-1070, May 1993

[3] Echard, R.; Shih-Chun Chang,"Irregular $\pi$-rotation LDPC codes," *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE* , vol.2, pp. 1274-1278, Nov. 2002

[4] Echard, R.; Shih-Chun Chang,"The $\pi$-rotation low-density parity check codes," *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, vol.2, pp. 980-984, 2001

[5] Echard, R.,"On the Construction of Some Deterministic low-density parity check codes" *Ph.D. thesis,George Mason University, VA*, 2003.

[6] Elias, P.,"Error free coding," *Information Theory, IEEE Transaction on*, vol.4, no.4, pp. 29-37, Sep 1954

[7] Fan,J.L.,"Constrained coding And Soft iterative decoding," *Information Theory Workshop, 2001. Proceedings. 2001 IEEE*, pp. 18-20, 2001

[8] Gallager, R.,"Low-density parity check codes,"*Information Theory, IEEE Transaction on* , vol.8, no.1, pp. 21-28, Jan 1962

[9] MacKay, D.J.C.,"Good error-correcting codes based on very sparse matrices,"*Information Theory. 1997. Proceedings.,1997 IEEE International Symposium on*, pp. 113-, Jul 1997

[10] Sae-Young Chung; Forney, G.D, Jr.; Richardson, T.J.; Urbanke, R.,"On the design of low-density parity-check codes within 0.0045 db of the Shannon limit,"*Communications Letters, IEEE*, vol.5, no.2, pp. 58-60, Feb 2001

[11] Shannon, C.E.,"A mathematical theory of communication," *Bell System Technical Journal*, vol.27, pp. 379-423, Oct 1948

[12] Shannon, C.E.,"Probability of error for optimal codes in a guassian channel," *Bell System Technical Journal*, vol.38, pp. 611-656, 1959

[13] Tretter, S.,"Error Correcting Codes,"*Class Notes, University of Maryland*, http://www.enee.umd.edu/∼tretter/enee722/

[14] Viterbi, A.,"Error bounds for convolutional codes an asymptotically optimum decoding algorithm,"*Information Theory, IEEE Transaction on*, vol.13, no.2, pp. 260-269, Apr 1967