2018

On Lowering the Error Floor of Short-to-Medium Block Length Irregular Low Density Parity Check Codes

Abdu-Aguye, Umar-Faruk

ON LOWERING THE ERROR-FLOOR OF SHORT-TO-MEDIUM BLOCK LENGTH IRREGULAR LOW DENSITY PARITY-CHECK CODES

Umar-Faruk Abdu-Aguye

Ph.D.

October, 2017

ON LOWERING THE ERROR-FLOOR OF SHORT-TO-MEDIUM BLOCK LENGTH IRREGULAR LOW DENSITY PARITY-CHECK CODES

by

UMAR-FARUK ABDU-AGUYE

A thesis submitted to Plymouth University

in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

Department of Communication and Electronic Engineering

School of Computing, Electronics and Mathematics

Faculty of Science and Engineering

Plymouth University

United Kingdom

October, 2017

On lowering the error-floor of short-to-medium block length irregular low density parity-check codes

Umar-Faruk Abdu-Aguye

Abstract

Gallager proposed and developed low-density parity-check (LDPC) codes in the early 1960s. LDPC codes were rediscovered in the early 1990s and shown to be capacity-approaching over the additive white Gaussian noise (AWGN) channel. Subsequently, density evolution (DE) optimized symbol node degree distributions were used to significantly improve the decoding performance of short-to-medium length irregular LDPC codes. Currently, the short-to-medium length LDPC codes with the lowest error-floor are DE optimized irregular LDPC codes constructed using progressive edge-growth (PEG) algorithm modifications which are designed to increase the *approximate cycle extrinsic message degrees* (ACE) in the LDPC code graphs constructed.

The aim of the present work is to find efficient means to improve on the error-floor performance published for short-to-medium length irregular LDPC codes over AWGN channels in the literature. An efficient algorithm for determining the girth and ACE distributions in short-to-medium length LDPC code Tanner graphs has been proposed. A cyclic PEG (CPEG) algorithm which uses an edge connections sequence that results in LDPC codes with improved girth and ACE distributions is presented. LDPC codes with DE optimized/'good' degree distributions which have larger minimum distances and stopping distances than previously published for LDPC codes of similar length and rate have been found. It is shown that increasing the minimum distance of LDPC codes lowers their error-floor performance over AWGN channels; however, there are threshold minimum distances values above which there is no further lowering of the error-floor performance. A *minimum local girth* (*edge skipping*) (MLG (ES)) PEG algorithm is presented; the algorithm controls the minimum local girth (global girth) connected in the Tanner graphs of LDPC codes constructed by forfeiting some edge connections. A technique for constructing optimal low-correlated-edge density (OED) LDPC codes based on modified DE optimized symbol node degree distributions and the MLG (ES) PEG algorithm modification is presented. OED rate-½ $(n,\ k) = (512,\ 256)$ LDPC codes have been shown to have lower error-floor over the AWGN channel than previously published for LDPC codes of similar length and rate. Similarly, consequent to an improved symbol node degree distribution, rate-½ $(n,\ k) = (1024,\ 512)$ LDPC codes have been shown to have lower error-floor over the AWGN channel than previously published for LDPC codes of similar length and rate.

An improved BP/SPA (IBP/SPA) decoder, obtained by making two simple modifications to the standard BP/SPA decoder, has been shown to result in an unprecedented generalized improvement in the performance of short-to-medium length irregular LDPC codes under iterative message-passing decoding. The superiority of the Slepian-Wolf distributed source coding model over other distributed source coding models based on LDPC codes has been shown.

Table of Contents

List of Tables

Table Page

List of Figures

List of Algorithms

Glossary

ACE	Approximate Cycle EMD
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BCH	Bose-Chaudhuri-Hocquenghem
BCJR	Bahl Cocke Jelinek Raviv
BP	Belief Propagation
BPSK	Binary Phase-Shift Keying
BP/SPA	Belief Propagation/Sum-Product Algorithm
B&B	Branch and Bound
CCSDS	Consultative Committee for Space Data Systems
CDMA	Code Division Multiple Access
CPEG	Cyclic PEG
CPU	Central Processing Unit
DE	Density Evolution
DO	Decoder Optimized
DOIPEG	Decoder Optimized IPEG
DOPEG	Decoder Optimized PEG
DSC	Difference-Set Cyclic
DVB	Digital Video Broadcasting
ECC	Error Correction Coding
EG	Euclidean Geometry
EMD	Extrinsic Message Degree
EMPEG	Error Minimization PEG
ETS	Elementary Trapping Set(s)

EVDO	EVolution, Data Optimized
FER	Frame Error Rate
FG	Finite Geometry
FPCES	Fixed Parity-Check Equation Sequence
G-ACE	Generalized ACE
GB	GigaBytes
GF	Galois Field
GNU	GNU is Not Unix!
GSM	Global System for Mobile communication
HSPA	High Speed Packet Access
IBP/SPA	Improved BP/SPA
IEEE	Institute of Electrical and Electronics Engineers
IMPA	Improved Message-Passing Algorithm
IPEG	Improved PEG
IRA	Irregular Repeat Accumulate
LDPC	Low-Density Parity-Check
LTE	Long Term Evolution (4G)
MAP	Maximum A Posteriori
ML	Maximum-Likelihood
MLG (ES) PEG	Minimum Local Girth (Edge Skipping) PEG
ModPEG	Modified PEG
MPA	Message-Passing Algorithm
MRL	MoRe Likely
MSA	Min-Sum Algorithm
$\boldsymbol{n}$**-DFR**	$\boldsymbol{n}$ detected Decoding Failure Retrials
NASA/ESA	National Aeronautics and Space Administration/European Space Agency
NP	Non-deterministic Polynomial-time

OSMLD	One-Step Majority-Logic Decodable
OED	Optimal low-correlated-Edge Density
PEG	Progressive Edge-Growth
PG	Projective Geometry
POTS	Plain Old Telephone Service
QC	Quasi-Cyclic
RA	Repeat-Accumulate
RAM	Random Access Memory
RandPEG	Randomized PEG
RM	Reed-Muller
RPCES	Randomized Parity-Check Equation Sequence
RS	Reed-Solomon
SISO	Soft Input Soft Output
SNR	Signal-to-Noise Ratio
SNR(dB)	Signal-to-Noise Ratio in decibels
SOVA	Soft-Output Viterbi Algorithm
SPA	Sum-Product Algorithm
SPC	Single Parity-Check
SW	Slepian-Wolf
TCM	Trellis Coded Modulation
UMTS	Universal Mobile Telecommunications System (3G)
WiMAX	Worldwide Interoperability for Microwave Access. IEEE 802.16x standards
Wi-Fi	IEEE 802.11x standards

Acknowledgements

It has been a privilege to have Dr. Marcel Adrian Ambroze as my director of studies. I would like to thank him for his kind patience, support, guidance and encouragement during my study. I would like to thank my second supervisor, Professor Martin Tomlinson, for his kind advice, support, guidance and encouragement; working with him has been a rare privilege.

I am grateful to Dr. Mohammed Zaki Ahmed for his effort in support of the tripartite agreement between Emerging Markets Telecommunications Services (EMTS) Nigeria Limited (9mobile Nigeria); Ahmadu Bello University, Zaria, Nigeria; and Plymouth University, UK; which created the opportunity for this PhD study.

I am forever indebted to Emerging Markets Telecommunications Services (EMTS) Nigeria Limited (9mobile Nigeria) for the scholarship award. To my direct contacts at 9mobile Nigeria, Mrs. Oyetola Oduyemi and Mrs. Rose Makinwa, I am sincerely grateful to you for your responsive and reliable support. I am also grateful to the Ahmadu Bello University, Zaria, Nigeria, for the award of the study fellowship. Both of these awards allowed me to pursue my study in comfort.

I appreciate the sacrifices of Professor Muhammed Bashir Muazu and Professor Muhammad Munzali Jibril for spearheading and nurturing the early days of the 9mobile Telecommunications Engineering Postgraduate Program (9mobile-TEP).

To my colleagues in the Fixed & Mobile Communication Research Group (Muhammad Bashir Abdulrazaq, Mohammed Dikko Almustapha and Saeed Anibaba Eleruja) and those in the Signal Processing & Multimedia Communications Research Group (Dr. Ishaka Mkwawa, Dr. Emmanuel Jammeh, Dr. Louis Anegekuh, Peng Zhao, Ali Al-Nuaimi, Shaymaa Al-Juboori, Ali Alfayly, Salah Al-Hassan, Alcardo Alex Barakabitze, Amulya Karadi, Ibraheem Mohammadamin, Achyut Parajuli, and Chimobi Stanley Eke), I am thankful for your friendship, knowledge sharing and assistance.

To Professor Emmanuel C. Ifeachor, I appreciate your mentoring and the support you offered during the 9mobile-TEP program documentary video coverage. To Carol Watson, my Doctoral College Administrator, for your consistent support and patience.

To my friends Dr. Suleiman Garba, Mr. Yahaya Bello and Mr. Sunday Paul Onyabe, thank you for always being there and your unending support and encouragement.

To my beloved wife, daughter and son - Munirat, Fareeda and Saleem, for their love, support and company.

Lastly, but by no means least, to my parents, brothers and sisters, for their love, prayers and support.

Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was fully financed with the aid of scholarship from Emerging Markets Telecommunications Services (EMTS) Nigeria Limited (9mobile Nigeria) and a study fellowship from the Ahmadu Bello University, Zaria, Kaduna State, Nigeria.

A programme of advanced study was undertaken, which included extensive reading of literature relevant to the research project; development of software for error correction code construction, analysis and performance simulations in the C programming language; writing conference papers; and attendance of international conferences on communications.

The author has presented papers in the following peer-reviewed international conferences:

1. 9th International Symposium on Turbo Codes & Iterative Information Processing (ISTC' 2016), Brest, France, 5 - 9 September 2016;
2. 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2016), Split, Croatia, 22 - 24 September 2016; and
3. 10th International Conference on Signal Processing and Communication Systems (ICSPCS' 2016), Surfers Paradise, Australia, 19 - 21 December 2016.

The author has co-authored a paper which has been presented at the following international conference:

1. 5th IEEE Global Conference on Signal and Information Processing (GlobalSIP 2017), Montreal, Canada, 14 - 16 November 2017.

Word count of main body of thesis: 91,802

Signed: _____

Umar-Faruk Abdu-Aguye

Date: _____

This thesis is dedicated to my family

Chapter 1

Introduction

Information theory studies the quantification, processing, transmission, extraction and utilization of information. The broad field of information theory has its origins in Claude Shannon's 1948 paper entitled 'A Mathematical Theory of Communication'. The five major concepts in Shannon's paper are *entropy* and *information content*, *channel capacity* and *the noisy-channel coding theorem*, *formal architecture* of communication systems, unification of all information media through *digital representation* of messages over communication channels, and *source coding* or *data compression*. In essence, Shannon's paper laid the foundations of the digital age and he is often called the '*father of the digital age*'. Figure 1.1 shows the formal architecture of a communication system.

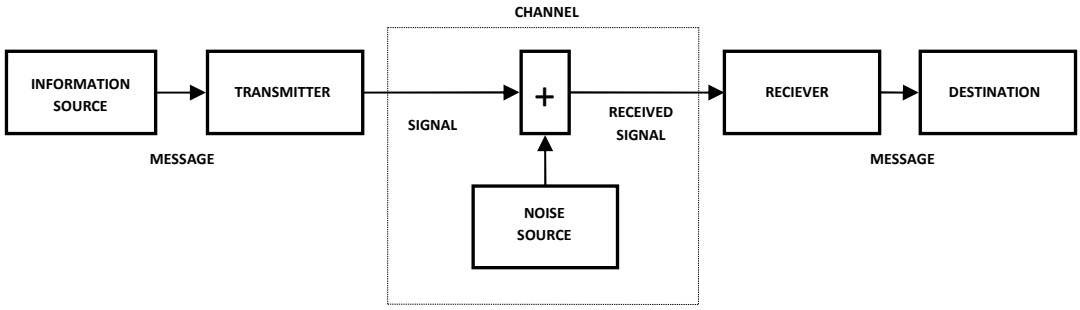

Figure 1.1: The formal architecture of a communication system [Aftab et al., 2001]

When messages are sent over a communication link it is essential that they are received with the lowest possible probability of error. Errors are mainly as a result of the channel. Sources of error

abound in practical communication systems and include noise, interference, distortion and attenuation. The probability of bit error, or bit error rate (BER), is defined as:

$$BER = \frac{Number\ of\ bit\ errors}{Total\ number\ of\ bits} \tag{1.1}$$

As an example, Figure 1.2. shows the value of the bits transmitted and received in a bit stream transmitted over a communication channel.

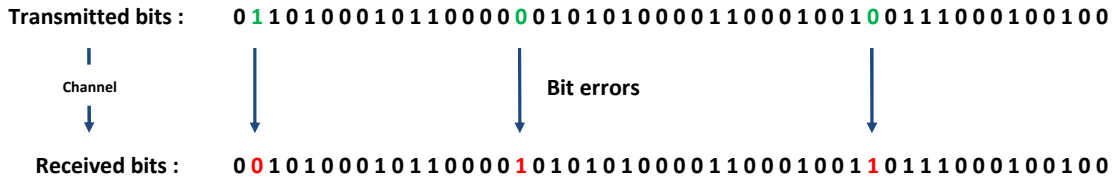

Figure 1.2: Bit errors in a received bit stream

Out of a total of 50 bits transmitted and received, 3 bit errors occurred at the receiver. Substituting in equation (1.1), the BER of the received bit stream is $BER = 0.06 = 6 \times 10^{-2}$.

For reliable transmission of information across noisy channels, information bits are often partitioned into blocks of length k bits, and then encoded by appending *redundant* (*parity*) bits to each block. *Codes (codewords)* of block length n bits are obtained when information blocks of length k bits are encoded. Although the information bits in a code block are generally independent, the parity bits should be dependent on all k information bits in the block. *Code bits* are comprised of either the information and parity bits combined or only the parity bits. Code bits are transmitted over the channel.

Equation (1.2) presents the formula for the *code rate, R,* which is defined as the ratio of the number of information bits in each code block, k to the number of code bits, n.

$$R = \frac{k}{n} \tag{1.2}$$

The *block error rate*, or *frame error rate* (FER) is defined as:

$$FER = \frac{Number\ of\ blocks\ decoded\ with\ at\ least\ one\ bit\ error}{Total\ number\ of\ blocks\ decoded} \tag{1.3}$$

According to Shannon, "The fundamental problem of communication is that of reproducing at one point, either exactly or approximately, a message selected at another point." Therefore, the fundamental challenge in communication engineering is finding the most efficient way(s) to attain the minimum achievable error rates (FER and/or BER) over any given channel.

The most noteworthy result of Shannon's paper is the channel capacity and noisy-channel coding theorem; the concept that if efficiently coded information of block length n is transmitted through a noisy communication channel at a rate equal to or below a limit called the *channel capacity* C, the probability of decoding error would approach zero exponentially as $n \rightarrow \infty$. This limit is also known as the *Shannon limit*. The channel capacity of a communication channel represents the Shannon limit in bits per second. Channel capacity is solely determined by the statistical model of the communication channel, it is a theoretical benchmark that states what can be achieved and challenges people to achieve it. The noisy-channel coding theorem gave rise to error correction codes and the entire field of channel coding theory: the concept of introducing redundant digits in digital representation to protect the information in transmitted messages against corruption. The channel capacity *C* for the additive white Gaussian noise (AWGN) channel is presented in equation (1.4),

$$C = W\ \log_2\left(1 + \frac{C}{W}\frac{E_b}{N_o}\right) \tag{1.4}$$

where W is the accessible bandwidth, and E_b/N_o is the energy per bit to noise power spectral density ratio. Equation (1.5) is obtained when equation (1.4) is solved for E_b/N_o.

$$\frac{E_b}{N_o} = \frac{2^{C/W} - 1}{C/W} \tag{1.5}$$

Consider channels with unlimited bandwidths ($W \rightarrow \infty$) and/or bit rates reducing to zero ($C \rightarrow 0$), in these situations $C/W \rightarrow 0$. Equation (1.6) resolves equation (1.5) in the limit $C/W \rightarrow 0$, and gives the Shannon limit for AWGN channels in terms of E_b/N_o.

$$\frac{E_b}{N_o} = \frac{2^{C/W}-1}{C/W} > \lim_{C/W \rightarrow 0} \frac{2^{C/W}-1}{C/W} = ln(2) = -1.6\text{dB} \quad (1.6)$$

From equation (1.6), the ultimate Shannon limit on AWGN channels is $E_b/N_o = -1.6\text{dB}$. Thus, E_b/N_o must not be less than $-1.6dB$ on AWGN channels. This implies that for $E_b/N_o \geq -1.6dB$ and any given bandwidth W, information can be transmitted on an AWGN channel with as few errors as desirable if the block length n is sufficiently large, and the transmission rate is not more than the value of C computed using equation (1.4) [Ambroze, 2000]. For any given modulation scheme in a practical communication system, the absolute limit of $E_b/N_o = -1.6dB$ for the AWGN channel should be achieved as the code rate R approaches zero. The E_b/N_o limit for non-zero code rate increases asymptotically with increasing code rate. [Dolinar et al., 1998]

From the noisy channel coding theorem, it is clear that in order to efficiently utilize communication channels while maintaining low decoding error probabilities in data transmissions, communication system implementations would generally require the use of long block lengths. Unfortunately, long block lengths results in significant overhead in both encoders and decoders in terms of computational time (delays) and equipment costs. A solution to this challenge was first proposed by Robert Gallager [Gallager, 1962], he introduced the combination of a new class of parity-check codes, called low-density parity-check (LDPC) codes, and a probabilistic message-passing decoding technique as a solution which allowed the utilization of long block lengths prerequisite for low error probabilities while maintaining low computational requirements and equipment costs.

Developments in coding theory have led to significant improvements in the efficiency and reliability of communication systems. Error correction coding (ECC) in particular, has proven to be a very powerful tool for reliable data transfer over a significant variety of channels. In many modern communication systems, despite efficient coding and transmission at rates below channel capacity, the

use of longer block lengths than are desired is inevitable in order to achieve the required decoding error probabilities. However, for many real-time processing applications long delays are not acceptable. Thus the maximum acceptable delay in real time applications limits the length of codes that can be used. [Sweeny, 1991].

The minimum acceptable (or recommended) decoding error probabilities for the different error correction coding applications in modern digital communication systems depends on the nature of the information being coded. For voice applications the acceptable BER is 10^{-3}. For fibre optical communication the typical BER is 10^{-9} [Mouri, 2017]. Data communications have more stringent requirements where 10^{-13} is often considered the minimum acceptable; the recommended BER for 10 Gigabit Ethernet is 10^{-13} [Chang, 1999]. As a consequence of the proliferation of bandwidth-hungry applications such as video transport, data over IP, wireless base-station, and medical applications, there is an increasing demand for higher data rates and even lower latency for serial data communication. For example, the Open Base Station Architecture Initiative (OBSAI) required a BER of less than 10^{-15} for their proposed wireless base-station networking standard [Subbiah, 2008].

When data sent over a communication channel is encoded using short-to-medium block lengths, higher signal-to-noise ratios (SNRs) (or E_b/N_o) are required in order to achieve the same error probability as those for long codes at significantly less SNR. The error correction capability of codes generally improves with code length. Improving the error correction capability of short-to-medium length codes decoded using probabilistic decoding is the subject of a significant amount of current research; such codes will significantly increase the overall efficiency of real-time communication systems by decreasing coding and decoding complexity, and processing delays while maintaining acceptable decoding error probabilities.

1.1 Linear Codes: Basic Definitions and Notation

1.1.1 Linear Codes

An $[n, k, d]_q$ linear code C is a k-dimensional vector subspace of $\mathbb{F}_q^n$, where $\mathbb{F}_q^n$ is an n-dimensional vector space over a finite field of q elements, $\mathbb{F}_q$. The codewords of linear codes are the k-dimensional vector subset of $\mathbb{F}_q^n$ which have length of n symbols. Each codeword is denoted by $c = [c_0, c_1, \ldots c_k. . , c_{n-1}]$. The maximum possible number of codewords in C is given by q^k, i.e. $|C| \leq q^k$. The variable d is called the minimum (Hamming) distance of the code (see Section 1.1.5). These codes are described as *linear* because a codeword can be obtained from linear combinations of other codewords, and the sum of all codewords in C is an all zeros vector. Mathematical operations such as addition and multiplication are performed under the algebra of $\mathbb{F}_q$. Throughout this thesis, if the subscript q is omitted from $[n, k, d]_q$ in the notation for C it should be understood as $[n, k, d]_2$; that is, it is a binary code, and $[n, k, d] \equiv [n, k, d]_2$.

1.1.2 Code Rate

As in equation (1.2), the code rate R of an $[n, k, d]_q$ linear code C is given by the ratio k/n.

1.1.3 Hamming Weight

The Hamming weight or simply *weight* of any vector $v \in \mathbb{F}_q^n$, where $v = [v_0, v_{1, \ldots ,} v_{n-1}]$, is the number of non-zero elements in v. The Hamming weight is denoted by $wt_H(v)$, and can be expressed as,

$$wt_H(v) = |\{v_i \neq 0 \ \ \forall \ 0 \leq i \leq n-1\}| \tag{1.7}$$

1.1.4 Hamming Distance

The Hamming distance between two vectors u and v, where $u, v \in \mathbb{F}_q^n$, is the number of coordinates in which u and v differ. The Hamming distance is denoted by $d_H(u, v)$. The Hamming distance between vectors u and v can be expressed as,

$$d_H(u,\ v) = |\ \{\ (u_i - v_i) \neq 0\ \forall\ 0 \leq i \leq n-1\ \}\ | \tag{1.8}$$

or as,

$$d_H(u,\ v) = wt_H(u - v) \tag{1.9}$$

1.1.5 Minimum Distance

The minimum distance of a code C, denoted by d_{min} (or just d), is the minimum Hamming distance between an two distinct codewords of C. A code of minimum Hamming distance d_{min} can always detect t symbol errors whenever $t < d_{min}$, and is capable of correcting all $0 \leq e \leq \lfloor (d_{min} - 1)/2 \rfloor$ symbol errors, where $\lfloor x \rfloor$ is the largest integer that is at most x. The minimum distance d_{min} of a code can be expressed as,

$$d_{min} = min\{d_H(c,\ c')\ \forall\ c, c' \in C\} \tag{1.10}$$

As a result of the linearity of C, including the existence of all zeros codeword, d_{min} can also be expressed as,

$$d_{min} = min\{wt_H(c)\ \forall\ c \in C\} \tag{1.11}$$

1.1.6 Parity-Check Matrix and Parity-Check Equation

The parity-check matrix H of a code C is an $(n-k) \times n$ matrix. It contains $n \times k$ linearly independent vectors of $\mathbb{F}_q^n$ so that $cH^T = 0$ for all codewords $c \in C$. Each row in the parity-check matrix H is a parity-check equation. H may be expressed in a reduced row echelon form so that the last $n-k$ columns of H form an $(n-k) \times (n-k)$ identity matrix I_{n-k}. Given H in the reduced row echelon form and an arbitrary information vector $u \in \mathbb{F}_q^k$, the parity-check symbols of the corresponding codeword $c \in C$ can be obtained by taking the component-wise product of u and the first k columns of H.

1.1.7 Regular and Irregular Codes

Codes with parity-check matrices which have a fixed number of non-zero symbols in each row, and a fixed number of non-zero symbols in each column are referred to as *regular* codes; otherwise they are referred to as *irregular* codes.

1.1.8 Generator Matrix

The generator matrix G of an $[n, k, d]_q$ linear code C is a $k \times n$ matrix which contains k linearly independent codewords of C. A codeword $c \in C$ can be obtained by taking any linear row combination of G. The corresponding codeword $c \in C$ for an arbitrary information vector $u \in \mathbb{F}_q^k$, is obtained using the generator matrix G as $c = uG$. Similar to H, the matrix G may be expressed in a reduced row echelon or *systematic* form by elementary row operations and, if necessary, some column permutations. In the systematic form, the first k coordinates of G is an identity matrix I_k. In the systematic codeword $c \in C$ for an arbitrary information vector $u \in \mathbb{F}_q^k$, which is obtained using $c = uG$ with G in the systematic form, the first k symbols are u, and the remaining $n - k$ coordinates contain the parity-check symbols.

1.1.9 Syndrome

Given an arbitrary vector $v \in \mathbb{F}_q^n$, the syndrome of v in a code whose parity-check matrix is H is a vector $s \in \mathbb{F}_q^{n-k}$ defined by,

$$s = vH^T \tag{1.12}$$

If the vector v is a codeword of code C with parity-check matrix H, i.e. $v \in C$, then $s = 0$. Otherwise, at least one coordinate of s would have a non-zero value and vector $s \neq 0$.

[Tjhai, 2007]

1.2 Background to Error Correction Coding

In his ground breaking work [Shannon, 1948], Shannon gave an example of how 4 bits of information ($k = 4$) can be encoded to a 7 bit codeword ($n = 7$). The coding scheme used in the example was due to Richard Hamming. For this coding scheme, assuming the use of binary antipodal signalling in a bandlimited AWGN channel, a plot of the probability of bit error P_B against E_b/N_o in decibels (dB) results in $P_B = 1 \times 10^{-5}$ at $E_b/N_o = 9.6dB$. This result is approximately $8.9dB$ away from the ultimate Shannon limit, and communications engineers and mathematicians enthusiastically took on the challenge of designing code schemes which better approach their corresponding Shannon limit.

In 1950, Hamming eventually published the coding scheme he invented, which are now known as Hamming codes [Hamming, 1950]. Hamming codes are a class of binary codes that are capable of correcting any single bit error, they have parameters given by $[n,\ k,\ d]_q = [2^m - 1,\ 2^m - 1 - m,\ 3]_2$ for $m > 2$, where d is the minimum (Hamming) distance of the code, and q is the base (number of elements in the finite field) of the code. The example given in Shannon's 1948 paper is for $m = 3$; it is a $[7,\ 4,\ 3]_2$ Hamming code. The $[23,\ 12,\ 7]_2$ binary Golay code was introduced by Marcel Golay in 1949 [Golay, 1949]. In this code, 11 redundant bits are used to protect 12 bits of information, and the code can correct up to 3 bit errors. The structure of the code is such that each of its 2^{11} syndromes represents a unique error pattern for up to 3 bit errors in a codeword. Codes that have this property are called *perfect*, *lossless* or *close-packed* codes. Hamming codes, the binary $[23,\ 12,\ 7]_2$ Golay code, and the ternary $[11,\ 6,\ 5]_3$ Golay code are the only non-trivial perfect codes known to date.

With the limited computing technology of the 1950s it was quickly realized that in addition to finding efficient coding schemes, the other major obstacle to approaching Shannon's limit was implementing the required decoding due to its complexity. The relatively simple *matched filter decoder* was one of the best decoders available at the time. Nonetheless, the complexity of the matched filter decoder required for codes that could hypothetically give good coding gain γ_c often proved prohibitive to implement. As an example, while exhaustive decoding of very short codes such as the $[7,\ 4,\ 3]_2$ Hamming code which has $\gamma_c = 2.34dB$ was practicable, in contrast, the $[23,\ 12,\ 7]_2$ Golay code which gives $\gamma_c = 5.63dB$ had impractical decoding complexity. As a consequence of the increase in decoding complexity required for codes with higher coding gain, significant effort was committed to constructing codes with efficient decoder structures.

A simple, non optimum, method for decoding using the available technology in the 1950s is the *majority logic decoding* which is based on majority voting. David Muller, introduced a class of binary codes that can be decoded using majority logic [Muller, 1954], and Irving Reed invented an efficient majority logic decoding algorithm that was suited to such codes [Reed, 1953]. Codes encoded and decoded using these methods are called Reed-Muller (RM) codes. According to

Peterson and Weldon, Jr. [Peterson and Weldon, Jr., 1972], N. Mitani had earlier discovered the RM error correction code method in 1951[Mitani, 1951]. RM codes have relatively poor error correcting capabilities and the search for better codes with easy decoding continued unabated.

Decoders that do not utilize the reliability information of symbols received from the channel are known as *hard-decision* decoders. The efficient decoders of the 1950s and 1960s were hard-decision decoders, and decoding involved a procedure to quantize the received signals into a number of levels which is equal to the field size of the code. For example, the received signals are quantized into two levels by decoders for binary codes, and into three levels by decoders for ternary codes. Inevitably, this quantization results in the loss of the reliability information of each symbol. It was realized that in order to reduce the gap to Shannon's limit, decoders that can use the symbol reliability information were essential. Taking binary codes as an example, the typical gain achieved by eliminating binary quantization and using the bit reliability information is as much as $2dB$. Decoders that take symbol reliability information into account are called *soft-decision* decoders. Soft-decision decoders are typically more complex to implement than hard-decision decoders. In this period, the inability to realize soft-decision decoding was attributed to its complexity and the limitations of the available technology.

The invention of *convolutional* or *recurrent* codes by Peter Elias in 1954 marked a very important milestone in error correction coding [Elias, 1954]. All the codes discussed up to this point are categorized as block codes. In block codes, user messages are partitioned into blocks of length k symbols, and then each of these blocks is translated to a longer block of length n symbols. The n symbol long blocks are used for transmission over the channel. However, partitioning of user messages is avoided with convolutional codes. The convolutional encoder maps a small fraction of the user message of arbitrary length k' symbols into a block of arbitrary length n' for transmission over the channel, where $n' > k'$. Unlike in block codes, the n' symbol long block in a convolutional code depends not only on the current k' user message symbols but also on some of the previous user message symbols. The *constraint length* of a convolutional code determines the number of previous user message symbols that are involved in producing the next block for transmission. The parity

symbols in convolutional codes are obtained by a 'convolution' of the message symbols over the encoder. This convolution is achieved via a sliding operation of a Boolean polynomial function to a stream of the message symbols. The sliding nature of convolutional codes allow for relatively low complexity soft-decision maximum-likelihood decoding based on time-invariant trellises.

In 1956, David Slepian's pioneering work in algebraic coding theory laid an algebraic framework for the subject of error correction coding [Slepian, 1956]. Prior to his work, the study of error-correction codes was specific to the type of code. Slepian provided a generalization of error-correction coding theory, and proposed a generalized decoding method based on the standard array. Slepian's generalized decoding method was only efficient for very short codes.

More than a decade before the introduction of trellis diagram representation of convolutional codes, a suboptimal soft-decision decoder for convolutional codes was introduced by John Wozencraft in 1957; this decoder is known as a *sequential decoder* [Wozencraft, 1957].

Following in the footsteps of Slepian, more communication engineers and mathematicians began to associate error-correction coding with algebra, this led to many new discoveries. Two of the most notable of these discoveries are Bose-Chaudhuri-Hocquenghem (BCH) codes, and Reed-Solomon (RS) codes.

BCH codes were invented independently in 1959 by Alexis Hocquenghem [Hocquenghem, 1959], and in 1960 by Raj Bose and D. Ray-Chaudhuri [Bose and Chaudhuri, 1960]. During the design of a BCH code, there is a precise control over t - the number of symbol errors which can be corrected by the code. BCH codes are t-error correcting linear block codes. BCH codes have very good mathematical structure and form a subclass of codes called cyclic codes or cyclic error-correction codes. Wesley Peterson was the first to reveal the cyclic nature of BCH codes, and explained the error correction procedure for binary BCH codes [Peterson, 1960]. RS codes are non binary linear block codes which were invented in 1960 by Irving Reed (the inventor of RM codes) and Gustave Solomon [Reed and Solomon, 1960]. RS codes were discovered to be a special class of BCH codes, they have the highest error correcting ability compared to any linear code of the same field and are considered to

be optimum. The discovery of a suitable decoding procedure for non binary BCH codes in 1961 is attributed to D. Gorenstein and N. Zierler [Gorenstein and Zierler, 1961].

Robert Gallager invented low-density parity-check (LDPC) codes at the Massachusetts Institute of Technology (MIT) in 1960 [Gallager, 1962]. LDPC codes are a class of capacity-approaching linear block codes with *sparse* or *low-density* parity-check matrices; the parity-check matrices of binary LDPC codes contain mostly 0's and a comparatively small number of 1's. Gallager proposed the *sum-product decoder*, a relatively low complexity probabilistic (soft decision) message-passing decoding technique that allows the use of the long code lengths prerequisite for low error probabilities while maintaining low equipment costs and computational requirements. The potential of Gallager's LDPC codes for error correction was not fully realized, mainly due to the limited technology of the time, and his invention was ignored for more than thirty years.

The performance of the sequential decoder was significantly improved by Roberto Fano in 1963. [Fano, 1963]. However, the sequential decoder remained suboptimal and had limited performance.

An important milestone was marked by the invention of an efficient decoding algorithm for BCH codes by Elwyn Berlekamp in 1966 [Berlekamp, 1966], this decoding algorithm was simplified and extended to linear feedback shift registers by James Massey in 1969 [Massey, 1969]. Nonbinary RS codes are also decodable using this decoding algorithm which is called the *Berlekamp-Massey algorithm*. RS codes subsequently received a lot of practical interest due to the availability of efficient decoding. The optimality of RS codes is defined symbol-wise, and each nonbinary symbol in an RS code can be mapped to a binary equivalent. Consequently, RS codes were used in many practical applications and are especially suited to applications that require burst error correction.

A new class of codes called *concatenated codes* was invented by David Forney, Jr. in 1966 [Forney, 1966]. In his coding scheme, two codes - a non binary outer code and a binary inner code, were cascaded to form a longer and more powerful binary code. The arrangement used a RS $[255, 223, 33]_{2^8}$ outer code and a rate-½ convolutional code as the inner code. In general, concatenation results in more powerful codes because the resultant longer codes have larger coding gain γ_c. The decoding of concatenated codes is done in two stages: the inner code is decoded in the

first stage, and in the second stage the outer code is decoded to correct any residual errors from decoding the inner code. The decoding complexity of a concatenated code is restricted to the decoding complexity of its component codes.

Following the precedence of RM codes, the class of majority logic decodable codes was expanded to include *difference cycle set codes* and *finite geometry codes*. Difference cycle set codes were introduced in 1966 by E. Weldon, Jr. [Weldon, Jr., 1966].

The sequential decoder used for convolutional codes in the early 1960s was suboptimal. The Viterbi algorithm, a maximum likelihood decoder for convolution codes, was invented by Andrew Viterbi in 1967 [Viterbi, 1967]. The Viterbi algorithm is a soft-decision decoder which returns a codeword which, among all the likely transmitted codewords, has the highest possibility of being correct for the given sequence received from the channel; the algorithm is optimal in terms of decoded codewords. The inventor of concatenated codes, David Forney, Jr., improved the Viterbi algorithm in 1973 using trellis diagrams [Forney, 1973]. The use of trellis diagram representations for convolutional codes marked a significant milestone in error correction coding because it made efficient optimum soft-decision decoding of convolutional codes possible.

With the availability of an optimum soft-decision decoder, convolutional codes decoded using the improved Viterbi algorithm were soon used in practical applications, such as broadcasting of digital audio and video, and mobile communications such as the global system for mobile communications (GSM) and the code division multiple access (CDMA-IS95) [Costello, Jr. et al., 1998].

There are optimal soft-decision decoders in terms of symbol; for any given symbol coordinate in a received sequence such decoders return a symbol value that has the highest probability of being correct compared to the other possible symbol values. The class of optimal symbol-by-symbol soft-decision decoders for convolutional codes include the BCJR algorithm introduced in 1974 by L. Bahl, J. Cocke, F. Jelinek, and J. Raviv [Bahl et al., 1974], and another algorithm introduced in 1976 by C. Hartmann and L. Rudolph [Hartmann and Rudolph, 1976]. The BCJR decoding algorithm makes use of trellis diagram representations and is one of the first trellis-based optimum soft-decision decoders applicable to block codes. However, optimum soft-decision decoding is only possible for

very short codes. A precursor to many of the suboptimum soft-decision decoders in existence today are the work of Wagner [Silverman and Balser, 1954], D. Chase [Chase, 1972], and B. Dorsch [Dorsch, 1974].

Mathematical generalizations of concatenated codes were provided by E. Blokh and V. Zyablov in 1974 [Blokh and Zyablov, 1974], and by V. Zinov'ev in 1976 [Zinov'ev, 1976]. Consequent to these generalizations and other succeeding work, concatenated codes with high asymptotic coding gain are now easily constructed.

In 1981, Michael Tanner proposed the use of bipartite graphs to state the constraints or equations used to specify error correction codes, and linked the study of LDPC codes with graph theory [Tanner, 1981]. Consequently, the term *Tanner Graph* is now indispensible in literature on LDPC codes. Tanner graphs are used to represent LDPC code parity-check matrices, and are very useful for describing the features, encoding, and decoding of LDPC codes.

A significant milestone in reducing the gap to the ultimate Shannon limit over the bandwidth limited telephone channel was set when Gottfried Ungerboeck published a detailed exposition of his *trellis coded modulation* (TCM) scheme in 1982 [Ungerboeck, 1982]. The TCM scheme combines coding and *M*-ary modulation. Before TCM, it was believed that coding would only be efficient for applications with high bandwidth availability, and would be inefficient for applications such as data transmission over *plain old telephone service* (POTS) lines which are limited to bandwidth of around 4 kilobits per second (kbits/s) only. The theoretical ultimate Shannon limit for POTS lines was approximately 35 kbits/s. The modem standard before the introduction of TCM typically achieved data rate of 9.6 kbits/s half-duplex, however with TCM data rate as high as 33.6 kbit/s full-duplex were attained under the V.34 modem standard.

In 1987, the Consultative Committee on Space Data Systems (CCSDS) adopted the concatenation scheme by David Forney, Jr. [Forney, 1966] as the NASA/ESA telemetry standard [Costello, Jr. et al., 1998]. The adopted scheme was decoded using the Viterbi algorithm. In 1988, the belief propagation (BP), a message-passing algorithm for performing inference on graphical models, was invented by Judea Pearl [Pearl, 1988]. Although the BP algorithm was formulated on the concept

of *trees*, it was soon shown to be a useful approximation on general graphs. When adopted to LDPC code Tanner graphs, the BP algorithm is identical to Gallager's sum-product algorithm and has demonstrated empirical success as a suboptimal soft-decision decoder.

Prompted by the observed benefits of soft input information, soft output in the context of decoding was researched and J. Hagenauer and P. Hoeher introduced the *soft-output Viterbi algorithm* (SOVA) in 1989 [Hagenauer and Hoeher, 1989]. Subsequently, soft output was used with the BCJR algorithm in 'separable symbol-by-symbol *maximum a posteriori* (MAP) *'filters'* ' for decoding multidimensional codes, product, and concatenated codes [Lodge et al., 1992] [Lodge et al., 1993].

Unlike convolutional codes which are represented by trellises which are invariant over time, block codes are generally represented by time-variant trellises, this constrains the development of efficient trellis based, optimum soft-decision decoders for block codes. The 1990s witnessed a large volume of research dedicated to the trellis complexity of block codes.

The gap to Shannon's limit remained significant in spite of the numerous achievements between 1948 and the early 1990s. The discovery of parallel-concatenated convolutional codes or *Turbo codes* in 1993 by C. Berrou, A. Glavieux, and P. Thatimajshima [Berrou et al., 1993] was a major breakthrough which significantly reduced the gap to the ultimate Shannon limit. The turbo code presented by Berrou et al. was made up of two convolutional codes separated by a pseudorandom interleaver, the overall code had a rate of $1/3$. At a block length of 65536 bits and a code rate of ½ (achieved by *puncturing*), a bit error probability of 1×10^{-5} was attained at $E_b/N_o = 0.7dB$. This represents a real coding gain γ_c of $8.9dB$ above unencoded binary antipodal signalling. The Shannon limit for a binary modulation with a rate of ½ is $P_e = 0$ (some authors take $P_e = 1 \times 10^{-5}$ as a reference) at $E_b/N_o = 0dB$. Therefore, the performance of the turbo code by Berrou et al. narrowed the gap to the ultimate Shannon limit to $0.7dB$ at a bit error probability of 1×10^{-5}. Two optimum soft decision decoders, which use the BCJR algorithm, are used for decoding the constituent convolutional codes of turbo codes. In the decoding procedure, each decoder produces *extrinsic information* (soft-output) which it sends to the other decoder and vice-versa, in an iterative manner. The exceptional

performance of turbo codes is attributed to the way it is decoded. However, a notable issue on turbo codes is the manifestation of performance flooring at low error probability.

Advancements made in iterative decoding and computational technology led to the rediscovery of Gallager's LDPC codes in 1996, more than three decades after its invention. LDPC codes were rediscovered independently by David MacKay and Radford Neal [MacKay and Neal, 1996], and by Niclas Wiberg [Wiberg, 1996]. Another major breakthrough in error correction coding was set in 1996 by MacKay and Neal when they demonstrated that besides turbo codes, Gallager's LDPC codes are also capacity-approaching in the AWGN channel. Similar to the convolutional decoders used for turbo codes, Gallager's sum-product decoder is a type of iterative decoder which can be regarded as a bank of BCJR decoders for $[n, n-1, 2]_2$ single parity-check codes. In the sum-product decoding procedure, each of the BCJR decoders produces extrinsic information which is processed and then exploited in the next iteration. The original LDPC codes proposed by Gallager were regular codes.

A class of low complexity encoding LDPC codes known as *repeat-accumulate* (RA) codes was invented by D. Divsalar, H. Jin, and R. McEliece in 1998 [Divsalar et al., 1998]. Irregular repeat-accumulate (IRA) codes, an extension to RA codes, were introduced in 2000 by H. Jin, A. Khandekar, and R. McEliece [Jin et al., 2000]. Array LDPC codes were introduced by J. L. Fan in 2000 [Fan, 2000].

In 2001, a very important contribution to LDPC code construction was made by M. Luby, M. Shokrollahi, M. Mizenmacher, and D. Spielman [Luby et al., 2001]. They showed that the performance of LDPC codes can be significantly improved by incorporating some degree of irregularity in the construction of their parity-check matrices. In the same year, work to design optimized parity-check matrix irregularities, or *degree distributions*, which lead to good coding performance by LDPC codes of different length and rate was carried out by T. Richardson, R. Urbanke and M. Shokrollahi [Richardson and Urbanke, 2001], [Richardson et al., 2001]. The degree distribution optimization method they employed is known as *density evolution* (DE). Using density evolution, S. Chung, D. Forney, Jr., T. Richardson and R. Urbanke presented an irregular binary

LDPC code at a block length of 10^7 which attained a distance of only $0.04dB$ from the ultimate Shannon limit at a bit error probability of 1×10^{-6} [Chung et al., 2001].

Turbo codes and LDPC codes have dominated modern research in error correction coding as a result of their capacity-approaching performances. Turbo codes and LDPC codes have currently been adopted in many industry standards, and there are various hardware encoders and decoders for these codes. There is a notable preference for LDPC over turbo codes. Initially, this might have been as a result of the fact that implementing LDPC codes was more economical because while Berrou's patent on turbo codes did not expire until August 29, 2013; Gallager's patent on LDPC codes had expired long before its rediscovery. Furthermore, due to recent advances in LDPC codes, they surpass turbo codes in terms of error-floor and performance in higher code rate range, which leaves turbo codes more suitable for lower code rates only ['Low-density parity-check code', 2017]. Finally, the bank of BCJR decoders for the single parity-check codes can be parallelized in decoders for long LDPC codes.

Implementation complexity has continued to be a major factor which influences the deployment of LDPC codes. In spite of superior performance, the encoding complexity of irregular LDPC codes makes them unattractive, and LDPC codes which have more regularity in their structure are favoured for deployment. Some types of LDPC codes which have structures that are more attractive for deployment are *quasi-cyclic* LDPC codes, RA codes, and IRA codes. The encoder for quasi-cyclic LDPC codes can be easily implemented using linear feedback shift registers.

Turbo codes have been deployed in 3G and 4G mobile communications such as universal mobile telecommunications system (UMTS), CDMA2000, HSPA, EV-DO and LTE; digital video broadcasting *interaction channel*, such as (DVB-RCS) and (DVB-RCS2); deep space communication such as the *Mars Reconnaissance Orbiter*; and in the IEEE 802.16 (WiMAX) wireless metropolitan area network standard [Costello and Forney, 2007], ['Turbo code', 2017]. LDPC codes have also been deployed in many contemporary communication standards as well as various data storage applications. The IRA code introduced by [Jin et al. 2000] outperformed selected turbo codes and was adopted as the error-correcting codes for the second version of digital video broadcasting standards (DVB-S2). Other communication standards which have adopted LDPC codes include the IEEE 802.3an

(Ethernet); IEEE 802.16e (WiMAX) standard for microwave communications; IEEE 802.11n-2009 (WiFi); and 2nd generation digital video broadcast standards such as DVB-S2, DVB-T2, DVB-C2 ['Low-density parity-check code', 2017].

Notable research in the area of suboptimal binary soft-decision decoding from the early 1990s to date include work by [Han et al., 1993], [Fossorier and Lin, 1995], [Gazelle and Snyders, 1997], [Valembois and Fossorier, 2004], and [Tomlinson et al., 2007]. Similarly, major contributions in the area of subpotimal non binary soft-decision decoding were made by [Sudan, 1997], [Guruswami and Sudan, 1998], and [Koetter and Vardy, 2003].

1.3 Thesis Aim and Objectives

The aim of this thesis is to study and contribute to the knowledge of the construction and decoding of short-to-medium length irregular LDPC codes for low error-floor performance over AWGN channels. Techniques to lower the error-floor of short-to-medium length LDPC codes using improved PEG algorithms and BP/SPA decoders will be researched. The primary objective of this thesis is to find efficient means to improve the error-floor performance published for short-to-medium length irregular LDPC codes in the literature. The second objective is to design and implement efficient methods to analyse short-to-medium length LDPC code matrices and Tanner graphs in order to evaluate parameters in them such as their symbol node degree distributions, check node degree distributions, global girth, local girths, and the connectivity of cycles. It is understood that the girth and degree of connectivity of cycles in LDPC code Tanner graphs have a significant impact on their error-floor performance. The third objective is to determine and study the distribution of some of the key parameters in LDPC code ensembles constructed with the same degree distribution using different PEG algorithm modifications. A study of how the distribution of these parameters vary from one PEG algorithm modification to the other, and a comparison of the error-floor performance of codes from the different ensembles will serve to verify/refute the premise behind each PEG algorithm modification.

The aspect of the research regarding short-to-medium length LDPC code design and construction will be carried out through a review of the PEG algorithms in the literature, identifying the parameters

required for optimum error-floor performance, experimentation with different degree distributions, and PEG algorithm modifications. The aspect of the research regarding improved decoding of short-to-medium length LDPC codes will be carried out via a thorough study of the decoding process of the BP/SPA decoder with a view to finding modifications which can improve the decoding performance. Modified BP/SPA decoders will be implemented and tested in decoding performance simulation experiments. It is hoped that by increasing the performance gap between irregular LDPC codes and their more regularly structured counterparts, and further advancements in computing technology, short-to-medium length irregular LDPC codes will become attractive for deployment in communication system standards of the near future.

1.4 Thesis Organisation

This thesis is organised into eight chapters including this introductory chapter.

Chapter 2 introduces LDPC codes and Tanner graphs, and describes their basic features and properties. The different types of LDPC codes and LDPC code construction techniques are discussed. A description of the standard/original PEG algorithm by [Hu et al., 2001] for constructing LDPC codes of arbitrary length and rate is given, and its implementation in this research is fully described. Using a simplified explanation of the iterative decoding processes in the standard BP/SPA decoder, the algorithm for the standard BP/SPA decoder implemented for this research is fully described. The standard BP/SPA decoder was used to simulate the decoding performance of most of the LDPC codes analysed in this thesis. The algorithm for determining the codeword weight d_{min} and stopping set weight s_{min} spectra in short-to-medium block length LDPC code matrices in [Rosnes et al., 2012] is introduced and briefly described. The minimum weights of the short-to-medium length LDPC codes analysed in this research were determined using this algorithm.

Chapter 3 discusses challenges to lowering the error-floor of short-to-medium length LDPC codes under iterative message-passing decoding. Some of the most noteworthy PEG algorithm modifications in the literature which were designed to lower the error-floor of the short-to-medium length LDPC codes constructed are examined. The modified PEG algorithms investigated are briefly explained, the

reported LDPC code performance results obtained are analysed, and the drawbacks of some of the modified PEG algorithms are highlighted.

Chapter 4 presents an efficient algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs. The algorithm is relatively easy to implement and finds all the short cycles through symbol nodes in Tanner graphs using a type of subgraph expansion which is similar to that of the standard PEG algorithm. The algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs is shown to have a lower complexity than those of similar algorithms in the literature. Additionally, the algorithm is shown to be faster than existing algorithms for counting cycles in LDPC codes when applied to irregular short-to-medium length LDPC codes. The algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs is used to verify the improved girth and ACE distributions in short-to-medium length LDPC codes constructed by the modified PEG algorithms in [Xiao and Banihashemi, 2004] and [Vukobratović and Šenk, 2008].

Chapter 5 presents a cyclic PEG (CPEG) algorithm which constructs short-to-medium length LDPC codes with improved girth and ACE distributions. The CPEG algorithm constructs LDPC codes using an edge connections sequence which is different from that of the original PEG algorithm. The d_{min} and s_{min} distributions, in four (4) ensembles of 6000 DE optimized irregular rate-½ $(n, k) = (512, 256)$ LDPC codes each are compared; these four code ensembles are constructed using different types of PEG algorithms. Many rate-½ $(n, k) = (512, 256)$ irregular LDPC codes with DE optimized/'good' degree distributions which have higher d_{min} and s_{min} than have been published for similar LDPC codes are found in the four code ensembles. Based on the DE optimized rate-½ $(n, k) = (512, 256)$ LDPC codes, the generalized effect of LDPC code minimum distance d_{min} on their error-floor performance over the AWGN channel is investigated. Given their improved girth and ACE distributions, the performance of LDPC codes constructed by CPEG algorithms are compared to the performance of codes constructed by conventional PEG algorithms.

Chapter 6 presents a minimum local girth (edge skipping) (MLG (ES)) PEG algorithm which controls the minimum local girth of cycles connected during the construction of LDPC code Tanner graphs.

The MLG (ES) PEG algorithm controls the global girth of codes it constructs by skipping all edge connections which would have resulted in undesired local girths. The concept of optimal low-correlated-edge density (OED) codes is introduced. An irregular rate-½ $(n, k) = (512, 256)$ OED code is shown to have lower error-floor than the lowest error-floor published for LDPC codes of identical rate and length. Similarly, consequent to an improved symbol node degree distribution, an irregular rate-½ $(n, k) = (1024, 512)$ LDPC code is shown to have lower error-floor than the lowest error-floor published for LDPC codes of identical rate and length. Random irregular rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ PEG constructed LDPC codes are shown to significantly outperform the structured rate-½ $(n, k) = (576, 288)$ and $(1056, 528)$ IEEE 802.16E (WiMAX) LDPC codes in the error-floor region.

Chapter 7 describes modifications made to the standard BP/SPA decoder in order to implement distributed source coding of two correlated sources for asymmetric Slepian-Wolf (SW) coding using binary LDPC codes. A BP/SPA syndrome decoder which implements a SW-like decoder is implemented. Compared to the conventional BP/SPA decoder, the BP/SPA syndrome decoder is shown to achieve a very good compression with no apparent loss in performance. Similarly, the modifications made to the standard BP/SPA decoder in order to implement SW coding of two binary correlated information sources over the BSC (a SW BSC decoder) is described. Using a rate-½ $(n, k) = (1000, 500)$ LDPC code, the SW coding model is shown to outperform other distributed source coding models in literature which utilized LDPC codes of identical length and rate. An improved BP/SPA (IBP/SPA) decoder is presented. Two novel modifications to the standard BP/SPA decoder are shown to result in unprecedented performance improvements to BP/SPA decoding of short-to-medium length LDPC codes over the AWGN channel. The modified BP/SP algorithm in the IBP/SPA decoder breaks trapping sets in LDPC codes.

In Chapter 8, the contributions to knowledge made by the thesis are summarized, the thesis is concluded, and recommendations for future work are presented.

Chapter 2

LDPC codes, the progressive edge-growth (PEG) algorithm, and the belief propagation or sum-product algorithm (BP/SPA) decoder

Following the rediscovery of LDPC codes in the mid 1990s, a lot of research effort has been dedicated to improving the construction of efficient LDPC codes and decoding algorithms. Currently, the most popular methods for constructing efficient short-to-medium length LDPC codes are based on the PEG algorithm introduced in [Hu et al., 2001]. Similarly, the most efficient practicable decoders for decoding LDPC codes received over AWGN channels are based on the suboptimal BP/SPA decoder.

This chapter provides an introduction to LDPC codes and Tanner graphs, and describes LDPC code features and properties. Following an introduction to the three LDPC code performance curve regions, the different types of LDPC codes and LDPC code construction techniques are discussed. This is followed by a detailed description of the standard PEG algorithm which was implemented in this research. All modified PEG algorithms are based on and incorporate the basic aspects of the described standard PEG algorithm. A simple explanation of the iterative decoding processes in the standard BP/SPA decoder which was used to simulate the decoding performance of most of the LDPC codes analysed in this research is given in this chapter. Finally, the algorithm for determining the codeword weight and stopping set weight spectra in short-to-medium block length LDPC code matrices

proposed in [Rosnes et al., 2012] is introduced. This algorithm was used to determine the minimum weight metrics in all the short-to-medium length LDPC codes analysed in this research.

2.1 LDPC Codes

LDPC codes are linear block codes with a relatively small number of non-zero entries in their parity-check matrices H. The dimension of a parity-check matrix H is $m \times n$; where m is the number of rows in the parity-check matrix, or the number of parity-check equations; and n is the number of columns in the parity-check matrix, or the variable/symbol length of the code described by the matrix. Figure 2.1 Shows the parity-check matrix H of an irregular LDPC code with $[n, k, d]_q = [25, 5, 8]_2$. The code rate (k/n) is $1/5$, and the number of rows $(m = n - k)$ is 20. Therefore, it is a 20×25 matrix. Apart from the requirement that H be sparse or of low-density, LDPC codes are not any different to other block codes. The sparseness of H results in a decoding complexity and a minimum distance which increase linearly with code length. LDPC codes are designed by constructing a sparse parity-check matrix first and then deriving a generator matrix for the code afterwards.

$$
H = \begin{array}{c}
\begin{array}{cccccccccccccccccccccccc}
s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_{19} & s_{20} & s_{21} & s_{22} & s_{23} & s_{24}
\end{array} \\
\left[\begin{array}{cccccccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}\right]
\begin{array}{c}
c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{14} \\ c_{15} \\ c_{16} \\ c_{17} \\ c_{18} \\ c_{19}
\end{array}
\end{array}
$$

Figure 2.1: The parity-check matrix of a $[25, 5, 8]_2$ LDPC code

Classical linear block codes are generally decoded using maximum likelihood (ML)-like decoding algorithms, consequently they are usually short and designed algebraically to minimize decoding complexity. However, LDPC codes are generally decoded iteratively using a graphical representation of their parity-check matrix and for this reason are designed with a focus on the properties of H [Johnson, 2006].

LDPC codes are iteratively decoded using the sum-product decoder [Gallager, 1963] or equivalently the belief propagation decoder [Pearl, 1988]. Due to their similarities, these two algorithms are collectively referred to as the *belief propagation / sum-product algorithm* (BP/SPA) decoder. Many researchers have shown that under BP/SPA decoding, long block length LDPC codes perform close to the channel capacity, e.g. [Chung et al., 2001]. The theory of LDPC codes is closely affiliated to the

branch of mathematics known as graph theory. As a result of work done by Tanner [Tanner, 1981], LDPC codes are often represented in graphical form by *bipartite* or Tanner graphs. Similar to the rows and columns in LDPC code parity-check matrices, the Tanner graph of LDPC codes consist of two sets of vertices: n vertices for the codeword symbols (called *symbol nodes*) and m vertices for the parity-check equations (called *check nodes*). An *edge* joins a symbol node to a check node if the symbol is included in the corresponding parity-check equation. Therefore, the number of edges in a Tanner graph is equal to the number of non-zero elements in the parity-check matrix. The following are some basic definitions and notations on graph theory and LDPC codes, some of which were adopted from [Tjhai, 2007].

2.1.1 Vertex, Edge, Bipartite or Tanner Graph, Subgraphs, Adjacency and Incidence, and Simple Graphs

A graph is made up of an ordered set of vertices and edges, and is denoted by $G(V, E)$.

- **Vertex:** A vertex represents a unique node in a graph. A node is basic unit used to build graphs and other data structures. The set $V(G)$ is made up of all the vertices in graph $G(V, E)$. Therefore, if v is a vertex of $G(V, E)$ then $v \in V(G)$. The number of vertices in $V(G)$ is denoted as $|V(G)|$. In an LDPC code Tanner graph there is a distinct symbol vertex for each of the n symbols, and a distinct parity-check vertex for each of the m parity-check equations. By convention, a symbol vertex in an LDPC code graph (which is a bipartite graph) is drawn as a round node (○/●), and a parity-check vertex (or simply *check vertex*) is drawn as a square node (□/■).

- **Edge:** An edge (u, v) is a direct connection between two vertices in graph $G(V, E)$, where $u \in V(G)$ and $v \in V(G)$. An edge in a graph is usually drawn as a line connecting vertices u and v in the graph. The set of edges in $G(V, E)$ is denoted by $E(G)$. Set $E(G)$ contains pairs of elements of $V(G)$, i.e. $E(G) = \{(u, v)$ where $u, v \in V(G)\}$.

- **Bipartite or Tanner Graph:** In a bipartite or Tanner graph $G(V, E)$ there are two disjoint sets of vertices, $V_s(G)$ and $V_c(G)$, such that $V(G) = V_s(G) \cup V_c(G)$. In an LDPC code Tanner graph, $V_s(G)$ is the set of symbol node vertices, and $V_c(G)$ is the set of parity-check node vertices. The edges in LDPC code Tanner graphs are made up exclusively of connections between one vertex in $V_s(G)$ and another vertex in $V_c(G)$, so that edge $(c_i, s_j) \in E(G)$, where $c_i \in V_c(G)$ and $s_j \in V_s(G)$ for some integers i and j.

  Figure 2.2 shows the Tanner graph $G(V, E)$ for the $[n, k, d]_q = [25, 5, 8]_2$ LDPC code whose parity-check matrix is shown in Figure 2.1. Each line drawn between the two sets of vertices is an edge which represents a '1' in the parity-check matrix.

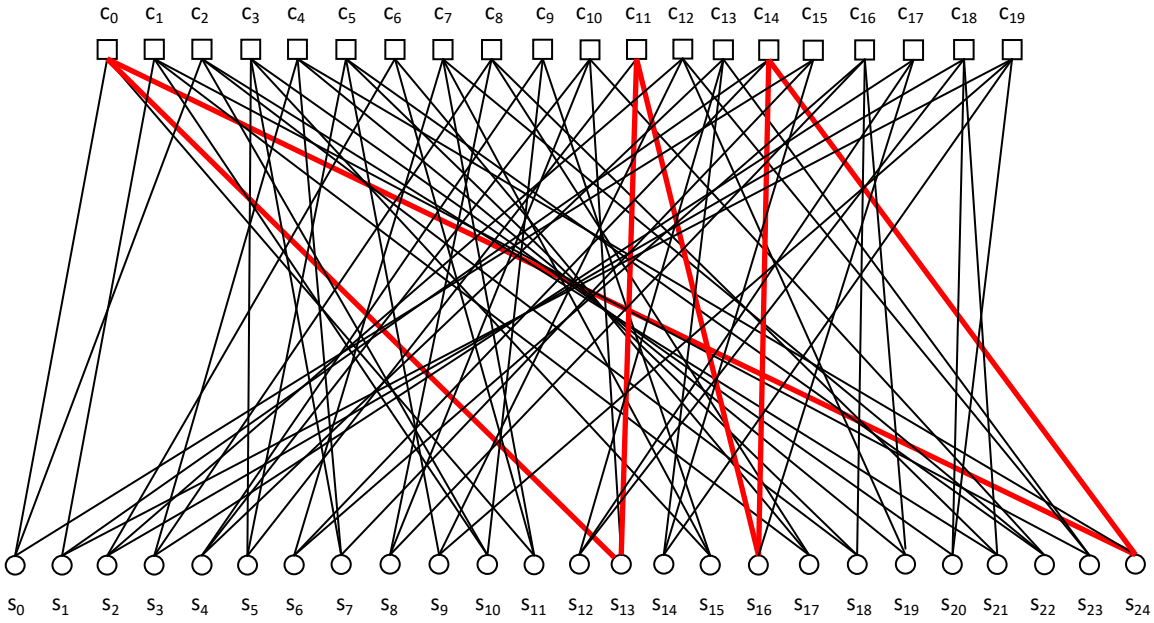

Figure 2.2: The Tanner graph of a $[25, 5, 8]_2$ LDPC code

The parity-check matrix H of a full row rank LDPC code (i.e. when H is devoid of linearly dependent parity-check equations) consists of $|V_c(G)| = n - k$ rows and $|V_s(G)| = n$ columns. Similarly, the Tanner graph of the code consists of $|V_c(G)| = n - k$ distinct check vertices and $|V_s(G)| = n$ distinct symbol vertices. Let $V_s(G) = \{s_0, s_1, \ldots, s_{n-1}\}$ and $V_c(G) = \{c_0, c_1, \ldots, c_{n-k-1}\}$, it can be observed from Figure 2.1 and Figure 2.2

that for each edge $(c_i, s_j) \in E(G)$, the element in the i-th row and j-th column of the matrix $\boldsymbol{H}$ is non-zero, i.e. $H_{i,j} \neq 0$ for $0 \leq i \leq n-k-1$ and $0 \leq j \leq n-1$.

- **Subgraph**: A subgraph $G' = (V', E')$ of Tanner graph $G(V, E)$ is an interconnected subset of G, so that for every edge $e \in E'$ both of the vertices of e lie in the set V'.
- **Adjacency and Incidence:** If $u, v \in V(G)$ and $(u, v) \in E(G)$, then vertices u and v are adjacent or *neighbouring* vertices of $G(V, E)$. Similarly, for any $(u, v) \in E(G)$ vertex u and vertex v are said to be incident with edge (u, v). As an example, in the bipartite graph of Figure 2.2, the check vertex c_0 and the symbol vertex s_{24} are adjacent vertices. Vertex c_0 and vertex s_{24} are incident with edge (c_0, s_{24}).
- **Simple Graphs:** A graph is simple if: i) it does not have any *self-loop*, i.e. an edge which joins a vertex to itself, ii) no unique pair of vertices have more than one edge between them, and iii) the graph is made up of nondirected edges exclusively. In a simple graph any two distinct vertices x and y are said to be *adjacent* if (x, y) is an edge, and the *neighbours* of vertex y is the set of all its adjacent vertices. LDPC code Tanner graphs are simple graphs.

2.1.2 Path, Diameter, Cycle, Local Girth and Global Girth

- **Path:** A path is a sequence of edges which connect distinct vertices. If $u, z \in V(G)$, a path between vertices u and z is a sequence of edges that connect distinct vertices which start at vertex u and end at vertex z. The *length* of a path between vertex u and vertex z is equal to the number of edges in that particular path between u and z, and the *distance* $d(u, z)$ between vertex u and vertex z is defined as the length of the shortest path joining them. Every edge $(u, v) \in E(G)$ in a graph $G(V, E)$ represents a path of length one (1) between vertex u and vertex v. As an example, a path between symbol vertex s_{13} and check vertex c_{14} in the graph of Figure 2.2 is: $s_{13} \rightarrow c_{11} \rightarrow s_{16} \rightarrow c_{14}$. Therefore, vertices s_{13} and c_{14} are indirectly *connected* through this path, or c_{14} is *reached* by s_{13} and vice versa. The length of this path is 3.

- **Diameter:** The *diameter* of a graph is defined as the maximum distance between distinct vertex pairs.
- **Cycle:** A cycle in a graph $G(V, E)$ is a closed path formed by edges which start and end at the same vertex in the graph; each vertex in a cycle is traversed only once. The length of a cycle is equal to the number of edges it contains. In a bipartite graph, half of the vertices in each cycle belong to set $V_s(G)$ and the other half belong to set $V_c(G)$. As an example, the edges of a cycle in the bipartite graph of Figure 2.2 are highlighted in red. This cycle can be described as: $s_{13} \rightarrow c_0 \rightarrow s_{24} \rightarrow c_{14} \rightarrow s_{16} \rightarrow c_{11} \rightarrow s_{13}$. A count of the number of edges in the cycle, i.e. the red lines in Figure 2.2, shows that this cycle has a girth of 6.
- **Local Girth:** The local girth at a vertex $v \in V(G)$ is the length of the shortest cycle which passes through v.
- **Global Girth:** The global girth (or just *girth*) of a graph $G(V, E)$ is the length of the shortest cycle in the entire graph. In other words, the girth of a graph $G(V, E)$ is the length of its shortest local girth.

2.1.3 Degree, Degree Sequence and Degree Distribution

- **Degree:** The degree of any vertex $u \in V(G)$ is equal to the number of edges that are directly connected to vertex u. As an example, if the edges directly connected to vertex u in *G(V, E)* are $(u, y_1), (u, y_2), (u, y_3), (u, y_4), (u, y_5) \in E(G)$, where $y_1, y_2, y_3\ y_4, y_5 \in V(G)$ and $y_i \neq y_j,\ \forall\ i \neq j$, then the degree of vertex u is 5. All the symbol vertices $s_j, \forall\ 0 \leq j \leq 24$, in the bipartite graph in Figure 2.2 have degrees of 3; i.e. $d_{s_j} = 3$ for all $0 \leq j \leq 24$. However, while check vertices c_6, c_9, c_{11}, c_{15}, c_{17} have degrees of 3, i.e. $d_{c_i} = 3$ for $i = 6, 9, 11, 15$ & 17, all the remaining check vertices have degrees of 4, i.e. $d_{c_{i'}} = 4$ for $0 \leq \{i' \neq i\} \leq 19$.
- **Degree Sequence:** In an irregular LDPC code Tanner graph, the symbol degree sequence is denoted by $D_s = \{d_{s_0}, d_{s_1}, \ldots\ldots\ldots, d_{s_{n-1}}\}$, where d_{s_j} is the degree of symbol node s_j, $0 \leq j \leq n-1$. Irregular LDPC codes are usually constructed with their symbol degree

sequences specified in the nondecreasing order, so that $d_{s_0} \leq d_{s_1} \cdots \leq d_{s_{n-1}}$. Similarly, the check degree sequence of an irregular LDPC code is denoted by $D_c = \{d_{c_0}, d_{c_1}, \ldots\ldots\ldots, d_{c_{m-1}}\}$, where d_{c_i} is the degree of check node c_i, $0 \leq i \leq m-1$.

The symbol degree sequence for the code shown in Figure 2.1 and Figure 2.2 is given as $D_s = \{3, 3, \ldots, 3\}$ and the check node degree is $D_c = \{4,4,4,4,4,4,3,4,4,3,4,3,4,4,4,3,4,3,4,4\}$.

- **Degree Distribution:** The symbol degree distribution in an irregular LDPC code Tanner graph $G(V, E)$ is often expressed using a polynomial $\lambda(x) = \sum_{i\geq 2}^{d_{s_{max}}} \lambda_i x^i$, where λ_i is the fraction of symbol vertices of degree i, $d_{s_{max}}$ is the highest symbol degree in $G(V, E)$, and $\sum_{i\geq 2}^{d_{s_{max}}} \lambda_i = 1$. Similarly, the check degree distribution is given by $\Phi(x) = \sum_{i\geq 2}^{d_{c_{max}}} \Phi_i x^i$, where Φ_i is the fraction of check vertices of degree i, $d_{c_{max}}$ is the highest check degree in $G(V, E)$, and $\sum_{i\geq 2}^{d_{c_{max}}} \Phi_i = 1$. [Hu et al., 2005]

  To obtain the symbol degree distribution of the code in Figure 2.1 and Figure 2.2 we observe that all the symbol nodes have a degree 3, consequently the fraction of symbol vertices which have degree 3 is calculated as $n/n = 1.00$. Therefore, the symbol degree distribution of this code is simply: $\lambda(x) = 1.00x^3$.

  Similarly, the check degree distribution is determined from the degree sequence for the code as follows. We observe that the check degrees in the code are 3 and 4, the fraction of check vertices which have degree 3 is $5/m = 5/20 = 0.25$ and the fraction of check vertices which have degree 4 is $15/m = 15/20 = 0.75$. Therefore, the check degree distribution of this code is: $\Phi(x) = 0.25x^3 + 0.75x^4$.

  We note that $\lambda(x)$ and $\Phi(x)$ are known as *vertex-oriented degree sequences*. An alternative representation of degree sequences for irregular LDPC codes is known as *edge-oriented degree sequences*, which consider the fraction of edges that are connected to vertices of different degrees. In this thesis, only vertex-oriented degree sequences are considered.

2.2 LDPC Code Performance Curves

By convention, the performance curve of an LDPC code and other iteratively decodable codes is demarcated into three distinct regions. These regions are the erroneous region, the waterfall region and the error-floor region. The performance of a typical LDPC code is shown in Figure 2.3. The figure shows a plot of frame error rate (FER) against E_b/N_o.

In the 'erroneous region', which occurs at low E_b/N_o, the iterative decoder is unable to correctly decode almost all of the transmitted codewords.

The 'waterfall region' describes the region of the performance curve where the error rate of the iterative decoder decreases rapidly with increasing E_b/N_o. The E_b/N_o value at which the waterfall region commences is known as the *convergence threshold*. The convergence threshold is defined as the Eb/No value from which the coded system becomes more efficient than the non-coded transmission system. As can be seen in Figure 2.3, the slope of the performance curve to the left of the convergence threshold is notably less than the slope of the performance curve to the right of the convergence threshold.

As the signal power is increased further, the rate of decrease in error rate with increasing E_b/N_o diminishes and the slope of the error rate curve starts to reduce. This reduction in the slope of the error rate curve at relatively high E_b/N_o introduces a so-called 'error-floor' in the FER curve; this is the 'error-floor region' of the performance curve of LDPC codes. As can be observed from Figure 2.3, the description of this region of the performance curve as an error-floor is a misnomer because the slope of the curve in this region continues to be significantly greater than zero. That is, the error rate curve does not flatten sufficiently enough to be described as a floor. However, in keeping with the convention, this region of the performance curve is described as the error-floor region in the rest of this thesis.

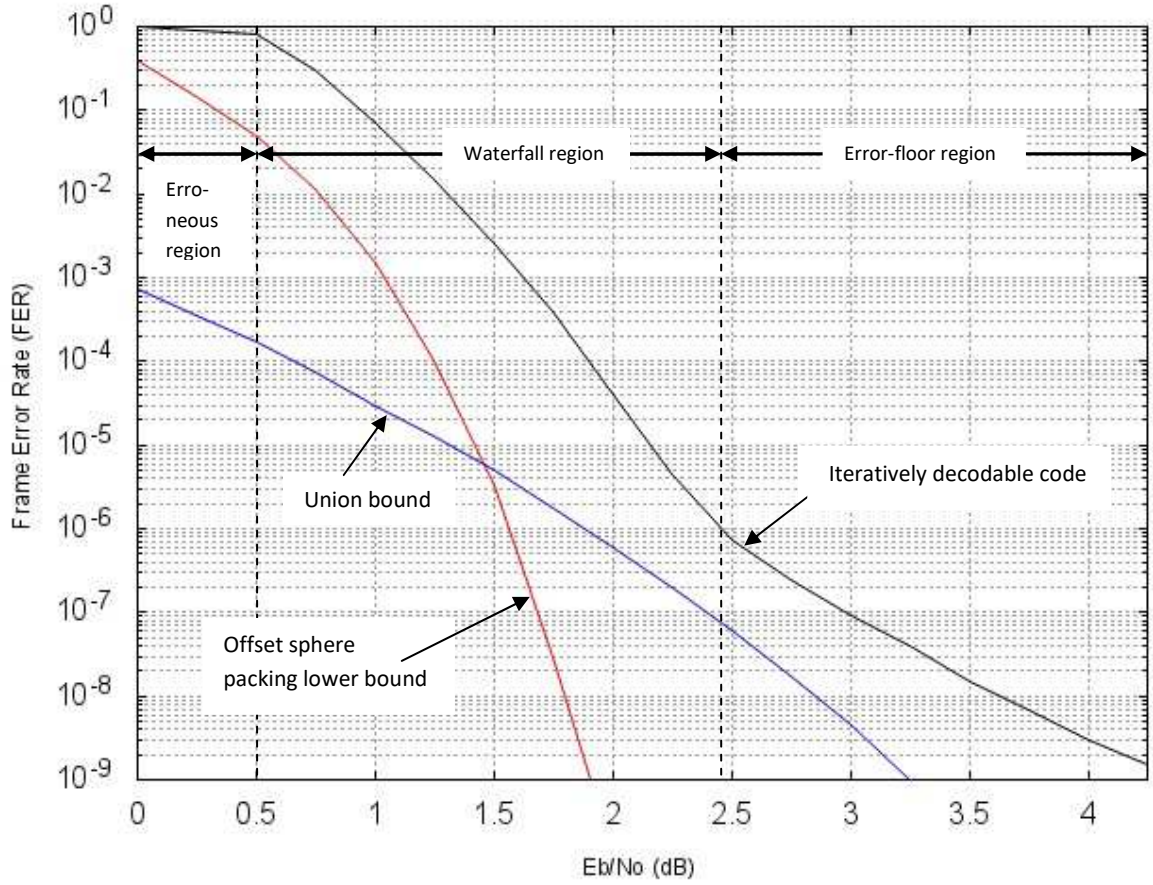

Figure 2.3: Typical FER performance of iteratively decodable codes over AWGN channels [Tjhai, 2007]

Two additional error rate curves, the *offset sphere packing lower bound* and the probability of error based on the *union bound argument* [Proakis, 2001], have also been included in Figure 2.3. The sphere packing lower bound represents the performance of an ideal coding system for identical code length and rate; the performance to the left of this lower bound is unattainable, and the performance to the right of this lower bound may be achieved by some coding and decoding schemes.

The union bound curve is an upper bound on the practical error rate of the code; the existence and quantification of this upper bound are mainly due to the presence of low Hamming weight codewords and the number of these low weight codewords, respectively. Generally, the larger the minimum Hamming weight (or distance) of a code, the lower the union bound curve. For iteratively decodable codes with poor (low value) minimum Hamming distances, the intersection between the union bound curve and the offset sphere packing lower bound curve occurs at relatively low E_b/N_o [Proakis, 2001].

As shown for the typical LDPC code performance curve in Figure 2.3, it is expected that even when using an ideal soft-decision decoder, such as the maximum likelihood (ML) decoder, the performance of a coding system would follow the curvature (albeit displaced to the right) of the sphere packing lower bound at the lower values of E_b/N_o in the waterfall region. However, at higher values of E_b/N_o, an error-floor would appear in the performance curve as a result of the limitations of decoding the minimum weight codewords. In Figure 2.3, it can be seen that there is a significant gap between the union bound and the error-floor of a typical LDPC code under iterative decoding. This behaviour is accepted as part and parcel of iteratively decoded codes and is usually attributed to the limitations of the iterative decoder. Apart from the minimum Hamming distance, there are many other causes of error which prevent iterative decoders from reaching the union bound. [Tjhai, 2007]

2.3 LDPC Code Constructions

There are two broad categories of LDPC codes based on their construction technique: random LDPC codes and algebraic LDPC codes. Codes constructed using either of these techniques can be further subcategorized as regular or irregular, depending on the structure of the parity-check matrix; or as binary or non-binary, depending on q - the field size of the elements in the parity-check matrix $\mathbb{F}_q$.

2.3.1 Random LDPC Codes

LDPC codes are often constructed pseudo-randomly. Consequently, discussions often center around sets (or *ensembles*) of codes with certain parameters, e.g. a certain construction method or/and a certain degree distribution, rather than about a particular parity-check matrix with those parameters. The *concentration theorem* says that the properties of randomly chosen codes from a code ensemble, concentrates around the ensemble average. For long codes, there is a high probability that a randomly chosen parity-check matrix would result in a good code; for very long codes this is guaranteed by the concentration theorem. The following are common random constructions of LDPC codes.

Gallager's construction

The LDPC codes introduced by Gallager in 1962 [Gallager, 1962] were random binary LDPC codes. Gallager denoted the parity-check matrices of his LDPC codes as $(n,\ j,\ k)$, where n is the block length or number of columns in the matrix, j is number of 1's per column, and k is the number of 1's per row. The short notation $(j,\ k)$ is often used for these codes and other regular codes, so that they are described as $(j,\ k)$-regular. Note that the definitions for variables *j* and *k* used here apply only to this description of Gallager's LDPC codes and should not be confused with their definitions elsewhere in this thesis. The code-rate of a Gallager $(j,\ k)$ code is given by: $R = 1 - {}^{j}/_{k}$.

An example of a low-density code matrix from [Gallager, 1962] is shown in Figure 2.4. Using Gallager's notation, this is an $(n,\ j,\ k) = (20, 3, 4)$ LDPC code with $j = 3$, and $k = 4$. Using the standard notation for linear codes, this is an $[n,\ k,\ d]_q = [20,\ 5,\ 4]_2$ code, where k in this notation is the number of information bits in each codeword block.

$$H = \begin{array}{c} \begin{matrix} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_{19} \end{matrix} \\ \left[\begin{array}{cccccccccccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \begin{matrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \\ \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{14} \end{matrix} \end{array}$$

Figure 2.4: An $(n,\ j,\ k) = (20, 3, 4)$ Gallager LDPC code matrix

The original LDPC codes introduced by Gallager are regular codes because in the parity-check matrix of an (n, j, k) Gallager code there is always a fixed number of non-zeros per column and per row, i.e. j and k respectively. In the Gallager construction, the matrix in Figure 2.4 is divided into j submatrices and each of these submatrices contains a single '1' in each column. The horizontal lines drawn in the matrix shown in Figure 2.4 are used to demarcate its $j = 3$ submatrices. The first submatrix contains all its 1's in the descending order, such that row c_i contains 1's in columns s_{ik} to $s_{(i+1)k-1}$, for $0 \leq i \leq k$. The remaining $j - 1$ submatrices are merely column permutations of the first submatrix. An ensemble of (n, j, k) Gallager codes is defined as the ensemble resulting from random permutation of the columns of the bottom $j - 1$ submatrices of a matrix such as that in Figure 2.4, with equal probability assigned to each permutation. Using this method, there is a high probability that some rows in the matrices constructed will be linearly dependent. The presence of linearly dependent parity-check equations in the parity-check matrix of a code simply means that the code specified by the parity-check matrix has a slightly higher information rate than indicated by the number of rows in the parity-check matrix constructed. A smaller number of linearly independent parity-check equations implies a smaller number of parity-check bits and a higher number of information bits per codeword; this translates to a higher code rate. This construction method is not efficiently scalable, and the performance of short block length Gallager codes is poor relative to those for turbo codes and other algebraic codes for the same code rate and length.

MacKay and Neal construction

A random construction method for LDPC codes was proposed in [MacKay and Neal, 1995]. In the proposed method, the columns of the parity-check matrix H are added one column at a time from left to right. The number of non-zero elements in each column and each row of H, i.e. the column and row weight, is pre-determined by the symbol and check degree distributions, respectively. During code construction, the placement of the non-zero entries in each column is chosen randomly from those rows which do not have the full number of non-zero entries. At any point during the construction, if there are rows with more non-zero positions unfilled than there are columns remaining to be added,

the check degree distributions for H will not be as prescribed. In order to correct this, the process may be restarted or backtracked by several columns. Figure 2.5 shows a length 20 (3, 4)-regular Mackay Neal LDPC code matrix. Using the standard notation for linear codes, this code is similar to the Gallager code in Figure 2.4 and is also a $[20, 5, 4]_2$ code.

$$H = \begin{array}{c} \begin{matrix} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_{19} \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{14} \end{matrix} \end{array}$$

Figure 2.5: A length 20 (3, 4)-regular MacKay Neal LDPC code matrix

When adding the column for symbol s_{18}, i.e. the 19th column which is shown in bold font, the unfilled rows were the 5th, 8th, 9th, 10th, 12th and 13th (c_4, c_7, c_8, c_9, c_{11} and c_{12}) out of which the 5th, 9th and 13th (c_4, c_8, and c_{12}) were randomly chosen.

Cycles of length 4, called *4-cycles*, can be avoided in LDPC codes constructed using MacKay and Neal method by ensuring that the non-zero elements in the new column being added do not overlap in two or more places with the non-zero elements in any of the previous columns. As an example, if a 4-cycle free graph was required for the length 12 (3, 4)-regular Mackay Neal LDPC code parity-check matrix shown in Figure 2.6, the fourth column (for symbol node s_3) would have been modified because it causes a 4-cycle with the second column (for symbol node s_1) column in H. The

non-zero elements which result in the 4-cycle of interest are shown in bold; there are several more 4-cycles in this code matrix. It may be observed that the creation of 4-cycles was avoided in the code matrix in Figure 2.5.

$$H = \begin{array}{c} \begin{matrix} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} \\ c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{matrix}$$

Figure 2.6: A 4-cycle in a length 12 (3, 4)-regular MacKay Neal LDPC code matrix

Repeat-accumulate code construction

Another class of pseudo-randomly constructed LDPC codes are the repeat-accumulate (RA) codes which were introduced in [Divsalar et al., 1998]. Parity-check matrices of RA codes have columns of weight two (2) which are arranged in a step pattern for the last m columns of H (Recall: For an $[n, k, d]_q$ linear code, $m = n - k$). This pattern makes RA codes systematic codes and allows them to be easily encoded. Figure 2.7 shows a length 20 rate-$1/4$ repeat-accumulate code matrix, the first five columns of the matrix correspond to the message bits. The first parity-bit, i.e symbol s_5, which is in the sixth column of H, is encoded as $s_5 = s_0$, the second parity-bit is encoded as $s_6 = s_5 \oplus s_0$, the next as $s_7 = s_6 \oplus s_1$, and so on. This forms a pattern, such that each parity-bit can be computed one after the other using only the message bits and the one previously calculated parity-bit. [Johnson, 2006]

$$
H = \begin{array}{c}
\begin{array}{cccccccccccccccccccc}
s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_{19}
\end{array} \\
\left[\begin{array}{cccccccccccccccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right]
\begin{array}{c}
c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{14}
\end{array}
\end{array}
$$

Figure 2.7: A length 20 rate-1/4 repeat-accumulate code matrix

Irregular repeat-accumulate (IRA) codes were introduced by [Jin et al., 2000] and are a generalization of the ideas behind RA codes. Like RA codes, IRA codes have a simple linear-time encoding algorithm. IRA codes have good performance, but generally perform better on binary erasure channels (BEC) than on AWGN channels.

The code construction methods discussed above are mainly focussed on the parity-check matrices. Following the rediscovery of LDPC codes in 1996, and based on the work by Tanner [Tanner, 1981], research on graph based encoding and decoding of LDPC codes were carried out by many researchers and significant successes were recorded. Some of these research include work by [Wiberg et al., 1995], [Sipser and Spielman, 1996], [McEliece et al., 1998], [Kschischang and Frey, 1998], [Köetter and Vardy, 1998], [Kschischang et al., 2001], [Calderbank et al., 1999], [Weiss, 2000], [Hu et al. 2001] and [Hu et al. 2005]. Common graph based LDPC code constructions are as follows.

Column splitting and row splitting

Short cycles and other undesirable configurations in the Tanner graph of LDPC codes can be removed by splitting a column or row in the parity-check matrix in half. In column splitting a column in H is replaced by two columns which share the non-zero entries of the original column between them. As a result of the extra column due to a column split, a new code with length one greater than the original code is produced; the new matrix is also more sparse than the original matrix. Figure 2.8 shows a column splitting operation used to remove a 4-cycle.

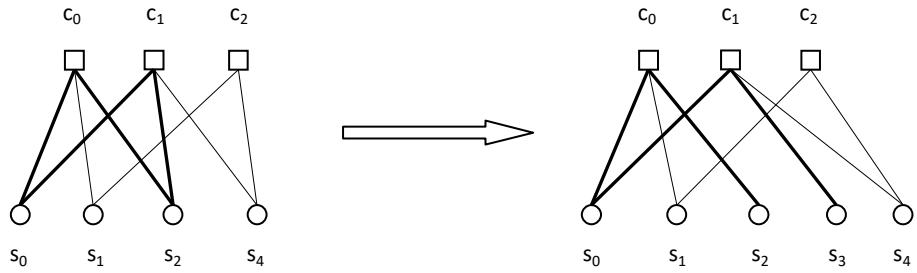

Figure 2.8: Column splitting to eliminate a 4-cycle

Similarly, in row splitting a row in H is replaced by two rows which share the non-zero entities of the original row between them. As a result of the extra row due to a row split, a new code with one more parity-check equation than the original code is produced; the new check matrix is also more sparse than the original. Figure 2.9 shows a row splitting operation used to remove a 4-cycle.

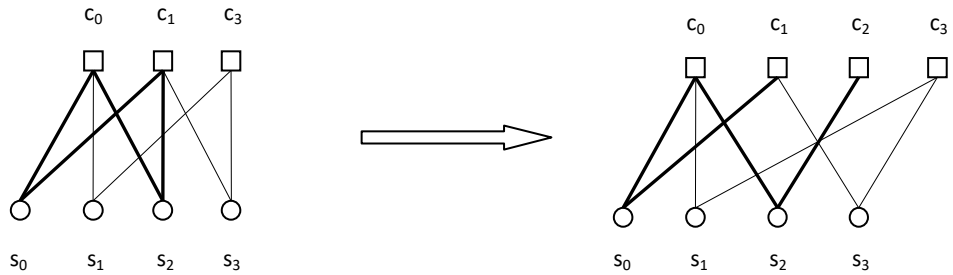

Figure 2.9: Row splitting to eliminate a 4-cycle

Bit filling and the progressive edge-growth (PEG) Tanner graphs

The bit filling method for constructing LDPC codes was introduced in [Campello et al., 2001]. In bit filling, symbol nodes are connected into a Tanner graph one at a time and the edges connecting the new symbol nodes to the graph are carefully selected to avoid cycles of girth g. In an irregular code, for each new symbol node s_j $(0 \leq j \leq n-1)$, d_{s_j} check nodes are selected to join s_j by an edge, where d_{s_j} is the degree of symbol node s_j as obtained from the degree sequence for the code $D_s = \{d_{s_0}, d_{s_1}, \ldots\ldots\ldots, d_{s_{n-1}}\}$. Similarly, in a regular code with a column weight of w_c, i.e. a fixed number of 1's per column, w_c check nodes are selected to join each new symbol node s_j by an edge.

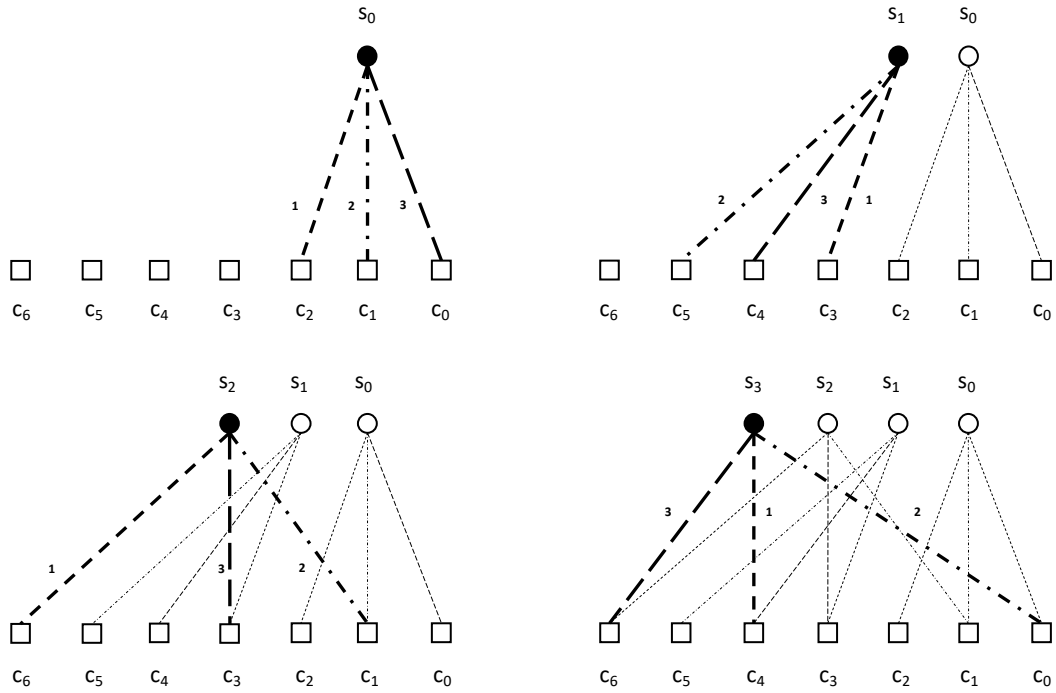

Figure 2.10: Bit filling to avoid 4-cycles

In order to satisfy the restriction on the girth g of a code, the set of eligible check nodes for connecting new edges at each stage of the code construction are those that are at distances

more than or equal to $g/2$ from all the check nodes already connected to s_j. Figure 2.10 illustrates the bit filling process to avoid cycles of length 4.

The progressive edge-growth (PEG) Tanner graph construction was introduced by [Hu et al., 2001]. The PEG construction is similar to bit filling and edges are connected into graph one at a time progressively, but instead of satisfying a specific girth restriction g, each new edge of s_j, i.e. the current symbol node being added to the graph, is simply connected in a way which maximizes the local girth at symbol s_j. PEG Tanner graph constructions are used extensively in this thesis and a detailed description of the original/standard PEG algorithm is given in Section 2.4.

2.3.2 Algebraic LDPC Codes

Algebraically constructed LDPC codes generally have a regular structure in their parity-check matrices. The following are some advantages of algebraically constructed LDPC codes over their randomly constructed counterparts.

i. The presence of regular structures in the parity-check matrices of algebraic LDPC codes allow for simple encoding schemes which make them attractive for deployment in practical systems. An algebraic code such as a cyclic or quasi-cyclic LDPC code can be completely described by a polynomial and message encoding may be easily achieved using a linear-feedback shift-register circuit which requires minimum memory. In contrast, prior to encoding messages using a code with a randomly constructed parity-check matrix, Gaussian elimination and other matrix operations have to be carried out on the parity-check matrix in order to obtain a codeword *generator matrix*, and the entire generator matrix has to be stored in memory.

ii. Important LDPC code parameters such as girth and minimum distance can be easily determined and, when this is not feasible, lower- and upper-bounds which are more accurate than those for random codes may be mathematically derived.

iii. In general, algebraic LDPC codes have larger minimum distances than those for their randomly constructed counterparts. Consequently, when decoded using non iterative decoding schemes, algebraic LDPC codes are less likely to have early error-floor.

iv. Algebraically constructed cyclic LDPC codes have n low weight parity-check equations, whereas, equivalent randomly constructed LDPC codes have $m = n - k$ parity-check equations. Therefore, cyclic LDPC codes have k extra parity-check equations for iterations in the iterative decoder and this invariably leads to superior performance [Tjhai et al., 2006].

Algebraically constructed codes like the difference-set cyclic (DSC) codes [Weldon, Jr., 1966] and one-step majority-logic decodable (OSMLD) codes [Lin and Costello, Jr, 2004] were shown to have significantly improved performance under iterative decoding compared to the majority-logic decoding which they were conventionally decoded with [Lucas et al., 2000]. These codes have sparse parity-check matrices and were subsequently reclassified as algebraic LDPC codes. A noteworthy work on LDPC code construction, before they were formally rediscovered in 1996, was by [Margulis, 1982] which is an explicit algebraic approach based on Ramanujan graphs.

Some cyclic code constructions which are suitable for iterative decoding include: binary cyclic LDPC codes derived from cyclotomic cosets, which is based on work by [MacWilliams and Sloan, 1997] on Mattson-Solomon polynomials, idempotents and cyclotomic cosets; Mattson-Solomon domain construction of binary cyclic LDPC codes, in which binary cyclic LDPC codes are obtained by working in the Mattson-Solomon domain instead of deriving them from cyclotomic cosets; and a non-binary extension of the cyclotomic coset based LDPC codes. [Tjhai, 2007]

Finite geometry (FG) codes are a class of algebraic LDPC codes which are based on Euclidean and projective geometries. FG constructions are known to produce LDPC codes which have relatively good minimum distances, contain no small trapping sets with sizes smaller than its minimum distance,

and have Tanner graphs which do not contain 4-cycles. Consequently, their error-floor performance are primarily determined by their minimum distance properties. FG LDPC codes constructed using high code rates and very long block lengths have performance that are very near Shannon limit under iterative decoding, these codes are discussed in detail in [Kou et al., 2001]. Another class of algebraic LDPC codes are those constructed using combinatorial techniques, see [Johnson, 2003], [Johnson and Weller, 2001], [Johnson and Weller, 2002], and [Vasic and Milenkovic, 2004].

As a result of the simplicity of encoding and decoding algebraically constructed LDPC codes, they were adopted in several industry standards [Costello and Forney, 2007].

2.3.3 Non-Binary LDPC Codes

The parity-check matrix of every LDPC code is defined on a finite-field of q elements $\mathbb{F}_q$, most LDPC codes are implemented in the binary domain $\mathbb{F}_2$, i.e. with $q = 2$, and symbols values are either 0 or 1. However, LDPC codes may be easily extended so that symbols take values from finite-field $\mathbb{F}_{2^m}$. Work in this area was spearheaded in [Davey and MacKay, 1998]. Based on the parity-check matrix H of an LDPC code over $\mathbb{F}_2$, an LDPC code over $\mathbb{F}_{2^m}$ with $m \geq 2$ is constructed by simply substituting all the 1's in H with the non-zero elements of $\mathbb{F}_{2^m}$, either randomly or in a structured fashion. Non-binary LDPC codes have better convergence performance under iterative decoding, and [Davey and MacKay, 1998] and [Hu et al., 2005] showed that the performance of LDPC codes can be improved by operating beyond the binary domain. The improved performance is attributed to an improvement in the girth or/and local girth distribution in the Tanner graph in the non-binary arrangement.

2.4 The Progressive Edge-Growth (PEG) Algorithm

The standard progressive edge growth (PEG) algorithm is a general, non-algebraic, method of using graph expansion to construct LDPC code Tanner graphs with large girths. The algorithm was first proposed by Hu, X.-Y., Eleftheriou, E. and Arnold D. M. in 2001 [Hu et al., 2001]. In a PEG Tanner graph construction, edges between symbol nodes and check nodes are connected into the graph one

after the other, i.e. progressively, until the graph is complete. The PEG algorithm uses graph expansions to select and place edges in a Tanner graph in a manner which ensures that each consecutive edge placed has the smallest possible impact on the girth of the graph. LDPC codes constructed using the PEG principle have good girth and minimum distance properties. In their pioneering work, Hu et al. used simulations to show that the PEG algorithm is a powerful tool for generating short-block-length LDPC codes which performed significantly better than randomly constructed codes of equivalent length and code rate.

There are two key features of the PEG algorithm compared with other constructions, the first is its simplicity or low-complexity; it can easily be used to construct regular and irregular LDPC codes within the short-block-length to long-block-length range with good girth and minimum distance properties. The computational complexity and storage requirement of the PEG algorithm are significantly less than those for Gallager's explicit construction. The second, and equally important, key feature of the PEG algorithm is its flexibility; using density-evolution (DE) optimized degree sequence [Richardson and Urbanke, 2001] , [Richardson et al., 2001], the PEG algorithm successfully generates good LDPC codes at any code rate [Hu et al., 2005]. More importantly, the PEG algorithm lends itself to easy modifications to enhance one or more of the features of the LDPC codes constructed in a bid to improve coding performance, or simply to tailor a feature of the code to some requirements. For example, the PEG algorithm can be easily modified to generate *linear-time* encodable codes, and by another slight modification, LDPC codes which are strictly regular can be produced. Researchers into LDPC codes have modified the PEG algorithm with many different criteria for check node selection when establishing edges for the symbols node being connected into the graph during construction. As an example, H. Xiao and A. H. Banihashemi took advantage of the flexibility of the PEG algorithm and the *approximate cycle extrinsic* (ACE) message degree criterion was implemented in the PEG algorithm to improve the interconnectivity of cycles in the Tanner graph of the LDPC codes [Xiao and Banihashemi, 2004]. Consequently, the gap to capacity of short length LDPC codes in the error-floor region was reduced significantly.

For consistency and uniformity, we adopt the same definitions and notations as in [Hu et al., 2001] [Hu et al., 2005] to describe the standard PEG algorithm for constructing an LDPC code matrices H with m rows and n columns. Some of the definitions and notations which follow have been given previously in this thesis and are reiterated.

The dimensions of a parity-check matrix H is $m \times n$, where m is the number of parity-check equations, and n is the number of symbols or block length of the code described by the matrix. A bipartite/Tanner graph with *m* parity-check vertices and n symbol vertices can be created using H as the incidence matrix for the two sets of vertices. It therefore follows that a unique parity-check matrix corresponds to a unique Tanner graph and vice versa. Consequently, the terms *bipartite graph*, *Tanner graph*, *parity-check matrix*, *LDPC code*, or just *codes* are frequently used interchangeably.

A Tanner graph, by convention, is denoted as $G(V, E)$, where V is the set of parity-check and symbol nodes, so that $V = V_s \cup V_c$ where $V_s = \{s_j\}$ for all $0 \leq j \leq n-1$, is the set of symbol nodes, and $V_c = \{c_i\}$ for all $0 \leq i \leq m-1$, is the set of parity-check nodes. E is the set of edges or connections between parity-check nodes and symbol nodes in the graph. Therefore, $E \subseteq V_s \times V_c$, with edge $(c_i,\ s_j) \in E$ if and only if $h_{i,j} \neq 0$, where $h_{i,j}$ is an entry of H at the i-th row and the j-th column, for all $0 \leq i \leq m-1$ and $0 \leq j \leq n-1$. The set of edges E can be partitioned in terms of V_s as $E = E_{s_0} \cup E_{s_1} \cup \ldots \cup E_{s_{n-1}}$, with the entire edges incident on symbol node s_j contained in E_{s_j}. The $(k+1)$-th edge incident on s_j is denoted as E_{sj}^{k}, for all $0 \leq k \leq d_{s_j} - 1$ where d_{s_j} is the degree of symbol node s_j.

Ensembles of LDPC code Tanner graphs are often characterized using their symbol node degree distributions only. The symbol-node degree distribution of an LDPC code is defined as $\lambda(x) = \sum_{i\geq 2}^{d_{s_{max}}} \lambda_i x^{\mathrm{i}}$, where λ_i is the fraction of symbol-nodes connected to exactly i check nodes; $d_{s_{max}}$ is the largest symbol node degree in the degree sequence $D_s = \{d_{s_0}, d_{s_1}, \ldots\ldots\ldots, d_{s_{n-1}}\}$; and $\sum_{i\geq 2}^{d_{s_{max}}} \lambda_i = 1$, i.e. the sum of all the fractions λ_i, for all $2 \leq i \leq d_{s_{max}}$, in the symbol node degree

distribution is unity. Figure 2.11 shows an irregular Tanner graph with symbol node degree sequence $D_s = \{2,\ 2,\ 2,\ 3,\ 3,\ 3\}$ which has a uniform (regular) parity-check degree of 5, i.e., $D_c = \{5,\ 5,\ 5\}$.

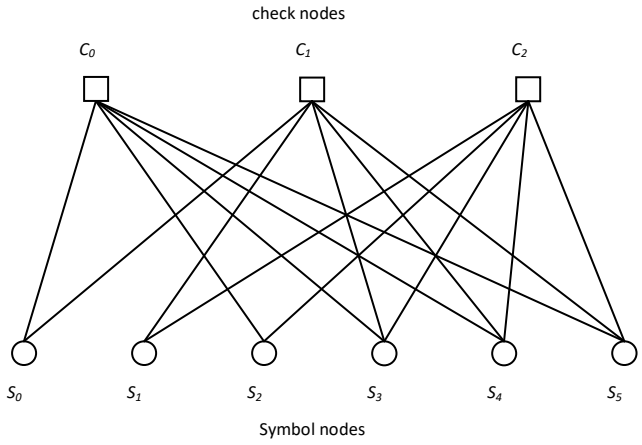

Figure 2.11: An irregular Tanner graph with symbol degree sequence $D_s = \{2,\ 2,\ 2,\ 3,\ 3,\ 3\}$

The girth at symbol vertex s_j is denoted by g_{s_j}, it is the length of the shortest cycle that passes through s_j; and the girth of a Tanner graph, denoted by g, is the length of the shortest cycle in the entire graph. The girth at a symbol vertex s_j is referred to as a *local girth*, and the girth of the entire Tanner graph is referred to as the *global girth* of the graph or simply as the *girth* of the graph. Therefore, it follows that $g = min\left\{g_{s_j}\right\}, 0 \leq j \leq n-1$.

$\left\{N_{s_j}^l\right\}$ is the set of neighbouring parity-check nodes of symbol node s_j within a subgraph expansion depth of l; it is the set consisting of all the parity-check nodes reached by a subgraph expanded from s_j within a depth of l. The complementary set of $N_{s_j}^l$ is $\dot{\mathrm{N}}_{s_j}^l$, which is defined as $V_c \,/\, N_{s_j}^l$, so that the set of check nodes $V_c = N_{s_j}^l \cup \dot{\mathrm{N}}_{s_j}^l$. The set $\left\{\dot{\mathrm{N}}_{s_j}^l\right\}$ represents the set of parity-check nodes which cannot be reached within a subgraph expansion depth l of s_j. Figure 2.12 illustrates the concept of a subgraph expansion, expansion depths, and neighbours.

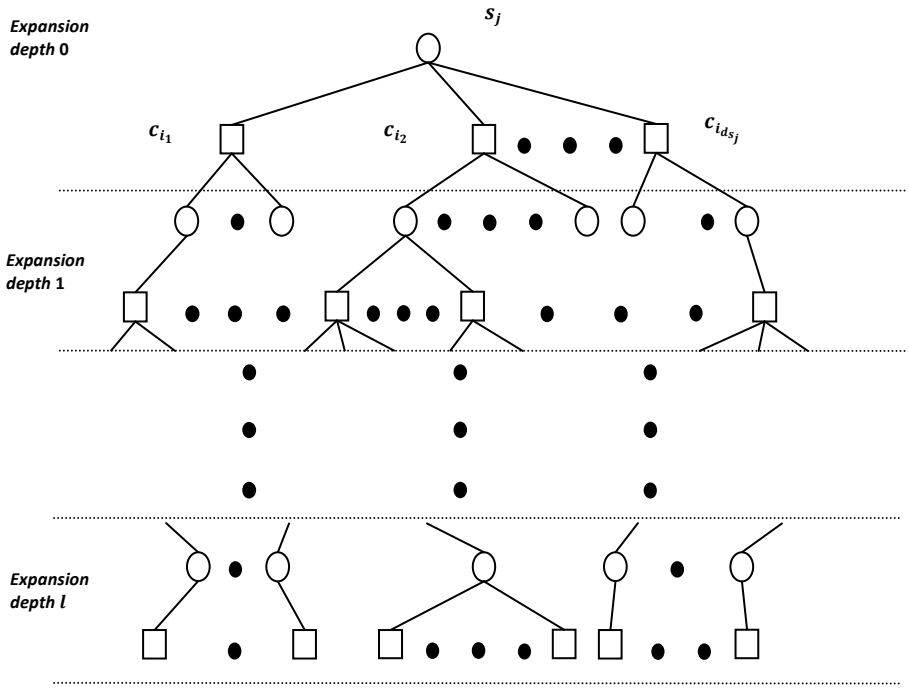

Figure 2.12: A subgraph expanded from symbol node s_j to expansion depth l

A subgraph expanded from symbol node s_j is obtained by unravelling the bipartite graph in an adjacency or neighbour relationship basis. Starting from symbol node s_j, all the edges incident on it are identified; let these edges be $(s_j, c_{i_1}), (s_j, c_{i_2}), \ldots, (s_j, c_{i_{d_{s_j}}})$. Then all the other edges incident on vertices $c_{i_1}, c_{i_2}, \ldots, c_{i_{d_{s_j}}}$, excluding $(s_j, c_{i_1}), (s_j, c_{i_2}), \ldots, (s_j, c_{i_{d_{s_j}}})$, are identified. This graph unravelling procedure continues up to a depth of l under the current graph setting, where either i) the cardinality of $N_{S_j}^l$ stops increasing but is less than m, or ii) $\dot{\mathrm{N}}_{S_j}^l \neq \emptyset$ but $\dot{\mathrm{N}}_{S_j}^{l+1} = \emptyset$. During subgraph expansions, duplicate vertices frequently occur at the same expansion depth. For any given parity-check node c_i, its neighbourhood within depth l, that is, $N_{C_i}^l$ and its complement $\dot{\mathrm{N}}_{C_i}^l$ can be similarly defined.

2.4.1 The Standard Progressive Edge-Growth (PEG) LDPC Code Construction Algorithm

The standard PEG algorithm is a good but suboptimum algorithm for constructing LDPC code Tanner graphs with large girths. It is suboptimum because constructing a graph with the largest possible girth is a very difficult combinatorial problem. In the standard PEG algorithm, the local girth of each symbol node s_j, $0 \leq j \leq n-1$, is made as large as possible before a new edge is added to it. By maximizing each local girth before the placement of a new edge in the graph, it aims to produce Tanner graphs with good global girth properties. In other words, the impact of connecting a new edge to the graph on the global girth is made to be as small as possible by maximizing the local girth of the symbol node s_j before a new edge is attached to it. The standard greedy PEG algorithm is summarized in Algorithm 2.1 [Hu et al., 2001].

Algorithm 2.1: The standard progressive edge-growth (PEG) algorithm

for $j = 0$ to $n-1$ **do**
begin
for $k = 0$ to $d_{s_j} - 1$ **do**
begin
if $k = 0$
$E_{s_j}^0 \leftarrow$ edge (c_i, s_j), where $E_{s_j}^0$ is the first edge incident to s_j,
and c_i is selected from the check nodes with the lowest check node
degree under the current graph setting $E_{s_0} \cup E_{s_1} \cup \ldots \cup E_{s_{(j-1)}}$.
else
expand a subgraph from symbol node s_j up to depth l under the
current graph setting such that the cardinality of $N_{s_j}^l$ stops increasing
but is less than m, or $\dot{N}_{s_j}^l \neq \emptyset$ but $\dot{N}_{s_j}^{l+1} = \emptyset$, then $E_{s_j}^k \leftarrow$ edge (c_i, s_j),
where $E_{s_j}^k$ is the $(k+1)$-th edge incident to s_j and c_i is a check node
selected from the lowest degree check nodes in the set $\dot{N}_{s_j}^l$.
end
end

The standard PEG algorithm for LDPC code construction was implemented in this work using the GNU/Linux based C programming language. The output of the PEG algorithm for LDPC code construction is an LDPC matrix configuration file. In our implementation, the PEG algorithm is very

versatile and can construct codes with a very wide range of block lengths, parity lengths, and code rates. Two different approaches to providing the PEG algorithm with parameters of the codes to be constructed were used: the manual input and the automated input. In the manual input version, the algorithm provides input prompts for user-entry of not only the parameters to be used during a code construction but also other parameters which are only required to put the LDPC matrix configuration file in the proper format for subsequent use by our BP/SPA decoder implementation. The following input parameters are universal to all PEG algorithm implementations in this research: code length n; intended parity length m, recall that $m = n - k$ (where k is the information/message length); the signal-to-noise ratio in decibels (SNR(dB)); the maximum number of decoding iterations; the symbol node degree distribution; the candidate parity-check node selection method; and the subgraph expansion type, i.e. greedy or nongreedy. If nongreedy subgraph expansion is selected when using the manual input version of the PEG algorithm, an additional input prompt to specify the desired maximum subgraph expansion depth, l_{max}, is provided.

The code length n and parity length m determine the dimensions of the LDPC parity-check matrix to be constructed by the PEG algorithm, and determine the lower and upper bounds of many other parameters of the code that will be constructed. Variables n and m also reflect the code rate intended by design, which is given by rate, $R = \frac{n-m}{n}$. Since $m = n - k$, the code rate R can be verified by substituting for m, so that $R = \frac{(n-(n-k))}{n} \Longrightarrow R = \frac{k}{n}$ as given in equation (1.2) . The design rate is only achieved if there are no linearly dependent parity-check equations in the rows of the LDPC code matrix which is eventually constructed. In this thesis, we work exclusively with rate-½ (i.e. $R = ½$) codes with code lengths of not more than 2048 ($n \leq 2048$), i.e. short-to-medium block lengths.

The specification of a SNR in decibels (SNR(dB)), and a maximum number of decoding iterations as inputs to our implementation of the PEG algorithm are required solely for the purpose of inserting the information in the header of the LDPC matrix configuration file so that it would be in the correct format for subsequent use by our BP/SPA decoder implementation. It is not required for code construction by the PEG algorithm.

The symbol node degree distribution is one of the most important set of variables in the design of good LDPC codes; it distinguishes regular LDPC codes from irregular LDPC codes. The best short block length LDPC codes available today are irregular codes constructed using DE optimized symbol node degree sequences. A symbol node degree sequence, D_s, is a sequence of numbers which represent the number of times each symbol node $s_j \ \forall \ 0 \leq j \leq n-1$, is evaluated in distinct parity-check equations. As an example, for a regular short-block-length code of length $n = 10$, the degree sequence can be expressed as say $D_s = \{2, 2, 2, 2, 2, 2, 2, 2, 2, 2\}$, or simply as $(d_s = 2)$ regular. Similarly, for an irregular short-block length code with length $n = 20$, the degree sequence can be expressed as say $D_s = \{2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5\}$, in which case a simpler expression can be obtained by pairing symbol node degrees with their frequency of repetition. In other words, $D_s = \{2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5\}$ can be more compactly written as $D_s = \{(2 \times 7), (3 \times 7), (4 \times 4), (5 \times 2)\}$, i.e. (symbol node degree *(times)* frequency) pairs. This shorthand method can also be used for specifying the symbol node degree sequence for regular codes, the shorthand for the regular degree sequence in our example, i.e. $D_s = \{2, 2, 2, 2, 2, 2, 2, 2, 2, 2\}$, would be written as $D_s = \{(2 \times 10)\}$. When the manual input version of the PEG algorithm provides input prompts for user-entry of the symbol node degree distribution, a shorthand method which uses (symbol node degree *(space)* frequency) pairs is used for entering the symbol node degree sequence for regular and irregular codes. Each (symbol node degree *(space)* frequency) pair is entered in a separate entry prompt line. Whenever necessary, the symbol node degree sequence entered is rearranged in the nondecreasing order, and the edge connection sequence in the standard PEG algorithm is always implemented using symbol node degree sequences in the nondecreasing order.

During code construction by the standard PEG algorithm, whenever there are multiple candidate parity-check nodes to choose from to connect a new edge for symbol node s_j, a choice is made from amongst the candidate parity-check nodes which have the lowest number of edges (symbol vertices) already connected to them under the current state of the Tanner graph, i.e. the *lowest degree* candidate check nodes. This parity-check node selection strategy results in Tanner graphs which have

parity-check node degree distributions that are as regular as possible while preserving the efficiency and versatility of the PEG algorithm. The check node degree distribution is rarely specified as an input parameter for the PEG algorithm and the strategy of establishing edges with the candidate parity-check nodes with the lowest degrees is usually sufficient to keep the check node degree distributions as uniform as possible without compromising code performance by enforcing specific check node degree distributions. As an example, compromises to code performance usually results when significantly irregular parity-check node degree distribution are enforced. This follows from strong evidence that a concentrated parity-check node degree sequence is optimum [Bazzi et al., 2004], [Shokrollahi, 1999].

At most stages of LDPC code constructions using the PEG algorithm, despite the aforementioned strategy to connect new edges of symbol nodes only to the candidate check nodes with the lowest degree under the current state of the graph, a multiple choice of candidate check nodes still remain because more than one of the candidate check nodes in $\dot{\mathrm{N}}_{s_j}^{l}$ have the same lowest degree. In this situation, [Hu et al., 2001] proposed two main approaches to solving the problem of selecting one out of these multiple lowest degree candidate parity-check nodes. In the first approach one of these parity-check nodes is selected randomly. In the second approach the parity-check nodes are selected according to their position in the order $c_0, c_1, c_2 \ldots c_{m-1}$. They proposed for instance, arranging the candidate parity-check nodes according to their subscripts in ascending order and then always selecting the first one, that is a *lowest index first* approach.

In all our PEG algorithm implementations, three options are available for selecting one from a multiple choice of lowest degree candidate parity-check nodes; the *random* approach, the *lowest index first* approach, and our proposed *flip-flop* approach. The flip-flop approach is similar to the *lowest index first* approach; the candidate parity-check nodes are also arranged according to their subscripts in ascending order $c_0, c_1, c_2 \ldots c_{m-1}$. However, an alternating sequence, that is *lowest index, highest index, lowest index, . , . ,* is used to select a check node from a choice of multiple candidate check nodes for connecting each subsequent new edge into the Tanner graph under construction; hence the name *flip-flop*. The *lowest index first* and *flip-flop* approaches are of interest

because of their deterministic nature. The manual input version of our PEG algorithm provides an user-entry input prompt for the selection of one of these three multiple choice candidate parity-check node selection methods.

The *random* approach produces better codes, evidence of this can be found in [Hu et al., 2005]. This fact was also corroborated by results of extensive simulation experiments carried out in the early stages of this research. Consequently, unless otherwise specified, the random approach to selecting one out of a multiple choice of candidate parity-check nodes after all other PEG constraints have been satisfied has been adopted throughout this research. In order to facilitate the random selection of a candidate parity-check node, a random number generator function in the C programming language called $ran2$ by [Teukolsky et al., 1992] was utilized in all PEG algorithm implementations in this research. The $ran2$ function returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint values) every time the function is called. The authors give assurance that, within the limits of its floating-point precision, $ran2$ provides perfectly random numbers. Given a fixed set of input code construction parameters to the PEG algorithm, completely different LDPC codes are constructed by simply changing the *random number seed* which is required to initialize the $ran2$ function. The random number seed is an arbitrary integer $\{\geq 1\}$ which is used to initiate the random number generation process in $ran2$. Ensembles of LDPC codes constructed using the different versions/modification to the PEG algorithm in this thesis, and fixed sets of code construction parameters, will be obtained by simply using different random number seeds to construct the constituent codes. In this research, we use the sequence of the set of natural numbers, i.e. $1, 2, 3, 4, 5, \ldots, B$, as the random number seeds for constructing a code ensemble of cardinality B.

The final input parameter which is common to all implementations of the PEG algorithm in this thesis is the subgraph expansion type. The version of the PEG algorithm implemented by [Hu et al., 2001], where the algorithm was first proposed, uses a greedy subgraph expansion. In a greedy expansion, the depth of subgraph expansions from symbol node $s_j \ \forall \ 0 \leq j \leq n-1$, is unrestricted and each graph expansion proceeds as deeply as possible. Therefore, at the later stages of graph construction, the

subgraph construction proceeds until $\dot{\mathrm{N}}_{s_j}^{l} \neq \emptyset$ but $\dot{\mathrm{N}}_{s_j}^{l+1} = \emptyset$, where l is the depth of subgraph expansion. However, Hu et al. proposed a nongreedy version where l is limited to a certain value, l_{max}, which they envisaged would achieve a strictly concentrated parity-check node sequence and would probably also reduce the diameter of the graph resulting in fewer decoding iterations being required. It is generally understood that, all other parameters being equal, the higher the girth and/or diameter of an LDPC code, the larger the number of decoding iterations required to attain a given performance (probability of decoding error) under iterative decoding algorithms.

The choice of greedy or nongreedy subgraph expansion is a built-in option in our implementation of the PEG algorithm. In the manual input version of the algorithm, an input prompt is provided for selecting one of the two subgraph expansion versions: greedy version, or nongreedy version. If the nongreedy version is selected, another input prompt is provided so that the maximum subgraph expansion depth l_{max}, to be used by the PEG algorithm can be specified. The PEG algorithm which includes the option of nongreedy subgraph expansion is summarized in Algorithm 2.2.

Algorithm 2.2: The standard progressive edge-growth (PEG) algorithm with optional nongreedy subgraph expansion

for $j = 0$ to $n - 1$ **do**
begin
for $k = 0$ to $d_{s_j} - 1$ **do**
begin
if $k = 0$
$E_{s_j}^0 \leftarrow$ edge $(c_i,\ s_j)$, where $E_{s_j}^0$ is the first edge incident to s_j,
and c_i is selected from the check nodes with the lowest check node degree under the current graph setting $E_{s_0} \cup E_{s_1} \cup \ldots \cup E_{s_{(j-1)}}$.
else
if *nongreedy* (with maximum subgraph expansion depth given as l_{max})
expand a subgraph from symbol node s_j up to depth l (where $l \leq l_{max}$) under the current graph setting such that the cardinality of $N_{s_j}^l$ stops increasing but is less than m, $\dot{\mathrm{N}}_{s_j}^l \neq \emptyset$ but $\dot{\mathrm{N}}_{s_j}^{l+1} = \emptyset$, or $\mathrm{N}_{s_j}^l = \mathrm{N}_{s_j}^{l_{max}}$ then $E_{s_j}^k \leftarrow$ edge $(c_i,\ s_j)$, where $E_{s_j}^k$ is the $(k+1)$-th edge incident to s_j, and c_i is a check node selected from the lowest degree check nodes in the set $\dot{\mathrm{N}}_{s_j}^l$.

else *greedy* (without a maximum subgraph expansion depth)
expand a subgraph from symbol node s_j up to depth l under the current graph setting such that the cardinality of $\mathrm{N}_{s_j}^l$ stops increasing but is less than m, or $\dot{\mathrm{N}}_{s_j}^l \neq \emptyset$ but $\dot{\mathrm{N}}_{s_j}^{l+1} = \emptyset$, then $E_{s_j}^k \leftarrow$ edge $(c_i,\ s_j)$, where $E_{s_j}^k$ is the $(k+1)$-th edge incident to s_j, and c_i is a check node selected from the lowest degree check nodes in the set $\dot{\mathrm{N}}_{s_j}^l$.
end
end

When all necessary code construction parameters, and other parameters required to format LDPC code configuration files for subsequent use by our BP/SPA decoder, have been provided to a PEG algorithm, an LDPC code matrix is constructed in line with the constraints of the specific PEG algorithm version. The C programming language code for the standard PEG algorithm implemented in this research is available in the Appendices (See Appendix A).

In order to construct ensembles of LDPC codes, all the different PEG algorithm versions/modifications implemented in this research work have an automated version. In automated versions, the code construction parameters and the other parameters required to format LDPC code configuration files for use by the BP/SPA decoder are provided within the lines of the C program code

and then Bash (Unix shell) commands are used to run the PEG algorithm program as many times as required using different *random number seeds* to initiate the $ran2$ function. The $ran2$ function generates the random numbers used for the *pseudorandom* process of choosing a parity-check node from multiple candidate check nodes during code constructions. This way, ensembles of codes are automatically generated and each LDPC matrix configuration file constructed is uniquely identified by being saved with the *random number seed* used for its construction as part of its filename.

2.4.2 The Edge Connections Sequence of the Standard PEG Algorithm

Figure 2.13 shows the standard PEG algorithm edge connections sequence, i.e. the sequence in which edges are connected into a Tanner graph, during the construction of a code with degree sequence $D_S = \{2, 2, 2, 2, 3, 3, 5, 5, 7, 7\}$. A '1' in the figure represents the connection of an edge in the graph; in other words, it is a $h_{i,j} = 1$ assignment in the parity-check matrix H, where $0 \leq i \leq m-1$ and $0 \leq j \leq n-1$.

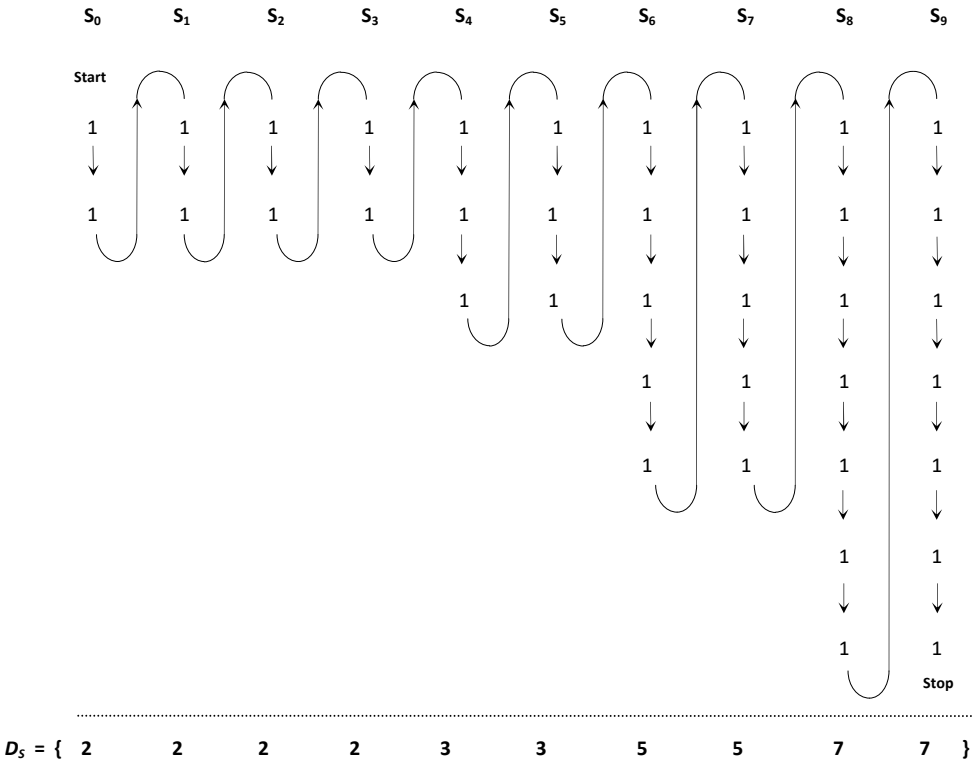

Figure 2.13: The edge connections sequence of the standard PEG algorithm

2.4.3 The LDPC Matrix Configuration File

The output of the PEG algorithm is a text file which we refer to as an *LDPC matrix configuration file*. An example of the contents of an LDPC matrix configuration file constructed using PEG algorithms is shown in Figure 2.14. The first row of the LDPC matrix configuration file contains the following parameters of the error correction code represented by the parity check matrix: The '$code\_length$', which is the total number of symbols in each block of the encoded message; the '$parity\_length$', which is the number of parity-check symbols (number of linearly independent parity-check equations) in each block of the encoded message; and the '$rows$', which is the total number of parity-check equations in the configuration file. In addition, the '*SNR*', i.e. the signal-to-noise ratio in decibels (SNR(dB)) at the receiver, and the '$iterations$', i.e. the maximum number of decoding iterations to be executed by the BP/SPA decoder, are also specified in the first row of the LDPC matrix configuration file.

```
code_length=512,parity_length=256,rows=256,SNR=4.00dB,iterations=100
35 219 243 357 445 461 490 506 -1          ← parity-check equation 0
3 226 305 386 409 458 480 507 -1           ← parity-check equation 1
87 172 252 348 412 469 480 508 -1
35 200 235 360 402 470 478 509 -1
81 187 312 319 432 470 489 503 -1
116 175 279 362 446 464 476 507 -1
127 143 255 373 426 461 492 507 -1
65 200 302 359 435 466 480 505 -1
93 206 307 383 413 471 477 501 -1
0 170 289 382 401 462 477 499 -1
38 153 293 376 446 460 476 500 -1
5 203 240 337 396 467 493 509 -1
100 167 298 355 405 473 490 505 -1
34 163 248 391 434 463 485 502 -1
48 169 301 334 417 467 485 497 -1
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
23 217 274 380 425 459 479 494 -1
92 219 247 368 425 469 477 498 -1
95 202 271 344 397 463 477 511 -1
37 181 252 377 435 471 486 496 -1
104 170 309 377 423 462 485 504 -1         ← parity-check equation 255
```

Note: ***symbol nodes are numbered from 0 to 511***

Figure 2.14: An example of the contents of an LDPC matrix configuration file created for an LDPC code constructed using the PEG algorithm

The remaining set of rows of text in the LDPC matrix configuration file make up a shorthand representation of the LDPC code matrix constructed. These set of rows have a cardinality of '$rows$', where $rows = m$, as specified in the first line of the configuration file, and each of these rows represents a distinct parity-check equation c_i, for $0 \leq i \leq m-1$. Each column of a parity check matrix represents a distinct symbol node s_j, for $0 \leq j \leq n-1$, where j is the index number of the column or symbol node in the parity-check matrix. In the shorthand LDPC code matrix representation, the row corresponding to check equation c_0 lists the indices of only the set of symbol nodes evaluated by c_0, and so on. In other words, each row c_i , $0 \leq i \leq m-1$, contains only the set of j's for which $H_{i,j} = 1,\ 0 \leq j \leq n-1$. The configuration file is formatted to be used directly by the BP/SPA decoder for code performance simulations implemented in this research.

2.5 The Belief Propagation or Sum-Product Algorithm (BP/SPA) Message-Passing Iterative Decoder

The standard belief propagation or sum-product algorithm (BP/SPA) decoder for simulating the performance of LDPC codes over the AWGN channel was implemented in the C programming language on the GNU/Linux UBUNTU® operating system platform. The C programming language code for the standard BP/SPA decoder implemented in this research is available in the Appendices (See Appendix B). The message-passing BP/SP iterative decoding algorithm was implemented as part of a larger program which reads an LDPC matrix configuration file, validates the integrity of the parity-check matrix H read from the file, and then carries out all the necessary pre-processing of the H matrix before simulating its error correction performance. For simplicity, the entire program is referred to as the BP/SPA decoder and binary codes are assumed in the descriptions that follow.

The program generates random binary messages using the $ran2$ random number generator function obtained from [Teukolsky et al., 1992]; messages are encoded using a codeword generator matrix G that is derived from the parity-check matrix H obtained from the LDPC matrix configuration file; the transmission of codeword bits using binary phase-shift keying (BPSK) modulation are simulated; and the reception of each encoded vector over an AWGN channel at a specified SNR(dB) is simulated.

Based on the received vectors, the *a posteriori* probabilities of the received bits are calculated and then used in the BP/SPA decoder for error correction. Using the Monte Carlo method, the probability of decoding error of LDPC codes at different SNR(dB)s at a specified maximum number of decoding iterations are determined. The BP/SPA decoder was used to carry out performance simulations on almost all the LDPC code matrices which were investigated in this research except for those in Chapter 7 which were decoded using improved BP/SPA decoders. The following is a detailed description of the operation of the BP/SPA decoder software used in this research to simulate the performance of short-to-medium length LDPC codes over the AWGN channel.

The BP/SPA decoder needs an LDPC matrix configuration file in order to carry out its function. The BP/SPA decoder starts its operation by reading the contents of an LDPC matrix configuration file into memory. The contents of a sample LDPC matrix configuration file used by the BP/SPA decoder are shown in Figure 2.14. The first line of the configuration file contains important information about the LDPC matrix which the BP/SPA decoder requires in order to:

i) determine the dimensions of the parity check matrix H in the file in order to read the file contents correctly; these are the variables called $code\_length$, $parity\_length$ and $rows$, and

ii) determine the SNR(dB) at which to simulate signal reception over the AWGN channel, and the maximum number of decoding iterations to execute; these are the variables called SNR and $iterations$ respectively.

Ideally, the parity length and the number of rows m of parity-check equations in a parity-check matrix would be identical, i.e. $parity\_length = rows, (m)$. However, sometimes there are one or more linearly dependent parity-check equations so that there are fewer linearly independent parity-check equations than there are rows m of parity-check equations in the LDPC matrix configuration file $parity\_length \leq rows, (m)$. This situation is searched for and detected by the BP/SPA decoder as the program attempts to obtain the codeword generator matrix G for a parity-check matrix in memory. The presence of linearly dependent parity-check equations in the parity-check matrix of a code simply means that the code described by the parity-check matrix has a slightly higher information rate than

indicated by the number of rows in the parity-check matrix. A smaller number of linearly independent parity-check equations implies a smaller number of parity-check bits and a higher number of information bits per codeword; this translates to a higher code rate.

All the subsequent rows after the first row in an LDPC matrix configuration file describe parity-check equations; these rows are assigned virtual numbers serially down the file, from parity-check equation 0 to parity-check equation $(m-1)$ in sequence. The indices of all the symbol nodes which are involved in each parity-check equation are listed in the corresponding row of parity-check equations which also correspond to rows of the parity-check matrix. Here, symbol nodes s_j for all $0 \leq j \leq n-1$ are represented by their indices and the j's of all symbol nodes from s_0 to s_{n-1}, which are involved in parity-check equation c_i for all $0 \leq i \leq m-1$ are listed in the corresponding rows of the LDPC matrix configuration file. The indices of the symbol nodes evaluated in each parity-check equation are listed with one or two '*space*' characters placed between them. The index numbers in each parity-check equation row indicate the positions of the 1's in the corresponding row of the binary parity check matrix. The '-1' placed at the end of each parity check equation row is used to terminate the parity-check equation after all the symbol nodes which it evaluates have been completely listed.

Before the parity check matrix in the configuration file is read into memory, a zero matrix with number of rows equal to the variable '*rows'* and number of columns equal to the variable '*code_length'* as specified in the first line of the configuration file is created. Thereafter, the positions of the 1's in each line of the LDPC code matrix to be read into memory are determined from the contents of corresponding lines of parity-check equations in the configuration file, and the bit values at corresponding positions in the originally all zero matrix in memory are changed from 0 to 1. Figure 2.15 (a) shows the contents of an LDPC matrix configuration file, and Figure 2.15 (b) is the [20, 10, 4] LDPC code matrix derived from it.

code_length=20,parity_length=10,rows=10,SNR=4.00,iterations=100

0 5 11 18 -1
0 9 12 15 -1
1 8 10 16 19 -1
4 7 15 16 -1
3 7 12 17 19 -1
3 6 14 16 18 -1
4 8 11 17 -1
2 6 10 17 -1
1 9 13 14 -1
2 5 13 15 19 -1

(a)

$H =$

s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}	s_{18}	s_{19}	
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	c_0
1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	c_1
0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	c_2
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	c_3
0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	c_4
0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	c_5
0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	c_6
0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	c_7
0	1	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	c_8
0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	1	c_9

(b)

Figure 2.15: (a) The contents of an LDPC matrix configuration file, and (b) the corresponding [20, 10, 4] LDPC code H matrix

When a '-1' is encountered while the indices of the symbol nodes involved in a parity-check equation are being read from the rows of the LDPC matrix configuration file, it is interpreted to mean that all symbol nodes involved in that parity-check equation have been completely listed. The program reads the contents of all the lines in the configuration file in this manner until it reaches the '-1' in the row where the number of parity-check equations read is equal to the number of $rows$ as specified in the first line of the configuration file. At that juncture, it is assumed that the complete LDPC code matrix has been read into memory, and any other text written thereafter in the LDPC matrix configuration file is ignored.

After an LDPC code matrix H has been successfully created in memory from a configuration file, the H matrix in memory is checked to ensure that it does not have redundant rows or columns. A redundant row in a H matrix is a row with less than two symbol nodes. A valid parity-check equation must have at least two elements, i.e. at least one parity-check bit and one information bit. Similarly, a redundant column in a H matrix is an all-zero column or a column without a single '1' bit. A symbol must be either an information bit or a parity-check bit, it cannot be neither. If redundant row(s) or column(s) are found in the H matrix read from an LDPC matrix configuration file, an error message

which specifies the redundant row or column number(s) is displayed and the execution of the BP/SPA decoder program is terminated.

2.5.1 Message Encoding

In the absence of errors in the configuration file, the BP/SPA decoder program proceeds to obtain a codeword generator matrix G for the LDPC code parity-check matrix H in memory. The well established *Gaussian elimination* procedure is performed on a copy of the H matrix in memory to obtain its *row echelon* form H_{re}; it is often necessary to perform column swapping operations to derive the row echelon form of a H matrix. Figure 2.16 shows matrix H_{re}, the row echelon form of the LDPC code matrix in Figure 2.15(b), columns 0 and 19 were swapped during the Gaussian elimination process.

$$H_{re} = \begin{array}{c} \begin{matrix} s_{19} & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_0 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \mathbf{0} & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{0} & \mathbf{0} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 \end{bmatrix} \begin{matrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{matrix} \end{array}$$

Figure 2.16: The [20, 10, 4] row echelon LDPC code matrix, H_{re}

Subsequently, using additional Gaussian elimination, the row echelon H matrix is converted to the *reduced row echelon* form in which the parity-check bits are expressed as explicit sums (modulo-2) of information bits. It is at this juncture that it can be easily determined if there are linearly dependent parity-check equations in the original parity-check matrix read from the configuration file. The presence of linearly dependent parity-check equations in a H matrix is clearly indicated by the presence of one or more all-zero rows in the reduced row echelon form of the H matrix. The number of linearly dependent rows is determined by the number of all-zero rows. If one or more linearly

dependent parity-check equations are detected and the $parity\_length$ which is specified in the first line of the LDPC matrix configuration file is different from the actual number of linearly independent check equations in the parity-check matrix of the code, an error message is displayed to indicate the presence and exact number of linearly dependent rows and the program terminates. In such cases, the LDPC matrix configuration file must be edited so that the $parity\_length$ specified in the first line reflects the actual parity length of the parity-check matrix; the actual parity length is obtained as the difference between the total number of parity-check equations (i.e. $rows$) and the number of linearly dependent parity-check equations in the H matrix. The actual parity length of a H matrix is known as its '*rank*', and it is the required value of m whenever equation $m = n - k$ is to be used. As explained previously, whenever the actual parity length is less than the number of rows in the H matrix as a result of the presence of linearly dependent parity-check equations, it simply implies that the LDPC code has a code rate which is slightly higher than intended during the code design. Figure 2.17 shows matrix H_{rre}, the reduced row echelon form of the LDPC code matrix in Figure 2.16. There are no all-zero rows in Figure 2.17. Therefore, there are no linearly dependent parity-check equations in the H matrix. The H matrix is of full rank.

$$H_{rre} = \begin{array}{c} \begin{matrix} s_{19} & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_0 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \end{array} \begin{matrix} \\ c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{matrix}$$

Figure 2.17: The [20, 10, 4] reduced row echelon LDPC code matrix, H_{rre}

After ensuring that the value of $parity\_length$ in the LDPC matrix configuration file is correct, and the H matrix has been expressed in the reduced row echelon form which is devoid of all-zero rows,

some elementary matrix algebra techniques are applied on the H matrix to arrive at a codeword generator matrix G for the code. The reduced row echelon H matrix is compared to the following matrix format:

$$H = [A|I_{n-k}], \tag{2.1}$$

where A is an $(n-k) \times k$ matrix; and I_{n-k} is an $(n-k) \times (n-k)$ identity matrix, i.e. an identity matrix of order $(n-k)$. Figure 2.18 shows the submatrix of H_{rre}, the reduced row echelon H matrix, which corresponds to A in equation (2.1).

The G matrix for a code whose H matrix is in the reduced row echelon form can be easily obtained as:

$$G = [I_k|A^T], \tag{2.2}$$

where G is a $k \times n$ matrix; A^T is the transpose of A and is consequently a $k \times (n-k)$ matrix; and I_k is an identity matrix of order k.

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

Figure 2.18: Submatrix A obtained from H_{rre} - the [20, 10, 4] reduced row echelon matrix

Submatrix A of the reduced row echelon parity-check matrix H_{rre} is extracted and a matrix transpose operation is carried out on it to obtain matrix A^T. Figure 2.19 shows matrix A^T which was obtained by transposing the A submatrix of H_{rre} shown in Figure 2.18.

$$A^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Figure 2.19: Matrix A^T - the transpose of submatrix A

Subsequently, an identity matrix I_k is generated and the two matrices are merged to form codeword generator matrix $G = [I_k|A^T]$. It is important to note that if column swapping operations were carried out during the Gaussian elimination to obtain a H matrix in the reduced row echelon form, the G matrix subsequently obtained generates the codewords for the reduced row echelon H matrix and not the codewords for the original H matrix read from the configuration file. Figure 2.20 shows G_{rre}, the G matrix obtained for H_{rre} - the reduced row echelon H matrix.

$$G_{rre} = \begin{array}{c} \begin{matrix} s_{19} & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_0 \end{matrix} \\ \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array} \begin{matrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{matrix}$$

Figure 2.20: The codeword generator matrix G_{rre} corresponding to matrix H_{rre}

If column swapping operations were carried out to obtain H_{rre}, the resulting codeword generator matrix G_{rre} can be converted to G in order to generate codewords for the original parity-check matrix in memory, i.e. H, by unswapping all swapped columns in the reverse sequence in which the columns were originally swapped. In our example, only columns 0 and 19 were swapped during Gaussian elimination to obtain H_{re}, consequently to obtain G from G_{rre} these two columns must be unswapped. Figure 2.21 shows the final codeword generator matrix G for the original parity matrix H which was read into memory from the LDPC matrix configuration file. The codeword generator matrix processing can be done offline, and subsequently just the G and H matrices provided to the encoder and decoder respectively.

$$G = \begin{array}{c} \begin{matrix} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 & s_9 & s_{10} & s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} & s_{17} & s_{18} & s_{19} \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{matrix}$$

Figure 2.21: The codeword generator matrix G for LDPC code matrix H

The row space of G is orthogonal to H, thus the following relationship holds true for corresponding pairs of G and H:

$$GH^T = 0 \tag{2.3}$$

A message vector is conventionally denoted by matrix $\boldsymbol{u} = [u_0 \ u_1 \ u_2 \ \ldots \ u_{k-1}]$, where the vector $\boldsymbol{u}$ consists of the k message bits. A code with k message bits has a maximum of 2^k codewords. These codewords are a subset of the 2^n possible binary vectors of length n. The codeword $\boldsymbol{c}$ corresponding to any binary message $\boldsymbol{u}$ can be obtained using matrix multiplication,

$$\boldsymbol{c} = \boldsymbol{u}G \tag{2.4}$$

In this encoding approach, unlike the parent H matrices, the G matrices derived are generally not of low density, this may be observed by comparing the matrices in Figure 2.15(b) and Figure 2.21. Consequently, the matrix multiplication at the encoder will have a complexity in the region of n^2 operations; the encoder can become prohibitively complex for LDPC codes with very large n. The use of structured parity-check matrices can reduce this implementation complexity significantly [Johnson, 2006].

A message vector for the 10×20 G-matrix in Figure 2.21, which has information bit length $k = 10$, may be denoted as,

$$\boldsymbol{u} = [u_0 \ u_1 \ u_2 \ u_3 \ u_4 \ u_5 \ u_6 \ u_7 \ u_8 \ u_9].$$

A k bit long random message vector can be easily generated using the $ran2$ function as follows. Recall that the $ran2$ function returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint values) every time the function is called. The value of each bit of a message, from bit u_0 to bit u_{k-1}, can be randomly determined using a separate call to the $ran2$ function. Denote the value of the deviate returned by the v-th call to the $ran2$ function as d_v, where v is an arbitrary integer which increases by one (1) for every call to $ran2$ ($v = 0$ at the start of program execution). To randomly assign a binary value to a message bit, say bit u_0, a call is made to $ran2$ which in turn returns a value d_v, if $0.0 < d_v < 0.5$ then assign bit $u_0 = 0$, else if $0.5 \leq d_v < 1.0$ assign bit $u_0 = 1$. This process is repeated k times to obtain random message $\boldsymbol{u} = [u_0 \ u_1 \ u_2 \ \ldots \ u_{k-1}]$. Codewords are obtained using matrix multiplications $\boldsymbol{c} = \boldsymbol{u}G$, and are denoted by $\boldsymbol{c} = [c_0 \ c_1 \ c_2 \ \ldots \ c_{n-1}]$,

Codeword vectors for the 10×20 G-matrix in Figure 2.21, which has codeword length $n = 20$, may be denoted as,

$$\boldsymbol{c} = [c_0 \ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6 \ c_7 \ c_8 \ c_9 \ c_{10} \ c_{11} \ c_{12} \ c_{13} \ c_{14} \ c_{15} \ c_{16} \ c_{17} \ c_{18} \ c_{19}].$$

2.5.2 The AWGN Channel

The BP/SPA decoder simulates the reception of signals transmitted using binary phase-shift keying (BPSK) modulation over an AWGN channel. The analogue amplitudes of transmitted symbol bits are assigned as follows: bit $0 = -1.0V$, and bit $1 = +1.0V$. As an example, we assume a twelve (12) bit long codeword $\boldsymbol{c}$, where

$$\boldsymbol{c} = [0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1],$$

the transmitted BPSK modulated vector $\boldsymbol{t}$ will be represented by,

$$\boldsymbol{t} = [-1.0 \ +1.0 \ +1.0 \ -1.0 \ -1.0 \ +1.0 \ +1.0 \ -1.0 \ -1.0 \ -1.0 \ -1.0 \ +1.0]$$

Each random variable received over the AWGN channel can be defined as:

$$r_j = t_j + N_j, \quad \text{where } 0 \leq j \leq n-1 \tag{2.5}$$

where t_j is a constant signal of $\pm 1.0V$ which represents a transmitted bit, and N_j is a random variable which represents the channel noise signal added at the instant of transmission. AWGN vectors N, where $N = \{N_j\}$ for all j, have an expected value (or mean) of zero. Using this model, the linear signal-to-noise ratio SNR can be expressed as:

$$SNR = \frac{t^2}{\sigma^2} \tag{2.6}$$

where t^2 is the mean squared value of the signal power, and σ^2 is the variance of the noise N on the channel. In the AWGN channel, noise power is equivalent to noise variance σ^2 because the noise on the channel has zero mean. The linear SNR may be expressed in decibels (dB) as:

$$SNR(dB) = 10 \log_{10} SNR \tag{2.7}$$

Conversely, the linear SNR can be obtained from its decibel value using:

$$SNR = 10^{\frac{SNR(dB)}{10}} \tag{2.8}$$

The signal-to-noise ratio in decibels (SNR(dB)) at which decoding performance simulation is to be carried out by the BP/SPA iterative decoder on an LDPC code matrix is specified in the first line of the LDPC matrix configuration file.

With the amplitudes of the transmitted signals given as $t = \pm 1.0V$, the signal power is $t^2 = 1$, therefore:

$$SNR = \frac{1}{\sigma^2} \tag{2.9}$$

and by making σ the subject of the formular,

$$\sigma = \frac{1}{\sqrt{SNR}} \tag{2.10}$$

The channel noise standard deviation, σ, is used to regulate the effect (amplitude) of AWGN in the channel on the received signal in proportion to the SNR at which decoding of received codeword signals are to be simulated. All transmitted messages/codewords have the same messages energies.

In order to correctly simulate the effects of AWGN at a given SNR on the sequence of bits transmitted over the AWGN channel using BPSK modulation, the BP/SPA decoder utilizes a function in C programming language called $gasdev$ in addition to the $ran2$ function which is used in the PEG algorithm. The $gasdev$ and $ran2$ functions were both obtained from [Teukolsky et al., 1992]; the use of the $ran2$ function has been explained in Section 2.4.1. The $gasdev$ function returns a normally distributed (Gaussian) random deviate with zero mean and unit variance. The $gasdev$ function implemented in the BP/SPA decoder uses the $ran2$ function as a source of uniform deviates for its operation. Calls to the $gasdev$ function are used to simulate the effects of AWGN noise on BPSK modulated signals because the random deviates it returns have a Gaussian distribution with zero mean and unit variance. It is important to note that the original $gasdev$ function in [Teukolsky et al., 1992] uses the '$ran1$' function as the source of uniform random deviates, however the $gasdev$ function implemented in this research uses the $ran2$ function instead because it has a significantly longer

period than $ran1$, and far greater than 100 million random numbers will be generated in the course of arriving at statistically accurate decoding error probabilities.

The deviates returned using successive calls to the $gasdev$ function are used to generate discrete AWGN vectors in proportion to the desired SNR of the received codeword signals. A sequence of n AWGN vectors $\{N_0, N_1, \ldots N_{n-1}\}$ can be represented as N_j where $0 \leq j \leq n-1$.

$$\boldsymbol{N} = [N_0 \;\; N_1 \;\; N_2 \ldots . \; N_{n-1}]$$

Each noise vector N_j is obtained by making a call to the $gasdev$ function and multiplying the value of the deviate returned by σ - the standard deviation of the noise at the given SNR, so that,

$$N_j = gsdv_v \times \sigma \tag{2.11}$$

where $gsdv_v$ is the value of the random deviate returned from the v-th call to the $gasdev$ function. Recall that calls to the $gasdev$ function returns normally (Gaussian) distributed random deviates with zero mean and unit variance.

For example, by using twelve (12) successive calls to the $gasdev$ function and then multiplying each deviate returned by σ for an arbitrary SNR(dB), the following noise vectors (rounded off to one decimal place for simplicity) were obtained:

$$\boldsymbol{N} = [-0.4 \; -1.1 \; +0.3 \; -0.1 \; +0.4 \; +0.2 \; +0.1 \; +0.8 \; -0.4 \; +0.2 \; -0.2 \; +0.3]$$

For transmissions over AWGN channels, each transmitted bit vector t_j and its corresponding received bit vector r_j in a codeword are related as described by equation (2.5). That is, $r_j = t_j + N_j$, where $0 \leq j \leq n-1$. Therefore, an AWGN channel noise vector N_j is added to each transmitted bit vector t_j before it is received as r_j. Recall that,

$$\boldsymbol{t} = [-1.0 \; +1.0 \; +1.0 \; -1.0 \; -1.0 \; +1.0 \; +1.0 \; -1.0 \; -1.0 \; -1.0 \; -1.0 \; +1.0].$$

When BPSK modulated codeword vector $\boldsymbol{t}$ is transmitted, AWGN noise vector string $\boldsymbol{N}$ is added to it so that $\boldsymbol{r} = \boldsymbol{t} + \boldsymbol{N}$. This addition is carried out in the component-by-component basis to obtain the received vector $\boldsymbol{r}$ as:

$$\boldsymbol{r} = [-1.4 \; -0.1 \; +1.3 \; -1.1 \; -0.6 \; +1.2 \; +1.1 \; -0.2 \; -1.4 \; -0.8 \; -1.2 \; +1.3]$$

It can be observed that the magnitude of the AWGN noise in the second code bit position of $\boldsymbol{N}$ (i.e. N_1) was sufficient to change the phase (polarity) of the corresponding received vector compared to that of the transmitted vector. The received bit in the second bit position $r_1 = t_1 + N_1$, where $t_1 = +1.0$ and $N_1 = -1.1$. Therefore, the received bit vector $r_1 = -0.1$ while the transmitted bit vector $t_1 = +1.0$. This change in polarity due to channel noise results in a symbol error if a hard decision is made on this received symbol vector.

On the AWGN channel, the probability of a received bit r_i being a 1 or a 0 is given by,

$$p_i^b = P(c_i = b|r_i) = ae^{\frac{-(r_i - v^b)^2}{2\sigma^2}} \tag{2.12}$$

where i is the position of the received bit in a received codeword vector $r = [r_0, r_1, .. , r_i, .. , r_{n-1}]$ corresponding to transmitted codeword vector $c = [c_0, c_1, .. , c_i, .. , c_{n-1}]$, and $0 \leq i \leq n-1$; b is the bit value, $b = 0, 1$; a is an arbitrary constant; $v^b = \pm 1.0$, such that $v^0 = -1.0$ and $v^1 = +1.0$; and $SNR = 1/\sigma^2$.

The following choice is made when evaluating the probabilities of received code bit vectors. One may choose to determine either,

i) p^0, the probability that each received bit vector is a 0, or

ii) p^1, the probability that each received bit vector is a 1.

Any one of the two alternatives may be freely chosen as they both result in identical information.

The probability that each received bit vector is a 1, i.e. p^1, was used in the BP/SPA decoder implemented in this research. p^1 is derived using,

$$p_i^{1o} = \frac{p_i^1}{p_i^0 + p_i^1} \tag{2.13}$$

$b = 0$ and $b = 1$ can be substituted in equation (2.12), to obtain

$$p_i^0 = ae^{\frac{-(r_i+1.0)^2}{2\sigma^2}} \tag{2.14}$$

and

$$p_i^1 = ae^{\frac{-(r_i-1.0)^2}{2\sigma^2}} \tag{2.15}$$

respectively. Therefore, substituting for p_i^0 and p_i^1 in equation (2.13),

$$p_i^{1o} = \frac{ae^{\frac{-(r_i-1.0)^2}{2\sigma^2}}}{ae^{\frac{-(r_i+1.0)^2}{2\sigma^2}} + ae^{\frac{-(r_i-1.0)^2}{2\sigma^2}}} \tag{2.16}$$

Equation (2.16) can be further simplified to obtain,

$$p_i^{1o} = \frac{1}{1+e^{\frac{-2r_i}{\sigma^2}}} \tag{2.17}$$

Based on the received vector $\boldsymbol{r}$ in our example, i.e.,

$$\boldsymbol{r} = [-1.4\ -0.1\ +1.3\ -1.1\ -0.6\ +1.2\ +1.1\ -0.2\ -1.4\ -0.8\ -1.2\ +1.3],$$

the probabilities that each of these received vectors are equal to bit 1 can be calculated using equation (2.17). Assuming that $SNR = 1/\sigma^2 = 1.239$ (equal to $SNR(dB) = 0.931$), the probabilities that the received vectors of $\boldsymbol{r}$ are equal to bit 1 are as follows,

$$P = [0.03 \quad 0.44 \quad 0.96 \quad 0.06 \quad 0.18 \quad 0.95 \quad 0.94 \quad 0.38 \quad 0.03 \quad 0.12 \quad 0.05 \quad 0.96]$$

These codeword bit probabilities are the required inputs from the channel to the iterative soft input soft output (SISO) BP/SPA decoding algorithm.

2.5.3 Iterative Decoding

In this section, the maximum a posteriori (MAP) decoding of a single parity check (SPC) code using codeword enumeration and Gallager (dual) decoding are discussed in order to establish some vital precepts to BP/SPA iterative decoding. For simplicity, we focus on the $k = 2, n = 3$ even parity SPC code; the discussion is similar for SPC codes of any length of k. The set of codewords for the chosen code are $\{[c_0 c_1 c_2]\} = \{000, 011, 101, 110\}$.

Codeword enumeration

Assuming the r_i values have been received, their probabilities are computed using equation (2.12), i.e.,

$$p_i^b = P(c_i = b|r_i) = ae^{\frac{-(r_i - v^b)^2}{2\sigma^2}}$$

where i is the bit position and b is the bit value, $i = 0, 1, 2$ and $b = 0, 1$. The probability of each codeword is calculated as shown in Table 2.1.

Table 2.1: MAP decoding of an $(n,\ k) = (3,\ 2)$ SPC code by codeword enumeration

$c_0\ c_1\ c_2$	$P(\boldsymbol{c}\|\boldsymbol{r})$
0 0 0	$p_0^0 p_1^0 p_2^0$
0 1 1	$p_0^0 p_1^1 p_2^1$
1 0 1	$p_0^1 p_1^0 p_2^1$
1 1 0	$p_0^1 p_1^1 p_2^0$

To determine the probability of output bit c_i being b, the probabilities of all the codewords for which $c_i = b$ are summed. Therefore, the output probability of bit c_1 being 1 is,

$$p_1^{1\,o} = p_0^0 p_1^1 p_2^1 + p_0^1 p_1^1 p_2^0 = p_1^1 (p_0^0 p_2^1 + p_0^1 p_2^0)$$

$$\Longrightarrow\ p_1^{1\,o} = p_1^1 (p_0^0 p_2^1 + p_0^1 p_2^0) \tag{2.18}$$

The quantity

$$p_1^{1\,i} = p_1^1 \tag{2.19}$$

is known as *intrinsic* information and was known before decoding the SPC code, and the quantity

$$p_1^{1\,e} = p_0^0 p_2^1 + p_0^1 p_2^0, \tag{2.20}$$

which was produced by decoding the SPC code, is known as *extrinsic* information.

Similarly, the output probability of c_1 being 0 is,

$$p_1^{0\,o} = p_0^0 p_1^0 p_2^0 + p_0^1 p_1^0 p_2^1 = p_1^0 (p_0^0 p_2^0 + p_0^1 p_2^1)$$

$$\Longrightarrow \; p_1^{0\,o} = p_1^0 (p_0^0 p_2^0 + p_0^1 p_2^1) \tag{2.21}$$

where the intrinsic information is,

$$p_1^{0\,i} = p_1^0, \tag{2.22}$$

and the extrinsic information is,

$$p_1^{0\,e} = p_0^0 p_2^0 + p_0^1 p_2^1. \tag{2.23}$$

Generally, it can be observed that $p_i^{0\,e}$ (the extrinsic information for output bit i to be 0) is the sum of all products of the other bit probabilities for which the sum of bits is *even*, and $p_i^{1\,e}$ (the extrinsic information for output bit i to be 1) is the sum of all products of the other bit probabilities for which the sum of bits is *odd*.

Gallager (dual) decoding

To determine the output probabilities for bit $i = 1$, as an example, Gallager observed that if we calculate the following products,

$$(p_0^0 + p_0^1)(p_2^0 + p_2^1) = 1 = \; p_0^0 p_2^0 + p_0^0 p_2^1 + p_0^1 p_2^0 + p_0^1 p_2^1 \tag{2.24}$$

$$(p_0^0 - p_0^1)(p_2^0 - p_2^1) = (1 - 2p_0^1)(1 - 2p_2^1) = \; p_0^0 p_2^0 - p_0^0 p_2^1 - p_0^1 p_2^0 + p_0^1 p_2^1 \tag{2.25}$$

and we add equation (2.24) and equation (2.25) we obtain twice the sum of all 'even' terms, which is equivalent to $2p_1^{0\,e}$. Similarly, if we subtract equation (2.25) from equation (2.24) we obtain twice the sum of all 'odd' terms, which is equivalent to $2p_1^{1\,e}$. Therefore, we obtain the following equations for $p_1^{0\,e}$ and $p_1^{1\,e}$ respectively,

$$p_1^{0\,e} = \frac{1+(1-2p_0^1)(1-2p_2^1)}{2} = p_0^0 p_2^0 + p_0^1 p_2^1 \tag{2.26}$$

$$p_1^{1\,e} = \frac{1-(1-2p_0^1)(1-2p_2^1)}{2} = p_0^0 p_2^1 + p_0^1 p_2^0 \tag{2.27}$$

Similarly, for bit $i = 2$,

$$p_2^{0\,e} = \frac{1+(1-2p_0^1)(1-2p_1^1)}{2} \tag{2.28}$$

$$p_2^{1\,e} = \frac{1-(1-2p_0^1)(1-2p_1^1)}{2} \tag{2.29}$$

In general for any bit i,

$$p_i^{0\,e} = \frac{1+\prod_{j\neq i}\left(1-2p_j^1\right)}{2} \tag{2.30}$$

$$p_i^{1\,e} = \frac{1-\prod_{j\neq i}\left(1-2p_j^1\right)}{2} \tag{2.31}$$

If the probability of a bit being a 0 is mostly required, the notation of equation (2.30) can be simplified by denoting $p_j = p_j^1 = 1 - p_j^0$, to obtain,

$$p_i^{0\,e} = \frac{1+\prod_{j\neq i}(1-2p_j)}{2} \tag{2.32}$$

and $p_j^0 = 1 - p_j$.

However, the probability of a bit being a 1 is mostly used in the BP/SPA decoder implemented in this research, the notation of equation (2.31) is simplified by denoting $p_j = p_j^1 = 1 - p_j^0$, to obtain,

$$p_i^{1\,e} = \frac{1-\prod_{j\neq i}(1-2p_j)}{2} \tag{2.33}$$

and $p_j^0 = 1 - p_j$.

The general algorithm for iterative decoding

The iterative decoding of linear codes introduced by Gallager is based on the H matrix of the code [Gallager, 1962]. Any H matrix is composed of at least $m = n - k$ parity check equations, and each of these parity-check equations can be separately decoded using one of the SPC algorithms. The general algorithm is as follows.

1. Receive the channel values $[r_i]$ where $0 \leq i \leq n-1$ for a transmitted codeword, and calculate the channel probabilities for each code bit to be a 1/0, p_i^d where $i = 0, \ldots, n-1$ and $d = 0, 1$.
2. Decode the first parity-check equation using the channel values, this produces extrinsic probabilities (additional information) for the bits involved in the first parity-check equation.
3. Starting from the second parity-equation, cycle through all the equations in the following way: Using the channel values **and** any other extrinsic information produced by previous equations **except** the current one, decode the current equation. Update the bit probabilities for the current equation.

   Note: Whenever there are several pieces of information about a bit, for example p_a, p_b and p_c - the probabilities that the bit is a 1 coming from the channel and extrinsic output from other equations, they are combined by multiplying the probabilities that the bit is a 1 together. This hides an assumption of independence of the probabilities, which is valid for the first iterations. To combine p_a and p_b we use the function,

$$f(p_a, p_b) = \frac{p_a p_b}{p_a p_b + (1-p_a)(1-p_b)} \tag{2.34}$$

   where the numerator is the probability multiplication and the denominator is a normalization factor. In order to combine more than two probabilities, say p_a, p_b and p_c, we can apply equation (2.34) repeatedly, $f(f(p_a, p_b), p_c)$.

4. Output the probabilities of each bit $p_i^{1\,o}/p_i^{0\,o}$. These contain all the information available from the channel and equations at this time, combined using equation (2.34).

An exemplification of iterative decoding

Consider a $(n,\ k) = (6,\ 3)$ linear code with the following H matrix,

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

Figure 2.22: The H matrix of an $(n,\ k) = (6,\ 3)$ linear code

Iterative decoding will be illustrated using this code and, since both k and $n - k$ are small, we are also able to perform MAP decoding. Consequently, we compare the output of iterative decoding with the output of optimal MAP decoding carried out on a set of received channel values. The H matrix of the code is systematic and the information bits make up the first three bits of codewords.

The generator matrix for this code can be easily derived as,

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

Figure 2.23: The G matrix for the $(n,\ k) = (6,\ 3)$ linear code

Consider the BPSK modulated transmission of a message $b = [\ b_0\ b_1\ b_2\ ] = [\ 1\ \ 1\ \ 1\ ]$ over the AWGN channel using this code. This message is encoded using matrix multiplication $\boldsymbol{c} = \boldsymbol{b}G = [111000]$. The following are the details of the transmission and reception of the message.

Information: 111

Codeword: 111000 $\Longrightarrow$ Transmitted codeword = [111000] (0 info error)

Channel: [+1.0,+1.0,+1.0,-1.0,-1.0,-1.0] $\Longrightarrow$ BPSK modulated codeword

Received: [-0.7,+1.6,+0.4,-1.1,-1.1,-0.4] $\Longrightarrow$ Received codeword = [011000] (1 info error)

Probabilities: [0.15, 0.98, 0.74, 0.05, 0.05, 0.26] (using equation (2.17) at $SNR = 1/\sigma^2 = 1.239$ which is equivalent to $SNR(dB) = 0.931$)

Table 2.2 shows MAP decoding by codeword enumeration for the received codebit vectors using,

a) the probabilities that each of the received codeword bits is a 1 (i.e. p_j^1 for all $0 \leq j \leq n-1$) which are calculated using equation (2.17) (with $SNR = 1/\sigma^2 = 1.239 \equiv SNR(dB) = 0.931$), and

b) the probabilities that each of the received codeword bits is a 0 (i.e. p_j^0 for all $0 \leq j \leq n-1$, which are obtained as $p_j^0 = 1 - p_j^1$).

The probabilities that each of the received codebits is a 1, calculated using equation (2.17), are given as follows (rounded off to 2 decimal places).

P(1) = $[p_j^1$ for all $0 \leq j \leq n-1] \equiv [p_0^1, p_1^1, p_2^1, p_3^1, p_4^1, p_5^1]$ = [0.15, 0.98, 0.74, 0.05, 0.05, 0.26]

Accordingly, the probabilities that each of the received bits is a 0 are obtained using $p_j^0 = 1 - p_j^1$, and are given as follows,

P(0) = $[(1 - p_j^1)$ for all $0 \leq j \leq n-1] \equiv [p_0^0, p_1^0, p_2^0, p_3^0, p_4^0, p_5^0]$ = [0.85, 0.02, 0.26, 0.95, 0.95, 0.74]

These two sets of probability values, i.e. P(1) and P(0), are used to calculate the probabilities in the 'Probability' column of Table 2.2. The last six columns in the table are used for the decoding of the three information bits b_0, b_1, and b_2. That is, two columns are used to determine the output probability for each of the three information bits. For example, columns 'p_0^1' and 'p_0^0' of Table 2.2 are used to determine the output probability of information bit b_0 as follows.

i) In the 'p_0^1' column, in order to determine the probability of output bit b_0 being 1, the probabilities (from the 'Probability' column) of all the codewords for which $b_0 = 1$ are summed, and

ii) in the 'p_0^0' column, in order to determine the probability of output bit b_0 being 0, the probabilities (from the 'Probability' column) of all the codewords for which $b_0 = 0$ are summed.

As shown in the 'Output probability calculation' row of Table 2.2, the output probability that decoded information bit j is a 1, i.e. p_j^{o1}, is determined using,

$$p_j^{o1} = \frac{p_0^1}{p_j^1 + p_j^0} \quad \text{where } 0 \leq j \leq n-1 \tag{2.35}$$

Similarly, the output probability that decoded information bit j is a 0, i.e. p_j^{o0}, may be determined using,

$$p_j^{o0} = \frac{p_j^0}{p_j^1 + p_j^0} \quad \text{where } 0 \leq j \leq n-1 \tag{2.36}$$

Note that, $p_j^{o0} = 1 - p_j^{o1}$.

Table 2.2: MAP decoding by codeword enumeration [Ambroze, 2014]

Codeword	Probability	p_0^1	p_0^0	p_1^1	p_1^0	p_2^1	p_2^0
000000	$p_0^0 p_1^0 p_2^0 p_3^0 p_4^0 p_5^0 = 0.00295$		0.00295		0.00295		0.00295
100011	$p_0^1 p_1^0 p_2^0 p_3^0 p_4^1 p_5^1 = 0.00001$	0.00001			0.00001		0.00001
010101	$p_0^0 p_1^1 p_2^0 p_3^1 p_4^0 p_5^1 = 0.00267$		0.00267	0.00267			0.00267
110110	$p_0^1 p_1^1 p_2^0 p_3^1 p_4^1 p_5^0 = 0.00007$	0.00007		0.00007			0.00007
001110	$p_0^0 p_1^0 p_2^1 p_3^1 p_4^1 p_5^0 = 0.00002$		0.00002		0.00002	0.00002	
101101	$p_0^1 p_1^0 p_2^1 p_3^1 p_4^0 p_5^1 = 0.00003$	0.00003			0.00003	0.00003	
011011	$p_0^0 p_1^1 p_2^1 p_3^0 p_4^1 p_5^1 = 0.00761$		0.00761	0.00761		0.00761	
111000	$p_0^1 p_1^1 p_2^1 p_3^0 p_4^0 p_5^0 = 0.07265$	0.07265		0.07265		0.07265	
Summed probabilities		0.07276	0.01326	0.08301	0.00301	0.08031	0.00571
Output probability calculation		$p_0^{o1} = \frac{p_0^1}{p_0^1 + p_0^0} = 0.85$		$p_1^{01} = \frac{p_1^1}{p_1^1 + p_1^0} = 0.96$		$p_2^{o1} = \frac{p_2^1}{p_2^1 + p_2^0} = 0.93$	
Decoder output $[p_0^{o1}\ p_1^{o1}\ p_2^{o1}]$		0.85		0.96		0.93	
Decoded information $[b_0\ \ b_1\ \ b_2]$		1		1		1	

It can be seen from the results of the MAP decoding by codeword enumeration in Table 2.2 that the MAP decoder correctly decoded the received codeword from the channel, i.e. [011000] (which contains 1 information bit error), to the transmitted codeword, i.e. [111000], in spite of the error caused by the channel.

Iterative decoding works on probabilities in a H matrix. A probability matrix H_P is created by duplicating the H matrix and then replacing all the 1's in the duplicate H matrix with probability values of exactly 0.50, which represents an indeterminate state of the final output. The 0's in H_P, as inherited from duplicated H matrix, are not used in the decoding process. Figure 2.24 shows H_P the initialized probability matrix for the H matrix in Figure 2.22 which the BP/SPA decoder will work on.

$$H_P = \begin{array}{c} \begin{matrix} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 \end{matrix} \\ \begin{bmatrix} \_ & 0.50 & 0.50 & 0.50 & \_ & \_ \\ 0.50 & \_ & 0.50 & \_ & 0.50 & \_ \\ 0.50 & 0.50 & \_ & \_ & \_ & 0.50 \end{bmatrix} \end{array}$$

Figure 2.24: An initialized probability matrix H_P

Gallager's iterative decoding process is exemplified in Table 2.3 to Table 2.11. In order to explain the decoding process, the paths from the information which serve as the iterative decoder's inputs to the decoding results, i.e. value of decoded information bits b_0, b_1, and b_2, have been highlighted in Table 2.3 and Table 2.4. The function in equation (2.34), i.e. $f(p_a, p_b) = \frac{p_a p_b}{p_a p_b + (1-p_a)(1-p_b)}$, is used to combine and normalize probabilities. In order to combine more than two probabilities, say p_a, p_b and p_c, $f()$ is applied repeatedly, $f(f(p_a, p_b), p_c)$. The function in equation (2.33), i.e. $p_i^e = \frac{1-\prod_{j \neq i}(1-2p_j)}{2}$, is used to calculate the output extrinsic information.

Table 2.3: Decoding iteration **0**, equation **0**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.50 (?)	0.50 (?)	0.50 (?)		
equation 1	0.50		0.50		0.50	
equation 2	0.50	0.50				0.50
Decoding iteration 0, equation 0						
Input probabilities to SPC decoder:						
$p_1 = 0.98 = f(0.98,0.50)$			$p_2 = 0.74 = f(0.74,0.50)$			$p_3 = 0.05 = f(0.05)$
Output extrinsic information:						
$p_1 = \frac{1-(1-2p_2)(1-2p_3)}{2} = 0.71$			$p_2 = \frac{1-(1-2p_1)(1-2p_3)}{2} = 0.93$			$p_3 = \frac{1-(1-2p_1)(1-2p_2)}{2} = 0.26$
Received probabilities (iterative decoder input):						
$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$						
Current iterative decoder output:						
$p_0 = 0.15 = f(0.15,0.50,0.50)$			$p_1 = 0.99 = f(0.98,0.71,0.50)$			$p_2 = 0.97 = f(0.74,0.93,0.50)$
Decoded data: $b_0 = 0$ (because $p_0 < 0.50$), $b_1 = 1$ (because $p_1 > 0.50$), $b_2 = 1$ (because $p_2 > 0.50$)						
Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$						

Table 2.3 illustrates the decoding of the first equation (equation 0) in the first decoding iteration (decoding iteration 0). Similar to the MAP decoding by codeword enumeration in Table 2.2, the iterative decoder works with the probabilities that each of the received codebits is a 1, which are calculated using equation (2.17) at $SNR = 1/\sigma^2 = 1.239 \equiv SNR(dB) = 0.931$. These probabilities are given as follows (rounded off to 2 decimal places).

$$[p_0, p_1, p_2, p_3, p_4, p_5] = [0.15, 0.98, 0.74, 0.05, 0.05, 0.26]$$

This is the information contained in the 'From channel' row of the 'Currently known' section of Table 2.3. Additionally, the last three rows in the 'Currently known' section contain the probability matrix H_P which was created by replacing all the 1's in the H matrix with probability values of exactly 0.50, which represents an unknown state of the final output. These last three rows in the

'Currently known' section of Table 2.3 are called 'equation 0', 'equation 1', and 'equation 2', respectively. That is, the 'Currently known' section consists of the calculated channel probabilities, i.e. the 'From channel' row, and bit probabilities in the parity-check equations, i.e. the 'equation 0', 'equation 1', and 'equation 2' rows.

The decoding procedure illustrated in Table 2.3 is for decoding iteration 0, equation 0. It is the first decoding iteration and the first parity-check equation to be decoded. Therefore, only the channel values are available; there is no extrinsic information. It can be seen that the 'equation 0' row contains the probabilities of codeword bits b_1, b_2, and b_3, which are p_1, p_2, and p_3, respectively. Each of these probabilities are contained in the columns of corresponding name in the table, i.e. columns 'p_1', 'p_2', and 'p_3' respectively. At the start of the decoding, all the probabilities in the 'equation' rows of the 'Currently known' section are equal to 0.5.

As shown by the arrows drawn from columns 'p_1', 'p_2', and 'p_3' of the 'Currently known' section to the 'Input probabilities to SPC decoder' section of Table 2.3, the input probabilities to the SPC decoder p_1, p_2, and p_3 are obtained by combining the probabilities in columns 'p_1', 'p_2', and 'p_3' of the 'Currently known' section using equation (2.34); however, **the bit probabilities contained in the 'equation 0' row are excluded from these probability combinations**. That is, the values of p_j in the 'Input probabilities to SPC decoder' section are obtained by using equation (2.34) to combine the bit probabilities in the column 'p_j' of the 'Currently known' section of the table, **except** the bit probabilities values in the row of the equation being decoded.

As shown by the arrows drawn from the 'Input probabilities to SPC decoder' section to the 'Output extrinsic information' section of Table 2.3, the probabilities obtained in the 'Input probabilities to SPC decoder' section are used to calculate the output extrinsic information using equation (2.33). Subsequently, the output extrinsic information, i.e. p_1, p_2, and p_3, are used:

i) to update probabilities p_1, p_2, and p_3 of equation 0 in the probability matrix H_P for all further decoding operations, and

ii) determine the current decoder output.

In our exemplification of Gallager's iterative decoding process in Table 2.3 to Table 2.11, the 'Current iterative decoder output' probabilities are only determined for the information bits, i.e. b_0, b_1, and b_2 which have bit probabilities p_0, p_1, and p_2, respectively. Probabilities p_0, p_1, and p_2 represent the probabilities that information bits b_0, b_1, and b_2 respectively are equal to 1.

As shown in Table 2.3, the 'Current decoder output' probability p_j, where $0 \leq j \leq 2$, is determined by combining:

i) probability p_j in the 'From channel' row of the 'Currently known' section,

ii) the available probabilities in the 'equation' rows of column 'p_j' of the 'Currently known' section **except** that in the row of the equation being decoded (if applicable), and

iii) the newly calculated output extrinsic information of p_j (if applicable).

Table 2.4: Decoding iteration **0**, equation **1**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.71	0.93	0.26		
equation 1	0.50 (?)		0.50 (?)		0.50 (?)	
equation 2	0.50	0.50				0.50

Decoding iteration 0, equation 1

Input probabilities to SPC decoder:

$p_0 = 0.15 = f(0.15,0.50)$ $p_2 = 0.97 = f(0.74,0.93)$ $p_4 = 0.05 = f(0.05)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_2)(1-2p_4)}{2} = 0.92$ $p_2 = \frac{1-(1-2p_0)(1-2p_4)}{2} = 0.18$ $p_4 = \frac{1-(1-2p_0)(1-2p_2)}{2} = 0.82$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.67 = f(0.15,0.92,0.50)$ $p_1 = 0.99 = f(0.98,0.71,0.50)$ $p_2 = 0.88 = f(0.74,0.93,0.18)$

Decoded data: $b_0 = 1$ (because $p_0 > 0.50$), $b_1 = 1$ (because $p_1 > 0.50$), $b_2 = 1$ (because $p_2 > 0.50$)

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.4 illustrates the decoding of the second equation (equation 1) in the first decoding iteration (decoding iteration 0), which is the next step in the iterative decoding process. Similar to Table 2.3, the path from the information which serve as the iterative decoder input to the decoding results, i.e. value of decoded information bits b_0, b_1, and b_2, have been highlighted in Table 2.4.

It can be observed that the output extrinsic information produced in Table 2.3 have been used to update bit probabilities p_1, p_2, and p_3 in 'equation 0' row of the 'Currently known' section of Table 2.4. The decoding in Table 2.4 takes place in an identical manner to that in Table 2.3. However, for the decoding in Table 2.4, the decoder uses the channel values and the output extrinsic information produced by decoding equation 0.

Generally, after decoding the first equation, the decoder uses the channel values **and** any other extrinsic information produced by previous equations **except** the current one. A decoding iteration is complete after the output extrinsic information for the last equation in H_p has been determined. The first decoding iteration, i.e. decoding iteration 0, ends at Table 2.5

Further decoding iterations are executed by repeating this process of determining the output extrinsic information for all the parity-check equations again in a cyclic manner . Only three decoding iterations are used in our decoding exemplification. Table 2.6 to Table 2.8 show the second decoding iteration (decoding iteration 1), and Table 2.9 to Table 2.11 show the third decoding (decoding iteration 2).

Table 2.5: Decoding iteration **0**, equation **2**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.71	0.93	0.26		
equation 1	0.92		0.18		0.82	
equation 2	0.50	0.50				0.50

Decoding iteration 0, equation 2

Input probabilities to SPC decoder:

$p_0 = 0.66 = f(0.15,0.92)$ $p_1 = 0.99 = f(0.98,0.71)$ $p_5 = 0.26 = f(0.26)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_1)(1-2p_5)}{2} = 0.73$ $p_1 = \frac{1-(1-2p_0)(1-2p_5)}{2} = 0.57$ $p_5 = \frac{1-(1-2p_0)(1-2p_1)}{2} = 0.34$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.85 = f(0.15,0.92,0.73)$ $p_1 = 0.99 = f(0.98,0.71,0.57)$ $p_2 = 0.88 = f(0.74,0.93,0.18)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.6: Decoding iteration **1**, equation **0**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.71	0.93	0.26		
equation 1	0.92		0.18		0.82	
equation 2	0.73	0.57				0.34

Decoding iteration 1, equation 0

Input probabilities to SPC decoder:

$p_1 = 0.98 = f(0.98,0.57)$ $p_2 = 0.38 = f(0.74,0.18)$ $p_3 = 0.05 = f(0.05)$

Output extrinsic information:

$p_1 = \frac{1-(1-2p_2)(1-2p_3)}{2} = 0.39$ $p_2 = \frac{1-(1-2p_1)(1-2p_3)}{2} = 0.93$ $p_3 = \frac{1-(1-2p_1)(1-2p_2)}{2} = 0.61$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.85 = f(0.15,0.92,0.73)$ $p_1 = 0.97 = f(0.98,0.39,0.57)$ $p_2 = 0.88 = f(0.74,0.93,0.18)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.7: Decoding iteration **1**, equation **1**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.39	0.93	0.61		
equation 1	0.92		0.18		0.82	
equation 2	0.73	0.57				0.34

Decoding iteration 1, equation 1

Input probabilities to SPC decoder:

$p_0 = 0.32 = f(0.15,0.73)$ $p_2 = 0.97 = f(0.74,0.93)$ $p_4 = 0.05 = f(0.05)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_2)(1-2p_4)}{2} = 0.92$ $p_2 = \frac{1-(1-2p_0)(1-2p_4)}{2} = 0.33$ $p_4 = \frac{1-(1-2p_0)(1-2p_2)}{2} = 0.66$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.85 = f(0.15,0.92,0.73)$ $p_1 = 0.97 = f(0.98,0.39,0.57)$ $p_2 = 0.94 = f(0.74,0.93,0.33)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.8: Decoding iteration **1**, equation **2**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.39	0.93	0.61		
equation 1	0.92		0.33		0.66	
equation 2	0.73	0.57				0.34

Decoding iteration 1, equation 2

Input probabilities to SPC decoder:

$p_0 = 0.66 = f(0.15,0.92)$ $p_1 = 0.96 = f(0.98,0.39)$ $p_5 = 0.26 = f(0.26)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_1)(1-2p_5)}{2} = 0.72$ $p_1 = \frac{1-(1-2p_0)(1-2p_5)}{2} = 0.57$ $p_5 = \frac{1-(1-2p_0)(1-2p_1)}{2} = 0.35$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.72)$ $p_1 = 0.97 = f(0.98,0.39,0.57)$ $p_2 = 0.94 = f(0.74,0.93,0.33)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.9: Decoding iteration **2**, equation **0**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.39	0.93	0.61		
equation 1	0.92		0.33		0.66	
equation 2	0.72	0.57				0.35

Decoding iteration 2, equation 0

Input probabilities to SPC decoder:

$p_1 = 0.98 = f(0.98,0.57)$ $p_2 = 0.57 = f(0.74,0.33)$ $p_3 = 0.05 = f(0.05)$

Output extrinsic information:

$p_1 = \frac{1-(1-2p_2)(1-2p_3)}{2} = 0.56$ $p_2 = \frac{1-(1-2p_1)(1-2p_3)}{2} = 0.93$ $p_3 = \frac{1-(1-2p_1)(1-2p_2)}{2} = 0.43$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.72)$ $p_1 = 0.98 = f(0.98,0.56,0.57)$ $p_2 = 0.94 = f(0.74,0.93,0.33)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.10: Decoding iteration **2**, equation **1**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.56	0.93	0.43		
equation 1	0.92		0.33		0.66	
equation 2	0.72	0.57				0.35

Decoding iteration 2, equation 1

Input probabilities to SPC decoder:

$p_0 = 0.31 = f(0.15,0.72)$ $p_2 = 0.97 = f(0.74,0.93)$ $p_4 = 0.05 = f(0.05)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_2)(1-2p_4)}{2} = 0.92$ $p_2 = \frac{1-(1-2p_0)(1-2p_4)}{2} = 0.32$ $p_4 = \frac{1-(1-2p_0)(1-2p_2)}{2} = 0.67$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.72)$ $p_1 = 0.98 = f(0.98,0.56,0.57)$ $p_2 = 0.94 = f(0.74,0.93,0.32)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 2.11: Decoding iteration **2**, equation **2**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.56	0.93	0.43		
equation 1	0.92		0.32		0.67	
equation 2	0.72	0.57				0.35
Decoding iteration 2, equation 2						
Input probabilities to SPC decoder: $p_0 = 0.66 = f(0.15,0.92)$ $p_1 = 0.98 = f(0.98,0.56)$ $p_5 = 0.26 = f(0.26)$						
Output extrinsic information: $p_0 = \frac{1-(1-2p_1)(1-2p_5)}{2} = 0.73$ $p_1 = \frac{1-(1-2p_0)(1-2p_5)}{2} = 0.57$ $p_5 = \frac{1-(1-2p_0)(1-2p_1)}{2} = 0.34$						
Received probabilities (iterative decoder input): $p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$						
Final iterative decoder output: $p_0 = 0.85 = f(0.15,0.92,0.73)$ $p_1 = 0.98 = f(0.98,0.56,0.57)$ $p_2 = 0.94 = f(0.74,0.93,0.32)$						
Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$						
Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$						

Table 2.12 is used to compare the decoding results obtained using the MAP decoder and the iterative decoder.

Table 2.12: MAP decoding versus iterative decoding

Decoder Type	Decoder Output [p_0 p_1 p_2]	Decoded information [b_0 b_1 b_2]	Information errors	Source
MAP decoder	[0.85 0.96 0.93]	[1 1 1]	0	Table 2.2
Iterative decoder	[0.85 0.98 0.94]	[1 1 1]	0	Table 2.11

It can be seen from the results in Table 2.12 that the iterative decoder has a similar performance to the MAP decoder, and both decoders correctly decoded the received information from the channel to the transmitted codeword despite the error caused by the channel.

For more powerful codes where both k and $n-k$ are large, MAP decoding complexity is of the order of $min(2^k, 2^{n-k})$. However, iterative decoding is significantly less complex with a complexity proportional to $(n-k) \times n_i$ where n_i is the number of iterations performed [Ambroze, 2014]. Generally, long codes where n, k, and $n-k$ are large also have large girths g. Consequently, these long codes need to be decoded using larger values of n_i. This is due to the fact that the minimum n_i required to attain a basic level of decoding convergence under iterative message-passing algorithms is equal to the girth of the code g, i.e. $min(n_i) = g$. However, researchers commonly set $n_i \geq 10g$ in order to obtain the best iterative decoding performance.

Algorithm 2.3 summarises the BP/SP decoding algorithm which was implemented in this research for decoding BPSK modulated LDPC codes received over the AWGN channel at different SNRs. Using the Monte Carlo method, the coding performance of most of the LDPC codes investigated in this thesis were determined using this decoder. Algorithm 2.3 describes only the auxiliary processes surrounding the BP/SP algorithm and excludes the preliminary aspects of the larger BP/SPA program which must be executed before actual decoding commences. That is, Algorithm 2.3 excludes the LDPC matrix configuration file read operation, and the processes leading up to the derivation of the G matrix - all of which can be done offline.

An improved BP/SP algorithm is proposed and discussed in Chapter 7.

Algorithm 2.3: The BP/SPA message-passing iterative decoding algorithm for decoding BPSK modulated LDPC codes received over AWGN channels

$blockerrors = 0$, $biterrors = 0$, $dminerrors = 0$
for $tcodewords = 1$ **to** $max.no\ of\ codewords\ in\ simulation$ **do**
begin

- generate a length k random message $\boldsymbol{u} = [u_0\ u_1\ u_2\ \ldots\ u_{k-1}]$ using the values of $\boldsymbol{d_v}$, i.e. the random deviates returned by the ***ran2*** function, to determine the value of each bit in $\boldsymbol{u}$. For example, if $|\boldsymbol{d_v}| \geq 0.5$, $u_0 = 1$, if $|\boldsymbol{d_{v+1}}| < 0.5$, $u_1 = 0$, and so on up to u_k.
- using $\boldsymbol{c} = \boldsymbol{u}G$, determine the corresponding n bit long codeword $\boldsymbol{c} = [c_0\ c_1\ c_2\ \ldots\ c_{n-1}]$.
- convert each bit of codeword $\boldsymbol{c}$ to the equivalent vector to be transmitted using $bit\ 0 = -1.0$ and $bit\ 1 = +1.0$, this simulates BPSK modulation to obtain the set of transmitted vectors $\boldsymbol{t}$, where $\boldsymbol{t} = [t_0\ t_1\ t_2\ \ldots\ t_{n-1}]$ and, depending on the binary value of c_j, $t_j = \pm 1.0$ for all $0 \leq j \leq n-1$.
- generate AWGN vectors $N = [N_0\ N_1\ N_2.\ \ldots\ N_{n-1}]$ to distort the transmitted vectors $\boldsymbol{t}$ in each bit position using $\boldsymbol{N_j} = \boldsymbol{\sigma_N} \times \boldsymbol{gasdev_v}$, for $0 \leq j \leq n-1$. Recall: a new function call is made to the ***gasdev*** function for each j.
- add the transmitted vector $\boldsymbol{t}$ to noise vector $\boldsymbol{N}$ to obtain the received vector $\boldsymbol{r} = [r_0,\ r_1,\ r_2\ \ldots\ r_{n-1}]$ where $r_j = t_j + N_j$ for $0 \leq j \leq n-1$.
- calculate the probabilities of the received bits r_j (probabilities that each bits is a 1) to obtain the code bit probabilities $P = [p_0, p_1, p_2, \ldots\ , p_{n-1}]$ where $0.00 \leq p_j \leq 1.00$ for all $0 \leq j \leq n-1$.
- create probability matrix H_P, from the original H matrix in the configuration file such that $bit\ 0 \rightarrow$ '$-$' and $bit\ 1 \rightarrow 0.50$.

$error = 0$
for $iterations = 1$ **to** $max.iterations$ **do**
begin

execute BP/SPA decoding, as exemplified in Table 2.3 to Table 2.11, using probability matrix H_P and the code bit probabilities P which were calculated from the received values.

end

- convert the decoded bit probabilities, i.e. $P = [p_0, p_1, p_2, \ldots\ , p_{n-1}]$, obtained after the final iteration into bits, using **if** $p_j \geq 0.5$ **then** $c'_j = 1$, **else** $c'_j = 0$, for all $0 \leq j \leq n-1$;
  the decoded codeword $\boldsymbol{c'} = [c'_0\ c'_1\ c'_2\ \ldots\ c'_{n-1}]$ is thus obtained
- compare the bits of the decoded output $\boldsymbol{c'}$ to those in the transmitted codeword $\boldsymbol{c}$,
  if all corresponding bits are identical, i.e. $\boldsymbol{c'_j} = \boldsymbol{c_j}$ for all $0 \leq j \leq n-1$, **then** $error = 0$, and
  if there are x differences, where $x > 0$, **then** $error = 1$
- calculate the syndrome of the decoded output using $s = c'H^T$
- **if** $error = 1$, **then** block$errors = blockerrors + 1$ **and** $biterrors = biterrors + x$
- **if** $error = 1$ **and** syndrome $s = 0$, **then** $dminerrors = dminerrors + 1$
- calculate and display the error rates, P_{ei} at a specified $tcodewords$ intervals.
  interim FER, $P_{eif} = blockerrors/tcodewords$
  interim BER, $P_{eib} = biterrors/(tcodewords \times n)$

end
output: $FER\ P_{ef} = blockerrors\ /\ max.no\ of\ codewords\ in\ simulation$
output: $BER\ P_{eb} = biterrors\ /\ (max.no\ of\ codewords\ in\ simulation \times n)$
output: $dminerrors$

Although the BP/SPA decoding algorithm implemented has the facility for error probabilities in terms of both FER and BER, in this thesis we focus mainly on the FER of LDPC codes. In order to minimize

decoding simulation times, the decoded codeword after each iteration $\boldsymbol{c}' = [c'_0\ c'_1\ c'_2\ \ldots\ c'_{n-1}]$ is compared to the transmitted codeword $\boldsymbol{c} = [c_0\ c_1\ c_2\ \ldots\ c_{n-1}]$, and whenever they are identical the decoding iteration cycle is terminated, and then the next random message is generated for reception over the simulated AWGN channel. This saves time by preventing the maximum number of decoding iterations from being executed whenever not necessary.

As a result of the minimum distance of codes, there are d_{min} *decoding errors* where the information received from the channel is decoded to a codeword which is different from the transmitted codeword, d_{min} decoding errors results in errors (codewords) which have zero syndrome. d_{min} decoding errors often occur when the channel distorts the transmitted codeword in such a manner that the received signals are *more likely* (MRL) to have been produced by the codeword which it is eventually decoded to (known as the MRL codeword). In other words, the Euclidean distance between the received codeword and the transmitted codeword is greater than the Euclidean distance between the received codeword and the MRL codeword which it is eventually decoded to [Papagiannis et al., 2004]. As a result of the zero syndrome characteristic associated with this type of error events, they are classified as *undetectable* errors at receivers. Conversely, decoding errors which do not result in valid codewords, and hence do not have zero syndromes, are classified as *detectable* errors. Decoding error probabilities are determined using the total number of decoding errors which is the sum of detectable and undetectable errors.

In Algorithm 2.3, the variable $dminerrors$ is a counter for the total number of undetectable errors which occur during a performance simulation, and the algorithm displays the total number of undetectable errors at the end of the simulation. The Hamming distance between the transmitted codeword and the decoded codeword for each undetectable error is also determined and displayed by the BP/SPA decoder. This facility helps to distinguish between d_{min} decoding errors and other causes of decoding failures, e.g. trapping sets (see Section 3.2.5). In fact, this output can be used as an indirect method to determine the d_{min} of a code because the Hamming distances between the transmitted codeword and the decoded codeword is never less than the d_{min} of the code.

2.6 Performance of the Implemented BP/SPA Iterative Decoder

Unless otherwise specified, the following rules are applied to all LDPC code performance simulations and comparisons carried out using the BP/SPA decoder implemented in this research: -

- The maximum number of decoding iterations is 100.
- For statistically accurate decoding error probabilities, a minimum of 100 frame errors are captured before the probability of decoding error (FER) at any Eb/No is determined.
- Identical noise patterns are used to compare two or more codes plotted in the same graph. This is done by: using identical random number seeds for codeword and noise generation, simulating the decoding of an identical number of codewords for each code, and determining the probability of decoding error (FER) only after attempting to decode all the codewords.
- Whenever it is easy to capture a minimum of 100 decoding errors, i.e. in the waterfall region, a minimum and maximum of 1,000,000 and 10,000,000 codewords, respectively, are processed for results.
- Where it is difficult to capture the required minimum of 100 decoding errors, i.e. in the error-floor region, the minimum and maximum number of codewords processed for results varies according to the length and efficiency of the code. A minimum of $100 \times {}^{1}/_{P_e}$ codewords are processed for the FER results at the Eb/No of simulation, where P_e is the probability of frame error at the Eb/No obtained at the end of the simulation. The required number of codewords for the performance simulation is estimated at the beginning of the simulation and adjusted (increased) as necessary until a minimum of 100 decoding errors are captured at any given Eb/No for each codes being compared.

The performance of the BP/SPA decoder implemented in this research was verified by using it to determine the performance of LDPC codes whose performances have been previously determined using a BP/SPA decoder in another research and comparing the results. The LDPC codes used for the verification are: an $[n,\ k,\ d] = [512, 256, 14]$ code, which is a rate-½ code with minimum distance $d_{min} = 14$; and an $[n,\ k,\ d] = [1024,\ 512, 16]$ code, which is a rate-½ code with minimum distance

$d_{min} = 16$. These codes were constructed using the PEG algorithm in the PhD research work carried out in [Tjhai, 2007]. They are used as benchmark codes; results obtained from their decoding performance simulations using the BP/SPA decoder implemented in this research are used as the benchmark of good performance for comparison to other LDPC codes of similar length and rate.

Using these two benchmark codes, Figure 2.25 shows a comparison of the decoding performance simulation results reported in [Tjhai, 2007] (called [Tjhai, 2007]) and the simulation results obtained using the BP/SPA decoder implemented in this research (called [Decoder]).

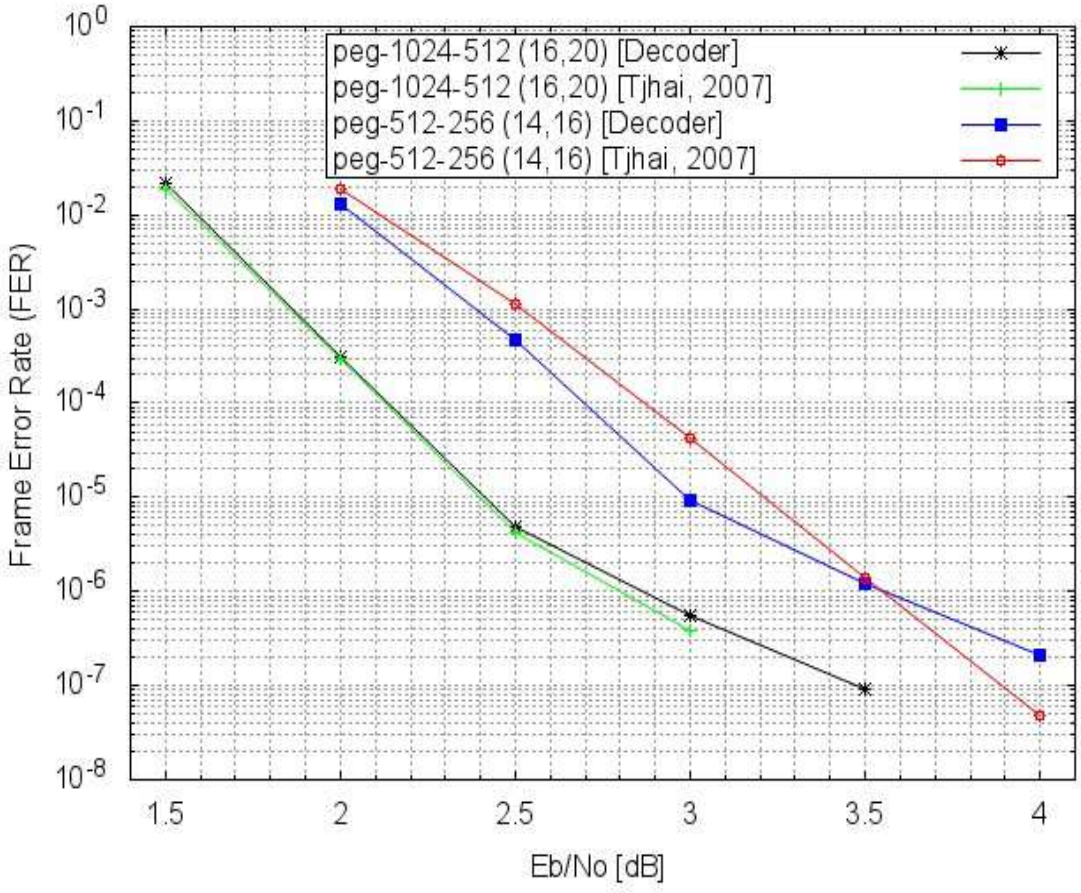

Figure 2.25: Performance verification for the BP/SPA decoder implemented

The results of the $[n, k, d] = [1024, 512, 16]$ LDPC code performance simulations obtained using our BP/SPA decoder are very similar to the results reported for the same code in [Tjhai, 2007], with the reported performance being marginally better. However, the performance simuation results for the $[n, k, d] = [512, 256, 14]$ code obtained using our BP/SPA decoder is notably better in the waterfall

region and worse in the error-floor region compared to the results reported for the same code in [Tjhai, 2007]. These dissimilarities may be attributed to precision differences between the decoder implementations and/or the different noise patterns encountered.

2.7 Determination of LDPC Code Minimum Weights: Minimum Distance Weights d_{min}, and Minimum Stopping Set Weights, s_{min}

The performance of LDPC codes under iterative message-passing decoding is dominated by graphical structures within the Tanner graphs of the codes on which these decoding algorithms rely for their operation. Over the binary erasure channel (BEC), at low erasure probabilities, the performance of LDPC codes are dominated by error-prone graphical structures called *stopping sets*, which were introduced in [Di et al, 2002]. Over the AWGN channel at high SNRs, the performance of LDPC codes are dominated by code minimum distances d_{min}, and error-prone graphical structures within codes called *trapping sets* (see Section 3.2.5).

2.7.1 Stopping Sets and Minimum Stopping Set Weights s_{min}

- **Stopping Sets:** A stopping set $\mathcal{S}$ is a subset of the set of symbol nodes V_s, such that all the neighbours of the symbol nodes in $\mathcal{S}$ are connected to $\mathcal{S}$ at least twice. The size or weight of a stopping set s is defined as the cardinality of $\mathcal{S}$, i.e. $|\mathcal{S}|$.
- **Minimum Stopping Set Weights s_{min} (Stopping Distances):** The minimum stopping set weight, or stopping distance, s_{min} of a code is the minimum weight (cardinality or size) of a nonempty stopping set of the code.

2.7.2 Linear Code Minimum Distance d_{min} and Stopping Distance s_{min} Evaluation

The problem of determining the d_{min} of binary linear codes was hypothesized to be NP-hard by [Berlekamp et al., 1978]. Two decades later, this hypothesis was affirmed by [Vardy, 1997]. Similarly, [Krishnan and Shankar, 2006], and other researchers have shown that determining the s_{min} of an arbitrary Tanner graph is an NP-hard problem. Various algorithms to approximate the d_{min} and s_{min}

of fixed LDPC code matrices have resulted from the inherent intractability of the problem. Some of these algorithms can be found in [Hu et al., 2004], [Hirotomo et al., 2005], [Hu and Eleftheriou, 2006], [Richter, 2006], [Krishnan and Shankar, 2007], [Declercq and Fossorier, 2008], and [Hirotomo et al., 2008].

In spite of the fact that exhausting all low-weight error-prone patterns is NP-hard, algorithms for determining the minimum weight spectra for short-to-medium block length LDPC code matrices exist. Notable examples include algorithms proposed by [Wang et al., 2006], [Wang et al., 2009], [Rosnes and Ytrehus, 2009], [Rosnes et al., 2012], and [Rosnes et al., 2014].

One of the most efficient algorithm for determining the minimum weight spectra for short-to-medium block length linear LDPC code matrices is the algorithm proposed by [Rosnes et al., 2012]. This algorithm, which is an improvement to the algorithm by [Rosnes and Ytrehus, 2009], implements a complete *branch-and-bound* (B&B) tree search [Land and Doig, 1960] in a short-to-medium length LDPC code graphs to determine the initial part of the codeword weight spectrum and stopping set weight spectrum up to a specified target τ (where $\tau \leq 32$). The minimum distance d_{min} and stopping set minimum weight s_{min} of a code are extracted from the weight spectra found during a B&B tree search. All the LDPC code d_{min} and s_{min} values determined in this research were carried out using the algorithm proposed by [Rosnes et al., 2012], which we shall refer to as the *LDPC code weight evaluation algorithm* in the rest of this thesis.

2.8 Summary

- LDPC codes and Tanner graphs have been introduced, and the basic features and properties of LDPC codes have been described in detail. The different types of LDPC codes and LDPC code construction techniques have been discussed.
- A detailed description of the standard/original PEG algorithm for constructing LDPC codes of arbitrary length and rate has been given. The standard PEG algorithm which was implemented in this research has been fully described. All modified PEG algorithms are based on and incorporate the basic aspects of the standard/original PEG algorithm as described in this chapter.

- The format of the contents of LDPC matrix configuration files created by the different PEG algorithms investigated in the research has been described in detail. LDPC matrix configuration files contain the LDPC code parity-check matrices constructed by PEG algorithms. The simulated decoding performance of the LDPC matrices in these configuration files are subsequently determined using iterative message-passing BP/SPA decoders.
- A detailed and simplified explanation of the iterative decoding processes in the standard BP/SPA decoder has been given. The algorithm of the standard BP/SPA decoder which was implemented for this research has been fully described. The standard BP/SPA decoder was used to simulate the decoding performance of most of the LDPC codes analysed in this research. An improved BP/SPA decoder will be introduced and used for code performance simulations in Chapter 7.
- The NP-hard problem of determining the minimum weights, i.e. d_{min} and s_{min}, of LDPC codes has been highlighted, and the algorithm for determining the codeword weight and stopping set weight spectra in short-to-medium block length LDPC code matrices proposed by Rosnes et al. has been introduced and briefly described. The minimum weights of the short-to-medium length LDPC codes analysed in this research were determined using the *LDPC code weight evaluation algorithm* described in [Rosnes et al., 2012].

Chapter 3

Challenges to lowering the error-floor of short-to-medium length LDPC codes over AWGN channels, and modified PEG algorithms

Following the independent rediscovery of Gallager's LDPC codes by [MacKay and Neal, 1996], and also by [Wiberg, 1996], extensive research into reducing the gap to capacity of LDPC codes has been ongoing. Challenges encountered in the diverse effort to lower the error-floor of LDPC codes have been extensively expatiated in the literature.

3.1 Introduction

In this chapter we highlight and discuss the many challenges to lowering the error-floor of short-to-medium length LDPC codes under iterative message-passing decoding. Additionally, out of the diverse effort to overcome the identified challenges through improved LDPC code constructions, we discuss, examine and review some notable PEG algorithm modifications that were aimed at lowering the error-floor of the short-to-medium length LDPC codes constructed.

3.2 Challenges to Lowering the Error-Floor of Short-to-Medium Length LDPC Codes over AWGN Channels

In this section the features of LDPC codes which have been identified in literature to affect the performance of short-to-medium length LDPC codes in the error-floor region are discussed, with

special focus on the AWGN channel and decoding using iterative message passing algorithms. Information theory has established that one of the primary determinants of the performance of a code is its length. This applies to any type of code, code rate and girth. Unfortunately, the use of long codes comes with significant computational overhead which inevitably results in longer delays. While many applications have a maximum acceptable delay which limits the length of codes that can be used, e.g. voice communications, other applications have a minimum acceptable error rate, e.g. data storage. Relatively short delays are incurred when short-to-medium length LDPC codes are used, and successfully lowering their error-floor performance would make them efficient enough for deployment to many low-delay and low-error-rate applications.

3.2.1 Girth Properties of LDPC Code Tanner Graph

In their rediscovery of Gallager's LDPC codes, MacKay and Neal used code construction methods which ensured that 4-cycles were not present in the Tanner graphs of the regular LDPC codes they experimented upon. This undoubtedly allowed the potentials of LDPC codes, which Gallager had invented about three decades earlier, to be revealed. It was soon apparent from empirical performance results that the performance of their regular LDPC codes were almost as close to the Shannon limit as that of Turbo codes [MacKay and Neal, 1996].

On the one hand, there is ample proof that linear block codes like LDPC codes which have cycle-free Tanner graphs have poor decoding performance not only under iterative message-passing decoding schemes but also under maximum likelihood decoding [Etzion et al., 1999]. On the other hand, the presence of cycles in the graph of LDPC codes causes symbol nodes to become interdependent (or correlated) after a small number of decoding iterations, which significantly reduces decoding efficiency under iterative message-passing algorithms. In his study of capacity-approaching LDPC codes, Sason showed that cycles are necessary in the Tanner graph of such codes [Sason, 2009].

Extensive research in the search for good LDPC codes have shown that the performance of an LDPC code depends not only on the length of the shortest cycle in the graph, i.e. the girth of the graph, but also on the number of shortest cycles, recent examples of such research include

[Mao and Banihashemi, 2001] and [Venkiah et al., 2008]. This implies that to identify the features of capacity-approaching LDPC codes it is important to look into their local girth distribution in addition to their global girths.

3.2.2 Regular vs. Irregular LDPC Codes

The original LDPC codes, as first proposed by Gallager, were regular codes. Similarly, using regular LDPC codes, MacKay and Neal showed that the performance of LDPC codes were almost as close to the Shannon limit as that of Turbo codes [MacKay and Neal, 1996]. However, Luby et al. made a very important contribution to LDPC code construction in 2001 when they showed that the performance of LDPC codes can be significantly improved by incorporating some degree of irregularity in the construction of their parity-check matrices [Luby et al., 2001]. In the same year, Richardson et al. proposed a method to arrive at optimized parity-check matrix irregularities in terms of *symbol node degree distributions*, which lead to good decoding performance of LDPC codes of different length and rate [Richardson and Urbanke, 2001], [Richardson et al., 2001]. The degree distribution optimization method they employed is known as *density evolution* (DE). DE determines the performance *threshold* for infinitely long LDPC codes under the assumption that their bipartite graphs are tree-like. However, the Tanner graphs of practicable finite-length LDPC codes without singly connected symbol nodes unavoidably have cycles and hence are not tree-like [Tian et al., 2004]. The presence of cycles in bipartite graphs compromises the optimality of BP/SPA decoding because cycles result in a situation where neighbours of a node are not conditionally independent in general, consequently *graph separation* (see [Pearl, 1988]) is imprecise. [Tian et al., 2003]

Randomly constructed rate-½ irregular LDPC codes with block lengths in the order of 10^4 were found to approach their DE threshold within $0.8dB$ [Richardson et al., 2001], these codes outperformed their regular counterparts by approximately $0.6dB$ [MacKay, 1999]. An irregular binary LDPC code, which had a DE optimized symbol node degree distribution and a block length of 10^7, was shown to attain a distance of only $0.04dB$ from the ultimate Shannon limit at a bit error probability of 1×10^{-6} [Chung et al., 2001]. Extensive research carried out to date show that irregular LDPC codes with good

degree distributions perform significantly better than regular LDPC codes of identical length and rate. These performance differences can be seen in both structured and randomly constructed LDPC codes.

3.2.3 The Minimum Distance d_{min} of LDPC Codes

Generally, there appears to be an inverse relationship between the minimum distance d_{min} properties of the different types of LDPC codes and their iterative decoding convergence over the AWGN channel. Regular LDPC codes generally have larger d_{min} than irregular LDPC codes of the same length and rate but do not perform as well. Similarly, irregular LDPC codes with relatively large d_{min} usually have 'bad' degree distributions and have poor performance compared to irregular LDPC codes constructed with 'good' degree distributions. The best performing short-to-medium length LDPC codes known to date are random (unstructured) irregular LDPC codes which have DE optimized or 'good' symbol node degree distributions.

The minimum distances d_{min} of LDPC codes in an ensemble whose constituent codes are differentiated only by the random number seeds used in their construction usually spans a wide range of values. We use an ensemble of six thousand (6000) rate-½ $(n, k) = (512, 256)$ LDPC codes as an example. All the codes in the ensemble were constructed using DE optimized symbol node degree distribution $Q_1(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$ in the standard PEG algorithm. The value of the minimum distance d_{min} of the codes in this ensemble ranged from 7 to 19. Simulation experiments show that there is a generalized improvement in the iterative message-passing decoding performance of these codes over the AWGN channel as the d_{min} increases from the lowest value of 7. However, beyond a certain d_{min} value (in this case, for $d_{min} > 14$) there is no observable generalized improvement in decoding performance with increasing d_{min}. In other words, it cannot be said for the ensemble of LDPC codes in our example that codes with $d_{min} = 19$ generally perform better than codes with $d_{min} = 14$ etc. Therefore, while high d_{min} values are desirable to minimize undetectable decoding errors, beyond some values of d_{min} other causes of decoding errors dominate the performance LDPC codes. This is particularly true in the error-floor region.

3.2.4 The Approximate Cycle Extrinsic Message Degree (ACE), and the Minimum Stopping Set Weight, s_{min} of LDPC Codes

Arguably, following the adoption of DE optimized degree distributions, the most remarkable improvements to short-to-medium length irregular LDPC code error-floor performance were brought about by modifications made to LDPC code construction algorithms in order to increase the *approximate cycle extrinsic message degree* (ACE) in the Tanner graph of the codes constructed.

Tian et al. introduced a metric called *extrinsic message degree* (EMD) in their work to lower the error-floor of irregular LDPC codes [Tian et al., 2003]. Their ultimate goal was to attain a minimum stopping set size in LDPC code graphs, however the algorithm they realized did not explicitly remove small stopping sets. Focusing on the EMD of symbol node sets instead, they proposed a technique to suppress small stopping sets by ensuring that cycles have enough neighbours that produce useful message flows. An extrinsic check node of a symbol node set is a check node that is singly connected to that set of symbol nodes. The extrinsic message degree (EMD) of a symbol node set is the number of extrinsic check nodes of the set. Therefore, from the definition of stopping sets (see Section 2.7.1): i) the EMD of a stopping set is zero and, ii) a symbol node set with a large EMD will require a significant number of additional symbol nodes to become a stopping set, i.e. for 'closure'.

Tian et al. proposed a conditioning algorithm to ensure that all cycles less than a given length have EMDs greater than a given value. Statistically, the technique increased the size of the smallest stopping set s_{min} in LDPC codes. Consider a generic cycle in the Tanner graph of an LDPC code, if there are no symbol nodes in this cycle that share common check nodes outside of this cycle, i.e. the cycle contains no subcycles, then the EMD of this cycle is $\sum_i (d_i - 2)$, where d_i is the degree of the i-th symbol node in this cycle, and the summation is taken over all the symbol nodes in the cycle. If the cycle contains subcycles, the EMD is reduced through check node sharing. In order to simplify the EMD metric, the effects of check node sharing is considered to be negligible and the *approximate* EMD *of a cycle* was defined.

The approximate cycle EMD (ACE) of a cycle in a Tanner graph is $\sum_i(d_i - 2)$, where d_i is the degree of the i-th symbol node in this cycle, and the summation is taken over all the symbol nodes in the cycle. In a cycle, the ACE of symbol node s_j is $(d_{s_j} - 2)$, where $0 \leq j \leq n - 1$, and the ACE of any check node is 0. Note that while ACE is defined with respect to symbol nodes in a given cycle, EMD is defined with respect to any symbol node set. Additionally, given a set of symbol nodes in a cycle, ACE is an upper bound on the EMD of these symbol nodes. [Tian et al., 2003] [Tian et al., 2004]

Xiao and Banihashemi proposed a PEG algorithm modification for constructing short-to-medium length LDPC codes with significantly lower error-floors. The proposed PEG algorithm modification, based on the ideas in [Tian et al., 2003] and [Tian et al., 2004], increases the ACE of cycles in the Tanner graph of LDPC codes in order to increase Tanner graph cycle connectivity [Xiao and Banihashemi, 2004]. From simulation experiments, it is clear that modifying the PEG algorithm to increase the ACE in the Tanner graph of short-to-medium length LDPC codes results in codes with significantly improved performance at high SNR region without any trade-off in the low SNR performance. The improved error-floors of these codes are primarily attributed to the Tanner graph ACE improvements. However, unlike in [Tian et al., 2003] and [Tian et al., 2004] where the improved code performance was additionally attributed to improved stopping sets, the improved code performance in [Xiao and Banihashemi, 2004] was additionally attributed to improved *trapping sets* (see Section 3.2.5). Therefore, it may be deduced that there is a direct relationship between the minimum stopping set size s_{min} and the size of the smallest trapping sets in LDPC codes.

Vukobratović and Šenk proposed a *Generalized ACE constrained* PEG algorithm, a modified PEG algorithm for constructing LDPC code graphs with considerably improved ACE properties compared to what is obtainable using the improved PEG (IPEG) construction by Xiao and Banihashemi. The main objective was to achieve even further improvements in the error-floor performance of short-to-medium length irregular LDPC codes. The work explored the possibility of further increases to LDPC code stopping set sizes by conditioning cycles with respect to their ACE properties. The simulation results presented showed improvements of the proposed algorithm, in terms of average and

best elements performance, over the standard PEG and IPEG algorithms [Vukobratović and Šenk, 2008].

3.2.5 Trapping Sets

Margulis proposed the Cayley graph construction of regular Gallager (LDPC) codes of rate-½ and parameters $(j, k) = (3, 6)$ [Margulis, 1982], the performance of these codes was first investigated by Rosenthal and Vontobel [Rosenthal and Vontobel, 2000]. MacKay and Postol attributed the early error-foor of the length $n = 2640$, girth $g = 8$, Margulis code, which appears at a FER of about 1×10^{-6}, to what they called *near-codewords* and not low-weight codewords. A (w, v) near-codeword of a code with parity-check matrix H is defined as a vector x with weight w whose syndrome, i.e. $s(x) = xH^T$, has a weight of v. Near-codewords with small v and relatively small w tend to be error states from which the standard BP/SPA decoding algorithm cannot break out. A small w relates to a reasonably probable error pattern, and a small v implies that only a few parity-check equations are affected by these error patterns. Typically, a (w, v) near-codeword has v check nodes that are connected to the bits in the word only once [MacKay and Postol, 2003].

In the continued effort to characterize failures of iterative decoding schemes over AWGN channels, particularly in the error-floor region, Richardson introduced the notion of *trapping sets* and proposed a computational technique that allows for the prediction of code error-floors in regions which are beyond the reach of simulation [Richardson, 2003]. Richardson extended the concept of near-codewords and redefined these error causing states into the more generalized term - trapping sets. Richardson defined trapping sets as sets of symbol nodes, which usually consist of a relatively small number of symbol nodes, such that the induced subgraph has only a small number of odd degree check nodes. Under iterative decoding, a decoding is defined as *successful* if all the bits of a codeword are eventually correct. Given an input y, the *failure set* $T(y)$ is defined to be the set of bits that are not eventually correct. Therefore, decoding is successful on y if and only if $T(y) = \emptyset$. If $T(y) \neq \emptyset$, then $T(y)$ is a trapping set. T is said to be an (a, b) trapping set if it has a symbol nodes and b odd degree check node neighbours in the subgraph induced by T.

Graphs with the smallest (dominant) trapping sets, e.g. $(3, 1)$ trapping sets, have the worst performance in the error-floor region. These relatively small trapping sets give rise to failure events that dominate the error-floor performance of LDPC codes under iterative message-passing decoding. However, the size of a trapping set is not linearly related to decoding failure rate as it is not the only relevant parameter. Given two trapping sets (a, b) and (a', b), with $a > a'$ and fixed b, it sometimes happens that (a, b) trapping sets fail at a higher rate than (a', b) trapping sets. It is feasible to discover, enumerate and evaluate the trapping sets involved in the error-floor, i.e. dominant trapping sets, because their structures are relatively small and their cardinality is not too large.

In practice, trapping sets are identified under simulation in the following way. A large fixed maximum number of decoding iterations (say 100) is performed unless the decoder converges to a codeword earlier. If it has not converged to a codeword after the fixed maximum number of iterations, the trapping set is identified as the union of all symbol nodes which have not been decoded to the correct bit value. Take the decoding of a codeword of length $n = 512$ code as an example, where the symbol nodes of this code s_j have indices within the range $0 \leq j \leq 511$. If the five (5) symbol nodes with index numbers 0, 37, 117, 499, and 506 have not been decoded to the correct bit value after the 100th decoding iteration, then this is a trapping set $T_1 = \{s_0, s_{37}, s_{117}, s_{499}, s_{506}\}$. This trapping set contains five (5) symbol nodes, therefore $a = |T_1| = 5$. If the subgraph induced by T_1 has only one degree 1 check node, i.e. $b = 1$, we describe T_1as a $(5, 1)$ trapping set.

We formally define trapping sets and some of its more harmful subclasses, and then briefly explain the processes through which a soft decision message-passing iterative decoder becomes trapped as follows. Let G be the Tanner graph of a binary LDPC code C given by the null space of an $m \times n$ parity-check matrix H over GF(2). The following definitions apply for $0 \leq a \leq n$ and $0 \leq b \leq m$:

a) An (a, b) trapping set is a set T of a symbol nodes in G which induces a subgraph of G, denoted by $G(T)$, with exactly b odd degree check nodes and an arbitrary number of even degree check nodes.

b) An (a, b) trapping set is an *elementary* trapping set if all the check nodes in the induced subgraph $G(T)$ have degree 1 or degree 2, and there are exactly b degree 1 check nodes.

c) An (a, b) trapping set is a *small* trapping set if $a \leq \sqrt{n}$ and $\frac{b}{a} \leq 4$.

d) An (a, b) trapping set is an *absorbing set* if every symbol node in the trapping set is connected in $G(T)$ to fewer check nodes of odd degree than check nodes of even degree. Furthermore, if every symbol node not in the trapping set is connected to fewer check nodes of odd degree in $G(T)$ than other check nodes, i.e. check nodes not in $G(T)$ or in $G(T)$ but of even degree, then the trapping set is a *fully absorbing set*.

On the AWGN channel, error patterns with a small number of symbol errors occur with higher probability than error patterns with large number of symbol errors. As a consequence, the most harmful (a, b) trapping sets in message-passing decoding algorithms are usually those with small values of a and b. The results of simulations carried out in [MacKay and Postol, 2003] and [Richardson, 2003], and other extensive studies carried out since then show that the trapping sets that result in high decoding failure rates and contribute significantly to high error-floors are those with small a values and small $\frac{b}{a}$ ratios (especially for $\frac{b}{a} \leq 1$). The notions of *elementary* trapping sets and *small* trapping sets capture these conclusions. In general, trapping sets with relatively large b compared to a result in relatively small decoding failure rates and have little contribution to error-floor.

For the binary symmetric channel (BSC), the notion of *absorbing* sets arises from the fact that if the channel causes errors in the symbol nodes corresponding to an absorbing set, then a Gallager type-B decoder [Gallager, 1962] or a one-step majority-logic decoder [Lin and Costello, Jr., 2004] will fail. Similarly, for soft-decision iterative decoding such as the BP/SPA over the AWGN channel, if most of the soft messages become saturated then the decoder will behave like a Gallager type-B decoder and will fail. To avoid numerical overflow due to probability saturation, which occasionally happens in isolated error events in the error-floor region, the magnitude of the soft messages (e.g. extrinsic

probabilities) are usually limited to within some finite values of saturation [Ambroze et al., 2000], [Butler and Siegel, 2012].

To understand trapping sets, let us assume that in the transmission of a codeword in C, an error pattern e which results in a errors at the locations of the a symbol nodes of an $(a,\ b)$ trapping set occurs. As a result of this error pattern there will be b parity-check failures. Under iterative decoding, another iteration must be carried out to correct the failed parity-check equations. Iterative decoding algorithms, such as the BP/SPA and the min-sum algorithm (MSA), are very susceptible to trapping sets of a code because they work locally in a manner similar to distributed-processing; each check node has a local processor unit to process the messages received from the symbol nodes connected to it and each symbol node has a local processor unit to process the messages received from the check nodes connected to it. It is expected that these local processor units, through message exchanges and iterations, collect sufficient information to make a globally optimum decision of the transmitted code bits.

In the course of each decoding iteration, a check node is referred to as a *satisfied* check node if it satisfies its corresponding check-sum constraint (i.e. the sum of the variable node bits in its corresponding parity-check equation is equal to zero), otherwise, it is referred to as an *unsatisfied* check node. In the process of decoding, the decoder undergoes *state transitions*, from one state to another, until all the check nodes satisfy their corresponding check-sum constraints or a predetermined maximum number of decoding iterations is attained. The q-th state of an iterative decoder is represented by the hard-decision of decoded sequence obtained at the end of the q-th iteration. During a decoding iteration, the messages from the satisfied check nodes try to reinforce the current decoder state, while the messages from the unsatisfied check nodes try to change some of the bit decisions to satisfy their check-sum constraints. If errors affect the a symbol nodes of an $(a,\ b)$ trapping set T, the b odd-degree check nodes which are each connected an odd number of symbol nodes in T, will not be satisfied while all the other check nodes will be satisfied. In such situations, the decoder will successfully correct the errors in T if the messages from the unsatisfied check nodes connected to the

symbol nodes in T are strong enough to overcome the messages coming from the satisfied check nodes. However, this is often not the case if the value of b is small. Consequently, the decoder might not converge to a valid codeword even if more decoding iterations are performed, and the decoder is said to be *trapped*. This non-convergence of decoding gives rise to the phenomenon of the error-floor.

Additionally, as decoding iterations increase and all parity-check equations are satisfied for a codeword, the positions of the non-zero bits in the codeword form an $(a,\ 0)$ trapping set, where a is the codeword weight. If an error pattern determined by these bit positions occurs, the decoder converges to an incorrect codeword which results in an undetected error. In such cases, the decoder is permanently trapped and cannot decode to the correct codeword. [Diao et al., 2013]

Following the introduction of the concept of trapping sets, a considerable amount of research has been dedicated to identifying, enumerating, and minimizing the effects of trapping sets on the error-floor performance of LDPC codes. Some of the more notable work on trapping sets, especially those which focus on short-to-medium length LDPC code performance improvements, will be briefly discussed in the following.

Many algorithms have been proposed for identifying and enumerating the trapping sets present in Tanner graphs of codes. Vasić et al. derived a systematic method to identify the most relevant trapping sets for decoding over the BSC in the error-floor region. They adopted a notion called the *critical number* as an indicator of the harmfulness of a trapping set so that the smaller the critical number the more harmful a trapping set. Trapping sets were classified and organized in a database known as the *trapping set ontology* on the basis of the relations between the topological structures. By exploiting the *parent-child* relationships between different subgraphs, the trapping set ontology can be used to simplify the enumeration of trapping set classes. The trapping set ontology is also useful for error-floor estimation, and in code construction where the harmful structures can be avoided while constructing the Tanner graph of codes [Vasić et al., 2009]. Using a geometric approach, Diao et al. analyzed the general trapping set structure of finite geometry (FG)-LDPC codes, which includes Euclidean geometry (EG)-LDPC codes and projective geometry (PG)-LDPC codes. In their approach,

the trapping sets in the Tanner graph of a FG-LDPC are represented by sub-geometries of the particular geometry on which the code is constructed. The size and configuration of a trapping set of a given size can be determined using this geometric approach. Diao et al. showed that the Tanner graph of a FG-LDPC code with a minimum distance of d_{min} contains no small (a, b) trapping sets with $a < (d_{min} - 3)$. Given that the d_{min} of FG-LDPC codes are relatively large, they inherently do not have small trapping sets. Consequently, FG-LDPC codes generally have lower error-floors than similar regular codes, and this is further enhanced by the row-redundancy of their parity-check matrices [Diao et al., 2013]. Many other trapping set enumeration algorithms have been proposed in literature, recent examples are the algorithms in [Kyung and Wang, 2000], [Wang et al., 2009], [Abu-Surra et al., 2010], [Zhang and Siegel, 2011], [Hashemi and Banihashemi, 2015] and [Hashemi and Banihashemi, 2016]. In a recent work on trapping set enumeration, a branch-and-bound (B&B) algorithm for exhaustively enumerating the elementary trapping sets (ETSs) in an arbitrary Tanner graph was proposed. For a given Tanner graph G and a positive integer v, a linear programming based B&B procedure was formulated for the NP-hard problem of finding the minimum w for which there exists an (w, v)-ETS in G. The proposed algorithm does not require the Tanner graphs it analyses to be of particular forms (e.g., symbol node regular), unlike most trapping set enumeration algorithms proposed before it [Falsafain and Mousavi, 2016].

The discovery of trapping sets has also spurred research to improve the error-floor performance of LDPC codes by modifying the various code construction algorithms to pre-empt and avoid creating small trapping sets in LDPC code Tanner graphs during code construction. In the pioneering work by Xiao and Banihashemi entiled *Improved PEG (IPEG) construction of Irregular LDPC codes*, which lowered the error-floor of PEG constructed LDPC codes by improving their ACE, the authors asserted that the increase in connectivity of the cycles in the graph improves not only the minimum distances d_{min} of the codes constructed but also their trapping sets. In the error-floor, d_{min} contributes to undetected errors, while trapping sets are responsible for detected errors. These assertions were supported by a comparison of the minimum distances and trapping sets of the irregular

$(n,\ k) = (1008,\ 504)$ rate-½ standard PEG and IPEG codes whose decoding simulation performance curves were presented. While the PEG code had a minimum distance of $d_{min} = 12$, the IPEG code had an improved minimum distance of $d_{min} = 15$. Similarly, the smallest $(a,\ b)$ trapping set found in the PEG code was a $(6,\ 1)$ trapping set, whereas the smallest trapping set found in the IPEG code was a $(15,\ 1)$ trapping set. [Xiao and Banihashemi, 2004]. The error-floor performance improvements as a result of improved cycle connectivity in irregular LDPC codes constructed using the IPEG construction compared to the standard PEG algorithm are undisputable. However, it is important to note that the minimum distance and trapping set improvements realised by the IPEG construction are collateral gains. The modifications made to the standard PEG algorithm to obtain the IPEG algorithm were made solely to increase the connectivity of cycles in the LDPC code Tanner graphs constructed; the algorithm modification was not a directly attempt to increase the minimum distances or the minimum size of the trapping sets of the LDPC codes constructed.

To obtain LDPC codes with low error-floors over the binary symmetric channel (BSC), Nguyen et al. proposed a method to construct LDPC codes without certain small trapping sets in their Tanner graphs. The trapping sets which were to be avoided during code constructions were selected from the trapping set ontology for the Gallager A/B decoder based on their relative harmfulness for a given decoding algorithm. However, only symbol node regular codes with $(d_s = 3)$ were evaluated in the work [Nguyen et al., 2012].

Similarly, to improve the error-floor performance of PEG constructed LDPC codes over the AWGN channel, Khazraie et al. modified the original PEG algorithm by equipping it with a subalgorithm which, before connecting each edge in an under-construction Tanner graph, identifies the candidate check nodes which will lead to dominant elementary trapping sets (ETSs), and then avoids connecting such edges to prevent the creation of dominant ETS's. Unlike other algorithms to prevent the creation of small trapping sets during code construction, which are restricted to symbol node regular codes, the proposed algorithm is among the first ones which are applicable to both regular and irregular symbol node degree distributions [Khazraie et al., 2012]. Following the work by Khazraie et al., a universal

algorithm for finding the dominant trapping sets in an arbitrary LDPC code graph was proposed by Karimi and Banihashemi. The proposed universal algorithm was observed to be approximately two orders of magnitude faster than previously existing dominant trapping set search algorithms. The algorithm was used to estimate the error-floor of regular and irregular LDPC codes. The proposed algorithm may be modified to find other graphical objects such as Zyablov-Pinsker trapping sets, which dominate LDPC code error-floor performance across different channels and different iterative decoding schemes, and absorbing sets. It may also be used in LDPC code construction algorithms to design codes with low error-floors [Karimi and Banihashemi, 2012(a)].

In a recent work, Kang et al. proposed a technique for breaking trapping sets during the iterative message-passing decoding of LDPC codes in an effort to improve the performance of LDPC codes in the error-floor region. The proposed technique for breaking trapping sets, called *collaborative decoding*, uses two different decoding modes: a) a main decoding mode that executes message passing based on an LDPC code's original parity-check matrix as usual, and b) a sub-decoding mode that executes message passing on a modified parity-check matrix derived by removing a portion of the check nodes in the factor graph representation of the LDPC code. Whenever necessary, the collaborative LDPC decoding scheme switches intelligently between the two decoding modes. The changes made to obtain the modified parity-check matrix from the original parity-check matrix are meant to promote the passing of correct information into erroneous symbol nodes in trapping sets. Collaborative LDPC decoding approach is applicable to both regular and irregular codes, and does not require prior knowledge of the trapping sets in the code. Although the proposed collaborative LDPC decoding scheme had better performance than decoding using a single decoder, it required a higher number of decoding iterations. However, compared to existing *backtracking* methods [Kang et al., 2011] the collaborative decoding scheme required significantly less iterations to achieve significant error-floor reduction [Kang et al., 2016].

3.3 Modified PEG Algorithms

In this section, a review of literature on the original PEG algorithm and some noteworthy PEG algorithm modifications aimed at improving the performance of the LDPC codes constructed is undertaken. Special attention is given to recent literature on improving the error-floor performance of short-to-medium length LDPC codes which have their construction based on the PEG principle. The review focuses on the objectives of the different PEG algorithm modifications and examines the consequent properties of the parity-check matrices/Tanner graphs constructed in relation to the reported code performance. The properties and parameters of LDPC codes and Tanner graphs that are of primary interest include their: symbol node degree distributions, global girths, local girth distributions, minimum distances d_{min}, minimum stopping set weights s_{min}, and trapping sets.

3.3.1 The Original/Standard PEG Algorithm

In their pioneering paper on the PEG algorithm for LDPC code construction, Hu et al. proposed the PEG algorithm and used it to construct a rate-½ $(n, m) = (504, 252)$, symbol node regular ($d_s = 3$) code and compared its performance to that of the following two codes [Hu et al., 2001]:

i) a MacKay code with similar parameters; rate-½ $(n, m) = (504, 252)$, regular with $d_s = 3$ and $d_c = 6$, and

ii) a 4-cycle free randomly constructed code; rate-½ $(n, m) = (504, 252)$, symbol node regular with $d_s = 3$.

The girth histograms for the three codes revealed that all the codes were 4-cycle free; the PEG Tanner graph had a global girth of 8 (three of its symbol nodes had local girths of 10) while the MacKay graph and the random graph both had global girths of 6 (with local girths of 6 or 8 only). At a maximum of 80 decoding iterations, the BER and FER rates of the PEG code were significantly lower than those for the MacKay code and the random code in the error-floor region. The MacKay code in turn performed significantly better than the random code in the error-floor region. In the waterfall region, the performance of the PEG code and MacKay code were very similar and both performed

significantly better than the random code. It was hypothesized that the degraded performance of the random code in both the waterfall and error-floor regions, despite having the same global girth as the MacKay code, was likely due to its lower girth histogram which had the highest percentage of symbol-nodes with local girths of only 6, and hence the lowest percentage of symbol nodes with local girths of 8.

Similar results were obtained for 4-cycle free rate-½ $(n, m) = (1008, 504)$ regular codes; the performance of a PEG code, a MacKay code and a random code with these parameters were compared. The PEG code had a global girth of 8 (with local girths of 8 or 10 only) and the MacKay and random graphs both had global girths of 6 (with local girths of 6, 8, and 10). Similar to the case of the $(504, 252)$ codes, although both the MacKay code and the random codes had global girths of 6, their girth histograms reveal that the random code had a higher percentage of local girths of 6 compared to the MacKay code. Consequently, the random code had the worst decoding performance. It was suggested that not only the girth but also the local girth distribution dominates the performance of iterative decoding. Indeed, the local girth distribution has been used as a heuristic tool for selecting good codes from random graphs for short block lengths [Mao and Banihashemi, 2001].

Finally, Hu et al. presented performance simulations for three irregular codes based on the arbitrary symbol node degree distribution: $\lambda(x) = 0.5x^2 + 0.2x^3 + 0.2x^4 + 0.075x^5 + 0.025x^7$. The three graphs evaluated were: a PEG Tanner graph with the *zigzag* pattern, a PEG graph without the zigzag pattern, and an irregular repeat-accumulate (IRA) random graph. Simulation results showed that both PEG codes (with and without the zigzag pattern) had almost identical BER and FER performances, and they both performed significantly better than the random IRA code. It was asserted that, with an appropriately optimized degree sequence, the performance of irregular PEG codes could be much better.

In a follow up to their pioneering paper on the PEG algorithm [Hu et al., 2005], irregular LDPC codes were constructed with DE optimized symbol node degree distributions using the PEG algorithm. The performance of five (5) DE optimized rate-½ $(n, m) = (504, 252)$ codes which have maximum

symbol node degrees - $d_{s(\max)}$ of 4, 7, 11, 15 and 30 were investigated. The performance of these PEG codes were compared to those of five (5) 4-cycle free irregular random graphs that had the same symbol node degree distributions which varied only in terms of their maximum symbol node degrees, i.e. $d_{s(\max)}$ of 4, 7, 11, 15 and 30. The PEG codes performed significantly better than the randomly constructed codes, especially in the high-SNR region. It was observed that the PEG code with $d_{s(\max)} = 15$ had the best performance. Subsequently, it was concluded that even at short block lengths, using DE optimized degree distributions in the PEG algorithm resulted in better LDPC codes. The performance of irregular PEG LDPC code Tanner graphs with a linear-time-encoding property, i.e. an LDPC code whose parity-check matrix is forced into an upper triangular form, was also investigated in the work. The BERs and FERs of an irregular PEG code, an irregular PEG code with its parity-check matrix in the upper triangular form, and a regular MacKay code were compared. Here, the three codes compared were rate-½ $(n, m) = (1008, 504)$ codes, and both of the irregular PEG codes were constructed using DE optimized symbol node degree distribution: $\lambda(x) = 0.23802x^2 + 0.20997x^3 + 0.03492x^4 + 0.12015x^5 + 0.01587x^7 + 0.00480x^{14} + 0.37627x^{15}$. It was observed that both of the irregular PEG codes had nearly identical performance, and they significantly outperformed the regular MacKay code. Consequently, it was suggested that linear-time encoding can be achieved with the PEG construction without noticeable performance degradation.

The reason that the optimized symbol-node-degree distribution with a maximum symbol-node degree $d_{s(\max)}$ of 15 was observed to achieve the best performance may be explained as follows: The codes with symbol-node-degree distributions with $d_{s(\max)} < 15$, i.e. those with $d_{s(\max)}$ of 4, 7, and 11, although most likely to be 4-cycle free and have good girths, would have had insufficient edge densities for the message passing decoding algorithm to perform optimally. Conversely, it is very likely that the code with symbol-node-degree distribution with $d_{s(\max)} = 30$ had a significant number of 4-cycles as a result of an excessive edge density, and this implies that its symbol nodes had high correlation which results in poor decoding performance by BP/SPA message passing algorithms. The

code with $d_{s(\max)} = 15$ was likely to be 4-cycle free (or almost) and had the best edge density, and consequently had the best overall decoding performance compared to the others.

3.3.2 An Improved PEG (IPEG) Construction of Irregular LPDC Codes

An improved construction method for irregular LDPC codes using the PEG algorithm was proposed by Xiao and Banihashemi in 2004. This was achieved by a simple modification to the PEG algorithm which maximizes the degree of connectivity of each new cycle created during a graph construction to the rest of the Tanner graph. The rationale behind having a high degree of connectivity of cycles in Tanner graphs is that such cycles will receive a large amount of extrinsic information from the rest of the graph and consequently would not impose as much harm on iteration convergence as they would if they were more isolated [Xiao and Banihashemi, 2004].

We adopt the same definitions and notations used for the standard PEG algorithm in [Hu et al., 2001] to describe the *improved PEG Construction* (IPEG) algorithm by Xiao and Banihashemi. Recall that in PEG Tanner graph construction, symbol nodes are connected into graphs according to their degrees in the nondecreasing order, the degree of a symbol node s_j is denoted as d_{s_j}, $\{N_{s_j}^l\}$ describes the set of all check nodes reached by a subgraph expanded from s_j within depth l, and $\{\dot{\mathrm{N}}_{s_j}^l\}$ describes the complementary set respectively. Recall also that it is often the case that when a new edge of s_j, i.e. $E_{s_j}^k$ $(0 \leq k \leq d_{s_j} - 1)$, is to be connected into a graph, multiple candidate check-nodes are frequently available to choose from. The PEG algorithm modification focused on situations where $k \geq 1$ and $\dot{\mathrm{N}}_{s_j}^l \neq \emptyset$ but $\dot{\mathrm{N}}_{s_j}^{l+1} = \emptyset$, where for each of the lowest degree candidate check-node in $\dot{\mathrm{N}}_{s_j}^l$, the establishment of a new edge in the graph creates new cycles of length $(2l + 2)$. Let $\odot_{s_j}^k$ denote the set of candidate check-nodes for connecting the (k+1)-th edge of symbol node s_j. In this modification, the check node whose associated cycles will have the highest degree of connectivity to the rest of the graph is selected.

The *approximate cycle extrinsic message degree* (ACE) is used as a measure of the connectivity of a cycle to the rest of a Tanner graph. The ACE is evaluated using $\sum_i (d_i - 2)$; the summation is taken

over all symbol nodes in the cycle, and d_i is the degree of the i-th symbol-node. The ACE is a count of the number of edges by which a cycle in a graph is connected to the rest of the Tanner graph through its symbol nodes. A high degree of interconnectivity for the new cycles is enforced by selecting the candidate check-node in $\odot_{Sj}^{k}$ that maximizes the minimum ACE for the new cycles. In the event that more than one check-node in $\odot_{Sj}^{k}$ results in the maximum value of the minimum ACE for the new cycles, a check node is randomly selected from amongst them.

The performance of two pairs of irregular rate-½ PEG constructed LDPC codes were compared in the work. All the codes compared were constructed using a DE optimized degree distribution with $d_{s(max)} = 15$ given by: $\lambda(x) = 0.23802x^2 + 0.20997x^3 + 0.03492x^4 + 0.12015x^5 + 0.01587x^7 + 0.00480x^{14} + 0.37627x^{15}$. Two of these codes were $(n, m) = (504, 252)$ and the other two were $(n, m) = (1008, 504)$ codes. For each pair of codes, with lengths $n = 504$ and $n = 1008$ respectively, one code was constructed using the standard PEG and the other one using the IPEG construction. Code performance analysis was carried out using BP/SPA decoding of BPSK modulated signals over an AWGN channel. The maximum number of decoding iterations was 80, simulations were run until at least 100 codewords were erroneously decoded and the same noise vectors were used for each pair of equal length codes.

From the simulation results which were presented in BER and FER curves for the four codes, it was observed that at high SNR, the improved PEG construction codes significantly outperformed standard PEG codes of identical length. However, at lower SNR the decoding performances of pairs of codes from the two PEG algorithms were very similar.

In their analysis of the lower error-floor of the IPEG construction compared to the standard PEG construction for the pair of codes with $(n, m) = (1008, 504)$, the following observations were made: the standard PEG code had $d_{min} = 12$ and the modified PEG code had an improved minimum distance of $d_{min} = 15$, and the standard PEG code had a minimum trapping set weight of 6 and the modified PEG code had an improved minimum trapping set weight of 15. Similar improvements in and trapping sets were observed in the pair of codes with $(n, m) = (504, 252)$. On the basis of these

observations it was concluded that the IPEG construction algorithm improved both the minimum distances and the trapping sets of the LDPC codes it constructs. The smaller percentage of undetected errors for the IPEG code compared to the standard PEG code (10% and 36% respectively) was attributed to the higher d_{min} of the former. The IPEG algorithm constructs LDPC codes with higher degrees of cycle connectivity in their Tanner graphs than the standard PEG algorithm. From the simulation results, it was clear that the IPEG algorithm enhances the performance of PEG LDPC codes at high SNR region without any trade-off in the low SNR performance.

3.3.3 A Modified PEG Construction for Irregular LPDC Codes Without Small Stopping Sets

Richter and Hof worked on an irregular LDPC code construction method based on the PEG algorithm to design codes without small stopping sets [Richter and Hof, 2006]. The PEG algorithm was modified in such a way that during Tanner graph construction, whenever there exists a choice of multiple candidate check-nodes to connect an edge of symbol node s_j, the algorithm identifies and avoids connections which could lead to small stopping sets and chooses others. It was also proffered that this PEG modification, i.e. prevention of small stopping sets, also led to the construction of LDPC codes with higher minimum distances because all codewords with small Hamming weights are caused by small size stopping sets.

In the standard PEG algorithm, whenever a multiple choice exist to connect a variable node with a check-node, the choice is made either on the basis of its index (usually the candidate check-node with the smallest index) or a check-node is chosen randomly. The PEG algorithm modifications proposed by Richter and Hof are effected just before this final check-node selection stage such that additional sub-algorithm(s) is/are inserted to evaluate the options and then choose a candidate check node which prevents a connection that could lead to a small stopping set. In this PEG modification, a random selection of a candidate check-node for connecting an edge of s_j is made only if a multiple choice of candidate check nodes still exist after the *small stopping set(s) prevention* stage.

Using the same notations and definitions for the PEG algorithm in [Hu et al., 2001] and [Hu et al., 2005], we note the following:

i) The PEG algorithm modifications made are only applicable when the number of degree-2 symbol nodes is smaller than m.

ii) The small stopping set prevention algorithm uses check node distance metric $d_c(i, j)$, which is the distance between check nodes c_i and c_j, in its evaluation of the check-nodes that could result in small stopping set and marks such candidate check nodes as not selectable.

iii) An important input to the modified algorithm is the value of the desired minimum stopping set weight $\hat{s}$ which serves as a target for the algorithm to work with.

iv) If all the check-nodes in the set $\dot{\mathrm{N}}_{s_j}^{l}$ are marked as not selectable, a check-node from the set $\dot{\mathrm{N}}_{s_j}^{l-1}$ that is not likely to lead to a small stopping set is chosen.

In code performance evaluation, LDPC codes constructed by the modified PEG algorithm are compared to those constructed using the standard PEG algorithm by Hu et al., and also with codes constructed using the improved PEG (IPEG) construction algorithm (for improved ACE) of [Xiao and Banihashemi, 2004]. Fifty (50) rate-½ $(n, m) = (1000, 500)$ codes with degree distribution $\lambda(x) = 0.283x^2 + 0.281x^3 + 0.436x^9$ were generated for each of the three algorithms. The different stopping set weights, the cardinality of stopping sets of each weight, and the average minimum distances for the code ensemble constructed using each algorithm were tabulated. The tabulated results suggest that the ensemble of codes generated by the IPEG construction algorithm had higher minimum stopping set weights than codes in the standard PEG algorithm ensemble. However, the code ensemble constructed by the modified PEG algorithm had the highest minimum stopping set weights and significantly outperformed the others algorithms in this regard. The average minimum distances of the code ensembles followed a similar trend. The codes with the best combination of minimum distance d_{min} and minimum stopping set weights s_{min} from both the standard PEG algorithm ensemble and the IPEG construction algorithm ensemble were given as $s_{min} = d_{min} = 15$ in both cases. For the modified PEG algorithm the best minimum distance parameter combination was $s_{min} = d_{min} = 18$.

Performance simulations over an AWGN channel were carried out using the *shuffled* belief propagation decoder by [Zhang and Fossorier, 2005]. Decoding simulations were carried out using the *linear approximation* detailed in [Richter et al., 2005], and a maximum of 500 decoding iterations. A plot of the FER vs. E_b/N_0 was presented for four codes selected from the code ensembles constructed as follows: the two codes which had the lowest and highest s_{min} values in the code ensemble constructed using the standard PEG algorithm, and the two codes which had the lowest and highest s_{min} values in the code ensemble constructed using the modified PEG algorithm. All four codes had similar performances in the low SNR region. However, in the high SNR region, the results presented shows that both of the codes constructed using the modified PEG algorithm performed better that both of the codes constructed using the standard PEG algorithm. Similar results were obtained for coding over the binary erasure channel (BEC) using an efficient erasure decoding algorithm which is detailed in [Luby et al., 2001].

Similar results were also presented for ensembles of fifty (50) rate-¾ codes constructed with parameters $(n, m) = (2000, 500)$ and a degree distribution given by: $\lambda(x) = 0.1245x^2 + 0.4460x^3 + 0.4078x^{11} + 0.0213x^{12}$. The improved performances of the LDPC codes constructed using the modified PEG algorithm were generally attributed to their improved minimum stopping set weight s_{min} properties.

Unfortunately, there is no evidence in the work that any of the symbol node degree distributions used for code constructions are DE optimized degree distributions. No mention is made of the girth properties of the codes constructed using either of the two PEG algorithms which were compared in simulations. The performance of codes constructed using the proposed modified PEG algorithm were not compared to the performance of codes constructed using the IPEG construction algorithm in [Xiao and Banihashemi, 2004] which have significantly improved ACEs.

It is debatable that the modified algorithm constructed codes with good girth properties (zero to very low fractions of 4-cycle symbol nodes) because the algorithm permits a reduction of the maximum subgraph expansion depth attained in order to select a candidate check node from the set $\dot{\mathrm{N}}_{s_j}^{l-1}$ if all the

check nodes in the set $\dot{\mathrm{N}}_{s_j}^{l}$ are marked as not selectable because they are likely to lead to a small stopping set if chosen. Permitting subgraph expansion depth l reductions and selecting candidate check nodes from set $\dot{\mathrm{N}}_{s_j}^{l-1}$ inevitably results in reduced girths in the Tanner graphs constructed.

3.3.4 A Randomized PEG (RandPEG) Algorithm

The PEG algorithm was modified by Venkiah et al. in order to improve the girth properties of the resulting Tanner graphs in terms of increasing the global girth g achievable by the algorithm and/or minimizing the multiplicity of the cycles of girth g [Venkiah et al., 2008]. The modified PEG algorithm by Venkiah et al. was called the *RandPEG*. In their work, only regular $(d_v,\ d_c)$ graphs were considered in order to compare the results to known bounds for regular graphs. Here d_v refers to the uniform degree of the symbol nodes (or variable nodes) and d_c refers to the uniform degree of the check nodes in a regular Tanner graph. Denote N as the length of an LDPC code, for a given regular graph setting represented by the triple parameters $(d_v,\ d_c,\ g)$, $N_g^{(d_v,\ d_c)}$ denotes a lower bound on N such that a $(d_v,\ d_c)$ graph of girth g exists.

The two main differences between the RandPEG and the standard PEG algorithm are: a different method of subgraph expansion, and the introduction of an *objective function* which is used for edge selection. The overall objective is to attain a given target girth g_t after all edges have been connected into a Tanner graph. During code construction, whenever it becomes impossible to establish an edge without decreasing the target girth g_t, the algorithm sees it as a failure and restarts the construction from the beginning. This procedure is executed a very large number of times and the best codes are stored. In the RandPEG, subgraph expansion is nongreedy and the maximum subgraph expansion depth is limited to a chosen value l_{max}. Following the *diameter argument,* a justification based on the target girth g_t, the maximum subgraph expansion depth used was given as $l = (g_t\text{-}2)/2$. It was further proffered that using such a nongreedy approach reduced the probability of construction failure, i.e. the inability of the algorithm to construct graphs with girth $g \geq g_t$.

In the general case when the graph size N is sufficiently large such that a regular (d_v, d_c) graph of girth g may exist, i.e. $N \geq N_g^{(d_v, d_c)}$, the objective function is used to discriminate amongst and choose the best candidate(s) in order to attain the target girth by minimizing the number of created cycles. After a subgraph expansion up to a maximal depth of l_{max} is carried out, the objective function discriminates amongst the set of candidate check-nodes in the set $\dot{\mathrm{N}}_{Sj}^{lmax}$ using the following sequence:

i) If candidates are found at depth l_{max}, mark all candidates not at l_{max} as unusable.

ii) For each available candidate c_j, evaluate $nbCycles_j$; the number of new cycles that will result from selecting c_j. Mark all candidates that have $nbCycles_j > min_j(nbCycles_j)$ as unusable.

iii) Calculate the lowest degree of the remaining candidates d_c^{min}, mark all the candidates that currently have degree $d_c > d_c^{min}$ as unusable.

iv) If there is more than one eligible candidate check node at this point, choose one randomly.

Further refinements for the subgraph expansion were applied whenever, for a given target girth g_t, the length N of the code is such that $N_g^{(d_v, d_c)} < N < N_{g+2}^{(d_v, d_c)}$ and the diameter argument failed to hold. Using the combination of i) a gap variable such that the maximal depth $l_{max} = (g_t + gap\text{-}2)/2$, ii) adjustments of the gap values as necessary and, iii) the objective function, the multiplicity of the target girth g_t in the graph was minimized. A typical value used for the gap variable is 2.

The RandPEG algorithm was used to construct binary and non-binary ultra-sparse graphs ($d_v = 2$) within a wide range of d_c with notably improved girth properties compared to the standard PEG algorithm. For the binary codes constructed, the belief propagation (BP) decoder was used for code performance simulations. Codes were simulated over the binary symmetric channel (BSC) and the FERs were calculated based on 100 frames in error. For the binary LDPC code analysis, the RandPEG algorithm was used to construct regular $(d_v = 3, d_c = 6)$ LDPC codes of two (2) lengths $N = 504$ and $N = 1008$. The performance of the $N = 504$ RandPEG code was compared to that of two other

regular codes with identical parameters; one constructed with the standard PEG algorithm and the other is a code optimized by MacKay. Similarly, the performance of the $N = 1008$ RandPEG code was compared to that of two other regular codes with identical parameters; one constructed with the standard PEG algorithm and the other a code optimized by MacKay. All the graphs simulated and plotted had a girth of 8. However, in both cases, the RandPEG codes were reported to have significantly smaller fractions of cycles of length 8 than the corresponding code constructed by the standard PEG algorithm. These differences were attributed to the effectiveness of the *objective function*. From the simulation results, it appeared that the girth multiplicity minimization of the RandPEG codes improved the performance of such codes in the error-floor region.

While it was emphasized that the limitation of the study to regular graphs is not a limitation of the RandPEG algorithm itself, it remains unverified that there would be any improvements in the girth properties of the resultant codes if the modified PEG algorithm were applied to construct irregular graphs using optimized symbol-node degree distributions. It can be observed from the graphical plots that the performance difference between the RandPEG codes and the standard PEG codes were not significant.

3.3.5 A Generalized ACE Constrained PEG Algorithm for Irregular LDPC Codes

Vukobratović and Šenk proposed a generalization of the PEG algorithm in order to construct LDPC code graphs with considerably improved approximate cycle extrinsic message degree (ACE) properties [Vukobratović and Šenk, 2008]. This modified PEG algorithm is known as the *generalized ACE constrained PEG algorithm*. The objective of the modification is to substantially increase the ACE properties, i.e. the degree of connectivity of the new cycles created during the PEG construction to the rest of the Tanner graph, compared to what is obtainable by the improved PEG (IPEG) construction algorithm of [Xiao and Banihashemi, 2004]. The main target of the work was to achieve further improvements to the error-floor performance of irregular PEG LDPC codes. Noting that cycles are always present in subgraphs induced from stopping sets; the work is an extension of previous work by Xiao and Banihashemi and explores the possibilities of further increases to LDPC code stopping set sizes by conditioning cycles with respect to their ACE properties. In effect, the generalized ACE

constrained PEG algorithm constructs LDPC codes with extremal (as large as possible) ACE spectrum components. The main modifications made to the standard PEG algorithm in the work comprised of two independent procedures which were applied sequentially for establishing a new edge in the Tanner graph; the *Check Node Shortest Path Discovery* and the *Check Node Selection* procedures. We adopt the definitions and notations introduced in [Hu et al., 2001] to describe these two procedures.

The Check Nodes Shortest Paths Discovery procedure used the subgraph expansion procedure of the standard PEG algorithm to create a list of all the shortest paths and their corresponding ACE metric between the root symbol-node, s_j and every candidate check node c_i. Consider the procedure for connecting the k-th edge $E_{s_j}^k$ of symbol node s_j, where $k > 0$ and the Tanner graph is connected under the current code construction setting. Recall from [Hu et al., 2001] that $N_{s_j}^l$ is the set of neighbouring check-nodes of s_j within depth l. The set $C_{s_j}^l$ is used to denote the set of check nodes within the shortest path distance of $(2l + 1)$ edges from s_j, so that $C_{s_j}^l = N_{s_j}^l \backslash N_{s_j}^{l-1}$. Subsequently, for each check node $c \in C_{s_j}^l$, the set of all distinct shortest paths from s_j to c_i are identified as the set P_{s_j,c_i}^l. Finally, for each shortest paths in P_{s_j,c_i}^l, the ACE metric is calculated. Recall that the maximization of the ACE of the newly created shortest cycles is the objective of the modification to the PEG algorithm.

In the Check Node Selection procedure, which immediately follows the Check Nodes Shortest Paths Discovery procedure, the candidate check nodes are classified and one is selected based on predefined selection criteria. These selection criteria are arranged in a sequence of decreasing priority. If more than one check node survives any selection criterion, the next criterion in the hierarchy is applied on the surviving check nodes only. If after the last (lowest priority) selection criterion multiple check nodes survive, one out of this last set of survivors is chosen randomly. It was noted that both the standard PEG algorithm and the IPEG construction algorithm enforce the selection of the minimum degree (check node degree) survivors early. This has the advantage of creating code graphs with concentrated check node distributions which, they proffered, consequently optimizes the waterfall performance. In order to increase the values of the ACE spectrum components, a different sequence of check node selection criteria were proposed. Given the sets $C_{s_j}^l$, P_{s_j,c_i}^l, and the path ACE metrics for all

$l \leq l_{max}$, the sequence of selection criteria used in the Check Node Selection procedure of the generalized ACE constrained PEG algorithm (arranged in decreasing priority) is as follows:

i) Select check node from the set $c \in C_{s_j}^{l_{max}}$,

ii) Select the survivor(s) with the largest minimum path ACE metric,

iii) Select the survivor(s) with the smallest number of minimum ACE shortest paths,

iv) Select the survivor(s) with the smallest total number of shortest paths,

v) Select the minimum degree survivor(s), and

vi) Select one survivor randomly if multiple check-nodes survive after selection criterion 'v'.

Performance simulations for short-to-medium length rate-½ LDPC codes which were constructed using the standard PEG [Hu et al., 2001], the IPEG construction [Xiao and Banihashemi, 2004], and the generalized ACE constrained PEG algorithms were carried out. All the codes compared were constructed using the DE optimized symbol node degree distributions used in [Xiao and Banihashemi, 2004] and [Hu et al., 2005]. Out of 100 codes constructed using each of the three algorithms, the best 10 codes from each ensemble were isolated based on the values of their ACE spectra. Subsequently, the best BER performance by a code constructed using each of the three PEG algorithms for block lengths of $n = 504$ and $n = 1008$ were presented. The BER values were estimated in Monte Carlo simulations using: BPSK transmission over the AWGN channel, the belief propagation (BP) iterative decoder, and a maximum number of decoding iterations of 100. In the results presented, a lower error-floor is observed for the generalized ACE constrained PEG algorithm codes compared to the IPEG construction algorithm codes. The IPEG algorithm was shown to have a better error-floor performance than the standard PEG algorithm code. In other words, the results presented showed that as the ACE spectrum properties increased from the standard PEG algorithm to the IPEG construction algorithm, and peaked with the generalized ACE constrained PEG algorithm, the error-floor performance of corresponding LDPC codes also improved.

The performance differences between the generalized ACE constrained PEG algorithm codes and the IPEG construction algorithm codes, as observed from the graphical plots, are not significant. Given

that only the best performing codes selected from ensembles of only 100 codes were reported, it is not improbable that choosing the best codes from a larger ensemble of codes constructed using these two algorithms would blur the reported performance differences.

3.3.6 A Decoder-Optimized PEG Algorithm Modification for Irregular LDPC Codes

Healy and de Lamare proposed a PEG algorithm modification to enhance the construction of short-to-medium block length irregular LDPC codes [Healy and de Lamare, 2012]. The modification is a supplement to existing PEG code construction constraints (or criteria), and is used to discriminate and select a check node from a multiple choice of candidate check nodes when establishing new edges in the Tanner graph under construction. The supplementary criterion is meant to enhance the choice of a check-node using the sum-product algorithm (SPA) in a decoder-based optimization of the final choice of a check-node. In the proposed *decoder optimised* (DO) PEG code construction algorithm, under the current graph settings, the SPA decoding performances of the code when temporary edges are established between the current symbol node s_j and each of the candidate check-nodes are compared and the check-node which results in the best performance is selected. The combination of the SPA decoding algorithm and the PEG algorithm in the DOPEG algorithm significantly increased the complexity of the code construction. Decoder optimization (DO) was applied to the standard PEG algorithm by [Hu et al., 2001] and the improved PEG (IPEG) construction algorithm by [Xiao and Banihashemi, 2004], and the resultant algorithms were called the DOPEG and DOIPEG algorithms respectively.

In the following discussion we adopt the definitions and notations in [Hu et al., 2001] and [Xiao and Banihashemi, 2004]. It was observed that the standard PEG algorithm is very frequently faced with a set of equivalent lowest degree candidate check-nodes in the set $\dot{\mathrm{N}}_{s_j}^{l}$ from which it must make a choice of a single check node. Similarly, for the IPEG algorithm, after the ACE metrics of the check-nodes from the set of lowest degree candidate check-nodes in $\dot{\mathrm{N}}_{s_j}^{l}$, i.e. from the set $\ominus_{s_j}^{k}$, are evaluated, it often happens that a further subset (more than one) of these check nodes would have the same best ACE metric from which the final choice of a single check node must be made. Whenever

these situations are encountered in both the PEG and IPEG algorithms, the final check node selection is made randomly from amongst the sets of candidates check node that have equivalent properties. The decoder based optimization implemented in the DOPEG and DOIPEG algorithms were designed to find the candidate from each of these sets which result in the best decoding performance. It was argued that the candidate check node which produces an intermediate code with the best decoding performance is the choice with results in the best graph connectivity, and that the final Tanner graph constructed using DO operations will have improved structure.

Simulations were carried out on rate-½ irregular codes that were constructed using a modified optimized degree distribution of: $\lambda(x) = 0.30013x^2 + 0.28395x^3 + 0.41592x^8$. BPSK modulation over the AWGN channel, and log-domain SPA decoding algorithm were used. The maximum number of decoding iterations was 50, and at least 100 block errors were gathered per point in the graph. The decoding performance of codes of length $n = 250$ which were constructed using PEG, DOPEG, IPEG and DOIPEG algorithms were compared. The decoder optimized (DO) stage in the DOPEG and DOIPEG were operated with identical AWGN vectors applied to the all-zero codeword at each SNR in the specified range. In the results presented, the DOPEG code was shown to have a better performance than the PEG constructed code. The IPEG algorithm codes performed better than both the PEG and DOPEG codes, and the best performance in the error-floor region was achieved by the DOIPEG code. The difference in performance between IPEG and DOIPEG was however not significant over the plotted range of $3.0dB$ to $4.5dB$. It was observed that the DOPEG and DOIPEG algorithms had very long construction times compared to the PEG and IPEG algorithms respectively. The construction times for DOIPEG codes were considerably less than those for DOPEG codes of equivalent length, this is consequent to the fact that the decoder optimization stage of DOIPEG operates on smaller sets of candidate check-nodes.

From the results presented for code lengths over a range of 250 to 2000, we observe that while the difference in code performance between the PEG and the DOPEG appear significant, the improved performance of DOIPEG over IPEG appears to be marginal. It can be observed from the results and conclusions that beyond a block length of approximately 2000, the performance improvements of

decoder optimised codes over the corresponding base codes will reduce in line with the concentration theorem. There is no indication of the girth and minimum distance properties of the Tanner graphs compared in the work. In order to ensure that there are no cycles composed of only degree-2 symbol nodes, as discussed in [Richardson et al., 2001], the chosen DE optimized symbol node degree distribution was modified so that the number of weight-2 symbol nodes was smaller than the number of check nodes, and hence the degree distribution used was suboptimal.

3.3.7 An Elementary Trapping Sets (ETS-) Constrained PEG Algorithm

Khazraie et al. modified the PEG algorithm in an attempt to improve the error-floor performance of regular and irregular LDPC codes by avoiding the creation of dominant elementary trapping sets (ETS's) in code Tanner graphs. This proposed algorithm modification is called the *elementary trapping set's (ETS-) constrained* PEG [Khazraie et al., 2012].

We adopt the definitions and notations used by Khazraie et al. in the following review. Let $\mathcal{S}$ denote a trapping set in a code with Tanner graph G. A trapping set $\mathcal{S}$ is described as an '(a, b) trapping set' if $\mathcal{S}$ has a size $|\mathcal{S}|$ of a, and the induced subgraph $G(\mathcal{S})$ of $\mathcal{S}$ in the Tanner graph of the code contains b unsatisfied check nodes. An *elementary* trapping set is an (a, b) trapping set for which every check node in $G(\mathcal{S})$ has a degree of either 1or 2. It is generally understood that elementary trapping sets are the most harmful, and that among any set of ETS's, those with the smaller values of a and b are more harmful. *Dominant* trapping sets refer to elementary trapping sets with smaller values of a and b. In addition the *depth-d ETS spectrum* of a Tanner graph G is defined as the d-tuple $(\xi_1, \ldots, \xi_d)$, where ξ_a, for $1 \leq a \leq d$, is the minimum value of b for all (a, b) elementary trapping sets ETS's in G.

The aim of the PEG algorithm modification was to prevent the creation of dominant elementary trapping sets in LDPC codes and achieve ETS spectrums with components as large as possible. The spectrum of elementary trapping sets was considered to be a direct measure of error-floor performance, and a search algorithm to find the dominant elementary trapping sets (ETS's) in an under-construction Tanner graph was incorporated into the PEG algorithm. In order to avoid creating dominant ETS's, at each step of the PEG algorithm, check nodes which do not result in the creation of

dominant ETS's are identified and chosen from the set of candidate check nodes which are available for connecting an edge of symbol node s_j.

Simulation results for regular and irregular LDPC codes constructed using the ETS-constrained PEG algorithm were presented in the work. Simulations were carried out for BPSK modulation over the AWGN channel, log-likelihood ratio domain BP/SPA decoding was used, and the maximum number of decoding iterations was 50. Simulation results plotted for each point on the error rate curves in the literature were recorded after 100 block (frame) errors were gathered.

The results for an irregular rate-½ $(n, m) = (1008, 504)$ ETS-constrained PEG code was presented. This code was constructed with optimum parameter input to the algorithm and had DE optimized degree distribution $\lambda_1(x) = 0.25105x^2 + 0.30938x^3 + 0.00104x^4 + 0.43853x^9$. The performance of this ETS-constrained PEG code was compared to that of two other codes with the same rate, block length and symbol node degree distribution which were constructed using the standard PEG algorithm [Hu et al., 2001] and the generalized ACE constrained PEG algorithm [Vukobratović and Šenk, 2008] respectively. It was reported that all three codes had a global girth of 6, however there was a significant variation in their ETS distribution. The generalized ACE constrained PEG algorithm code had a better ETS distribution (a lower number of the smaller ETS's with small values of b) than the standard PEG algorithm code, and the ETS-constrained PEG algorithm code had an even better ETS distribution (a smaller number of dominant (a, b) ETS's) than in the generalized ACE constrained PEG algorithm code. In the simulation results plotted for the three codes, the ETS-constrained PEG code was shown to perform better than the two other codes, particularly in the error-floor region. Additionally, the generalized ACE constrained PEG algorithm code was shown to perform better than the standard PEG code in the error-floor region. The performances of all three codes were shown to be nearly identical in the water-fall region.

Similarly, the results for a regular rate-½ $(n, m) = (504, 252)$ ETS-constrained PEG code which was constructed with optimum parameter input to the algorithm and a uniform symbol-node degree 3 $(d_s = 3)$ was presented. Similar to the irregular code, the performance of this code was compared to

that of two other codes with the same rate and block length which were constructed by the standard PEG algorithm and the generalized ACE constrained PEG algorithm. However, it should be noted that the constraints in the generalized ACE constrained PEG algorithm is only effective if used to construct irregular codes. In this case, as reported, although all three codes had a global girth of 8, there was a significant variation in their ETS distribution. In a trend which is similar to that for the irregular codes, the ETS-constrained PEG code had a superior ETS distribution than both the standard PEG code and the generalized ACE constrained PEG code. In the simulation results plotted for these three codes the ETS-constrained PEG code was shown to perform better than the other two LDPC codes, particularly in the error-floor region. The generalized ACE constrained PEG code was also shown to perform better than the standard PEG code in the error-floor region. The performances of these three regular codes in the waterfall region were even more similar than was the case for the irregular codes.

Finally, similar results as obtained for the two cases above were observed for two rate-½ irregular ETS-constrained PEG codes with (n, k) of $(1008, 504)$ and $(2016, 1008)$ when their ETS distributions and simulation performances were compared with those of codes constructed using the genaralized ACE constrained PEG algorithm and the standard PEG algorithms. Both of the ETS-constrained PEG algorithm codes were constructed using a DE optimized symbol-node degree distribution given by $\lambda_2(x) = 0.23802x^2 + 0.20997x^3 + 0.03492x^4 + 0.12015x^5 + 0.01587x^7 + 0.00480x^{13} + 0.37627x^{14}$. The improved error-floor performances observed for both irregular and regular codes constructed using the ETS-constrained PEG algorithm were attributed to their superior ETS distributions.

At all stages of code construction, the search algorithm used for finding dominant trapping sets is restricted to searching only for dominant ETS's which contain the symbol-node that is being connected into the Tanner graph at that stage of execution of the PEG algorithm. This restriction potentially allows trapping sets which do not include this particular symbol-node but have resulted due to the changing graph interconnectivity to go undetected. The ETS-constrained PEG algorithm does not directly attempt to improve the girth and minimum distance properties of the codes constructed.

The girth and minimum distance properties of ETS-constrained PEG codes are incidental to the symbol node degree distributions used and the greedy subgraph expansion of the PEG algorithm.

3.3.8 An Error Minimizing PEG Algorithm

Sharon and Litsyn proposed an approach to constructing Tanner graphs which yields LDPC codes with minimized block error probability over binary erasure channels (BEC) [Sharon and Litsyn, 2008]. The method was extended to easy implementations of LDPC codes based on lifted graphs. The modification was called the *error minimization PEG* algorithm (EMPEG). The proposed algorithm is based on the greedy PEG algorithm. Combinatorial structures in Tanner graphs which were named *minimal cycle sets* were used to determine the expected block error probability of codes transmitted over the BEC. In the EMPEG algorithm, the edges in a graph are selected such that their contribution to the expected block error probability is minimized. Monte Carlo simulations were carried out on two irregular LDPC codes based on lifted graphs with *lifting factor* $Z = 24$ over the BEC and AWGN channels. Belief propagation (BP) decoding with a maximum of 50 decoding iterations was used, and at least 50 block errors were counted for each simulated point presented in the graphical plots. The two codes simulated were selected from the ensemble which have symbol node and check node degree distributions, $\lambda(x) = 264x^2 + 192x^3 + 120x^6$ and $\Phi(x) = 192x^6 + 96x^7$, respectively.

These two codes were:-

i) An $[n = 576, m = 288, Z = 24]$ optimized code used in the IEEE802.16e standard, and

ii) An $[n = 576, m = 288, Z = 24]$ code generated using the proposed EMPEG algorithm for lifted graphs.

The performance of the EMPEG code was shown to be better than that of the IEEE802.16e code, particularly in the error-floor region for both BEC and AWGN channels. The simulation results suggest that the EMPEG algorithm can generate high performance codes based on lifted graphs.

Results presented for code performance simulations over the BEC showed that an EMPEG code had better error-floor performance than the standard PEG code despite the fact that the EMPEG graph contained many 4-cycles but the PEG code did not contain any. Based on these observations it was

asserted that the common belief that 4-cycles are harmful and should be removed from the Tanner graph of the code is not always true.

3.4 Summary

- The challenges to lowering the error-floor of short-to-medium length LDPC codes under iterative message-passing decoding have been discussed. These include the following properties of LDPC codes: girth, symbol node degree distributions, minimum distance weights d_{min}, minimum stopping set weights s_{min}, the ACE of cycles in Tanner graphs, and trapping sets.
- Some of the most noteworthy PEG algorithm modifications in recent literature which were designed to lower the error-floor of the short-to-medium length LDPC codes constructed have been examined. The modified PEG algorithms discussed have been briefly explained, the reported LDPC code performance results obtained have been analysed, and the drawbacks of some of these PEG algorithm modifications have been highlighted.
- In the opinion of the author, of all the notable modified PEG algorithms reviewed in this chapter, the most promising PEG algorithm modifications for improving the error-floor performance of short-to-medium length irregular LDPC codes over the AWGN channel are the 'improved PEG (IPEG) construction' [Xiao and Banihashemi, 2004], and the 'generalized ACE constrained PEG (G-ACE PEG) algorithm' [Vukobratović and Šenk, 2008]. Both of these algorithms are designed to maximize the ACE of cycles in the Tanner graph of the codes constructed. A comparison of the error-floor performance of rate-½ (512, 256) and (1024, 512) irregular LDPC codes constructed using the IPEG and G-ACE PEG algorithms (presented later in Chapter 6) to those published for similar LDPC codes constructed using the other modified PEG algorithms shows that the IPEG construction and the G-ACE PEG algorithm result in LDPC codes with the lowest error-floor.

Chapter 4

Determining the girth and ACE of cycles in LDPC code Tanner graphs

This chapter presents an efficient algorithm for determining the local girth and ACE distributions in short-to-medium length LDPC code Tanner graphs. The performance of the proposed algorithm was verified by using it to determine the local girth and ACE distributions in short-to-medium length LDPC code Tanner graphs with different global girth, local girth distribution and ACE properties which are constructed using different PEG algorithms and a wide range of symbol node degree distributions. The algorithm for determining the local girth and ACE distributions in short-to-medium length LDPC code Tanner graphs was crucial to obtaining the research results published in [Abdu-Aguye et al., 2016(b)] and [Abdu-Aguye et al., 2016(c)]. Parts of this chapter appear in the following conference proceeding: Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, December). An efficient algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs. In *Signal Processing and Communication Systems (ICSPCS), 2016 10th International Conference on* (pp. 1-7). IEEE.

4.1 Introduction

The presence of cycles in the Tanner graphs of low-density parity-check (LDPC) codes have a significant impact on their performance under iterative message-passing decoding algorithms such as the belief propagation or sum-product algorithm (BP/SPA). It is generally understood that BP/SPA

decoders work better if the girth of an LDPC code is more than 4, i.e. in the absence of 4-cycles. Even as Gallager invented LDPC codes, the LDPC code construction techniques he proposed involved procedures to avoid 4-cycles [Gallager, 1962]. Discussions on the harmful effects of 4-cycles on the performance of LDPC codes abound in literature; some early literature on this include [MacKay and Neal, 1996] and [Wiberg, 1996], and some of the more recent literature on this include [Campello et al., 2001], [Johnson and Weller, 2001] [Fan, 2001] and [Kim et al., 2004]. In fact, there is evidence that the performance of an LDPC code depends not only on its global girth, i.e. the length of the shortest cycle in the graph, but also on the number of shortest cycles [Mao and Banihashemi, 2001], [Venkiah et al., 2008]. Therefore, the performance of a short-to-medium length LDPC code also depends on the *local girth distribution* in its Tanner graph. Under iterative message-passing decoding algorithms, the presence of short cycles cause the edges (and symbol nodes) involved in the cycles to become correlated after a few decoding iterations which significantly reduces decoding efficiency [Zhang and Schlegel, 2011]. Conversely, empirical evidence shows that cycle-free codes, or codes with very large girths, have poor decoding performance not only under iterative decoding schemes but also under maximum likelihood (ML) decoding [Etzion et al., 1999]. It is therefore clear that cycles are necessary in the bipartite (Tanner) graphs of capacity-approaching LDPC codes [Sason, 2009]. Accordingly, in order to identify the features of good practical-length LDPC codes, it is imperative to determine not only their global girths but also their local girth distributions.

The ACE of cycles in LDPC code graphs is another important parameter which significantly affects the error-floor performance of short-to-medium length LDPC codes. The ACE metric was introduced in [Tian et al., 2003], [Tian et al., 2004]. Xiao and Banihashemi modified the standard progressive edge-growth (PEG) algorithm [Hu et al., 2001] to construct LDPC codes with improved ACE [Xiao and Banihashemi, 2004]. Subsequently, algorithms which further improved the ACE in the LDPC codes constructed were proposed in [Vukobratović and Šenk, 2008] and [Vukobratović and Šenk, 2009]. Generally, improved ACE in LDPC codes results in lower error-floors without sacrificing the performance in the waterfall region.

4.2 Existing Algorithms for Determining Girth Properties and Counting Cycles in LDPC Code Tanner Graphs

Many algorithms have been proposed for analyzing LDPC code Tanner graphs in order to determine local girths g_{s_j}, global girths g, and enumerating cycles in bipartite graphs of girth g. Some of these algorithms can be found in [Halford and Chugg, 2004], [Lee et al., 2005], [Fan and Xiao, 2006] and [Karimi and Banihashemi, 2012(b)].

Some noteworthy algorithms for determining the girth properties or enumerating the cycles in LDPC code Tanner graphs which have been recently published include:

i) an algorithm for counting cycles of length g, $g + 2$, and $g + 4$ [Halford and Chugg, 2006],

ii) an efficient message-passing algorithm for counting short cycles of length $g, g + 2, \ldots, 2g - 2$ [Karimi and Banihashemi, 2013], and

iii) an improved message-passing algorithm for counting short cycles, which reduces the complexity and computing time of the algorithm proposed in ii) by a factor of two [Li et al., 2015].

These algorithms employ indirect approaches to finding the cycles in bipartite graphs and consequently have significant complexities.

4.3 Existing Algorithms for Determining the ACE Spectra in LDPC Code Tanner Graphs

Although numerous studies have utilized the ACE metric to improve the design of LDPC codes for better performance in the error-floor region, there are only a few algorithms dedicated to enumerating and quantifying the ACE parameters in LDPC code graphs. The most noteworthy of such algorithms was developed in [Vukobratović et al., 2007]. Unfortunately, similar to all other published literature on cycle connectivity, the simple notion of the ACE spectrum in an LDPC code graphs is expressed here using complex notation and conceptualizations.

4.4 An Efficient Algorithm for Determining the Local Girth and ACE Distributions in Short-to-Medium Length LDPC Code Tanner Graphs

We propose a simple and efficient algorithm for determining the local girth and ACE distributions in short-to-medium length LDPC code Tanner graphs. The proposed algorithm is based on a subgraph expansion technique which is a slight modification of the subgraph expansion used in the standard PEG algorithm [Hu et al., 2001], [Hu et al., 2005]. We refer to the modified subgraph expansion technique as the *single-edge tree-apex* (SETA) subgraph expansion. SETA subgraph expansions are used to find short cycles in code graphs. We propose that the ACE of cycles in a graph should be evaluated with respect to symbol nodes in the graph. Consequently, the ACE spectrum in a graph can be represented by a local ACE distribution, and the degree of connectivity of the entire network of cycles in a graph can be concisely represented by an average ACE metric.

4.4.1 The Single-Edge Tree-Apex (SETA) Subgraph Expansion Technique for Finding Short Cycles in LDPC Code Graphs

The subgraph expansion technique used in the standard PEG algorithm is described in Section 2.4 and illustrated in Figure 2.12. We propose a modification to the PEG algorithm subgraph expansion technique in order to find all the short cycles that pass through a symbol node in an LDPC code Tanner graph. The SETA subgraph expansion proceeds as follows. Firstly, all the edges attached to symbol node s_j, i.e. $E_{s_j}^k$, $0 \leq k \leq d_{s_j} - 1$, are identified. As an example, in Figure 2.12 the edges of s_j are (s_j, c_{i_1}), (s_j, c_{i_2}), . . . , $(s_j, c_{i_{d_{s_j}}})$. In a SETA subgraph expansion, subgraphs are expanded from only one of these edges at a time, this is illustrated in Figure 4.1. Let the first edge $E_{s_j}^k$ be (s_j, c_{i_1}), then at expansion depth 0, only vertices s_j and c_{i_1} would be involved in the subgraph expansion. At all other expansion depths, i.e. $l \geq 1$, the subgraph expansion proceeds in the usual way; at expansion depth $l = 1$, all the edges incident on vertex c_{i_1} excluding (s_j, c_{i_1}), i.e. (s_{a_1}, c_{i_1}), (s_{b_1}, c_{i_1}), . . . , (s_{z_1}, c_{i_1}), where $a_1, b_1, \ldots z_1 \neq j$, are identified, then all the other edges incident on vertices $s_{a_1}, s_{b_1}, \ldots s_{z_1}$ excluding (s_{a_1}, c_{i_1}), (s_{b_1}, c_{i_1}), . . . (s_{z_1}, c_{i_1}), are identified. This unravelling

procedure continues until another edge of s_j, i.e. (s_j, c_{i_2}), (s_j, c_{i_3}), . . . , or $(s_j, c_{i_{d_{s_j}}})$, is reached. At this stage, a short cycle through symbol node s_j, which passes through edge (s_j, c_{i_1}), has been found. The girth of this cycle is calculated using $g_{E_{s_j}^k} = (2l + 2)$, where l is the expansion depth that the check vertex to which the other vertex of symbol node s_j was found to be attached. Note that the other vertex of s_j is found at depth (l+1) just as the subgraph expansion proceeds to identify the symbol nodes involved at that level. To avoid an infinite loop in the algorithm in the unlikely event that $d_{s_j} > 1$ and unconnected edges of s_j exist in a Tanner graph, a maximum subgraph expansion depth l_{max} is set. During a SETA subgraph expansion from an edge of s_j, where $d_{s_j} > 2$, duplicate vertices of s_j may be found at different subgraph expansion depths.

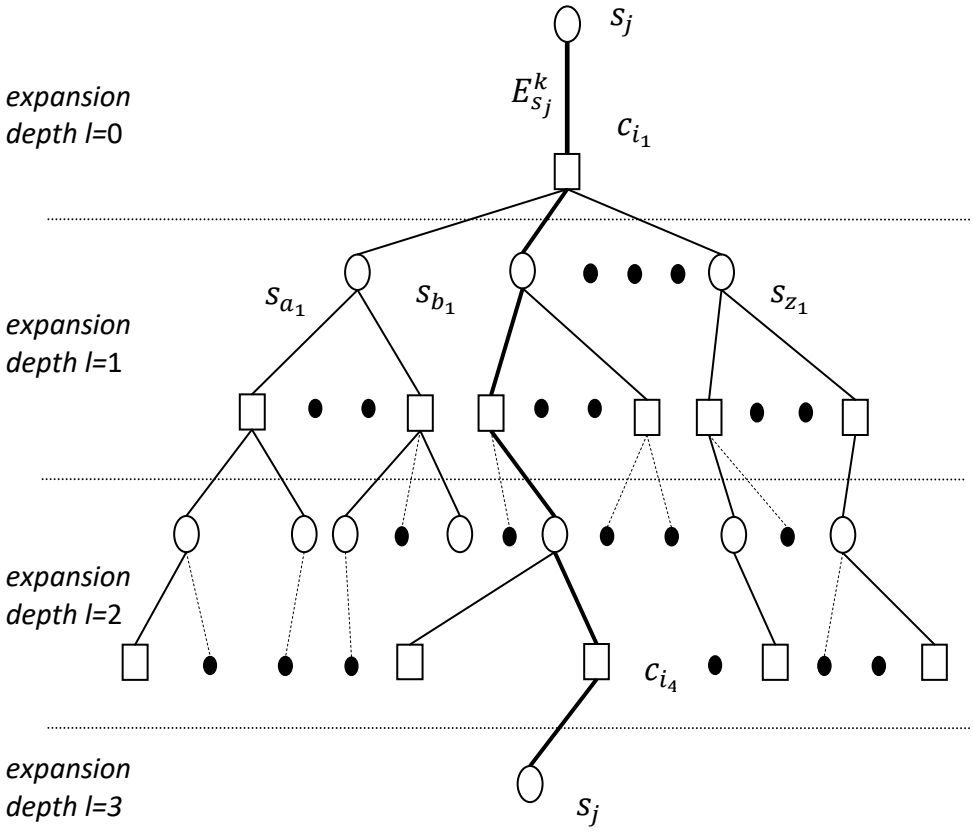

Figure 4.1: A SETA subgraph expansion to find a short cycle of symbol node s_j, the path from edge (s_j, c_{i_1}) to edge (c_{i_4}, s_j) is a cycle, and the girth of this cycle $g_{E_{s_j}^k} = 6$ because vertex c_{i_4} is at depth $l = 2$

In the SETA subgraph expansion technique, a subgraph is expanded from each of the edges of s_j, i.e. $E_{S_j}^k$, $\forall$ ($0 \leq k \leq d_{s_j}$- 1). The girth of the shortest cycle through $E_{S_j}^k$, a distinct edge of symbol node s_j, depends on the subgraph expansion depth l at which the check vertex of another edge of s_j is found in a SETA subgraph expansion from $E_{S_j}^k$. This is calculated as $g_{E_{s_j}^k} = (2l + 2)$. When the girth of all the short cycles through s_j have been determined, i.e. $\{g_{E_{s_j}^0}, \ldots, g_{E_{s_j}^{ds_j-1}}\}$, the shortest girth value in the set is chosen as the local girth at symbol node s_j, which is denoted as g_{s_j}. Therefore, it follows that $g_{s_j} = \min\{g_{E_{s_j}^0}, \ldots, g_{E_{s_j}^{ds_j-1}}\}$.

The ACE of a cycle is obtained by counting the number of edges through which the cycle is connected to the rest of the graph via its symbol nodes. This is calculated using ACE = $\sum_i(d_i - 2)$, where d_i is the degree of the i-th symbol node in the cycle and the summation is taken over all the symbol nodes in the cycle.

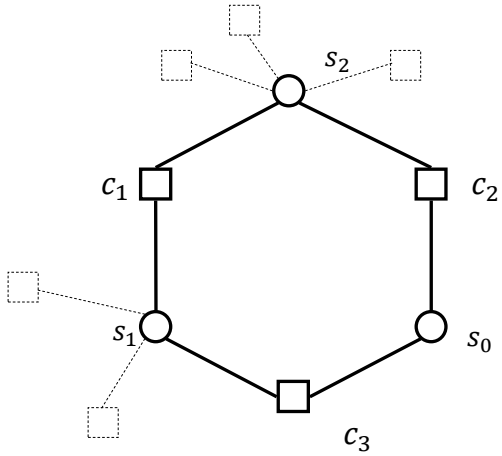

Figure 4.2: Evaluating the ACE of a cycle of girth $g = 6$

Figure 4.2 illustrates the evaluation of the ACE of a short cycle of girth $g = 6$. The degrees of the three symbol nodes in this cycle s_0, s_1, and s_2 are $d_{s_0} = 2$, $d_{s_1} = 4$, and $d_{s_2} = 5$, respectively. All the edges of these symbol nodes are shown in the figure; the edges involved in the cycle are shown using continuous line segments, and the edges connected to the rest of the graph are shown using dashed line

segments. The ACE of this cycle, represented by the number of dashed lines, is 5. Both edges of symbol node s_0 are involved in the cycle, hence it has no external check node(s) to contribute extrinsic information to the set of symbol nodes in the cycle. Alternatively using the equation, ACE = $\sum_i (d_i - 2)$, the ACE of this cycle is easily calculated as ((d_{s_0} - 2) + (d_{s_1} - 2) + (d_{s_2} - 2)) = 5.

Each and every cycle in a Tanner graph has a girth and an ACE metric. Therefore, we propose that parameters similar to those used for Tanner graph girth descriptions can be adopted for ACE descriptions. For example, the entire spectrum of cycle ACE values in a Tanner graph may be represented with respect to symbol nodes in the graph using a *local ACE distribution*. Accordingly, we introduce and define the following ACE parameters for LDPC code graphs:

i) the '*local ACE* at a symbol node s_j' or a '*symbol node ACE'*, denoted by ACE_{s_j}, which is the minimum ACE of the short cycles that pass through symbol node s_j in a graph,

ii) the '*modal symbol node ACE*' in a graph, which is the '*symbol node ACE'* value which appears most often in the graph,

iii) the '*average ACE*' of a graph, which is the sum of the n symbol node ACEs in the graph divided by n, where n is the number of symbol nodes in the graph, and

iv) the '*ensemble average ACE*' of an ensemble of codes, which is the sum of the *average ACE* of all the codes in the ensemble divided by the number of codes in the ensemble.

The average ACE of an LDPC code is a concise single-valued representation of the degree of connectivity of the entire network of cycles its Tanner graph.

4.4.2 Determining the ACE of Short Cycles using SETA Subgraph Expansions

In addition to determining the girth of the short cycles found in a Tanner graph, SETA subgraph expansions can be easily augmented to determine the ACE of cycles. The ACE of each cycle found during a SETA subgraph expansion can be evaluated as follows. Starting from the check vertex which makes an edge with symbol vertex s_j at the apex of a tree, i.e. at expansion depth 0, all the check

vertices encountered during a SETA subgraph expansion are assigned ACE weights which represents the ACE of the cycle that would be detected if another vertex of s_j is found (at the next expansion depth) to be attached to the check vertex. This is illustrated in Figure 4.3, where at expansion depth 3, another vertex of s_8 has been found to be attached to check vertex c_4 at depth 2. The detection of this edge occurred when the subgraph expansion proceeded to expansion depth 3. That is, in the process of identifying the symbol node vertices at depth 3 that are attached to the check vertices at depth 2, another vertex of s_8 was found. In this case, the ACE of the cycle found is equal to the ACE weight assigned to check vertex c_4.

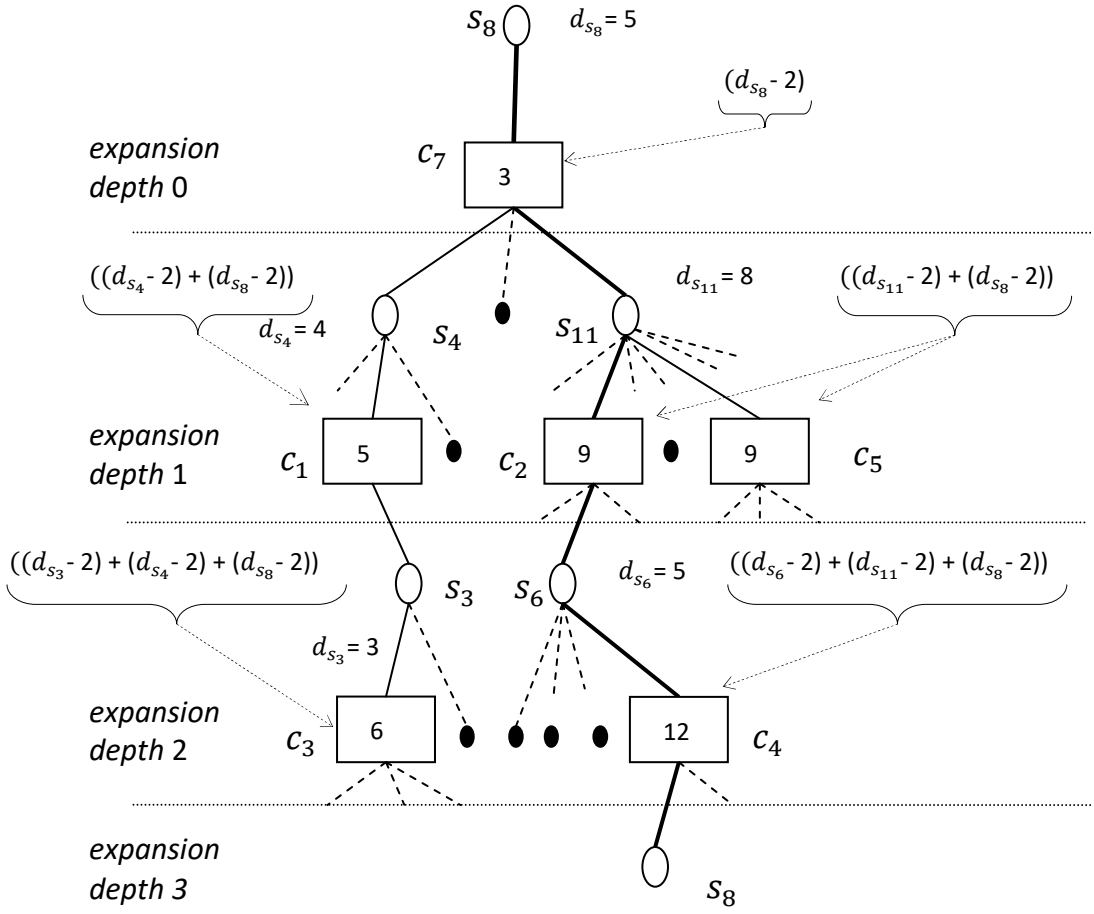

Figure 4.3: Check vertex ACE weight assignments during a SETA subgraph expansion to find a short cycle of symbol s_8 through edge (s_8, c_7)

ACE weight assignments to check vertices at each expansion depth during a SETA subgraph expansion are cumulative and take into cognizance all the symbol vertices preceding each check vertex in the subgraph expansion. The ACE of the short cycle through the (k+1)-th edge of symbol node s_j is denoted as $ACE_{E_{s_j}^k}$. The local ACE at symbol node s_j, denoted as ACE_{s_j}, is the minimum ACE of all the short cycles that pass through s_j. This is expressed mathematically, as $ACE_{s_j} = \min\{ ACE_{E_{s_j}^0}, ACE_{E_{s_j}^1}, \ldots, ACE_{E_{s_j}^{d_{s_j}-1}} \}$. Therefore, ACE_{s_j} can only be determined after a SETA subgraph expansion has been carried out from all the d_{s_j} edges of s_j in the graph.

Figure 4.3 shows check vertex ACE weight assignments made during a SETA subgraph expansion to determine the ACE of the shortest cycle through the (k+1)-th edge of symbol node s_8, i.e. $ACE_{E_{s_8}^k}$. The (k+1)-th edge illustrated in Figure 4.3 is edge (s_8, c_7). At subgraph expansion depth 3, another vertex of symbol node s_8 was found to be attached to check vertex c_4 at expansion depth 2. Substituting this expansion depth $l = 2$ in the equation $g_{E_{s_j}^k} = (2l + 2)$, the girth of this cycle is $g_{E_{s_8}^k} = \big((2 \times 2) + 2\big) = 6$. Therefore, the girth of the shortest cycle of symbol node s_8 which passes through edge (s_8, c_7) is 6, and the ACE of cycle s_8-c_7-s_{11}-c_2-s_6-c_4-s_8 is 12, which is the ACE weight assigned to c_4 during the SETA subgraph expansion from edge (s_8, c_7).

During SETA subgraph expansions, duplicate check vertices frequently occur at the same expansion depth. Consequently, ACE weight assignments made to each duplicate must be uniquely tracked.

4.4.3 The SETA Subgraph Expansion Based Algorithm for Determining the Girth and ACE Distributions in LDPC Code Tanner Graphs

In terms of computational load, the proposed algorithm for determining the local girth and ACE distributions in LDPC code Tanner graphs, which is based on SETA subgraph expansions, is very similar to the standard PEG LDPC code construction algorithm. Consequently, based on complexity estimations for the standard PEG algorithm in [Hu et al., 2005], the worst case computational complexity of the proposed algorithm is $O(nm)$. Therefore, the complexity of the proposed algorithm is considerably less than the complexity of the algorithms in [Halford and Chugg, 2006],

[Karimi and Banihashemi, 2013], and [Li et al., 2015]. The proposed algorithm for determining the local girth and ACE distributions in LDPC code Tanner graphs is summarized in Algorithm 4.1.

Algorithm 4.1: The algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs

for $j = 0$ to $n - 1$ **do**
begin
for $k = 0$ to d_{s_j} - 1 **do**
begin
if $d_{s_j} > 1$

- ❖ execute a SETA subgraph expansion from the $(k + 1)$-th edge of s_j, $E_{s_j}^k$, up to depth $(l + 1)$ where another vertex of s_j is found attached to a check vertex at depth l (i.e. another edge of s_j is found), or up to the maximum subgraph expansion depth l_{max}.
  if (another vertex of s_j is found)
  calculate the girth of the short cycle of s_j through edge $E_{s_j}^k$ using $g_{E_{s_j}^k} = (2l + 2)$
  else
  assign an arbitrarily large value for the girth (i.e. an infinite girth), and an ACE of 0; this indicates that no short cycle of s_j through edge $E_{s_j}^k$ has been found.
  e.g. $g_{E_{s_j}^k} = 1000$, and $ACE_{E_{s_j}^k} = 0$. (Either l_{max} is too small or the edges of s_j have no connecting path.)
- ➢ starting from the single check vertex at depth 0, use $ACE = \sum_i (d_i - 2)$ and the symbol node degrees of all the symbol vertices in the path leading to each check vertex to assign an ACE weight to every check vertex encountered during the SETA subgraph expansion.
  if (another vertex of s_j is found)
  $ACE_{E_{s_j}^k}$, the ACE of the short cycle of s_j through edge $E_{s_j}^k$, is the ACE weight assigned to the check vertex at depth l which makes an edge with symbol vertex s_j found at depth $(l + 1)$.

(Note that ❖ and ➢ are executed concurrently. That is, for speed and efficiency, the girth and ACE of the first short cycle found in each SETA subgraph expansion are determined concurrently.)

else
$d_{s_j} = 1$, there are no local cycles through s_j because it has only one edge $E_{s_j}^0$. Assign an arbitrarily large value for the girth (i.e. an infinite girth), and an ACE of 0.
e.g. $g_{E_{s_j}^0} = 1000$, and $ACE_{E_{s_j}^0} = 0$.
end
determine $g_{s_j} = \min\{g_{E_{s_j}^0}, \ldots, g_{E_{s_j}^{d_{s_j}-1}}\}$
determine $ACE_{s_j} = \min\{ ACE_{E_{s_j}^0}, ACE_{E_{s_j}^1}, \ldots, ACE_{E_{s_j}^{d_{s_j}-1}}\}$
end

determine the local girth and local ACE distributions in the Tanner graph from the set of local girth and local ACE values, $\{g_{s_j}\}$ and $\{ACE_{s_j}\}$, respectively, where $0 \leq j \leq n$-1.

In our implementation, l_{max} was set at 6 because most practical capacity-approaching short-to-medium length LDPC codes do not contain symbol nodes with local girths of up to 14; however, l_{max} can be easily adjusted for different types of codes. The algorithm was augmented to provide other statistics of girth and ACE distributions, including the average girth ($(\sum_{j=0}^{n-1} g_{s_j}) / n$), and the average ACE ($(\sum_{j=0}^{n-1} ACE_{s_j}) / n$) of graphs. A two dimensional array was used to assign ACE weights to all the check vertices encountered during each SETA graph expansion. One dimension of the array caters for the number of check nodes in the graph *m*, and the other dimension caters for ACE weights assignments for duplicate check vertices which are frequently encountered in subgraph expansions. Duplicate check vertices are reached through different paths in a graph expansion and are very likely to have different ACE weight assignments. All lower level (higher depth) check vertices attached through duplicated check vertices must inherit the cumulative ACE weight corresponding to its unique path in the subgraph expansion.

The algorithm for determining the local girth and ACE distributions in LDPC code Tanner graphs can be made significantly faster in Tanner graphs which have symbol nodes with degree 2 by using a single SETA subgraph expansion to determine the girth and ACE of the shortest cycle through them. This is because SETA subgraph expansions start and end at two different vertices of the same symbol node and, for degree 2 symbol nodes, SETA expansions from either edge to the other results in identical shortest cycle girth and ACE values. The C programming language code for the SETA subgraph expansion based algorithm for determining the local girth and ACE distributions in LDPC code Tanner graphs is available in the Appendices (See Appendix C).

4.4.4 Evaluating the Performance of the Algorithm for Determining the Girth and ACE Distributions in LDPC Code Tanner Graphs

The SETA subgraph expansion based algorithm for determining the girth and ACE distributions in LDPC code graphs was implemented using the C programming language running in the UBUNTU® Linux operating system on a 3.4GHz CPU machine with 16 GB RAM. In Table 4.1, the CPU running times required for determining the girth and ACE distributions in short length LDPC codes using the

proposed algorithm were compared to the CPU running times of the MPA and IMPA in Table II of [Li et al., 2015]. The MPA and IMPA were implemented using the MATLAB 2013a software in a 3.4GHz CPU machine with 8 GB RAM. Code I is an irregular LDPC code with maximum symbol node degree of 15 and a maximum check node degree of 9. Code II is a (3, 8)-regular LDPC code. Code I and Code II were obtained from [MacKay, 2002], where they are called *PEGirReg504x1008* and *PEGReg504x1008* respectively, and were also analyzed in [Li et al., 2015]. CPU running time evaluations were carried out using two versions of the SETA based algorithm: the *SETA (girth only)* version which determines local girth distributions only, and the *SETA (girth + ACE)* version which determines local girth and ACE distributions concurrently.

Table 4.1: CPU running times using: SETA (girth only), SETA (girth + ACE), and the MPA and IMPA

CPU time in seconds (s)	Code I	Code II
SETA (girth only)	17.86	104.02
SETA (girth + ACE)	18.10	106.47
MPA	471.97	139.14
IMPA	217.34	70.17

As can be seen from Table 4.1, for Code I both of the SETA subgraph expansion based algorithms took significantly shorter time to determine the girth and ACE distributions than it took the MPA and IMPA algorithms to count the short cycles in the code. However, for Code II, while the SETA based algorithms took a shorter time than the MPA algorithm, the IMPA algorithm required a shorter time than both SETA based algorithms. SETA based algorithms are significantly faster than the MPA and IMPA when applied to short length irregular LDPC codes. However, the IMPA is slightly faster than SETA based algorithms when applied to short length regular LDPC codes.

The SETA subgraph expansion based algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs was applied to three groups of short-to-medium length rate-½ LDPC codes. These groups of codes are refered to as Type I, Type II, and Type III codes. The performance of the algorithm for each code type were evaluated using three code dimensions; $(n, k) = (512, 256)$,

(1024, 512), and (2048, 1024), respectively. Recall: in a dimension (n, k) LDPC code, n is the code length and k is the number of information bits.

Type I codes were constructed using DE optimized symbol node degree distributions in the standard PEG algorithm. The (512, 256) Type I code was constructed with degree distribution:

$$Q_1(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14},$$

and the (1024, 512) and (2048, 1024) Type I codes were constructed with degree distribution:

$$Q_2(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}.$$

To obtain Type II codes, the maximum symbol node degree $d_{s(max)}$ in the degree distributions of Type I codes of corresponding dimensions were increased, and the resultant degree distributions were used to construct codes with graphs of identical girth but significantly higher edge densities than the Type I codes. During construction, the girth of Type II codes were maintained identical to corresponding Type I code girths by discarding all edge connections which would have resulted in reduced girth, Type II codes were constructed using the MLG (ES) PEG algorithm (see Section 6.2). The (512, 256), (1024, 512), and (2048, 1024) Type II codes have the following degree distributions**:**

$Q_3(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.00195x^{13} + 0.015625x^{14} + 0.01172x^{15} + 0.07422x^{16}$,

$Q_4(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.01172x^{19} + 0.01074x^{20} + 0.08105x^{21}$, and

$Q_5(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.00049x^{24} + 0.00244x^{25} + 0.00195x^{26} + 0.09863x^{27}$, respectively.

Type III codes were constructed to be as symbol node regular as possible while maintaining global girths $g = 8$. The (512, 256) Type III code was constructed with regular degree sequence $d_S = 3$, using the standard PEG algorithm; the (1024, 512) Type III code was constructed with regular degree

sequence $d_S = 4$, using the standard PEG algorithm; and the $(2048, 1024)$ Type III code was constructed using regular degree sequence $d_S = 5$ as input to the MLG (ES) PEG algorithm which discards some edge connections during the code construction in order to enforce a global girth $g = 8$. The Type III code constructed had degree distribution: $Q_6(x) = 0.01318x^4 + 0.98682x^5$. The MLG (ES) PEG algorithm is described in Section 6.2.

Table 4.2: CPU running times for the three rate-½ LDPC code types and dimensions

Code dimension (n, k)	Code type	Global girth (g)	Number of edges	CPU time (s) for girth and ACE evaluation
$(512, 256)$	Type I	6	2041	4.05
	Type II	6	2122	5.01
	Type III	8	1536	3.78
$(1024, 512)$	Type I	6	4115	18.95
	Type II	6	4716	41.48
	Type III	8	4096	108.00
$(2048, 1024)$	Type I	6	8230	199.10
	Type II	6	10757	416.39
	Type III	8	10213	3156.30

Table 4.2 shows the global girths, number of edges, and the corresponding CPU running times in seconds required by the SETA based algorithm to determine the girth and ACE distributions in the three rate-½ LDPC code types and dimensions. Figure 4.4 shows a plot of CPU running times against code length n for Type I, II, and III codes, as presented in Table 4.2. For the three code lengths analyzed, the proposed algorithm required longer CPU running times to analyze Type II codes compared to corresponding Type I codes. Despite having identical girths, Type II codes have higher edge densities than corresponding Type I codes. Consequently, the algorithm requires not only a larger number of SETA subgraph expansions but also more memory per subgraph expansion to analyze Type II codes compared to Type I codes.

Generally, Type III codes require longer CPU running time than corresponding Type I and Type II codes; the (512, 256) Type III code is an exception because it has a significantly lower edge density than the other two (512, 256) codes.

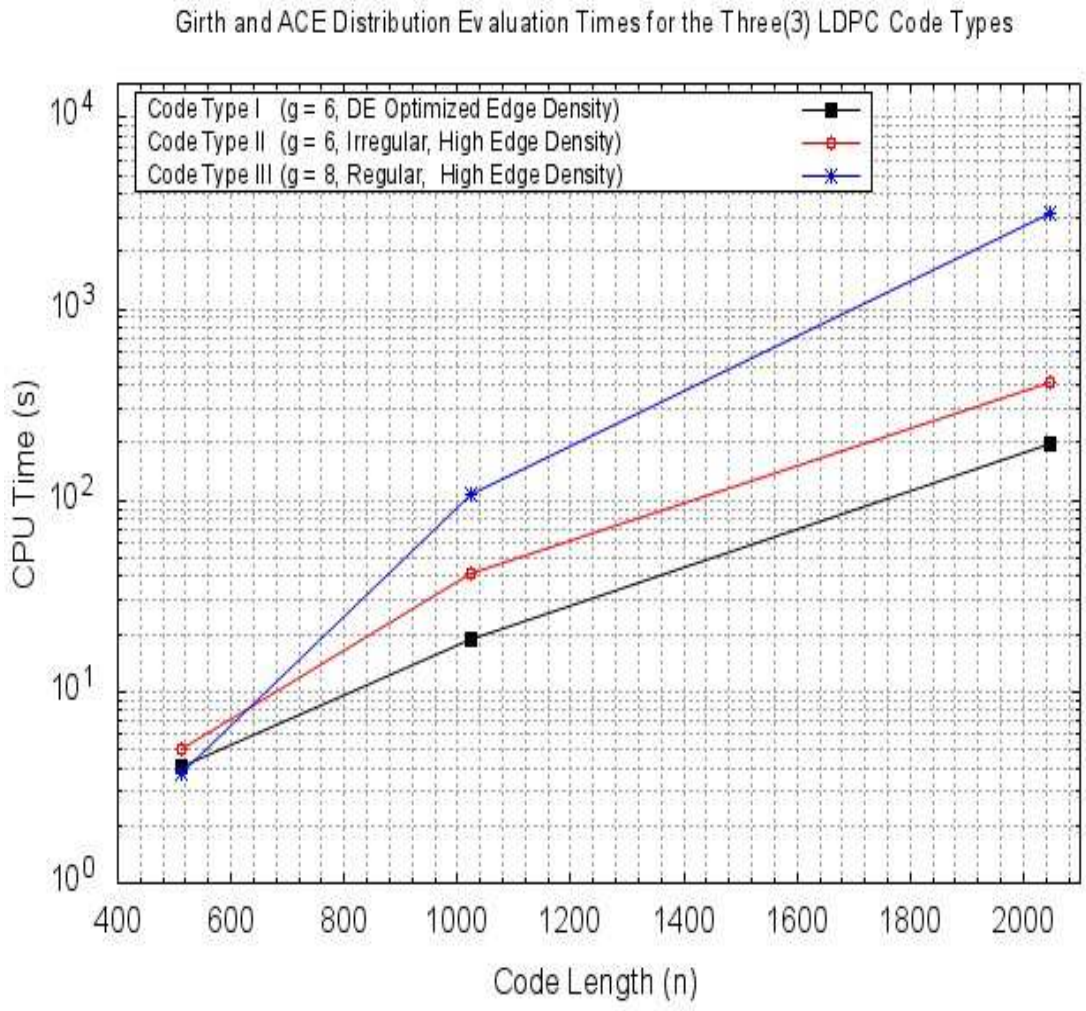

Figure 4.4: CPU running time in seconds (s) for determining the girth and ACE distributions in Type I, Type II, and Type III codes

The (1024, 512) Type III code has a lower density than the corresponding Type I and Type II codes, and the (2048, 1024) Type III code has a lower density than the corresponding Type II code. Despite having relatively lower densities, these two Type III codes took significantly longer time for their local girth and ACE distribution determination than corresponding Type I and Type II codes. This is

because Type III codes have global girths $g = 8$ (In fact, 100% of their local girths are equal to 8), and each SETA subgraph expansion proceeds to lower depths and has even more memory requirement than for Type II codes. This is because during subgraph expansions there is an exponential increase in the number of elements to be stored in memory (symbol node and check node vertices) as the graph expansion depth increases. Table 4.2 and Figure 4.4 highlight the effects of code lengths, densities, and girths on the CPU running time of the algorithm.

Table 4.3, Table 4.4, and Table 4.5 provide detailed results obtained when the proposed algorithm was used to determine the girth and ACE distributions in the Type I, Type II, and Type III code graphs of dimension $(512, 256)$, $(1024, 512)$, and $(2048, 1024)$, respectively. From the results in these tables, it can be seen that the irregular Type I and Type II codes have superior ACE parameters than the regular (or near regular) Type III codes. In addition, as a result of higher symbol node degrees in their degree distributions, Type II codes have higher ACE parameters than corresponding Type I codes.

Table 4.3: Girth and ACE distributions in Type I, Type II, and Type III rate-½ $(512, 256)$ LDPC code graphs

Code dimension (n, k)	Code type	Girth distribution		Average local girth	ACE distribution		Average ACE
		Local girth (g_{s_j})	Percent of symbol nodes		ACE_{s_j}	Percent of symbol nodes	
$(512, 256)$	Type I	6	100.000	6.000	12 13 14 15 16 24	5.273 59.961 25.195 7.812 0.391 1.367	13.518
	Type II	6	100.000	6.000	12 13 14 15 16 17 25 26	3.516 14.648 27.930 41.016 10.938 1.562 0.195 0.195	14.504
	Type III	8	100.000	8.000	4	100.000	4.000

Table 4.4: Girth and ACE distributions in Type I, Type II, and Type III rate-½ $(1024, 512)$ LDPC code graphs

Code dimension (n, k)	Code type	Girth distribution		Average local girth	ACE distribution		Average ACE
		Local girth (g_{s_j})	Percent of symbol Nodes		ACE_{s_j}	Percent of symbol nodes	
$(1024, 512)$	Type I	6	99.512	6.010	7 13 14 15 16 17 18 26 27	0.098 9.082 33.496 13.281 21.484 5.273 0.195 16.113 0.977	16.691
		8	0.488				
	Type II	6	100.000	6.000	17 18 19 20 21 22 23 25 35 36 37	1.953 9.180 28.906 37.500 12.891 7.227 0.098 0.098 0.098 1.465 0.586	20.099
	Type III	8	100.000	8.000	8	100.000	8.000

Table 4.5: Girth and ACE distributions in Type I, Type II, and Type III rate-½ $(2048, 1024)$ LDPC code graphs

Code dimension (n, k)	Code type	Girth distribution		Average local girth	ACE distribution		Average ACE
		Local girth (g_{s_j})	Percent of symbol nodes		ACE_{s_j}	Percent of symbol nodes	
$(2048, 1024)$	Type I	6	90.137	6.197	1	0.040	18.557
					3	0.049	
					4	0.195	
					5	0.391	
					6	0.146	
					7	0.244	
					8	0.049	
					9	0.098	
		8	9.863		11	0.049	
					13	3.516	
					14	23.828	
					15	12.109	
					16	16.699	
					17	9.521	
					18	1.025	
					19	0.244	
					26	24.121	
					27	7.031	
					28	0.146	
					29	0.488	
	Type II	6	100.000	6.000	22	0.146	27.507
					23	0.781	
					24	2.051	
					25	15.771	
					26	44.434	
					27	12.646	
					28	17.871	
					29	1.025	
					47	0.293	
					48	1.514	
					49	0.635	
					50	2.832	
	Type III	8	100.000	8.000	10	2.344	11.103
					11	85.010	
					12	12.646	

Finally, the SETA subgraph expansion based algorithm for determining the girth and ACE distributions in LDPC code graphs was used to compare the girth and ACE distributions in ensembles of LDPC codes constructed using the following three PEG algorithms,

i) PEG, i.e. the standard PEG algorithm [Hu et al., 2001], [Hu et al., 2005],

ii) ACE PEG, i.e. the improved PEG (IPEG) construction for higher ACE in LDPC code Tanner graphs [Xiao and Banihashemi, 2004], and

iii) G-ACE PEG, i.e. the generalized ACE constrained PEG algorithm [Vukobratović and Šenk, 2008].

The C programming language codes for the ACE PEG algorithm, and the G-ACE PEG algorithm which were implemented in this research are available in the Appendices (See Appendix D and Appendix E, respectively).

An ensemble of 200 irregular rate-½ $(512, 256)$ LDPC codes were constructed using the PEG, ACE PEG, and G-ACE PEG algorithms; therefore, a total of 600 codes were analyzed. For a fair comparison of the girth and ACE properties of codes constructed by these three PEG algorithms, all the codes in the three code ensembles were constructed with the same DE optimized symbol node degree distribution: $Q_1(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$. Table 4.6 summarizes the results obtained from an analysis of the local girth distributions of codes in the three code ensembles.

Table 4.6: Local girth distribution of codes in ensembles of LDPC codes constructed using standard PEG, ACE PEG, and G-ACE PEG algorithms

Algorithm	No of codes in ensemble	No of codes with 100% of local girths = 6	No of codes which contain local girths of 8	Percentage of codes which contain local girths of 8
PEG	200	193	7	3.5%
ACE PEG	200	167	33	16.5%
G-ACE PEG	200	28	172	86.0%

In Table 4.6, the code ensemble constructed using the G-ACE PEG algorithm has a significantly higher percentage of codes which contain local girths of 8 than the code ensemble constructed using the ACE PEG algorithm. Similarly, the code ensemble constructed using the ACE PEG algorithm has a significantly higher percentage of codes which contain local girths of 8 than the code ensemble constructed using the standard PEG algorithm. Table 4.7 summarizes the results obtained from an analysis of the ACE distributions of codes in the three code ensembles.

Table 4.7: ACE distributions of codes in ensembles of LDPC codes constructed using PEG, ACE PEG, and G-ACE PEG algorithms

Algorithm	Lowest average ACE of code in ensemble	Highest average ACE of code in ensemble	Ensemble average ACE
PEG	13.334	13.707	13.534
ACE PEG	15.152	16.164	15.702
G-ACE PEG	17.109	17.773	17.449

As can be seen in Table 4.7, all the ACE parameters of the code ensemble constructed using the G-ACE PEG algorithm are higher than the ACE parameters of the code ensemble constructed using the ACE PEG algorithm. Similarly, all the ACE parameters of the code ensemble constructed using the ACE PEG algorithm are higher than the ACE parameters of the code ensemble constructed using the standard PEG algorithm. These results are in line with the premise of the work in [Xiao and Banihashemi, 2004] and [Vukobratović and Šenk, 2008].

4.5 Summary

- An efficient algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs has been presented. The algorithm is relatively easy to implement and finds all the short cycles through symbol nodes in Tanner graphs using SETA subgraph expansions.
- With a worst case computational complexity of $O(nm)$, the complexity of the algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs is considerably less

than the complexity of the algorithms in [Halford and Chugg, 2006], [Karimi and Banihashemi, 2013], and [Li et al., 2015].

- Using a *local ACE distribution*, a technique for evaluating the entire spectrum of cycle ACE values in Tanner graphs with respect to the symbol nodes in the graph has been proposed. The following ACE parameters for LDPC code graphs have been introduced and defined: i) the '*local ACE* at a symbol node s_j' or a '*symbol node ACE'*, ii) the '*modal symbol node ACE*' of a Tanner graph, iii) the '*average ACE*' of a graph, and iv) the '*ensemble average ACE*' of an ensemble of codes.
- The importance of having a concise single-valued representation of the degree of connectivity of the entire network of cycles in the Tanner graph of an LDPC code, i.e. the '*average ACE*' of an LDPC code, has been underscored.
- The algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs has been shown to be faster than existing algorithms for counting cycles in LDPC codes when applied to irregular short-to-medium length LDPC codes.
- The impact of code length, Tanner graph density, and local girth distributions on the CPU running time of the algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs has been explained.
- The algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs was used to compare the girth and ACE distributions in three (3) ensembles of irregular rate-½ $(n, k) = (512, 256)$ LDPC codes (all with identical symbol node degree distributions) which were constructed using the PEG, ACE PEG, and G-ACE PEG algorithms respectively. The results show that G-ACE PEG codes have higher girth and ACE metrics than ACE PEG codes, and ACE PEG codes have higher girth and ACE metrics than standard PEG codes. These result are in line with the premise of work by [Xiao and Banihashemi, 2004] and [Vukobratović and Šenk, 2008].

Chapter 5

A cyclic PEG (CPEG) algorithm & The minimum weight, girth, and ACE distributions in irregular LDPC codes constructed using PEG and CPEG algorithms

Parts of this chapter appear in the proceedings of the following conference: Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, September). Improved minimum weight, girth, and ACE distributions in ensembles of short block length irregular LDPC codes constructed using PEG and cyclic PEG (CPEG) algorithms. In *Turbo Codes and Iterative Information Processing (ISTC), 2016 9th International Symposium on* (pp. 186-190). IEEE.

5.1 Introduction

The minimum distance d_{min} of a code is the weight of the lowest weight non-zero codeword of the code. The stopping distance s_{min} of a code is the minimum weight (or size) of a nonempty stopping set of the code. Extensive research has shown that d_{min} and s_{min} have significant bearing on the cardinality of decoding errors under iterative message passing algorithms. In additive white Gaussian noise (AWGN) channels, the two main causes of errors in the error-floor region are low weight codewords (low d_{min}) which cause undetected errors, and trapping sets which cause detected errors. In binary erasure channels (BEC), the two main causes of errors in the error-floor region are low weight codewords which cause undetected errors, and low weight stopping sets (low s_{min}) which cause detected errors. Consequently, codes which have high values of these minimum weight parameters are preferred.

In this chapter, a novel PEG algorithm which uses an alternative edge establishment sequence to construct LDPC codes is proposed. The proposed algorithm is called the *cyclic* PEG (CPEG) algorithm. The CPEG algorithm constructs LDPC codes using an edge connections sequence which is different from the edge connections sequence in the original PEG algorithm in [Hu et al., 2001]. The LDPC code weight evaluation algorithm of [Rosnes et al., 2012] is used to determine the minimum weight distributions, i.e. d_{min} and s_{min} distributions, in ensembles of irregular rate-½ LDPC codes of dimension $(n,\ k) = (512,\ 256)$. Code minimum weight distributions are used to compare ensembles of LDPC codes constructed using standard PEG and CPEG algorithms. The effect of improving the ACE metrics of code graphs on the minimum weight distributions in ensembles of codes constructed using the standard PEG algorithm and the CPEG algorithm are investigated. The ACE of codes constructed using the standard PEG construct and the CPEG construct were improved by implementing the PEG algorithm modification for improving the ACE in LDPC code graphs proposed by [Xiao and Banihashemi, 2004] in both PEG algorithms. Additionally, the algorithm for determining local girth and ACE distributions in short-to-medium length LDPC code Tanner graphs proposed in [Abdu-Aguye et al., 2016(a)] was used to compare global girths, local girth distributions, and local ACE distributions of codes from the different PEG algorithm ensembles. Among the aims of these investigations is to find codes with higher d_{min} and s_{min} than published for similar length irregular LDPC codes which are constructed with 'good' degree distributions, like those reported in [Rosnes et al., 2012]. The impact of d_{min} on the performance of DE optimized irregular rate-½ $(n,\ k) = (512,\ 256)$ LDPC codes is investigated. Finally, the simulated decoding performances of CPEG and standard PEG codes, with and without improved ACE metrics, are compared.

5.2 The Cyclic Progressive Edge-Growth (CPEG) Algorithm

It is necessary to briefly describe some important features of LDPC code construction using the original/standard PEG algorithm before the discussion on the proposed CPEG algorithm is presented. In the standard PEG algorithm, during a Tanner graph construction, the local girth at each symbol node s_j, $\forall \{0 \leq j \leq n\text{-}1\}$, is made as large as possible before a new edge is added to the graph. This results in graphs with large girths. LDPC code Tanner graphs are constructed using predetermined symbol node degree sequences $D_s = \{d_{s_0}, d_{s_1}, \ldots\ldots\ldots, d_{s_{n-1}}\}$, where d_{s_j} is the degree of symbol node s_j. For irregular codes, the symbol node degrees are ordered and connected into LDPC code parity-check matrices/Tanner graphs in the nondecreasing order, i.e. $d_{s_0} \leq d_{s_1} \cdots \leq d_{s_{n-1}}$.

In the standard PEG algorithm, edge connections into a Tanner graph under construction takes place in the following sequence. Firstly, all d_{s_0} edges of symbol node s_0 are connected into the graph, then all the d_{s_1} edges of symbol node s_1 are connected into the graph, and then all the d_{s_2} edges of s_2 are connected into the graph, and so on. The code construction terminates when all the $d_{s_{n-1}}$ edges of symbol node s_{n-1}, i.e. from the 1st to the $(d_{s_{n-1}})^{\text{th}}$ edge of s_{n-1} serially, have been connected into the graph. The edge connections sequence of the standard PEG algorithm can be described as *symbol node serial*. Figure 2.13 shows the edge connections sequence of the standard PEG algorithm when constructing a code with symbol node degree sequence D_{S_1} = {2, 2, 2, 2, 3, 3, 5, 5, 7, 7}.

A PEG algorithm that implements an alternative edge connections sequence to the standard PEG algorithm is proposed and investigated. The proposed cyclic PEG (CPEG) algorithm for constructing LDPC codes with n symbol nodes and m check nodes is summarized in Algorithm 5.1.

In the CPEG algorithm, edge connections into a Tanner graph under construction proceeds in the following sequence. Firstly, the first of the d_{s_0} edges incident to symbol node s_0 is connected into the graph, and then the first of the d_{s_1} edges incident to s_1 is connected into the graph, and then the first of the d_{s_2} edges incident to s_2 is connected into the graph and so on, up to the last symbol node s_{n-1}. After all the first edges incident to each of the symbol nodes in the graph have been connected, the

algorithm returns to the lowest index symbol node s_j with symbol degree $d_{s_j} \geq 2$ and connects the second of the d_{s_j} edges incident to symbol node s_j into the graph, then the second of the $d_{s_{j+1}}$ edges incident to symbol node s_{j+1} is connected into the graph and so on, up to symbol node s_{n-1}. CPEG code construction terminates when the $(d_{s_{n-1}})^{\text{th}}$ edge of symbol node s_{n-1} has been connected into the graph. The CPEG algorithm edge connections sequence can be described as *symbol node cyclic*.

Algorithm 5.1: The cyclic progressive edge-growth (CPEG) algorithm

for $cycles = 0\ to\ (d_{s_{max}} - 1)$ **do**
begin
$k = cycles$
for j = 0 to (n-1) **do**
begin
if $k \leq (d_{s_j}\text{-}1)$
if $k = 0$
$E_{s_j}^0 \leftarrow$ edge (c_i, s_j), where $E_{s_j}^0$ is the first edge incident to s_j, and c_i is selected from the check nodes with the lowest check node degree under the current graph setting $E_{s_0} \cup E_{s_1} \cup \ldots \cup E_{s_{(j-1)}}$.
else
expand a subgraph from symbol node s_j up to depth l under the current graph setting such that the cardinality of $N_{s_j}^l$ stops increasing but is less than m, or $\dot{\mathrm{N}}_{s_j}^l \neq \emptyset$ but $\dot{\mathrm{N}}_{s_j}^{l+1} = \emptyset$, then $E_{s_j}^k \leftarrow$ edge (c_i, s_j), where $E_{s_j}^k$ is the $(k+1)$-th edge incident to s_j and c_i is a check node selected from the lowest degree check nodes in the set $\dot{\mathrm{N}}_{s_j}^l$.
end
end

In the CPEG algorithm, the highest symbol node degree $d_{s_{max}}$ in the degree distribution D_s of the code under construction determines the number of edge connections cycles (see the variable called '$cycles$' in Algorithm 5.1) which must be executed to complete a code construction. In each cycle, edges are connected only for symbol nodes with an incomplete number of edges under the current state of the graph. In cycle k, this is the set of symbol nodes for which $k \leq (d_{s_j}\text{-}1)$. Figure 5.1 illustrates the edge connections sequences for the CPEG algorithms when constructing a code with symbol node degree sequence D_{S_1} = {2, 2, 2, 2, 3, 3, 5, 5, 7, 7}. There are seven edge connections

cycles in the CPEG algorithm edge connections sequence shown in Figure 5.1 because the highest symbol node degree $d_{s_{max}}$ in symbol node degree sequence D_{S_1} is seven (7).

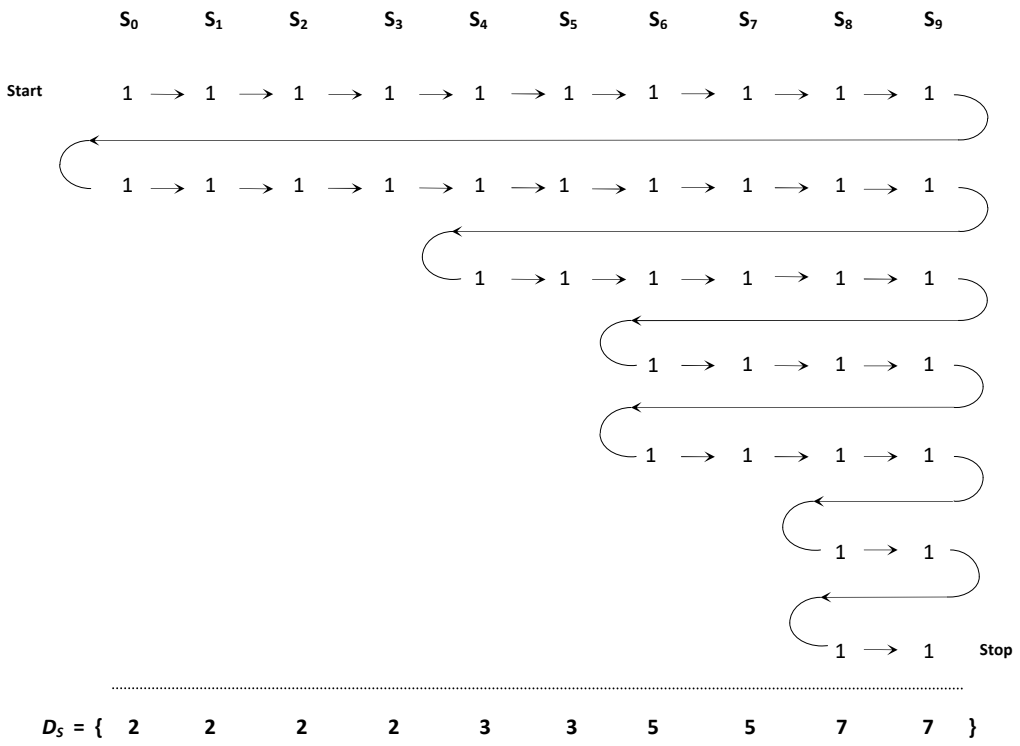

Figure 5.1: The edge connections sequence of the CPEG algorithm

The edge connections sequence of the CPEG algorithm is similar to that of the 'ModPEG' algorithm of [Sy et al., 2011]. The edge connections sequence of the ModPEG algorithm is described using degree distribution D_{S_1} as follows. Let $\{S_d\}$ denote the set of symbol nodes of degree d, where $1 \leq d \leq d_{s_{max}}$. There are four sets of equal degree symbol nodes in D_{S_1}, i.e. $\{S_2\}$, $\{S_3\}$, $\{S_5\}$, and $\{S_7\}$. Where $\{S_2\} = \{s_0, s_1, s_2, s_3\}$, $\{S_3\} = \{s_4, s_5\}$, $\{S_5\} = \{s_6, s_7\}$, and $\{S_7\} = \{s_8, s_9\}$. Starting from $\{S_2\}$, 2 edge connections cycles are executed to connect the 2 edges for each symbol node in the set (the first cycle for the 1st edges, and the second cycle for the 2nd edges). Similarly, 3 cycles are executed to connect the 3 edges for each symbol node in $\{S_3\}$, and so on. The ModPEG code construction terminates after the 7 cycles required to connect all the edges of $\{S_7\}$ are executed. Figure 5.2 illustrates the edge connections sequence of the ModPEG algorithms of [Sy et al., 2011].

It is clear from a comparison of the edge connections sequences illustrated in Figure 5.1 and Figure 5.2 that the ModPEG and the CPEG algorithms are different.

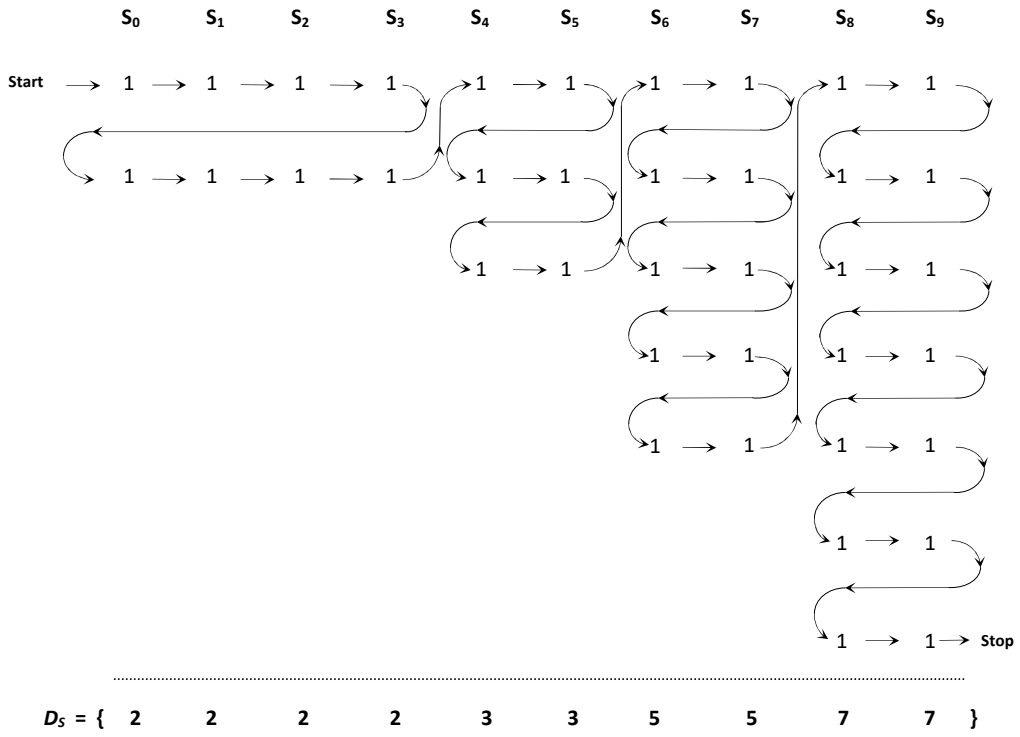

Figure 5.2: The edge connections sequence of the ModPEG algorithm

5.3 Minimum Codeword Weight and Minimum Stopping Set Weight Distributions in PEG, ACE PEG, CPEG, and ACE CPEG Code Ensembles

In this section, we compare the minimum weight distributions, i.e. d_{min} and s_{min} distributions, in four ensembles of LDPC codes which all have the same density evolution (DE) optimized symbol node degree distribution but are constructed using different PEG algorithms. Two of these code ensembles were constructed using the standard PEG alogithm and CPEG algorithm, respectively. The other two code ensembles were constructed using the algorithms derived when the PEG algorithm modification for improving the ACE in LDPC code graphs proposed by [Xiao and Banihashemi, 2004] was implemented in both the standard PEG algorithm and CPEG algorithm, these are the

ACE PEG algorithm and the ACE CPEG algorithm, respectively. The C programming language code for the CPEG algorithm and the ACE CPEG algorithm for LDPC code construction are available in the Appendices (See Appendix F and Appendix G, respectively).

An LDPC code ensemble was constructed using each of the following PEG algorithms.

i) **PEG**, i.e. the standard PEG algorithm,

ii) **ACE PEG**, i.e. the standard PEG algorithm modified for improved ACE in codes,

iii) **CPEG**, i.e. the cyclic PEG algorithm, and

iv) **ACE CPEG**, i.e. the CPEG algorithm modified for improved ACE in codes.

The four code ensembles were made up of irregular rate-½ codes of dimension $(n,\ k) = (512,\ 256)$, where n is the code length and k is the information length. All codes in the four code ensembles were constructed using the following density evolution (DE) optimized symbol node degree distribution: $\lambda_A(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$. This degree distribution was obtained from [Tjhai, 2007]. A random candidate check node selection, as adopted in [Hu et al., 2001] and [Hu et al., 2005], was used to construct all codes in the four code ensembles. The only code design parameter which differentiates one code from another within an ensemble is the random number seed used for its construction. Each code ensemble comprised of 6000 LDPC code matrices.

The LDPC code weight evaluation algorithm was used to determine the minimum weight distributions for codes in each of the four code ensembles. The minimum weights in the rate-½ $(512,\ 256)$ LDPC code ensembles were compared to the highest minimum weights obtained for the rate-½ $(576,\ 288)$ IEEE 802.16E (WiMAX) LDPC code published in [Rosnes et al, 2012], which are $d_{min} = 16$ and $d_{min} = 15$. These minimum weights were obtained using the alternative construction for the WiMAX code (see [Rosnes et al, 2012] for details).

Figure 5.3 shows the minimum distance d_{min} weight distributions in LDPC code ensembles constructed using PEG, ACE PEG, CPEG, and ACE CPEG algorithms. It can be seen that a

significant fraction of codes in each ensemble have minimum distances greater than or equal to that of the WiMAX code, i.e. $d_{min} \geq 16$. PEG algorithms that use the edge connections sequence of the standard PEG algorithm produced significantly higher fractions of codes with improved d_{min} than corresponding CPEG algorithms. PEG and ACE CPEG algorithms have similar d_{min} distributions. The ACE PEG algorithm constructed the code ensemble with the highest fraction of high d_{min} codes. Codes with $d_{min} = 20$ were found in the ACE PEG code ensemble.

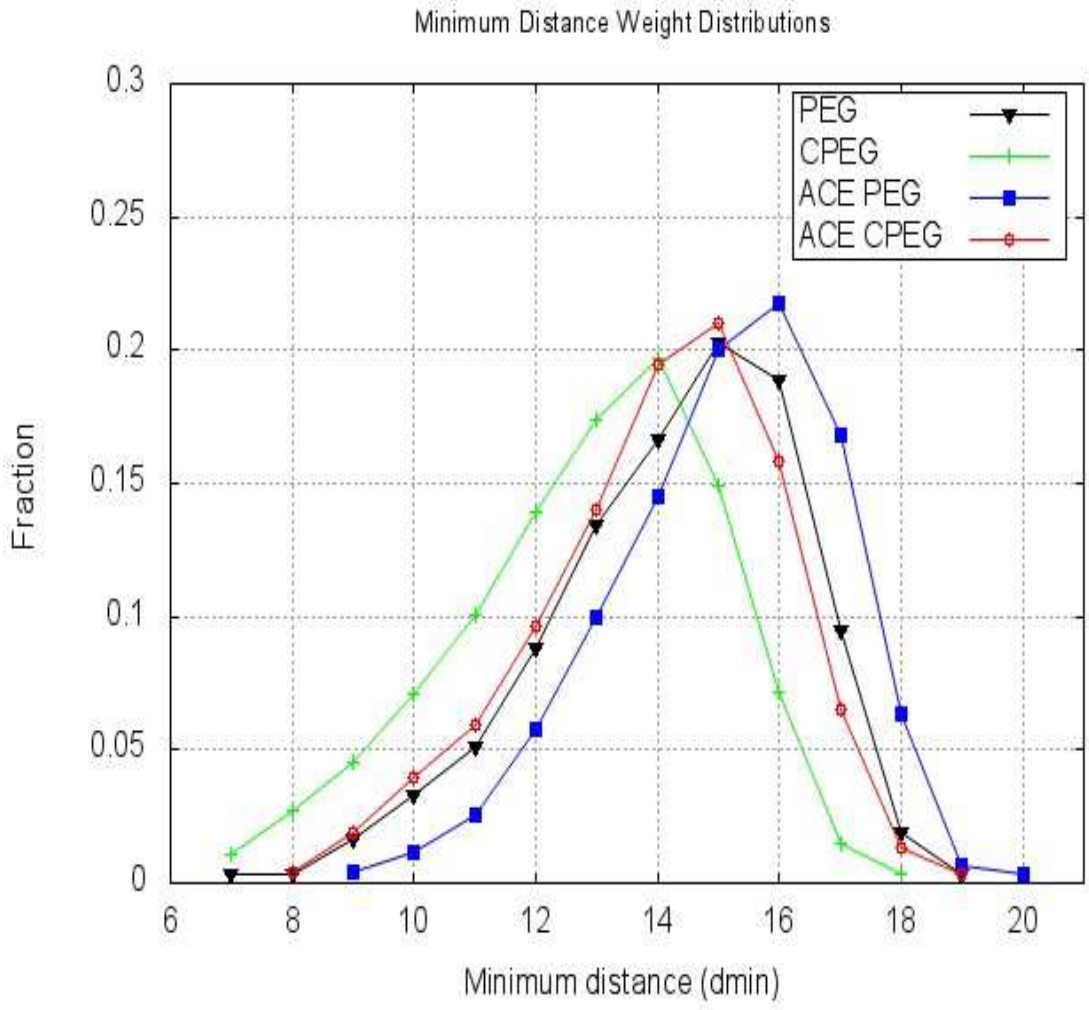

Figure 5.3: Minimum distance weight d_{min} distributions in ensembles of rate-½ $(512, 256)$ LDPC codes constructed using PEG, ACE PEG, CPEG, and ACE CPEG algorithms

Figure 5.4 shows the minimum stopping set s_{min} weight distributions in the four LDPC code ensembles. A significant fraction of codes in each ensemble were found to have minimum stopping set weights greater than or equal to that of the WiMAX code, i.e. $s_{min} \geq 15$. PEG algorithms produced

significantly higher fractions of codes with improved s_{min} than corresponding CPEG algorithms. PEG and ACE CPEG code ensembles have very similar s_{min} distributions. Clearly, the ACE PEG algorithm constructed the code ensemble with the highest fraction of high s_{min} codes. Codes with $s_{min} = 20$ were found in the ACE PEG code ensemble.

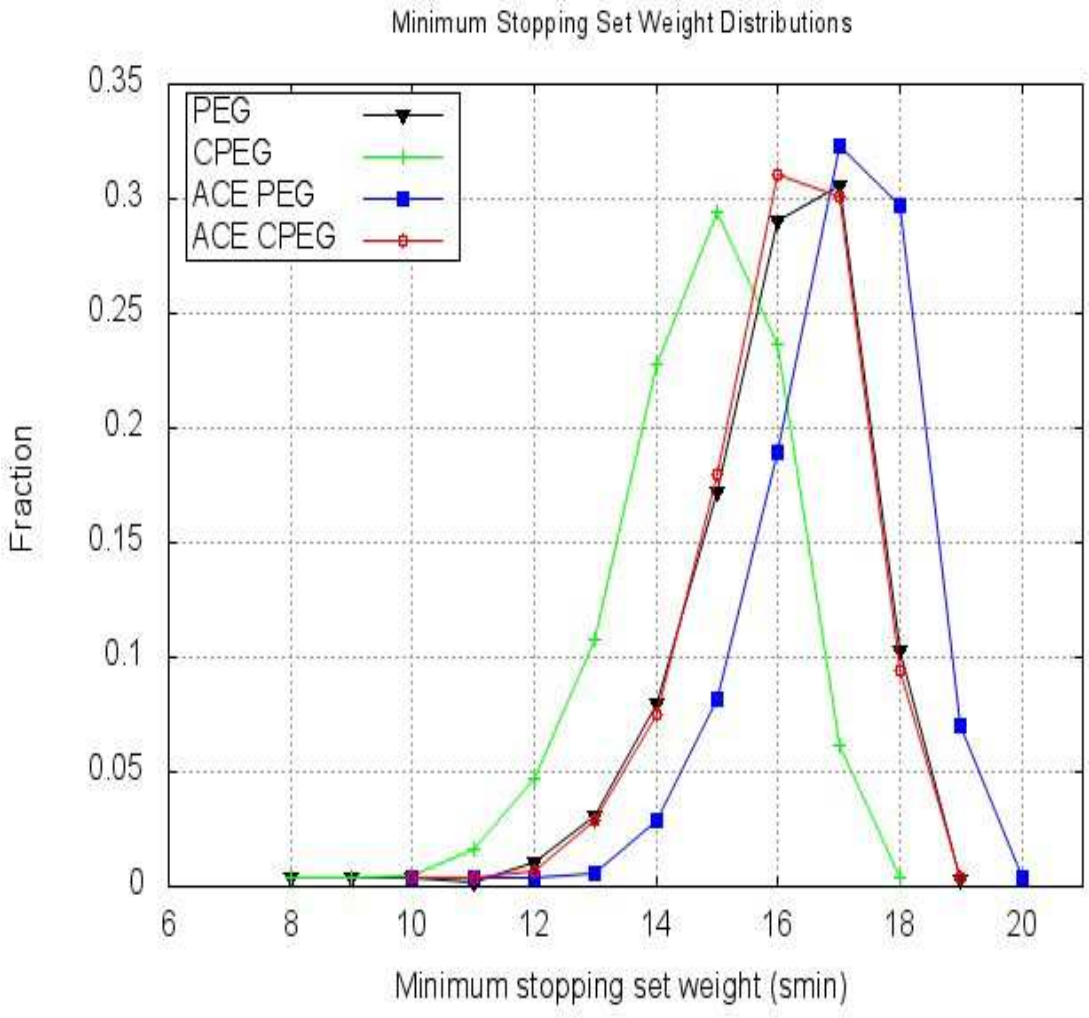

Figure 5.4: Minimum stopping set weight s_{min} distributions in ensembles of rate-½, $(512, 256)$ LDPC codes constructed using PEG, ACE PEG, CPEG, and ACE CPEG algorithms

Although codes with higher minimum weights than the WiMAX code, i.e. with $d_{min} > 16$ and $s_{min} > 15$, were found in all four code ensembles, the ACE PEG algorithm constructed the LDPC code ensemble with the largest fraction of high minimum weight codes.

For the same code rate and dimensions (n, k), higher minimum weights, i.e. d_{min} and s_{min}, than those found in the four DE optimized LDPC code ensembles can be easily found in regular LDPC codes and irregular LDPC codes with 'bad' degree distributions. Unfortunately, regular codes and irregular codes with these 'bad' degree distributions have poor decoding performance compared to irregular codes with 'good'/DE optimized degree distributions.

As an example, the highest d_{min} and s_{min} for a code in the ensemble of rate-½ $(n, k) = (1000, 500)$ LDPC codes in [Richter and Hof, 2006] had $d_{min} = s_{min} = 18$. All the codes in the ensemble had unoptimized degree distribution $\lambda_{RH}(x) = 0.283x^2 + 0.281x^3 + 0.436x^9$. Consequently, it was necessary to use the shuffled belief propagation decoder [Zhang and Fossorier, 2005], a maximum of 500 iterations, and the 'linear approximation' method in [Richter et al., 2005] in order to attain the reported decoding performance.

As another example, the DE optimized degree distribution used to construct the four LDPC code ensembles investigated for d_{min} and s_{min} distributions, i.e.,

$$\lambda_A(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$$

was modified so that all symbol nodes with degrees of 2 were converted to symbol nodes with degrees of 3. Consequently, the following 'bad'/unoptimized symbol node degree distribution was obtained,

$$\lambda_B(x) = 0.74805x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$$

When degree distribution $\lambda_B(x)$ was used to construct an ensemble of rate-½ $(n, k) = (512, 256)$ LDPC codes using the ACE PEG algorithm, most of the codes had $d_{min} > 24$ and $s_{min} > 24$. However, simulation experiment results show that the error-floor for rate-½ $(n, k) = (512, 256)$ ACE PEG codes with symbol node degree distribution $\lambda_B(x)$ were significantly higher than the error-floor for rate-½ $(512, 256)$ ACE PEG codes with symbol node degree distribution $\lambda_A(x)$. The results of performance simulation at $4.00dB$ and a maximum of 100 decoding iterations showed that $P_{e(\lambda_B)} = 2.55 \times 10^{-7}$, i.e. the average probability of error for 10 randomly selected ACE PEG codes with 'bad'/unoptimized degree distribution $\lambda_B(x)$ is 2.55×10^{-7}, and $P_{e(\lambda_A)} = 8.35 \times 10^{-8}$, i.e. the average probability of error for 10 randomly selected ACE PEG codes with DE optimized degree

distribution $\lambda_A(x)$ was 8.35×10^{-8}. Therefore, $P_{e(\lambda_B)} \sim 3P_{e(\lambda_A)}$. In order to minimize the effect of the d_{min} and s_{min} of the simulated codes on the values of $P_{e(\lambda_B)}$ and $P_{e(\lambda_A)}$, all the codes used in these simulation experiments had $d_{min} \geq 14$ and $s_{min} \geq 14$.

5.4 The Effect of Minimum Distance d_{min} on the Error-Floor Performance of Short-to-Medium Length LDPC Codes over the AWGN Channel

In this section, the result of simulation experiments to observe the generalized effect of minimum distances d_{min} on the error-floor performance of DE optimized rate-½ $(n, k) = (512, 256)$ LDPC codes over the AWGN channel is presented. All the codes investigated in this section were constructed using the ACE PEG algorithm with DE optimized symbol node degree distribution $\lambda_A(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$. Code performance simulations were carried out using the standard BP/SPA message passing iterative decoder which was implemented as described in Section 2.5. Decoding was carried out using the standard BP/SPA decoder for BPSK modulated LDPC codes over the AWGN channel. Error-floor performance results were obtained using a maximum of 100 decoding iterations at Eb/No of 4.00dB only. At 4.00dB the decoding performance curves for the ACE PEG algorithm constructed rate-½ (512, 256) codes over the AWGN channel is in the error-floor region. Although the entire error-floor performance of these LDPC codes cannot be completely characterized at one Eb/No value, the singular Eb/No value of 4.00dB was chosen for these experiments because extensive simulations carried out in this research have shown that these LDPC codes maintain their performance hierarchies across the entire range of Eb/No values which were utilized in the error-floor region.

It can be seen from Figure 5.3 that the rate-½ $(n, k) = (512, 256)$ LDPC codes in the ensemble of 6000 ACE PEG codes with degree distribution $\lambda_A(x)$ have minimum distances d_{min} in the range of 9 to 20. For each of the minimum distances in this range, i.e. $9 \leq d_{min} \leq 20$, ten (10) LDPC codes were randomly selected from the ACE PEG code ensemble. That is, ten codes with $d_{min} = 9$, ten codes with $d_{min} = 10$, ten codes with $d_{min} = 11$, . . . , and ten codes with $d_{min} = 20$ were randomly

selected. However, it was ensured that the minimum stopping set weight, s_{min} of each of the ten randomly selected codes for each d_{min} was at least 16, i.e. $s_{min} \geq 15$ for each code. This restriction on the s_{min} values of the selected codes was made in an attempt to minimize the influence of stopping sets on the results of the subsequent experiments which are intended to investigate only the effect of the d_{min} of short-to-medium length LDPC codes on their BP/SPA decoding performance.

The FER performance of the ten (10) LDPC codes selected for each d_{min} in the range $9 \leq d_{min} \leq 20$ were determined using the standard BP/SPA decoder at $Eb/No = 4.00$dB. Table 5.1 shows the performance simulations results obtained for the 120 randomly selected ACE PEG codes with d_{min} in the range of $9 \leq d_{min} \leq 20$, and $s_{min} \geq 16$.

Table 5.1: Probabilty of error for 10 randomly selected DE optimized rate-½ $(512, 256)$ ACE PEG codes with d_{min} in the range $9 \leq d_{min} \leq 20$

	Probability of decoding error, P_e, at $Eb/No = 4.00$dB ($\times \mathbf{10^{-7}}$)										
d_{min}	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6	Code 7	Code 8	Code 9	Code 10	Average
9	18.007	16.627	18.375	16.896	17.810	17.385	17.932	16.701	17.568	16.779	17.408
10	7.155	6.945	7.573	7.558	7.372	6.930	7.330	6.757	7.413	7.268	7.231
11	2.174	2.235	2.194	2.352	2.192	2.281	2.316	2.448	2.315	2.463	2.297
12	1.548	1.542	1.615	1.485	1.433	1.538	1.520	1.389	1.563	1.687	1.532
13	1.229	1.324	1.204	1.228	1.184	1.216	1.180	1.255	1.183	1.307	1.231
14	0.667	0.693	0.723	0.730	0.657	0.704	0.724	0.685	0.575	0.732	0.689
15	0.789	0.820	0.769	0.785	0.755	0.736	0.822	0.749	0.798	0.827	0.785
16	0.663	0.748	0.697	0.723	0.642	0.617	0.686	0.758	0.615	0.621	0.677
17	0.814	0.734	0.636	0.750	0.615	0.628	0.807	0.781	0.692	0.713	0.717
18	0.725	0.704	0.594	0.657	0.723	0.732	0.670	0.693	0.724	0.678	0.690
19	0.723	0.627	0.664	0.720	0.728	0.697	0.647	0.680	0.714	0.730	0.693
20	0.614	0.803	0.636	0.680	0.734	0.672	0.713	0.740	0.628	0.810	0.703

From these average probabilities of decoding error shown in Table 5.1, P_e at $Eb/No = 4.00$dB, it can be observed that as the minimum distance of the rate-½ $(n, k) = (512, 256)$ ACE PEG codes

increases from $d_{min} = 9$ to $d_{min} = 14$, there is a significant lowering of the average error-floor performance of these LDPC codes. Figure 5.5 shows a plot of the minimum distance d_{min} of ACE PEG codes against the average probability of decoding error at $Eb/No = 4.00$dB.

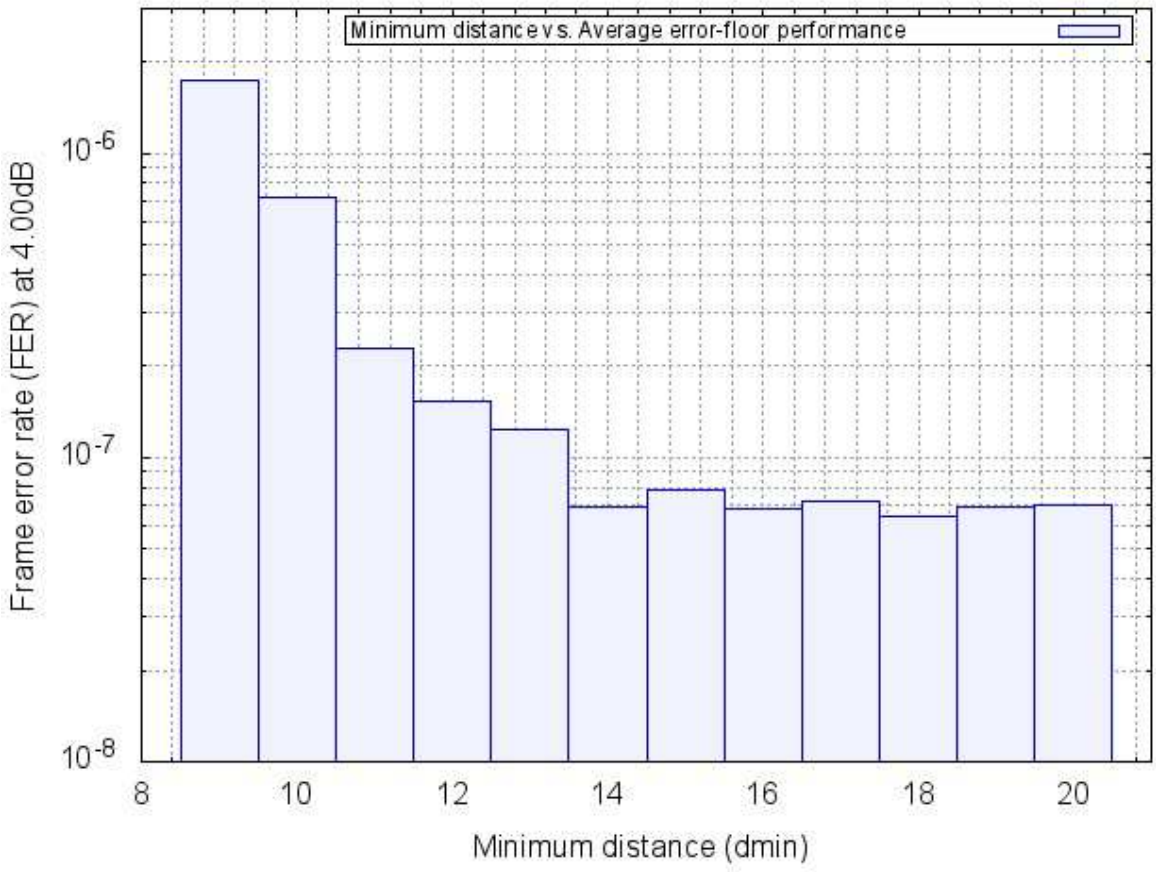

Figure 5.5: Minimum distance d_{min} vs. average error-floor performance of DE optimized rate-½ $(512, 256)$ ACE PEG codes (Average FER for 10 codes at $Eb/No = 4.00$dB)

It can be observed from Table 5.1 and Figure 5.5 that there is a generalized improvement in the error-floor performance of DE optimized rate-½ $(512, 256)$ ACE PEG codes as the d_{min} of these codes increases from $d_{min} = 9$ to $d_{min} = 14$. However, as the d_{min} of these codes increases beyond $d_{min} = 14$, i.e. within the range $15 \leq d_{min} \leq 20$, there is no observable generalized improvement in the error-floor performance consequent to increasing d_{min}. Most of the codes selected and analysed in the range $15 \leq d_{min} \leq 20$ had stopping distances s_{min} which were either equal to or larger that the code minimum distances d_{min}. That is, in order to minimize the effect of s_{min} on the decoding performance results obtained for codes within the range $15 \leq d_{min} \leq 20$, wherever possible the codes

analysed were selected such that $s_{min} \geq d_{min}$.

These results suggest that there are threshold d_{min} values for LDPC codes used over AWGN channels such that deploying LDPC codes with d_{min} above these threshold values have little or no effect on improving the error-floor performance under BP/SPA message-passing iterative decoding. Therefore, it may be concluded that beyond certain threshold d_{min} values, higher values of d_{min} in short-to-medium length LDPC codes will not necessarily result in a generalized lower error-floor performance over AWGN channels. Given a fixed code rate, these threshold d_{min} values will depend on the degree distributions and length of the LDPC codes under consideration.

A decoding error which is as a result of the minimum distance of a code, or a d_{min} decoding error, occurs when a transmitted codeword is decoded into another codeword of the code such that the Hamming distance between the two codewords is equal to the d_{min} of the code. These d_{min} decoding errors occur when the channel noise distortion causes the Euclidean distance between the received codeword vector and the decoded codeword to be less than the Euclidean distance between the received codeword vector and the transmitted codeword. It was observed, from the simulation results obtained for the DE optimized rate-½ $(512, 256)$ ACE PEG codes, that for codes with $d_{min} \geq 14$ the fraction of decoding errors which are due to d_{min} decoding errors are very low compared to the fraction decoding errors which are due to d_{min} decoding errors for codes with $d_{min} \leq 14$. In fact, while d_{min} decoding errors occasionally occured in codes with $d_{min} = 14$, the decoding performance of almost all the ACE PEG codes with $15 \leq d_{min} \leq 20$ were determined without a single d_{min} decoding error being encountered. Consequently, for rate-½ $(512, 256)$ ACE PEG codes constructed with DE optimized degree distribution $\lambda_A(x)$ the threshold $d_{min} = 14$, and other causes of decoding failures apart from d_{min} decoding errors dominate the error-floor performance of the codes with d_{min} values in the range $15 \leq d_{min} \leq 20$.

5.5 Global Girth, Local Girth and Average ACE Distribution in PEG, ACE PEG, CPEG and ACE CPEG Code Ensembles

In this section, the SETA subgraph expansion based algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs [Abdu-Aguye et al., 2016(a)] was used to compare the global girths, local girth distributions, and ACE distributions in sub-ensembles of the irregular rate-½ $(512, 256)$ LDPC code ensembles constructed using the PEG, ACE PEG, CPEG, and ACE CPEG algorithms. The first 200 codes from each of the four code ensembles (i.e. constructed using random number seeds 1 to 200) were chosen for these comparisons. Consequently, a total of 800 codes from the four code ensembles to be compared were analysed.

5.5.1 Global Girth Distributions

As a consequence of using the same DE optimized degree sequence to construct the codes in the four ensembles, all of the 800 codes analyzed had global girth $g = 6$. None of the 200 codes selected from each of the four code ensembles had a symbol node with a local girth g_{s_j} of less than 6. That is, for the 800 codes analyzed, $g_{s_j} \geq 6, \forall \{0 \leq j \leq n - 1\}$.

5.5.2 Local Girth Distributions

Table 5.2 is a summary of the local girth distributions in the chosen sub-ensembles of the PEG, ACE PEG, CPEG, and ACE CPEG code ensembles. In most codes constructed using the four PEG algorithms, 100% of the symbol nodes in their graphs had local girths of 6. The remaining codes constructed had slightly improved girths with at least one symbol node in their code graph having a local girth of 8. The CPEG code ensemble had slightly higher local girth properties than the PEG code ensemble. However, the ACE PEG and ACE CPEG code ensembles had significantly higher local girth properties than PEG and CPEG code ensembles. Of the four code ensembles, the ACE CPEG ensemble had the largest local girth distribution with 44.0% of its codes containing symbol nodes with local girths of 8. Out of a total of 144 code graphs with local girths of 8, it was observed that the number of symbol nodes which had local girths of 8 did not exceed 2 in PEG, ACE PEG and CPEG code ensembles. However, the number of symbol nodes which had local girths of 8 improved to a range of 1 to 4 in the ACE CPEG code ensemble.

Table 5.2: Local girth distributions in the four sub-ensembles of irregular rate-½ $(512, 256)$ PEG codes

Algorithm	No of codes in ensemble	No of codes with 100% of local girths = 6	No of codes which contain local girths of 8	Percentage of codes with local girths of 8
PEG	200	193	7	3.5%
CPEG	200	184	16	8.0%
ACE PEG	200	167	33	16.5%
ACE CPEG	200	112	88	44.0%
Total	800	656	144	18.0%

It is clear from these results that the edge connections sequence of CPEG algorithms results in code ensembles with higher girth properties than those of code ensembles constructed using the edge connections sequence of standard PEG algorithms.

It can be observed that the CPEG and ACE CPEG code ensembles have higher local girth distributions than the PEG and ACE PEG code ensembles respectively. The ACE CPEG code ensemble had the highest local girth distribution.

5.5.3 ACE Distributions

Table 5.3 is a summary of the result of ACE distribution analysis carried out on the four code sub-ensembles. The CPEG code ensemble has a higher ensemble average ACE than the PEG code ensemble, and the ACE CPEG code ensemble has a higher ensemble average ACE than the ACE PEG code ensemble. The ACE CPEG algorithm has the highest ensemble average ACE. This shows that the edge connections sequence of CPEG algorithms results in code ensembles with higher ACEs than standard PEG algorithms.

In the edge connections sequence of the CPEG algorithm, edges of high degree symbol nodes are connected into the graph in each code construction cycle, but in the edge connections sequence of the standard PEG algorithm, edges of high degree symbol nodes are connected into the graph only at the final stages of graph construction. The edge connections sequence of the CPEG algorithm results in an

early and uniform involvement of high degree symbol nodes at the early stages of graph constructions. This generally results in code graphs with higher ACE parameters.

Table 5.3: A summary of the ACE distributions in the four sub-ensembles of irregular rate-½ $(512, 256)$ PEG codes

Algorithm	Lowest average ACE of code in ensemble	Highest average ACE of code in ensemble	Modal symbol node ACE | Percentage of ensemble		Ensemble average ACE
PEG	13.334	13.707	13 14	99.5% 0.5%	13.534
CPEG	13.299	13.904	13	100%	13.623
ACE PEG	15.152	16.164	14 15	96.0% 4.0%	15.702
ACE CPEG	15.736	16.705	14 15	98.5% 1.5%	16.103

Figure 5.6 shows frame error rate (FER) simulation curves for the best codes found in the PEG, ACE PEG, CPEG, and ACE CPEG code sub-ensembles. These simulations were carried out for binary phase-shift keying (BPSK) modulation over the AWGN channel. Codes were decoded using the standard BP/SPA decoder at a maximum of 100 iterations.

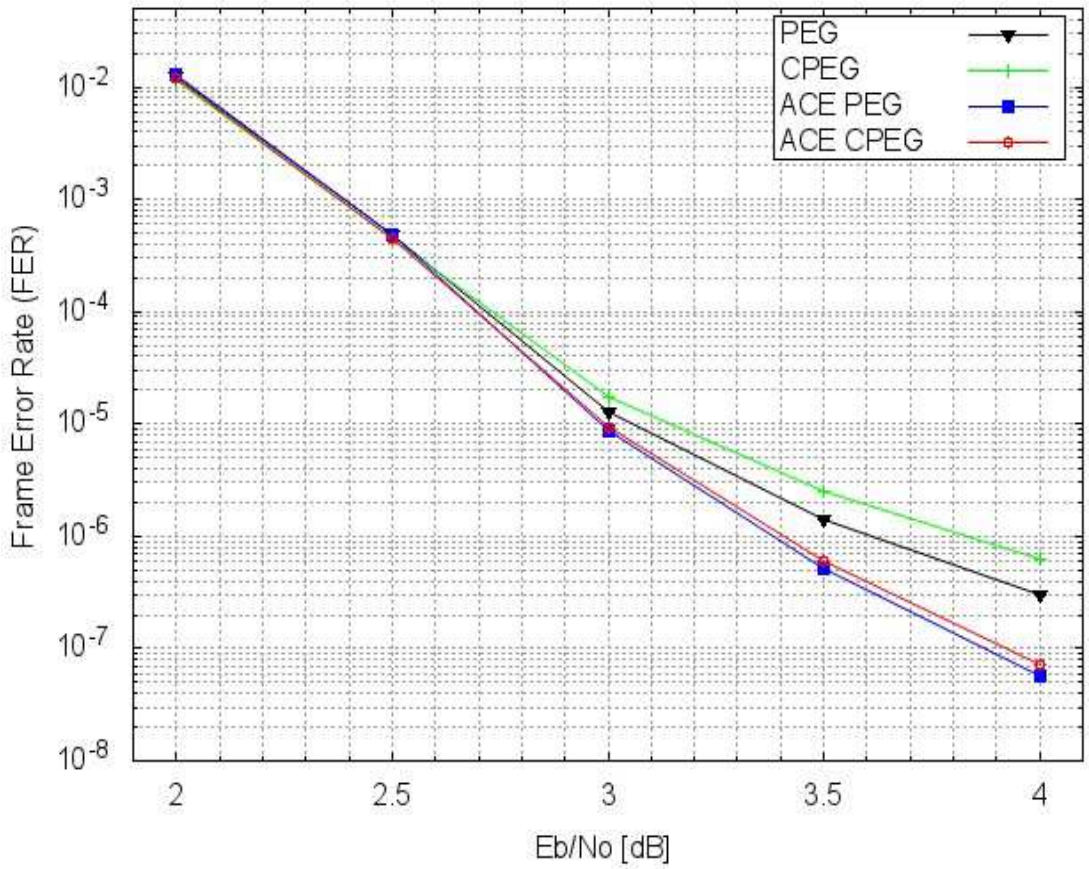

Figure 5.6: FER curves for irregular rate-½ $(512, 256)$ codes constructed using PEG, CPEG, ACE PEG, and ACE CPEG algorithms

It can be seen in Figure 5.6 that the best PEG code found has a significantly lower error-floor than the best CPEG code. Similarly, the ACE PEG code slightly outperforms the ACE CPEG code in the error-floor region. As expected, ACE PEG and ACE CPEG codes have lower error-floor than PEG and CPEG codes. Despite constructing code ensembles with higher girth and ACE distributions, the use of CPEG algorithms did not result in LDPC codes with lower error-floor than achieved by LDPC codes constructed using the standard PEG algorithm edge connections sequence shown in Figure 2.13.

5.6 Summary

- In a bid to improve the ACE in short-to-medium length LDPC codes, a cyclic PEG (CPEG) algorithm has been presented. The CPEG algorithm constructs LDPC codes using an edge connections sequence which is different from that of the original PEG algorithm.

- The minimum weigth distributions, i.e. d_{min} and s_{min} distributions, in four (4) ensembles of 6000 irregular rate-½ $(n, k) = (512, 256)$ LDPC codes each were compared. These code ensembles were constructed using the PEG, ACE PEG, CPEG, and ACE CPEG algorithms respectively. All of the 24000 codes in the four code ensembles were constructed using the same DE optimized degree distribution: $\lambda_A(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$. Rate-½ $(n, k) = (512, 256)$ irregular LDPC codes with DE optimized/'good' degree distributions which have higher d_{min} and s_{min} than have been published for similar LDPC codes were found in the four code ensembles.

- Rate-½ $(512, 256)$ irregular LDPC codes with higher d_{min} and s_{min} than published for the rate-½ $(576, 288)$ IEEE 802.16E (WiMAX) LDPC code in [Rosnes et al, 2012], i.e. $d_{min} = 16$ and $d_{min} = 15$, were found in the PEG, ACE PEG, CPEG, and ACE CPEG code ensembles.

- Comparisons of the minimum weight distributions in the PEG, ACE PEG, CPEG, and ACE CPEG code ensembles reveals that the codes in the ACE PEG ensemble had the highest d_{min} and s_{min} distributions. Some DE optimized rate-½ $(n, k) = (512, 256)$ LDPC codes with $d_{min} = 20$, and others with $s_{min} = 20$ were found in the ACE PEG code ensemble.

- Based on the DE optimized rate-½ $(n, k) = (512, 256)$ LDPC codes in the ACE PEG code ensemble, the generalized effect of LDPC code minimum distance d_{min} on their error-floor performance over the AWGN channel have been investigated. The codes in the ACE PEG code ensemble have d_{min} in the range $9 \leq d_{min} \leq 20$. Results show that while there is a generalized improvement in the error-floor performance of the ACE PEG codes as d_{min} increases from $d_{min} = 9$ to $d_{min} = 14$, above $d_{min} = 14$, i.e. within the range $15 \leq d_{min} \leq 20$, there is no observable generalized improvement in the error-floor performance consequent to increasing the value of d_{min}. There appears to be threshold values for the d_{min} of LDPC codes above which further increments to d_{min} do not result in lower error-floor over AWGN channels.

- The global girths, girth distributions, and ACE distributions in four (4) ensembles of irregular rate-½ $(n, k) = (512, 256)$ LDPC codes have been compared. The four code ensembles of 200 codes each were selected from the larger PEG, ACE PEG, CPEG, and ACE CPEG code ensembles. Results show that the ACE CPEG algorithm constructed the code ensemble with the highest local girth and ACE distributions. However, simulation results show that despite improved girth and ACE distributions, CPEG and ACE CPEG codes do not improve on the error-floor performance achievable by PEG and ACE PEG codes respectively.

Chapter 6

Lowering the error-floor of short-to-medium length LDPC codes: Improved PEG algorithms and degree distributions

This chapter discusses the construction and search for irregular rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ LDPC codes with the lowest-error-floor performance over the AWGN channel under standard BP/SPA decoding. The primary focus of the chapter is on the construction of short-to-medium length LDPC codes with the lowest possible error-floor using improved PEG algorithms and symbol node degree distributions.

Parts of this chapter appear in the proceedings of the following conference: Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, September). Lowering the error floor of short-to-medium length LDPC codes using optimal low-correlated-edge density (OED) PEG Tanner graphs. In *Software, Telecommunications and Computer Networks (SoftCOM), 2016 24th International Conference on* (pp. 1-5). IEEE.

6.1 Introduction

Cycles are necessary in the Tanner graph of capacity-approaching LDPC codes. However, under iterative message-passing decoding algorithms such as the BP/SPA, the presence of cycles cause edges (and the corresponding symbol nodes) to become correlated (i.e. interdependent) after a few iterations. High edge correlations significantly reduces the decoding efficiency of iterative message-passing decoding algorithms. The shorter a cycle the higher the correlation of its member symbols. Consequently, 4-cycles are generally undesirable in linear code graphs. Most of the good symbol node

degree distributions used for constructing irregular LDPC codes, such as those optimized through DE, avoid constructing graphs with local girths of 4, and others avoid local girths of 4 and 6. When used in the PEG algorithm to construct LDPC codes, DE optimized degree distributions typically result in Tanner graphs with edge densities which are below the highest densities achievable in codes of identical length, rate, and girth. That is, at the desired global girth, LDPC codes with DE optimized degree distributions typically have submaximal Tanner graph edge densities.

A modification to the PEG algorithm in order to control the girth of LDPC code Tanner graphs constructed is proposed. Using the proposed modified PEG algorithm, the edge densities in short-to-medium length irregular LDPC code graphs can be increased without increasing the correlation between edges in graphs by creating cycles of undesirably short girth. A DE optimized symbol node degree distribution used for constructing short-to-medium length LDPC codes is further optimized (slightly altered) while maintaining the maximum edge correlation (minimum local girths) that exists in graphs constructed using the unaltered DE optimized distributions. As a result of improved Tanner graph edge densities and symbol node degree distributions, rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ LDPC codes with lower error-floor than published for LDPC codes of identical length and rate have been found. Based on slight modifications to one of the best known DE optimized degree distributions for short-to-medium length irregular rate-½ LDPC codes, when the proposed technique for obtaining denser LDPC codes is implemented in addition to the improved PEG (IPEG) construction of [Xiao and Banihashemi, 2004], the error-floor of the rate-½ $(n, k) = (512, 256)$ LDPC codes constructed are generally lower than the error-floor of LDPC codes constructed using the unmodified DE optimized degree distribution and the IPEG construction.

Investigations into the performance of rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ LDPC codes constructed using the generalized ACE constrained PEG (G-ACE PEG) algorithm of [Vukobratović and Šenk, 2008] are undertaken. Experiments are carried out in an attempt to improve the performance of G-ACE PEG codes through further Tanner graph edge density optimizations.

The performance of regular and irregular rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ MacKay codes are compared to the rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ benchmark and lowest-error-floor codes found in the research, respectively. The performance of the regular and irregular rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ LDPC codes with the best performances found in the course of this research are presented and discussed.

Finally, the error-floor performance of the lowest-error-floor irregular LDPC codes found are compared to those of LDPC codes obtained using other PEG algorithm modifications for lowering the error-floor of short-to-medium length irregular LDPC codes which have been published in the literature. These comparisons are made only with rate-½ LDPC codes that have similar (n, k) dimensions to those investigated in this research.

6.2 The Minimum Local Girth (Edge Skipping) (MLG (ES)) PEG Algorithm and Optimal low-correlated-Edge Density (OED) LDPC Codes

The PEG algorithm was modified in order to control the minimum local girth that can be connected in the LDPC codes constructed. The proposed modified PEG algorithm is summarized in Algorithm 6.1.

Algorithm 6.1: The minimum local girth (edge skipping) (MLG (ES)) progressive edge-growth (PEG) algorithm

$l_{g_{(min)}} = x$ (for a minimum local girth $g_{(\min)} = (2x + 2)$)
$create\_cycles = 0$
for $j = 0$ **to** $n-1$ **do**
begin
if (*cycles creation stage has been reached)* (see ❖)
$create\_cycles = 1$
for $k = 0$ **to** $d_{s_j} - 1$ **do**
begin
if $(k = 0)$
$E_{S_j}^0 \leftarrow$ edge (c_i, s_j), where $E_{S_j}^0$ is the first edge incident to s_j, and c_i is selected from the check nodes with the lowest check node degree under the current graph setting $E_{s_0} \cup E_{s_1} \cup \ldots \cup E_{s_{(j-1)}}$.
else
$skip = 0$
expand a subgraph from symbol node s_j up to depth l under the current graph setting such that the cardinality of $N_{S_j}^l$ stops increasing but is less than m, or $\dot{N}_{S_j}^l \neq \emptyset$ but $\dot{N}_{S_j}^{l+1} = \emptyset$.
if $((l < l_{g_{(min)}})$ **and** $(create\_cycles = 1))$
$skip = 1$
if $(skip = 0)$
$E_{S_j}^k \leftarrow$ edge (c_i, s_j), where $E_{S_j}^k$ is the $(k+1)$-th edge incident to s_j and c_i is a check node selected from the lowest degree check nodes in the set $\dot{N}_{S_j}^l$.
end
end

❖ this is determined based on the length n, rate, and symbol node degree distribution of the code under construction. Based on the DE degree distribution used in this section, the cycle creation stage is reached when $j = n/2$.

During graph constructions, all edge connections which would result in undesired local girths in the graph are skipped, i.e. forfeited. At many stages during a Tanner graph construction, establishing an

additional edge $E_{S_j}^k$ $(k \neq 0)$ of symbol node s_j on any of the candidate check nodes at a subgraph expansion depth of l, results in the creation of new cycles of length $(2l + 2)$ in the graph. In order to avoid connecting undesired local girths during an LDPC code construction, the two variables named $create\_cycles$ and $l_{g_{(min)}}$ in Algorithm 6.1 control the decision of whether to skip or allow the connection of the 2nd, 3rd, . , . , (d_{s_j})-th edges (i.e. $E_{S_j}^k \ \forall \ 1 \leq k \leq \ d_{s_j} - 1$) of symbol node s_j at subgraph expansion depth l. At the beginning of a Tanner graph construction, when no cycles are being connected, the value of variable $create\_cycles$ is 0. This prevents the algorithm from monitoring the depth of subgraph expansion l at which new edges are being connected, because at this stage of the graph construction subgraph expansion depths are non existent $(l = 0)$ or very shallow $(l = 1 \ or \ 2)$ and cycles are not yet being created. However, shortly before the graph construction stage where cycle creation commences, the variable $create\_cycles$ is assigned a value of 1 in order to prevent the about-to-be connected cycles from having girths which are less than the desired minimum local girth of the final Tanner graph. It was empirically determined, for the DE optimized degree distribution chosen for the investigations in the following sections, that the cycle creation stage commences shortly after half of the symbol nodes of the code have been connected into the graph, i.e. when $j > n/2$. The variable $l_{g_{(min)}}$ is set at the beginning of code construction as the minimum subgraph expansion depth at which creation of cycles are allowed. It determines the target minimum girth $g_{(min)}$ of the code to be constructed; the minimum girth $g_{(min)} = (2l_{g_{(min)}} + 2)$. Therefore, 4-cycles are prevented if $l_{g_{(min)}} = 2$, 4-cycles and 6-cycles are prevented if $l_{g_{(min)}} = 3$, etc.

The C programming language code for the MLG (ES) PEG algorithm for LDPC code construction is available in the Appendices (See Appendix H).

DE optimized degree distributions are used in standard PEG algorithms to construct ensembles of codes with predetermined minimum girths $g_{(min)}$. Graphs of codes constructed with almost all DE optimized distributions have $g_{(min)} > 4$, and consequently, all their edges have low correlation. However, these codes attain the desired girth and correlation properties at submaximal edge densities. In order to optimize the densities of short LDPC codes that were originally constructed using a DE

optimized degree distribution while maintaining the predetermined girth $g_{(min)}$, the maximum symbol node degree in the distribution $d_{s_{max}}$ is increased to extents that when the resultant degree distributions are used in the minimum local girth (edge skipping) (MLG (ES)) PEG algorithm (with $g_{(min)}$ set to the predetermined girth), edge connections are skipped during code constructions. More than one value of increased $d_{s_{max}}$ may result in denser LDPC codes. However, excessive $d_{s_{max}}$ increments decrease Tanner graph densities, increase the non-uniformity of check node degrees, and worsen decoding performance. The value of $d_{s_{max}}$ which results in the ensemble of denser LDPC codes with the best improved error-floor, i.e. the optimal value of $d_{s_{max}}$, can be subsequently determined via decoding performance simulations. LDPC codes constructed using the MLG (ES) PEG algorithm and the modified DE optimized degree distribution with the optimal value of $d_{s_{max}}$ as the input degree distribution are referred to as optimal low-correlated-edge density (OED) PEG codes.

6.3 Performance of OED Codes

The performance of the following five (5) types of irregular binary LDPC codes are compared in this section:

i) The benchmark codes and other LDPC codes constructed using the original/standard PEG algorithm of [Hu et al., 2001],

ii) OED PEG codes constructed using the proposed MLG (ES) PEG algorithm,

iii) ACE PEG codes constructed using the improved PEG (IPEG) construction of [Xiao and Banihashemi, 2004],

iv) OED ACE PEG codes, constructed using a combination of the PEG algorithms in ii) and iii), and

v) IEEE 802.16E (WiMAX) LDPC codes.

Only rate-½ LDPC codes with $(n, k) = (512, 256)$ and $(1024, 512)$ which were constructed using the different PEG algorithms are considered in the following comparisons. The performance of an irregular rate-½ $(n, k) = (512, 256)$ PEG code, and an irregular rate-½ $(n, k) = (1024, 512)$ PEG code are used as benchmarks of good error-floor performance. These codes were obtained from the

PhD work by C. J. Tjhai and have arguably the lowest error-floor performance previously published for irregular rate-½ LDPC codes of similar/identical length over the AWGN channel using standard BP/SPA iterative decoding [Tjhai, 2007].

The LDPC code weight evaluation algorithm by [Rosnes et al., 2012] was used to determine the stopping distance s_{min}, and the minimum codeword weight d_{min} of all codes simulated. Additionally, the SETA subgraph expansion based algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs [Abdu-Aguye et al., 2016(a)] was used to determine the girths, ACE spectra, and average ACE of all the LDPC codes simulated.

Groups of PEG codes constructed on the same *base-graph* were investigated. Codes constructed on the same base-graph have identical length and rate, and are constructed using the same random number seed. The only difference between codes constructed on the same base graph is the $d_{s_{max}}$ of the symbol node degree distributions used in their construction. Consequently, codes constructed on the same base-graph differ only at the last stages of graph construction; they have identical d_{min} and s_{min} but have different edge densities.

Results are presented for Monte Carlo simulations using binary phase-shift keying (BPSK) modulation over the AWGN channel. Codes were decoded using the standard BP/SPA algorithm at a maximum of 100 iterations. Identical noise patterns were used for each set of points at the same E_b/N_o on the error rate curves. The same numbers of blocks (frames) were used for all codes at each E_b/N_o ratio simulated.

6.3.1 Performance of Rate-½ $(n, k) = (512, 256)$ OED Codes

The investigations here are based on the DE optimized degree distribution of the (512, 256) benchmark code which is: $Q_3(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$. The benchmark code has $d_{min} = 14$ and $s_{min} = 16$. Using degree distribution $Q_3(x)$, all codes constructed by the standard PEG algorithm have a girth of 6. That is, for degree distribution $Q_3(x)$ the predetermined minimum local girth $g_{(\min)} = 6$. Consequently, when codes are constructed using

modifications to degree distribution $Q_3(x)$ as the input to the MLG (ES) PEG algorithm, the value of variable $l_{g_{(min)}}$ is set at 2 in order to maintain $g_{(\min)} = 6$ irrespective of the value of $d_{s_{max}}$. The following degree distributions were used to construct $(n, k) = (512, 256)$ Tanner graphs of different edge densities:

$$Q_1(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{12}$$
$$Q_2(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{13}$$
$$Q_3(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$$
$$Q_4(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{15}$$
$$Q_5(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{16}$$

In order to determine OED PEG codes based on degree distribution $Q_3(x)$, only the maximum symbol node degrees $d_{s_{max}}$, which was originally 14, was varied while the random number seed used for code constructions remained the same. This ensures that, for the same random number seed, the early graph construction stages are identical for all codes constructed using degree distributions $Q_1(x)$ to $Q_5(x)$. That is, the sequence of edge connections for the unaltered parts of degree distributions $Q_1(x)$ to $Q_5(x)$, i.e. $0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5$, are identical.

Identical Tanner graphs were obtained from both the standard PEG algorithm and the MLG (ES) PEG algorithm (with $g_{(\min)} = 6$) when degree distributions $Q_1(x)$, $Q_2(x)$, $Q_3(x)$, and $Q_4(x)$ with $d_{s_{max}}$ of 12, 13, 14 and 15, respectively, were applied to construct codes using the same random number seed. That is, no edges were skipped during code construction by the MLG (ES) PEG algorithm. All these codes had global girth $g = 6$ with 100% of local their girths $g_{s_j} = 6$ for all $0 \leq j \leq 512$, and all these codes had identical minimum weight metrics, i.e. $d_{min} = 19$ and $s_{min} = 16$. However, when degree distribution $Q_5(x)$ (with $d_{s_{max}} = 16$) was used, different codes were obtained from the two PEG algorithms. When $Q_5(x)$ was used in the standard PEG algorithm, the graph constructed had global girth $g = 4$ (88.87% of the symbol nodes had $g_{s_j} = 6$ and the remaining 11.13% had $g_{s_j} = 4$). In contrast, when $Q_5(x)$ was used in the MLG (ES) PEG algorithm (with $g_{(min)} = 6$), edge connections which would have resulted in 4-cycles were skipped and the graph constructed had a girth of 6

(100% of the symbol nodes had $g_{s_j} = 6$). Because these two codes were constructed on the same base graph as the $Q_1(x)$, $Q_2(x)$, $Q_3(x)$, and $Q_4(x)$ codes, they also had minimum weight metrics $d_{min} = 19$ and $s_{min} = 16$.

Due to skipped edge connections when degree distribution $Q_5(x)$ was used in the MLG (ES) PEG algorithm (as the *input degree distribution*), the degree distribution of the Tanner graph constructed (the *output degree distribution*) was different from the input degree distribution to the algorithm. The degree distribution of the constructed graph, i.e. the output degree distribution, depends on the number of edge connections skipped and the random number seed used during code construction. For the selected random number seed, a total of 25 edges from symbol nodes with degree $d_{s_{max}} = 16$ in $Q_5(x)$ were skipped. The output degree distribution is given by,

$$Q_6(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.00195x^{13} + 0.01563x^{14} + 0.01172x^{15} + 0.07422x^{16}.$$

Comparing output degree distribution $Q_6(x)$ to the input degree distribution $Q_5(x)$, it can be seen that there is a reduced fraction of symbol nodes of degree 16, and symbol nodes with degrees of 13, 14, and 15 appeared in $Q_6(x)$. These observed differences between the two degree distributions are due to skipped edge connections.

The effects of implementing the PEG algorithm modification by [Xiao and Banihashemi, 2004] to improve the ACE in standard PEG codes with DE optimized degree distributions and the OED PEG codes which are subsequently derived from them were investigated. An ensemble of 100 rate-½ $(512, 256)$ ACE PEG codes with DE optimized degree distribution $Q_3(x)$ was constructed (see Appendix D for the C programming language code for the ACE PEG algorithm implemented in this research). Within this ensemble, the ACE PEG code with the lowest-error-floor and the random number seed used in its construction was identified. The PEG algorithm modification by Xiao and Banihashemi for improved ACE (IPEG/ACE PEG) was implemented in the MLG (ES) PEG algorithm in order to construct OED ACE PEG codes. The C programming language code for the combined ACE PEG and MLG (ES) PEG algorithms used for constructing OED ACE PEG codes is available in

the Appendices (See Appendix I). Using degree distribution $Q_5(x)$ and the random number seed used to construct the ACE PEG code with the lowest-error-floor performance as input to the MLG (ES) PEG algorithm, an OED ACE PEG code with the following symbol node degree distribution was constructed.

$$Q_7(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.00391x^{13} + 0.01563x^{14} + 0.00977x^{15} + 0.07422x^{16}$$

The ACE PEG code and the OED ACE PEG code subsequently derived using the same random number seed were constructed on the same base graph and have $d_{min} = s_{min} = 18$.

Table 6.1 shows: a) the symbol node degree distributions used as input to respective LDPC code construction algorithms (*input degree distribution*), b) the degree distributions of the constructed codes (*output degree distribution*), c) code densities in terms of the *number of edges, E*, d) the *maximum symbol node degrees* $d_{s_{max}}$ in codes, e) the global *girth*, and f) the average ACE of the following rate-½ $(n, k) = (512, 256)$ and $(576, 288)$ LDPC codes which were compared in decoding performance simulations.

i) the $(n, k) = (512, 256)$ standard PEG codes constructed using degree distributions $Q_1(x)$ to $Q_5(x)$, including the benchmark code with degree distribution $Q_3(x)$,

ii) the $(n, k) = (512, 256)$ OED PEG code with degree distribution $Q_6(x)$,

iii) the $(n, k) = (512, 256)$ ACE PEG code with degree distribution $Q_3(x)$,

iv) the $(n, k) = (512, 256)$ OED ACE PEG code with degree distribution $Q_7(x)$, and

v) an $(n, k) = (576, 288)$ IEEE 802.16E (WiMAX) code with degree distribution: $Q_8(x) = 0.45833x^2 + 0.33333x^3 + 0.20833x^6$.

From Table 6.1, the three densest codes, i.e. the codes with the highest number of edges E, are those with output degree distributions $Q_5(x)$, $Q_6(x)$ and $Q_7(x)$, which all have $d_{s_{max}} = 16$. The densest codes with girths of 6 are those with degree distribution $Q_6(x)$ (the OED PEG code) and degree distribution $Q_7(x)$ (the OED ACE PEG code).

The standard PEG code with 2147 edges is denser than the OED PEG and the OED ACE PEG codes with 2122 and 2120 edges respectively. In contrast, the OED PEG code and the OED ACE PEG codes have girths of 6, which is higher than the girth of 4 for the standard PEG code. Consequently, edges in the OED PEG and OED ACE PEG codes graph have lower correlation than edges in the standard PEG code graph with degree distribution $Q_5(x)$. Although degree distribution $Q_5(x)$ was the input degree distribution to the respective PEG algorithms used for constructing these three codes, there is a difference in edge density between them as a result of edge connections which were skipped in order to avoid creating 4-cycles in the OED PEG and the OED ACE PEG codes.

Table 6.1: Input and output degree distributions, maximum symbol node degrees $d_{s_{(max)}}$, global girths, edge densities (number of edges), and average ACEs of rate-½ $(n,\ k) = (512,\ 256)$ and $(576,\ 288)$ LDPC codes

Input degree distribution (Design)	Output degree distribution **(Actual Code)**	Maximum symbol node degree ($d_{s_{(max)}}$)	Global girth (g)	Number of edges (E)	Average ACE
rate-½ $(n,\ k) = (512,\ 256)$ standard PEG codes					
$Q_1(x)$	$\boldsymbol{Q_1(x)}$	12	6	1935	11.701
$Q_2(x)$	$\boldsymbol{Q_2(x)}$	13	6	1998	12.719
$Q_3(x)$	$\boldsymbol{Q_3(x)}$	14	6	2041	13.518
$Q_4(x)$	$\boldsymbol{Q_4(x)}$	15	6	2094	14.398
$Q_5(x)$	$\boldsymbol{Q_5(x)}$	16	4	2147	15.307
rate-½ $(n,\ k) = (512,\ 256)$ benchmark PEG code					
$Q_3(x)$	$\boldsymbol{Q_3(x)}$	14	6	2041	13.607
rate-½ $(n,\ k) = (512,\ 256)$ OED PEG code					
$Q_5(x)$	$\boldsymbol{Q_6(x)}$	16	6	2122	14.504
rate-½ $(n,\ k) = (512,\ 256)$ ACE PEG code					
$Q_3(x)$	$\boldsymbol{Q_3(x)}$	14	6	2041	15.846
rate-½ $(n,\ k) = (512,\ 256)$ OED ACE PEG code					
$Q_5(x)$	$\boldsymbol{Q_7(x)}$	16	6	2120	15.701
rate-½ $(n,\ k) = (576,\ 288)$ IEEE 802.16E (WiMAX) code					
$Q_8(x)$	$\boldsymbol{Q_8(x)}$	6	6	1824	5.708

Figure 6.1 shows the FER performance simulations results for the five codes constructed using degree distributions $Q_1(x)$ to $Q_5(x)$ in the standard PEG algorithm. These codes were constructed on the same base-graph and differ only in the value of $d_{s_{max}}$ in their degree distributions; $d_{s_{max}}$ was varied from 12 to 16. In Figure 6.1, the performance of these five standard PEG codes are compared to that of the benchmark code of identical dimension, i.e. rate-½ $(n,\ k) = (512,\ 256)$.

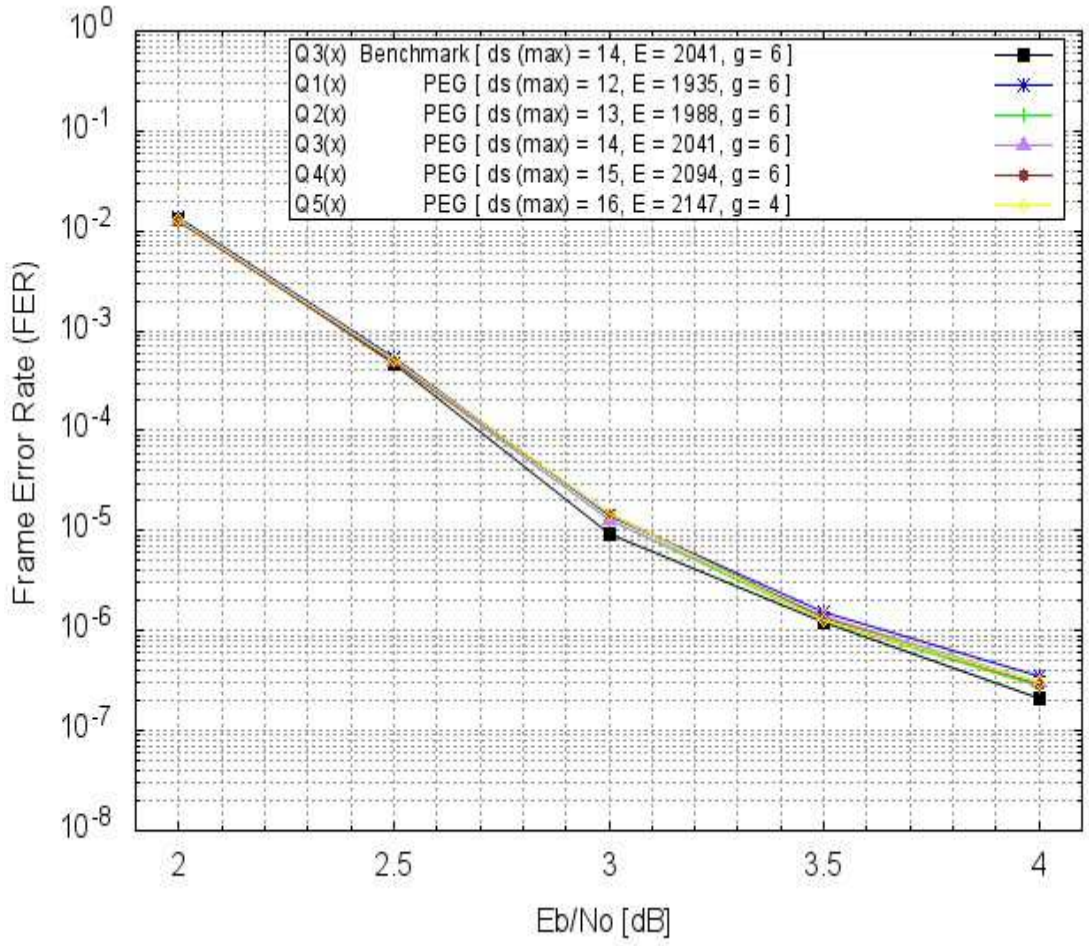

Figure 6.1: FER for five rate-½ $(n,\ k) = (512,\ 256)$ standard PEG codes with increasing edge densities, and the $(512,\ 256)$ benchmark PEG code

Despite their edge density differences, all five standard PEG codes in Figure 6.1 had similar performance and none of them performed better than the benchmark code.

Figure 6.2 shows the FER performance simulation results for the following two irregular rate-½ $(n,\ k) = (512,\ 256)$ LDPC codes:

i) the OED PEG code with degree distribution $Q_6(x)$, constructed using $Q_5(x)$ as input to the MLG (ES) PEG algorithm, and

ii) the benchmark code.

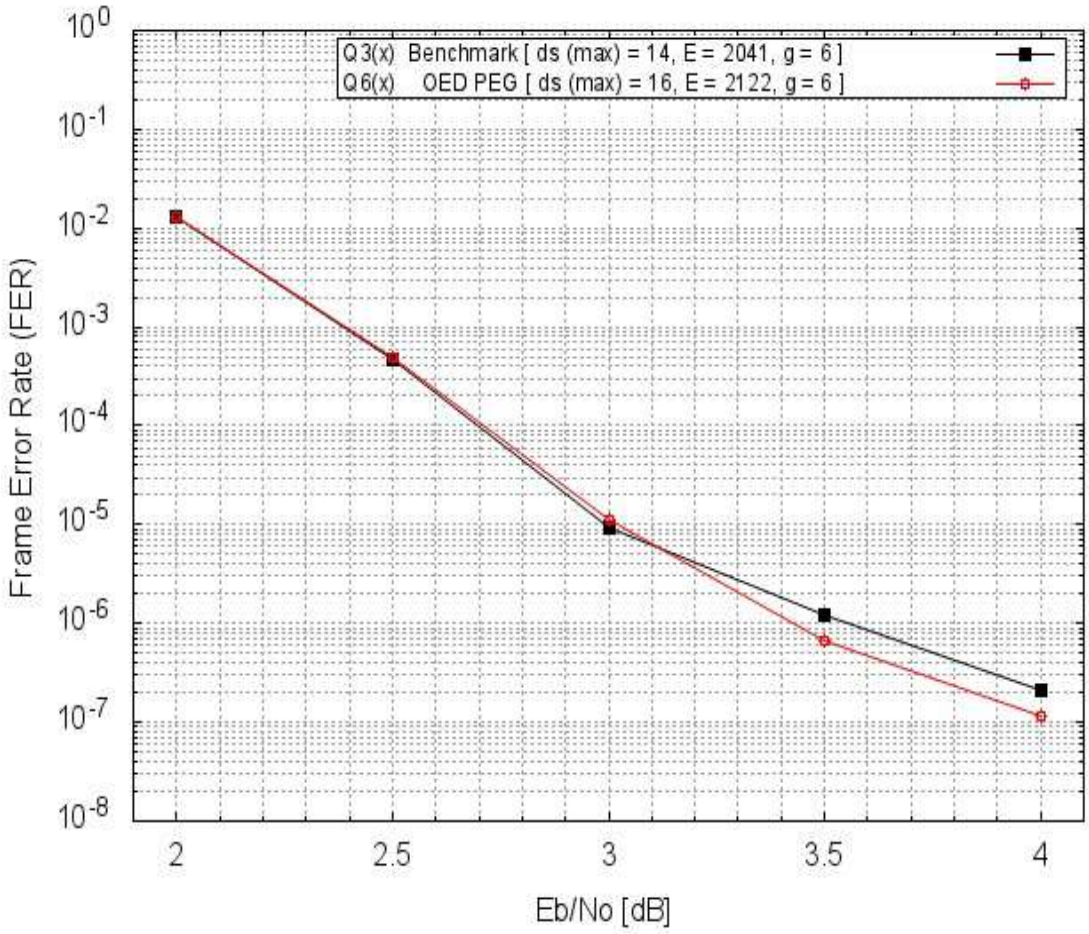

Figure 6.2: FER for the rate-½ $(n,\ k) = (512,\ 256)$ OED PEG and benchmark codes

The OED PEG code outperformed the benchmark code in the error-floor region with approximately half the FER at an E_b/N_o of 4.0 dB. Extensive performance simulations carried out on rate-½ $(n,\ k) = (512,\ 256)$ codes show that OED PEG codes have consistently lower error-floors than corresponding DE optimized standard PEG codes constructed on the same base graph. This is significant because, similar to the improved PEG (IPEG) construction [Xiao and Banihashemi, 2004], it offers an original technique that may be used to improve the performance of LDPC codes constructed using the PEG algorithm and DE optimized symbol node degree distributions.

Figure 6.3 shows the FER simulation results for the following five irregular rate-½ LDPC codes:

i) the $(n,\ k) = (512,\ 256)$ benchmark code,

ii) the $(n,\ k) = (512,\ 256)$ OED PEG code with degree distribution $Q_6(x)$,

iii) the $(n,\ k) = (512,\ 256)$ ACE PEG code with the same degree distribution as the benchmark code in **i)**, i.e. $Q_3(x)$,

iv) the $(n,\ k) = (512,\ 256)$ OED ACE PEG code with degree distribution $Q_7(x)$, and

v) the $(n,\ k) = (576,\ 288)$ IEEE 802.16E WiMAX code with distribution $Q_8(x)$, $d_{min} = 13$ and $s_{min} = 18$.

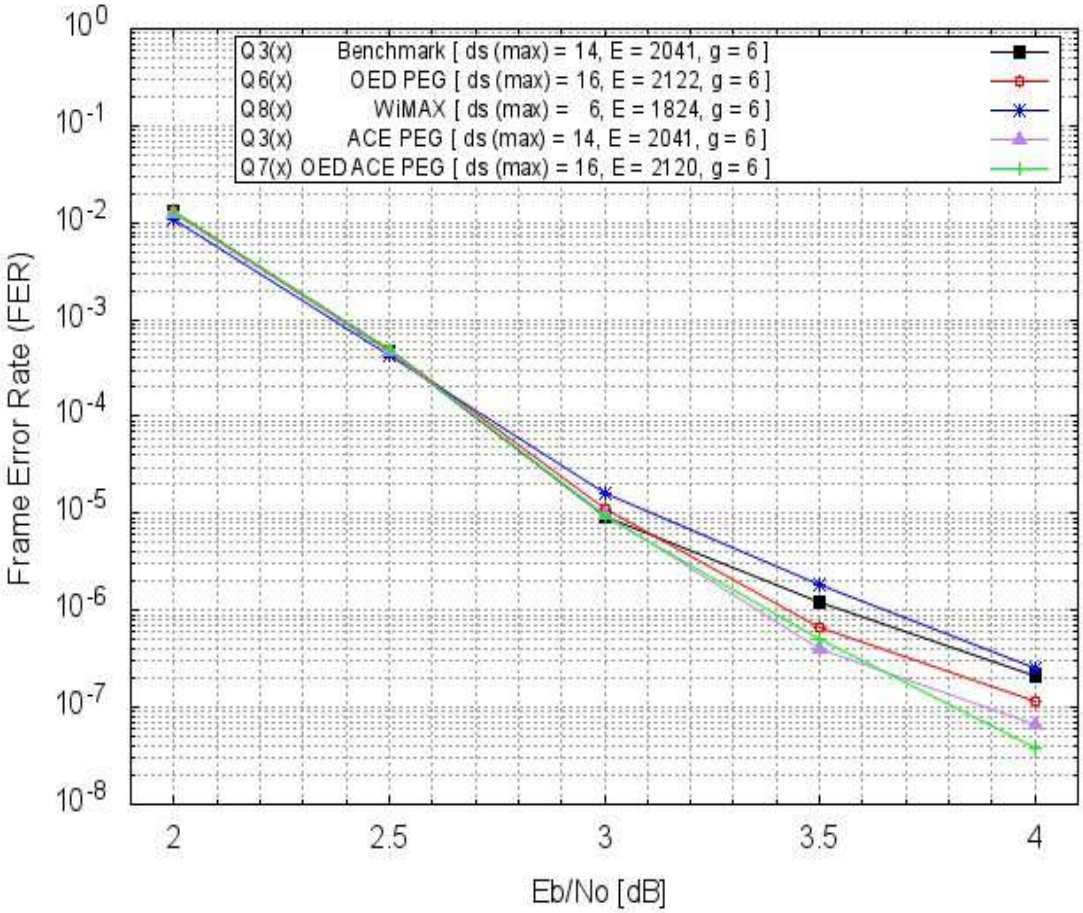

Figure 6.3: FER curves for, i) the $(512,\ 256)$ benchmark code, ii) the $(512,\ 256)$ OED PEG code, iii) the (576, 288) WiMAX code, iv) the $(512,\ 256)$ ACE PEG code, and v) the $(512,\ 256)$ OED ACE PEG code

It can be seen from Figure 6.3 that all the PEG constructed LDPC codes have lower error-floor than the IEEE 802.16E (WiMAX) code. The relatively poor performance of the WiMAX code may be attributed to the fact that it has the degree distribution with the smallest $d_{s_{max}}$, the lowest edge density

(number of edges, E), and the lowest average ACE of all the codes compared; these metrics are shown in Table 6.1. It is interesting to note that the simulated performance of the rate-½ (576, 288) WiMAX code in Figure 6.3 is identical to that of the best WiMAX code of the same rate and length published in [Bocharova et al., 2016].

The OED PEG, ACE PEG, and OED ACE PEG codes all outperformed the benchmark code in the error-floor region. As a result of the improved ACE in their Tanner graphs, the ACE PEG and OED ACE PEG codes have lower error-floor than the OED PEG code. However, the OED ACE PEG code had the best performance in the error-floor region. Extensive performance simulations carried out on rate-½ $(n,\ k) = (512,\ 256)$ codes show that most OED ACE PEG codes have lower error-floor than corresponding DE optimized ACE PEG codes constructed on the same base graph. A similar observation was made for OED PEG codes compared to DE optimized standard PEG codes constructed on the same base graph. The LDPC matrix configuration file for the rate-½ $(n,\ k) = (512,\ 256)$ OED ACE PEG code with the lowest error-floor, as presented in Figure 6.3, is available in the Appendices (See Appendix J).

When the technique used to realise OED codes is applied, the following consistent observations were made using the LDPC code weight evaluation algorithm, and the SETA subgraph expansion based algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs:

i) OED ACE PEG codes and DE optimized ACE PEG codes constructed on the same base graph have identical d_{min} and s_{min}. Similarly, OED PEG codes and DE optimized standard PEG codes constructed on the same base graph have identical d_{min} and s_{min}.

ii) An OED ACE PEG code has an identical local girth distribution, a significantly increased range of local ACE values, and slightly smaller average ACE compared to the DE optimized ACE PEG code constructed on the same base graph. Similarly, an OED PEG code has an identical local girth distribution, a significantly increased range of local ACE values, and slightly reduced average ACE compared to the DE optimized standard PEG code constructed on the same base graph.

Observations of decoding failure events indicates that the improved error-floor performance of OED codes are due to a slight reduction in the number of small trapping sets. Higher Tanner graph edge densities and increased local ACE distributions are likely to be responsible for the reduction in the number of harmful trapping sets in OED PEG and OED ACE PEG codes.

6.3.2 Performance of Rate-½ $(n, k) = (1024, 512)$ OED Codes

The rate-½ $(n, k) = (1024, 512)$ benchmark code was constructed using DE optimized degree distribution $Q_9(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}$. This code has $d_{min} = 16$ and $s_{min} = 20$. The predetermined girth $g_{(min)}$ of codes constructed using $Q_9(x)$ is 6. Attempts to derive $(n, k) = (1024, 512)$ OED PEG codes with improved error-floors based on modifications to degree distribution $Q_9(x)$ were unsuccessful. In experiments to find alternative degree distributions, it was discovered that using DE optimized degree distribution $Q_3(x)$, which was used to construct the $(n, k) = (512, 256)$ benchmark code, to construct rate-½ $(n, k) = (1024, 512)$ LDPC codes in the standard PEG algorithm result in codes with error-floors which are very similar to that of the rate-½ $(n, k) = (1024, 512)$ benchmark code. However, with degree distribution $Q_3(x)$ there is a trade-off in the low SNR performance, and code performance in waterfall region becomes worse. Figure 6.4 shows the FER performance simulation results obtained in this comparison.

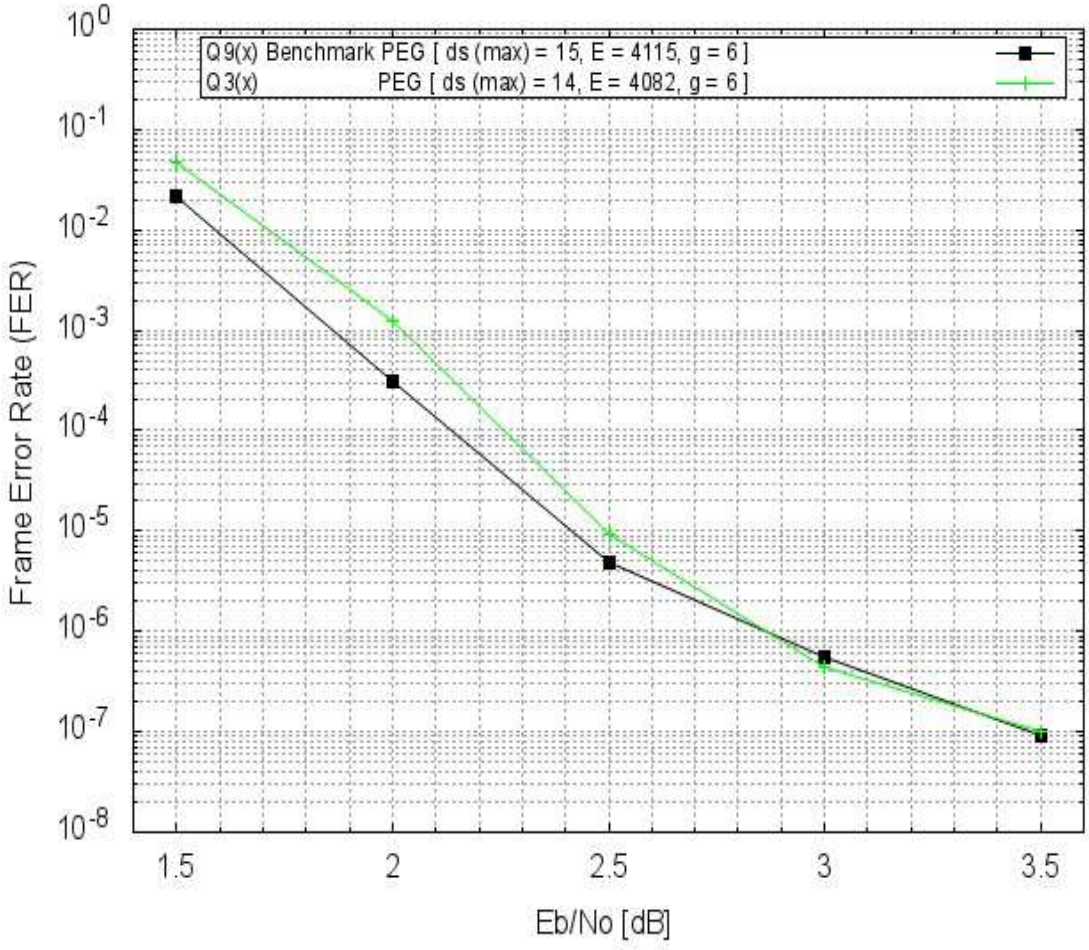

Figure 6.4: FER curves for, i) the $(n,\ k) = (1024,\ 512)$ benchmark code with degree distribution $Q_9(x)$, and an $(n,\ k) = (1024,\ 512)$ standard PEG code constructed with degree distribution $Q_3(x)$

As a result of the error-floor performance of rate-½ $(n,\ k) = (1024,\ 512)$ standard PEG codes constructed with DE optimized degree distribution $Q_3(x)$, as exemplified in Figure 6.4, subsequent investigations carried out to improve the error-floor performance of rate-½ $(n,\ k) = (1024,\ 512)$ LDPC codes using the OED PEG codes derivation technique are based on DE optimized degree distribution $Q_3(x)$. In order to construct rate-½ $(n,\ k) = (1024,\ 512)$ OED PEG codes based on $Q_3(x)$, a minimum $d_{s_{max}}$ of 20 is required for edge connections to be skipped when using the MLG (ES) PEG algorithm (with $g_{(min)} = 6$). The resultant codes had higher error-floor than the benchmark code. That is, all attempts to derive $(n,\ k) = (1024,\ 512)$ OED PEG codes with improved error-floors based on modifications to degree distribution $Q_3(x)$ were also unsuccessful.

However, codes constructed with degree distributions obtained when $Q_3(x)$ is modified so that $d_{s_{max}} = 16$ to 19 in either the standard PEG algorithm or the MLG (ES) PEG algorithm (with $g_{(min)} = 6$) had lower error-floor than the benchmark code. These codes maintained the predetermined girth of degree distribution $Q_3(x)$ without edge connections being skipped during their construction using the MLG (ES) PEG algorithm, and therefore cannot technically be described as OED PEG codes . The code with the lowest error-floor found within this range had $d_{s_{max}} = 19$, with degree distribution: $Q_{10}(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{19}$.

Investigations to compare the performance of the rate-½ $(n, k) = (1024, 512)$ benchmark code to the performance of LDPC codes of the same length and rate which are constructed using the improved PEG construction (IPEG) by [Xiao and Banihashemi, 2004] were undertaken. Three ensembles of 100 rate-½ $(n, k) = (1024, 512)$ ACE PEG codes each were constructed using degree distributions $Q_4(x)$, $Q_5(x)$ and DE optimized degree distribution $Q_3(x)$. Performance simulation results show that ACE PEG codes from all three code ensembles significantly outperform both the benchmark code and the improved standard PEG code with degree distribution $Q_{10}(x)$ in the error-floor region. The ACE PEG code with the lowest error-floor found had degree distribution $Q_4(x)$.

Similar to the case for OED PEG codes, in order to construct rate-½ $(n, k) = (1024, 512)$ OED ACE PEG codes based on DE optimized degree distribution $Q_3(x)$, a minimum $d_{s_{max}}$ of 20 is required for edge connections to be skipped when using a combination of the IPEG construction algorithm [Xiao and Banihashemi, 2004] and the MLG (ES) PEG algorithm (with $g_{(min)} = 6$). Performance simulation results show that the resultant rate-½ $(n, k) = (1024, 512)$ OED ACE PEG codes have lower error-floor than the rate-½ $(n, k) = (1024, 512)$ benchmark code. However, within the range of results obtained, OED ACE PEG codes did not perform any better than ACE PEG codes constructed on the same base graph. That is, results show that OED ACE PEG code have either identical or worse performance than ACE PEG codes constructed on the same base graph. Additionally, increasing $d_{s_{max}}$ from 14 in DE optimized degree distribution $Q_3(x)$ to 20 in order to

obtain OED PEG and OED ACE PEG codes results in significantly higher Tanner graph edge densities which also significantly increases decoding times using BP/SPA decoding algorithms.

Table 6.2 shows: a) the symbol node degree distributions, b) code densities in terms of the *number of edges, E*, c) the *maximum symbol node degrees* $d_{s_{max}}$ of codes, d) the global *girth*, and e) the average ACE of the following rate-½ $(n,\ k) = (1024,\ 512)$ and $(1056,\ 528)$ LDPC codes compared using decoding performance simulations.

i) an $(n,\ k) = (1024,\ 512)$ standard PEG code with degree distribution $Q_3(x)$,

ii) the $(n,\ k) = (1024,\ 512)$ benchmark code with degree distribution $Q_9(x)$,

iii) the $(n,\ k) = (1024,\ 512)$ standard PEG code with improved degree distribution $Q_{10}(x)$,

iv) the $(n,\ k) = (1024,\ 512)$ ACE PEG code with degree distribution $Q_4(x)$, and

v) an $(n,\ k) = (1056,\ 528)$ IEEE 802.16E (WiMAX) code with degree distribution: $Q_8(x) = 0.45833x^2 + 0.33333x^3 + 0.20833x^6$.

Table 6.2: Symbol node degree distributions, maximum symbol node degrees $d_{s_{(max)}}$, global girths, edge densities (number of edges), and average ACEs of rate-½ $(n,\ k) = (1024,\ 512)$ and $(1056,\ 528)$ LDPC codes

Symbol node degree distribution	Maximum symbol node degree ($d_{s_{(max)}}$)	Global girth (g)	Number of edges (E)	Average ACE
rate-½ $(n,\ k)$ = (1024, 512) standard PEG code				
$Q_3(x)$	14	6	4082	15.779
rate-½ $(n,\ k)$ = (1024, 512) benchmark PEG code				
$Q_9(x)$	15	6	4115	16.963
rate-½ $(n,\ k)$ = (1024, 512) improved standard PEG code				
$Q_{10}(x)$	19	6	4612	19.218
rate-½ $(n,\ k)$ = (1024, 512) ACE PEG code				
$Q_4(x)$	15	6	4188	23.072
$Q_9(x)$	15	6	4115	23.813
rate-½ $(n,\ k)$ = (1056, 528) IEEE 802.16E (WiMAX) code				
$Q_8(x)$	6	6	3344	6.125

Figure 6.5 shows the performance simulation results for the following irregular rate-½ LDPC codes:

i) the $(n, k) = (1024, 512)$ benchmark code with degree distribution $Q_9(x)$, which has $d_{min} = 16$ and $s_{min} = 20$,

ii) the $(n, k) = (1024, 512)$ standard PEG code constructed using improved degree distribution $Q_{10}(x)$, which has $d_{min} = 22$ and $s_{min} = 23$,

iii) the $(n, k) = (1024, 512)$ ACE PEG code constructed by the IPEG algorithm using degree distribution $Q_4(x)$, which has $d_{min} = 22$ and $s_{min} = 22$, and

iv) an $(n, k) = (1056, 528)$ WiMAX code constructed using degree distribution $Q_8(x) = 0.45833x^2 + 0.33333x^3 + 0.20833x^6$, which has $d_{min} = 21$ and $s_{min} = 19$.

The error-floor performance of the rate-½ $(n, k) = (1056, 528)$ IEEE 802.16E WiMAX code was significantly worse than those for all the rate-½ $(n, k) = (1024, 512)$ PEG codes. The ACE PEG code with degree distribution $Q_4(x)$ had the best performance in the error-floor region. The relatively poor performance of the WiMAX code is attributed to the fact that it has the degree distribution with the smallest $d_{s_{max}}$, the lowest edge density (number of edges, E), and the lowest average ACE of all the codes compared, as shown in Table 6.2. Therefore, for rate-½ $(n, k) = (1024, 512)$ LDPC codes constructed with degree distribution $Q_4(x)$, the ACE PEG codes have superior error-floor performance compared to OED ACE PEG codes.

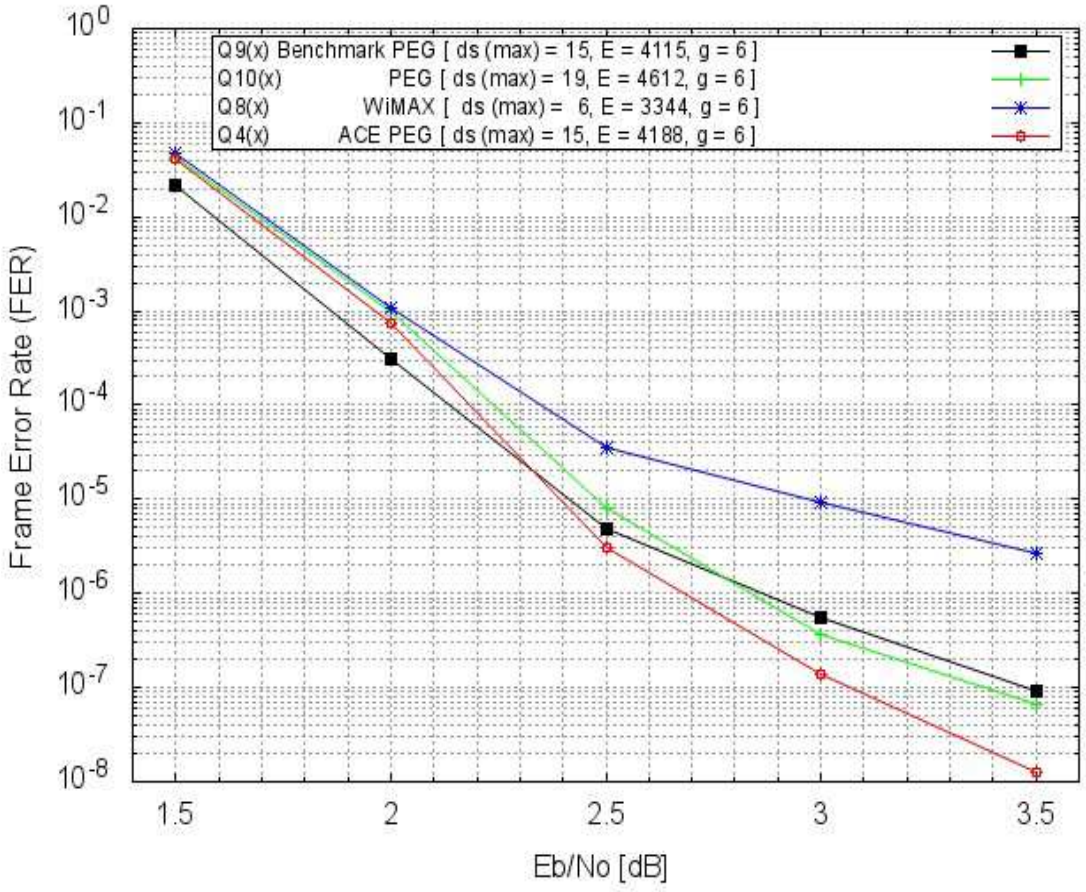

Figure 6.5: FER curves for the benchmark, standard PEG, and ACE PEG rate-½ $(1024, 512)$ codes; and the WiMAX rate-½ $(1056, 528)$ code

Figure 6.6 compares the performance of the rate-½ $(n, k) = (1024, 512)$ benchmark code with degree distribution $Q_9(x)$ to the performance of the lowest-error-floor rate-½ $(n, k) = (1024, 512)$ ACE PEG codes which were constructed with,

i) degree distribution $Q_9(x)$ (the same degree distribution as the benchmark code) which has $d_{min} = 14$ and $s_{min} = 19$; and

ii) degree distribution $Q_4(x)$, which was obtained by increasing $d_{s_{max}}$ in DE optimized degree distribution $Q_3(x)$ from 14 to 15, which has $d_{min} = 22$ and $s_{min} = 22$.

It may be recalled that degree distribution $Q_3(x)$ was selected over $Q_9(x)$ for investigations to find rate-½ $(n, k) = (1024, 512)$ OED codes with lower error-floor. The choice of $Q_3(x)$ over $Q_9(x)$ was due to observations made about the error-floor performances of LDPC codes constructed using these

two degree distributions. As exemplified in Figure 6.4, it is often the case for irregular LDPC codes that degree distributions which result in superior performance in the waterfall region result in inferior performance in the error-floor region, and degree distributions which result in superior performance in the error-floor region result in inferior performance in the waterfall region.

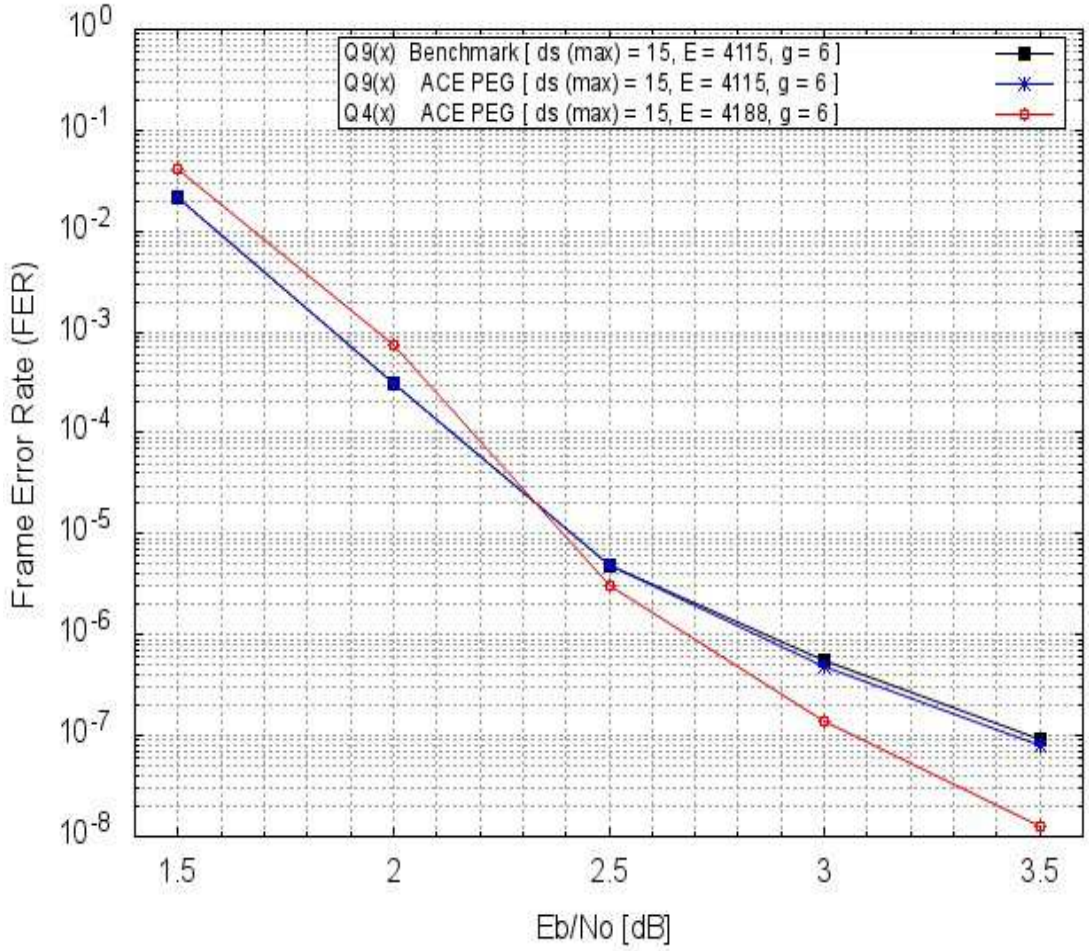

Figure 6.6: FER curves for rate-½ $(n,\ k) = (1024,\ 512)$ codes, i) the benchmark code with distribution $Q_9(x)$, ii) the ACE PEG code with distribution $Q_9(x)$, and iii) the ACE PEG code with distribution $Q_4(x)$

The FER decoding performance results for the rate-½ $(n,\ k) = (1024,\ 512)$ codes in Figure 6.6 serve as a further justification of the choice of degree distribution $Q_3(x)$ over $Q_9(x)$. The performance of the two lowest-error-floor codes with degree distribution $Q_9(x)$, i.e. the benchmark code and the ACE PEG code, are very similar; the ACE PEG code performs only slightly better that the benchmark code in the error-floor region. This is in spite of the fact that the ACE PEG code has a significantly

improved average ACE at 23.813 compared to the benchmark code which has an average ACE of 16.963, as shown in Table 6.2. However, as exemplified in Figure 6.3, the lowest-error-floor ACE PEG code with degree distribution $Q_3(x)$ has a significantly improved error-floor performance compared to the performance of the standard PEG code (the Benchmark code) with the same degree distribution. Therefore, it may be inferred that the extent to which the error-floor of short-to-medium length LDPC codes can be lowered through Tanner graph ACE improvements depends on the symbol node degree distributions being used.

The proposed OED code derivation technique is not suitable for constructing irregular rate-½ codes with dimensions higher than $(n, k) = (1024, 512)$. At these code lengths, in order for edge connections to be skipped by the MLG (ES) PEG algorithm, the $d_{s_{max}}$ in DE optimized degree distributions must be increased to such extents that the edge densities of codes constructed result in significantly longer decoding times without commensurate improvement in decoding performance.

6.4 Performance and Features of LDPC Codes Constructed using the Generalized ACE Constrained PEG (G-ACE PEG) Algorithm

Ensembles of rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ LDPC codes were constructed using the generalized ACE constrained PEG (G-ACE PEG) algorithm proposed by [Vukobratović and Šenk, 2008]. The G-ACE PEG algorithm constructs LDPC codes with very good girth and extremal ACE properties. The d_{min} and s_{min} distributions in ensembles of G-ACE PEG codes are very similar to those of ACE PEG codes as presented in Figure 5.3 and Figure 5.4, respectively. A comparison of the girth and ACE metrics in ensembles of rate-½ $(n, k) = (512, 256)$ G-ACE PEG codes to those of standard PEG codes [Hu et al., 2001] and ACE PEG [Xiao and Banihashemi, 2004] are discussed in Section 4.4.4 and summarized in Table 4.6 and Table 4.7, respectively. It is clear that G-ACE PEG codes have superior girth and ACE metric compared to LDPC codes constructed using the other PEG algorithms.

In order to increase the values of the ACE spectrum components, the sequence of check node selection criteria implemented by the G-ACE PEG algorithm, as outlined in Section 3.3.5, gives a low priority

to selecting lowest degree check nodes among the candidate check nodes for connecting edges during Tanner graph construction. Consequently, G-ACE PEG codes parity-check matrices are not as check-node regular as LDPC codes constructed using other PEG algorithms.

As an example, the check node degree distributions for five randomly selected rate-½ $(n, k) = (512, 256)$ ACE PEG codes constructed using DE optimized symbol node degree distribution $Q_3(x)$ are as follows,

$$\Phi_1(x) = 0.07031x^7 + 0.88672x^8 + 0.04297x^9$$
$$\Phi_2(x) = 0.05469x^7 + 0.91797x^8 + 0.02734x^9$$
$$\Phi_3(x) = 0.06520x^7 + 0.90234x^8 + 0.03516x^9$$
$$\Phi_4(x) = 0.06641x^7 + 0.89453x^8 + 0.03906x^9$$
$$\Phi_5(x) = 0.05469x^7 + 0.91797x^8 + 0.02734x^9$$

In contrast, the check node degree distributions for five randomly selected rate-½ $(n, k) = (512, 256)$ G-ACE PEG codes constructed using the same DE optimized symbol node degree distribution $Q_3(x)$ are as follows,

$$\Phi_6(x) = 0.28125x^7 + 0.49219x^8 + 0.19922x^9 + 0.02734x^{10}$$
$$\Phi_7(x) = 0.27344x^7 + 0.50000x^8 + 0.20703x^9 + 0.01953x^{10}$$
$$\Phi_8(x) = 0.25391x^7 + 0.53906x^8 + 0.19141x^9 + 0.01172x^{10} + 0.00391x^{11}$$
$$\Phi_9(x) = 0.26563x^7 + 0.51953x^8 + 0.19141x^9 + 0.02344x^{10}$$
$$\Phi_{10}(x) = 0.25000x^7 + 0.55079x^8 + 0.17578x^9 + 0.02344x^{10}$$

Similarly, the check node degree distributions for five randomly selected rate-½ $(n, k) = (1024, 512)$ ACE PEG codes constructed using symbol node degree distribution $Q_4(x)$ were found to be identical as follows,

$$\Phi_{11}(x) = 0.82031x^8 + 0.17969x^9$$
$$\Phi_{12}(x) = 0.82031x^8 + 0.17969x^9$$
$$\Phi_{13}(x) = 0.82031x^8 + 0.17969x^9$$
$$\Phi_{14}(x) = 0.82031x^8 + 0.17969x^9$$
$$\Phi_{15}(x) = 0.82031x^8 + 0.17969x^9$$

In contrast, the check node degree distributions for five randomly selected rate-½ $(n, k) = (1024, 512)$ G-ACE PEG codes constructed using the same symbol node degree distribution $Q_4(x)$ are as follows,

$$\Phi_{16}(x) = 0.12695x^7 + 0.61133x^8 + 0.21875x^9 + 0.04102x^{10} + 0.00195x^{11}$$
$$\Phi_{17}(x) = 0.13477x^7 + 0.58008x^8 + 0.25586x^9 + 0.02930x^{10}$$
$$\Phi_{18}(x) = 0.14453x^7 + 0.56250x^8 + 0.26172x^9 + 0.03125x^{10}$$
$$\Phi_{19}(x) = 0.16797x^7 + 0.52539x^8 + 0.26563x^9 + 0.04102x^{10}$$
$$\Phi_{20}(x) = 0.14258x^7 + 0.57422x^8 + 0.24414x^9 + 0.03906x^{10}$$

It can be seen from these check-node degree distributions that rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ ACE PEG codes have concentrated check node degrees. The $(512, 256)$ ACE PEG codes have check node degrees of only 7, 8 and 9 with more than 88% of the check nodes having a degree of 8, and the $(1024, 512)$ ACE PEG codes have identical symbol node degree distributions with symbol node degrees of only 8 and 9 with approximately 82% of the check nodes having the check node degree of 8. In contrast, rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ G-ACE PEG codes have a wider spread in the values of the check node degrees. The $(512, 256)$ G-ACE PEG codes have check node degrees of 7, 8, 9, 10 and 11 with only approximately 50% of the check nodes having the concentrated check node degree of 8. Similarly, the $(1024, 512)$ G-ACE PEG codes have check node degrees of 7, 8, 9, 10 and 11 with only approximately 52% to 61% of the check nodes having the concentrated check node degree of 8.

Extensive simulation experiments on rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ G-ACE PEG codes show that they have very similar performance to ACE PEG codes. However, the rate-½ $(512, 256)$ and $(1024, 512)$ G-ACE PEG codes with the lowest error-floor found within the range of codes simulated did not perform as well as the lowest-error-floor rate-½ $(512, 256)$ OED ACE PEG found in Section 6.3.1 and the lowest-error-floor rate-½ $(1024, 512)$ ACE PEG codes found in Section 6.3.2, respectively. That is, within the wide range of code performance simulation experiments carried out in this research, the lowest-error-floor rate-½ $(n, k) = (512, 256)$ codes found were not

G-ACE PEG codes but were OED ACE PEG codes, and the lowest-error-floor rate-½ $(n,\ k) = (1024,\ 512)$ codes found were not G-ACE PEG codes but were ACE PEG codes.

Figure 6.7 shows the FER simulation results for the following rate-½ $(n,\ k) = (512,\ 256)$ codes:

i) the lowest-error-floor G-ACE PEG code found, which has DE optimized degree distribution $Q_3(x)$, $d_{min} = 13$, $s_{min} = 18$, and

ii) the lowest-error-floor OED ACE PEG code found, which has output degree distribution $Q_7(x)$, and $d_{min} = s_{min} = 18$.

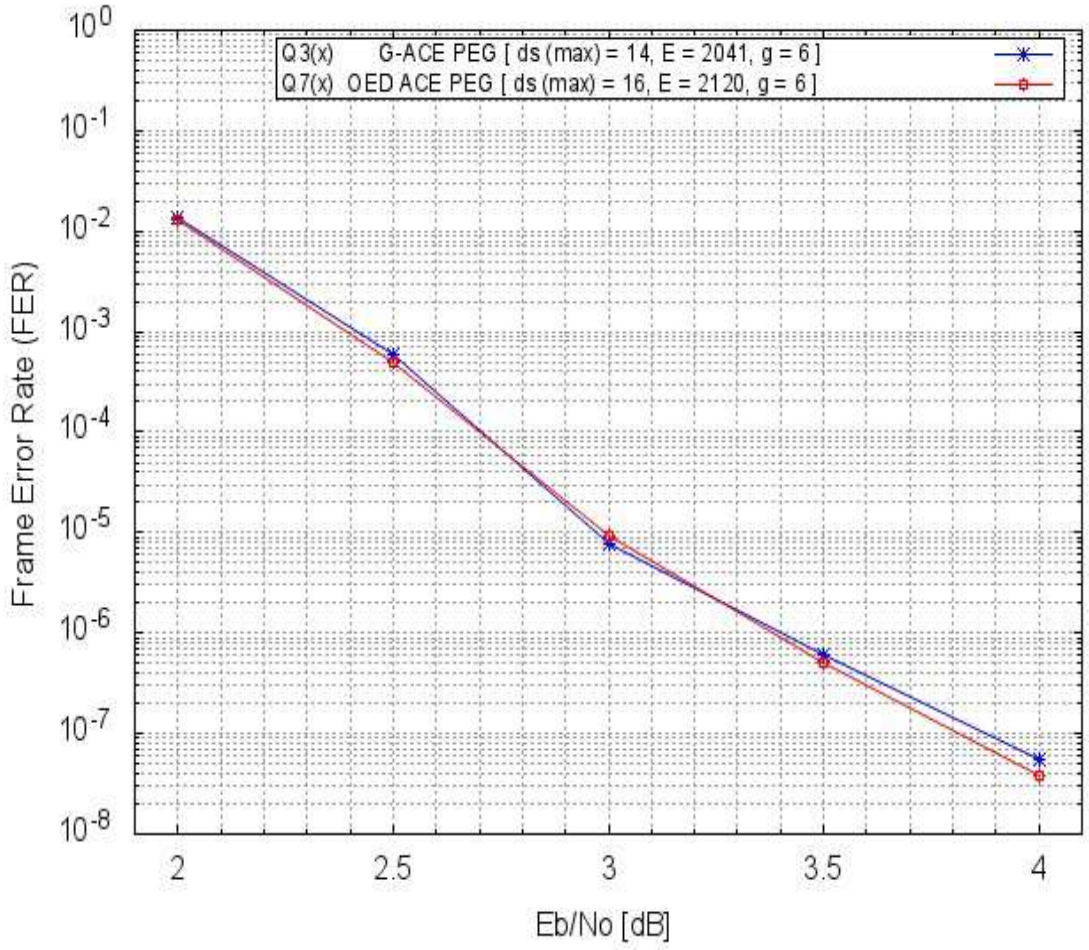

Figure 6.7: FER curves for rate-½ $(n,\ k) = (512, 256)$ codes, i) the lowest-error-floor G-ACE PEG code, and ii) the lowest-error-floor OED ACE PEG code

The relative performances of the lowest-error-floor rate-½ $(n,\ k) = (512,\ 256)$ G-ACE PEG code, and the lowest-error-floor rate-½ $(n,\ k) = (512,\ 256)$ OED ACE PEG code are shown in Figure 6.7.

Figure 6.8 shows the FER simulation results for the following rate-½ $(n,\ k) = (1024,\ 512)$ codes:

i) the lowest-error-floor G-ACE PEG code found, which has degree distribution $Q_4(x)$, with $d_{min} = 13$, $s_{min} = 18$, and

ii) the lowest-error-flooor ACE PEG code found, which also has degree distribution $Q_4(x)$, but with $d_{min} = s_{min} = 22$.

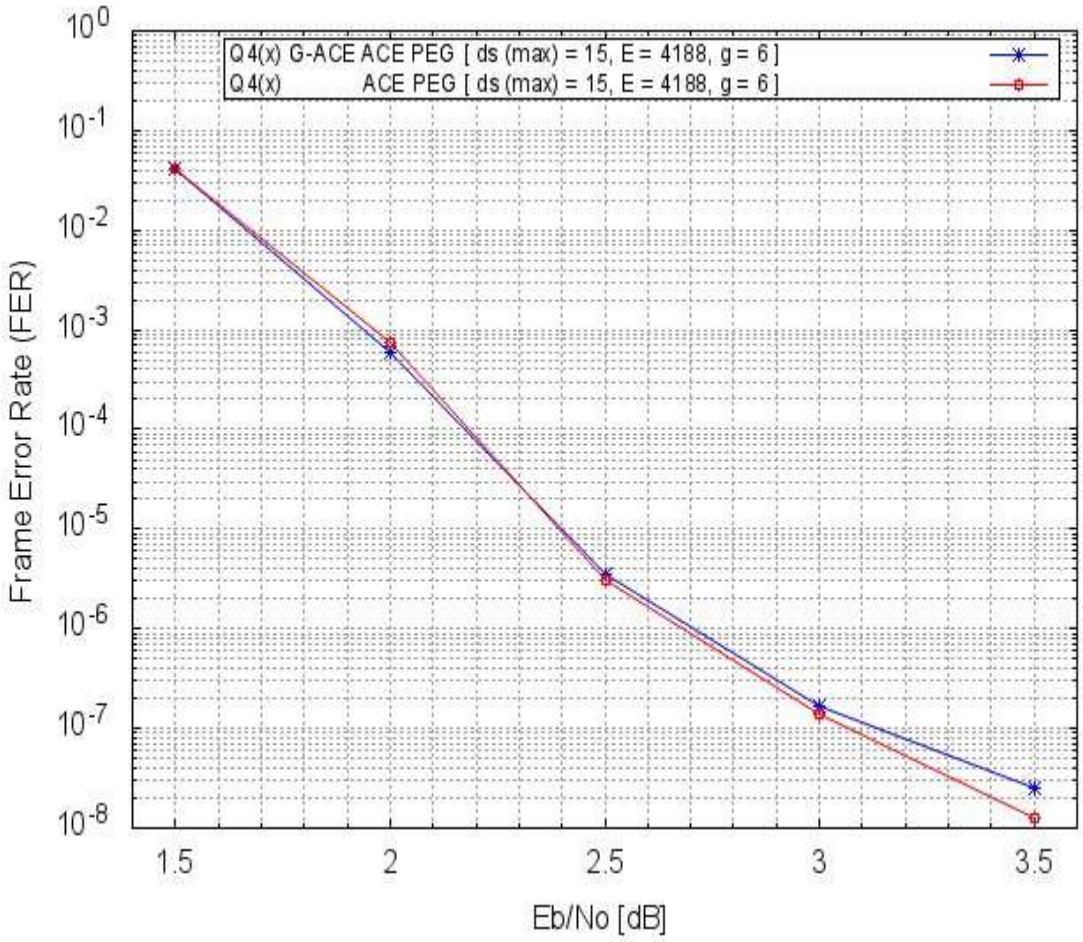

Figure 6.8: FER curves for rate-½ $(n,\ k) = (1024,\ 512)$ codes, i) the lowest-error-floor G-ACE PEG code, and ii) the lowest-error-floor ACE PEG code

The relative performances of the lowest-error-floor rate-½ $(n,\ k) = (1024,\ 512)$ G-ACE PEG code, and the lowest-error-floor rate-½ $(n,\ k) = (1024,\ 512)$ ACE PEG code are shown in Figure 6.8.

Although G-ACE PEG codes and ACE PEG codes have very similar d_{min} and s_{min} distributions, G-ACE PEG codes have girth and ACE properties which are superior to those of ACE PEG codes.

The distributed check node degrees in G-ACE PEG codes matrices may explain why they do not have significantly lower error-floor than ACE PEG codes in spite of the fact that they have superior girth and ACE properties. It may be inferred that the widely distributed check node degrees of G-ACE PEG codes compared to the more concentrated check node degrees of ACE PEG codes impacts negatively on the error-floor performance of G-ACE PEG codes. This follows from strong evidence that a concentrated parity-check node degree sequence is optimum [Bazzi et al., 2004], [Shokrollahi, 1999].

Experiments were carried out in an attempt to lower the error-floor of short-to-medium length LDPC codes by constructing OED codes which,

i) use a combination of the G-ACE PEG algorithm and the MLG (ES) PEG algorithm, and

ii) are based on modifications to DE optimized degree distributions of the benchmark codes, i.e. $Q_3(x)$ and $Q_9(x)$.

The rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ OED G-ACE PEG codes constructed in these experiments had even more widely distributed check node degrees and performed consistently worse than codes constructed using the DE optimized degree distributions in the G-ACE PEG algorithm.

6.5 Irregular Rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ MacKay LDPC Codes and Symbol Node Degree Distributions

The performance of regular and irregular rate-½ $(n, k) = (504, 252)$ and $(1024, 512)$ LDPC codes obtained from [MacKay, 2002], which are often used as reference point codes or benchmarks codes of good LDPC code performance, were simulated. The regular and irregular rate-½ $(504, 252)$ codes obtained from [MacKay, 2002] are called *PEGReg252x504* and *PEGirReg252x504*, respectively. Similarly, the regular and irregular rate-½ $(1008, 504)$ codes are called *PEGReg504x1008* and *PEGirReg504x1008*, respectively.

Irregular code *PEGirReg252x504* has symbol node degree distribution, $M_1(x) = 0.47817x^2 + 0.27976x^3 + 0.03571x^4 + 0.09722x^5 + 0.00794x^7 + 0.00198x^{14} + 0.09921x^{15}$, and irregular code *PEGirReg504x1008* has a similar symbol node degree distribution, $M_2(x) = 0.47718x^2 + 0.28075x^3 + 0.03472x^4 + 0.09722x^5 + 0.00893x^7 + 0.00099x^{14} + 0.10020x^{15}$. Both of the regular codes,

PEGReg252x504 and *PEGReg504x1008*, have regular symbol node degree ($d_s = 3$). These codes are not perfectly regular in check node degrees.

In Figure 6.9, the performance of the rate-½ $(504, 252)$ MacKay code, *PEGirReg252x504*, is compared to those of the rate-½ $(512, 256)$ benchmark and lowest-error-floor OED ACE PEG codes.

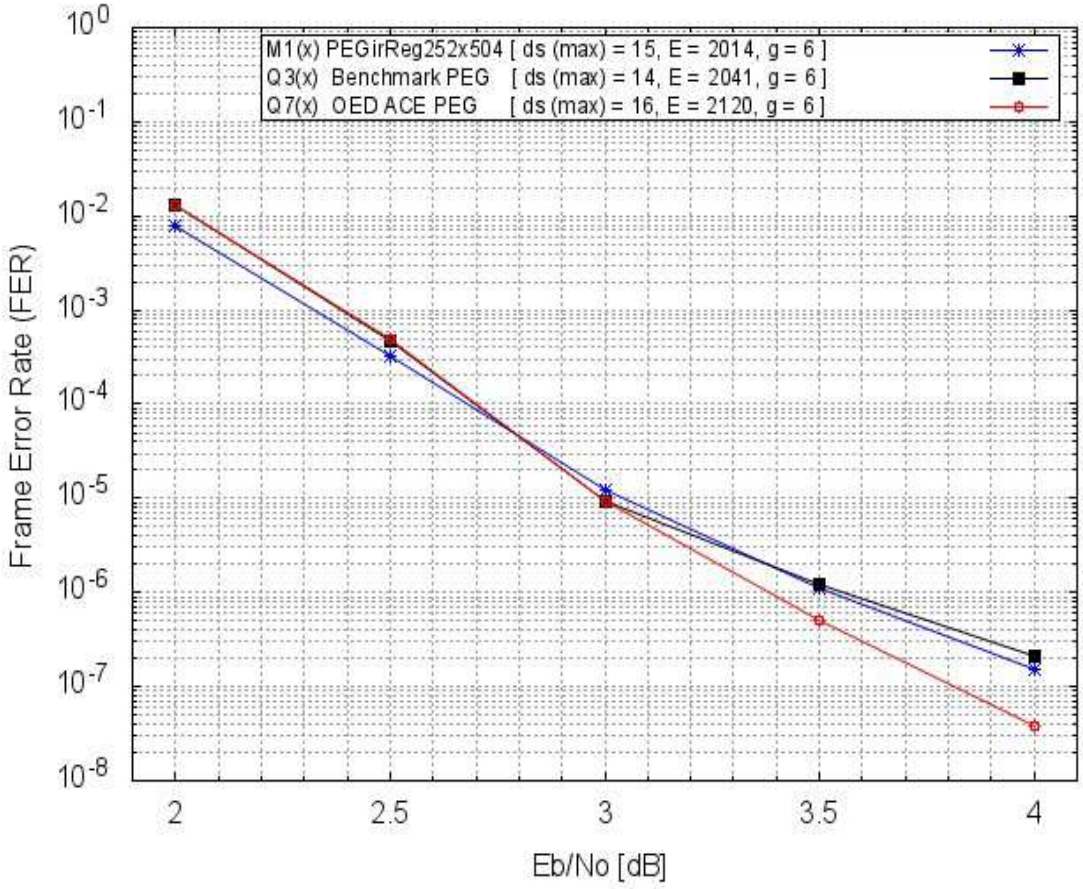

Figure 6.9: FER curves for the rate-½ $(504, 252)$ MacKay code, the rate-½ $(512, 256)$ benchmark code, and the lowest-error-floor rate-½ $(512, 256)$ OED ACE PEG code

From the results plotted in Figure 6.9, the rate-½ $(504, 252)$ MacKay code performs notably better than the rate-½ $(512, 256)$ benchmark code in both the waterfall and error-floor regions. The MacKay code also performs better than the OED ACE PEG code in the waterfall region. However, the OED ACE PEG code significantly outperforms the other two codes in the error-floor.

As a result of the superior performance of the MacKay code compared to the benchmark code, as shown in Figure 6.9, experiments to determine the performance of rate-½ $(n, k) = (512, 256)$

OED PEG, ACE PEG and OED ACE PEG codes which are constructed based on the degree distribution of the rate-½ (504, 252) MacKay code, i.e. $M_1(x)$, were carried out.

Figure 6.10 compares the lowest error-floor performance found for rate-½ (512, 256) OED PEG, ACE PEG and OED ACE PEG codes with constructions based on degree distribution $M_1(x)$ to the rate-½ (504, 252) MacKay code - *PEGirReg252x504*, the rate-½ (512, 256) benchmark code ($Q_3(x)$), and lowest-error-floor rate-½ (512, 256) OED ACE PEG code ($Q_7(x)$).

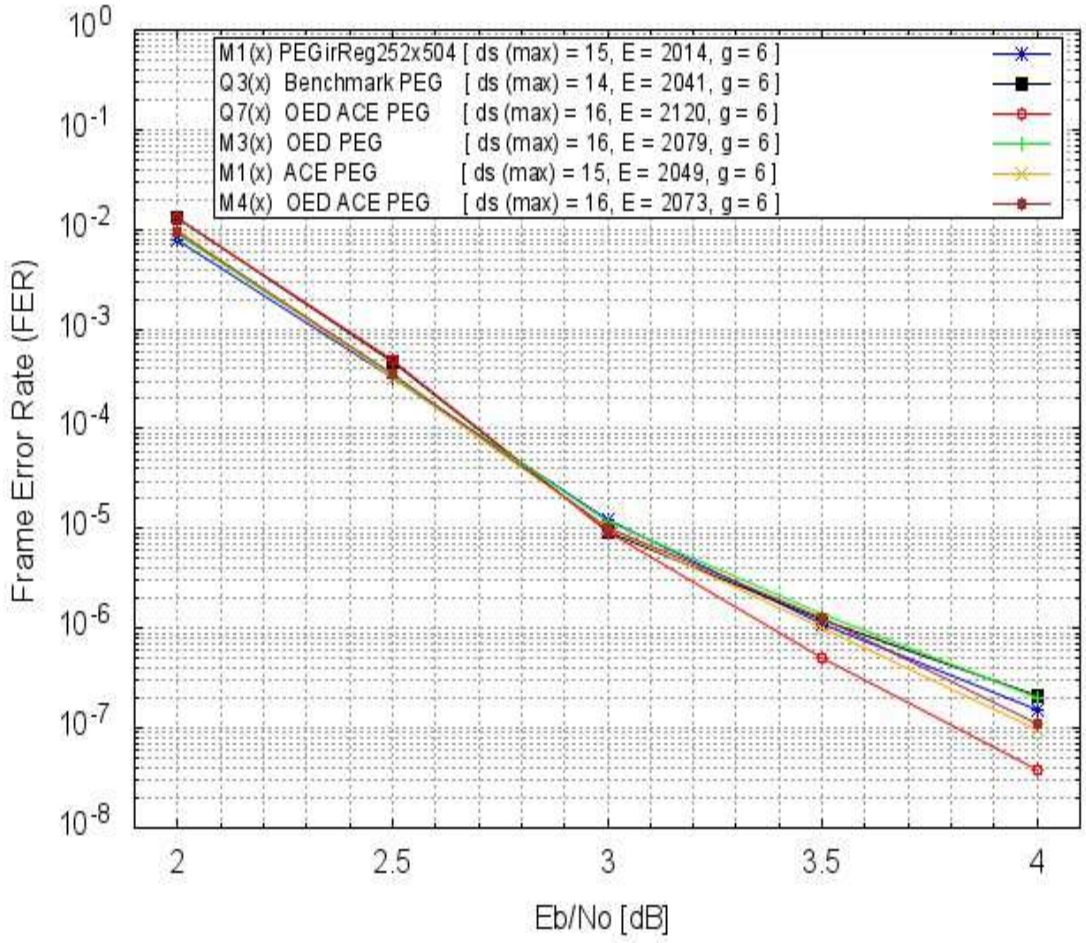

Figure 6.10: The rate-½ (512, 256) OED PEG, ACE PEG and OED ACE PEG codes based on MacKay distribution $M_1(x)$ vs. the rate-½ (504, 252) MacKay code, the (512, 256) benchmark code, and the lowest-error-floor (512, 256) OED ACE PEG code

The rate-½ (512, 256) OED PEG code based on the rate-½ (504, 252) MacKay code degree distribution $M_1(x)$ has output degree distribution $M_3(x)$, $M_3(x) = 0.47852x^2 + 0.27930x^3$

$+ 0.03516x^4 + 0.09766x^5 + 0.00781x^7 + 0.00195x^{13} + 0.00781x^{14} + 0.02148x^{15} + 0.07031x^{16}$; the $d_{s_{max}}$ of $M_1(x)$ was increased from 15 to 16. The rate-½ (512, 256) ACE PEG code based on degree distribution $M_1(x)$ has an unchanged degree distribution $M_1(x)$, but more edge connections due to the increased code length, and the rate-½ (512, 256) OED ACE PEG code based on degree distribution $M_1(x)$ has output degree distribution $M_4(x)$, $M_4(x) = 0.47852x^2 + 0.27930x^3 + 0.03516x^4 + 0.09766x^5 + 0.00781x^7 + 0.01953x^{14} + 0.015625x^{15} + 0.06641x^{16}$; the $d_{s_{max}}$ of $M_1(x)$ was increased from 15 to 16.

As shown in Figure 6.10, rate-½ (512, 256) OED PEG, ACE PEG and OED ACE PEG codes constructed based on the rate-½ (504, 252) MacKay code degree distribution $M_1(x)$ have improved performance in the waterfall region and perform better than both the rate-½ (512, 256) benchmark code ($Q_3(x)$) and the lowest-error-floor rate-½ (512, 256) OED ACE PEG code ($Q_7(x)$) in this region. As a result of improved ACE, the rate-½ (512, 256) ACE PEG and OED ACE PEG codes with degree distributions $M_1(x)$ and $M_4(x)$ respectively, performed better than the rate-½ (504, 252) MacKay code in the error-floor region. The rate-½ (512, 256) OED PEG code with degree distribution $M_3(x)$ had a very similar error-floor performance as the benchmark code and performed worse than the rate-½ (504, 252) MacKay code in this region. However, none of the rate-½ (512, 256) OED PEG, ACE PEG and OED ACE PEG codes constructed based on $M_1(x)$ had an error-floor as low as the lowest-error-floor rate-½ (512, 256) OED ACE PEG code ($Q_7(x)$) constructed based on the benchmark code degree distribution $Q_3(x)$. The gap in the error-floor performance of the rate-½ (512, 256) OED ACE PEG code ($Q_7(x)$) compared to the other codes in Figure 6.10 is significant. Therefore, attempts to improve on the best known error-floor performance for rate-½ (512, 256) LDPC codes by adopting the degree distribution of the rate-½ (504, 252) MacKay code were unsuccessful.

In Figure 6.11, the performance of the rate-½ (1008, 504) MacKay code, *PEGirReg504x1008*, is compared to those of the rate-½ (1024, 512) benchmark and lowest-error-floor ACE PEG codes.

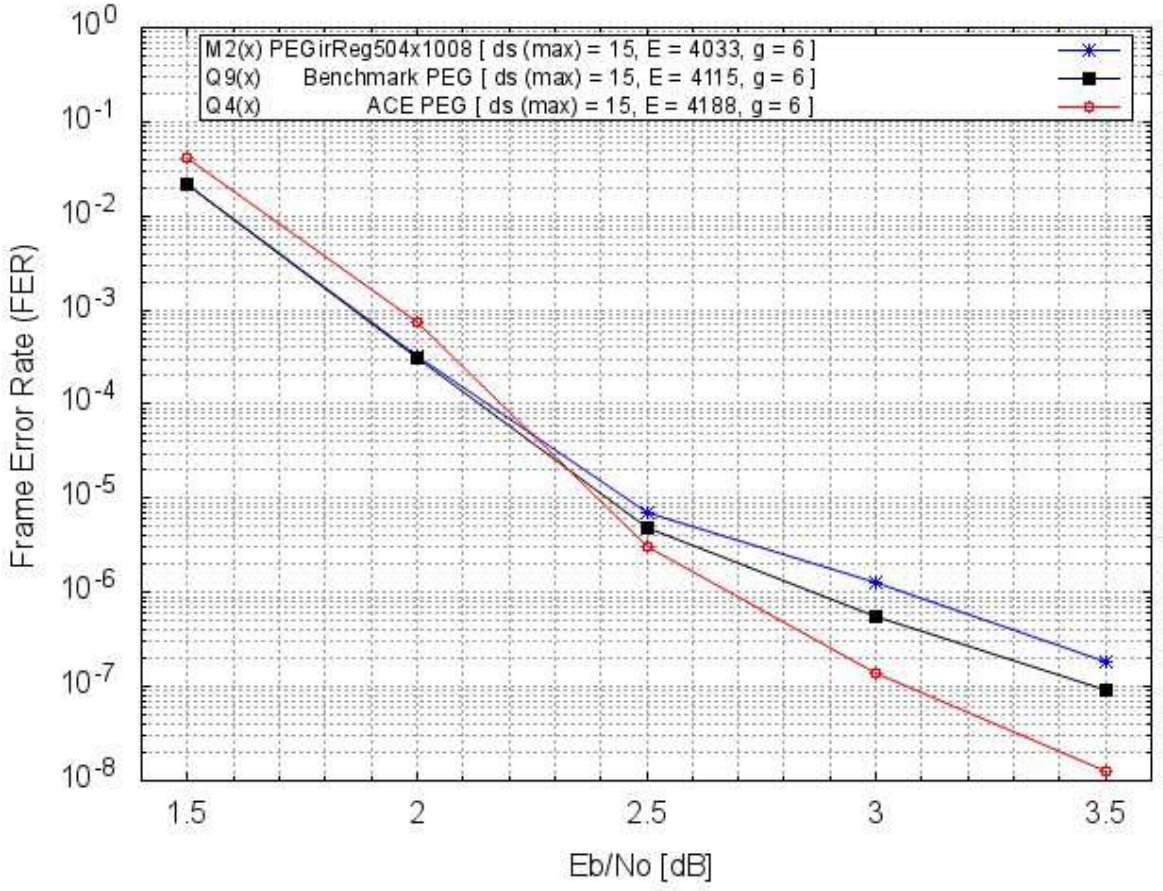

Figure 6.11: FER curves for the rate-½ (1008, 504) MacKay code, the rate-½ (1024, 512) benchmark code and the lowest-error-floor rate-½ (1024, 512) ACE PEG code

As shown in Figure 6.11, the rate-½ (1024, 512) benchmark and ACE PEG codes perform better than the rate-½ (1008, 504) MacKay code in the error-floor region. Although, the ACE PEG code significantly outperforms both the MacKay and benchmark codes in the error-floor region, it has the worst performance in the waterfall region. The MacKay and benchmark codes have very similar performance in the waterfall region.

When the degree distribution of the rate-½ (1008, 504) MacKay, i.e. $M_2(x)$, was used to construct rate-½ (1024, 512) ACE PEG codes, the resultant codes had slightly lower error-floor than the rate-½ (1024, 512) benchmark code. However, within the range of decoding performance experiments

carried out, none of the ACE PEG codes constructed using the MacKay code degree distribution $M_2(x)$ performed as well as lowest-error-floor ACE PEG code with degree distribution $Q_4(x)$.

6.6 Regular Rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ MacKay Codes versus Regular Rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ Standard PEG Codes

In this section we investigate the performance of the regular rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ MacKay codes called *PEGReg252x504* and *PEGReg504x1008*, respectively [MacKay, 2002]. Both of these regular codes have regular symbol node degree $d_s = 3$ and are not perfectly regular in check node degrees. Four ensembles of 10 regular rate-½ codes each were constructed using the standard PEG algorithm [Hu et al., 2001]; an ensemble of $(n, k) = (512, 256)$ codes with $d_s = 3$, an ensemble of $(512, 256)$ codes with $d_s = 4$, an ensemble of $(1024, 504)$ codes with $d_s = 3$, and an ensemble of $(1024, 504)$ codes with $d_s = 4$. Four codes were randomly selected, one from each of these four code ensembles, and their decoding performances were simulated. Similar to the MacKay codes, these regular standard PEG codes are not perfectly regular in check node degrees. The superior performance of the lowest-error-floor irregular rate-½ $(512, 256)$ and $(1024, 504)$ codes over i) their regular counterparts, and ii) the regular rate-½ $(504, 252)$ and $(1008, 504)$ MacKay codes are graphically illustrated.

The Monte Carlo simulation results presented in this section are for BPSK modulation over the AWGN channel. Codes were decoded using the standard BP/SPA decoder at a maximum of 100 decoding iterations. Identical noise patterns were used for each set of points at the same E_b/N_o on the error rate curves. The same numbers of blocks (frames) were used for all codes at each E_b/N_o ratio simulated.

In Figure 6.12, the performance of the regular rate-½ $(n,\ k) = (504, 252)$ MacKay code is compared to that of the two randomly selected regular rate-½ $(n,\ k) = (512, 256)$ standard PEG codes with symbol node degrees $d_s = 3$ and 4 respectively, and the lowest-error-floor irregular rate-½ $(n,\ k) = (512, 256)$ OED ACE PEG code.

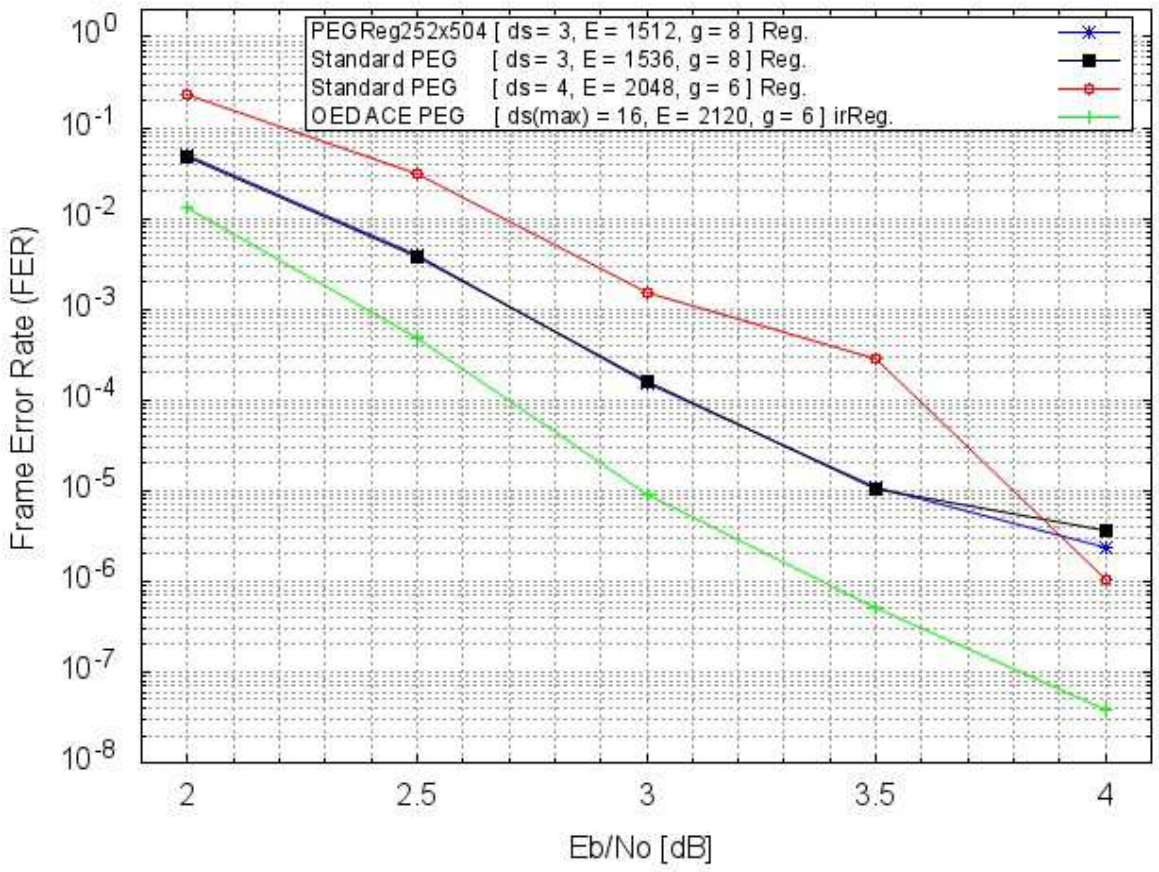

Figure 6.12: FER for the regular rate-½ $(504,\ 252)$ MacKay code, regular rate-½ $(512,\ 256)$ standard PEG codes with $d_s = 3$ and 4 respectively, and the lowest-error-floor irregular rate-½ $(512,\ 256)$ OED ACE PEG code

Among the regular codes in Figure 6.12, the regular rate-½ $(504,\ 252)$ MacKay code has the best performance in both the waterfall and error-floor regions. The regular rate-½ $(512,\ 256)$ standard PEG code with $d_s = 3$ has an almost identical performance to the MacKay code; the performance of the MacKay code surpasses the standard PEG code only for $E_b/N_0 > 3.50dB$. Although the regular rate-½ $(512,\ 256)$ standard PEG code with $d_s = 4$ has the worst general performance, its performance in the error-floor region improves dramatically at $E_b/N_0 > 3.50dB$ so that at $E_b/N_0 =$

$4.00dB$ and beyond it has the lowest error-floor of all the regular codes. Compared to the regular codes, the irregular rate-½ $(512, 256)$ OED ACE PEG code with degree distribution $Q_7(x)$ significantly outperforms all the regular codes in both the waterfall and error-floor regions. At $E_b/N_0 > 3.00dB$, the OED ACE PEG code has less than a tenth of the error rate of the best regular codes of similar length and rate.

In Figure 6.13, the performance of the regular rate-½ $(n, k) = (1008, 504)$ MacKay code is compared to that of the two randomly selected regular rate-½ $(n, k) = (1024, 512)$ standard PEG codes with symbol node degrees $d_s = 3$ and 4 respectively, and the lowest-error-floor irregular rate-½ $(n, k) = (1024, 512)$ ACE PEG code.

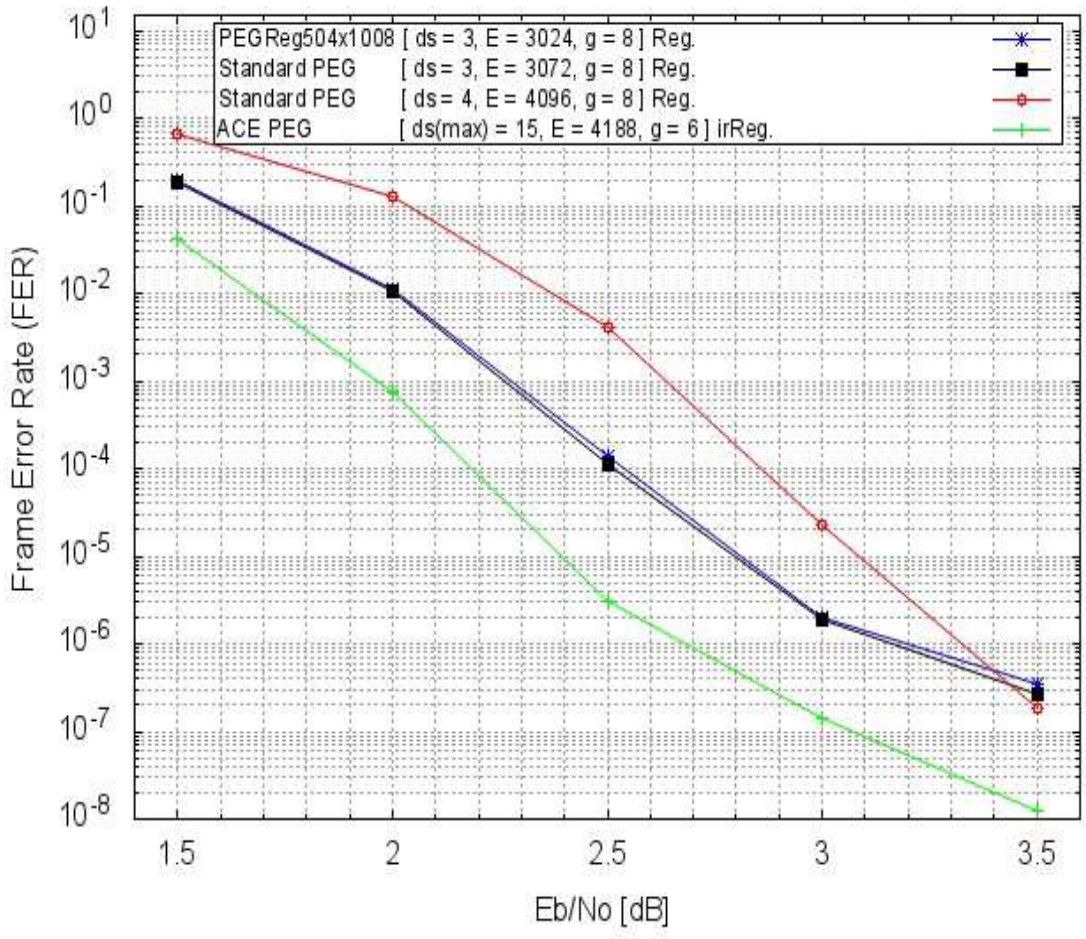

Figure 6.13: FER for the regular rate-½ $(1008, 504)$ MacKay code, the regular rate-½ $(1024, 512)$ standard PEG codes with $d_s = 3$ and 4 respectively, and the lowest-error-floor irregular rate-½ $(1024, 512)$ ACE PEG code

Among the regular codes in Figure 6.13, the regular rate-½ $(n,\ k) = (1024,\ 512)$ standard PEG code with $d_s = 3$ has the best performance in both the waterfall and error-floor regions. This standard PEG code has an almost identical performance as the rate-½ $(n,\ k) = (1008,\ 504)$ MacKay code; the slightly better performance of the standard PEG code is attributed to its larger $(n,\ k)$ dimensions compared to the MacKay code. Although the regular rate-½ $(1024,\ 512)$ standard PEG code with $d_s = 4$ has the worst general performance, its performance in the error-floor region improves dramatically at $E_b/N_0 > 3.00dB$ so that at $E_b/N_0 = 3.50dB$ and beyond it has the lowest error-floor of all the regular codes. Compared to the regular codes, the irregular rate-½ $(1024,\ 512)$ ACE PEG code with degree distribution $Q_4(x)$ significantly outperforms all the regular codes in both the waterfall and error-floor regions. At $E_b/N_0 \geq 2.00dB$, the ACE PEG code has less than a tenth of the error rate of the best regular codes of similar length and rate.

The LDPC matrix configuration file for the rate-½ $(n,\ k) = (1024,\ 512)$ ACE PEG code with the lowest error-floor, as presented in Figure 6.5, Figure 6.6, Figure 6.8, Figure 6.11, and Figure 6.13, is available in the Appendices (See Appendix K).

6.7 The Error-Floor Performance of Short-to-Medium Length Irregular LDPC Codes from Recent Modified PEG Algorithms

In this section, the performance of the rate-½ $(n,\ k) = (512,\ 256)$ and $(1024,\ 512)$ irregular LDPC codes with the lowest error-floor found in this research are compared to those of LDPC codes obtained from other recently proposed modifications to the PEG algorithm aimed at lowering the error-floor of the short-to-medium length LDPC codes constructed. Comparisons are only feasible and fair if the published performance results for respective PEG algorithm modifications are in FER for,

i) BP/SPA decoding over AWGN channels using a similar maximum number of decoding iterations as in this research, that is ~ 100 , and

ii) rate-½ LDPC codes which have similar $(n,\ k)$ dimensions as those investigated in this research.

- **The Error-Floor Performance of standard PEG codes:** In their pioneering work on the PEG algorithm, Hu et al. presented BER and FER performance results for only regular LDPC codes [Hu et al., 2001]. However, in their follow-up work of 2005, performance simulation results were presented for rate-½ $(504, 252)$ and $(1008, 504)$ irregular LDPC codes with DE optimized degree distributions (see Fig. 6 and Fig. 7 of [Hu et al., 2005]). In the waterfall region, for $E_b/N_o \leq 2.25dB$, the best rate-½ $(504, 252)$ PEG code (see Fig. 6 of [Hu et al., 2005]) and the rate-½ $(512, 256)$ OED ACE PEG code with degree distribution $Q_7(x)$ found in this research have similar performances. However, as the error-floor region is approached, i.e. for $E_b/N_o \geq 2.50dB$, the rate-½ $(512, 256)$ OED ACE PEG code significantly outperforms the best rate-½ $(504, 252)$ PEG code in Fig. 6 of [Hu et al., 2005]. Similarly, in the waterfall region, i.e. $E_b/N_o \leq 2.00dB$, the rate-½ $(1008, 504)$ irregular PEG and upper triangular irregular PEG codes (see Fig. 7 of [Hu et al., 2005]) slightly outperform the rate-½ $(1024, 512)$ ACE PEG code with degree distribution $Q_4(x)$ found in this research. However, as the error-floor region is approached, i.e. for $E_b/N_o \geq 2.25dB$, our rate-½ $(1024, 512)$ ACE PEG code significantly outperforms the rate-½ $(1008, 504)$ irregular PEG and upper triangular irregular PEG codes in Fig. 7 of [Hu et al., 2005].

- **The Error-Floor Performance of IPEG codes:** Xiao and Banihashemi provided BER and FER performance results for rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ LDPC codes obtained using the improved PEG (IPEG) construction (see Fig. 1 of [Xiao and Banihashemi, 2004]). At $E_b/N_o = 2.00dB$, the rate-½ $(504, 252)$ ACE PEG (or IPEG) code (see the 2nd to the highest MER (- - -) curve in Fig. 1 of [Xiao and Banihashemi, 2004]), and the rate-½ $(512, 256)$ OED ACE PEG code with degree distribution $Q_7(x)$ found in this research have identical performance. However, as the error-floor region is approached, i.e. $E_b/N_o \geq 2.50dB$, the rate-½ $(512, 256)$ OED ACE PEG code outperforms the rate-½ $(504, 252)$ ACE PEG code in Fig. 1 of [Xiao and Banihashemi, 2004]. Similarly, in the waterfall region, at $E_b/N_o = 2.00dB$, the rate-½ $(1008, 504)$ ACE PEG code (see the lowest MER (- - -) curve in Fig. 1 of

[Xiao and Banihashemi, 2004]) and the rate-½ $(1024, 512)$ ACE PEG code with degree distribution $Q_4(x)$ found in this research have identical performance. However, as the error-floor region is approached, i.e. for $E_b/N_o \geq 2.50dB$, the rate-½ $(1024, 512)$ ACE PEG code significantly outperforms the rate-½ $(1008, 504)$ ACE PEG code in Fig. 1 of [Xiao and Banihashemi, 2004]. Improved degree distributions, in addition to a higher code dimension (n, k) and a higher maximum number of decoding iterations (100 versus 80), are responsible for the superior error-floor performance of the LDPC codes constructed in this research compared to Xiao and Banihashemi's IPEG (or ACE PEG) codes.

- **The Error-Floor Performance of Modified PEG codes:** Richter and Hof provided the FER performance result for a rate-½ $(n, k) = (1000, 500)$ LDPC code obtained using their modified PEG algorithm (see Fig. 11 of [Richter and Hof, 2006]). The performance of their modified PEG code is only slightly better than that of the rate-½ $(1024, 512)$ ACE PEG code with degree distribution $Q_4(x)$ found in this research. This is in spite of the fact that the code performance simulation presented in the work was carried out using the shuffled BP decoder of [Zhang and Fossorier, 2005] with a maximum of 500 decoding iterations, and the linear approximation explained in [Richter et al., 2005]. A significantly improved BP/SPA decoder which does not require as many iterations will be introduced in Chapter 7 of this thesis.

- **The Error-Floor Performance of EMPEG codes:** Sharon and Litsyn gave the FER performance result for a rate-½ $(n, k) = (576, 288)$ LDPC code obtained using their error minimization PEG (EMPEG) algorithm (see Fig. 9 of [Sharon and Litsyn, 2008]). The rate-½ $(512, 256)$ OED ACE PEG code with degree distribution $Q_7(x)$ found in this research slightly outperforms the rate-½ $(n, k) = (576, 288)$ EMPEG code in both the waterfall and error-floor regions. This is in spite of the larger dimensions of the EMPEG code.

- **The Error-Floor Performance of ETS-constrained PEG codes:** Khazraie et al. provided the FER performance results for a rate-½ $(n, k) = (1008, 504)$ irregular LDPC code with a DE

optimized degree distribution which was constructed using the proposed elementary trapping set (ETS) constrained PEG algorithm (ETS-constrained PEG) (see Fig. 1 of [Khazraie et al., 2012]). A comparison of the performance results for the rate-½ $(1008, 504)$ ETS-constrained PEG code and the rate-½ $(1024, 512)$ ACE PEG code with degree distribution $Q_4(x)$ in this research shows that the ACE PEG code has a superior performance in both the waterfall and error-floor regions. The performance gap is more significant in the error-floor region; the ACE PEG code only slightly outperforms the ETS-constrained PEG code in the waterfall region. It is pertinent to note that while the ETS-constrained PEG code performance results were obtained using a maximum of 50 decoding iterations, the ACE PEG code performance results were obtained using a maximum of 100 decoding iterations. However, the differences in the maximum number of decoding iterations used is not likely to be sufficient to fully account for the superior error-floor of the ACE PEG code compared to the ETS-constrained code.

6.8 Summary

- A minimum local girth (edge skipping) (MLG (ES)) PEG algorithm which controls the minimum local girths connected during the construction of LDPC code Tanner graphs has been presented. The MLG (ES) PEG algorithm controls the global girth of the codes it constructs by skipping all edge connections which would have resulted in undesired local girths.

- The concept of optimal low-correlated-edge density (OED) codes has been introduced.

- Simulation results show that irregular rate-½ $(n, k) = (512, 256)$ OED ACE PEG codes have lower error-floor than the lowest error-floor published for LDPC codes of identical rate and length constructed using the improved PEG (IPEG) construction [Xiao and Banihashemi, 2004], and the generalized ACE constrained PEG (G-ACE PEG) algorithm [Vukobratović and Šenk, 2008].

- Consequent to an improved symbol node degree distribution, irregular rate-½ $(n, k) = (1024, 512)$ ACE PEG codes which have lower error-floors than published for LDPC codes of

identical rate and length have been found.

- Random rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ PEG constructed LDPC codes have been shown to significantly outperform structured rate-½ $(n, k) = (576, 288)$ and $(1056, 528)$ IEEE 802.16E (WiMAX) LDPC codes respectively in the error-floor region.

- As a result of the prioritization used in the candidate check node selection criteria of the G-ACE PEG algorithm, the LDPC codes constructed are not as check node regular as ACE PEG codes. Therefore, G-ACE PEG codes have more distributed check node degrees. Although G-ACE PEG codes and ACE PEG codes have very similar d_{min} and s_{min} distributions, G-ACE PEG codes have girth and ACE properties which are superior to those of ACE PEG codes. Nevertheless, G-ACE PEG codes do not have lower error-floor than ACE PEG codes. The widely distributed check node degrees of G-ACE PEG codes appears to have a negative impact on their error-floor performance.

- Irregular rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ OED ACE PEG and ACE PEG codes significantly outperform irregular rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ MacKay codes respectively. The adoption of MacKay code degree distributions do not result in irregular rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ LDPC codes with lower error-floors than already obtained. Regular rate-½ $(n, k) = (512, 256)$ and $(1024, 512)$ codes with similar or better performance than regular rate-½ $(n, k) = (504, 252)$ and $(1008, 504)$ MacKay codes respectively have been found.

Chapter 7

An improved BP/SPA (IBP/SPA) decoder

7.1 Introduction

Two novel modifications to the standard iterative message-passing BP/SPA decoder which result in an unprecedented improvement to BP/SPA decoding performance are described in this chapter. Using these simple modifications to the BP/SPA decoding algorithm, irregular LDPC code performance in both the waterfall and error-floor regions are significantly improved. Results of performance simulations carried out using the proposed improved BP/SPA (IBP/SPA) decoder show a generalized improvement in the decoding performance of short-to-medium length LDPC codes. The modifications made to the conventional BP/SPA decoding process to obtain the IBP/SPA decoder results in an efficient and effective technique for breaking trapping sets in LDPC codes. To the best of the author's knowledge, the generalized improvement in the decoding performance of short-to-medium length LDPC codes achieved using the proposed IBP/SPA decoder are significantly better than all the performance improvements published for other modified BP/SPA decoders in the literature.

7.2 A Modified BP/SP Algorithm: Fixed Parity-Check Equation Sequence (FPCES) versus Randomized Parity-Check Equation Sequence (RPCES)

This section describes a modification to the iterative message-passing BP/SP algorithm used for decoding in standard BP/SPA decoders which results in not only faster decoding convergence but also a generalized improvement of the performance of short-to-medium length LDPC codes over AWGN

channels. The standard BP/SPA decoder was implemented as described in Section 2.5 and, used to obtain the LDPC code decoding performance simulations results presented in all the previous chapters of this thesis.

In each decoding iteration, the BP/SP algorithm cycles through the equations in the probability matrix H_P in a fixed sequence. The equation sequence most commonly used in each decoding iteration is the nondecreasing serial numbers sequence. That is, in each iteration the decoding takes place using the following parity-check equation sequence,

$$equation\ 0,\ equation\ 1,\ equation\ 2,\ equation\ 3,\ equation\ 4,\ \ldots\ldots,\ equation\ m$$

This is exemplified in Table 2.3 - Table 2.11 where the decoding in each of the three decoding iterations executed, i.e. *iteration* 0, *iteration* 1, and *iteration* 2, were carried out using a fixed equation sequence, i.e. *equation* 0, *equation* 1, and then *equation* 2, as follows,

Table 2.3	$Decoding\ iteration\ 0, equation\ \mathbf{0}$
Table 2.4	$,\ ,\ ,\ 0, equation\ \mathbf{1}$
Table 2.5	$,\ ,\ ,\ 0, equation\ \mathbf{2}$
Table 2.6	$Decoding\ iteration\ 1, equation\ \mathbf{0}$
Table 2.7	$,\ ,\ ,\ 1, equation\ \mathbf{1}$
Table 2.8	$,\ ,\ ,\ 1, equation\ \mathbf{2}$
Table 2.9	$Decoding\ iteration\ 2, equation\ \mathbf{0}$
Table 2.10	$,\ ,\ ,\ 2, equation\ \mathbf{1}$
Table 2.11	$,\ ,\ ,\ 2, equation\ \mathbf{2}$

The same equation sequence, i.e. $equation\ 0 \rightarrow equation\ 1 \rightarrow equation\ 2$, is used for each decoding iteration. The standard BP/SPA decoder uses a fixed parity-check equation sequence (FPCES). Consequently, the BP/SP algorithm in the standard BP/SPA decoder is described as a FPCES BP/SP algorithm, and the standard BP/SPA decoder will also be referred to as the FPCES BP/SPA decoder in the rest of this chapter.

An experiment to discover the effects of using a random parity-check equation sequence in the decoding iterations of the BP/SP algorithm was carried out. The decoding exemplification in Table 7.1 - Table 7.9 was applied in the same circumstance as that in Table 2.3 - Table 2.11. However, a random equation sequence was used in the decoding iterations of Table 7.1 - Table 7.9. The

following random equation sequence was used in the randomized parity-check equation sequence (RPCES) BP/SP algorithm exemplified in Table 7.1 to Table 7.9.

Table 7.1	$Decoding\ iteration\ 0, equation\ \mathbf{2}$
Table 7.2	$,\quad ,\quad ,\quad 0, equation\ \mathbf{1}$
Table 7.3	$,\quad ,\quad ,\quad 0, equation\ \mathbf{0}$
Table 7.4	$Decoding\ iteration\ 1, equation\ \mathbf{1}$
Table 7.5	$,\quad ,\quad ,\quad 1, equation\ \mathbf{2}$
Table 7.6	$,\quad ,\quad ,\quad 1, equation\ \mathbf{0}$
Table 7.7	$Decoding\ iteration\ 2, equation\ \mathbf{2}$
Table 7.8	$,\quad ,\quad ,\quad 2, equation\ \mathbf{0}$
Table 7.9	$,\quad ,\quad ,\quad 2, equation\ \mathbf{1}$

In order to observe the effect of using random equation sequences in the decoding iterations of the proposed RPCES BP/SPA decoder, the same $(n,\ k) = (6,\ 3)$ linear code H matrix, information codeword, received codeword vector, and received bit probabilities used in the FPCES BP/SP algorithm exemplification in Table 2.3 - Table 2.11 of Section 2.5 were used in the decoding exemplification of Table 7.1 - Table 7.9. The inputs used for both decoding algorithms are as follows,

Information: 111

Codeword: 111000 $\Longrightarrow$ Transmitted codeword = [111000] (0 info error)

Channel: [+1.0,+1.0,+1.0,-1.0,-1.0,-1.0] $\Longrightarrow$ BPSK modulated codeword

Received: [-0.7,+1.6,+0.4,-1.1,-1.1,-0.4] $\Longrightarrow$ Received codeword = [011000] (1 info error)

Probabilities: [0.15, 0.98, 0.74, 0.05, 0.05, 0.26] (using equation (2.17) at $SNR = 1/\sigma^2 = 1.239$ which is equivalent to $SNR(dB) = 0.931$)

Table 7.1: Decoding iteration **0**, equation **2**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.50	0.50	0.50		
equation 1	0.50		0.50		0.50	
equation 2	0.50	0.50				0.50

Decoding iteration 0, equation 2

Input probabilities to SPC decoder:

$p_0 = 0.15 = f(0.15,0.50)$ $p_1 = 0.98 = f(0.98,0.50)$ $p_5 = 0.26 = f(0.26)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_1)(1-2p_5)}{2} = 0.73$ $p_1 = \frac{1-(1-2p_0)(1-2p_5)}{2} = 0.33$ $p_5 = \frac{1-(1-2p_0)(1-2p_1)}{2} = 0.84$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.32 = f(0.15,0.50,0.73)$ $p_1 = 0.96 = f(0.98,0.50,0.33)$ $p_2 = 0.74 = f(0.74,0.50,0.50)$

Decoded data: $b_0 = 0$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.2: Decoding iteration **0**, equation **1**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.50	0.50	0.50		
equation 1	0.50		0.50		0.50	
equation 2	0.73	0.33				0.84

Decoding iteration 0, equation 1

Input probabilities to SPC decoder:

$p_0 = 0.32 = f(0.15,0.73)$ $p_2 = 0.74 = f(0.74,0.50)$ $p_4 = 0.05 = f(0.05)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_2)(1-2p_4)}{2} = 0.72$ $p_2 = \frac{1-(1-2p_0)(1-2p_4)}{2} = 0.34$ $p_4 = \frac{1-(1-2p_0)(1-2p_2)}{2} = 0.59$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.55 = f(0.15,0.72,0.73)$ $p_1 = 0.96 = f(0.98,0.50,0.33)$ $p_2 = 0.59 = f(0.74,0.50,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.3: Decoding iteration **0**, equation **0**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.50	0.50	0.50		
equation 1	0.72		0.34		0.59	
equation 2	0.73	0.33				0.84

Decoding iteration 0, equation 0

Input probabilities to SPC decoder:

$p_1 = 0.96 = f(0.98,0.33)$ $p_2 = 0.59 = f(0.74,0.34)$ $p_3 = 0.05 = f(0.05)$

Output extrinsic information:

$p_1 = \frac{1-(1-2p_2)(1-2p_3)}{2} = 0.58$ $p_2 = \frac{1-(1-2p_1)(1-2p_3)}{2} = 0.91$ $p_3 = \frac{1-(1-2p_1)(1-2p_2)}{2} = 0.42$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.55 = f(0.15,0.72,0.73)$ $p_1 = 0.97 = f(0.98,0.58,0.33)$ $p_2 = 0.94 = f(0.74,0.91,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.4: Decoding iteration **1**, equation **1**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.58	0.91	0.42		
equation 1	0.72		0.34		0.59	
equation 2	0.73	0.33				0.84

Decoding iteration 1, equation 1

Input probabilities to SPC decoder:

$p_0 = 0.32 = f(0.15,0.73)$ $p_2 = 0.97 = f(0.74,0.91)$ $p_4 = 0.05 = f(0.05)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_2)(1-2p_4)}{2} = 0.92$ $p_2 = \frac{1-(1-2p_0)(1-2p_4)}{2} = 0.34$ $p_4 = \frac{1-(1-2p_0)(1-2p_2)}{2} = 0.67$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.73)$ $p_1 = 0.97 = f(0.98,0.58,0.33)$ $p_2 = 0.94 = f(0.74,0.91,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.5: Decoding iteration **1**, equation **2**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.58	0.91	0.42		
equation 1	0.92		0.34		0.67	
equation 2	0.73	0.33				0.84

Decoding iteration 1, equation 2

Input probabilities to SPC decoder:

$p_0 = 0.67 = f(0.15,0.92)$ $p_1 = 0.99 = f(0.98,0.58)$ $p_5 = 0.26 = f(0.26)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_1)(1-2p_5)}{2} = 0.73$ $p_1 = \frac{1-(1-2p_0)(1-2p_5)}{2} = 0.58$ $p_5 = \frac{1-(1-2p_0)(1-2p_1)}{2} = 0.33$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.73)$ $p_1 = 0.99 = f(0.98,0.58,0.58)$ $p_2 = 0.94 = f(0.74,0.91,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.6: Decoding iteration **1**, equation **0**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.58	0.91	0.42		
equation 1	0.92		0.34		0.67	
equation 2	0.73	0.58				0.33

Decoding iteration 1, equation 0

Input probabilities to SPC decoder:

$p_1 = 0.99 = f(0.98,0.58)$ $p_2 = 0.59 = f(0.74,0.34)$ $p_3 = 0.05 = f(0.05)$

Output extrinsic information:

$p_1 = \frac{1-(1-2p_2)(1-2p_3)}{2} = 0.58$ $p_2 = \frac{1-(1-2p_1)(1-2p_3)}{2} = 0.94$ $p_3 = \frac{1-(1-2p_1)(1-2p_2)}{2} = 0.41$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.73)$ $p_1 = 0.99 = f(0.98,0.58,0.58)$ $p_2 = 0.96 = f(0.74,0.94,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.7: Decoding iteration **2**, equation **2**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.58	0.94	0.41		
equation 1	0.92		0.34		0.67	
equation 2	0.73	0.58				0.33

Decoding iteration 2, equation 2

Input probabilities to SPC decoder:

$p_0 = 0.67 = f(0.15,0.92)$ $p_1 = 0.99 = f(0.98,0.58)$ $p_5 = 0.26 = f(0.26)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_1)(1-2p_5)}{2} = 0.73$ $p_1 = \frac{1-(1-2p_0)(1-2p_5)}{2} = 0.58$ $p_5 = \frac{1-(1-2p_0)(1-2p_1)}{2} = 0.33$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.73)$ $p_1 = 0.99 = f(0.98,0.58,0.58)$ $p_2 = 0.96 = f(0.74,0.94,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.8: Decoding iteration **2**, equation **0**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.58	0.94	0.41		
equation 1	0.92		0.34		0.67	
equation 2	0.73	0.58				0.33

Decoding iteration 2, equation 0

Input probabilities to SPC decoder:

$p_1 = 0.99 = f(0.98,0.58)$ $p_2 = 0.59 = f(0.74,0.34)$ $p_3 = 0.05 = f(0.05)$

Output extrinsic information:

$p_1 = \frac{1-(1-2p_2)(1-2p_3)}{2} = 0.58$ $p_2 = \frac{1-(1-2p_1)(1-2p_3)}{2} = 0.94$ $p_3 = \frac{1-(1-2p_1)(1-2p_2)}{2} = 0.41$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Current iterative decoder output:

$p_0 = 0.84 = f(0.15,0.92,0.73)$ $p_1 = 0.99 = f(0.98,0.58,0.58)$ $p_2 = 0.96 = f(0.74,0.94,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.9: Decoding iteration **2**, equation **1**

Currently known:						
	p_0	p_1	p_2	p_3	p_4	p_5
From channel	0.15	0.98	0.74	0.05	0.05	0.26
equation 0		0.58	0.94	0.41		
equation 1	0.92		0.34		0.67	
equation 2	0.73	0.58				0.33

Decoding iteration 2, equation 1

Input probabilities to SPC decoder:

$p_0 = 0.32 = f(0.15,0.73)$ $p_2 = 0.98 = f(0.74,0.94)$ $p_4 = 0.05 = f(0.05)$

Output extrinsic information:

$p_0 = \frac{1-(1-2p_2)(1-2p_4)}{2} = 0.93$ $p_2 = \frac{1-(1-2p_0)(1-2p_4)}{2} = 0.34$ $p_4 = \frac{1-(1-2p_0)(1-2p_2)}{2} = 0.67$

Received probabilities (iterative decoder input):

$p_0 = 0.15$ $p_1 = 0.98$ $p_2 = 0.74$

Final iterative decoder output:

$p_0 = 0.86 = f(0.15,0.93,0.73)$ $p_1 = 0.99 = f(0.98,0.58,0.58)$ $p_2 = 0.96 = f(0.74,0.94,0.34)$

Decoded data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Transmitted Data: $b_0 = 1$ $b_1 = 1$ $b_2 = 1$

Table 7.10: MAP decoding, FPCES BP/SPA decoding, and RPCES BP/SPA decoding

Decoder Type	Decoder Output [p_0 p_1 p_2]	Decoded information [b_0 b_1 b_2]	Information errors	Source
MAP decoder	[0.85 0.96 0.93]	[1 1 1]	0	Table 2.2
Iterative FPCES BP/SPA decoder	[0.85 0.98 0.94]	[1 1 1]	0	Table 2.11
Iterative RPCES BP/SPA decoder	[0.86 0.99 0.96]	[1 1 1]	0	Table 7.9

From the final decoding results presented in Table 7.10, it can be seen that the three decoders correctly decoded the received information from the channel to the transmitted codeword despite the error caused by the channel, and both iterative decoders have similar performance to the MAP decoder. However, within the three decoding iterations, the RPCES BP/SPA decoder resulted in more saturated output probabilities than the FPCES BP/SPA decoder.

The RPCES BP/SPA decoder was implemented by making a slight modification to the previously implemented standard BP/SPA decoder described in Section 2.5, which is an FPCES BP/SPA decoder. The C programming language code for the RPCES BP/SPA decoder implemented is available in the Appendices (See Appendix N).

Recall, the random number generator function in the C programming language called $ran2$ which was utilized in all the PEG algorithms and the FPCES BP/SPA decoder implemented in this thesis [Teukolsky et al., 1992]. The $ran2$ function returns a uniform random deviate between 0.0 and 1.0 (exclusive of the endpoint values) every time the function is called.

A duplicate $ran2$ function was added to the FPCES BP/SPA decoder and used to randomize the parity-check equation sequence during each decoding iteration executed by the BP/SP algorithm; this slight modification converts a FPCES BP/SPA decoder to a RPCES BP/SPA decoder. The $ran2$ function was duplicated so that the sequence of random deviates produced by the first $ran2$ function which is used for random message and noise pattern generation is not altered. The two $ran2$ function implemented in the RPCES BP/SPA decoder had no variables in common and ran independently of each other. This ensures that when comparing the performance of an LDPC code using the FPCES BP/SPA decoder and the proposed RPCES BP/SPA decoder, identical messages and noise patterns are encountered by the two decoding algorithms.

The proposed RPCES BP/SPA decoder was used to decode the following lowest-error-floor rate-½ $(n, k) = (512, 256)$ and $(n, k) = (1024, 512)$ LDPC codes which were found in this research (see Chapter 6):

i) the rate-½ $(512, 256)$ OED ACE PEG code which has degree distribution $Q_7(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.00391x^{13} + 0.01563x^{14} + 0.00977x^{15} + 0.07422x^{16}$, and

ii) the rate-½ $(1024, 512)$ ACE PEG code with degree distribution $Q_4(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{15}$.

All performance results presented in this section were obtained using a maximum of 100 decoding iterations. Identical noise patterns were used for each set of points at the same E_b/N_o on the error rate curves, and the same numbers of blocks (frames) were used for all codes at each E_b/N_o ratio simulated.

Figure 7.1 shows the performance of the rate-½ $(n, k) = (512, 256)$ OED ACE PEG code when decoded using the standard BP/SPA decoder (FPCES BP/SPA decoder) and the proposed RPCES BP/SPA decoder.

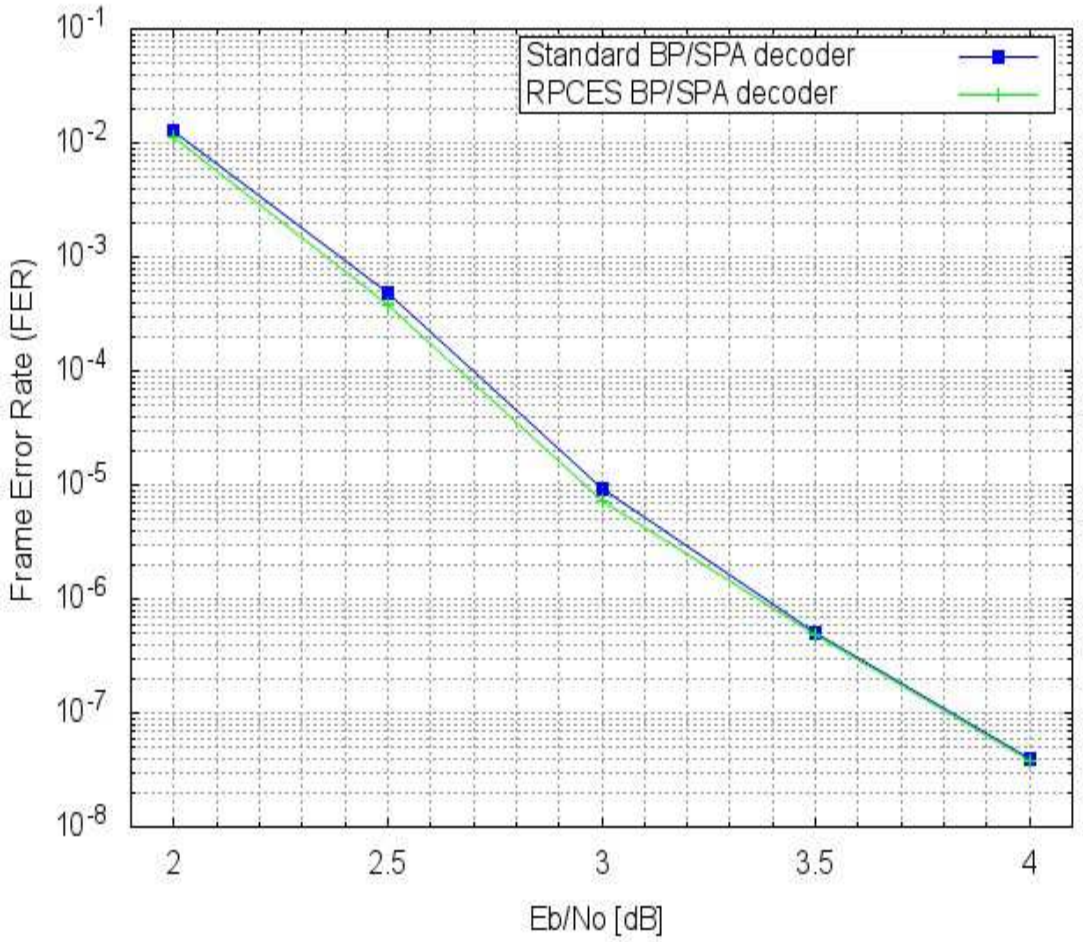

Figure 7.1: FER curves for the lowest-error-floor rate-½ $(n, k) = (512, 256)$ OED ACE PEG code using: i) the standard BP/SPA decoder (FPCES BP/SPA decoder), and
ii) the RPCES BP/SPA decoder

It can be seen from Figure 7.1 that using the RPCES BP/SPA decoder results in a slightly better performance by the rate-½ (512, 256) OED ACE PEG code in both the waterfall and error-floor

regions. However, the performance improvement in the waterfall region appears to be more significant. It was also observed that the decoding convergence time for the RPCES BP/SPA decoder is significantly less than that for the FPCES BP/SPA decoder. Using an identical number of frames, the simulation results obtained from the RPCES BP/SPA decoder were obtained in less than 90% of the time required to obtain the final simulation results from the FPCES BP/SPA decoder (the standard BP/SPA decoder).

Figure 7.2 shows the performance of the rate-½ $(n,\ k) = (1024,\ 512)$ ACE PEG code when decoded using the standard BP/SPA decoder (FPCES BP/SPA decoder) and the proposed RPCES BP/SPA decoder.

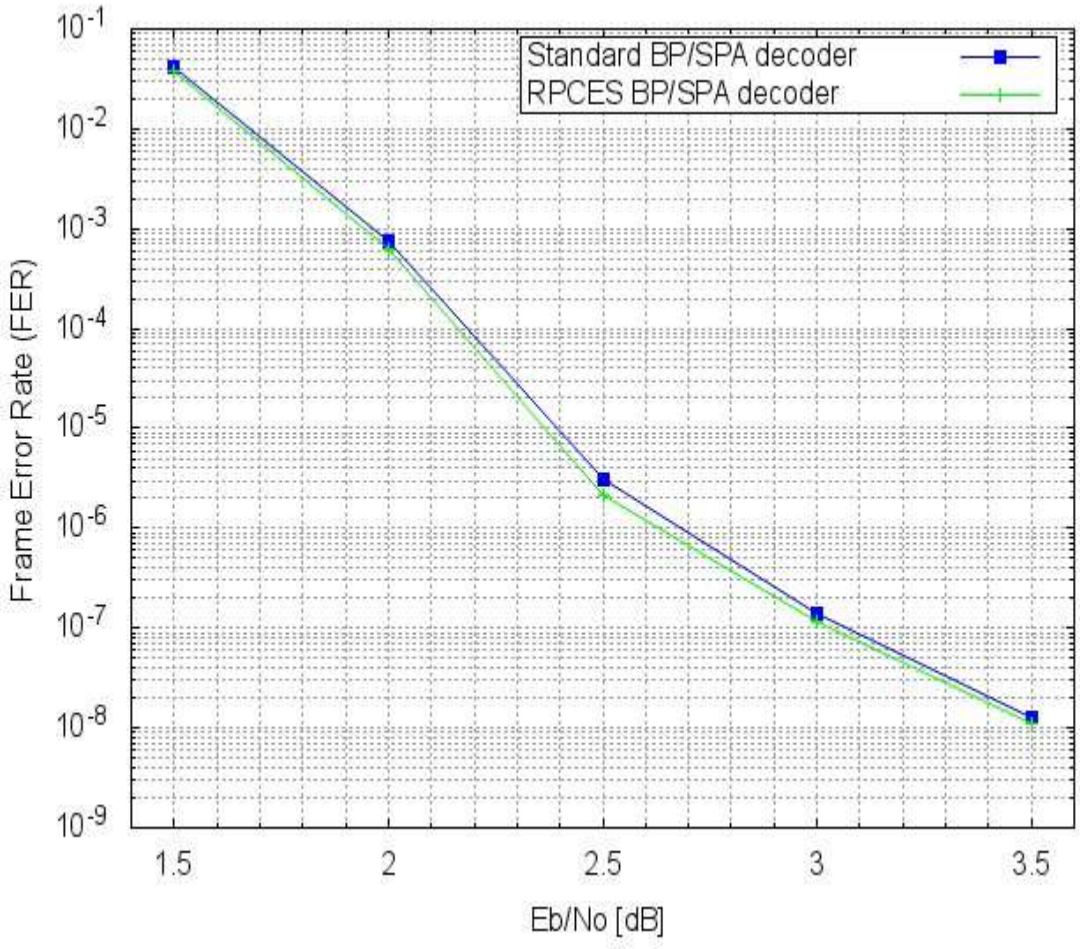

Figure 7.2: FER curves for the lowest-error-floor rate-½ $(n,\ k) = (1024,\ 512)$ ACE PEG code using:
i) the standard BP/SPA decoder (FPCES BP/SPA decoder), and
ii) the RPCES BP/SPA decoder

Similar to the case for the rate-½ $(512, 256)$ OED ACE PEG code, Figure 7.2 shows that using the RPCES BP/SPA decoder results in slightly better performance by the rate-½ $(1024, 512)$ ACE PEG code in both the waterfall and error-floor regions. Given an identical number of codewords (frames), the simulation results obtained from the RPCES BP/SPA decoder were obtained in less than 90% of the time required to obtain simulation results from the FPCES BP/SPA decoder (standard BP/SPA decoder).

7.3 An Improved BP/SPA Decoder: BP/SPA Decoding using a Randomized Parity-Check Equation Sequence and n Detected Decoding Failure Retrials (RPCES + n-DFR)

As expected, it was observed that different decoding performances are obtained when the RPCES BP/SPA decoder uses different series of random parity-check equation sequences for decoding the same LDPC code. Given a reasonably high maximum number of decoding iterations and SNR, most received codeword vectors will be correctly decoded by the RPCES BP/SPA decoder using any series of random decoding equation sequences. However, one of the following situations applies to a small fraction of received codeword vectors.

i) Due to channel distortion and other error-prone graphical structures within LDPC codes such as trapping sets, correct decoding of some received codeword vectors is highly dependent on the series of random decoding equation sequences used in their decoding.

ii) Due to channel distortion, the Euclidean distance between the received codeword and the transmitted codeword is greater than the Euclidean distance between the received codeword and the *more likely* (MRL) codeword which it is eventually decoded to [Papagiannis et al., 2004]. Correct decoding of the received codeword vector is not possible using any series of random or fixed decoding equation sequences.

iii) Due to the channel distortion, there exists no series of random or fixed decoding equation sequences which can result in decoding the received codeword vector to any valid codeword of the code.

A method to take advantage of the RPCES BP/SPA decoder in the event of decoding failures arising from the situation described in i) was incorporated into the RPCES BP/SP algorithm.

After a BP/SP algorithm has executed the maximum number of decoding iterations on a received codeword vector $\boldsymbol{r}$, there are three possible outcomes as follows.

a) A successful decoding; the decoding results in the transmitted codeword, where $\boldsymbol{c} \in C$, and the syndrome of $\boldsymbol{c}$ is,

$$\boldsymbol{s}(\boldsymbol{c}) = \boldsymbol{c}H^T = 0.$$

b) An undetectable decoding error; the decoding results in a codeword $\boldsymbol{c}'' \in C$ which is different from the transmitted codeword $\boldsymbol{c}$, i.e. $\boldsymbol{c}'' \neq \boldsymbol{c}$, and the syndrome of $\boldsymbol{c}''$ is,

$$\boldsymbol{s}(\boldsymbol{c}'') = \boldsymbol{c}''H^T = 0.$$

c) A detectable decoding error; the decoding results in an n bit long binary string $\boldsymbol{v}$ which is not a valid codeword of the code, where $\boldsymbol{v} \notin C$, and the syndrome of $\boldsymbol{v}$ is,

$$\boldsymbol{s}(\boldsymbol{v}) = \boldsymbol{v}H^T > 0.$$

In the case of an undetectable decoding error, although the decoded codeword is not the same as the transmitted codeword, there is no way of knowing this at the decoder because the syndrome of the decoded codeword is 0. However, detectable decoding errors are easily identified by the fact that they have non-zero syndromes; the decoded *n* bit long binary string $\boldsymbol{v}$ is not a codeword of the code.

The implemented RPCES BP/SPA decoder was modified such that, given a received codeword vector $\boldsymbol{r} = [r_0, r_1, r_2 \ldots . r_{n-1}]$, whenever a detectable decoding failure is detected after the maximum number of decoding iterations have been executed, a maximum of $\boldsymbol{n}$ retrials (additional attempts) to decode $\boldsymbol{r}$ using a different series of random decoding equation sequences are made. The proposed

improved BP/SPA (IBP/SPA) decoder is obtained by making this simple modification to the RPCES BP/SPA decoder.

Note that the variable $\boldsymbol{n}$ which is the maximum number of retrials is different from the variable n which is the length of a code. Decoding retrials in the modified RPCES BP/SPA are only made for detectable decoding errors. If a received codeword vector $\boldsymbol{r}$ is decoded to a valid codeword of the code $\boldsymbol{c}$ before the $\boldsymbol{n}$-th retrial, further retrials on $\boldsymbol{r}$ are not attempted and the algorithm proceeds to decode a new codeword vector. If the decoded codeword $\boldsymbol{c'}$ is identical to the transmitted codeword $\boldsymbol{c}$, i.e. $\boldsymbol{c'} = \boldsymbol{c}$, the error count is not increased. Conversely, if the decoded codeword $\boldsymbol{c'}$ is not identical to the transmitted codeword $\boldsymbol{c}$, i.e. $\boldsymbol{c'} \neq \boldsymbol{c}$, then an undetectable error has occurred and the error count is increased. However, if after the $\boldsymbol{n}$-th decoding retrial the modified RPCES BP/SPA decoder fails to decode the received codeword vector to a valid codeword, the error count is increased and the algorithm proceeds to decode a new codeword vector.

The IBP/SPA decoder is also referred to as the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder, i.e. the RPCES BP/SPA decoder with $\boldsymbol{n}$ detected decoding failure retrials ($\boldsymbol{n}$-DFR). The C programming language code for the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder implemented is available in the Appendices (See Appendix O).

In this thesis, only the performance of the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder with maximum number of decoding retrials $\boldsymbol{n} = 10$ and 100 are reported. That is, only the performance of a RPCES + 10-DFR BP/SPA decoder and a RPCES + 100-DFR BP/SPA decoder are reported. This is due to the fact that the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder was invented at a rather late stage of the research.

Figure 7.3 shows the performance of the rate-½ $(n, k) = (512, 256)$ OED ACE PEG code when decoded using the standard BP/SPA decoder (FPCES BP/SPA decoder), the RPCES BP/SPA decoder, the RPCES + 10-DFR BP/SPA decoder, and the RPCES + 100-DFR BP/SPA decoder.

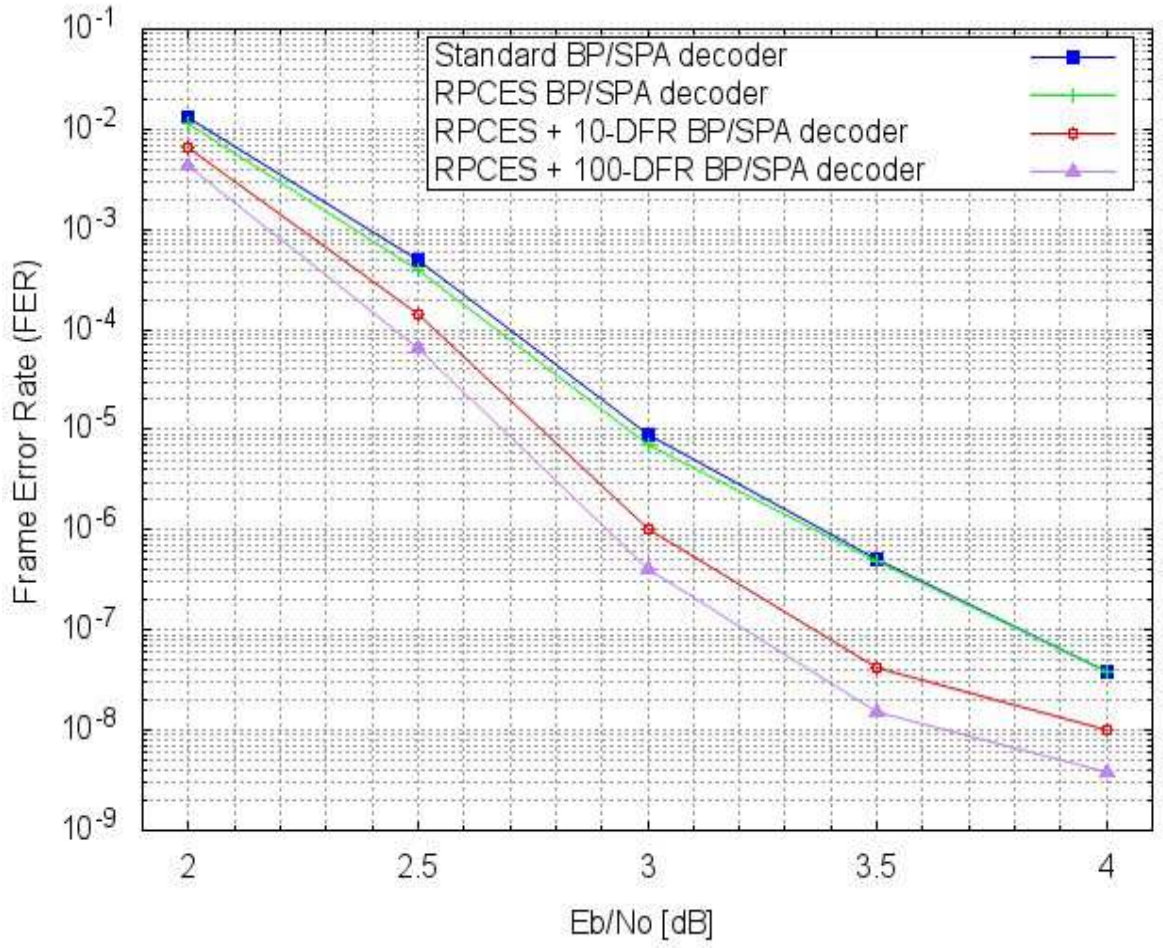

Figure 7.3: FER curves for the lowest-error-floor rate-½ $(n, k) = (512, 256)$ OED ACE PEG code using: i) the standard BP/SPA decoder (FPCES BP/SPA decoder), ii) the RPCES BP/SPA decoder, iii) the RPCES + 10-DFR BP/SPA decoder, and iv) the RPCES + 100-DFR BP/SPA decoder

As shown in Figure 7.3, using the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders results in superior performance by the rate-½ $(512, 256)$ OED ACE PEG code in both the waterfall and error-floor regions. As a result of a high frequency of detectable decoding failures and the attendant decoding retrials in the waterfall region, obtaining the improved FER performance of LDPC codes in the waterfall region using the two RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders require significantly longer decoding times

compared to the RPCES BP/SPA decoder and the standard BP/SPA decoder. Naturally, the higher the maximum number of decoding retrials $\boldsymbol{n}$, the longer the decoding performance simulation time; the RPCES + 100-DFR BP/SPA decoder takes a longer time than the the RPCES + 10-DFR BP/SPA decoder. However, in the error-floor region where there are very rare occurrences of decoding retrials due to detectable decoding failures, using RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders to obtain the improved FER performance of LDPC codes requires only slightly more time compared to the RPCES BP/SPA decoder. The RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders (with $\boldsymbol{n}$ = 10 and 100) were significantly faster than the standard BP/SPA decoder when used in simulations to determine code performance in the error-floor region. This is because in the error-floor region, the fast decoding convergence of the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders more than compensate for the occasional delays due to decoding retrials.

As expected, it is clear from Figure 7.3 that increasing the maximum number of decoding retrials, $\boldsymbol{n}$, in the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder results in an improved decoding performance. However, as is to be expected, the rate at which the decoding performance of the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder improves with increasing $\boldsymbol{n}$ diminishes as $\boldsymbol{n} \rightarrow \infty$.

Figure 7.4 shows the performance of the rate-½ $(n, k) = (1024, 512)$ ACE PEG code when decoded using the standard BP/SPA decoder (FPCES BP/SPA decoder), the RPCES BP/SPA decoder, the RPCES + 10-DFR BP/SPA decoder, and the RPCES + 100-DFR BP/SPA decoder.

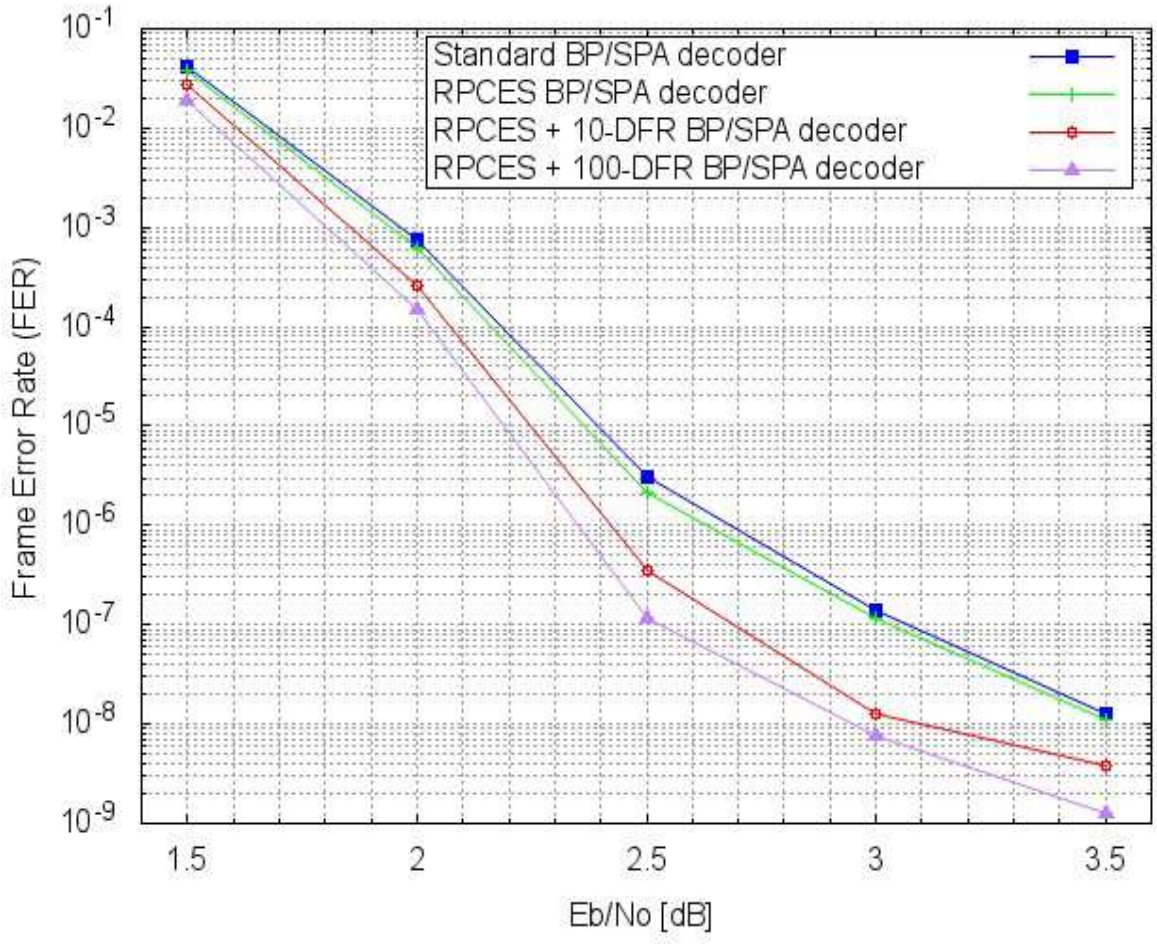

Figure 7.4: FER curves for the lowest-error-floor rate-½ $(n, k) = (1024, 512)$ ACE PEG code using: i) the standard BP/SPA decoder (FPCES BP/SPA decoder), ii) the RPCES BP/SPA decoder, iii) the RPCES + 10-DFR BP/SPA decoder, and iv) the RPCES + 100-DFR BP/SPA decoder

Similar to the case for the rate-½ $(512, 256)$ OED ACE PEG code, Figure 7.4 shows that the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders with $\boldsymbol{n} = 10$ and 100 results in superior performance by the rate-½ $(1024, 512)$ ACE PEG code in both the waterfall and error-floor regions. The decoding times in the waterfall and error-floor differ in the same manner as described for decoding the rate-½ $(512, 256)$ OED ACE PEG code.

Table 7.11 shows the distribution of the type of decoding errors that occurred when the RPCES + 10-DFR BP/SPA decoder and the RPCES + 100-DFR BP/SPA decoder were used to decode the lowest-error-floor rate-½ $(n, k) = (512, 256)$ code. Similarly, Table 7.12 shows the distribution of the type of decoding errors that occurred when the RPCES + 10-DFR BP/SPA decoder and the RPCES + 100-DFR BP/SPA decoder were used to decode the lowest-error-floor rate-½ $(n, k) = (1024, 512)$ code.

For the decoding error distributions in Table 7.11 and Table 7.12, decoding errors were classified under two broad types: Hamming distance errors and trapping set errors. Hamming distance errors occur when the received codebit vectors are decoded to a valid codeword which is different from the codeword that was sent. Hamming distance errors include minimum distance errors, i.e. d_{min} errors, where the Hamming distance between the sent codeword and the decoded codeword is equal to the d_{min} of the code. Hamming distance errors have zero syndromes and are therefore undetectable errors. All decoding errors which have non-zero syndromes, or detectable errors, are considered to be as a result of trapping sets, and are classified as trapping set errors.

Table 7.11: Decoding error distributions for the lowest-error-floor rate-½ $(n, k) = (512, 256)$ OED ACE PEG code using the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder with $\boldsymbol{n}$ = 10 and 100

	Decoding error distributions at different Eb/No (in percentages)									
	$2.00dB$		$2.50dB$		$3.00dB$		$3.50dB$		$4.00dB$	
Type of IBP/SPA decoder	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors
RPCES + 10-DFR	0.06%	99.94%	0.36%	99.64%	2.00%	98.00%	0.00%	100.00%	0.00%	100.00%
RPCES + 100-DFR	0.07%	99.93%	0.47%	99.53%	3.00%	97.00%	0.00%	100.00%	0.00%	100.00%

Table 7.12: Decoding error distributions for the lowest-error-floor rate-½ $(n, k) = (1024, 512)$ ACE PEG code using the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder with $\boldsymbol{n}$ = 10 and 100

	Decoding error distributions at different Eb/No (in percentages)									
	$1.50dB$		$2.00dB$		$2.50dB$		$3.00dB$		$3.50dB$	
Type of IBP/SPA decoder	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors	Hamming distance errors	Trapping set errors
RPCES + 10-DFR	0.004%	99.996%	0.076%	99.924%	0.00%	100.00%	0.00%	100.00%	0.00%	100.00%
RPCES + 100-DFR	0.00%	100.00%	0.00%	100.00%	0.00%	100.00%	1.00%	99.00%	0.00%	100.00%

It can be observed from Table 7.11 and Table 7.12 that the highest percentage of decoding failures are due to trapping sets. In fact, irrespective of the Eb/No and/or the decoder used, a minimum of 99% of the decoding errors were caused by trapping sets. It is therefore obvious from the results in Table 7.11 and Table 7.12 that trapping sets are the main obstacle to improving the performance of short-to-medium length LDPC codes decoded using iterative message-passing algorithms and reducing their gap to the ultimate Shannon limit over the AWGN channel. Additionaly, as can be observed from Table 7.11 and Table 7.12, the maximum number of decoding failure retrials, i.e. $\boldsymbol{n}$**,** has no discernible effect on the distributions of the type of decoding errors.

The generalized improvement in the decoding performance of short-to-medium length LDPC codes achieved using the proposed RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder are significantly better than the performance improvements published for check node removal and collaborative decoding as proposed in [Kang et al., 2016], and the shuffled iterative decoding [Zhang and Fossorier, 2005] which was used to decode the rate-½ $(n, k) = (1000, 500)$ LDPC codes in Fig. 11 of [Richter and Hof, 2006]. In each decoding iteration, the RPCES BP/SP algorithm cycles through the equations in the probability matrix H_P in a randomized sequence. This implies that for a code with a large number of parity-check equations m, the probability of repeating the same decoding equation sequence while iteratively decoding a received codeword vector is extremely small ($\cong 0$). The randomized equation sequence

can be seen as a technique for breaking up dominant trapping sets using the complete set of parity-check nodes. This technique for breaking up trapping sets is very similar to the collaborative decoding described in [Kang et al., 2016]. However, collaborative decoding has the drawback of using only a subset of the parity-check nodes in the sub-decoding mode. This undoubtedly results in the loss of information about the relationship between the sets of symbol node in the excluded parity-check nodes during decoding iterations in the sub-decoding mode. In the two proposed RPCES BP/SPA decoders, the complete set of parity-check nodes are used in each decoding iteration and there is no loss of parity-check information in any decoding iteration. The proposed RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder significantly improves the decoding performance of all types of short-to-medium length LDPC codes.

7.4 Summary

- Two novel modifications to the standard BP/SPA decoder, the RPCES BP/SPA decoder and the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder, have resulted in an unprecedented performance improvement to BP/SPA decoding over the AWGN channel. The RPCES BP/SPA decoder implements a randomized probability matrix H_P parity-check equation sequence in each decoding iteration.

- Compared to the standard BP/SPA decoder, the RPCES BP/SPA decoder results in improved LDPC code performance in both the waterfall and error-floor regions, and the decoding convergence time for the RPCES BP/SPA decoder is significantly less than that for the standard BP/SPA decoder (the FPCES BP/SPA decoder).

- The use of different series of random decoding equation sequences in the RPCES BP/SPA decoder results in different decoding outcomes. Consequently, the probability of decoding a received vector to a codeword (even if its the wrong one) is increased by trying different series of random decoding equations for its decoding. The RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder augments the RPCES BP/SPA decoder with the ability to reattempt each unsuccessful decoding up to $\boldsymbol{n}$ times.

- Compared to using the standard BP/SPA decoder and the RPCES BP/SPA decoder, the RPCES + $\boldsymbol{n}$-BP/SPA decoder results in an unprecedented improvement to the performance of short-to-medium length irregular LDPC codes in the waterfall and error-floor regions. The RPCES + $\boldsymbol{n}$-BP/SPA decoder is also referred to as an improved BP/SPA (IBP/SPA) decoder.

- The modifications made to the conventional BP/SPA decoding process to obtain the IBP/SPA decoder results in an efficient and effective technique for breaking trapping sets in LDPC codes.

- Given the IBP/SPA decoder's capacity for breaking up trapping sets, it is clear that the decoding performance of IBP/SPA decoders (RPCES + $\boldsymbol{n}$-DFR BP/SPA decoders) improves with increasing values of $\boldsymbol{n}$. However, the rate at which the decoding performance of the RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder improves with increasing $\boldsymbol{n}$ diminishes as $\boldsymbol{n} \rightarrow \infty$.

Chapter 8

Modifying the BP/SPA decoder for Slepian-Wolf coding

Parts of this chapter have been presented in the following conference: Eleruja, S., Abdu-Aguye, U. F., Ambroze, M., Tomlinson, M., & Zaki, M. (2017, November). Design of binary LDPC codes for Slepian-Wolf coding of correlated information sources. *5th IEEE Global Conference on Signal and Information Processing*, 14-16 November, 2017, Montreal, Canada. IEEE.

8.1 Introduction

Slepian-Wolf (SW) coding is a method of coding two lossless compressed correlated (i.e. statistically dependent) sources. Correlated information sources currently abound in practical communication systems. Information generated and transmitted by correlated sources usually conform to identical protocols which implies that certain information in transmitted frames/blocks will be common to the sources. Common information from different sources can be carefully managed to avoid duplication and achieve optimal channel capacity utilization.

Slepian and Wolf laid the foundation of distributed source coding (DSC) in 1973, when they proved the counter-intuitive result that separate encoding with joint decoding achieves the same compression rate as that of joint encoding [Slepian and Wolf, 1973]. DSC problems pertain to the compression of multiple correlated information sources that do not communicate with each other and therefore are encoded in a distributed manner. By modeling the correlation between multiple sources at the decoder side with channel codes, DSC shifts the computational complexity from the encoder side to the decoder side, which provides appropriate frameworks for applications with complexity-constrained

sender, such as sensor networks and video/multimedia compression. One of the main properties of distributed source coding is that the computational burden in encoders is shifted to the joint decoder. Over the past four decades, communications has shifted from point-to-point communication to network communication with multiple senders and receivers. The emergence of these distributed systems has increased the requirement for more energy efficient and effective use of limited resources such as bandwidth and power.

Slepian and Wolf showed that multiple sources can be jointly compressed at a rate greater than the sum of their respective rates if compressed separately [Slepian and Wolf, 1973]. Given a signal X_1 which is transmitted conventionally at full rate of $R_{X_1} = H(X_1)$ and is faithfully recovered at the decoder, while the correlated signal X_2 is compressed as close as possible to the SW limit $H(X_2|X_1)$. The SW rate region for two correlated sources X_1 and X_2 is bounded by the following inequalities,

$$R_{X_1} \geq H(X_1|X_2) \tag{8.1}$$

$$R_{X_2} \geq H(X_2|X_1) \tag{8.2}$$

$$R_{X_1} + R_{X_2} \geq H(X_1, X_2) \tag{8.3}$$

If the encoder and decoder of the two sources are independent, the lowest rate that can be achieved for lossless compression is $H(X_1)$ and $H(X_2)$ for X_1 and X_2 respectively, where $H(X_1)$ and $H(X_2)$ are the entropies of X_1 and X_2. However, the Slepian–Wolf theorem shows that much better compression rate can be achieved with joint decoding. As long as the total rate of X_1 and X_2 is larger than their joint entropy $H(X_1, X_2)$ and none of the sources is encoded with a rate larger than its entropy, distributed coding can achieve arbitrarily small error probability for long sequences. In other words, the two isolated sources can compress data as efficiently as if they were communicating with each other.

A special case of distributed coding is compression with decoder side information, where source X_2 is available at the decoder side but not accessible at the encoder side. This can be treated as the condition that $R_{X_2} = H(X_2)$ has already been used to encode X_2, while we intend to use $H(X_1/X_2)$ to encode X_1. The whole system is operating in an asymmetric way, i.e. the compression rate for the two sources are

asymmetric ['Distributed source coding', 2017], ['Slepian-Wolf coding', 2017]. Slepian-Wolf coding entails lossless coding of a source with the help of side information available at the decoder only. [Liveris et al., 2002] and [Schonberg et al., 2004] showed that distributed source coding based on LDPC codes results in better performance than those based on turbo codes.

This chapter describes the modifications made to the implemented standard BP/SPA decoder, which is described in detail in Section 2.5, to study distributed source coding of two correlated sources for asymmetric SW coding using binary LDPC codes. The performance of PEG algorithm constructed binary LDPC codes of short-to-medium length using conventional channel models are compared to their performance using SW channel models. More details of the investigations carried out and the results obtained can be found in [Eleruja et al., 2017].

8.2 Modifying the BP/SPA Decoder for Slepian-Wolf Coding of Correlated Information Sources

In this section, a description of the necessary modifications to the BP/SPA decoder in order to implement the Slepian-Wolf channel model for the coding of correlated information sources are given. To obtain the best understanding of the descriptions that follow, the reader should first be familiar with the BP/SPA decoder implementation which is explained in detail in Section 2.5.

Let X and Y be two correlated binary information sources. Define the correlation between source X and source Y as: Y is noisy version of X. Source Y can be obtained by adding AWGN vectors at an arbitrary SNR (or noise standard variation, σ_N) to BPSK bit vectors of source X and then quantizing the result to obtain the bit string for correlated source Y. Alternatively, Y may be considered to be obtained as a result of passing X through a BSC with an unknown bit crossover probability p.

$$Y = \begin{cases} X, & with\ probability\ 1-p \\ X \oplus 1, & with\ probability\ p \end{cases} \tag{8.4}$$

In either case, sources X and Y are considered to be correlated because they differ in only a few bit positions. Unlike in classical coding, information from correlated sources X and Y are not necessarily

codewords $\boldsymbol{c} \in C$, and include all bit strings $\boldsymbol{v} \in V$ of length n, so that $C \subset V$ and $\boldsymbol{c} \in V$. Let $\boldsymbol{v}$ be an n bit long bit string from source X or Y, where $\boldsymbol{v} = [v_0 \ v_1 \ v_2 \ldots . \ v_{n-1}]$. Given a parity-check matrix H, the syndrome $\boldsymbol{s}$ of any bit string $\boldsymbol{v}$ from source X or Y is given by,

$$\boldsymbol{s} = \boldsymbol{v}H^T \begin{cases} = 0, & v = c \\ \geq 1, & v \neq c \end{cases} \tag{8.5}$$

In classical coding, only the codewords $\boldsymbol{c} = [c_0 \ c_1 \ c_2 \ldots . \ c_{n-1}]$ of a code defined by an H matrix are transmitted, and the syndromes of all codewords are zero, i.e.,

$$\boldsymbol{s} = \boldsymbol{c}H^T = 0 \tag{8.6}$$

Consequently, the classical BP/SPA decoder is built around the expectation that all binary vectors received from the channel $\boldsymbol{c}' = [c'_0 \ c'_1 \ c'_2 \ldots . \ c'_{n-1}]$ when correctly decoded to a codeword $\boldsymbol{c} \in C$ would have zero syndromes.

The syndrome of any n bit long bit string $\boldsymbol{v}$ is obtained using $\boldsymbol{s} = \boldsymbol{v}H^T$, and the transposed H matrix H^T has dimensions $n \times m$. Consequently, the syndrome of $\boldsymbol{v}$ is also m bits long.

Consider an LDPC code parity-check matrix of dimension $m \times n$, where n is the codeword length, k is the number of information bits, and m is the number of parity-check bits. It follows that $n = k + m$.

If the code rate $R =$ ½, then $R = \frac{k}{n} = \frac{n-m}{n} = \frac{1}{2} \ \Rightarrow \ m = \frac{n}{2}$. Therefore, for a rate-½ code the syndrome $\boldsymbol{s}$ of each n bit long bit string $\boldsymbol{v}$ will be $\frac{n}{2}$ bits long. In other words, for the H matrix of a full rank rate-½ code, the syndrome of messages from a source are half the lengths of the original messages from the source.

At any code rate, whether the H matrix is of full rank or not, the syndrome $\boldsymbol{s}$ of a n bit long bit string $\boldsymbol{v}$ can be represented as $\boldsymbol{s} = [s_0 \ s_1 \ s_2 \ldots . \ s_{m-1}]$. The syndrome $\boldsymbol{s}$ of every n bit long bit string $\boldsymbol{v}$ is made up of a set of m bits $\{s_i\}$ for all $0 \leq i \leq m-1$. There are as many syndrome bits as there are parity-check equations in a H matrix, and each syndrome bit s_i corresponds to the parity-check

equation with index i. That is, given the syndrome $\boldsymbol{s}$ of an arbitrary n bit long bit string $\boldsymbol{v}$, bit s_0 of $\boldsymbol{s}$ verifies if parity-check equation 0 is satisfied by $\boldsymbol{v}$, bit s_1 verifies if parity-check equation 1 is satisfied by $\boldsymbol{v}$, and so on. The last bit s_{m-1} verifies if parity-check equation $(m-1)$ is satisfied by $\boldsymbol{v}$. These m parity-check equation verifications are done even if there are some linearly dependent parity-check equations and the actual parity-length of a code is less than the number of parity-check equations in the H matrix.

If only codewords of an H matrix are transmitted, the decoder expects that the syndrome of any correctly decoded codeword $\boldsymbol{c}$ would be equal to 0. That is, $\boldsymbol{s} = \boldsymbol{c}H^T = 0$, because all the syndrome bits s_i are equal to 0,

$$\boldsymbol{s} = \sum_{i=0}^{m-1} s_i = 0 \quad \text{where } s_i = 0, \ \forall \ 0 \leq i \leq m-1 \tag{8.7}$$

This implies that for codewords, the sum of the symbol node bits in each parity-check equation is even. However, this is not always so when n bit long arbitrary bit strings $\boldsymbol{v}$ are transmitted from sources.

For an arbitrary n bit long bit string $\boldsymbol{v}$ where $\boldsymbol{v} \notin C$, the syndromes $\boldsymbol{s}$ are more than zero. That is, $\boldsymbol{s} = \boldsymbol{v}H^T \geq 1$, because at least one of the m bits of the syndrome of $\boldsymbol{v}$ is a 1,

$$\boldsymbol{s} = \sum_{i=0}^{m-1} s_i \geq 1 \quad \text{where } 0 \leq i \leq m-1, a, b \in i, |b| \geq 1, |a| + |b| = |i| = m, s_a = 0, \text{ and } s_b = 1 \tag{8.8}$$

This implies that for these n bit long bit strings $\boldsymbol{v} \notin C$, the correct sum of the symbol node bits involved in at least one of the parity-check equations is expected to be a 1.

In an ideal implementation of the SW coding model for distributed sources, the syndromes of the noise-free source X, and the n bit long vectors of correlated source Y (which is a quantized noisy

version of X) will be transmitted separately over a noiseless channel to the decoder. Compression is achieved by transmitting only the syndrome bits of one of the sources (i.e., X) instead of all the bits of the source.

8.2.1 Modifying the BP/SPA Decoder for Syndrome Decoding of Correlated Information Sources over the AWGN Channel: A SW-Like Coding Model

This section explains how the standard BP/SPA decoder implemented in this research was modified to implement syndrome decoding of correlated information sources over the AWGN channel (BP/SPA syndrome decoder). The implementation of the standard BP/PSA decoder is described in Section 2.5. The C programming language code for the BP/SPA syndrome decoder implemented in this research is available in the Appendices (See Appendix L).

Unlike conventional channel coding, correlated sources X and Y are not restricted to transmitting only codewords, and the transmitted messages can be seen as arbitrary n bit strings $\boldsymbol{v}$. Therefore, there is no need for a G matrix (generator matrix) for encoding messages from information sources in SW channels; the BP/SPA syndrome decoder is built around the SW coding model. Consequently, all the Gaussian elimination operations for obtaining the G matrix corresponding to the H matrix to be used for decoding were removed from the BP/SPA decoder implemented for the conventional channel. Instead, an n bit long random bit string $\boldsymbol{v}$ is generated using the same procedure which was originally used to generate random binary k bit long message bit strings $\boldsymbol{u} = [u_0 \ u_1 \ . \ . \ . \ u_{k-1}]$. An n bit long random bit strings generated in this way can be represented as,

$$\boldsymbol{v} = [v_0 \ v_1 \ v_2 \ . \ . \ . \ v_{n-1}]$$

where each $\boldsymbol{v}$ generated in this manner is a complete information frame from SW information source X (and Y). After a random message is generated at source X, its syndrome is calculated using equation (8.5). The syndrome of a message from source X can be denoted by,

$$\boldsymbol{s} = [s_0 \ s_1 \ s_2 \ . \ . \ . \ s_{m-1}]$$

For rate-½ codes, the syndromes of messages from source X are only half the length of the complete message from the source, i.e. $m = {}^{n}/_{2}$. Instead of the complete n bit long message from source X, only its ${}^{n}/_{2}$ bit long syndrome is transmitted noiselessly to the BP/SPA syndrome decoder. The syndrome is the side information required at the BP/SPA syndrome decoder in order to decode the message transmitted from the other source (source Y) at any particular instant.

In order to generate messages for transmission from information source Y which are correlated to those from source X, AWGN is added to the BPSK modulated representation of the message from source X in proportion to the required SNR in the same manner as in the conventional BP/SPA decoder, so that if the resultant BPSK signals were quantized back into bits, a message from source Y will differ from that from source X in a few bit positions. That is, the information from source Y and source X will be correlated. It is important to note that in the SW coding model, the ideal justification for the correlation between two sources requires both sources to be in the same form. The BP/SPA syndrome decoder is therefore not a proper SW coding model due to the deficiency in defining the correlation between the sources. The proposed BP/SPA syndrome decoding arrangement is used only to introduce the concept of syndrome decoding.

Assuming the received noisy vector $\boldsymbol{r}$ which represents source Y is,

$$\boldsymbol{r} = [-1.4\ -0.1\ +1.3\ -1.1\ -0.6\ +1.2\ +1.1\ -0.2\ -1.4\ -0.8\ -1.2\ +1.3],$$

the probabilities that each of these received vectors are equal to bit 1 can be calculated using equation (2.17) as follows,

$$p_i^{1o} = \frac{1}{1 + e^{\frac{-2r_i}{\sigma^2}}}$$

Assuming that $SNR = 1/\sigma^2 = 1.239$ (equal to $SNR(dB) = 0.931$), the probabilities that the received vectors of $\boldsymbol{r}$ are equal to bit 1 are as follows,

$$P = [0.03 \quad 0.44 \quad 0.96 \quad 0.06 \quad 0.18 \quad 0.95 \quad 0.94 \quad 0.38 \quad 0.03 \quad 0.12 \quad 0.05 \quad 0.96]$$

These received bit probabilities are the required inputs from the AWGN channel to the BP/SP algorithm. Subsequently, using these bit probabilities, the BP/SP algorithm is used to decode the noisy BPSK modulated message from source Y using the noiseless side information received from source X regarding the correct values of the set of syndrome bits $\{s_i\}\ \forall\ 0 \leq i \leq m-1$ for each n bit long message vector $\boldsymbol{v}$ transmitted.

The following is a description of the modification to the BP/SP algorithm in order to utilize side information. In the conventional BP/SPA decoder for the AWGN channel, only codewords of the H matrix used for decoding are transmitted. The bit probability calculation method which was adopted in the implemented BP/SPA decoder is $p_i^{1\,e}$ - '*the probability that a received bit is a* 1', and equation (2.33) was used to calculate the extrinsic information for every parity-check equation in each decoding iteration. Equation (2.33) is as follows,

$$p_i^{1\,e} = \frac{1 - \prod_{j \neq i}(1 - 2p_j)}{2}$$

However, equation (2.33) is used to evaluate extrinsic information for every equation in the probability matrix H_P of the conventional BP/SPA decoder because the correct syndrome bit value corresponding to each parity-check equations is 0 for codewords. The correlated information sources in the BP/SPA syndrome decoder (a SW-like coding model), i.e. source X and source Y, are not restricted to transmitting only codewords, and the correct syndrome bit value for each parity-check equation is determined from the side information obtained from source X. Therefore, a dynamic 'side-information-dependant' system for calculating extrinsic information must be adopted. Consequently, in each decoding iteration, when calculating the extrinsic information for parity-check equations for which the correct syndrome bit value is 0, equation (2.33) is used as usual, and when calculating the extrinsic information for parity-check equations for which the correct syndrome bit value is 1, equation (2.32) is used instead. Equation (2.32) is as follows,

$$p_i^{0\,e} = \frac{1 + \prod_{j \neq i}(1 - 2p_j)}{2}$$

The side information is used to determine whether equation (2.32) or equation (2.33) should be used to calculate the extrinsic information for an equation in the probability matrix H_P. This decision depends on the value of the syndrome bit corresponding to the parity-check equation being evaluated at any instant in a decoding iteration. All other aspects of the BP/SP algorithm in the conventional BP/SPA decoder and the BP/SP algorithm in the BP/SPA syndrome decoder are identical.

Using the rate-½ $(n, k) = (1024, 512)$ benchmark code with degree distribution $Q_9(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}$, the following two coding models were compared,

i) Conventional BP/SPA decoder: Codeword transmissions in the AWGN channel from a single information source, and decoded using the standard BP/SPA decoder.

ii) BP/SPA syndrome decoder: Transmissions from two correlated information sources X and Y, and decoded using the BP/SPA syndrome decoder.

Information compression is achieved in the BP/SPA syndrome decoder of ii) by using,

a) only the syndromes of messages from source X (instead of the the full message) received noiselessly, which is used as the side information for the modified BP/SP algorithm in the BP/SPA syndrome decoder, and

b) the full messages from source Y which is to be decoded by the modified BP/SP algorithm.

Figure 8.1 shows the simulated frame error rate (FER) performance results obtained for the two coding models that were compared.

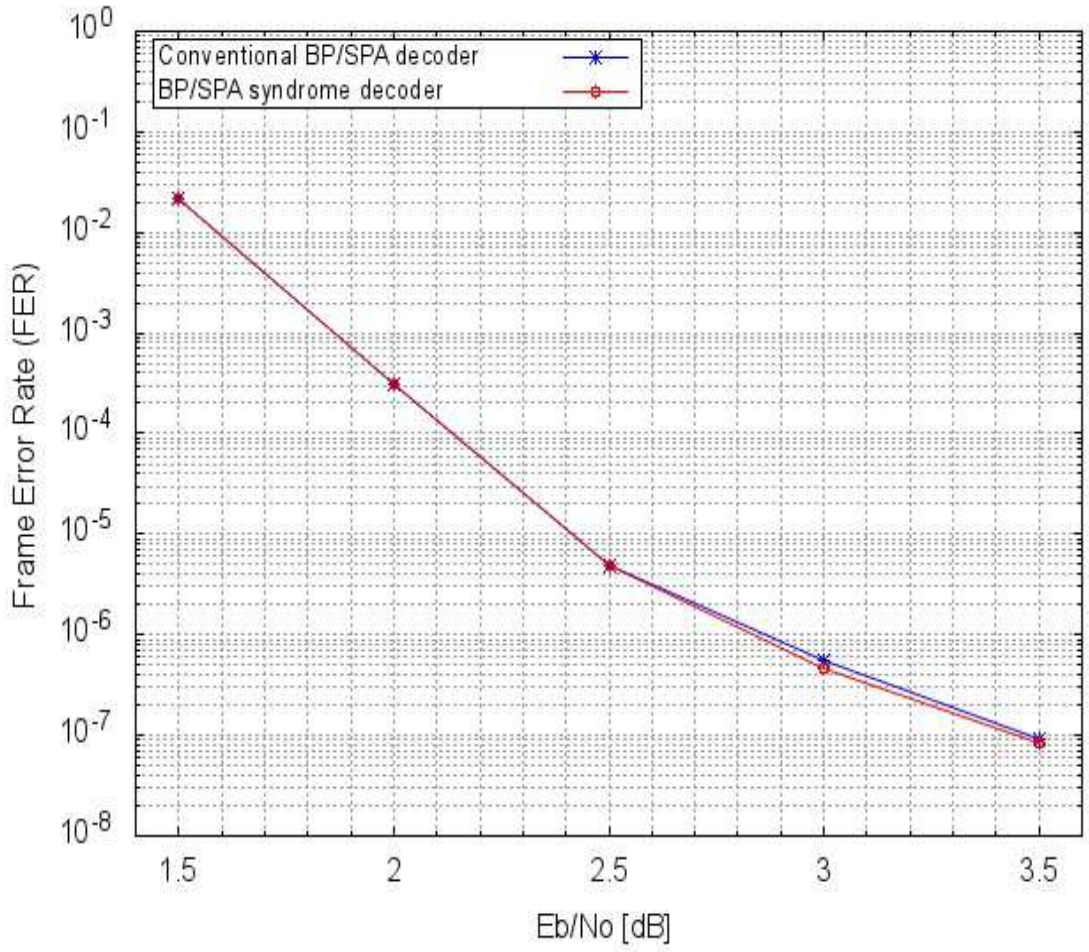

Figure 8.1: Conventional BP/SPA decoder vs. the BP/SPA syndrome decoder using an LDPC code

From the results plotted in Figure 8.1, it can be seen that the BP/SPA syndrome decoder (the SW-like coding model) has a very similar performance to that of the conventional coding model. Therefore, compared to the conventional BP/SPA decoder, the BP/SPA syndrome decoder achieved a very good compression with no apparent loss in performance.

8.2.2 Modifying the BP/SPA Decoder for Slepian-Wolf Coding of Correlated Information Sources over the BSC

This section explains how the standard BP/SPA decoder implemented in this research was modified to implement the proposed SW coding of correlated information sources over the BSC (SW BSC decoder). However, due to the similarities between the algorithm for syndrome decoding of correlated information sources over the AWGN channel (BP/SPA syndrome decoder), as presented in

Section 8.2.1, and the SW coding of correlated information sources over the BSC (SW BSC decoder) which is proposed in this section, emphasis is placed only on the differences between the two BP/SPA decoder modifications. The C programming language code for the SW BSC decoder implemented is available in the Appendices (See Appendix M).

As discussed in Section 8.2.1, there is no need for a G matrix (generator matrix) for encoding messages from information sources in SW coding. Consequently, all the Gaussian elimination operations for obtaining the G matrix corresponding to the H matrix used for decoding were removed from the BP/SPA decoder which was implemented for the conventional channel. Instead, an n bit long random bit string $\boldsymbol{v}$ is generated using the same procedure originally used to generate random binary k bit long message bit strings, i.e. $\boldsymbol{u} = [u_0 \ u_1 \ u_2 \ . \ . \ . \ u_{k-1}]$. So that,

$$\boldsymbol{v} = [v_0 \ v_1 \ v_2 \ . \ . \ . \ v_{n-1}]$$

Where $\boldsymbol{v}$ represents a complete message from SW information source X (and Y). After each random message is generated, its syndrome is calculated using equation (8.5). The syndrome of a message from source X can be denoted by,

$$\boldsymbol{s} = [s_0 \ s_1 \ s_2 \ . \ . \ . \ s_{m-1}].$$

Instead of the complete message from source X, only its syndrome is transmitted noiselessly to the SW BSC decoder. The syndrome is the side information required at the SW BSC decoder in order to decode the message transmitted from the other source (source Y) at any particular instant.

In order to generate the messages to be transmitted from information source Y which are correlated to those from source X, AWGN is added to the BPSK modulated representation of the message from source X in proportion to the required SNR in the same manner as in the conventional BP/SPA decoder. However, unlike in the BP/SPA syndrome decoder, the resultant BPSK signals obtained after AWGN is added to the BPSK modulated representation of the message from source X are quantized back into bits, and the resultant binary message from source Y differs from the message from source X

in a few bit positions (depending on the SNR). For the proposed SW BSC decoder, the information from source Y and source X are in the same form (bits). Therefore, the correlation between the two information sources is ideal.

Borrowing from the example in Section 2.5, assume that a twelve (12) bit long binary string $\boldsymbol{v_X}$ is generated as the binary information from source X, where

$$\boldsymbol{v_X} = [0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$$

The BPSK modulated vector representation of $\boldsymbol{v}$ will be represented by $\boldsymbol{v_{Xm}}$.

Therefore,

$$\boldsymbol{v_{Xm}} = [-1.0 \ +1.0 \ +1.0 \ -1.0 \ -1.0 \ +1.0 \ +1.0 \ -1.0 \ -1.0 \ -1.0 \ -1.0 \ +1.0]$$

Let us assume that the following AWGN vector string was obtained using twelve (12) successive calls to the $gasdev$ function and then multiplying each deviate returned by the noise standard deviation σ for an arbitrary SNR(dB):

$$\boldsymbol{N} = [-0.4 \ -1.1 \ +0.3 \ -0.1 \ +0.4 \ +0.2 \ +0.1 \ +0.8 \ -0.4 \ +0.2 \ -0.2 \ +0.3]$$

In order to obtain correlated information source Y, AWGN vector $\boldsymbol{N}$ is added to the BPSK modulated information from source X, i.e. $\boldsymbol{v_{Xm}}$, to obtain a noisy BPSK modulated information source Y, i.e. $\boldsymbol{v_{Ym'}}$, as follows:

$$\boldsymbol{v_{Ym'}} = \boldsymbol{v_{Xm}} + \boldsymbol{N}, \tag{8.9}$$

This addition is carried out in the component-by-component basis to obtain the received vector $\boldsymbol{v_{Ym'}}$ as:

$$\boldsymbol{v_{Ym'}} = [-1.4 \ -0.1 \ +1.3 \ -1.1 \ -0.6 \ +1.2 \ +1.1 \ -0.2 \ -1.4 \ -0.8 \ -1.2 \ +1.3]$$

It can be observed that the magnitude of the AWGN noise in the second code bit position of $\boldsymbol{N}$ (i.e. N_1) was sufficient to change the polarity (phase of a BPSK modulation) of $v_{Ym'_1}$ compared to that

of v_{Xm_1}. The noisy vector $\boldsymbol{v_{Ym'}}$ in the second bit position is given by $v_{Ym_1'} = v_{Xm_1} + N_1$, where $v_{Xm_1} = +1.0$ and $N_1 = -1.1$. Therefore, $v_{Ym_1'} = -0.1$ while the value of bit vector $v_{Xm_1} = +1.0$.

In order to obtain source Y in the same form as source X, i.e. as a binary string, $\boldsymbol{v_{Ym'}}$ is quantized back to bits to obtain $\boldsymbol{v_Y}$ as,

$$\boldsymbol{v_Y} = [0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$$

Recall that,

$$\boldsymbol{v_X} = [0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$$

Comparing the message from information source Y, i.e. $\boldsymbol{v_Y}$, to the message from information source X, i.e. $\boldsymbol{v_X}$, reveals that the two messages differ in one bit position. Therefore, the described process has successfully generated two correlated information sources. That is, source Y and source X are correlated.

Due to received vector quantization in BSC decoding, the received vector $\boldsymbol{r}$ which represents source Y is obtained simply as the noiseless BPSK modulated vector representation of $\boldsymbol{v_Y}$, i.e.,

$$\boldsymbol{r} = [-1.0 \ -1.0 \ +1.0 \ -1.0 \ -1.0 \ +1.0 \ +1.0 \ -1.0 \ -1.0 \ -1.0 \ -1.0 \ +1.0]$$

The noiseless BPSK modulated vector representation of $\boldsymbol{v_Y}$ may also be denoted by $\boldsymbol{v_{Ym}}$ so that $\boldsymbol{r} = \boldsymbol{v_{Ym}}$.

In the BSC, the probability that a received bit corresponds to the transmitted bit is determined using the crossover probability p which in turn is calculated using the $Q(x)$ function in equation (8.10).

$$Q(x) = 0.5 \times erfc(\tfrac{x}{\sqrt{2}}) \tag{8.10}$$

From equation (2.9), $SNR = 1/\sigma^2$, therefore $1/\sigma = \sqrt{(SNR)}$. By substituting $1/\sigma$ for x in equation (8.10), we obtain the bit crossover probability p as,

$$p = 0.5 \times erfc(\frac{1}{\sigma\sqrt{2}}) \tag{8.11}$$

or

$$p = 0.5 \times erfc(\sqrt{\frac{SNR}{2}}) \tag{8.12}$$

Assuming $SNR = 1/\sigma^2 = 1.239$ (equal to $SNR(dB) = 0.931$), then using equation (8.11) the bit crossover probability $p = 0.13$, and the probabilities that the received vectors in $\boldsymbol{r}$ are equal to bit 1 are as follows,

$$P = [0.13 \quad 0.13 \quad 0.87 \quad 0.13 \quad 0.13 \quad 0.87 \quad 0.87 \quad 0.13 \quad 0.13 \quad 0.13 \quad 0.13 \quad 0.87]$$

That is, if $r_i = -1.0$ then the probability that a crossover has occurred p is used, and if $r_i = +1.0$ then the probability that a crossover has not occurred, i.e. $(1 - p)$, is used. These crossover probabilities are used as input from the BSC to the BP/SP algorithm. Subsequently, using the bit crossover probabilities, the BP/SP algorithm is used to decode the noiseless BPSK modulated message received from source Y using the noiseless side information received from source X regarding the correct values of the set of syndrome bits $\{s_i\}$ $\forall$ $0 \leq i \leq m - 1$ for each n bit long message vector $\boldsymbol{v}$ transmitted.

In order to utilize the side information, the SW BSC decoder uses the same modification to the BP/SP algorithm as described for the BP/SPA syndrome decoder. All other aspects of the BP/SP algorithm in the conventional BP/SPA decoder and the BP/SP algorithm in the SW BSC decoder are identical.

Using the degree distribution of the rate-½ $(n, k) = (1024, 512)$ benchmark code discussed in Chapter 6, i.e. $Q_9(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}$, a rate-½ $(n, k) = (1000, 500)$ LDPC code was constructed. The bit error rate (BER) performance of the rate-½ $(1000, 500)$ LDPC code, when used in the proposed SW BSC decoder, was compared to the

BER performance of other implementations of distributed source coding at asymmetric and symmetric rates using the same code lengths. In Figure 8.2, the asymmetric rate performance of the proposed SW BSC decoder is compared to those of the asymmetric rates in [Sartipi and Fekri, 2004] and [Pradhan and Ramchandran, 2000], and the symmetric rate in [Sartipi and Fekri, 2005].

Figure 8.2 shows the BER results averaged over the two sources, i.e. X_1 and X_2, as a function of the joint entropy for asymmetric rates.

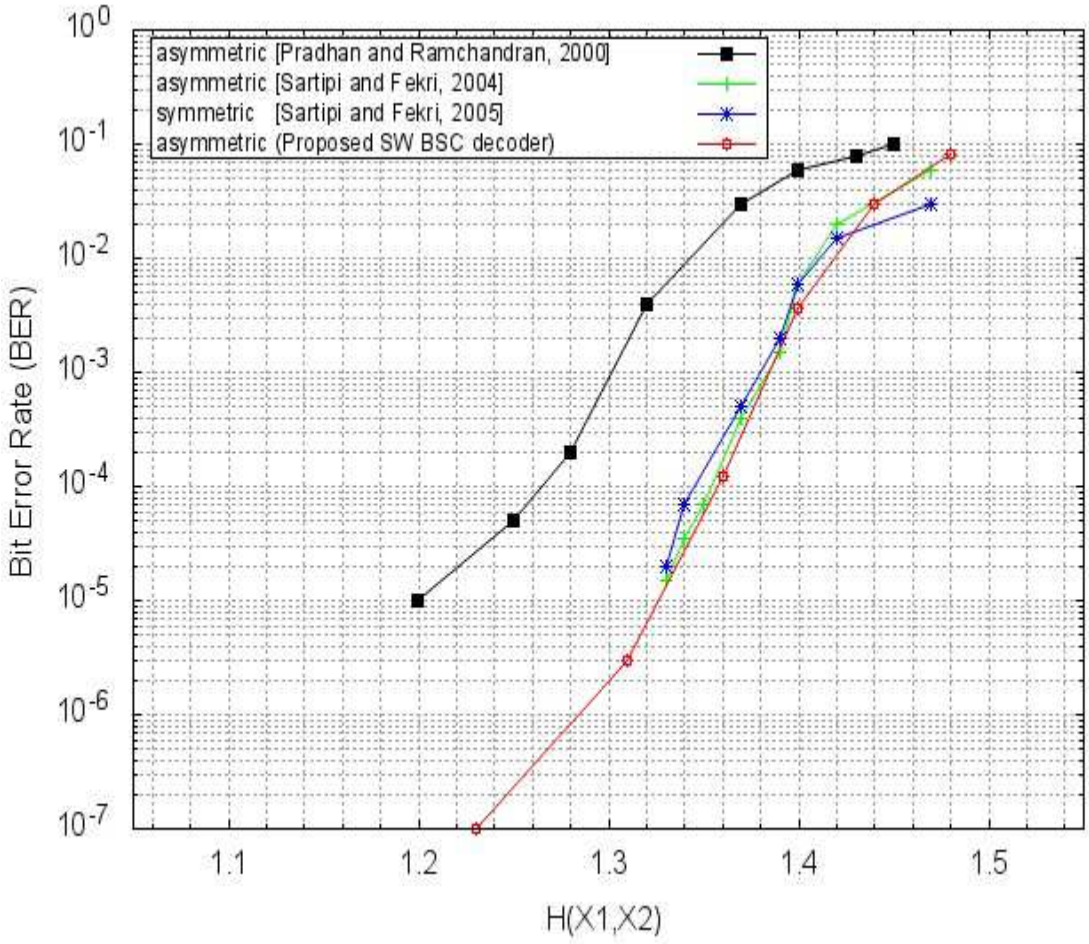

Figure 8.2: Distributed source coding using LDPC codes of length 1000 at symmetric and asymmetric rates

As shown in Figure 8.2, the proposed distributed source coding model, as implemented in the SW BSC decoder, outperforms the coding models presented in the three previously published works.

8.3 Summary

- The standard BP/SPA decoder has been modified to implement distributed source coding of two correlated sources for asymmetric Slepian-Wolf (SW) coding using binary LDPC codes.

- In an investigation into the concept of syndrome decoding, a BP/SPA syndrome decoder which implements a SW-like decoding arrangement was implemented. Compared to the conventional BP/SPA decoder, the BP/SPA syndrome decoder achieved a very good compression with no apparent loss in performance.

- The standard BP/SPA decoder has been modified to implement SW coding of two binary correlated information sources over the BSC (the SW BSC decoder).

- Using a rate-½ $(n, k) = (1000, 500)$ LDPC code with symbol node degree distribution $Q_9(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}$, the SW coding model outperformed other distributed source coding models in literature which utilized LDPC codes of identical length and rate.

Chapter 9

Conclusions

9.1 Contributions to Knowledge

- A detailed and simplified explanation of the iterative decoding procedure in the standard BP/SPA decoder has been given. {Section 2.5.3}
- The *single-edge tree-apex* (SETA) subgraph expansion for finding short cycles in LDPC code Tanner graphs has been proposed. The SETA subgraph expansion is a modification of the subgraph expansion used in the standard PEG algorithm. {Section 4.4.1}
- A technique of evaluating the ACE of cycles in LDPC codes with respect to the symbol nodes of Tanner graphs has been proposed. The technique is based on a '*local ACE*' metric. Consequently, the entire ACE spectrum in a Tanner graph can be described using a *local ACE distribution*. {Chapter 4}
- An efficient algorithm for determining the local girth and ACE distributions in LDPC code Tanner graphs has been presented. The algorithm is relatively easy to implement, has a worst case computational complexity of $O(nm)$, and finds all the short cycles through the symbol nodes in a Tanner graph using SETA subgraph expansions. When applied to irregular short-to-medium length LDPC codes, the algorithm is faster than existing algorithms for counting cycles in LDPC codes. {Chapter 4}

- A cyclic PEG (CPEG) algorithm has been presented. The CPEG algorithm constructs LDPC codes using an edge connections sequence which is different from that of the original PEG algorithm. The edge connections sequence of the CPEG algorithm results in codes with improved girth and ACE distributions compared to codes constructed using the edge connections sequence of the standard PEG algorithm. However, the improved girth and ACE distributions in CPEG based LDPC codes do not translate to lower error-floors than are achievable by standard PEG algorithm based LDPC codes. {Chapter 5}

- DE optimized irregular rate-½ $(n,\ k) = (512,\ 256)$ LDPC codes that have higher d_{min} and s_{min} than have been published for similar LDPC codes have been found. Some of these irregular rate-½ $(512,\ 256)$ LDPC codes have $d_{min} = 20$, and others have $s_{min} = 20$. {Section 5.3}

- From the error-floor performance of ensembles of DE optimized irregular rate-½ $(n,\ k) = (512,\ 256)$ LDPC codes over the AWGN channel, it has been shown that there is a lowering of the error-floor of LDPC codes as their d_{min} increases up to a certain threshold d_{min}. However, no further lowering of the error-floor of LDPC codes are observed as d_{min} is increased above the threshold d_{min} value. Increasing the d_{min} of LDPC codes above the threshold d_{min} does not result in further lowering of their error-floor over the AWGN channel. {Section 5.4}

- A minimum local girth (edge skipping) (MLG (ES)) PEG algorithm has been presented. The MLG (ES) PEG algorithm controls the global girth of codes it constructs by skipping all edge connections which would have resulted in undesired local girths. {Section 6.2}

- The concept of optimal low-correlated-edge density (OED) in LDPC codes has been introduced, OED codes are constructed using DE optimized degree distributions and the MLG (ES) PEG algorithm modification. An irregular rate-½ $(n,\ k) = (512,\ 256)$ LDPC code with the lowest known error-floor has been found, and it is an OED ACE PEG code. That is, it was constructed using a combination of the IPEG construction [Xiao and Banihashemi, 2004], the MLG (ES) PEG algorithm modification, and the proposed technique for obtaining OED codes. {Chapter 6}

- Consequent to an improved symbol node degree distribution, irregular rate-½ $(n, k) = (1024, 512)$ ACE PEG codes that have lower error-floor than published for LDPC codes of identical rate and length have been found. {Section 6.3.2}

- An improved BP/SPA (IBP/SPA) decoder has been presented. The IBP/SPA decoder results in an unprecedented improvement to the performance of short-to-medium length LDPC codes in the waterfall and error-floor regions. The modifications made to the conventional BP/SPA decoding process in the IBP/SPA decoder results in an efficient and effective technique for breaking trapping sets in the decoding of LDPC codes. {Chapter 7}

- The Slepian-Wolf coding model for two binary correlated information sources over the BSC (SW BSC decoder) was implemented and, using a rate-½ $(n, k) = (1000, 500)$ LDPC code with symbol node degree distribution $Q_9(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}$, the SW coding model has been shown to outperform other distributed source coding models in literature which utilized LDPC codes of identical length and rate. {Chapter 8}

9.2 Conclusions and Recommendations for Future Work

This thesis focused mainly on improving on the error-floor performance published for short-to-medium length irregular LDPC codes over AWGN channels in the literature. There are two approaches to this problem: Improving short-to-medium length LDPC code construction methods, and improving the performance of the iterative BP/SPA message-passing decoders used for LDPC codes.

In the first approach, the standard PEG algorithm [Hu et al., 2001] and other existing PEG algorithms were reviewed. New PEG algorithm modifications intended to construct LDPC codes with lower error-floor were investigated. Noteworthy PEG algorithm modifications in the literature which are designed to construct short-to-medium length LDPC codes with lower error-floor were investigated. The standard PEG algorithm, the improved PEG (IPEG) construction [Xiao and Banihashemi, 2004], and the generalized ACE constrained PEG (G-ACE PEG) algorithm [Vukobratović and Šenk, 2008] were implemented and compared. DE optimized short-to-medium length LDPC codes constructed

using the IPEG algorithm (i.e. ACE PEG codes) have significantly improved cycle connectivity in their Tanner graphs and have been shown to significantly improve on the performance of codes of identical dimensions which are constructed using the standard PEG algorithm. G-ACE PEG codes are constructed to have improved ACE properties compared to ACE PEG codes. In Chapter 4, it has been shown that G-ACE PEG codes also have improved girth and ACE properties compared to ACE PEG codes. However, no evidence has been found in the research that G-ACE PEG codes perform better than ACE PEG codes. Investigation results suggests that the widely distributed check node degrees of G-ACE PEG codes are responsible for the fact that the performance of G-ACE PEG codes is only similar to that of ACE PEG codes despite their superior girth and ACE properties.

In Chapter 4, as a necessary aid to the investigations carried out in the thesis, an efficient algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs has been presented. The algorithm is relatively easy to implement and finds all the short cycles through the symbol nodes in Tanner graphs using single-edge tree-apex (SETA) subgraph expansions. The algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs has been shown to be faster than existing algorithms for counting cycles in LDPC codes when applied to irregular short-to-medium length LDPC codes. Additionally, with a worst case computational complexity of $O(nm)$, the algorithm is considerably less complex than similar algorithms in the literature.

A cyclic PEG (CPEG) algorithm which uses an alternative edge connections sequence to that in the standard PEG algorithm has been presented in Chapter 5. The minimum weigth, i.e. d_{min} and s_{min}, distributions in four ensembles of irregular rate-½ $(n, k) = (512, 256)$ LDPC codes which were constructed using the PEG, ACE PEG, CPEG, and ACE CPEG algorithms respectively with DE optimized degree distribution $\lambda_A(x) = 0.34961x^2 + 0.39844x^3 + 0.09961x^4 + 0.04883x^5 + 0.10352x^{14}$ were compared. Each of the four LDPC codes ensembles constructed contained 6000 codes. ACE CPEG codes are obtained using a combination of the IPEG modification and the CPEG algorithm. DE optimized rate-½ $(n, k) = (512, 256)$ irregular LDPC codes which have higher d_{min} and s_{min} than have been published for similar LDPC codes were found in all four code ensembles.

Comparisons of the minimum weight distributions in the PEG, ACE PEG, CPEG, and ACE CPEG code ensembles reveals that the codes in the ACE PEG ensemble had the highest d_{min} and s_{min} distributions. Some DE optimized rate-½ $(n, k) = (512, 256)$ LDPC codes with $d_{min} = 20$, and others with $s_{min} = 20$ were found in the ACE PEG code ensemble. In Section 5.4, it has been shown that increasing the minimum distance of LDPC codes lowers their error-floor performance over AWGN channels. However, there are threshold minimum distances values above which there is no further lowering of the error-floor performance consequent to increasing code minimum distances. In Section 5.5, the global girths, girth distributions, and ACE distributions of the irregular rate-½ $(n, k) = (512, 256)$ LDPC codes in the PEG, ACE PEG, CPEG, and ACE CPEG code ensembles were compared. Results show that the ACE CPEG algorithm constructed the code ensemble with the highest local girth and ACE distributions. However, simulation results show that despite improved girth and ACE distributions, CPEG and ACE CPEG codes do not improve on the error-floor performance achievable by PEG and ACE PEG codes respectively.

A minimum local girth (edge skipping) (MLG (ES)) PEG algorithm which controls the minimum cycle girth connected during the construction of LDPC code Tanner graphs has been presented in Chapter 6. The algorithm controls the global girth of codes it constructs by skipping all edge connections which would have resulted in undesired local girths. A technique for constructing optimal low-correlated-edge density (OED) LDPC codes based on modifying DE optimized symbol node degree distributions and the MLG (ES) PEG algorithm modification has been presented. Results show that irregular rate-½ $(n, k) = (512, 256)$ OED ACE PEG codes have lower error-floor than the lowest error-floor published for LDPC codes of identical rate and length. However, similar to the case for ACE improvements, the proposed method for constructing OED codes only result in lower error-floor LDPC codes when used with some DE optimized degree distributions. Consequent to an improved symbol node degree distribution, rate-½ $(n, k) = (1024, 512)$ ACE PEG codes have been shown to have lower error-floor over the AWGN channel than previously published for LDPC codes of similar length and rate.

In the second approach, modifications to the iterative BP/SPA message-passing decoder with the potential to improve the decoding performance of short-to-medium length LDPC codes over AWGN channels were investigated. In Chapter 7, two simple modifications to the standard BP/SPA decoder have resulted in an unprecedented performance improvement to BP/SPA decoding over AWGN channels. The randomized parity-check equation sequence (RPCES) BP/SPA decoder implements a random probability matrix H_P parity-check equation sequence in each decoding iteration. Compared to the standard BP/SPA decoder, the RPCES BP/SPA decoder results in improved LDPC code performance in both the waterfall and error-floor regions, and the decoding convergence time for the RPCES BP/SPA decoder is significantly less than that for the standard BP/SPA decoder. The use of different series of random decoding equation sequences in the RPCES BP/SPA decoder results in different decoding outcomes. Consequently, the probability of decoding a received vector to a codeword (even if its the wrong one) is increased by trying different series of random decoding equations for its decoding. The RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder augments the RPCES BP/SPA decoder with the ability to reattempt each unsuccessful decoding up to $\boldsymbol{n}$ times. Compared to using the standard BP/SPA decoder and the RPCES BP/SPA decoder, the RPCES + $\boldsymbol{n}$-BP/SPA decoder results in an unprecedented improvement to the performance of short-to-medium length irregular LDPC codes in the waterfall and error-floor regions. The RPCES + $\boldsymbol{n}$-BP/SPA decoder is also referred to as the improved BP/SPA (IBP/SPA) decoder. The modifications made to the conventional BP/SPA decoding process in the IBP/SPA decoder results in an efficient and effective technique for breaking trapping sets in LDPC codes. The IBP/SPA decoder has near maximum likelihood (ML) decoding performance.

In Chapter 8, the standard BP/SPA decoder has been modified to implement distributed source coding of two correlated sources for asymmetric Slepian-Wolf (SW) coding using binary LDPC codes. In an investigation into the concept of syndrome decoding, a BP/SPA syndrome decoder which implements a SW-like decoding arrangement was implemented. Compared to the conventional BP/SPA decoder, the BP/SPA syndrome decoder achieved a very good compression with no apparent loss in

performance. The standard BP/SPA decoder was modified to implement the SW coding of two binary correlated information sources over the BSC (SW BSC decoder). Using a rate-½ $(n, k) = (1000, 500)$ LDPC code with symbol node degree distribution $Q_9(x) = 0.47461x^2 + 0.27930x^3 + 0.03418x^4 + 0.10840x^5 + 0.10352x^{15}$ it has been shown that the SW coding model outperforms other distributed source coding models in the literature which used LDPC codes of identical length and rate.

In addition to the many existing challenges to realizing capacity-approaching short-to-medium length LDPC codes which predate this work, several new problems have emerged in the course of this work. The following is a list of recommendations for future investigations which may contribute to further advancements in short-to-medium length LDPC code construction and decoding performance. A few suggestions to improve the performance of some of the algorithms presented in this thesis are also proffered.

- Arguably, the most important factor which determines the performance of an ensemble of LDPC codes constructed using any type of PEG algorithm is the symbol node degree distribution used to construct the codes. The continued research into finding symbol node degree distributions which are optimized for the construction of short-to-medium length LDPC codes with improved error-floor performance is of a high priority.

- The algorithm for determining the local girth and ACE distributions in LDPC code Tanner graphs can be made significantly faster in fully interconnected Tanner graphs which have symbol nodes of degree 2 by using a single SETA subgraph expansion to determine the girth and ACE of the shortest cycle through them. This is because SETA subgraph expansions start and end at two different vertices of the same symbol node and, for degree 2 symbol nodes, SETA expansions from either edge to the other results in identical shortest cycle girth and ACE values. Hence a single SETA subgraph expansion suffices for symbol nodes of degree 2.

- An investigation into a modified PEG algorithm which adopts the candidate check node selection criteria of the generalized ACE constrained PEG (G-ACE PEG) algorithm by [Vukobratović and Šenk, 2008] but rearranges their prioritization in order to give a higher

priority to the selection of lowest degree candidate check nodes is recommended. Such an algorithm may result in more check node regular short-to-medium length LDPC codes with better ACE and lower error-floor than LDPC codes obtained using the improved PEG (IPEG) construction by [Xiao and Banihashemi, 2004].

- The performance of short-to-medium length OED codes based on modified DE optimized symbol node degree distributions, and constructed using a combination of ACE CPEG algorithm and the MLG (ES) PEG algorithm modification have not been investigated in this thesis. This may be investigated in future work.

- The IBP/SPA decoder presented in this thesis has demonstrated a good capacity for breaking trapping sets and approaches ML decoding performance. It has been observed that the waterfall and error-floor performance of short-to-medium length regular LDPC codes suffer from significantly more trapping sets than their irregular counterparts. Regular LDPC codes generally have significantly higher minimum distances than irregular codes. If it successfully breaks most of the trapping sets in regular LDPC codes, the IBP/SPA decoder may result in smaller gaps in performance between irregular and regular short-to-medium length LDPC codes. Therefore, the performance of short-to-medium length regular LDPC codes using the IBP/SPA decoder presented in this thesis needs to be investigated.

Appendices

The appendices of are stored in and can be accessed from the compact disk (CD) attached to the inner side of the back cover of the hardcopy of this thesis. The following is a list of appendix designations, appendix names, and the folder names under which the document(s) corresponding to each appendix has been saved in the CD.

Appendix A

The standard PEG algorithm — opegz

Appendix B

The standard BP/SPA decoder — klingon

Appendix C

The SETA subgraph expansion based algorithm algorithm for determining the local girth and ACE distributions in LDPC code Tanner Graphs — PCMacegirth1000

Appendix D

The ACE PEG/IPEG algorithm proposed by [Xiao and Banihashemi, 2004] — acepegz

Appendix E

The generalized ACE constrained PEG algorithm Proposed by [Vukobratović and Šenk, 2008] — gacepegfixz

Appendix F

The cyclic PEG (CPEG) algorithm — cpegz

Appendix G

The ACE cyclic PEG (ACE CPEG) algorithm acecpegz

Appendix H

The MLG (ES) PEG algorithm opegfixz

Appendix I

The combined ACE PEG and MLG (ES) PEG algorithm for

constructing OED ACE PEG codes acepegfixz

Appendix J

The LDPC matrix configuration file for the lowest-error-floor

rate-½ $(n,\ k) = (512,\ 256)$ OED ACE PEG code acefixpegz512key29

Appendix K

The LDPC matrix configuration file for the lowest-error-floor

rate-½ $(n,\ k) = (1024,\ 512)$ ACE PEG code acepegz1024key2

Appendix L

The BP/SPA syndrome decoder SYNDAWGNFER

Appendix M

The SW BSC decoder SWBSCBER

Appendix N

The RPCES BP/SPA decoder klingonrp

Appendix O

RPCES + $\boldsymbol{n}$-DFR BP/SPA decoder **or** IBP/SPA decoder klingonrepeat

References

A

[Abu-Surra et al., 2010] Abu-Surra, S., DeClercq, D., Divsalar, D., & Ryan, W. E. (2010, January). Trapping set enumerators for specific LDPC codes. In *Information Theory and Applications Workshop (ITA), 2010* (pp. 1-5). IEEE.

[Ambroze, 2000] Ambroze, M. A. (2000, August). On Turbo Codes and Other Concatenated Schemes in Communication Systems. PhD Thesis. University of Plymouth, United Kingdom.

[Ambroze et al., 2000] Ambroze, A., Wade, G., & Tomlinson, M. (2000). Practical aspects of iterative decoding. *IEE Proceedings-Communications*, *147*(2), 69-74.

[Aftab et al., 2001] Aftab, Cheung, Kim, Thakkar & Yeddanapudi (2001), Information Theory: Information Theory and the Digital Age. [Online]. Website: *web.mit.edu/6.933/www/Fall2001/Shannon2.pdf*

[Ambroze, 2014] Ambroze, M. A. (2014, January). ELEC 508 Lecture Notes. University of Plymouth, United Kingdom.

[Abdu-Aguye., 2016(a)] Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, December). An efficient algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs. In *Signal Processing and Communication Systems (ICSPCS), 2016 10th International Conference on* (pp. 1-7). IEEE.

[Abdu-Aguye., 2016(b)] Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, September). Improved minimum weight, girth, and ACE distributions in ensembles of short block length irregular LDPC codes constructed using PEG and cyclic PEG (CPEG) algorithms. In *Turbo Codes and Iterative Information Processing (ISTC), 2016 9th International Symposium on* (pp. 186-190). IEEE.

[Abdu-Aguye., 2016(c)] Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, September). Lowering the error floor of short-to-medium length LDPC codes using optimal low-correlated-edge density (OED) PEG Tanner graphs. In *Software, Telecommunications and Computer Networks (SoftCOM), 2016 24th International Conference on* (pp. 1-5). IEEE.

B

[Bose and Chaudhuri, 1960] Bose, R. C., & Ray-Chaudhuri, D. K. (1960). On a class of error correcting binary group codes. *Information and control*, *3*(1), 68-79.

[Berlekamp, 1966] Berlekamp, E. R. (1966, December). Nonbinary BCH decoding. University of North Carolina.

[Bahl et al., 1974] Bahl, L., Cocke, J., Jelinek, F., & Raviv, J. (1974). Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Transactions on information theory*, *20*(2), 284-287.

[Blokh and Zyablov, 1974] Blokh, È. L., & Zyablov, V. V. (1974). Coding of generalized concatenated codes. *Problemy Peredachi Informatsii*, *10*(3), 45-50.

[Berlekamp, 1978] Berlekamp, E., McEliece, R., & Van Tilborg, H. (1978). On the inherent intractability of certain coding problems (Corresp.). *IEEE Transactions on Information Theory*, *24*(3), 384-386.

[Berrou et al., 1993] Berrou, C., Glavieux, A., & Thitimajshima, P. (1993, May). Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on* (Vol. 2, pp. 1064-1070). IEEE.

[Bazzi et al., 2004] Bazzi, L., Richardson, T. J., & Urbanke, R. L. (2004). Exact thresholds and optimal codes for the binary-symmetric channel and Gallager's decoding algorithm A. *IEEE Transactions on Information Theory*, *50*(9), 2010-2021.

[Butler and Siegel, 2012] Butler, B. K., & Siegel, P. H. (2012, December). Numerical issues affecting LDPC error floors. In *Global Communications Conference (GLOBECOM), 2012 IEEE* (pp. 3201-3207). IEEE.

[Butler and Siegel, 2014] Butler, B. K., & Siegel, P. H. (2014, December). Error floor approximation for LDPC codes in the AWGN channel. *IEEE Transactions on Information Theory*, *60*(12), 7416-7441.

[Bocharova et al., 2016] Bocharova, I. E., Kudryashov, B. D., & Johannesson, R. (2016). Searching for binary and nonbinary block and convolutional LDPC codes. *IEEE Transactions on Information Theory*, *62*(1), 163-183.

C

[Chase, 1972] Chase, D. (1972). Class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information theory*, *18*(1), 170-182.

[Costello, Jr. et al., 1998] Costello, D. J., Hagenauer, J., Imai, H., & Wicker, S. B. (1998). Applications of error-control coding. *IEEE Transactions on Information Theory*, *44*(6), 2531-2560.

[Calderbank et al., 1999] Calderbank, A. R., Forney, G. D., & Vardy, A. (1999). Minimal tail-biting trellises: The Golay code and more. *IEEE Transactions on Information Theory*, *45*(5), 1435-1455.

[Chang, 1999] Chang, E. S., (1999, July). Recommendation of 10^{-13} bit error rate for 10 Gigabit ethernet. http://grouper.ieee.org/groups/802/3/10G_study/public/july99/chang_2_0799.pdf

[Campello et al., 2001] Campello, J., Modha, D.S. and Rajagopalan, S., 2001, June. Designing LDPC codes using bit-filling. In *Communications, 2001. ICC 2001. IEEE International Conference on* (Vol. 1, pp. 55-59). IEEE.

[Chung et al., 2001] Chung, S. Y., Forney, G. D., Richardson, T. J., & Urbanke, R. (2001). On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications letters*, *5*(2), 58-60.

[Costello and Forney, 2007] Costello, D. J., & Forney, G. D. (2007). Channel coding: The road to channel capacity. *Proceedings of the IEEE*, *95*(6), 1150-1177.

D

[Dorsch, 1974] Dorsch, B. (1974). A decoding algorithm for binary block codes and J-ary output channels (Corresp.). *IEEE Transactions on Information Theory*, *20*(3), 391-394.

[Dolinar et al., 1998] Dolinar, S., Divsalar, D., and Pollara, F. (1998). Code performance as a function of block size. JPL TDA Progress Report, 42-120:29-39.

[Divsalar et al., 1998] Divsalar, D., Jin, H., & McEliece, R. J. (1998, September). Coding theorems for 'turbo-like' codes. In *Proceedings of the annual Allerton Conference on Communication control and Computing* (Vol. 36, pp. 201-210). University of Illinois.

[Davey and MacKay, 1998] Davey, M. C., & MacKay, D. (1998). Low-density parity check codes over GF (q). *IEEE Communications Letters*, *2*(6), 165-167.

[Di et al., 2002] Di, C., Proietti, D., Telatar, I. E., Richardson, T. J., & Urbanke, R. L. (2002, June). Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Transactions on Information Theory*, 48(6), 1570-1579.

[Declercq and Fossorier, 2008] Declercq, D., & Fossorier, M. (2008, July). Improved impulse method to evaluate the low weight profile of sparse binary linear codes. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on* (pp. 1963-1967). IEEE.

[Diao et al., 2013] Diao, Q., Tai, Y. Y., Lin, S., & Abdel-Ghaffar, K. (2013, February). Trapping set structure of LDPC codes on finite geometries. In *Information Theory and Applications Workshop (ITA), 2013* (pp. 1-8). IEEE.

['Distributed source coding', 2017] (2017, October 3). Distributed source coding. From Wikipedia, the free encyclopedia. [Online] https://en.wikipedia.org/wiki/Distributed_source_coding

E

[Elias, 1954] Elias, P. (1954). Error-free coding. *Transactions of the IRE Professional Group on Information Theory*, *4*(4), 29-37.

[Etzion et al., 1999] Etzion, T., Trachtenberg, A., & Vardy, A. (1999). Which codes have cycle-free Tanner graphs?. *IEEE Transactions on Information Theory*, *45*(6), 2173-2181.

[Eleruja et al., 2017] Eleruja, S., Abdu-Aguye, U. F., Ambroze, M., Tomlinson, M., & Zaki, M. (2017, November). Design of binary LDPC codes for Slepian-Wolf coding of correlated information sources. Presented at the *5th IEEE Global Conference on Signal and Information Processing* (GlobalSIP 2017), 14-16 November, 2017, Montreal, Canada. IEEE

F

[Fano, 1963] Fano, R. (1963). A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, *9*(2), 64-74.

[Forney, 1966] Forney, G. D. Concatenated Codes. PhD Thesis. MIT Press, Cambridge, MA 1966.

[Forney, 1973] Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268-278.

[Fossorier and Lin, 1995] Fossorier, M. P., & Lin, S. (1995). Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, *41*(5), 1379-1396.

[Fan, 2000] Fan, J. L. (2000). Array codes as low-density parity-check codes. In *Proc. 2nd Int. Symp. on Turbo Codes Rel. Topics, Brest, France, Sept. 2000*. 543-546.

[Fan, 2001] Fan, J. L. (2001). Message-Passing Algorithm. *Constrained Coding and Soft Iterative Decoding*, 23-96.

[Fan and Xiao, 2006] Fan, J., & Xiao, Y. (2006, August). A method of counting the number of cycles in LDPC codes. In *Signal Processing, 2006 8th International Conference on* (Vol. 3). IEEE.

[Falsafain and Mousavi, 2016] Falsafain, H., & Mousavi, S. R. (2016). Exhaustive Enumeration of Elementary Trapping Sets of an Arbitrary Tanner Graph. *IEEE Communications Letters*, *20*(9), 1713-1716.

G

[Golay, 1949] Golay, M. J. (1949). Notes on digital coding. *Proceedings of the Institute of Radio Engineers*, *37*(6), 657-657.

[Gorenstein and Zierler, 1961] Gorenstein, D., & Zierler, N. (1961). A class of error-correcting codes in p^m symbols. *Journal of the Society for Industrial and Applied Mathematics*, *9*(2), 207-214.

[Gallager, 1962] Gallager, R. G. (1962). Low-density parity-check codes. *IRE Transactions on information theory*, *8*(1), 21-28.

[Gallager, 1963] Gallager, R. G. (1963). Low-density parity-check codes. Cambridge MA: MIT Press.

[Gazelle and Snyders, 1997] Gazelle, D., & Snyders, J. (1997). Reliability-based code-search algorithms for maximum-likelihood decoding of block codes. *IEEE Transactions on Information Theory*, *43*(1), 239-249.

[Guruswami and Sudan, 1998] Guruswami, V., & Sudan, M. (1998, November). Improved decoding of Reed-Solomon and algebraic-geometric codes. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on* (pp. 28-37). IEEE.

H

[Hamming, 1950] Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell Labs Technical Journal*, *29*(2), 147-160.

[Hocquenghem, 1959] Hocquenghem, A. (1959). Codes correcteurs d'erreurs. *Chiffres*, *2*(2), 147-56.

[Hartmann and Rudolph, 1976] Hartmann, C., & Rudolph, L. (1976). An optimum symbol-by-symbol decoding rule for linear codes. *IEEE Transactions on Information Theory*, *22*(5), 514-517.

[Hagenauer and Hoeher, 1989] Hagenauer, J., & Hoeher, P. (1989, November). A Viterbi algorithm with soft-decision outputs and its applications. In *Global Telecommunications Conference and Exhibition 'Communications Technology for the 1990s and Beyond' (GLOBECOM), 1989. IEEE* (pp. 1680-1686). IEEE.

[Han et al., 1993] Han, Y. S., Hartmann, C. R., & Chen, C. C. (1993). Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes. *IEEE Transactions on Information Theory*, *39*(5), 1514-1523.

[Hu et al., 2004] Hu, X. Y., Fossorier, M. P., & Eleftheriou, E. (2004, June). On the computation of the minimum distance of low-density parity-check codes. In *Communications, 2004 IEEE International Conference on* (Vol. 2, pp. 767-771). IEEE.

[Hirotomo et al., 2005] Hirotomo, M., Mohri, M., & Morii, M. (2005, September). A probabilistic computation method for the weight distribution of low-density parity-check codes. In *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on* (pp. 2166-2170). IEEE.

[Hirotomo et al., 2008] Hirotomo, M., Konishi, Y., & Morii, M. (2008, July). On the probabilistic computation algorithm for the minimum-size stopping sets of LDPC codes. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on* (pp. 295-299). IEEE.

[Healy and de Lamare, 2012] Healy, C. T., & de Lamare, R. C. (2012). Decoder-optimised progressive edge growth algorithms for the design of LDPC codes with low error floors. *IEEE Communications Letters*, *16*(6), 889-892.

[Hashemi and Banihashemi, 2015] Hashemi, Y., & Banihashemi, A. H. (2015). On characterization and efficient exhaustive search of elementary trapping sets of variable-regular LDPC codes. *IEEE Communications Letters*, *19*(3), 323-326.

[Hashemi and Banihashemi, 2016] Hashemi, Y., & Banihashemi, A. H. (2016). New characterization and efficient exhaustive search algorithm for leafless elementary trapping sets of variable-regular LDPC codes. *IEEE Transactions on Information Theory*, *62*(12), 6713-6736.

[Halford and Chugg, 2004] Halford, T. R., & Chugg, K. M. (2004, May). Enumerating and counting cycles in bipartite graphs. In *IEEE Communication Theory Workshop* (Vol. 63).

[Halford and Chugg, 2006] Halford, T. R., & Chugg, K. M. (2006). An algorithm for counting short cycles in bipartite graphs. *IEEE Transactions on Information Theory*, *52*(1), 287-292.

[Hu and Eleftheriou, 2006] Hu, X. Y., & Eleftheriou, E. (2006, April). A probabilistic subspace approach to the minimal stopping set problem. In *Turbo Codes&Related Topics; 6th International ITG-Conference on Source and Channel Coding (TURBOCODING), 2006 4th International Symposium on* (pp. 1-6). VDE.

J

[Jin et al., 2000] Jin, H., Khandekar, A., & McEliece, R. (2000, September). Irregular repeat-accumulate codes. In *Proc. 2nd Int. Symp. Turbo codes and related topics*, Brest, France, (pp. 1-8).

[Johnson and Weller, 2001] Johnson, S. J., & Weller, S. R. (2001). Regular low-density parity-check codes from combinatorial designs. In *Information Theory Workshop, 2001. Proceedings. 2001 IEEE* (pp. 90-92). IEEE.

[Johnson and Weller, 2002] Johnson, S. J., & Weller, S. R. (2002). Codes for iterative decoding from partial geometries. In *Information Theory, 2002. Proceedings. 2002 IEEE International Symposium on* (p. 310). IEEE.

[Johnson, 2003] Johnson, S. (2003). *Low-density parity-check codes from combinatorial designs*. Ph.D Thesis, University of Newcastle. Australia

[Johnson, 2006] Johnson, S. J. (2006). Introducing low-density parity-check codes. *University of Newcastle, Australia*.

K

[Kschischang and Frey, 1998] Kschischang, F. R., & Frey, B. J. (1998). Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, *16*(2), 219-230.

[Kou et al., 2001] Kou, Y., Lin, S., & Fossorier, M. P. (2001). Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Transactions on Information Theory*, *47*(7), 2711-2736.

[Kschischang et al., 2001] Kschischang, F. R., Frey, B. J., & Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, *47*(2), 498-519.

[Kotter and Vardy, 1998] Kotter, R., & Vardy, A. (1998, August). Factor graphs: constructions, classification, and bounds. In *Information Theory, 1998. Proceedings. 1998 IEEE International Symposium on* (p. 14). IEEE.

[Kyung and Wang, 2000] Kyung, G. B., & Wang, C. C. (2012). Finding the exhaustive list of small fully absorbing sets and designing the corresponding low error-floor decoder. *IEEE Transactions on Communications*, *60*(6), 1487-1498.

[Koetter and Vardy, 2003] Koetter, R., & Vardy, A. (2003). Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, *49*(11), 2809-2825.

[Krishnan and Shankar, 2006] Krishnan, K. M., & Shankar, P. (2006, March). On the complexity of finding stopping set size in Tanner graphs. In *40th Annual Conference on Information Sciences and Systems* (pp. 157-158).

[Krishnan and Shankar, 2007] Krishnan, K. M., & Shankar, P. (2007). Computing the stopping distance of a Tanner graph is NP-hard. *IEEE transactions on information theory*, *53*(6), 2278-2280.

[Karimi and Banihashemi, 2012(a)] Karimi, M., & Banihashemi, A. H. (2012). Efficient algorithm for finding dominant trapping sets of LDPC codes. *IEEE Transactions on Information Theory*, *58*(11), 6942-6958.

[Karimi and Banihashemi, 2012(b)] Karimi, M., & Banihashemi, A. H. (2012). Counting short cycles of quasi cyclic protograph LDPC codes. *IEEE Communications Letters*, *16*(3), 400-403.

[Khazraie et al., 2012] Khazraie, S., Asvadi, R., & Banihashemi, A. H. (2012). A PEG construction of finite-length LDPC codes with low error floor. *IEEE Communications Letters*, *16*(8), 1288-1291.

[Karimi and Banihashemi, 2013] Karimi, M., & Banihashemi, A. H. (2013). Message-passing algorithms for counting short cycles in a graph. *IEEE Transactions on Communications*, *61*(2), 485-495.

[Kang et al., 2011] Kang, J., Huang, Q., Lin, S., & Abdel-Ghaffar, K. (2011). An iterative decoding algorithm with backtracking to lower the error-floors of LDPC codes. *IEEE Transactions on Communications*, *59*(1), 64-73.

[Kang et al., 2016] Kang, S., Moon, J., Ha, J., & Shin, J. (2016). Breaking the Trapping Sets in LDPC Codes: Check Node Removal and Collaborative Decoding. *IEEE Transactions on Communications*, *64*(1), 15-26.

[Kim et al., 2004] Kim, J. L., Peled, U. N., Perepelitsa, I., Pless, V., & Friedland, S. (2004). Explicit construction of families of LDPC codes with no 4-cycles. *IEEE transactions on information theory*, *50*(10), 2378-2388.

L

[Land and Doig, 1960] Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 497-520.

[Lodge et al., 1992] Lodge, J., Hoeher, P., & Hagenauer, J. (1992). The Decoding of Multidimensional Codes Using Separable Map 'Filters'. In *Proc. of the 16th Biennial Symp. on Comm., Canada,* (pp. 343-346).

[Lodge et al., 1993] Lodge, J., Young, R., Hoeher, P., & Hagenauer, J. (1993, May). Separable MAP 'filters' for the decoding of product and concatenated codes. In *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on* (Vol. 3, pp. 1740-1745). IEEE.

[Lucas et al., 2000] Lucas, R., Fossorier, M. P., Kou, Y., & Lin, S. (2000). Iterative decoding of one-step majority logic deductible codes based on belief propagation. *IEEE Transactions on Communications*, *48*(6), 931-937.

[Luby et al., 1998] Luby, M., Mitzenmacher, M., Shokrollah, A., & Spielman, D. (1998, May). Analysis of low density codes and improved designs using irregular graphs. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (pp. 249-258). ACM.

[Luby et al., 2001] Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., & Spielman, D. A. (2001). Improved low-density parity-check codes using irregular graphs. *IEEE Transactions on information Theory*, *47*(2), 585-598.

[Liveris et al., 2002] Liveris, A. D., Xiong, Z., & Georghiades, C. N. (2002). Compression of binary sources with side information at the decoder using LDPC codes. *IEEE communications letters*, *6*(10), 440-442.

[Lin and Costello, Jr., 2004] Lin, S., & Costello, D. J. (2004). *Error control coding* (Vol. 2). Englewood Cliffs: Prentice Hall.

[Lee et al., 2005] Lee, S. H., Kim, K. S., Kim, Y. H., & Ahn, J. Y. (2005). A cycle search algorithm based on a message-passing for the design of good LDPC codes. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *88*(6), 1599-1604.

[Li et al., 2015] Li, J., Lin, S., & Abdel-Ghaffar, K. (2015, June). Improved message-passing algorithm for counting short cycles in bipartite graphs. In *Information Theory (ISIT), 2015 IEEE International Symposium on* (pp. 416-420). IEEE.

['Low density parity-check code', 2017] (2017, July 12). Low-density parity-check code. From Wikipedia, the free encyclopedia. [Online] https://en.wikipedia.org/wiki/Low-density_parity-check_code

M

[Mitani, 1951] Mitani, N. (1951, November). On the transmission of numbers in a sequential computer. In *National Convention of Inst. Elect Engineers of Japan*.

[Muller, 1954] Muller, D. E. (1954). Application of Boolean algebra to switching circuit design and to error detection. *Transactions of the IRE Professional Group on Electronic Computers*, (3), 6-12.

[Massey, 1969] Massey, J. (1969). Shift-register synthesis and BCH decoding. *IEEE transactions on Information Theory*, *15*(1), 122-127.

[MacWilliams and Sloan, 1977] MacWilliams, F. J., & Sloane, N. J. A. (1977). *The theory of error-correcting codes*. Elsevier. North Holland.

[Margulis, 1982] Margulis, G. A. (1982). Explicit constructions of graphs without short cycles and low density codes. *Combinatorica*, *2*(1), 71-78.

[MacKay and Neal, 1995] MacKay, D. J., & Neal, R. M. (1995, December). Good codes based on very sparse matrices. In *IMA International Conference on Cryptography and Coding* (pp. 100-111). Springer, Berlin, Heidelberg.

[MacKay and Neal, 1996] MacKay, D. J., & Neal, R. M. (1996). Near Shannon limit performance of low density parity check codes. *Electronics letters*, 32(18), 1645.

[MacKay and Neal, 1997] MacKay D. J., & Neal, R. M. (1997, March). Near Shannon limit performance of low density parity check codes. *Electronics letters,* 33(6), 457-458.

[McEliece et al., 1998] McEliece, R. J., MacKay, D. J. C., & Cheng, J. F. (1998). Turbo decoding as an instance of Pearl's" belief propagation" algorithm. *IEEE Journal on selected areas in communications*, *16*(2), 140-152.

[MacKay, 1999] MacKay, D. J. (1999). Good error-correcting codes based on very sparse matrices. *IEEE transactions on Information Theory*, *45*(2), 399-431.

[Mao and Banihashemi, 2001] Mao, Y., & Banihashemi, A. H. (2001, June). A heuristic search for good low-density parity-check codes at short block lengths. In *Communications, 2001. ICC 2001. IEEE International Conference on Communications* (Vol. 1, pp. 41-44). IEEE.

[MacKay, 2002] MacKay, D. J. (2002, April 11). D. MacKay's Gallager code resources [Online]. Website: http://www.inference.phy.cam.ac.uk/mackay/codes/

[MacKay and Postol, 2003] MacKay, D. J., & Postol, M. S. (2003, October). Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes. *Electronic Notes in Theoretical Computer Science*, *74*, 97-104.

[Mouri, 2017] Mouri, S. P., (2017, September). Website: https://www.researchgate.net/post/What_is_the_acceptable_BER_value_for_voice_communication_data_communication_and_Video_communication

N

[Nguyen et al., 2012] Nguyen, D. V., Chilappagari, S. K., Marcellin, M. W., & Vasic, B. (2012). On the construction of structured LDPC codes free of small trapping sets. *IEEE Transactions on Information Theory*, *58*(4), 2280-2302.

P

[Peterson, 1960] Peterson, W. (1960). Encoding and error-correction procedures for the Bose-Chaudhuri codes. *IRE Transactions on Information Theory*, *6*(4), 459-470.

[Peterson and Weldon Jr., 1972] Peterson, W., & Weldon Jr, E. J. (1972). Error-correcting codes. MIT Press, Cambridge, MA 1972.

[Pearl, 1988] Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Franscisco, CA: Morgan Kaufmann, 1988.

[Pradhan and Ramchandran, 2000] Pradhan, S. S., & Ramchandran, K. (2000). Distributed source coding: Symmetric rates and applications to sensor networks. In *Data Compression Conference, 2000. Proceedings. DCC 2000* (pp. 363-372). IEEE.

[Proakis, 2001] Proakis, J. G. (2001) Digital Communications. 4th Edition, McGraw-Hill, New York.

[Papagiannis et al., 2004] Papagiannis, E., Ambroze, M. A., & Tomlinsom, M. (2004). Approaching the ML performance with iterative decoding. In *Communications, 2004 International Zurich Seminar on* (pp. 220-223). IEEE.

R

[Reed, 1953] Reed, I. S. (1953). *A class of multiple-error-correcting codes and the decoding scheme* (No. TR-44). MIT Lexington Lab.

[Reed and Solomon, 1960] Reed, I. S., & Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, *8*(2), 300-304.

[Roseenthal and Vontobel, 2000] Rosenthal, J., & Vontobel, P. O. (2000). Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis. In *in Proc. of the 38-th Allerton Conference on Communication, Control, and Computing*.

[Richardson and Urbanke, 2001] Richardson, T. J., & Urbanke, R. L. (2001). The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on information theory*, *47*(2), 599-618.

[Richardson et al., 2001] Richardson, T. J., Shokrollahi, M. A., & Urbanke, R. L. (2001). Design of capacity-approaching irregular low-density parity-check codes. *IEEE transactions on information theory*, *47*(2), 619-637.

[Richardson, 2003] Richardson, T. (2003, October). Error floors of LDPC codes. In *Proceedings of the annual Allerton conference on communication control and computing*. Monticello. IL, (Vol. 41, No. 3, pp. 1426-1435). The University; 1998.

[Richter et al., 2005] Richter, G., Schmidt, G., Bossert, M., & Costa, E. (2005, May). Optimization of a reduced-complexity decoding algorithm for LDPC codes by density evolution. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on* (Vol. 1, pp. 642-646). IEEE.

[Richter, 2006] Richter, G. (2006, April). Finding small stopping sets in the Tanner graphs of LDPC codes. In *Turbo Codes&Related Topics; 6th International ITG-Conference on Source and Channel Coding (TURBOCODING), 2006 4th International Symposium on* (pp. 1-5). VDE.

[Richter et al., 2005] Richter, G., Schmidt, G., Bossert, M., & Costa, E. (2005, May). Optimization of a reduced-complexity decoding algorithm for LDPC codes by density evolution. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on* (Vol. 1, pp. 642-646). IEEE.

[Richter and Hof, 2006] Richter, G., & Hof, A. (2006, June). On a construction method of irregular LDPC codes without small stopping sets. In *Communications, 2006. ICC'06. IEEE International Conference on* (Vol. 3, pp. 1119-1124). IEEE.

[Rosnes and Ytrehus, 2009] Rosnes, E., & Ytrehus, Ø. (2009). An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices. *IEEE Transactions on Information Theory*, *55*(9), 4167-4178.

[Rosnes et al., 2012] Rosnes, E., Ytrehus, Ø., Ambroze, M. A., & Tomlinson, M. (2012). Addendum to "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices". *IEEE Transactions on Information Theory*, *58*(1), 164-171.

[Rosnes et al., 2014] Rosnes, E., Ambroze, M. A., & Tomlinson, M. (2014). On the minimum/stopping distance of array low-density parity-check codes. *IEEE Transactions on Information Theory*, *60*(9), 5204-5214.

S

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal* 27(3): 379-423.

[Silverman and Balser, 1954] Silverman, R., & Balser, M. (1954). Coding for constant-data-rate systems. *Transactions of the IRE Professional Group on Information Theory*, *4*(4), 50-63.

[Slepian, 1956] Slepian, D. (1956). A class of binary signaling alphabets. *Bell Labs Technical Journal*, *35*(1), 203-234.

[Slepian and Wolf, 1973] Slepian, D., & Wolf, J. (1973). Noiseless coding of correlated information sources. *IEEE Transactions on information Theory*, *19*(4), 471-480.

[Shokrollahi, 1999] Shokrollahi, M. A. (1999, November). New sequences of linear time erasure codes approaching the channel capacity. In *AAECC* (pp. 65-76).

[Sweeney, 1991] Sweeney, P. (1991). Error Control Coding: An Introduction. Prentice Hall, UK (pp. 185-192).

[Sipser and Spielman, 1996] Sipser, M., & Spielman, D. A. (1996). Expander codes. *IEEE Transactions on Information Theory*, *42*(6), 1710-1722.

[Sudan, 1997] Sudan, M. (1997). Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of complexity*, *13*(1), 180-193.

[Schonberg et al., 2004] Schonberg, D., Ramchandran, K., & Pradhan, S. S. (2004, March). Distributed code constructions for the entire Slepian-Wolf rate region for arbitrarily correlated sources. In *Data Compression Conference, 2004. Proceedings. DCC 2004* (pp. 292-301). IEEE.

[Sartipi and Fekri, 2004] Sartipi, M., & Fekri, F. (2004, October). Source and channel coding in wireless sensor networks using LDPC codes. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on* (pp. 309-316). IEEE.

[Sartipi and Fekri, 2005] Sartipi, M., & Fekri, F. (2005, March). Distributed source coding in wireless sensor networks using LDPC coding: the entire Slepian-Wolf rate region. In *Wireless Communications and Networking Conference, 2005 IEEE* (Vol. 4, pp. 1939-1944). IEEE.

[Sharon and Litsyn, 2008] Sharon, E., & Litsyn, S. (2008). Constructing LDPC codes by error minimization progressive edge growth. *IEEE Transactions on Communications*, *56*(3).

[Subbiah, 2008] Subbiah, P. (2008, November). Bit-Error Rate (BER) for high speed serial data communication. http://www.cypress.com/file/88426/download

[Sason, 2009] Sason, I. (2009, July). On universal properties of capacity-approaching LDPC code ensembles. *IEEE Transactions on Information Theory*, *55*(7), 2956-2990.

[Sy et al., 2011] Sy, L. P., Savin, V., Declercq, D., & Pham, N. (2011, April). Scheduled-PEG construction of LDPC codes for Upper-Layer FEC. *Workshop on Coding and Cryptography WCC 2011*, 429-432. *arXiv preprint arXiv:1103.2690*.

['Slepian-Wolf coding', 2017] (2017, November 22). Slepian-Wolf coding. From Wikipedia, the free encyclopedia. [Online] https://en.wikipedia.org/wiki/Slepian-Wolf_coding

T

[Tanner, 1981] Tanner, R. (1981). A recursive approach to low complexity codes. *IEEE Transactions on information theory*, *27*(5), 533-547.

[Teukolsky et al., 1992] S.A. Teukolsky, W.T. Vetterling and B.P. Flanner, "Numerical Recipes in C. The Art of Scientific Computing," W.H. Press, Second Edition, Cambridge University Press, 1992.

[Tian et al., 2004] Tian, T., Jones, C. R., Villasenor, J. D., & Wesel, R. D. (2004, August). Selective avoidance of cycles in irregular LDPC code construction. *IEEE Transactions on Communications*, *52*(8), 1242-1247.

[Tian et al., 2003] Tian, T., Jones, C., Villasenor, J. D., & Wesel, R. D. (2003, May). Construction of irregular LDPC codes with low error floors. In *Communications, 2003. ICC'03. IEEE International Conference on* (Vol. 5, pp. 3125-3129). IEEE.

[Tjhai et al., 2006] Tjhai, C., Tomlinson, M., Horan, R., Ahmed, M., & Ambroze, M. (2006, April). GF (2m) Low-Density Parity-Check Codes Derived from Cyclotomic Cosets. In *Turbo Codes&Related Topics; 6th International ITG-Conference on Source and Channel Coding (TURBOCODING), 2006 4th International Symposium on* (pp. 1-6). VDE.

[Tomlinson et al., 2007] Tomlinson, M., Tjhai, C., & Ambroze, M. (2007). Extending the Dorsch decoder towards achieving maximum-likelihood decoding for linear codes. *IET communications*, *1*(3), 479-488.

[Tjhai, 2007] Tjhai, C. J. (2007, September). A Study of Linear Error Correcting Codes. PhD Thesis, University of Plymouth. United Kingdom.

['Turbo code', 2017] (2017, June 20). Turbo code. From Wikipedia, the free encyclopedia. [Online] https://en.wikipedia.org/wiki/Turbo_ code

U

[Ungerboeck, 1982] Ungerboeck, G. (1982). Channel coding with multilevel/phase signals. *IEEE transactions on Information Theory*, *28*(1), 55-67.

V

[Viterbi, 1967] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, *13*(2), 260-269.

[Vardy, 1997] Vardy, A. (1997). The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, *43*(6), 1757-1766.

[Valembois and Fossorier, 2004] Valembois, A., & Fossorier, M. (2004). Box and match techniques applied to soft-decision decoding. *IEEE Transactions on Information Theory*, *50*(5), 796-810.

[Vasić and Milenkovic, 2004] Vasic, B., & Milenkovic, O. (2004). Combinatorial constructions of low-density parity-check codes for iterative decoding. *IEEE Transactions on information theory*, *50*(6), 1156-1176.

[Vasić et al., 2009] B. Vasić, S. K. Chilappagari, D. V. Nguyen and S. K. Planjery, "Trapping Set Ontology" *In 47th Annual Allerton Conference on Communication, Control and Computing 2009, Allerton, Illinois*, September 2009, pp. 1–7. IEEE

[Venkiah et al., 2008] Venkiah, A., Declercq, D., & Poulliat, C. (2008, September). Randomized progressive edge-growth (RandPEG). In *Turbo Codes and Related Topics, 2008 5th International Symposium on* (pp. 283-287). IEEE.

[Vukobratović and Šenk, 2007] Vukobratovic, D., Djurendic, A., & Senk, V. (2007, June). ACE spectrum of LDPC codes and generalized ACE design. In *Communications, 2007. ICC'07. IEEE International Conference on* (pp. 665-670). IEEE.

[Vukobratović and Šenk, 2008] Vukobratović, D., & Šenk, V. (2008, January). Generalized ACE constrained progressive edge-growth LDPC code design. *IEEE Communications Letters*, *12*(1), 32-34.

[Vukobratović and Šenk, 2009] Vukobratovic, D., & Senk, V. (2009, August). Transactions papers evaluation and design of irregular LDPC codes using ACE spectrum. *IEEE Transactions on Communications*, *57*(8), (pp. 2272-2278). IEEE

W

[Wozencraft, 1957] Wozencraft, J. M. (1957). Sequential decoding for reliable communication. Technical Report 325. Research Laboratory of Electronics MIT

[Weldon, 1966] Weldon, E. J. (1966). Difference-Set Cyclic Codes. *Bell Labs Technical Journal*, *45*(7), 1045-1055.

[Wiberg et al., 1995] Wiberg, N., Loeliger, H. A., & Kotter, R. (1995). Codes and iterative decoding on general graphs. *Transactions on Emerging Telecommunications Technologies*, *6*(5), 513-525.

[Wiberg, 1996] Wiberg, N. (1996). Codes and Decoding on General Graphs. PhD Thesis, Linköping University, Sweden, 1996.

[Weiss, 2000] Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural computation*, *12*(1), 1-41.

[Wang et al., 2006] Wang, C. C., Kulkarni, S. R., & Poor, H. V. (2006). Exhausting error-prone patterns in LDPC codes. *arXiv preprint cs/0609046* or ArXiv: http://arxiv.org/pdf/cs/0609046, 2006.

[Wang et al., 2009] Wang, C. C., Kulkarni, S. R., & Poor, H. V. (2009). Finding all small error-prone substructures in LDPC codes. *IEEE Transactions on Information Theory*, *55*(5), 1976-1999.

X

[Xiao and Banihashemi, 2004] Xiao, H., & Banihashemi, A. H. (2004). Improved progressive-edge-growth (PEG) construction of irregular LDPC codes. *IEEE Communications Letters*, *8*(12), 715-717.

Y

[Yazdani and Banihashemi, 2004] Yazdani, M., & Banihashemi, A. H. (2004, June). On construction of rate-compatible low-density parity-check codes. In *Communications, 2004 IEEE International Conference on* (Vol. 1, pp. 430-434). IEEE.

Z

[Zinov'ev, 1976] Zinov'ev, V. A. (1976). Generalized cascade codes. *Problemy Peredachi Informatsii*, *12*(1), 5-15.

[Zhang and Fossorier, 2005] Zhang, J., & Fossorier, M. P. (2005). Shuffled iterative decoding. *IEEE Transactions on Communications*, *53*(2), 209-213.

[Zhang and Siegel, 2011] Zhang, X., & Siegel, P. H. (2011, December). Efficient algorithms to find all small error-prone substructures in LDPC codes. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE* (pp. 1-6). IEEE.

[Zhang and Schlegel, 2011] Zhang, S., & Schlegel, C. (2011, September). Causes and dynamics of LDPC error floors on AWGN channels. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on* (pp. 1025-1032). IEEE.

Publications

9th International Symposium on Turbo Codes & Iterative Information Processing (ISTC' 2016) Conference Paper

Lowering the Error Floor of Short-to-Medium Length LDPC Codes using Optimal Low-Correlated-Edge Density (OED) PEG Tanner Graphs

---

Publication has been removed due to Copyright restrictions.

Paper: Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, September). Lowering the error floor of short-to-medium length LDPC codes using optimal low-correlated-edge density (OED) PEG Tanner graphs. In *Software, Telecommunications and Computer Networks (SoftCOM), 2016 24th International Conference on* (pp. 1-5). IEEE.

24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2016) Conference Paper

Improved Minimum Weight, Girth and ACE Distributions in Ensembles of Short Block Length Irregular LDPC Codes Constructed using PEG and Cyclic PEG (CPEG) Algorithms

Publication has been removed due to Copyright restrictions.

Paper: Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, September). Improved minimum weight, girth, and ACE distributions in ensembles of short block length irregular LDPC codes constructed using PEG and cyclic PEG (CPEG) algorithms. In *Turbo Codes and Iterative Information Processing (ISTC), 2016 9th International Symposium on* (pp. 186-190). IEEE.

10th International Conference on Signal Processing and Communication Systems (ICSPCS' 2016) Conference Paper

An Efficient Algorithm for Determining the Girth and ACE Distributions in LDPC Code Tanner Graphs

Publication has been removed due to Copyright restrictions.

Paper: Abdu-Aguye, U. F., Ambroze, M. A., & Tomlinson, M. (2016, December). An efficient algorithm for determining the girth and ACE distributions in LDPC code Tanner graphs. In *Signal Processing and Communication Systems (ICSPCS), 2016 10th International Conference on* (pp. 1-7). IEEE.

5th IEEE Global Conference on Signal and Information Processing (GlobalSIP 2017) Conference Paper

Design of Binary LDPC Codes for Slepian-Wolf Coding of Correlated Information Sources

Publication has been removed due to Copyright restrictions.

Paper: Eleruja, S., Abdu-Aguye, U. F., Ambroze, M., Tomlinson, M., & Zaki, M. (2017, November). Design of binary LDPC codes for Slepian-Wolf coding of correlated information sources. Presented at the *5th IEEE Global Conference on Signal and Information Processing* (GlobalSIP 2017), 14-16 November, 2017, Montreal, Canada. IEEE.