A STUDY OF ITERATIVE CAPACITY-APPROACHING CODES AND THEIR OPTIMAL DECODING ALGORITHMS

by

Li Yang

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Computing and Mathematics
Faculty of Technology

August 2010

A Study of Iterative Capacity-Approaching Codes And Their Optimal Decoding Algorithms

by

Li Yang

Abstract

Since the invention of turbo codes in 1993 and the rediscovery of Gallager's LDPC codes in 1999, Shannon's determination of the capacity of memoryless channels has become touchable and achievable by adopting these codes iterative decoding scheme. Theoretically, turbo codes and LDPC codes have essentially revolutionised the coding field and become the most focused research topic in recent years. Although these remarkable codes have demonstrated an asymptotic performance close to the Shannon limit, there exists a fact that the practice still lags behind the theory by some margins, for instance, the code constructional difficulty, decoding complexity and the hardware or other implementation issues. In terms of the near optimum decoding performance, it still seems infeasible in practice.

This research work endeavours to fill some of these gaps concerning the design, analysis, application and algorithm optimisation of these simple but good codes, which aims to provide a near optimum decoding performance with much less computational complexity.

After the study of these codes and their iterative decoding scheme, we introduce two hybrid decoding arrangements for the erasure channel and the AWGN channel. Both decoding arrangements are designed to achieve the near optimum or sub-optimal performance with much less computational complexity compared to the maximum likelihood decoder.

The second main contribution is to introduce an efficient algorithm by exploring the codes tree representation to help analyse the codes weight spectra and stopping sets. Furthermore, an extended decoding method based on the state-of-art tree search is proposed to ensure the optimum decoding performance for sparse structural codes in moderate codeword length.

Contents

List of Figures

List of Tables

List of Algorithms

NOMENCLATURE

AWGN	*addictive white Gaussian noise*
BSC	*binary symmetric channel*
BEC	*binary erasure channel*
ML	*maximum likelihood*
BP	*belief propagation*
APP	*a posteriori probability*
MAP	*maximum a posteriori*
PDF	*probability density function*
LLR	*logarithm of likelihood ratio*
RSC	*recursive systematic convolutional*
BER	*bit error rate*
FER	*frame error rate*
LDPC	*low density parity check*
TGC	*turbo Gallager code*
LUT	*look up table*
CRSC	*circular recursive systematic convolutional*
OSD	*ordered statistics decoder*
MRB	*most reliable bits*
LRB	*least reliable bits*
SNR	*signal noise ratio*
EG	*Euclidean geometry*
PEG	*progressive edge growth*
PG	*projective geometry*
R_c	*code rate*
d_{min}	*minimum distance of codewords*
s_{min}	*minimum distance of stopping sets*
$\mathbf{H}$	*parity check matrix*
$\mathbf{G}$	*generator matrix*
$\mathfrak{F}$	*a constraint set*
$\mathbb{N}$	*the set of natural numbers*
$\mathbb{R}$	*the set of real numbers*

I would like to dedicate this thesis to my loving parents Mr Heping Yang and Mrs Yingxin Li and my lovely fiancee Dr Jing Cai ...

Acknowledgements

First of all, I would like to thank my director of study Professor Martin Tomlinson, second supervisor Dr. Marcel Adrian Ambroze and third supervisor Dr. Mohammed Ahmed Zaki for their help and patience during the research work. They are always supporting and encouraging to make me confident.

I would like to thank my former MSc supervisor Dr. Licha Mued, the research group colleagues Dr. Jing Cai, Dr. Cen Tjhai, Mr Ismail, Dr. Purav Shah, Dr. Xin Xu, Mr Mubarack Jibril, Dr. Marco Gomes, Mr Fan Wu, Dr. Zhong Peng, Miss Peng Zhao, Miss Tianlan Zhou, Mrs Carmen, Miss Gina Tjhai and Mr Fudong Li for their patient help and advice, you guys are the source of pushing me to work hard.

And thanks to each family member for their unlimited support and love. Especially to my parents, they are not only supporting me in finance, but also respecting my decision and teach me the right way of becoming a mature man.

And also thanks the people, who have contributed on my research work.

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 without prior agreement of the Graduate Committee.

The relevant scientific seminars and conferences were regularly attended, three major IEEE conference papers ([131], [2], [132]) have been published. And another paper [133] was submitted for IEEE conferences. The author also participated the organised skill development programme by University of Plymouth

Conference Publications:

- *L. Yang*, M. Ambroze and M. Tomlinson, "Comparison of Decoding Turbo Gallager Codes in Hybrid Decoding Arrangements with Different Iterative Decoders over the Erasure Channel", *the 11th IEEE International Conference on Communications Systems (ICCS)*, Nov, 2008, Guangzhou, China. The related work and iterative decoding algorithms complexity analysis could be referred to Chapter 4 and Chpater 6.
- M.Ambroze, M. Tomlinson and *L. Yang*, "Exhaustive Weight Spectrum Analysis of some well known LDPC Codes", *the 10th International Symposium on Communication Theory and Applications (ISCTA)*, July, 2009, Ambleside, Lake District, UK. The related work and proposed algorithms could be referred to Chapter 8.

- *L. Yang*, M. Tomlinson and M. Ambroze, "Decoding Low-Density Parity-Check Codes with Error-Floor Free over the AWGN Channel", *the 2010 IEEE International Conference on Wireless Communication, Networking and Information Security (WCNIS2010)*, June, 2010, Beijing, China. The related decoding arrangement algorithm could be referred to Chapter 7.
- *L. Yang*, M. Tomlinson and M. Ambroze, "Extended Optimum Decoding for LDPC Codes based on Exhaustive Tree Search Algorithm", submitted to *t*he 12th IEEE International Conference on Communications Systems (ICCS), November, 2010, Singapore. The related tree search based decoding algorithm could be referred to Chapter 9.

Word count of main body of thesis: (approximate) 47158

Signed [illegible]

Date 23/08/2010

Chapter 1

Introduction

1.1 A Brief History of Coding Theory

Since the landmark paper was published by Shannon [108] in 1948, the researchers in the field of information theory have been striving to discover the practical coding schemes that could approach channel capacity, also called *Shannon limit.* In the past decades, from "algebraic coding" to "probabilistic coding", from the famous "coding is dead" workshop to the "turbo revolution", from the rediscovery of "LDPC codes" to the near Shannon limit performances, pioneers in coding theory have already successfully achieved the near Shannon limit performances in practice.

The algebraic codes led in the first couple of decades due to their better correction and detection abilities, which were designed to approach the optimum decoding performance by improving the minimum distance in a reasonable information length k, due to the decoding complexity of searching 2^k codewords. The first single-error-correcting binary linear codes $(31, 26, 3)$ were developed by Hamming [50], which are "perfect" in the sense of providing a Hamming radius 1 over the codes spheres to each of the 2^k codewords. Meanwhile, Golay [46] introduced another "perfect" binary linear codes $(23, 12, 7)$ with triple-error corrections, which have Hamming radius 3 of the codes spheres, and also a double-error-correcting ternary code $(11, 6, 5)$. Soon, another class of error-correcting codes, called *Reed-Muller* (RM) codes, was introduced by Muller [87] and Reed [93] proposed its efficient decoding algorithm. For all those well known algebraic codes, the hard-decision decoding algorithm is not sufficient to reach the Shannon limit target. Thus the soft received value over the noisy channel was demanded to be adapted for the modern coding scheme. The earliest soft-decision decoding algorithm, called *Wagner decoding* was described by Silverman & Balser [109]. Since then, the reliability of received channel output was gradually taken into account as the soft decoding algorithm. In 1960s, it was the dominative period of cyclic codes [91] inspired by RM codes. Cyclic codes are a class of codes that could be invariant under cyclic shifts of n-tuple codewords. With such cyclic characteristic, the invention of *Bose-Chaudhuri-Hocquenghem* (BCH) codes were independently introduced by Hocquenghem [54] and Bose & Ray-Chaudhuri [14]. Meanwhile, a class of non-binary BCH codes, recognised as *Reed Solomon* (RS) codes, was investigated by Reed & Solomon [94]. The more efficient decoding algorithm using finite-field arithmetic was realised by Peterson [90] with complexity on the order of d_{min}^3. In 1969, Massey [83] interpreted the reduced complexity on the order of d_{min}^2 decoding algorithm devised by Berlekamp [7]. The *Berlekamp-Massey* algorithm became the standard for the next decade. Since then, another approach of decoding block codes algorithm by adapting the channel measurement information was introduced by Chase [19].

Upon the inspiration by Shannon's probabilistic approach to coding, another class of codes, which aimed to provide an intermediate decoding performance in a balance between performance and computational complexity, was called *convolutional codes* and invented by Elias [35]. Based on the tree structure of convolutional codes, a sequential search decoding algorithm was proposed by Wozencraft & Reiffen [129]. Then the fast sequential decoding with memory-free was developed by Fano [37]. Subsequently, Massey [82] proposed a very simple decoding method for convolutional codes, called *threshold decoding.* In 1967, the "asymptotically optimal" decoding algorithm for convolutional codes , called *Viterbi algorithm* (VA), was introduced by Viterbi [126]. The adaptation of soft decisions in VA sparkled the light of the new approach to compute the *a posteriori probability* (APP) based on the reliability information. Gallager [45] proposed an iterative message-passing decoding algorithm by adapting the APP values for his invention

LDPC codes. Meanwhile, Massey [82] developed an APP version of threshold decoding algorithm. In 1974, Bahl *et al.* [5] published the APP based decoding algorithm originally for convolutional codes, called BCJR algorithm, which is a forward-backward bidirectional decoding algorithm over the codes trellis rather than the forward-only directional VA algorithm. BCJR algorithm as the first *soft-input and soft-output* (SISO) algorithm became the key to trigger the turbo decoding revolution. Since the concatenated structure introduced by Forney, Jr. [40], the more successful VA decoding by adapting soft decisions in SISO, called the *soft-output Viterbi algorithm* (SOVA), was proposed by Hagenauer & Hoeher [48] in 1989.

With the BCJR algorithm as the basis decoding algorithm by using soft APP values and the iterative decoding algorithm for concatenated codes [74], the revolutionary class of codes, called *turbo codes*, was introduced by Berrou *et al.* [10] in 1993. The parallel concatenated turbo codes structure ensures the state of "random-like" code generation rule by permuting information sequence to generate a new sequence linked by a pre-defined interleaver. The idea of *extrinsic information* is introduced and used to exchange the additional reliable knowledge about an information bit between each independent BCJR decoder during the decoding iteration. Due to the small minimum distance of most turbo codes, although the performance could easily approach probability 10^{-5} at low SNR by using a randomly constructed interleaver, the performance flattens out soon as the increased SNR , such phenomenon is called *error floor.* Several approaches were tried in order to lower the error floor. S.Benedetto *et al.* [105] proposed the serial concatenation structure for turbo codes instead of using parallel concatenation. A multi-dimensional concatenated class of turbo codes was proposed by Boutillon & Gnaedig [15] to eliminate the low-weight codeword set.

After the more than 30-year invention of LDPC codes by Gallager [45], MacKay & Neal [81] "rediscovered" such LDPC codes with sparse parity-check matrix structure. They showed that the near Shannon limit performance could be obtained by long LDPC-type codes with the iterative decoding. LDPC codes could be represented by a *bi-partite* graph, also called *Tanner graph* introduced by Tanner [117]. In graphical representation, it shows that the check nodes and variable nodes are defined by a fixed number of degrees corresponding to each parity-check equation, such codes are called *regular* LDPC codes. Luby *et al.* [77] proposed a new class of LDPC codes with *irregular* graphs to optimise the degree sequences, which determine the edge connections over the parity-check matrix and the sparsity of the code. The asymptotic results, by designing the irregular LDPC codes with arbitrary degree sequences and following *density evolution* criteria proposed by Richardson *et al.* [95], were presented in [23]. In such design structure, irregular LDPC codes could clearly outperform turbo codes in block length on the order of 10^5 or more. For the erasure channel [36], the first iterative decoding for LDPC codes was introduced by Luby *et al.* [75] in 1997. Since then, turbo codes and LDPC codes have been extending their competition over the erasure channel. Luby *et al.* [76] showed that the possibility of designing a LDPC code to approach the channel capacity arbitrarily closely. The finite-length analysis of LDPC codes over the erasure channel was described by Di *et al.* [32]. Due to turbo codes near-capacity performance and low encoding complexity, they also became a candidate for the erasure channel. It is shown by Rosnes & O.Ythehus [100] that stopping sets also exist for turbo codes and that they characterise exactly the performance of turbo decoding on the binary erasure channel. Turbo codes are easier to design due to the existence of efficient weight spectrum algorithms by Rosnes & O.Ythehus [99], however LDPC codes could potentially have lower error floors [45, 89]. A new approach was proposed by Colavolpe [24], which is a new code scheme that combines the advantages of turbo codes with those of LDPC codes, now known as *turbo Gallager codes*, which can be decoded either by BCJR algorithm or *belief propagation* (BP) algorithm. Besides the random construction of LDPC codes in [45, 77, 81], another algebraic class of LDPC codes based on finite geometries was introduced by Kou *et al.* [62], which shows that such cyclic or quasi cyclic LDPC codes could be successfully decoded by BP decoding algorithm with simple encoding complexity, especially most of those codes with high code-rate in variable codeword length. The well structured LDPC codes featured by the simple encoding complexity in cyclic format could perform equally well as their equivalent random LDPC codes in terms of bit-error performance, frame-error performance and error-floor collectively [72, 73]. In 60 years, Shannon's idea "A Mathematical Theory of Communications" has successfully been developed into practice, which will fully accelerate the developing rhythm of modern digital communication system.

1.1.1 Turbo codes and LDPC codes

Both classes of codes are characterised by the term of iterative decoding technique. Turbo codes as recursive systematic component codes in parallel, could iteratively pass the decoded codeword to the other decoder in terms of evaluating the soft output for each bit accumulated by the extrinsic information. But due to the size of the interleaver, there exists the *latency* issue, which limits the capability of approaching the Shannon limit, since the shannon limit is approachable if the latency is unlimited. For limited latency, Shannon subsequently showed that capacity is further reduced, and at a latency of 320, this loss is equivalent to nearly 2 dB. In practice, turbo codes scheme also suffers by a further issue: the so-called *error floor*. This research work deals with the new efficient decoding algorithm in terms of achieving optimum or sub-optimal decoding performance with much less computational complexity for the practical channels, like the erasure channel. Two main problems motivate this research. The first problem concerns the fact that the optimum decoding performance is still infeasible in practice due to some margins, like decoding complexity and the stopping sets from the iterative decoding scheme. Since most of maximum likelihood decoding algorithm are generally referred to be n^3 decoding complexity, the excellent iterative decoding performance with well selected codes inspires a new hybrid decoding arrangement by taking the iterative decoder as the initial decoder and the more complex maximum likelihood decoder becomes optional. Such idea is applied in the erasure channel and the AWGN channel resulting with optimum performance achievement and lowered error floor. The second problem is an attempt to analyse the codes weight spectra and stopping sets efficiently for LDPC codes. Based on the introduced efficient algorithm, an extended decoding algorithm for LDPC codes is proposed aiming to achieve an optimum decoding performance.

1.2 A General Model of Digital Communication System

A digital communication system aims to provide a cost-effective system for transmitting information from a sender to an end-user at the rate and level of accuracy that the user requires [85]. The technique of *error correcting codes* (ECC), plays an important role in the achievement of accurate transmission of digital data. It is designed to provide the functions of error detection and error correction by appending the addition of redundant codes to the transmitted data. A digital communication system could be simply described as a block diagram as shown in Figure 1.1, it is generally comprised of seven elements as follows:

- Information Source: it is the original data for transmission.
- Encoder: it is the channel encoder, which converts the source information data into format of information symbols with the corresponding parity symbols as a complete codeword. Thus the error detection and correction abilities could perform certain partial recovery for data receiving correctness.
- Modulation: it is the process of transforming a signal as a waveform for transmission over a medium channel. To be noted that, in this thesis, the *binary phase-shift keying* (BPSK) modulation is applied during the AWGN channel transmission, which is to map binary data from $\{1, 0\}$ to $\{1, -1\}$.
- Channel: it is the transmission medium, which could produce the noise or erasure to affect the original transmitted codeword.
- Demodulation: it is the process of transforming the signals into symbols, which is used for decoding and correction.
- Decoder: it is the place of utilising the redundant symbols to detect and correct the channel errors at the receiver end.
- Information User: it is the destination end, where user receives the corrected information data.

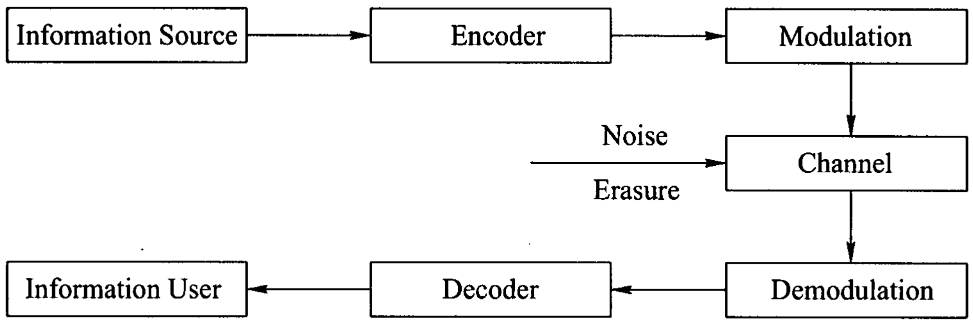


Figure 1.1: Block Diagram of a Digital Communication System

1.2.1 Encoding

All the error correcting codes are based on the same basic principle: redundancy is added to information in order to correct any error that may occur in the process of storage or transmission [86]. In general, the redundant symbols are appended to information symbols to obtain a coded sequence, which is also called a *codeword.* A codeword with n symbols is basically comprised of two parts:

- The information symbols occupy the k positions of a codeword.
- The remaining $n - k$ symbols in a codeword are functional to provide the redundancy to enable the capabilities of error detection and correction.

The set of all coded sequences is called an *error correcting code.*

1.2.1.1 Code

According to the manner in which redundancy is added to information data, error correcting codes are generally categorised into two categories:

- Block codes: in block codes, the information data is on a block-by-block basis. The information sequence is divided into message blocks of k information bits each. A message block is represented by the binary k-tuple $\mathbf{u} = (u_0, u_1, ..., u_{k-1})$, called a message or information bits. Each message could be represented by $(a_0 \times u_0, a_1 \times u_1, ..., a_{k-1} \times u_{k-1})$, where each value of a_i, is either 0 or 1. Thus there are 2^k different possible messages. The encoder transforms each message $\mathbf{u}$ independently into an n-tuple $\mathbf{v} = (v_0, v_1, ..., v_{n-1})$ of discrete symbols. Therefore, corresponding to the 2^k different possible messages, there are 2^k different possible codewords at the encoder output. The set of 2^k codewords of length n is called an (n, k) block code. The ratio $R_c = k/n$ is called the code rate, which means the number of information bits entering the encoder per transmitted symbol.

In a binary code, each codeword $\mathbf{v}$ is also in binary. For achieving an useful binary code, it is necessary to have a different codeword assigned to each message, $k \leq n$, or $R_c \leq 1$. When $k < n$, $n - k$ redundant bits are added in each information message to form a codeword. The redundant bits enable the code having the ability to correct the errors caused by the channel noise. For a fixed code rate R_c, which means the ratio k/n should stay constant, it is implemented by increasing the number of message bits k and the block length n of the code to produce more redundant bits.

- Convolutional codes: in convolutional codes, the codeword is also comprised of k-symbol blocks of the information sequence $\mathbf{u}$ and an encoded sequence $\mathbf{v}$ of n-symbol blocks. Each encoded block does not only depend on the corresponding k-symbol message blocks but also on M previous message blocks. Hence, the encoder should have a *memory order* of M. Then the set of all possible encoded output sequence produced by the encoder forms the code. In a binary convolutional code, redundant bits are added to the information sequence when $k < n$, or $R_c < 1$.

Since block code is based on block-by-block scheme to treat each block of information bits independently from others, block coding is considered as a memoryless operation. In contrast, the output of a convolutional encoder does not only depend on the current input information, but also on the previous input or output, either on a block-by-block or a bit-by-bit basis. According to the research work on the trellis structure of block codes [64] and the tail-biting structure of convolutional codes [78], work on convolutional codes is sometimes referred to the block codes as "codes with time-varying trellis structure", and work on block codes is to consider convolutional codes as "codes with a regular trellis structure" [86].

1.2.1.2 Hamming Weight, Hamming Distance and Minimum Distance

1.1 Definition. The *Hamming weight* $\omega(v)$ of an n-tuple v is defined as the number of non-zero elements of v [70].

For instance, if vector $v = (10010110001)$, the Hamming weight $\omega(v) = 5$.

1.2 Definition. Let u and v be two n-tuples. The *Hamming distance* $d(u, v)$ between u and v is defined as the number of elements in which they differ [70].

For example, if $u = (10010110001)$ and $v = (11001010101)$, then the Hamming distance $d(u, v)$ between vectors u and v is 5.

1.3 Definition. Let u be a codeword of a code $\mathcal{C}$ with Hamming weight $\omega(u)$, and $\omega(u)$ is the smallest Hamming distance of the entire 2^k codewords of $\mathcal{C}$. The weight of $\omega(u)$ is defined as the *minimum distance* of $\mathcal{C}$, denoted as d_{min}.

1.2.2 Channel Model

Let $\mathbf{X}$ be the q-ary input symbols from the encoder, where $X = \{x_0, x_1, ..., x_{q-1}\}$, $\mathbf{Y}$ be the Q-ary output symbols after the channel, where $Y = \{y_0, y_1, ..., y_{Q-1}\}$. In a typical channel model called the *discrete memoryless channel* (DMC) based on the input symbols $\mathbf{X}$ and output symbols $\mathbf{Y}$, a set of conditional probabilities called *transition probabilities* are defined as:

$$P(Y = y_i | X = x_j) \equiv P(y_i | x_j) \tag{1.1}$$

where $i \in \{0, 1, ..., Q-1\}$ and $j \in \{0, 1, ..., q-1\}$.

An information measurement for general source is provided by Shannon [108], it is called *entropy* function of H, which is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i) \tag{1.2}$$

where $H(X)$ is the average self-information per source symbol, n represents the number of output symbols. The entropy function H provides a measure of average amount of information "produced" per symbol by the source [85].

1.2.2.1 Binary Erasure Channel

1.4 Definition. A discrete memoryless channel with binary-input and ternary-output is called the *binary erasure channel* (BEC).

The binary erasure channel as another special case of DMC was first introduced by Elias [36] in 1956. In recent years, the BEC has been shown to be useful in evaluating an error correcting codes performance. The binary erasure channel is depicted by the transition diagram in Figure 1.2, where ϵ represents the erasure probability at which the received value of Y is erased, "?" denotes ambiguity or erasure. In the BEC, each bit is erased at erasure probability ϵ, or received cor-

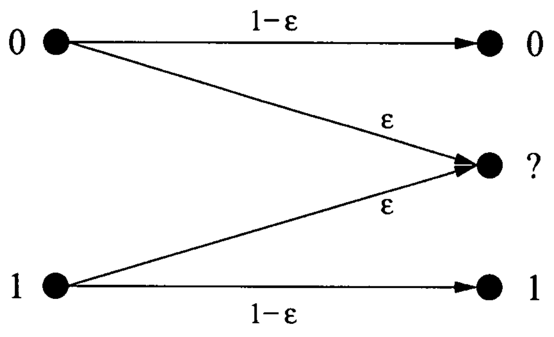


Figure 1.2: Transition Diagram for Binary Erasure Channel Model

rectly at probability $1 - \epsilon$. Currently, it is widely used to model the information transmission over the Internet.

According to (1.1), the set of conditional probabilities for the erasure channel are defined as:

$$\begin{aligned} P(Y=0|X=0) = P(Y=1|X=1) &= 1-\epsilon \\ P(Y=?|X=0) = P(Y=?|X=1) &= \epsilon \end{aligned} \tag{1.3}$$

By extending (1.3) to compute the capacity of the erasure channel, Table 1.1 shows the probability transitions after passing the BEC. Let $H(A)$ be the input

$X=0$	0	$1-\epsilon$	$P(Y=0\|X=0)=\frac{1-\epsilon}{2}$
$P(X=0)=\frac{1}{2}$	?	ϵ	$P(Y=?\|X=0)=\frac{\epsilon}{2}$
$X=1$	?	ϵ	$P(Y=?\|X=1)=\frac{\epsilon}{2}$
$P(X=1)=\frac{1}{2}$	1	$1-\epsilon$	$P(Y=1\|X=1)=\frac{1-\epsilon}{2}$

Table 1.1: Conditional Probabilities for the Binary Erasure Channel

entropy, $H(A,B)$ be the entropy during the channel and $H(B)$ be the output entropy. The entropy of input symbols $H(A)$ is computed as follow:

$$\begin{aligned} H(A) &= P(X=0)\log_2\left(\frac{1}{P(X=0)}\right)+P(X=1)\log_2\left(\frac{1}{P(X=1)}\right) \\ &= \frac{1}{2}\times\log_2(2)+\frac{1}{2}\times\log_2(2)=1 \end{aligned}$$

The entropy of $H(A,B)$ is obtained as:

$$\begin{aligned} H(A,B) &= P(Y=0|X=0)\log_2\left(\frac{1}{P(Y=0|X=0)}\right)+ \\ &\quad P(Y=?|X=0)\log_2\left(\frac{1}{P(Y=?|X=0)}\right)+ \\ &\quad P(Y=?|X=1)\log_2\left(\frac{1}{P(Y=?|X=1)}\right)+ \\ &\quad P(Y=1|X=1)\log_2\left(\frac{1}{P(Y=1|X=1)}\right) \\ &= (1-\epsilon)\times\log_2\left(\frac{2}{1-\epsilon}\right)+\epsilon\times\log_2\left(\frac{2}{\epsilon}\right) \end{aligned}$$

Since $P(Y=0)=P(Y=1)=\frac{1-\epsilon}{2}$ and $P(Y=?)=\frac{\epsilon}{2}+\frac{\epsilon}{2}=\epsilon$, hence the entropy of output symbols $H(B)$ is obtained as:

$$\begin{aligned} H(B) &= P(Y=0)\log_2\left(\frac{1}{P(Y=0)}\right) + P(Y=1)\log_2\left(\frac{1}{P(Y=1)}\right) + \\ &\quad P(Y=?)\log_2\left(\frac{1}{P(Y=?)}\right) \\ &= (1-\epsilon)\times\log_2\left(\frac{2}{1-\epsilon}\right) + \epsilon\times\log_2\left(\frac{1}{\epsilon}\right) \end{aligned}$$

Hence, the capacity C_{BEC} is obtained as:

$$\begin{aligned} C_{BEC} &= H(A) + H(B) - H(A,B) \\ &= 1 + \epsilon\times\log_2\left(\frac{1}{\epsilon}\right) - \epsilon\times\log_2\left(\frac{2}{\epsilon}\right) = 1-\epsilon \end{aligned} \tag{1.4}$$

According to (1.4), Figure 1.3 shows the relationship between capacity of the BEC and erasure probability ϵ. From the curve, we can see the capacity is

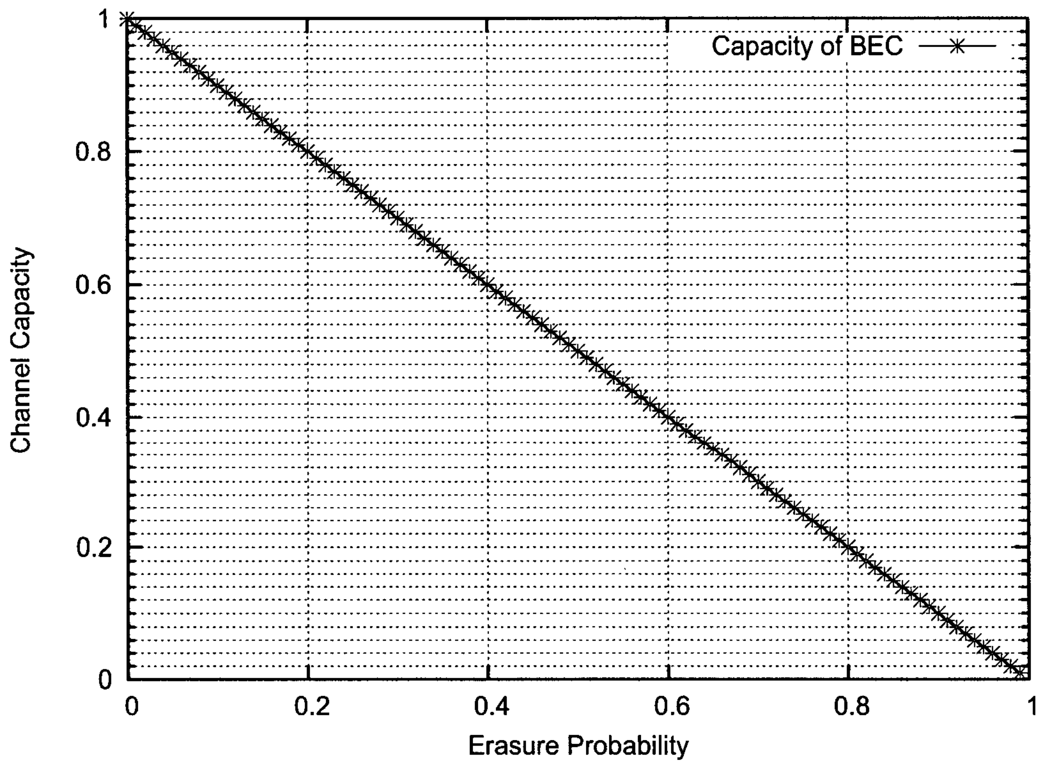


Figure 1.3: Relationship between Capacity and Probability Erasure in the BEC

increased while the erasure probability ϵ decreases. For $\epsilon = 0$, which means the erasure-free transmission, C_{BEC} equals to one bit per transmitted symbol. This erasure property is exactly same as the block or packet lost during the network transmission, that is the reason why it is suitable to model the data transmission over the internet transmission medium.

1.2.2.2 The Additive White Gaussian Noise Channel

1.5 Definition. A typical memoryless channel, which produces wide-band noise, whose amplitude is normally Gaussian distributed random variable, to the transmitted encoded data, is called the *additive white Gaussian noise* (AWGN) channel.

Generally, when the encoded data is transmitted through the channel, the input data is affected by the feature of the channel. Such so-called AWGN channel has great theoretical and practical importance over the digital system design model. In the AWGN channel, the output is produced by simply adding the input with white Gaussian noise. In terms of input x and output y, the AWGN channel is simply described as:

$$y = x + n_G \tag{1.5}$$

where n_G is a zero-mean Gaussian random variable with *variance* σ^2 and the input x can have any one of z discrete values, where $z \leq 2$. Thus the conditional probability density function of the output y given by an input x_i, is obtained as:

$$p(y|x = x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y-x_i)^2/2\sigma^2} \tag{1.6}$$

1.6 Definition. A *signal-to-noise ratio* (SNR) is used to represent the ratio relationship between power and noise.

Then SNR is defined as:

$$\text{SNR } \textit{in power} = 10^{\frac{E_b/N_o}{10}} \tag{1.7}$$

where E_b/N_o is in unit "dB". Then variance σ^2 is obtained as:

$$\sigma^2 = \frac{1.0}{2 \times \text{SNR} \times R_c} \tag{1.8}$$

where R_c is the code-rate. The AWGN channel is an accurate model for many communication links, for instance: satellite and deep-space communications, where the communication performance is only limited by the additive thermal or galactic noise. Furthermore, the AWGN channel is also commonly used to simulate background noise of the channel under study, in addition to multipath, terrain blocking, and interference *etc.*

1.2.3 Decoding

In general, all the binary decoding algorithms are developed based on the hard-decision outputs of the matched filter in the receiver demodulator,which means the output for each signaling interval is quantised in two levels. The two levels are denoted as "0" and "1", which results in a hard-decision binary received sequence. Hard-decision decoding is to process the hard-decision received sequence based on a specific decoding method. It uses the algebraic structures and Hamming distance as its metric. Hard-decision decoding aims to decode the hard-decision received sequence to the closest codeword in the Hamming distance.

Since hard-decision decoding results in a loss of information, which could significantly degrade the performance, and if the output of the matched filter are unquantised or quantised into more than two levels, the soft-decision algorithms is used to complement the weakness. Soft-decision is capable of providing a better error performance than hard-decision decoding by using the additional information contained in the unquantised received samples to recover the transmitted codeword. The soft-decision decoding normally uses the likelihood function, Euclidean distance correlation, and correlation discrepancy as the metric. Generally, soft-decision maximum likelihood decoding of a code has about 3dB of coding gain over the algebraic decoding of the code [71]. As the extrinsic information term introduced, the soft output with accumulated additional reliable knowledge could help determine a more accurate binary sequence in higher confidence.

1.3 Outlines of the Research Work

1.3.1 Objective and Scope of the Research

In this research work, the entire investigations based on the study of turbo codes and LDPC codes could be viewed as a development diagram as shown in Figure 1.4. As the diagram shown, the research work initiates from the study of

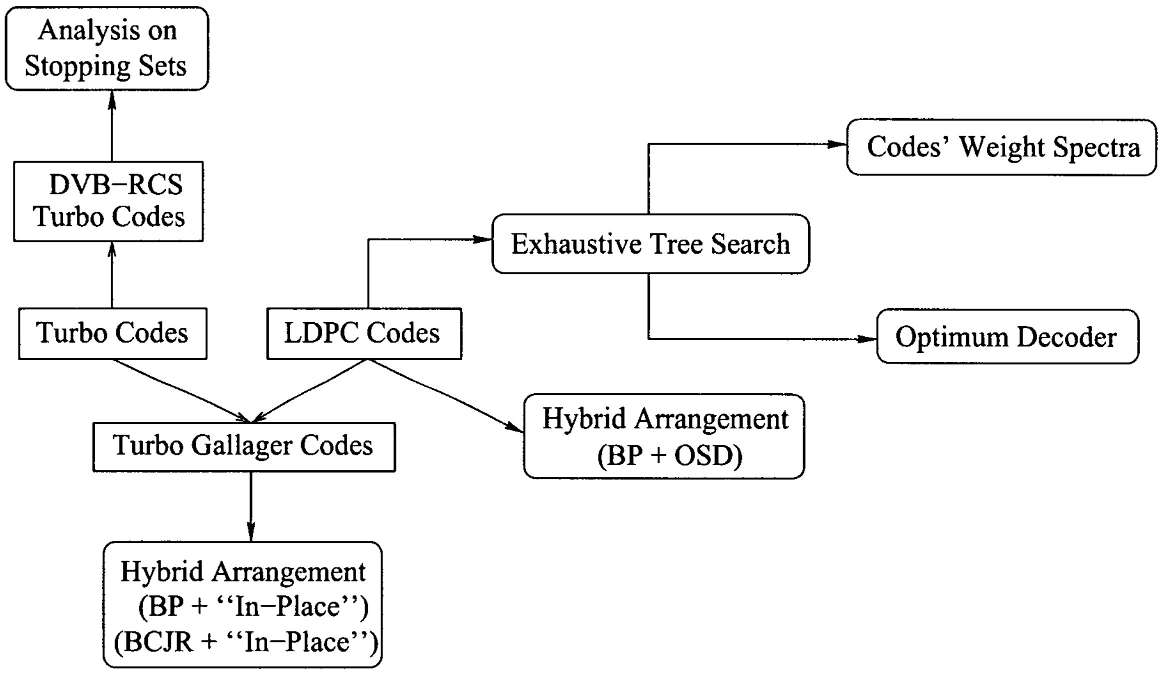


Figure 1.4: An overview of the research work

turbo codes and LDPC codes, and their iterative decoding schemes for the erasure channel and the AWGN channel. In the design of optimisation and simplification of iterative decoding algorithms, we investigate the corresponding BCJR algorithm for turbo codes and BP algorithm for LDPC codes for the erasure channel. For LDPC codes over the erasure channel, we consider the optimised BP decoding algorithm introduced by Luby *et al.* [76] as the specified iterative decoder. For turbo codes, the BCJR algorithm in terms of APP decoding is analysed and optimised for the erasure channel. According to the analysis work on the component convolutional codes in exact probability of erasure and the invention of the table look-up based approach by Kurkoski *et al.* [66], which is also detailed in Chapter 5 of Kurkoski [65]'s dissertation, we propose a further simplified *look-up table* (LUT) based decoding algorithm, which is the optimised replacement of the standard BCJR algorithm. Following the same method of complexity analysis introduced by Wu [130], each iterative decoding algorithm's computational complexity is evaluated in terms of the equivalent additions, which is under the assumption of logical and mathematical operations having similar complexity. As it is well known that the *maximum likelihood* (ML) decoder for the erasure channel is practically realisable but with computational complexity proportional to n^3, where n is the length of the codeword. We propose a decoding scheme to achieve same level of performance as ML decoder does with much less decoding complexity. The hybrid decoding scheme is composed of an iterative decoder as the initial decoder, which is designated to decode most of erasures in each frame. Then the frame still containing erasures is passed to get corrected by a ML decoder. In our hybrid arrangement, a complexity reduced ML decoding algorithm introduced by Cai *et al.* [16] by avoiding the need of column-permutations over the parity-check matrix is considered as our optional optimum decoder. We focus on the turbo Gallager codes, which could be reasonably decoded by either BP decoder or LUT decoder. Then the proposed decoding scheme for turbo Gallager codes over the erasure channel is evaluated with supporting analysis in terms of number of iterations, convergent performance and computational complexity.

Due to the fact that the existing turbo codes stopping sets are the key impact on the decoding performance [100] and the extensive analysis on turbo codes weight spectra [99], our interest is focused on the DVB-RCS turbo codes introduced by Berrou *et al.* [11] over the erasure channel. Although DVB-RCS turbo codes characterise beneficially [18] in the *digital video broadcasting* (DVB) standard [59], there still exists an issue in the design of symbol-based interleaver for DVB-RCS turbo codes, which becomes more obvious on the erasure channel. The BCJR algorithm with two binary-input as a pair of symbol is investigated. Then the bit-based interleavers in design of DRP and S-random [26, 33, 134] for DVB-RCS turbo codes are evaluated, a phenomenon has been observed that there exists a set of stopping sets, which affects the bit-based codes performance worse than the symbol-based codes performance at high erasure probability. According to the analysis on the BCJR algorithm with two binary-input, a probabilistic based algorithm is introduced to help break such set of stopping sets and the decoding performance is improved close to the symbol-based codes performance with lower error floor.

According to the asymptotic performance introduced by Chung *et al.* [23], LDPC codes, especially in irregular structure, could potentially have lower error floors [45]. But LDPC codes as a class of code suitable for iterative decoding, also have the similar issue as turbo codes have in the practical channels [32]. How to lower the error floor caused by the stopping sets of LDPC codes in such simple decoding algorithm becomes one of the most desirable challenge. According to the knowledge and experience in the previous hybrid decoding arrangement, we introduce a new decoding schedule by adopting *ordered statistics decoder* (OSD) algorithm, which is different and much simpler than Fossorier [42]'s reliability-based iterative decoding algorithm for LDPC codes. The proposed decoding arrangement is composed of a BP iterative decoder followed by an OSD-i. Since the objective of the arrangement is to lower the error-floor, the soft output from BP decoder seems not sufficiently reliable to help break the stopping sets, which could be due to the same soft output. Even though, the iterative BP decoding algorithm provides the optimal output, if the Tanner graph of the code is a tree [112, 113]. By directly adopting the soft output as input for a sub-optimal decoding algorithm still seems destructive due to the highly scaled magnitude of soft output, especially when the error frame contains a stopping set. By analysing the iterative output distribution in magnitude, the conditioned iterative output from iterative BP decoder is introduced instead of the standard soft output to become the input to an OSD-i decoder. Such iterative decoder coupled with OSD-i decoder successfully makes the significant improvement than OSD-i decoder alone. Furthermore it guarantees the decoding performance with lower error-floor, even free for a range of LDPC codes.

With the design of code for more advanced property, like the higher minimum distance, the problem of computing the minimum distance of an arbitrary binary linear code was conjectured to be NP-hard [8, 124], which is hard to solve in non-deterministic polynomial time. In the iterative coding scheme, for the raised stopping sets issue, the problem of finding the minimum distance of stopping sets was also claimed to be NP-hard by Krishnan & Shankar [63]. The verdict of NP-hardness does not stop researchers' step to decrypt such a mystic puzzle. On the contrary, different approaches were successfully achieved for computing the minimum distance of a linear block code, in probabilistic approach [17, 52, 68, 115], and in error impulse approach [12, 31, 57]. In the meantime, the problem of finding the minimum size of stopping sets was also established based on the mentioned approaches of computing the minimum distance [53, 96, 127]. Based on the success of the thorough analysis on the turbo codes stopping sets weight distribution [100, 101], a markable paper was published by Rosnes & Ytrehus [102] to find all the small size of stopping sets for LDPC codes. According to Rosnes' previous work, we investigate the novel algorithm and devise a new bound algorithm, which helps restrict the search size and reduce the computational complexity. A range of well known LDPC codes are evaluated, the previous research results are confirmed and extended. Furthermore, the entire class of WiMax LDPC codes [1] is extensively evaluated in standardised construction and "modulo" construction as a complement for results in [103].

According to the APP version of threshold decoding algorithm introduced by Massey [82], the optimum symbol-by-symbol based decoding rule was proposed by Hartmann & Rudolph [51]. It is shown that the decoded codeword is maximised in terms of the cross-correlation, where every dual code is evaluated. But in practice, such decoding rule could be used only if there exists a small set of dual codes, which are very likely to be the high-rate codes. It is different from the BCJR algorithm, which is a trellis-based intermediate algorithm. It performs on the parity-check matrix to search for the maximised dual code, and it ensures the optimum decoding performance, which BCJR decoding algorithm could not achieve due to the existing stopping sets for turbo codes. Moreover, a relationship between the erasure corrections of a binary linear code and its low-weight codewords was discovered by Tomlinson *et al.* [122]. It is shown that the number and weight of low-weight codewords of a binary linear code determine its decoding performance on the erasure channel. Furthermore, for many linear block codes (n, k), their average erasure corrections are almost $n - k$, which is significantly larger than $d_{min} - 1$. Based on such linear codes implicit significance, an extended Dorsch decoder [34] for linear block codes was proposed by Tomlinson *et al.* [121]

over the AWGN channel. Such decoder towards optimum decoding achievement is designated to search the error-vector against a set of low-weight codewords based on the received vector. Inspired by the idea of error-vector searching and the efficient low-weight stopping sets search for LDPC codes, we devise the searching algorithm to find the error-vector, which costs least in terms of cross-correlation. Since the stopping sets search algorithm is more efficient and suitable for linear codes with sparse parity-check matrix, it also determines that the error-vector search algorithm is more suitable for the linear codes in very sparse structure, especially fits the LDPC codes with moderate codeword length.

1.3.2 Organisation of the Dissertation

This dissertation is organised into four parts including nine chapters following this introduction.

Part One: Iterative Decoding

Chapter 2 describes the detail of turbo codes, which includes the codes construction and decoding algorithm. Section 2.2 introduces turbo codes component code *Convolutional code* and its construction and properties. Its corresponding soft-decision decoding algorithm BCJR algorithm is introduced in Section 2.2.5. A range of convolutional code with variant memory orders are evaluated in Section 2.2.8. Turbo codes are introduced in Section 2.3 with its preliminary information. The BCJR based turbo decoder and related extrinsic information exchange are described in Section 2.3.3. Section 2.3.4 shows a range of turbo codes simulation results over the AWGN channel in terms of different interleavers, variant iterations, and different memory orders.

Chapter 3 investigates LDPC codes and turbo Gallager codes in their codes evaluations over the AWGN channel. Section 3.2 describes the preliminary information about LDPC codes encoder and the *belief propagation* (BP) decoding algorithm. The iterative BP decoding algorithm is detailed in Section 3.2.2. Numerical results of LDPC codes with block update and row update are shown in

Section 3.2.3. Turbo Gallager codes is described in Section 3.3, where the numerical results decoded by turbo decoder and BP decoder over the AWGN channel are evaluated in Section 3.3.1.

Investigation of iterative decoding over the erasure channel is presented in Chapter 4. The complexity considerations are generalised in Section 4.2, and the complexity analysis for standard turbo decoder and iteration analysis are presented in Section 4.3. The BP decoding algorithm is analysed in Section 4.4 in terms of the computational complexity. An optimised iterative Look-Up Table (LUT) based decoder over the erasure channel is introduced in Section 4.5. The BCJR algorithm is reviewed and analysed for the erasure channel in Section 4.5.1. Based on the BCJR analysis, the look-up table based iterative decoder is introduced in Section 4.5.3. The comparisons of simulation results by using different iterative decoding algorithms for turbo Gallager codes over the erasure channel are presented in Section 4.5.6.

The DVB-RCS turbo codes are investigated in Chapter 5. The corresponding parity-check matrix construction is determined in Section 5.2. The interleaver design in symbol-based and bit-based is investigated in Section 5.3. The simulation results for DVB-RCS turbo codes on the erasure channel are described in Section 5.3.4. A probabilistic based algorithm is introduced according to the observation on the bit-based interleaver in Section 5.4. The simulation results by adopting the proposed algorithms for DVB-RCS turbo codes are shown in Section 5.4.1.

Part Two: Iterative Coding in Optimal Decoding Arrangement

Chapter 6 presents the hybrid decoding scheme for turbo Gallager codes on the erasure channel. The "In-Place" ML algorithm and its complexity analysis is described in Section 6.2. The hybrid decoding arrangement is described in Section 6.3. The analysis on iterative decoding output is shown in Section 6.3.1, and the complexity analysis for different hybrid arrangements are presented in Section 6.3.2. The numerical results for a range of iterative decodable codes by adopting hybrid decoding arrangements are evaluated in Section 6.3.3.

Chapter 7 describes the iterative decoding arrangement for linear block codes with sparse parity-check matrices, like LDPC codes, over the AWGN channel to help lower the error-floor. By analysing the soft iterative output from the BP decoder, the soft output is conditional selected to help break the error floor caused by the BP stopping sets. Such conditioned output is treated as the input to OSD-i. The new proposed decoding arrangement for linear block code is described in Section 7.2. The new revised iterative soft output is compared to standard soft output with various supporting analysis in Section 7.3. The simulation results by adopting different iterative output for a range of well known LDPC codes are shown in Section 7.4. The remarked decoding results for cyclic LDPC codes over the AWGN channel are presented in Section 7.4.3.

Part Three: Exhaustive Tree Search in Code Spectra and Decoding Algorithm

The exhaustive tree search based algorithm is described in Chapter 8. The code representation and the relevant code definitions are described in Section 8.2. The exhaustive search algorithm for finding stopping sets based on the parity-check matrix and its simplified bound algorithm are presented in Section 8.3. The evaluated stopping sets weight spectra for well known LDPC codes and the class of WiMax LDPC codes are shown in Section 8.4.

By adopting the efficient tree search algorithm, the extended optimum decoding scheme for linear block codes with sparse parity-check matrices over the AWGN channel is detailed in Chapter 9. The tree search algorithm used to find the error pattern according to the searched codeword in small weight, is presented in Section 9.2. The new bounded decoding algorithm in terms of near-optimum decoding scheme is described in Section 9.3. A further improvement on searching the error-vector is introduced in Section 9.4. The optimum decoding results for a range of LDPC codes are presented in Section 9.5.

Part Four: Conclusions and Future Research Work

Chapter 10 describes conclusions and future work direction. Section 10.1 concludes the entire research work. The immediate research direction and potential research work are proposed in Section 10.2. A supplementary proof of the relationship between the received vector and the iterative soft output is shown in Appendix A. The sample of a stopping set, which gives a new representation based on its parity-check matrix, is presented in Appendix B. The comprehensive results for WiMax LDPC codes are shown in Appendix C and D. The submitted and published technical papers for IEEE conferences are attached in Appendix E.

Part I

Iterative Decoding

Chapter 2

Turbo Codes

2.1 Introduction

In this Chapter, we first describe the basis of turbo codes, the component code *convolutional code*, and its soft decoding BCJR algorithm, which leads the success of iterative turbo decoding and the usage of extrinsic information. In Section 2.2, the parameters of convolutional codes are described, and different representative diagrams depict the different graphical view of convolutional codes. The BCJR decoding algorithm is described in Section 2.2.5. And the simulation results for convolutional codes with different memory orders are presented in Section 2.2.8. Section 2.3 describes the construction of turbo code and its trellis diagram for recursive systematic convolutional code. The turbo decoding algorithm on how to exchange the extrinsic information knowledge to help improve the confidence on each information bit is detailed in Section 2.3.3. Section 2.3.4 presents the simulation decoding performances for turbo codes over the AWGN channel.

2.2 Convolutional Codes

Convolutional codes as a class of codes differentiated from block codes were first proposed by Elias [36]. They can be considered as a subset of the *tree* codes, but have distinguishing feature as linear codes, regardless of codeword length. A convolutional code has an encoding unit called *shift-register* produces a continuous stream of encoded output by operating on the information data source. During the encoding process, each information bit stored in the shift-register is capable of affecting a finite number of consecutive symbols in the output stream.

2.2.1 Convolutional Codes Encoding

A convolutional code is primarily defined by three integer parameters (n, k, M), where M denotes the memory size of the shift register, k and n are slightly different from the definitions in block codes. Here k represents the size of the input sequences corresponding to the shift register; and n determines the size of the output sequences. Thus the ratio k/n denotes the code rate for convolutional codes. Let $v = M + 1$ indicate the *constraint length* of the shift-register, it represents the number of k-tuple stages in the encoding shift register. The constraint length represents the number of k-bit shifts over which a single information bit can influence the encoder output. At each unit time t, k bits are shifted into the first k stages of the register; all bits in the register are shifted k stages to the right, and the output of n adders are sequentially sampled to yield the binary code symbols or code bits. Since there are n code bits for each input group of k message bits, the code rate is also k/n message bit per code bit, where $k < n$. For instance, in a shift register encoder with $k = 1$, the message bits are shifted into the encoder one bit at a time. At the ith unit of time, message bit m_i is shifted into the first shift register stage, all the previous bits in the register are shifted by one stage to the right, and the output of the n adders are sequentially sampled and transmitted. Since there are n code bits for each message bit, the code rate is defined as $1/n$. The n code symbols occurring at time t_i comprise the ith branch word, $\mathbf{u_i} = \{u_{1i}, u_{2i}, ..., u_{ni}\}$, where u_{ji}, $j \in \{1, 2, ..., n\}$, is the jth code symbol belonging to the ith branch word.

Figure 2.1 shows a convolutional encoder with polynomial $(7, 5)$ in *octal*, which has constraint length $v = 3$, and shift register structure $(111, 101)$ in *binary*. For such an encoder, there exist $n = 2$ modulo-2 adders and $k = 1$ information bit, then the code rate of convolutional codes $(7, 5)$ is $k/n = \frac{1}{2}$. A set of n connection vectors is utilised to represent the encoder, each vector is for each of the n modulo-2 adders. And each vector has dimension v and describes the

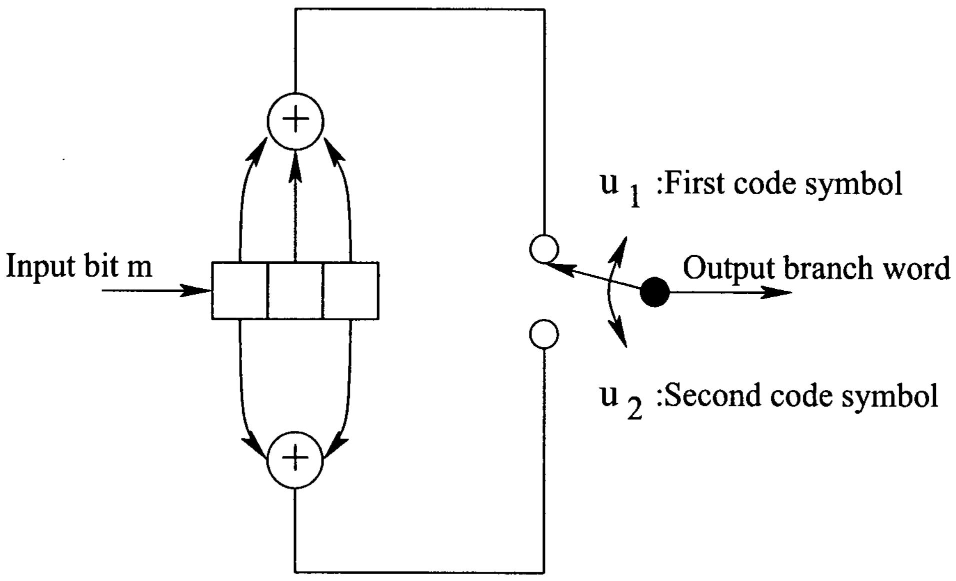


Figure 2.1: Convolutional Code's Encoder $(7, 5)$, $v = 3$

connection of the encoding shift register to that modulo-2 adder. A value "1" in a given position indicates that the corresponding stage in the shift register is connected to the modulo-2 adder; and a value "0" indicates that no connection exists between the stage and the modulo-2 adder. For example in Figure 2.1, the connection vectors can be written as $g_1 = 111$ for the upper connection and $g_2 = 101$ for the lower connection. The encoder connection also could be represented by polynomial representation. Each polynomial is of degree M or less and describes the connection of the encoding shift register to the modulo-2 adder. The coefficient of each term in the M-degree polynomial is either 1 or 0, which depends on whether a connection does exist or not between the shift register and the modulo-2 adder. Thus the above connection vectors could be represented as $g_1(x) = 1 + x + x^2$ and $g_2(x) = 1 + x^2$. The generator matrix $\mathbf{G}$ for this convolutional codes encoder is constructed as:

$$\mathbf{G} = \begin{bmatrix} 11 & 10 & 11 & & & \\ & 11 & 10 & 11 & & \\ & & 11 & 10 & 11 & \\ & & & 11 & 10 & 11 \\ & & & & \ddots & & \ddots \end{bmatrix}$$

Then the generalised generator matrix **G** of a convolutional code with code rate 1/2 and input message size of m is constructed as follows:

$$\mathbf{G} = \begin{bmatrix} g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] & & & \\ & g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] & & \\ & & g_0[0]g_1[0] & & \cdots & g_0[m]g_1[m] \\ & & \ddots & & & \ddots \end{bmatrix}$$

Then the output sequence is concluded as:

$$U(x) = m(x) \cdot g_1(x) \textit{ interlaced with } m(x) \cdot g_2(x) \tag{2.1}$$

For simulation work, the convolutional codes encoder is established by:

1. Defining the shift register with initialisation of all zeroes.
2. Generating the message by v elements.
3. Defining the connection vectors for g_i, $(0 < i < n)$.
4. Shifting the message sequence m_i, $(0 < i < k)$ to produce the elements u_i, $(i < n)$ for each message sequence.
5. Constructing the encoded output until the shift register returns to all zeroes.

2.2.2 Trellis for Convolutional Codes

Due to the specific feature of convolutional codes encoder, it is possible to derive the relationship between the state transitions. In general, the trellis diagram is used to describe convolutional codes, it provides a more manageable encoder description than the tree diagram by exploiting the repetitive structure. The trellis construction is concluded:

- A solid line is to denote the output generated by an input bit zero.
- A dashed line is to denote the output generated by an input bit one.
- The node of the trellis characterises the encoder states. Each set of row nodes corresponds to each distinct state.

- At each unit of time, the trellis requires 2^M sets of nodes to represent the 2^M possible encoder states.

In the constructed trellis, one time-interval section of a fully-formed encoding trellis structure completely defines the code. For instance, Table 2.1 shows all the possible input and output values for a convolutional code $(7,5)$ with $\frac{1}{2}$ code rate, where m_i, $i \in \{0,1,...,M\}$, represents the data stored in the shift register.

Input (m_0)	m_1	m_2	Output (u_1)	Output (u_2)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

Table 2.1: Input and Output of Convolutional Code's Encoder $(7,5)$

And u_1 and u_2 are the corresponding output based on the input and state. The relationship between the states, input and output could be obviously realised. In Table 2.1, columns m_1 and m_2 form the current state, and columns m_0 and m_1 form the next state corresponding to the current state and input value. According to the relationship, Figure 2.2 depicts those state transitions in trellis diagram, where the first digit before "/" represents the input value, followed by the two outputs. The other representation about the relationship, called "*state diagram*", is shown in Figure 2.3, where the 3-digit number represents the input data followed by the two output data.

2.2.3 Decoding of Convolutional Codes

The maximum likelihood hard-decision decoding algorithm, called *Viterbi* decoding, for convolutional codes was first proposed by Viterbi [126], who introduced an optimal recursive decoding algorithm to estimate the possible state sequence by using *Euclidean distance* as a metric. And a forward algorithm working on the

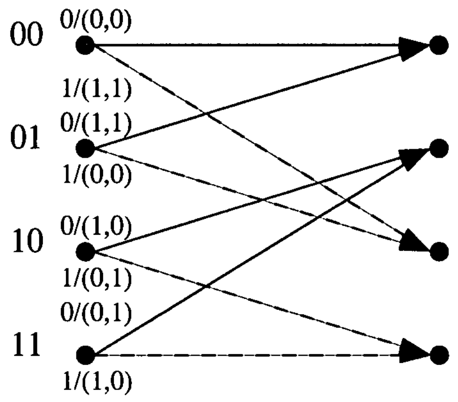


Figure 2.2: Trellis Diagram of Convolutional Codes (7, 5)

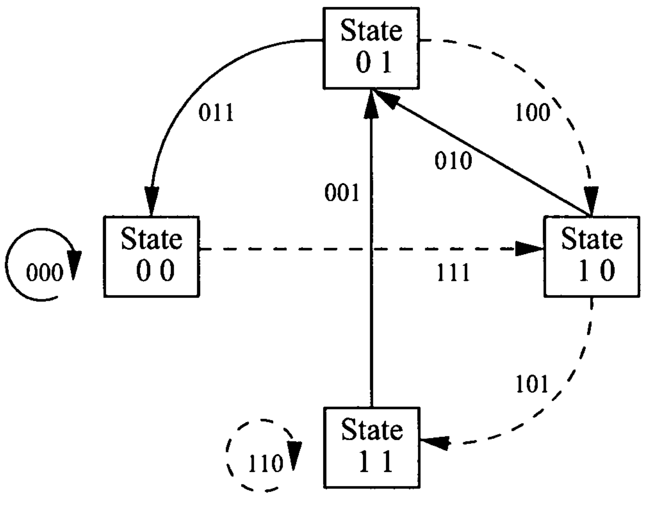


Figure 2.3: State Diagram of Convolutional Codes (7, 5)

trellis for finding the closest codeword was utilised in Viterbi algorithm proposed by Forney [38]. According to the codes trellis, the soft-decision based BCJR decoding algorithm was introduced by Bahl *et al.* [5]. Such decoding algorithm is based on the formation of *a posteriori probabilities* (APPs), and chooses the data-bit value that corresponds to the *maximum a posteriori* (MAP) probability for each data bit [111], thus it is also called MAP decoding. It aims to determine the most likely information bit to have been transmitted at each bit time. The BCJR algorithm could be considered by implementing the Viterbi algorithm in both directions due to its forward and backward recursions.

2.2.4 Likelihood functions

Based on the Bayes' theorem [6], the a posteriori probability (APP) of a decision in terms of a continuous-valued random variable x is expressed:

$$P(d = i|x) = \frac{p(x|d = i)P(d = i)}{p(x)} \; i = \{1, 2, ..., L\} \tag{2.2}$$

and

$$p(x) = \sum_{i=1}^{L} p(x|d = i)P(d = i) \tag{2.3}$$

where $P(d = i|x)$ is the APP value, and $(d = i)$ represents data d belonging to the ith signal class from a set of L classes. $p(x|d = i)$ represents the *probability density function* (PDF) of a received continuous-valued data-plus-noise signal x, conditioned on the signal class $(d = i)$. And $p(d = i)$ called the priori probability, is the probability of occurrence of the ith signal class. Since x is an "observable" random variable or a test statistic that is obtained at the output of a demodulator or some other signal processor. Therefore, $p(x)$ is the pdf of the received signal x, yielding the test statistic over the entire space of signal classes.

2.2.5 BCJR Algorithm

Let $\mathbf{R}_1^k = \{R_1, ..., R_t, ..., R_k\}$ be a received vector corresponding to the output sequence from the channel, where $R_t = \{x_t, y_t\}$ is defined in

$$x_t = |d_t - z_t| \tag{2.4}$$

$$y_t = |Y_t - q_t| \tag{2.5}$$

where z_t and q_t are two independent noises affected by variance σ^2, and Y_t is the parity bit corresponding to the information bit d_t. For computing the probability of a decoded bit d_t, we have

$$P(d_t = i) = P(d_t = i|\mathbf{R}_1^k) \tag{2.6}$$

For each associated node in the trellis, the corresponding APP is defined by

$$P(S_t = m|\mathbf{R}_1^k) \tag{2.7}$$

where $(S_t = m)$ represents the current state m at time t. The corresponding APP of each associated branch in the trellis is defined by

$$P(S_{t-1} = m', S_t = m|\mathbf{R}_1^k) \tag{2.8}$$

where $(S_{t-1} = m')$ represents the previous state m' at time $t-1$ corresponding to the current state m at time t. Thus the probability is computed as the sum of the probability of all transitions that are generated by $d_t = i$, then we have

$$P(d_t = i) = \sum_{m',m|d_t(m',m)=i} P(S_{t-1} = m', S_t = m|\mathbf{R}_1^k) \tag{2.9}$$

By using Bayes' Rule, we have

$$P(d_t = i) = \frac{1}{P(\mathbf{R}_1^k)} \sum_{m',m|d_t(m',m)=i} \lambda_t(m', m) \tag{2.10}$$

where

$$\lambda_t(m', m) = P(S_{t-1} = m', S_t = m, \mathbf{R}_1^k) \tag{2.11}$$

is the joint probability of $S_t = m$ and $S_{t-1} = m'$. The term $P(\mathbf{R}_1^k)$ could be ignored due to its constancy. Then the $\lambda_t(m', m)$ can be expanded based on Bayes' rules as follow:

$$\begin{aligned} \lambda_t(m', m) &= P(S_t = m, S_{t-1} = m', \mathbf{R}_1^k) \\ &= P(\mathbf{R}_{t+1}^k|S_t = m) \cdot P(S_t = m, R_t|S_{t-1} = m') \cdot \\ &\quad P(S_{t-1} = m', \mathbf{R}_1^{t-1}) \end{aligned} \tag{2.12}$$

We define the terms of $\alpha_t(m)$, $\beta_t(m)$, $\gamma_t(m', m)$ as follows:

$$\begin{aligned} \alpha_t(m) &= P(S_t = m, \mathbf{R}_1^t) && (2.13) \\ \beta_t(m) &= P(\mathbf{R}_{t+1}^k | S_t = m) && (2.14) \\ \gamma_t(m', m) &= P(S_t = m, R_t | S_{t-1} = m') && (2.15) \end{aligned}$$

where $t \in \{0, ..., k\}$. By applying the terms of $\alpha_t(m)$, $\beta_t(m)$, $\gamma_t(m', m)$, we have the joint probability

$$\lambda_t(m', m) = \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m) \qquad (2.16)$$

where α computes the probability of state $S_{t-1} = m'$ based on the received data vectors $\mathbf{R}_1^{t-1}$ before time t, β computes the probability of state $S_t = m$ based on the received data vector $\mathbf{R}_{t+1}^k$ after time t, and γ is the transition probability based on the current received value R_t.

2.2.5.1 The α Recursion

By following the Bayes' rule, the α value can be obtained recursively. Then we have

$$\begin{aligned} \alpha_t(m) &= \sum_{m'} P(S_{t-1} = m', S_t = m, \mathbf{R}_1^t) \\ &= \sum_{m'} P(S_{t-1} = m', S_t = m, \mathbf{R}_1^{t-1}, R_t) \\ &= \sum_{m'} P(S_t = m, R_t, S_{t-1} = m', \mathbf{R}_1^{t-1}) \\ &= \sum_{m'} P(S_t = m, R_t | S_{t-1} = m', \mathbf{R}_1^{t-1}) \cdot P(S_{t-1} = m', \mathbf{R}_1^{t-1}) \quad (2.17) \end{aligned}$$

The events of S_t and R_t after time $t-1$ do not depend on the knowledge of $\mathbf{R}_1^{t-1}$. If S_{t-1} is known, thus

$$\begin{aligned} \alpha_t(m) &= \sum_{m'} P(S_{t-1} = m', \mathbf{R}_1^{t-1}) \cdot P(S_t = m, R_t | S_{t-1} = m') \\ &= \sum_{m'} \alpha_{t-1}(m') \cdot \gamma_t(m', m) \qquad (2.18) \end{aligned}$$

2.2.5.2 The β Recursion

Similarly, values of β also can be obtained recursively by using Bayes' rule. Then we have:

$$\begin{aligned}
\beta_t(m) &= \frac{P(\mathbf{R}_{t+1}^k, S_t = m)}{P(S_t = m)} \\
&= \frac{\sum_{m'} P(\mathbf{R}_{t+2}^k, S_{t+1} = m', R_{t+1}, S_t = m)}{P(S_t = m)} \\
&= \frac{\sum_{m'} P(\mathbf{R}_{t+2}^k | S_{t+1} = m', R_{t+1}, S_t = m) \cdot P(S_{t+1} = m', R_{t+1}, S_t = m)}{P(S_t = m)} \\
&= \sum_{m'} P(\mathbf{R}_{t+2}^k | S_{t+1} = m', R_{t+1}, S_t = m) \cdot \\
&\quad P(S_{t+1} = m', R_{t+1} | S_t = m) \qquad (2.19)
\end{aligned}$$

Similarly, the event of $\mathbf{R}_{t+2}^k$ after time t does not depend on the knowledge of $\mathbf{R}_1^{t+1}$ and $S_t = m$. Then if S_{t+1} is known, thus

$$\begin{aligned}
\beta_t(m) &= \sum_{m'} P(\mathbf{R}_{t+2}^k | S_{t+1} = m') \cdot P(S_{t+1} = m', R_{t+1} | S_t = m) \\
&= \sum_{m'} \beta_{t+1}(m') \cdot \gamma_{t+1}(m, m') \qquad (2.20)
\end{aligned}$$

2.2.5.3 The Transition Probability

The transition probability $\gamma_t(m', m)$ at time t could be derived:

$$\begin{aligned}
\gamma_t(m', m) &= P(S_t = m, R_t | S_{t-1} = m') \\
&= \frac{P(R_t, S_t = m, S_{t-1} = m')}{P(S_{t-1} = m')} \\
&= \frac{P(R_t | S_t = m, S_{t-1} = m') \cdot P(S_t = m, S_{t-1} = m')}{P(S_{t-1} = m')} \\
&= P(R_t | S_t = m, S_{t-1} = m') \cdot P(S_t = m | S_{t-1} = m') \qquad (2.21)
\end{aligned}$$

In (2.21), the first term is computed with the statistical description of the channel. The second term is a "0" - "1" probability distribution based on the trellis, where "1" indicates the possible transition, and "0" vice versa. If the condition of the transition is possible, then we have

$$\gamma_t(m', m) = P(R_t | S_t = m, S_{t-1} = m') \tag{2.22}$$

Since $R_t = \{x_t, y_t\}$, then we have

$$\gamma_t(m', m) = P(x_t, y_t | S_t = m, S_{t-1} = m') \tag{2.23}$$

where x_t, y_t is mutually exclusive, then we have

$$\gamma_t(m', m) = P(x_t | S_t = m, S_{t-1} = m') \cdot P(y_t | S_t = m, S_{t-1} = m') \tag{2.24}$$

where the transition probability is defined by the product of the probability of received information bit value x_t and the probability of received parity bit value y_t in condition of from state m' to state m.

2.2.6 Implementation of BCJR Decoder

We conclude the steps of BCJR decoding algorithm implementation for binary codes, where $i \in \{0, 1\}$ as follows:

1. The first step is to calculate the channel probabilities of $P(d_t = 0)$ and $P(d_t = 1)$ for each received information bit and corresponding parity bit.
2. Compute the γ value for each transition in terms of probability at each section.
3. Based on the trellis of the code, and the computed γ value at each section, implement the forward recursion to get α value for each state.
4. Based on the trellis of the code, and the corresponding computed γ value at each section, implement the backward recursion to get β value for each state.
5. According to (2.16), get the soft output about each information bit.
6. Transform the soft output value into the decoded symbols.

2.2.7 Tail Biting: the Terminology of Block Convolutional Codes

In general, additional bits are necessary for terminating the trellis during the decoding process for convolutional codes. Consequently, it may cause a large fractional code rate loss, a modified form of terminated code, called *tail-biting* convolutional codes, was introduced by Ma & Wolf [78]. Such a modified class of convolutional codes is capable of transforming convolutional codes into sized block codes, which allows no code rate loss transmission. The tail-biting block convolutional codes C_{TB} are those code sequences associated with paths in the trellis that start from a state equal to the last m bits of an information vector of k data bits [86]. Thus all codewords in C_{TB} generated by the corresponding encoder begin and end at the same state.

For example, a convolutional codes encoder $(7, 5)$, which is with code rate $1/2$ and memory of 2 for 5 message bits, could be modified to be a $(10, 5)$ block code with tail biting. Then the corresponding generator matrix $\mathbf{G}_{C_{TB}}$ is given by

$$\mathbf{G}_{C_{TB}} = \begin{bmatrix} 11 & 10 & 11 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 11 & 10 & 11 \\ 11 & 00 & 00 & 11 & 10 \\ 10 & 11 & 00 & 00 & 11 \end{bmatrix}$$

2.2.8 Numerical Results

In general, in a given code rate, the performance of a convolutional code could be viewed as a function of the codes constraint length, which is the error probability is improved with the increased constraint length [128]. Thus the comparison results between convolutional codes $(7, 5)$ with $M = 2$ and $(133, 171)$ with $M = 6$, (both codes have 200 binary input message bits), decoded by BCJR decoder over the AWGN channel are shown in Figure 2.4. It provides a good evidence that the convolutional codes $(133, 171)$ with $M = 6$ achieve significant better performance than convolutional codes $(7, 5)$ with $M = 2$. And there exists exactly 2dB coding gain at *Bit Error Rate* (BER) 4×10^{-7} between the two results. Such

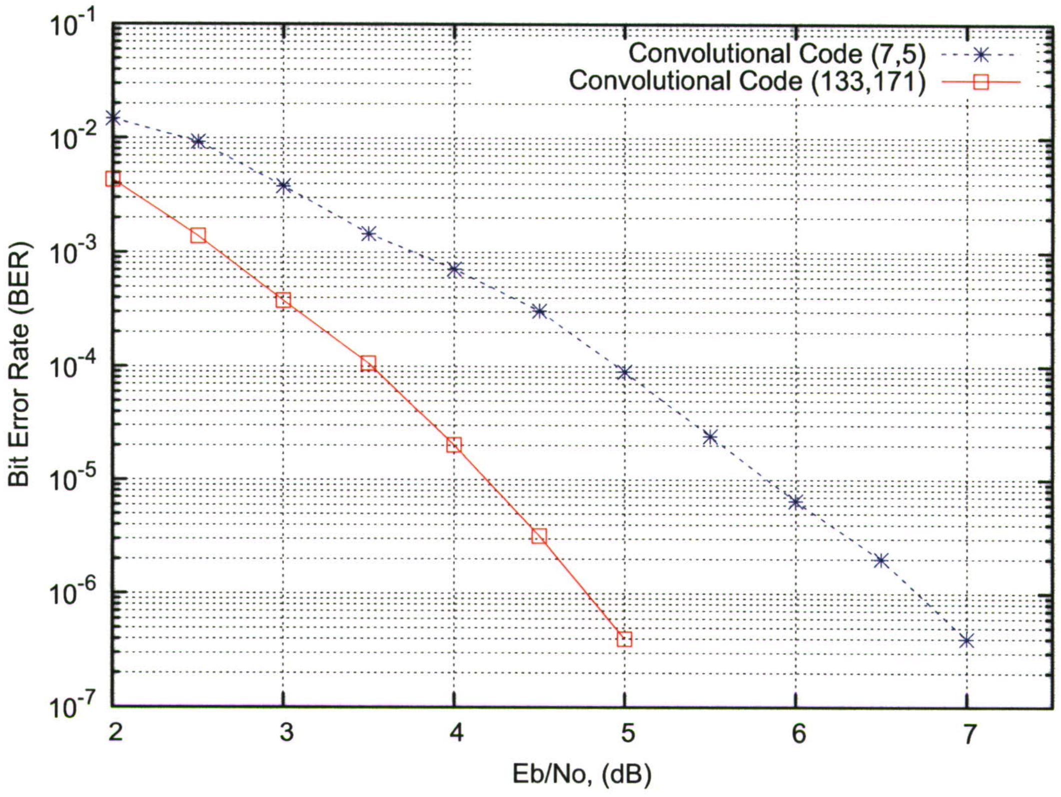


Figure 2.4: Results of Convolutional Codes with BCJR Decoder

improvement comes at the expense of increased decoder complexity, for instance, larger memory of shift register defines more complicated number of states in trellis representation, which could lead to more specific codewords.

2.3 Turbo Codes

In 1993, Berrou *et al.* [10] introduced a new class of codes, called *turbo codes.* Such code could purportedly achieve near Shannon limit performance with modest decoding complexity. This new class code with significant performance stunned the coding research community, and the "*turbo revolution*" was launched. Since then, the new style decoding scheme jumped to the core of coding design, which was the iterative decoding scheme by exchanging the extra soft information knowledge between independent decoders during iterations.

2.3.1 Turbo Codes Encoder

With the successful usage of soft output during the decoding process for convolutional codes by adopting the BCJR algorithm, the *parallel* concatenation and *serial* concatenation of convolutional codes attracted huge interest during the digital communication design. Since the codes should be "random-like", in the sense that the distribution of distances from a typical codeword to all other codewords should resemble the distance distribution in a random code [25]. The standard turbo codes encoder is designated to comprise two parallel *recursive systematic convolutional* (RSC) encoders and an interleaver which enables the turbo codes to follow the somewhat "random-like" principles. Figure 2.5 shows a turbo encoder structure in polynomials of (15/13), where u and u' is the information bit and

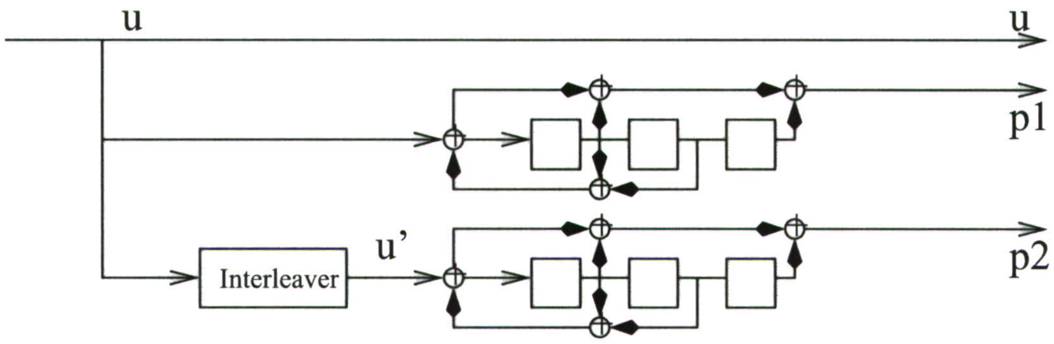
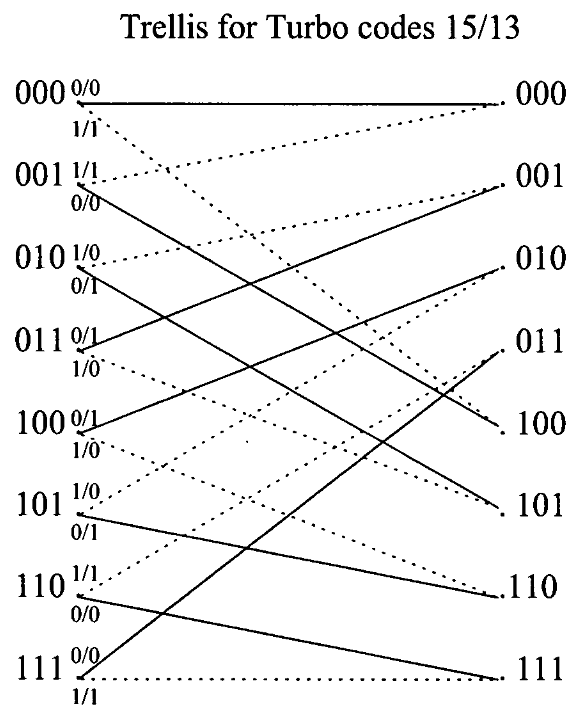


Figure 2.5: Parallel Concatenation Turbo Encoder (15/13) with $R_c = \frac{1}{3}$

the permuted information bit, p_1 and p_2 are the parity bits corresponding to the first RSC encoder and the second permuted RSC encoder. Let $\mathbf{c}$ be the code set containing the information bit and two parity bits, $\mathbf{c} = \{u, p_1, p_2\}$. Figure 2.6 depicts turbo codes (15/13) in the trellis diagram and the input-output computations, where X is the hidden value in the end of the feedback shift register, which is the input for forward shift register. And values of a, b and c represent the current state; X, a, b represent the next state stored in the shift register. In the trellis diagram, the solid line indicates the "0" value input, and the value "1" input is represented by the dashed line. The input and output values through the shift register are depicted in form of "input/output" in Figure 2.6.



Data for trellis 15/13

Input	X	a	b	c	Output
0	0	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	0
0	0	1	1	1	0
1	1	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	1	1	1	1

Figure 2.6: Trellis Diagram of Turbo Codes (15/13)

2.3.2 Interleaver

The idea of designing *interleaver* was originally aimed to solve the *error propagation* for the concatenation codes, which could cause an error occurrence results in a number of data errors. The *block* interleaver consists of a two-dimensional array, into which the data is read along its rows. Then the columns once the array is full, thus the order of data is permuted. For turbo codes, the interleaver unit with pseudo-random property is normally utilised in large block length of codeword. It is required that the designed interleaver is able to break the patterns in input sequence; break up almost all the bad patterns; and achieve the full effort of spectral thinning in order to ensure that the low-weight parity sequences from the first encoder get matched with high-weight parity sequence from the second encoder.

At present, there are two types of interleaver that have been commonly investigated in terms of pseudo-random, they are *random* interleaver and so-called *S-random* interleaver [33, 104]. For pseudo-random interleaving patterns, there are some ways to design the interleaver.

- By using a primitive polynomial to generate a maximum-length shift register sequence where cycle structure determines the permutation.

- An interleaver [71] uses a computationally simple algorithm based on the quadratic congruence to generate an index mapping function $c_i \rightarrow c_{i+1}($ mod $K)$,

$$c_i \equiv \frac{j \cdot i(i+1)}{2} (\text{mod } K), \ 0 \leq i < K, \tag{2.25}$$

  where K is the interleaver size, and j is an odd integer. For instance, for $K = 16$ and $j = 1$, we obtain

$$(c_0, c_1, \cdots, c_{15}) = (0, 1, 3, 6, 10, 15, 5, 12, 4, 13, 7, 2, 14, 11, 9, 8)$$

  which implies that index 0 (input bit u'_0) in the interleaved sequence $\mathbf{u}'$ is mapped into index 1 in the original sequence $\mathbf{u}$ (i.e., $u'_0 = u_1$), index 1 in $\mathbf{u}'$ is mapped into index 3 in $\mathbf{u}$ (i.e., $u'_1 = u_3$), then we can get permutation information bits index set.

$$\prod_{16} = [1, 3, 14, 6, 13, 12, 10, 2, 0, 8, 15, 9, 4, 7, 11, 5]$$

- S-random interleaver is also called "spread" interleaver, it was recognised early on that good spreading properties are desirable for both fast convergence and good distance properties.

In order to attain highly spread property, Crozier [27, 30] introduced the dithered golden interleavers with high spread and high distance for turbo codes. In order to design a good interleaver, there are several criteria to satisfy:

- The interleaver bank should provide a wide range of interleaver lengths with good resolution.

- The amount of memory required to define and store each interleaver should be low
- The algorithm used to generate the interleaver indices should be simple to accommodate the term of "on-the-fly".
- The interleaver bank should provide good error rate performance for all block lengths, which is the hardest part of interleaver design.

In order to satisfy the above criteria, Crozier & Guinand [28] proposed a new class of interleavers, so called *dithered relative prime* (DRP), which provides a good solution to the interleaver bank design for turbo codes. Especially such class of interleaver could significantly improve the minimum distance in terms of high-spread and high-distance. In this research work, most of the turbo codes are designated by adopting the DRP interleaver designs [29]. Since the size of an interleaver could be large, then this number of bits must inevitably be stored in the interleaver in the encoder and/or the decoder at any given time. Thus there exists a *latency* issue due to the large size of interleaver. It results in a delay through the encoder/decoder combination. For instance, for an information rate of 8kbit/s with interleaver size of 65536 bits (64k bits), which is appropriate for speech transmission, there is a delay of 65536/8=, or more than 8 seconds. Such delay is unacceptable in a telephone system.

2.3.3 Turbo Decoder

By adopting the BCJR algorithm [5], the optimal decoding scheme for the new class of turbo codes was introduced by Berrou *et al.* [10]. In turbo decoding scheme, a new term is suggested, called *extrinsic information*, which is designated to provide the confidence about each information bit during the iterative turbo decoding process. Thus the updated reliable information knowledge about the information bit is re-computed by each BCJR decoder at every iteration. In general, the extrinsic information is the information knowledge which contains the previous reliable knowledge to help compute the current reliable knowledge. A turbo decoding scheme comprises two BCJR decoders for the constituent information sequence and the permuted constituent information sequence. During the decoding iterations, more and more reliable information knowledge is exchanged between the BCJR decoders, and the most likely information bit is determined in terms of the *maximum a posteriori* probability for each information bit. The parallel concatenated iterative turbo decoder structure is shown in Figure 2.7, where

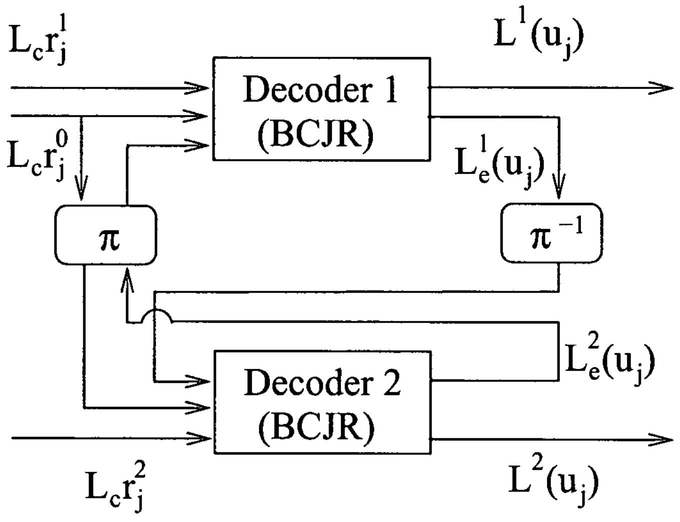


Figure 2.7: Iterative Decoding Scheme for Parallel Concatenated Turbo Codes

"Decoder 1" and "Decoder 2" are the BCJR decoders, π is the same interleaver function as used in the encoder, π^{-1} is the deinterleaver. Let $L^l(u_j)$ be the soft output APP from decoder l, where $j \in \{0, ..., k-1\}$ and $l \in \{1, 2\}$. And $L_e^l(u_j)$ represents the extrinsic information about bit u_j from decoder l, it is passed as a priori input to the other decoder. Let $L_c r_j^i$ be the channel measurement from received code set $\mathbf{r}_j$, where $\mathbf{r}_j$ comprises of r_j^0 the received information bit, r_j^1 and r_j^2 the received parity bit and the received permuted parity bit corresponding to set code $\mathbf{c}_j$. Since the channel measurement with a priori input has already been known prior to the decoder, hence the extrinsic information provides additional knowledge about the corresponding information bit at each iteration. When the iterative decoder reaches the given maximum number of iterations, the soft output should be more reliable than by only passing to one single BCJR decoder.

By taking the logarithm of the likelihood ratio, we obtain the *log-likelihood ratio* (LLR) metric, which is a real number to represent a soft decision output.

It is described as:

$$L(d|x) = \log \left[\frac{P(d=1|x)}{P(d=0|x)}\right] \tag{2.26}$$

By following Bayes' rule, it could be extended as:

$$\begin{aligned} L(d|x) &= \log \left[\frac{p(x|d=1)P(d=1)}{p(x|d=0)P(d=0)}\right] \\ &= \log \left[\frac{p(x|d=1)}{p(x|d=0)}\right] + \log \left[\frac{P(d=1)}{P(d=0)}\right] \end{aligned} \tag{2.27}$$

Then we have

$$L(d|x) = L(x|d) + L(d) \tag{2.28}$$

where $L(x|d)$ is the LLR of the channel measurement under the conditions that $(d=1)$ or $(d=0)$ may have been transmitted, and $L(d)$ is the a priori LLR of the data bit d. At beginning, $L(d)$ is always initialised as 0. For simplification, (2.28) is rewritten as:

$$L'(\hat{d}) = L_c(x) + L(d) \tag{2.29}$$

where $L_c(x)$ is the result of a channel measurement made at the receiver. According to [10], Berrou has shown that for a systematic code the LLR soft output of the decoder is equal to

$$L(\hat{d}) = L'(\hat{d}) + L_e(\hat{d}) \tag{2.30}$$

where $L'(\hat{d})$ is the LLR of the input to the decoder. $L_e(\hat{d})$, called the *extrinsic* LLR, represents the extra knowledge that is gleaned from the iterative decoding process. From (2.29) and (2.30), the output LLR of turbo decoder is obtained as:

$$L(\hat{d}) = L_c(x) + L(d) + L_e(\hat{d}) \tag{2.31}$$

Equation (2.31) shows that the output LLR of a systematic decoder can be represented with three LLR terms:

- Channel measurement, $L_c(x)$
- Priori knowledge of the data, $L(d)$
- Extrinsic LLR stemming solely from the decoder, $L_e(\hat{d})$

During the iterative decoding, the channel measurement of $L_c(x)$ stays same, which could be considered as a constant. And the priori knowledge is replaced by the extrinsic knowledge $L_e(\hat{d})$ from the previous decoder. If the sign of $L_e(\hat{d})$ due to the decoding has the same sign as $L_c(x) + L(d)$, it acts to help improve the reliability of $L(\hat{d})$, otherwise vice versa. When reaching the maximum iterations, the most reliable extra knowledge is accumulated with the channel measurement to get the final output LLR of the data bit. The principle of the iterative decoding process is depicted in Figure 2.8.

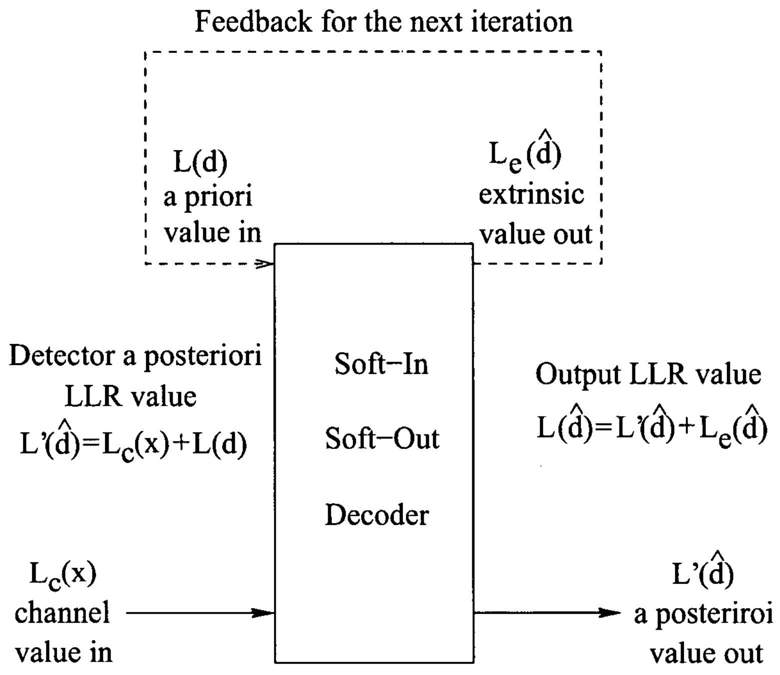


Figure 2.8: Principle Diagram of an Iterative Turbo Decoder

2.3.4 Numerical Results

The turbo decoding performances with different iterations for turbo codes (5/7) with length (3074, 1024) permuted by DRP [28] interleaver are shown in Figure 2.9. It clearly shows that the significant improvements are achieved by more

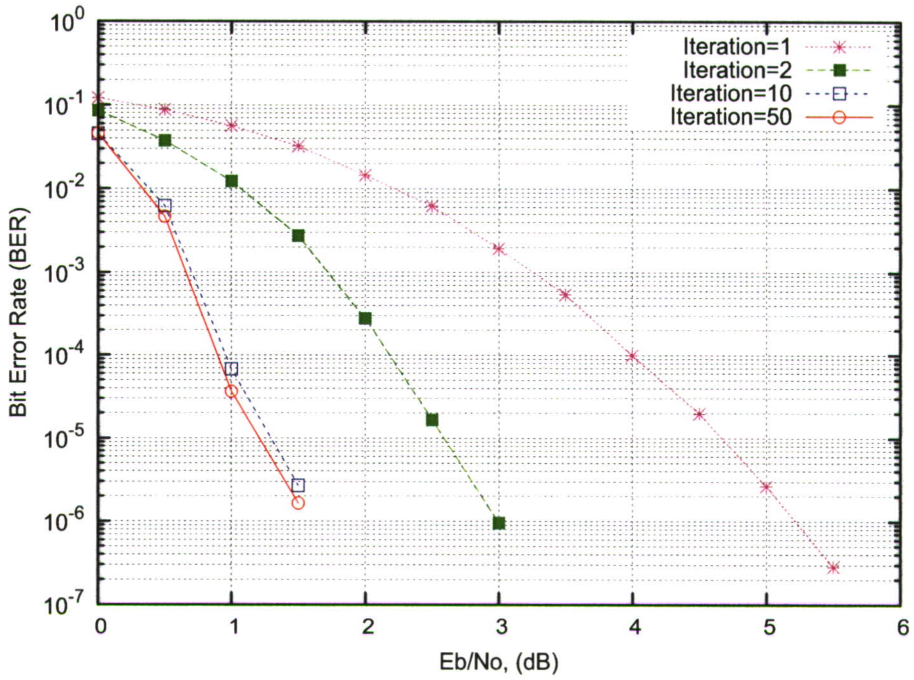


Figure 2.9: Results of Turbo Codes (5/7) with Different Iterations

decoding iterations.

The turbo decoding performances with different iterations for turbo code (15/13) with length (3074, 1024) permuted by DRP [28] interleaver are shown in Figure 2.10. It does not only show the improved achievement over iterations, but also shows better performance is achieved by using larger memory of (15/13) than results of (5/7) in Figure 2.9.

The turbo decoding performances for turbo codes (15/13) with length (1368, 456) and 50 iterations in different interleavers are shown in Figure 2.11. It clearly shows that different interleaver could significantly affect the decoding performance, especially at high SNR. A well designed interleaver could asymptotically break most of stopping set patterns to achieve a much lower error floor, like code DRP-rp-r13-d38 than DRP-d35 and Srandom-r12-d32. Although their minimum distances are close, but they perform quite different in the AWGN channel.

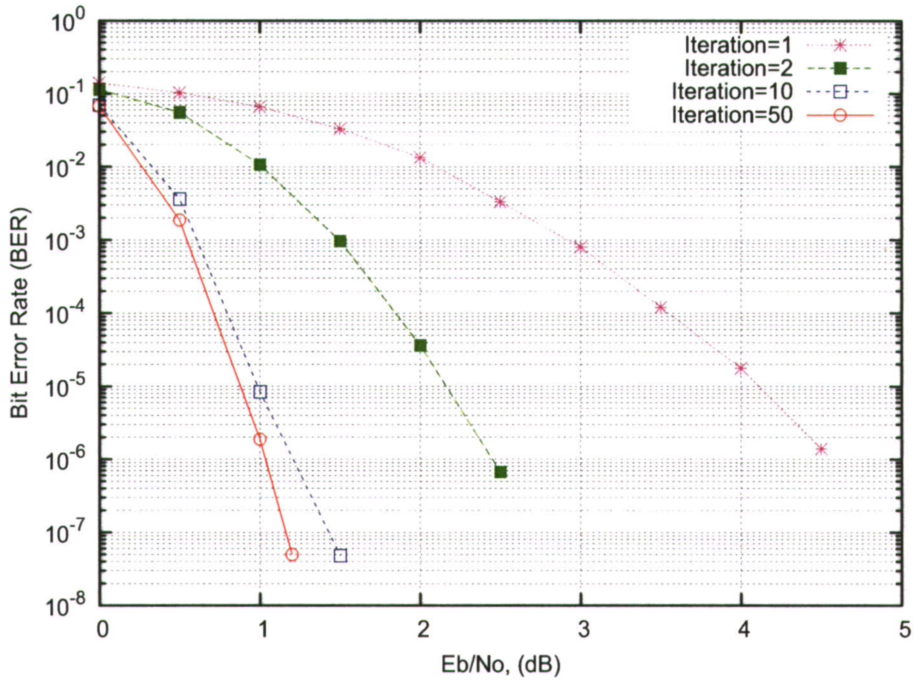


Figure 2.10: Results of Turbo Codes (15/13) with Different Iterations

2.4 Summary

The SISO BCJR decoding algorithm for convolutional codes has been explained and derived in equations in this Chapter. The extrinsic information has been introduced for turbo codes and explained in equations to provide the benefit of using extrinsic information to improve the confidence about each information bit after certain iterations. And the simulation results over the AWGN channel have been presented by adopting different interleavers, memory orders, codeword lengths and iterations.

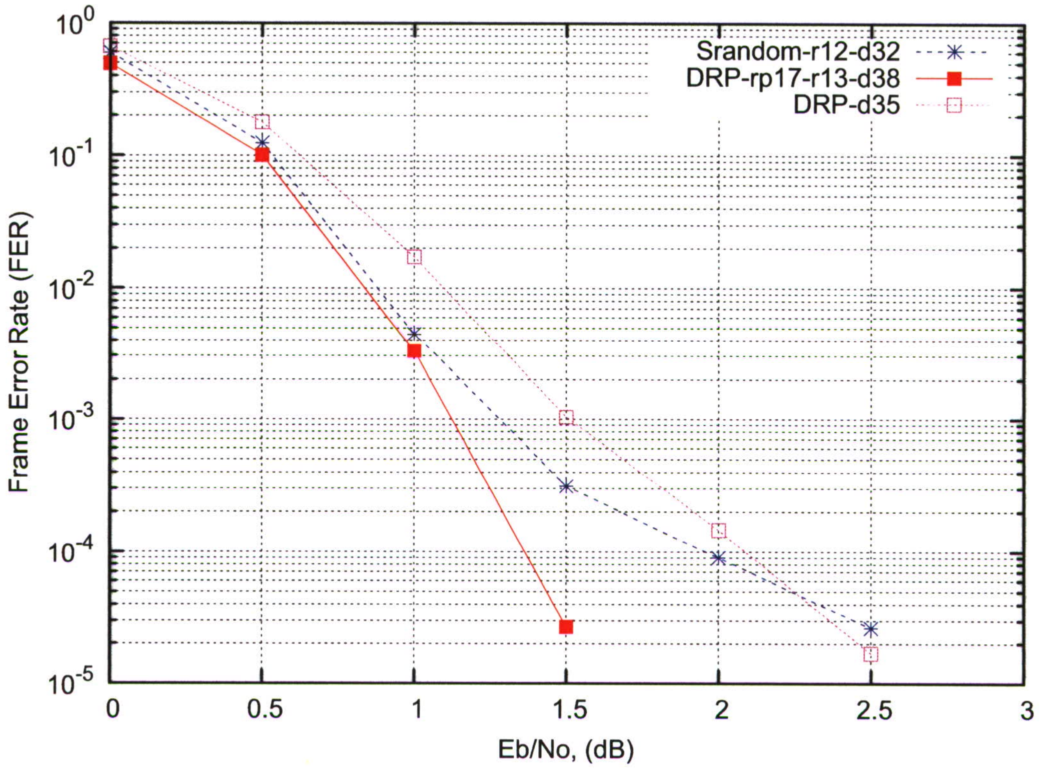


Figure 2.11: Results of Turbo Codes (15/13) with Different Interleavers

Chapter 3

LDPC Codes and Turbo Gallager Codes

3.1 Introduction

Low-density parity-check (LDPC) codes are one of the traditional types of linear block codes since their first introduction by Gallager [45] in 1963 and have attracted a great deal of interest in recent years, following the rediscovery by MacKay [80] in 1999. The so-called LDPC code with its sparse structure in parity-check matrix shows its perfect match of message-passing algorithm. The corresponding message passing decoding algorithm (also known as *belief propagation* (BP) introduced by Pearl [88]) as an iterative decoding algorithm has successfully brought traditional codes back into the modern digital communications research area after the invention of turbo codes [10] in 1993. The competition between LDPC codes and turbo codes has probably peaked, although the outcome is by no means clear. Following the introduction of irregular LDPC codes [95], the asymptotic approach to the Shannon limit by LDPC codes coupled with iterative decoding has proven the benefits of the approach [23]. In this chapter, the construction and characteristic of LDPC codes are described in Section 3.2. The classical low decoding complexity algorithm of message passing (BP) is detailed in Section 3.2.2. Section 3.2.3 presents the simulation results over the AWGN channel for LDPC codes following different extrinsic information update methods. The turbo Gallager codes, by combining the simplicity of turbo encoding feature and BP or BCJR decoding properties, are detailed in Section 3.3. Section 3.3.1 presents a series of turbo Gallager codes evaluation results by adopting BP decoding and BCJR decoding over the AWGN channel.

3.2 LDPC Codes

Low-Density Parity-Check (LDPC) codes were first introduced by Gallager [45] in 1960. Due to the high computational complexity of designing the code structure and superior competition from concatenated Reed-Solomon and convolutional codes, it was completely ignored for decades. In 1999, LDPC codes were rediscovered by MacKay [80], it was shown that LDPC code was a suitable candidate for iterative decoding with low decoding complexity. He also indicated that in practice moderate-length LDPC codes ($10^3 - 10^4$ bits) could attain near Shannon-limit performance [81]. Whereas [110, 114] showed that in theory, as codeword length $n \rightarrow \infty$, they could approach the Shannon limit with linear decoding complexity. Besides the regular LDPC codes, the irregular LDPC codes, which have varied column weights from column to column, have proved the channel-capacity approachable performance [23, 95]. Comparing to turbo codes, LDPC codes are featured by the following advantages:

- LDPC codes do not require a long interleaver to achieve good error performance.
- LDPC codes generally have better frame error performance.
- LDPC codes could have a much lower error floor at low BER level (about 10^{-6} - 10^{-7}).
- The corresponding BP decoding is not a trellis based iterative decoding algorithm, which provides low decoding complexity.

Although Gallager proposed LDPC codes as a class of error correcting codes, he did not provide a specific method on how to construct good LDPC codes algebraically and systematically. A method of constructing LDPC codes in terms of pseudo-random was proposed in Gallager's PhD thesis [45]. MacKay rediscovered Gallager's work and proved the possible random method of generating good LDPC codes based on the sparse matrices MacKay [80]. Thus good random LDPC codes have been largely found by computer generation, especially long codes, but their encoding is very complex due to the lack of structure. In 2000, Kou, Lin and Fossorier [61] proposed the first algebraic and systematic construction of LDPC codes based on finite geometries. Such large classes of LDPC codes based on finite geometries have relatively good minimum distances, and contain no short cycles on their Tanner graphs. These codes can be decoded with low decoding complexity with good error performance. Besides the finite geometry branch from the combinatorial mathematics, other combinatorial mathematics are adopted to the construction of LDPC codes. Based on the Kirkman triple systems, Johnson & Weller [60] introduced another approach of $(3, k)$-regular LDPC codes, whose Tanner graph is free of 4-cycles for any k. Another construction of LDPC based on Steiner 2-designs was proposed by Vasic & O.Milenkovic [125]. Inspired by Vasic's work, another class of quasi cyclic LDPC codes based on the so-called *balanced incomplete block design* (BIBD) [13, 49], which forms another branch of combinatorial mathematics, was introduced by Ammar *et al.* [3]. Such BIBD-LDPC codes by exploiting decomposition techniques could significantly reduce the number of cycles of length 6. Since these codes are either *cyclic* or *quasi-cyclic* (QC) LDPC codes [21, 41], their encoding could be simply implemented with linear shift registers, whose computational complexity is linearly proportional to the number of parity-check bits for serial coding and to the code length for parallel encoding [69]. In general, long random LDPC codes perform closer to the Shannon limit than their equivalent structured LDPC codes, but with high encoding complexity. On the other hand, structured LDPC codes, especially cyclic or QC-LDPC codes, are featured by the simple encoding advantage over random codes. In fact, for practical lengths, well designed structured LDPC codes could perform equally well as their equivalent random LDPC codes with less encoding complexity in terms of bit-error rate, frame-error rate and error-floor collectively [72, 73]. The powerful capabilities of LDPC codes have led to their recent inclusion in several standards, such as IEEE 802.16e (WiMax), IEEE 802.20, IEEE 802.3, DVB-RS2.

LDPC codes are a class of linear block codes whose parity-check matrix is sparse. Thus the structure of LDPC codes could be either described by the generator matrix **G** or the parity-check matrix **H**. The capability of correcting symbol errors in a codeword is determined by the minimum distance (d_{min}). In **H** matrix, d_{min} is the least number of linearly dependent columns of the **H** matrix. The regular LDPC code is designed by w_c and w_r ($w_c < w_r$), which has w_c ones in each column and w_r ones in each row. Thus the parity-check matrix is constructed by at least $(n-k)$ independent rows and n columns, where k is the length of the information bits. The capacity of the codeword is defined by

$$R_c = 1 - w_c/w_r \tag{3.1}$$

Since the parity-check matrix of LDPC codes could be represented by a bipartite graph, such a graph comprises a vertex set of *variable nodes* $\mathbf{V} = \{v_0, ..., v_{n-1}\}$, and a vertex set of *check nodes* $\mathbf{C} = \{c_0, ..., c_{m-1}\}$. The variable nodes correspond to the columns of **H**, and the check nodes correspond to the rows of **H**. H_{ji} represents the connection relationship between the variable node v_i and check node c_j.

3.2.1 LDPC Codes Encoding

According to [116], all block codes have the common properties, then the generator matrix **G** is constructed as

$$\mathbf{G} = [\mathbf{I}_k | \mathbf{P}] \tag{3.2}$$

where $\mathbf{I}_k$ is the $k \times k$ *identity matrix*, which only has "1"s in the diagonal positions, **P** is a $k \times (n-k)$ matrix that determine the $n-k$ redundant bits or parity bits. Since $\mathbf{G} \times \mathbf{H} = 0$, then the parity-check matrix **H** is defined by

$$\mathbf{H} = [\mathbf{I}_{n-k} | \mathbf{P}^T] \tag{3.3}$$

where $\mathbf{P}^T$ is the transpose of **P** matrix with size of $(n-k) \times k$, $\mathbf{I}_{n-k}$ is the $(n-k) \times (n-k)$ identity matrix. The design of the LDPC codes is normally based on the parity-check matrix, which is not exactly same as the structure as shown in (3.3). Gaussian elimination is an linear algebra algorithm which can be used to find the rank of a matrix and calculate the inverse of an invertible square matrix [4]. By using Gaussian elimination algorithm, the $\mathbf{H}$ matrix of LDPC codes can be transformed into an echelon form, which satisfies the structure of (3.3). By transposing the $\mathbf{P}^T$ matrix into $\mathbf{P}$, we can get the generator matrix $\mathbf{G}$, according to (3.2). Let $\mathbf{C}$ be a codeword vector, and $\mathbf{X}$ be a message information vector, then the encoding process in terms of the generator matrix $\mathbf{G}$, the parity-check matrix $\mathbf{H}$ is defined:

$$\mathbf{C} = \mathbf{X} \times \mathbf{G} \tag{3.4}$$

$$\mathbf{C} \times \mathbf{H}^T = 0 \tag{3.5}$$

It is noted that the number of parity-check equations m might be larger than $n-k$ for some types of LDPC codes, for instance Tanner codes $(155, 64, 20)$ by Tanner *et al.* [118] having 93 parity-check equations. Thus the parity-check matrix $\mathbf{H}$ could be represented by a $m \times n$ matrix, where $m \geq (n-k)$ and there exists $n-k$ independent parity-check equations, which define the $n-k$ parity bits, and $(m-n+k)$ dependent parity-check equations, which have been found that the additional parity-check equations could help eliminate the low weight stopping sets [2].

3.2.2 Belief Propagation Decoding

The *belief propagation* algorithm as an instance of *message passing* algorithm is recommended by Gallager [45] for LDPC codes decoding. Here we describe the general steps for the implementation:

1. **Input:** Computation of posteriori probabilities for $Pr_j(d_j = 0)$ and $Pr_j(d_j = 1)$ for each code bit c_j, $j \in \{0, 1, ..., (n-1)\}$, where $\mathbf{c} = \{c_0, c_1, ..., c_{n-1}\}$ represents the code vector, and a maximum number of iteration is set as L

2. **Initialisation:** Set $q_{ij}(0) = Pr_j(d_j = 0)$ for all $(h_{ij} = 1)$, where h_{ij} is the edge value between variable node i and check node j of parity-check matrix $\mathbf{H}$, $i \in \{0, 1, ..., (m-1)\}$. Set $q_{ij}(1) = Pr_j(d_j = 1)$ for all $(h_{ij} = 1)$.

3. **Horizontal Computation:**

   (a) For each $h_{ij} = 1$, computation of $\delta q_{ij} = q_{ij}(0) - q_{ij}(1)$ to form the $\delta \mathbf{q}$ matrix; "0" is filled in the matrix, where $h_{ij} = 0$.

   (b) For each $h_{ij} = 1$, let δr_{ij} be the product of $\delta \mathbf{q}$ matrix elements along its row i, excluding the (i, j) position.

$$\delta r_{ij} = \prod_{l \in \{0,1,\ldots,(n-1)\} \backslash j} \delta q_{il} \tag{3.6}$$

   (c) Computation of $r_{ij}(1) = (1 - \delta r_{ij})/2$ and $r_{ij}(0) = (1 + \delta r_{ij})/2$.

4. **Vertical Computation:** For each $h_{ij} = 1$, let $q_{ij}(0)$ be the product along its column j, excluding position (i, j), then times $Pr_j(d_j = 0)$, same criteria is applied for $q_{ij}(1)$.

$$q_{ij}(0) = \alpha_{ij} \times Pr_j(d_j = 0) \prod_{l \in \{0,1,\ldots,(m-1)\} \backslash i} r_{lj}(0) \tag{3.7}$$

$$q_{ij}(1) = \alpha_{ij} \times Pr_j(d_j = 1) \prod_{l \in \{0,1,\ldots,(m-1)\} \backslash i} r_{lj}(1) \tag{3.8}$$

   where α_{ij} is the factor in condition of $q_{ij}(0) + q_{ij}(1) = 1$.

5. Computation of probabilities $q_j(0)$ and $q_j(1)$

$$q_j(0) = \alpha_j \times Pr_j(d_j = 0) \prod_{l \in \{0,1,\ldots,(m-1)\}} r_{lj}(0) \tag{3.9}$$

$$q_j(1) = \alpha_j \times Pr_j(d_j = 1) \prod_{l \in \{0,1,\ldots,(m-1)\}} r_{lj}(1) \tag{3.10}$$

   where α_j is the normalisation factor in condition of $q_j(0) + q_j(1) = 1$

6. Decision on $c_j = 1$, if $(q_j(1) > 0.5)$, else $c_j = 0$.

7. If $\mathbf{H}^T \times \mathbf{c} = 0$, it means code vector $\mathbf{c}$ is a valid codeword. Otherwise repeat steps from **Horizontal Step**, if number of iterations < maximum iteration L, else a decoding failure is indicated.

3.2.3 Numerical Results

There are two different schedules for passing the extrinsic information to the next decoding process.

- **Row Update:** involves passing the extrinsic information about the bit to the next equation, which has "1" at the same position, for getting the latest posteriori probability value. The decoding proceeds from the top equation to the bottom, each bit is decoded with the most updated extrinsic information.

- **Block Update:** involves passing the extrinsic information about each bit from last iteration to the current decoding round. At each equation, the same extrinsic information for the bit is used to get the posteriori probability. The posteriori probability provides the current extrinsic information and it is stored to get the average extrinsic information about the bit during the current iteration. Then the accumulated average extrinsic information is passed to the next iteration.

The results for QC-LDPC codes [22] $(824, 415)$ and regular LDPC codes [80] $(2048, 1024)$ by following the row update and the block update are shown in Figure 3.1. It is clear to see that there exist slight differences between block update and row update, the different schedules of passing extrinsic information may affect the performance slightly better or worse at different SNR. The LDPC codes $(2048, 1024)$ with longer codeword length, which might indicate a larger d_{min} after a well design, are capable of achieving a better performance than shorter QC LDPC codes $(824, 415)$. But the longer codes have the error floor occurring much earlier than the shorter codes, which might be due to the poor smaller size of stopping sets, which might be poorly designed. And it reaches the error floor at a very early stage of 2dB $\frac{E_b}{N_o}$, meanwhile QC LDPC codes $(824, 415)$ achieve a lower error floor. From the results, we can see that the well designed LDPC codes are able to lower the error floor due to their better minimum distance d_{min} and larger stopping sets' weight. According to the partial stopping set distribution for the testing codes as shown in Table 3.1, LDPC codes $(2048, 1024)$ with poor d_{min} 6 could not provide an ideal lower error-floor performance, since its small

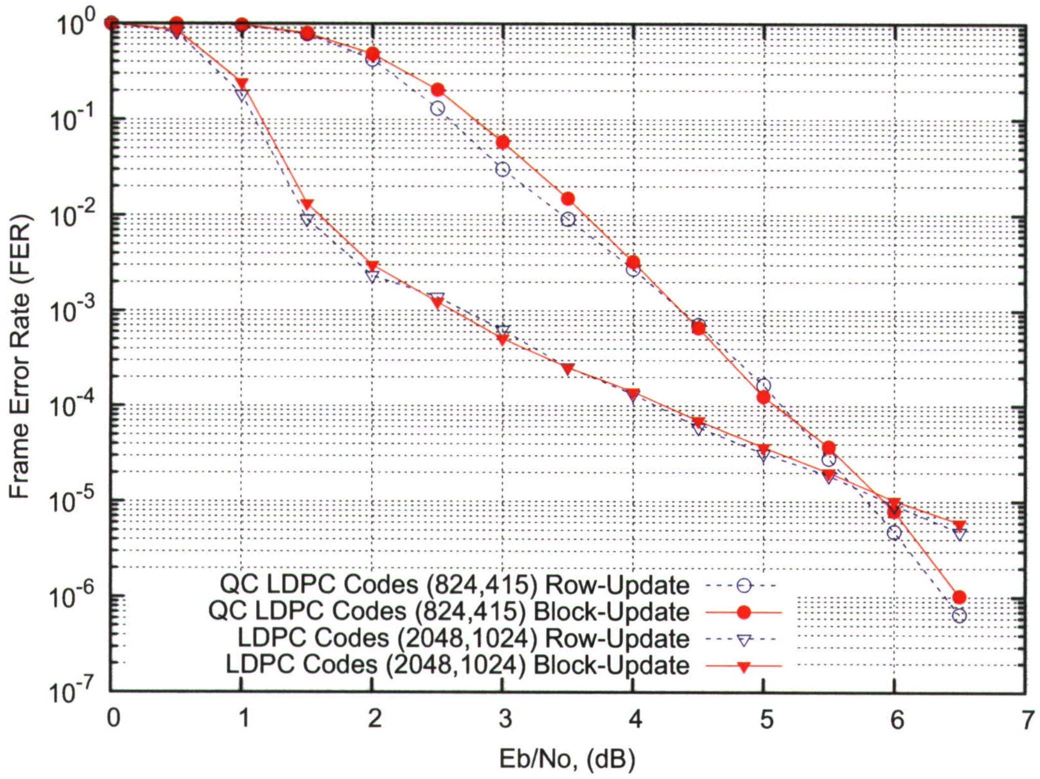


Figure 3.1: Comparison Results for LDPC Codes with BP Decoding

multiplicity of the stopping sets distribution, they performs better at smaller SNR. On the other hand QC codes (824, 415) with higher d_{min} 9 successfully provide a lower error floor than the longer codes.

Code Name	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
$(824, 415)$	9	1 (0)	18 (0)	36 (0)	4146 (3296)	4572 (0)
$(2048, 1024)$	6	1 (1)	1 (1)	0 (0)	3 (3)	1 (0)

Table 3.1: Partial stopping set distribution of codes $(824, 415)$ and $(2048, 1024)$

3.3 Turbo Gallager Codes

In the modern digital communication system, LDPC codes and turbo codes are the major competitors, which could achieve channel-capacity performance by using the iterative decoding technique. In practical, the fundamental difference between turbo codes and LDPC codes is

- Turbo codes tend to have low encoding complexity but high decoding complexity. In contrast, LDPC codes tend to low decoding complexity but with relatively high encoding complexity.
- The interleaver during the turbo codes design causes the latency issue, which does not exist in the LDPC codes design. In addition, the corresponding BP decoding algorithm for LDPC codes could be implemented in a fully parallel manner, and can be closely approximated with decoders of very low complexity.
- Even though, both codes are suffered by the error-floor issue, the well structured LDPC codes could generally have a lower error-floor than turbo codes in the AWGN channel.

The trade-offs between turbo codes and LDPC codes could vary based on the specific system design requirements. Then an open question could be simply asked? Is it possible to adapt the advantages from each type of code to design a new class of codes in terms of low complexity? The answer is definite. In 2004, Colavolpe [24] introduced a new class of codes, which is so called the *Turbo Gallager codes* (TGC). It is a special type of turbo codes which can be successfully decoded by means of the decoding algorithm used for LDPC codes by properly choosing the component convolutional codes. Since BP decoding algorithms are very simple and characterised by a decoding complexity which does not directly depend on the component code *constraint length*, they can be used to decode the turbo codes with a enlarged constraint length, and hence, potentially characterised by a large free distance. The large free distance for turbo codes is hard to implement in practical, since the BCJR decoding complexity grows exponentially with the code constraint length. By using Forney's normal graphs [39], common features can be identified between LDPC codes and turbo codes. Both codes can be represented in graph structure joined by a randomly selected permutation π. This graphical representation also exposes the turbo decoding algorithm as an instance of message passing decoding on code graph [84]. It was originally suggested by Tanner [117], that LDPC codes could be represented by *bipartite graphs*, also called *Tanner* graph. These graphs include two sets of nodes, the *variable nodes*, which represent the elements of a codeword; and the *check nodes*, which correspond to the set of parity-check constraints, which define the code. By considering turbo codes containing RSCs in the similar structure of regular LDPC codes, the Tanner graph of turbo codes can be represented in Figure 3.2, where

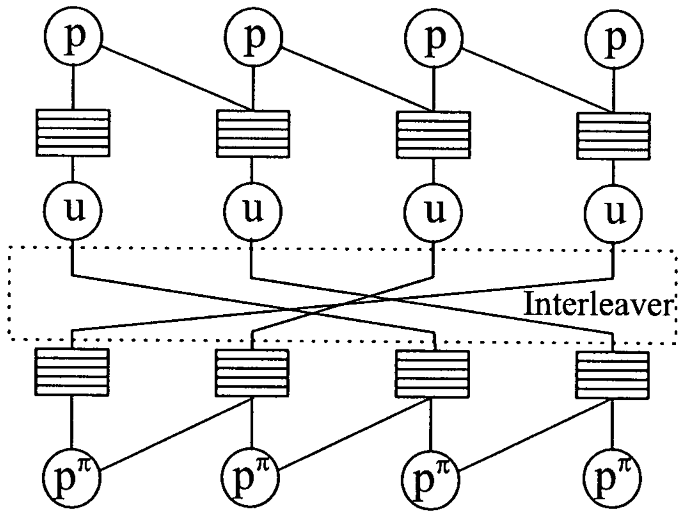


Figure 3.2: Tanner Graph for Turbo Codes

u is the information sequence, p and p^π are the parity bits for the encoder and the interleaved encoder respectively. As the figure shown, it is a graph for a rate 1/3 code. The component codes are rate 1/2 systematic codes, and for simplicity, the code termination is ignored. Let $\hat{p}$ be the set of parity bits corresponding to the systematic information bit u, where $\hat{p}_i = \{p_i, p_i^\pi\}$ and $i \in \{0, 1, ..., k-1\}$. And $c_i = \{u_i, \hat{p}_i\}$, where $i \in \{0, 1, ..., k-1\}$. In the upper part of the graph, the information bits u_i and the parity bits p_i of the first component code are connected by the corresponding check nodes. The permuted information bits u_i^π are also connected with the lower check nodes related to the second component code having parity bits p_i^π. Let c be a code set corresponding to information u, which comprises u and $\hat{p}$. After transmission over the noise channel, the received codeword set c' is depicted as $c' = \{u', \hat{p}'\}$.

According to the algebraic rules for selecting the proper turbo codes as *turbo Gallager codes* (TGCs) [24], here we consider the turbo codes which are recursive convolutional codes of rate $1/n$, where the codes polynomial exponent differences are constrained to be distinct in order to ensure that message-passing decoding can work successfully on the Tanner graph of the overall code due to the absence of cycles of least length 4. For instance, the standard UMTS turbo code (15/13) in binary is (1101/1011), which corresponds to the polynomials $1 + x + x^3$ and $1 + x^2 + x^3$. For each polynomial the exponent differences are distinct and form difference sets $(1, 2, 3)$ and $(2, 1, 3)$. Hence, we consider the UMTS turbo codes (15/13) as the basic turbo Gallager codes in this research work, due to its excellent iterative performance and small memory in turbo decoding complexity.

3.3.1 Numerical Results

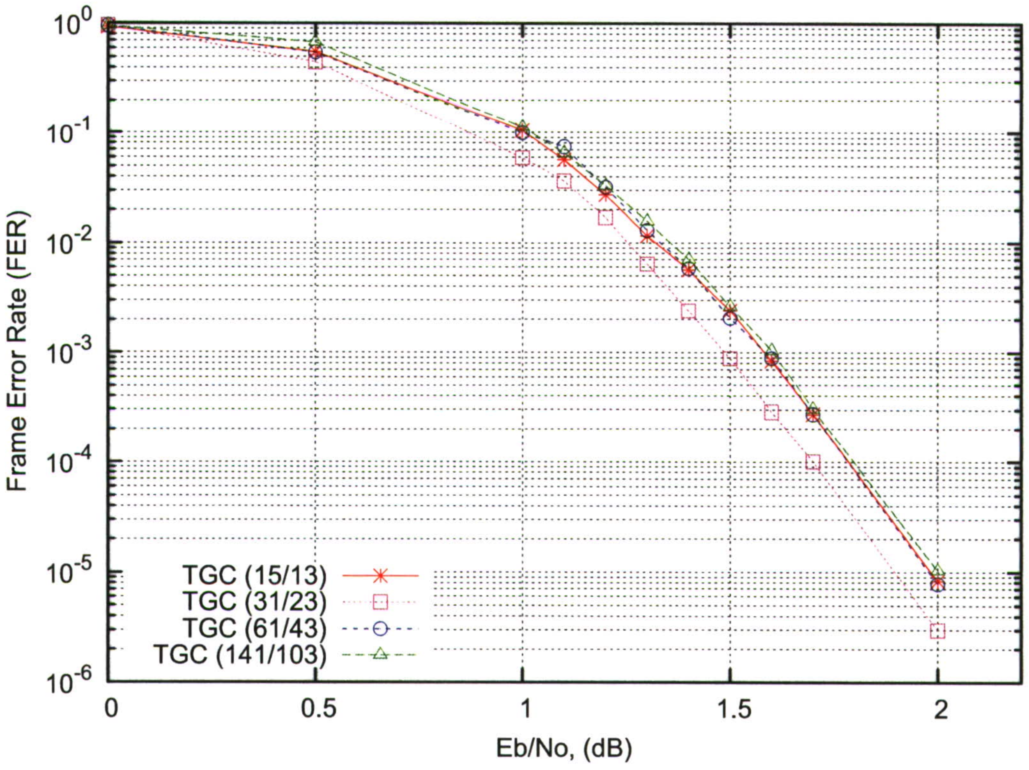


Figure 3.3: The Results of TGCs $(1536, 512)$ by Turbo Decoder

As turbo Gallager codes can be decoded by either turbo decoder or BP decoder, then we consider the UMTS turbo codes structure as the investigative code structure. By adding more "0"s in the middle of the shift register in structure of (11...1/1...11), the memory of the encoder is increased. Then the designed turbo Gallager codes in size of $(1536, 512)$ with DRP [29] interleavers include (15/13), (31/23), (61/43), (141/103) and (6001/4003). The results of these codes excluding (6001/4003) decoded by turbo decoder with 50 iterations over the AWGN channel are shown in Figure 3.3. From the comparison results, we can see that most of the turbo Gallager codes achieve the similar performances over the AWGN channel, excluding TGC (31/23) differing with 0.2 coding gain. Thus turbo codes with turbo decoder don't correspond the similar performance as convolutional codes do that the performance is improved by increasing the memory order of the shift register. The reason of having no improvement by increasing the memory order could be due to the interleaver design, which has not successfully separated the connected information bits after permutation and resulted in a smaller weight of stopping sets or similar minimum distance. On the other hand, the complexity of turbo decoder is exponentially increased by the increment of shift register memory. Thus there should exist the convergence issue between the iterative decoder and the design of turbo codes in terms of memory order. Here we consider TGC (15/13), which includes a low decoding complexity, as the reference performance for the further comparison.

The results of the set of "UMTS-like" codes decoded by BP decoder are shown in Figure 3.4. As the results shown, by increasing the memory of the shift register, the performance of BP decoder is improved for most cases. Furthermore, the performance of TGC (141/103) seems to reach the best moderate performance in BP decoder with maximum affordable memory for turbo decoder. Although the performance of TGC (6001/4003) achieves lower error floor and starts outperforming since 3dB $\frac{E_b}{N_o}$ by enlarging 5 memory orders, it is impossible for BCJR decoder to decode. The reason of causing error-floor for codes with larger memory like (31/23) worse than codes (15/13) could be the low-weight BP stopping sets for codes (31/23), which does not converge properly by using turbo codes interleaver design for BP decoder.

According to [24], it was proposed a component code $(0, 3, 4/0, 14, 34)$ for code rate $1/n$, where the integer number indicates the index of polynomial with

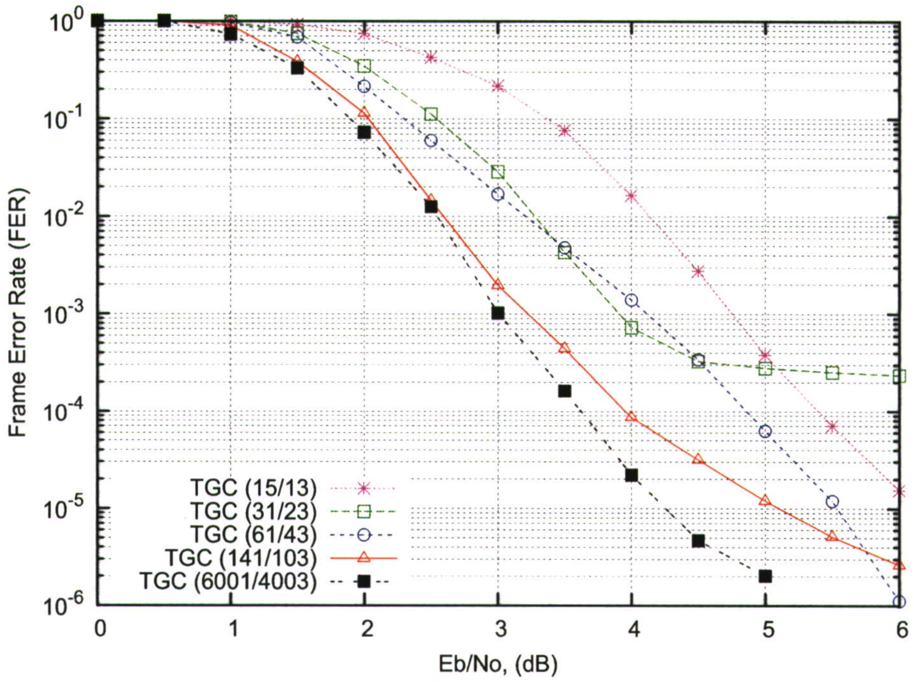


Figure 3.4: The Results of TGCs (1536, 512) by BP Decoder

coefficient “1”. And such code is evaluated and compared in the following research work, it is noted that such code with memory order 35 is only doable for BP decoder. Then we compare the results of TGCs (15/13) and (141/103) by BCJR decoder and BP decoder and the performance of code $(0, 3, 4/0, 14, 34)$ by BP decoder at the same size and number of iterations as shown in Figure 3.5. The code $(0, 3, 4/0, 14, 34)$ achieves the best BP decoding performance than TGC (141/103), the reason could be the longer memory order 35 of the code enables the message passing work through the entire codeword [24]. However, the BCJR decoding results for TGCs (15/13) and (141/103) still achieve the better performances than the best BP result of code $(0, 3, 4/0, 14, 34)$ with difference of at least 1dB coding gain.

3.4 Summary

In this chapter, the LDPC codes have been described and its corresponding iterative decoding algorithm has been explained. The simulations results have shown

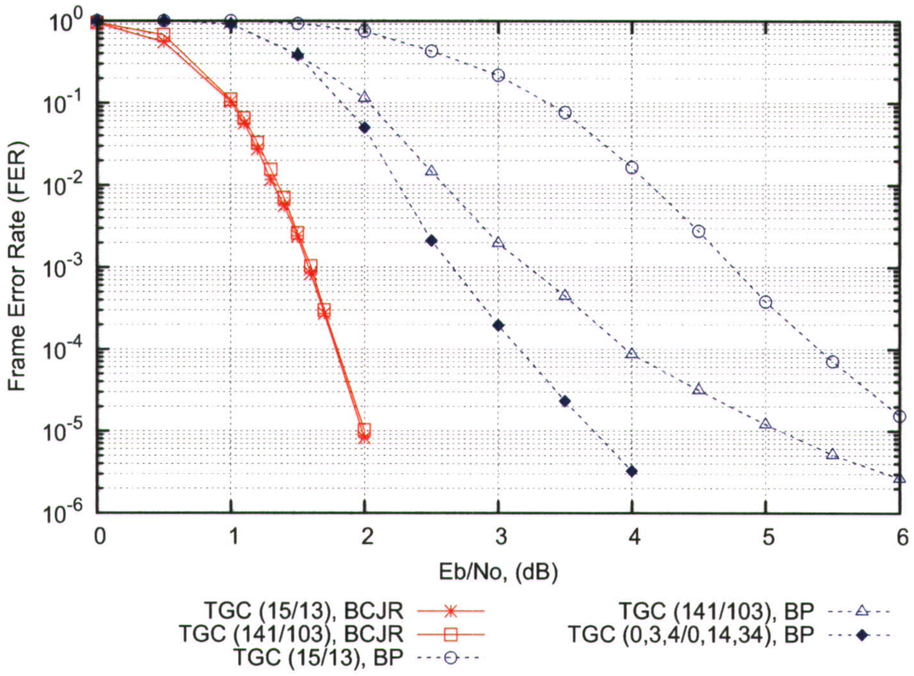


Figure 3.5: The Results of TGCs (1536,512) in the AWGN Channel

the impact from error floor and update schedules on the decoding performance. The turbo Gallager codes has been introduced in terms of the Tanner graph and normal code graph. The simulation results over the AWGN channel for a range of turbo Gallager codes have been compared by using BP decoder and BCJR decoder.

Chapter 4

Iterative Decoding in the Erasure Channel

4.1 Introduction

Chapter 2 and Chapter 3 describe the study of turbo codes, LDPC codes and their iterative decoding algorithms over the AWGN channel. In this chapter, the relevant decoding algorithms are analysed in terms of computational complexity for the erasure channel. Section 4.2 is to clarify the various operations in terms of equivalent additions to help analyse the algorithms' computational complexity. The standard turbo decoding complexity is analysed and discussed in Section 4.3. The evaluations for a range of turbo codes with different iterations over the erasure channel are presented in Section 4.3.3. Section 4.4 describes the complexity analysis for belief propagation decoding algorithm over the erasure channel. According to the feature of the erasure channel, the optimised look-up table decoder for turbo codes is introduced in Section 4.5. The BCJR algorithm is reviewed and analysed for the erasure channel in Section 4.5.1. The decoding complexity analysis for LUT decoder is discussed and evaluated in Section 4.5.5. The comparison numerical results for turbo Gallager codes over the erasure channel are presented in Section 4.5.6. The relevant decoding complexity analysis for iterative BP decoder and BCJR decoder was presented in [131].

4.2 Complexity Analysis Considerations

The purpose of the complexity analysis is to determine the relative speed of the different decoders. The decoding algorithms considered for iterative decodable codes are optimised LUT decoder, standard turbo decoder and belief propagation decoder. The LUT decoder requires the look-up table to be constructed once, and this has no impact on decoding speed. Consequently the look-up table construction complexity is not included during the comparison of different decoding algorithms. A decoding complexity analysis between MAP, Max-Log-MAP and Log-MAP algorithms was presented by Robertson *et al.* [97]. In order to simplify the comparison, it was assumed that logical and mathematical operations have similar complexity. A more thorough investigation on turbo decoding algorithm was performed by Wu [130], where each operation is quantified as a number of *equivalent additions.* A re-derived complexity analysis for those decoding algorithms was investigated by Chatzigeorgiou *et al.* [20]. In the erasure channel, the iterative decoding algorithms are much simpler than the AWGN channel. In our analysis, the basic operations performed by the various decoding algorithms include addition (ADD), subtraction (SUB), multiplication (MUL), division (DIV), comparison (CP), assignment (ASSI) and table look-up (LKUP). The ASSI operation is to assign a relative value to a variable. The LKUP operation is used in the optimised LUT decoder, it corresponds to three equivalent additions, since 3 CP operations are used to map the input parameters to the decoded value stored in the look-up table. The number of equivalent additions for the various operations is shown in Table 4.1.

Operations	Number of Equivalent Additions
Addition, Subtraction	1
Multiplication, Division	1
Comparison	1
Assignment	1
Table Look-up	3

Table 4.1: Number of Equivalent Additions Per Operation

4.3 Standard Turbo Decoding

4.3.1 Complexity Analysis

To determine the complexity of a coding scheme, the arithmetic complexity of the iterative decoding algorithm is measured in terms of number of elementary arithmetic operations per decoded bit. In general, the complexity of encoding is neglected, as it is relatively small compared to decoding complexity. Due to the properties of the erasure channel, the MAP algorithm is sufficient to provide the accurate conditional probability for each data bit. Thus in this comparison, the BCJR algorithm is considered as the standard turbo decoding algorithm. Let n_e be the number of data erasures at the input to the decoder, where $\bar{n_e} = (k \cdot \epsilon)$, k is the length of information bits. At each decoding iteration, the number of erasures is reduced. Let n_e^i be the number of erasures at iteration i, and 2^M be the number of states in the trellis, where M is the encoder memory, N_o represents the required number of operations in equivalent additions, L is the maximum number of iterations.

In standard turbo decoding algorithm, the required procedures are classified as follows:

1. **Branch Metrics Calculation** (Proc. A), requires 2 CPs and 2 ASSIs for each data bit and each corresponding parity bit, 1 MUL operation for updating the data bit estimation. For each valid branch, it consists of 2 CPs and 1 MUL. There are 2^{M+1} branches in total.

2. **Forward Metrics Calculation** (Proc. B), each state consists of 4 CPs, 2 MULs and 1 ADD for both branches. There are 2^M states in total. Normalisation requires $2^M - 1$ ADDs and 2^M DIVs.

3. **Backward Metrics Calculation** (Proc. C), the computation is the same as that in the forward metric calculation.

4. **Soft Decoding** (Proc. D), computation for $d_t = 1$ includes 3 CPs and 2 MULs for each state, where d_t is the data bit at time t; there are 2^M states in total. Computation for $d_t = 0$ is same as $d_t = 1$. The soft decision is

summed by $2^M - 1$ ADDs for $d_t = \pm 1$. Normalisation includes 1 ADD and 2 DIVs.

Procedure	ADD	MUL	DIV	CP	ASSI
Proc. A	–	$1 + 2 \times 2^M$	–	$4 \times 2^M + 2$	2
Proc. B	$2 \times 2^M - 1$	2×2^M	2^M	4×2^M	–
Proc. C	$2 \times 2^M - 1$	2×2^M	2^M	4×2^M	–
Proc. D	$2 \times 2^M - 1$	4×2^M	2	6×2^M	–

Table 4.2: Number of Computations for Standard Turbo Decoding Algorithm

The required computations for the turbo decoding algorithm in terms of the number of equivalent additions are shown in Table 4.2. In a 1/3 code-rate turbo decoding scheme, in decoding one frame, k trellis sections and 2 corresponding BCJR decoders are required. The number of operations N_o^T in terms of equivalent additions for k trellis sections with L iterations is obtained as:

$$N_o^T(k) = \sum_{i=1}^{L}(4k + 74k2^M) \tag{4.1}$$

4.3.2 Iteration Analysis

We define that the block successfully decoded by iterative decoder is denoted as "Decoded"; the block contains erasures after iterative decoder is denoted as "Undecoded". The required average maximum numbers of iterations are analysed for both types of blocks. Table 4.3 shows the average maximum numbers of iterations required for "Decoded" and "Undecoded" at different erasure probability ϵ, where "Avg. Iter." represents the average maximum number of iterations.

The relationship between efficiency and iterations for TGC (15/13) in size of (1536, 512) with standard turbo decoder is shown in Figure 4.1. It is apparent that at iteration 6, the decoder has already achieved over 90% decoding performance. According to Table 4.3 and Figure 4.1, an ideal maximum number of iterations is provided for standard turbo decoder over the erasure channel, which is much smaller than the number of iterations required in the AWGN channel without performance degradation.

Erasure Probability	Avg. Iter. Decoded	Avg. Iter. Undecoded	Frame Error Rate
0.63	9	2	0.981818
0.62	8	2	0.927272
0.61	7	3	0.790000
0.60	6	3	0.560000
0.59	5	3	0.236000
0.58	4	4	0.088333
0.57	4	4	0.022000
0.56	3	4	0.003160
0.55	2	4	0.000312
0.54	2	4	0.000018

Table 4.3: Iterations Analysis of Standard Turbo Decoder in the Erasure Channel

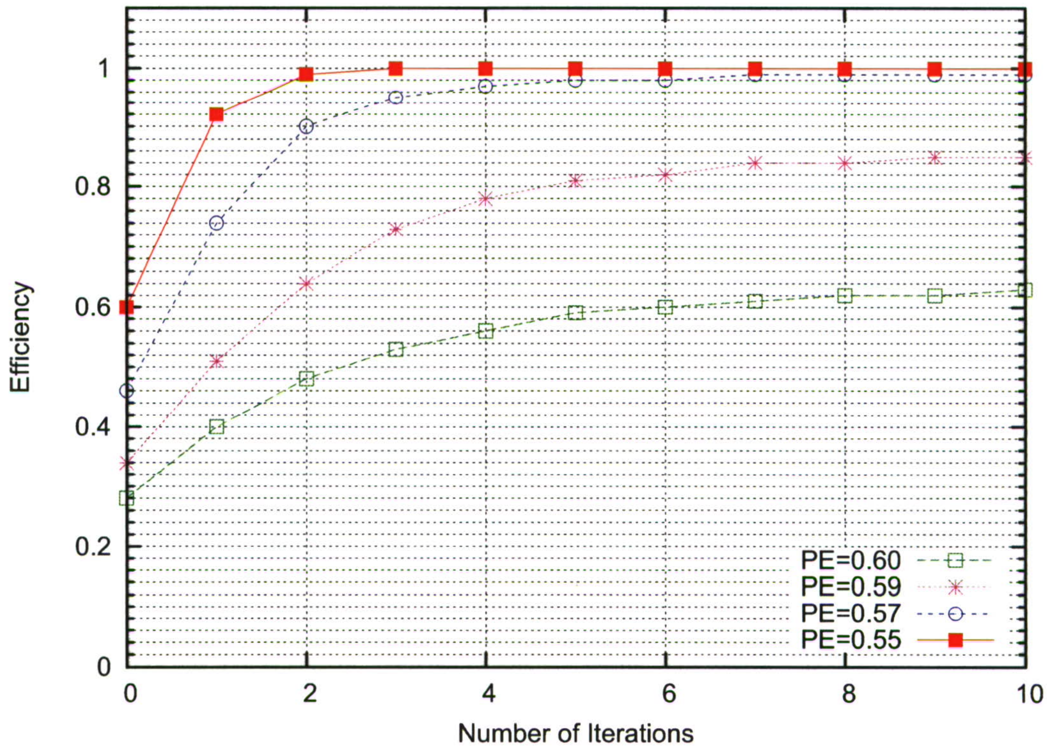


Figure 4.1: Relationship between Efficiency and Iterations for TGC (15/13), (1536, 512) with Turbo Decoder

4.3.3 Numerical Results

The results of TGC (15/13) in length of (3072, 1024) with DRP [29] interleaver decoded by standard turbo decoder at different iterations are shown in Figure 4.2. It clearly shows that how the channel capacity could be achieved by increasing

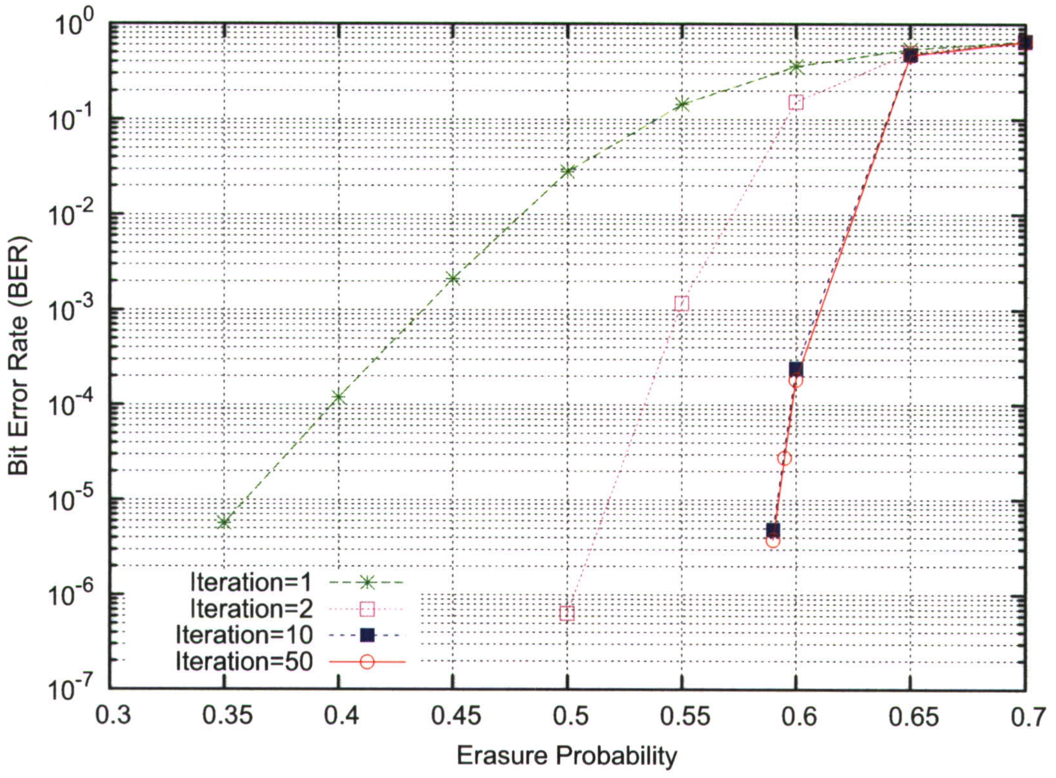


Figure 4.2: Turbo Decoding Results of TGC (15/13) with length (3072, 1024) at Different Iterations

the number of iterations of turbo decoder. Moreover, the required number of iterations could be reduced to a small value to still attain the similar results.

The results of TGC (15/13) with standard turbo decoder at lengths (600, 200) and (1536, 512) permuted by DRP [29] interleavers are shown in Figure 4.3. From the results, we can see that the result with longer codeword length (1536, 512) at same code-rate 1/3 and code structure (15/13) is able to achieve a significant improvement than result of codeword length (600, 200). And the error floor starts to occur at $\epsilon = 0.52$ for code (1536, 512) instead of $\epsilon = 0.55$ for code (600, 200).

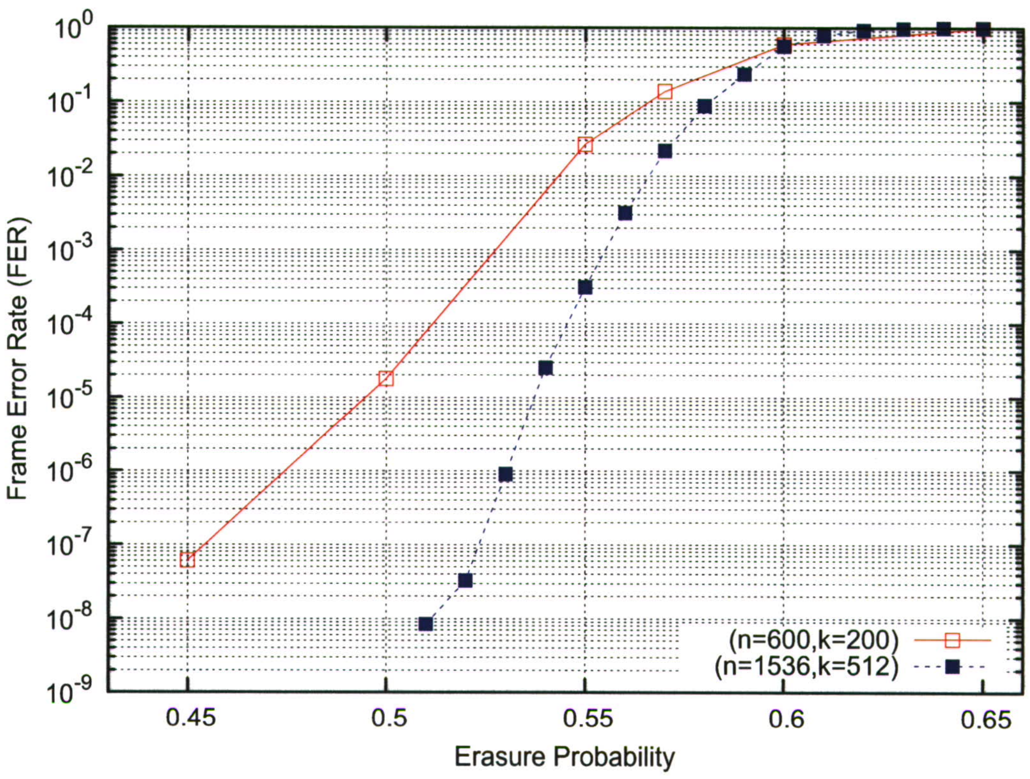


Figure 4.3: Results of TGC (15/13) with Turbo Decoder

4.4 Belief Propagation Decoding

In the erasure channel, the extrinsic probability may be represented by the value of the codeword coordinate c'_t. $c'_t = 1$ or 0 (its data value), if probability 1; and $c'_t = -1$, if probability $= 0.5$ (erased bit). Based on the decoding algorithm for BEC [75] and the simplicity of the computation of conditional probabilities of the erasure channel, the belief propagation decoding algorithm may be simplified as follows:

1. Initialise the extrinsic information for codeword **c'**.

2. Let **H** be the parity-check matrix, h_{ij} be the position on the matrix, $\{0 \leq i < m, 0 \leq j < n\}$, where i represents the row index, j is the column index, n is the codeword length and $m \geq n - k$ is the number of parity-check equations including $n - k$ independent parity-check equations and their

corresponding parity bits. Select the positions $h_{ij} = 1$, $\{i = 0, 0 \le j < n\}$ to construct a subset $\mathbf{v}_i$.

3. If there exists only one erasure in subset $\mathbf{v}_i$, the erasure $c_t^{\prime e} = (\sum_{j=0, j\neq t}^{|\mathbf{v}_i|} c_j^{\prime})$ mod 2, where $|\mathbf{v}_i|$ is the number of "1"s at row i. Then update the decoded information $c_t^{\prime d}$ as the extrinsic information for $c_t^{\prime}$. Otherwise, repeat step 2, to construct the next subset $\mathbf{v}_i, i = i + 1$.

4. Repeat the step 2 and 3 till $i = m - 1$, where one decoding iteration completes.

5. If there exists any erasure in the decoded sequence after the previous iteration, repeat the Step 2 to 4 until maximum number of iterations is reached.

4.4.1 Complexity Analysis

If at row i, there exist more than one erasure, then there is insufficient information to decode the intersected erasures. Thus these rows are not considered during the computational complexity excluding checking the subset. Hence, only if subset $\mathbf{v}_i$ contains one erasure, this computation is counted regarding the decoding complexity. Then the algorithm is classified as follows:

1. **Equation Check** (Proc. K), each equation consists of n CPs, there are $m = n - k$ equations in total, here $m = n - k$ is assumed to simplify the complexity computation.

2. **Decoding Erasure** (Proc. L), for equations containing only one erasure, each of these requires $n - 1$ MULs and $n - 2$ ADDs.

The required equation computation for BP decoding algorithm in terms of the number of equivalent additions is shown in Table 4.4. For 1/3 code-rate turbo codes or turbo Gallager codes, $n - k = 2k$. Thus, to decode one frame, the required number of operations N_o^{BP} in equivalent additions in terms of n_e with L iterations for BP algorithm is obtained as:

$$N_o^{BP}(n_e) = \sum_{i=1}^{L}(6k^2 + (n_e^{i-1} - n_e^{i})(6k - 3)) \tag{4.2}$$

Procedure	ADD	MUL	CP
Proc. K	–	–	$n-k$
Proc. L	$n-2$	$n-1$	–

Table 4.4: Computation Requirement of BP Decoding Algorithm

For the purpose, again we assume $n-k$ as the maximum number of erasures decoded by BP decoder after L iterations, thus there exist maximum $n-k$ (Proc. L) for decoding one block, then the required number of operations $N_o^{BP_s}$ in terms of k is obtained as:

$$N_o^{BP_s}(k) = 6k^2L + 12k^2 - 6k \tag{4.3}$$

4.4.2 Iteration Analysis

The similar statistics for iteration analysis is applied in BP decoding, Table 4.5 shows the required average maximum numbers of iterations for "Decoded" and "Undecoded" blocks. It may be considered as a function of the overall FER to

Erasure Probability	Avg. Iter. Decoded	Avg. Iter. Undecoded	Frame Error Rate
0.56	40	36	0.840336
0.55	46	21	0.584795
0.54	45	25	0.401606
0.53	49	24	0.171821
0.52	52	17	0.046816
0.51	38	25	0.009778
0.50	35	15	0.001701
0.49	34	21	0.000214

Table 4.5: Iteration Analysis of BP Decoder over Erasure Channel

indicate the decoding failure due to stopping sets, based on the required number of maximum iterations.

4.4.3 Numerical Results

The BP decoding performances for UMTS type of TGCs (15/13), (31/23), (61/43), (141/103), and the reference code (0, 3, 4/0, 14, 34) are shown in Figure 4.4. From

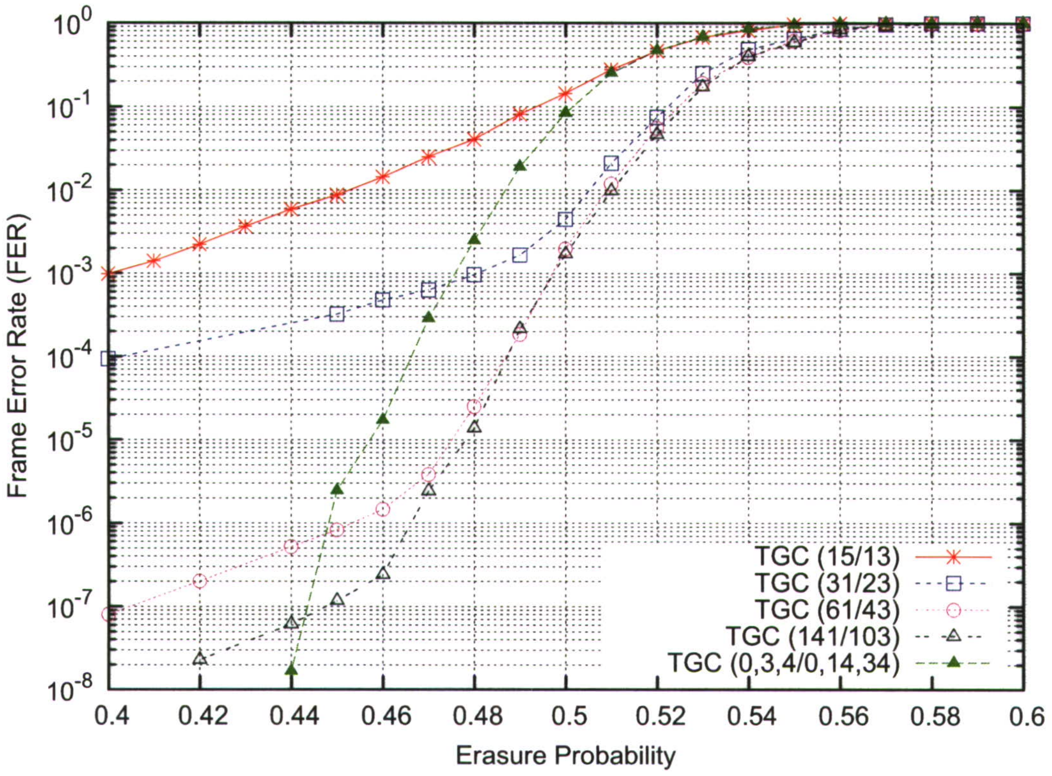


Figure 4.4: Results of TGCs (1536, 512) with BP Decoder in the Erasure Channel

the results, it may be seen that the performance is improved by increasing the code constraint length with the runs of "0" inside. When the longer code constraint length is selected, some of the stopping sets caused by the shorter constraint length code are broken and a lower error floor is realised. When the constraint length is chosen at a certain distance, there is only slight improvement in the waterfall region, like the results between TGC (61/43) and TGC (141/103) before $\epsilon = 0.47$. The result of code (0, 3, 4/0, 14, 34) does not achieve the best performance as it does in the AWGN channel as shown in Figure 3.5. At erasure probability $\epsilon = 0.46$, result of code (141/103) starts to reach the error floor region due to the stopping sets, however the result of code (0, 3, 4/0, 14, 34) still hasn't met the error floor, even at FER=10^{-8}. It also shows that by selecting a longer memory of the shift register on design of TGC is able to ensure the message-passing work well through the whole codeword in either the AWGN channel or the erasure channel, which could obviously lower the error floor.

4.5 Optimised Iterative Decoding

4.5.1 Remark on the BCJR Algorithm for the Erasure Channel

According to (2.18), we can see that the value of $\alpha_t(m)$ only depends on the value of $\gamma_t(m', m)$, if $\alpha_{t-1}(m')$ is known. Similarly, from (2.20), the value of $\beta_t(m)$ only depends on the value of $\gamma_{t+1}(m, m')$, if $\beta_{t+1}(m')$ is known. If we only consider the possible non-zero values during the computation, then we have some conditions should be satisfied:

- $\alpha_{t-1}(m')$ is non-zero value.
- $\beta_{t+1}(m')$ is non-zero value.
- $\gamma_t(m, m')$ is non-zero value.

In order to make sure $\gamma_t(m, m')$ is non-zero. We have some conditions to satisfy. First of all, the transition should be possible which means $P(S_t = m|S_{t-1} = m') = 1$. The second is that the probabilities of both data and parity bits on the transition should be non-zero, which means $P(x_t|S_t = m, S_{t-1} = m') > 0$ and $P(y_t|S_t = m, S_{t-1} = m') > 0$.

By analysing the state-transitions in the trellis based on different starting states, we find the following relationships. The transition and state relationship for $S_{Start} = 1$ is shown in Table 4.6, where S_{Start} is the number of start states, d represents the bit is known, e is the bit is erased. The transitions and states relationship for $S_{Start} = 2$ is shown in Table 4.7. The transitions and states relationship for $S_{Start} = 4$ is shown in Table 4.8. The transitions and states relationship for $S_{Start} = 8$ is shown in Table 4.9. According to these analysis and observations, we define $N_{start} = 2^i$ to represent the number of start states,

Data/ Parity	Number of End States	Number of Transitions	Number of Transition for each ending state
d/d	1→1	1	1
d/e	1→1	1	1
e/d	1→1	1	1
e/e	1→2	2	1

Table 4.6: Transitions and States Relationship for $S_{Start} = 1$

Data/ Parity	Number of End States	Number of Transitions	Number of Transition for each ending state
d/d	2→1, 2→2(4)	1,2(4)	1
d/e	2→2	2	1
e/d	2→2	2	1
e/e	2→2, 2→4(4)	4	2,1(4)

Table 4.7: Transitions and States Relationship for $S_{Start} = 2$

Data/ Parity	Number of End States	Number of Transitions	Number of Transition for each ending state
d/d	4→2, 4→4(8)	2,4(8)	1
d/e	4→4	4	1
e/d	4→4	4	1
e/d	4→4, 4→8(8)	8	2,1(8)

Table 4.8: Transitions and States Relationship for $S_{Start} = 4$

Data/ Parity	Number of End States	Number of Transitions	Number of Transition for each ending state
d/d	8→4, 8→8(16)	4,8(16)	1
d/e	8→8	8	1
e/d	8→8	8	1
e/e	8→8, 8→16(16)	16	2,1(16)

Table 4.9: Transitions and States Relationship for $S_{Start} = 8$

where $i \in \mathbb{N}$. Thus we conclude the relationship between states and transitions in Table 4.10. From Table 4.10, at conditions of d/d, d/e, e/d, there is only one

Data/ Parity	Number of End States	Number of Transitions	Number of Transition for each ending state
d/d	$2^i \rightarrow 2^{i-1}$, $2^n \rightarrow 2^i(2^{i+1})$	$2^{i-1}(2^{i+1})$	1
d/e	$2^i \rightarrow 2^i$	2^i	1
e/d	$2^i \rightarrow 2^i$	2^i	1
e/e	$2^i \rightarrow 2^i$, $2^i \rightarrow 2^{i+1}(2^{i+1})$	2^{i+1}	2,1(2^{i+1})

Table 4.10: Transitions and States Relationship for S_{Start}=2^i, $i \in \mathbb{N}$

possible transition available for each ending state. According to (2.18), we have

$$\begin{aligned} \alpha_t(m) &= \alpha_{t-1}(m')\gamma_t(m',m) \\ &= \alpha_{t-1}(m')P(x_t|S_t = m, S_{t-1} = m')P(y_t|S_t = m, S_{t-1} = m') \end{aligned} \tag{4.4}$$

Similarly, according to (2.20), we have

$$\beta_t(m) = \beta_{t+1}(m') \cdot P(x_t|S_t = m, S_{t+1} = m') \cdot P(y_t|S_t = m, S_{t+1} = m') \tag{4.5}$$

Since $\alpha_{t-1}(m')$ and $\beta_{t+1}(m')$ are known, thus can be considered as constant. For the transition probability of $P(x_t)$ and $P(y_t)$, we have the possible product values shown in Table 4.11. Since the transition probabilities only take the values of "0"

Data/Parity	Data (x_t)	Parity (y_t)	Transition Probability
d/d	1,0	1,0	1,0
d/e	1,0	0.5,0	0.5,0
e/d	0.5,0	1,0	0.5,0
e/e	0.5,0	0.5,0	0.25,0

Table 4.11: Possible Product of Transition Probabilities

or the same non-zero value which is from the set of "$\{1, 0.5, 0.25\}$". According to (2.18) and (2.20), $\alpha_t(m)$ or $\beta_t(m)$ if non zero, will be computed with the same value. Hence during the α and β recursions, the non-zero values of α or β are equal at any given trellis section. At condition of e/e, there are two possible transitions for each ending state. Since the transition probability is either 0 or 0.25 from Table 4.11, and $m' = 2$, then we have

$$\alpha_t(m) = 2 \times \alpha_{t-1}(m') \times 0.25 \tag{4.6}$$

Then it has been proved that non zero values of $\alpha_t(m)$ or $\beta_t(m)$ at any condition, are identical at any given trellis section.

4.5.2 Remark on the Extrinsic Information

Iterative decoding is based on the additional knowledge of previous information knowledge to obtain better reliability. According to [10], the extrinsic information $L_e(\hat{d})$ is defined as:

$$L_e(\hat{d}) = \log \sum_m \alpha_t(m) \cdot \beta_t(m) \tag{4.7}$$

In Figure 4.5, it depicts an example of trellis with received constituent code $R_t = \{0.5, 0\}$. The highlighted black dots represent the linked states with non-zero value. The extrinsic information is calculated as:

$$\begin{aligned} Ex_0 &= \sum \alpha_t^0(m) \cdot \beta_t^0(m) = 2 \times 0.125 \times 0.5 \\ Ex_1 &= \sum \alpha_t^1(m) \cdot \beta_t^1(m) = 2 \times 0.125 \times 0.5 \end{aligned}$$

After normalisation, $Ex_0 = Ex_1 = 0.5$. Thus the extrinsic information is not helpful for the constituent code in the next decoder. But from the trellis, it is clear to identify the data bit can be decoded as "1". Thus we have in the erasure channel, once the extrinsic information for the erasure has been determined, the extrinsic information does not change any more during the further iterations. In order to provide more accurate and reliable knowledge during the iterative decoding process, the full information exchange might be realised instead. Then

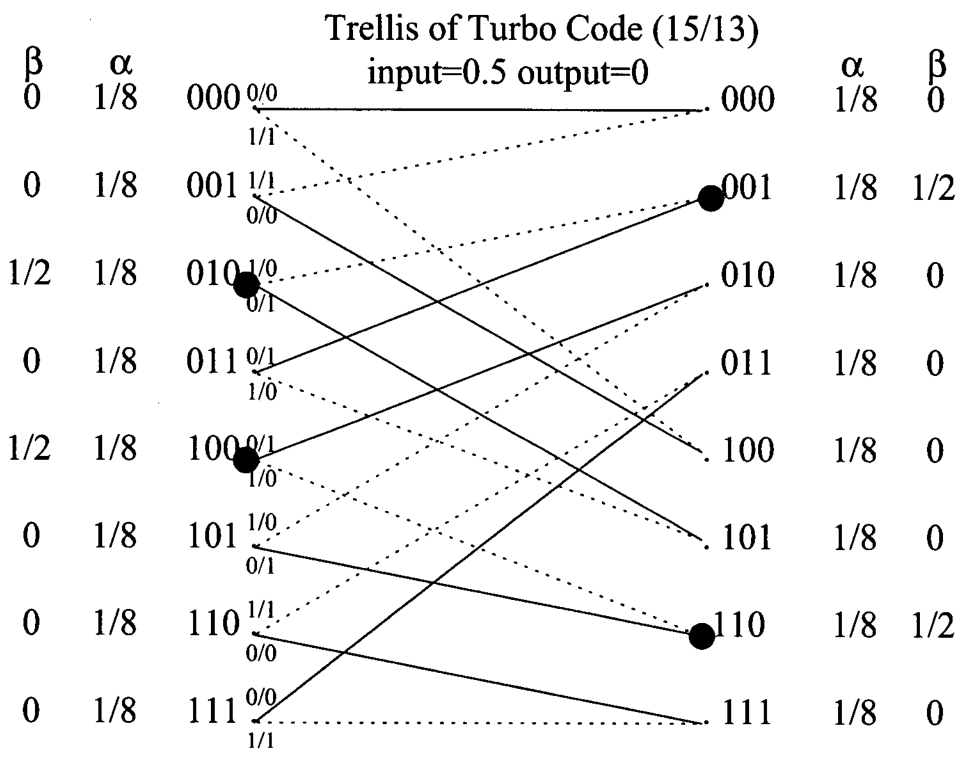


Figure 4.5: Sample Trellis for Turbo Code (15/13)

we have

$$L_e(\hat{d}) = \log \sum_m \alpha_t(m) \cdot \gamma_t(m) \cdot \beta_t(m) \tag{4.8}$$

4.5.3 Look-Up Table Decoder

The first look-up table based approach for decoding based on the syndrome trellis was introduced by Schalkwijk & Vinck [106] for convolutional codes over the binary symmetric channel. A fast LUT decoding algorithm working on the decoding trellis was introduced by Kurkoski *et al.* [66] for convolutional codes over the erasure channel. The fast LUT decoding algorithm uses three LUTs to represent the forward recursion α_i, backward recursion β_i and output probability function. It is a simplification in complexity compared to the standard BCJR decoder. However the LUT decoding algorithm might be further simplified. It is shown in Section 4.5.1 that the non-zero values of α or β are always identical.

Consequently the values of α_i and β_i may be represented by a single bit with a "1" representing a non-zero value state and a "0" representing a zero value state. Thus two binary vectors can be used to represent the values of α and β at each trellis section and a more efficient decoding arrangement may be realised by constructing a *look-up table*, which includes two vectors of binary numbers to represent the final transitions, and one vector to represent the conditions of received bits. In the decoding process, the trellis section containing erased information is converted to obtain the final transition binary numbers. By looking up the table, it is possible to directly obtain the corrected information bit, or decoding failure. For instance, assuming a trellis as shown in Figure 4.5, it depicts an example of trellis. The received probabilities of information bit and parity bit are 0.5 and 0. The highlighted black dots represent the linked states with non-zero value, which are denoted as "1", the rest of states with zero values are denoted as "0". Thus we have the starting binary number 00101000 and end binary number 01000010, in decimal, $40-66$. By looking up the table, with "$40-66$", and information bit is unknown, parity is known as 0, we can easily find the erasure of information as value of "1".

4.5.4 Iteration Analysis

Table 4.12, as a function of the erasure probability ϵ, shows the average maximum number of iterations for "Decoded" blocks and the average maximum number of iterations needed for "Undecoded" blocks to indicate the decoding failure due to the stopping sets at each corresponding Frame Error Rate (FER).

Probability Erasure	Avg. Iter. for Decoded	Avg. Iter. for Undecoded	Frame Error Rate
0.62	9	3	0.884955
0.61	9	4	0.746268
0.60	10	5	0.448430
0.59	10	5	0.220750
0.58	10	5	0.069156
0.57	10	5	0.014423
0.56	11	6	0.002180

Table 4.12: Iterations Analysis for the Optimised LUT Decoder

By analysing the decoding efficiency at each iteration, we have Figure 4.6 showing the histogram between decoding efficiency and number of iterations during the iterative decoding process. From the figure, we can see that at

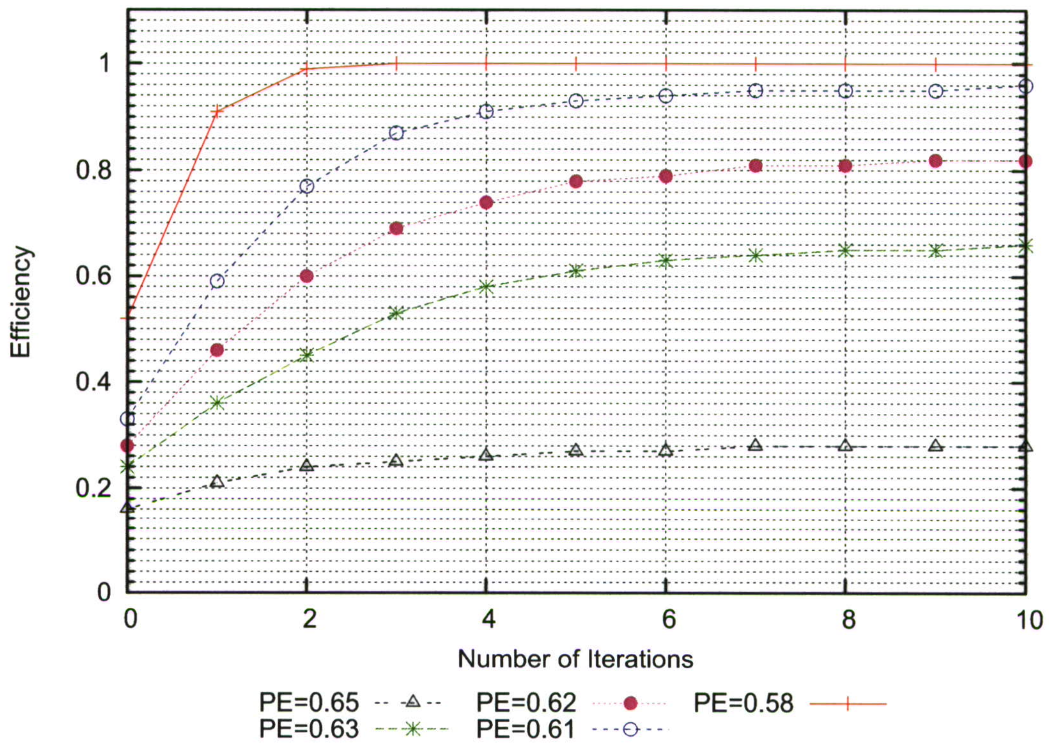


Figure 4.6: Relationship between Efficiency and Iterations for TGC (15/13), (1536, 512) with Optimised Iterative LUT Decoder

iteration = 10, LUT decoding has already achieved the best performance. At *iteration* = 5, there is a slight difference comparing to the further iterations. Thus maximum number of iterations is ideal to choose.

The relationship between the average iterations and erasure probability for "Decoded" and "Undecoded" blocks is shown in Figure 4.7. From the results, it shows the required average maximum number of iterations for "Decoded" is reduced with the decreased erasure probability, whereas the average maximum number of iterations for "Undecoded" is increased. However the maximum iterations for all blocks can still be determined under 10 iterations.

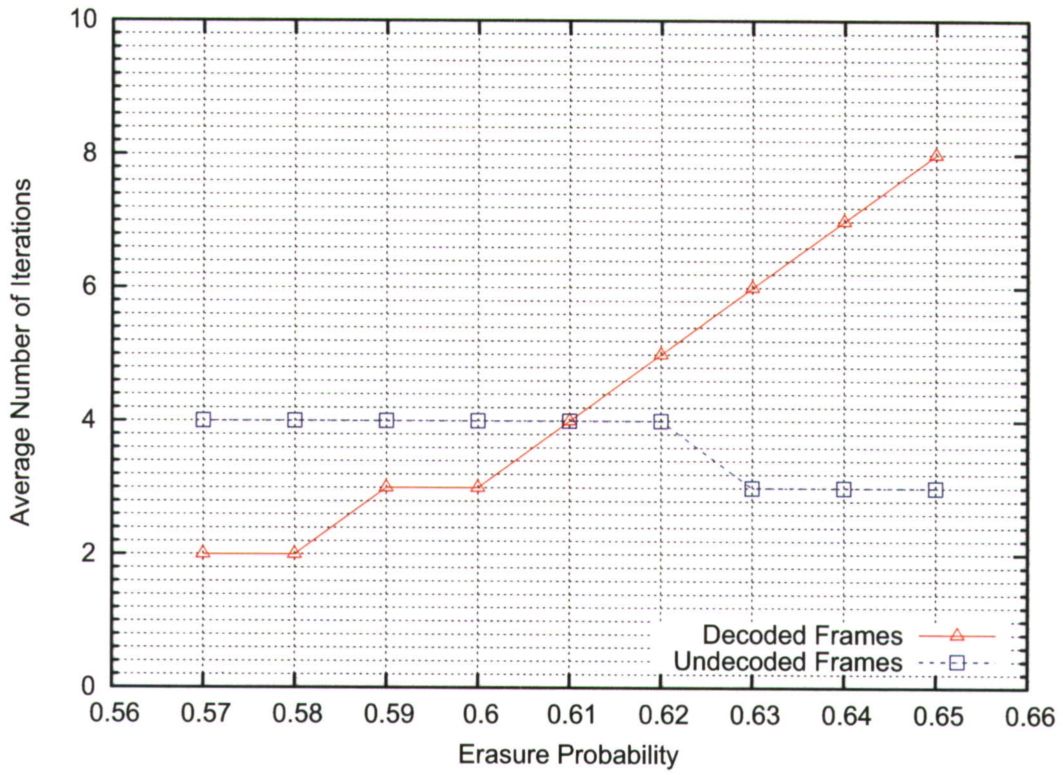


Figure 4.7: Relationship Between Probability Erasure and Iterations for TGC (15/13), (1536,512) with Optimised Iterative LUT Decoder

4.5.5 Complexity Analysis

For the optimised iterative LUT decoder, the look-up table is constructed once only. The decoding procedures including look-up table construction are classified as follows:

1. **Look-Up Table Construction** (Proc. F), for each trellis section, there are three possible conditions to be considered. The condition includes *a*) both data and parity bits are erased; *b*) data is erased and parity is not erased, parity is either 0 or 1. Each branch requires 2 CPs for its validation, there are 2^{M+1} branches in total. One trellis section includes 2^{M+1} CPs to decide the decoding success or failure. There are $2 \cdot \sum_{i=0}^{M} \binom{2^M}{2^i}$ possible trellis sections.

2. α **Metric** (Proc. G), each state requires 2 CPs and 1 ASSI for its validation, there are 2^M states in total.

3. β **Metric** (Proc. H), its computation is as same as the Proc. G.

4. **Valid Trellis Construction** (Proc. I), it consists of 1 ADD for each state.

5. **Hard Decision** (Proc. J), it requires $2^{M+1} - 2$ ADDs and 2^{M+1} MULs for getting starting states and ending states; 2 CPs and 1 ASSI for getting decoding options, and 1 LKUP and 1 ASSI to assign the decoded bit.

The required computations for the optimised LUT decoding algorithm in terms of the number of equivalent additions are shown in Table 4.13. Thus

Procedure	ADD	MUL	CP	ASSI	LKUP
Proc. F	–	–	$9 \times 2^{M+1}$	–	–
Proc. G	–	–	2^{M+1}	2^M	–
Proc. H	–	–	2^{M+1}	2^M	–
Proc. I	2^M	–	–	–	–
Proc. J	$2^{M+1} - 2$	2^{M+1}	2	2	1

Table 4.13: Number of Computations for Optimised LUT Decoding Algorithm

in a 1/3 code-rate turbo decoding scheme, decoding one frame requires k trellis sections, 2 corresponding LUT decoders and n_e^i decoding operations at each iteration. The required number of operations N_o^{LUT} in equivalent additions in terms of n_e with L iterations for the optimised LUT decoding algorithm is obtained as:

$$N_o^{LUT}(n_e) = \sum_{i=1}^{L}(12k2^M + (2^{M+2} + 5)n_e^i) \tag{4.9}$$

For comparison, we assume k as the maximum number of data erasures decoded by LUT decoder after L iterations. Thus there exist maximum k (Proc. J) for decoding one block, then the required number of operations $N_o^{LUT_s}$ in terms of k is obtained as:

$$N_o^{LUT_s}(k) = 12k2^M L + (2^{M+2} + 5)k \tag{4.10}$$

The complexity comparison between different iterative decoders for decoding one block, in size of ($n = 1536, k = 512$), is shown in Figure 4.8. From the

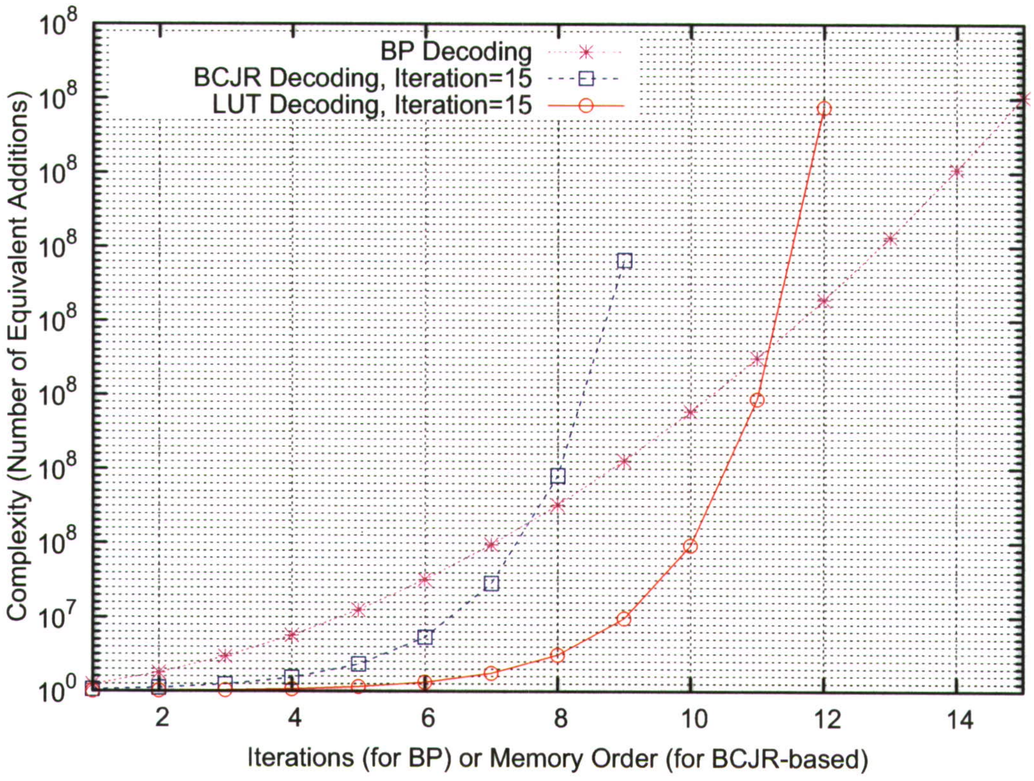


Figure 4.8: Complexity Comparison between BCJR-based Decoding and BP Decoding

figure, it is clear that the complexity of the BCJR-based decoding algorithms is exponential as a function of memory order, meanwhile, the complexity of BP decoding is only increased by the number of iterations. Since the turbo decoder and the LUT decoder both use the BCJR algorithm, their performance will be identical. By ignoring the look-up table construction computation, the LUT algorithm provides reduced complexity compared to the standard turbo decoding algorithm.

4.5.6 Numerical Results

The BCJR-based decoding performances for TGC (15/13) and TGC (141/103) permuted by DRP interleaver [29] are shown in Figure 4.9. According to Fig-

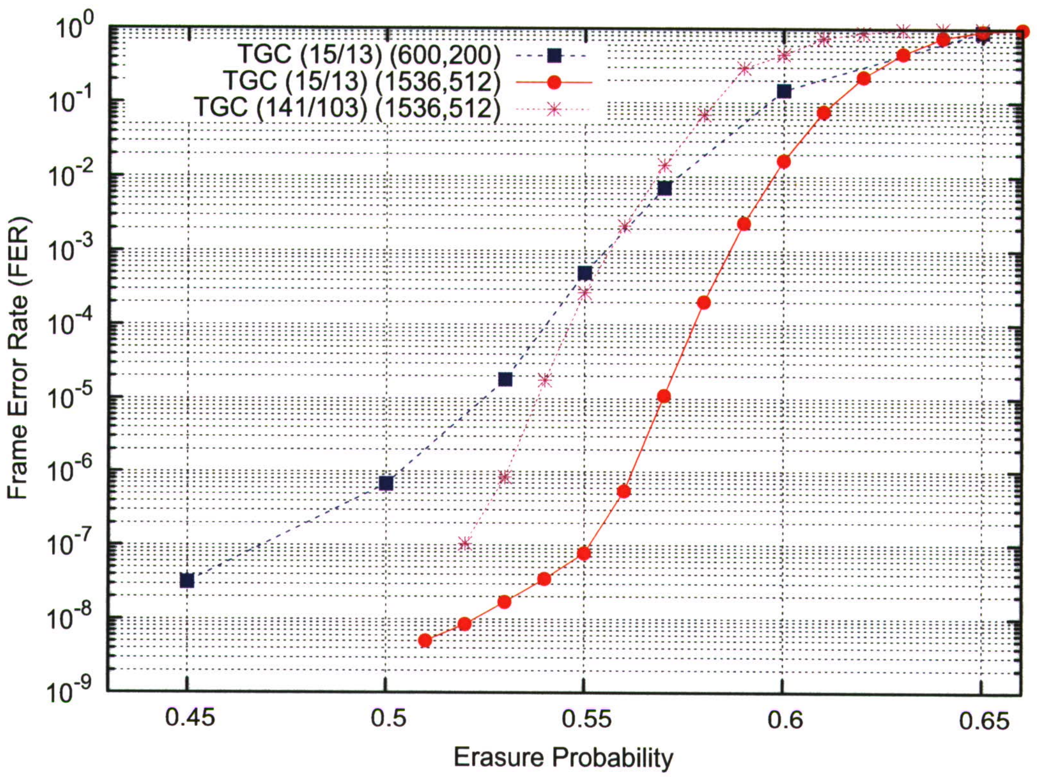


Figure 4.9: LUT Decoding Results of TGC (15/13) and (141/103)

ure 3.3, result of TGC (141/103) achieves very similar performance as TGC (15/13) in the AWGN channel. However in the erasure channel, TGC (141/103) does not perform similar result as TGC (15/13). We can see that there should exist the convergence issue between turbo Gallager codes and optimised iterative decoder due to either the code design or the stopping sets. Furthermore, TGC (15/13) with optimised iterative LUT decoder does not only achieve the best performance, and also implement the algorithm at least computational complexity due to its small memory order.

4.6 Summary

According to the analysis on the BCJR algorithm for the erasure channel, the look-up table based BCJR decoding algorithm has been introduced. In terms of the equivalent additions, the standard BCJR decoding algorithm, the optimised

LUT decoding algorithm and the BP decoding algorithm over the erasure channel have been analysed in computational complexity and derived in equations. The simulation results by adopting different iterative decoding algorithms for turbo Gallager codes have been presented over the erasure channel.

Chapter 5

DVB-RCS Turbo Codes

5.1 Introduction

According to the latest digital broadcasting standard [59], DVB-RCS turbo codes $(11, 13/15)$ are adopted in the DVB-RCS system. The selected DVB-RCS turbo codes are optimised to support a range of frame sizes from 12 bytes to 216 bytes. A series of code rates are supported ranging from $r_c = 1/3$ to $r_c = 6/7$.

In order to ensure a best iterative BCJR decoding performance, the *circular recursive systematic convolutional* (CRSC) codes were proposed by Berrou & M.Jezequel [9], Berrou *et al.* [11]. CRSC codes are able to perform at the similar concept of *tail-biting* by Ma & Wolf [78], where the ending state matches the starting state during the decoding trellis to avoid the terminated bits' losses. The main advantage of using DVB-RCS turbo codes is that the trellis contains half as many states as a binary code of identical constraint length (but the same number of edges) and therefore only needs half as much memory. The detailed benefits of using DVB-RCS turbo codes are described in [18]. In this chapter, the erasure decoding performance for DVB-RCS turbo codes is evaluated in symbol-based interleaver and bit-based interleaver. The corresponding parity-check matrix of DVB-RCS turbo codes is discussed and derived in Section 5.2. The observation of utilising bit-interleaver is realised and analysed in Section 5.3. Section 5.3.4 presents the numerical results by using different interleavers in symbol-based and bit-based. According to the observation, an extended probabilistic guessing algorithm is introduced in Section 5.4. Improved results of using the guessing algorithm are presented in Section 5.4.1.

5.2 The Parity-Check Matrix for DVB-RCS Turbo Codes

As the standard defines, the encoding structure of DVB-RCS turbo codes $(11, 13/15)$ is depicted in Figure 5.1, where A, B are the input information sequences, I_1, I_2 and I_3 are the input to each delay memory of the shift-register. And D is the delay memory storage of the register, W and Y are the output sequences corresponding to information A, B. For permuted information sequences A', B' generated by the interleaver factor of π, we have the corresponding output sequences W' and Y'.

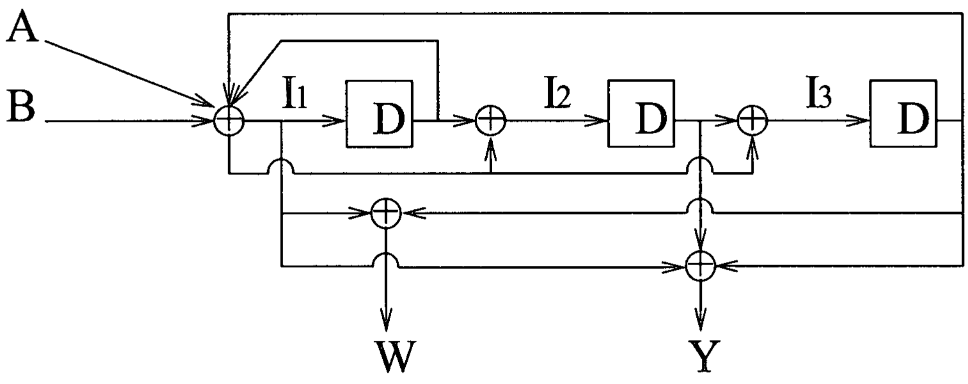


Figure 5.1: Encoder Scheme for DVB-RCS Turbo Codes

According to the above figure, values of I_1, I_2 and I_3 could be represented in terms of A, B, D and I_1. The relationships are concluded as follows:

$$I_1 = A + B + I_1D + I_3D \tag{5.1}$$

$$I_2 = I_1D + B \tag{5.2}$$

$$I_3 = I_2D + B = (I_1D + B)D + B = I_1D^2 + BD + B \tag{5.3}$$

By putting (5.3) to replace I_3, (5.1) is extended as:

$$\begin{aligned} I_1 &= A+B+I_1D+I_1D^3+BD^2+BD \\ I_1(1+D+D^3) &= A+B(1+D+D^2) \\ I_1 &= \frac{A+B(1+D+D^2)}{1+D+D^3} \end{aligned} \tag{5.4}$$

For output W, we have

$$\begin{aligned} W &= I_1+I_3D \\ &= A+B+I_1D+I_3D+I_3D \\ &= A+B+I_1D \\ &= A+B+\frac{D(A+B(1+D+D^2))}{1+D+D^3} \\ &= \frac{(1+D^3)A+(1+D^2)B}{1+D+D^3} \end{aligned}$$

Then we have the relationship in a parity-check equation,

$$(1+D+D^3)W+(1+D^3)A+(1+D^2)B=0 \tag{5.5}$$

According to (5.5), we have the polynomials for A, B and W as shown in Table 5.1. Since output Y is independent of output W, in order to construct the parity-check

A	B	W
$1+D^3$	$1+D^2$	$1+D+D^3$
1001	1010	1101

Table 5.1: Polynomials for the Parity-Check Equation of W

matrix, the first part of the matrix representing the relationship between input A, B and output W based on (5.5) and Table 5.1 is shown in Table 5.2, where N is the length of the DVB-RCS information bits, L_r represents the row length of each column section, L_c is the column length.

A, $L_r = N$	B, $L_r = N$	W, $L_r = N$	Y, $L_r = N$	W', $L_r = N$	Y', $L_r = N$
1001...	1010...	1101...	All	All	All
1001...	1010...	1101...	Zeros	Zeros	Zeros
$\vdots$	$\vdots$	$\vdots$	"0"	"0"	"0"

Table 5.2: First Part of the Parity-Check Matrix for Output W, $L_c = N$

For output Y, we have

$$\begin{aligned}
Y &= I_1 + I_2 D + I_3 D \\
&= I_1 + (I_1 + B)D + (I_1 D^2 + BD + B)D \\
&= I_1 + I_1 D^2 + BD + I_1 D^3 + BD^2 + BD \\
&= I_1(1 + D^2 + D^3) + BD^2 \\
&= \frac{[A + B(1 + D + D^3)](1 + D^2 + D^3) + BD^2(1 + D + D^3)}{1 + D + D^3} \\
&= \frac{(1 + D^2 + D^3)A + (1 + D + D^2 + D^3)B}{1 + D + D^3}
\end{aligned}$$

Then we have the parity-check equation for Y

$$(1 + D^2 + D^3)A + (1 + D + D^2 + D^3)B + (1 + D + D^3)Y = 0 \qquad (5.6)$$

According to (5.6), we have the polynomials for A, B and Y as shown in Table 5.3. Consequently, we can construct the second part of the parity-check matrix for output Y based on (5.6) and Table 5.3 as shown in Table 5.4.

A	B	Y
$1 + D^2 + D^3$	$1 + D + D^2 + D^3$	$1 + D + D^3$
1011	1111	1101

Table 5.3: Polynomials for the Parity-Check Equation of Y

Since output W', which corresponds to the permuted information sequence, only depends on the permuted input A' and B'. Then we have the third part of the parity-check matrix as shown in Table 5.5, where π represents the interleaver

A, $L_r = N$	B, $L_r = N$	W, $L_r = N$	Y, $L_r = N$	W', $L_r = N$	Y', $L_r = N$
1011... 1011... ⋮	1111... 1111... ⋮	All Zeros "0"	1101... 1101... ⋮	All Zeros "0"	All Zeros "0"

Table 5.4: Second Part of the Parity-Check Matrix for Output Y, $L_c = N$

function. Respectively, the fourth part of the parity-check matrix for output Y'

A, $L_r = N$	B, $L_r = N$	W, $L_r = N$	Y, $L_r = N$	W', $L_r = N$	Y', $L_r = N$
Permuted $A' = \pi(A)$ ⋮	Permuted $B' = \pi(B)$ ⋮	All Zeros "0"	All Zeros "0"	1101... 1101... ⋮	All Zeros "0"

Table 5.5: Third Part of the Parity-Check Matrix for Output W', $L_c = N$

is shown in Table 5.6. Hence, we have the constructed parity-check matrix of

A, $L_r = N$	B, $L_r = N$	W, $L_r = N$	Y, $L_r = N$	W', $L_r = N$	Y', $L_r = N$
Permuted $A' = \pi(A)$ ⋮	Permuted $B' = \pi(B)$ ⋮	All Zeros "0"	All Zeros "0"	All Zeros "0"	1101... 1101... ⋮

Table 5.6: Fourth Part of the Parity-Check Matrix for Output Y', $L_c = N$

DVB-RCS turbo codes $(11, 13/15)$ shown in Table 5.7.

The relationship between input A, B and output W, Y during the shift register is shown in Table 5.8, where S represents the state, $S \in \{0, 1, ..., (2^M - 1) = 7\}$ $M = 3$. And for each duo binary input (A, B), (S_1, S_2, S_3) represents the current state, and (X, S_1, S_2) indicates the next state.

The trellis diagram of DVB-RCS turbo codes $(11, 13/15)$ is shown in Figure 5.2. It clearly shows that in the same number of distinct state transitions, DVB-RCS turbo code only requires half number of states with 2^M to acquire 2^{M+2} number of state transitions.

A, $L_r = N$	B, $L_r = N$	W, $L_r = N$	Y, $L_r = N$	W', $L_r = N$	Y', $L_r = N$
1001... 1001... $\vdots$	1010... 1010... $\vdots$	1101... 1101... $\vdots$	All Zeros "0"	All Zeros "0"	All Zeros "0"
1011... 1011... $\vdots$	1111... 1111... $\vdots$	All Zeros "0"	1101... 1101... $\vdots$	All Zeros "0"	All Zeros "0"
Permuted $A' = \pi(A)$ $\vdots$	Permuted $B' = \pi(B)$ $\vdots$	All Zeros "0"	All Zeros "0"	1101... 1101... $\vdots$	All Zeros "0"
Permuted $A' = \pi(A)$ $\vdots$	Permuted $B' = \pi(B)$ $\vdots$	All Zeros "0"	All Zeros "0"	All Zeros "0"	1101... 1101... $\vdots$

Table 5.7: The Parity-Check Matrix ($4N \times 6N$) of DVB-RCS Turbo Codes

5.3 Interleaver Design for DVB-RCS Turbo Codes

5.3.1 Symbol-based Interleaver

According to the DVB-RCS standard, the symbol-based interleaver is designed for input sequence in pair, it normally includes two levels of interleaver.

level 1: the permutation is performed inside the pair of bits. Let j be the index of information bit, $j \in \{0, ..., N-1\}$, where N is the length of information bits in pair.

$$\text{if } j \mod 2 = 0, \text{ let } (A, B) = (B, A) \text{ by inverting the pair} \tag{5.7}$$

level 2

- if $j \mod 4 = 0$, then $P = 0$
- if $j \mod 4 = 1$, then $P = N/2 + P_1$
- if $j \mod 4 = 2$, then $P = P_2$
- if $j \mod 4 = 3$, then $P = N/2 + P_3$

	A	B	X	S_1	S_2	S_3	W	Y		A	B	X	S_1	S_2	S_3	W	Y
$S=0$	0	0		0	0	0			$S=4$	0	0		1	0	0		
			0	0	0		0	0				1	1	0		1	1
	0	1		0	0	0				0	1		1	0	0		
000			1	1	1		1	1	100			0	0	1		0	0
	1	0		0	0	0				1	0		1	0	0		
			1	0	0		1	1				0	1	0		0	0
	1	1		0	0	0				1	1		1	0	0		
			0	1	1		0	0				1	0	1		1	1
$S=1$	0	0		0	0	1			$S=5$	0	0		1	0	1		
			1	0	0		0	0				0	1	0		1	1
	0	1		0	0	1				0	1		1	0	1		
001			0	1	1		1	1	101			1	0	1		0	0
	1	0		0	0	1				1	0		1	0	1		
			0	0	0		1	1				1	1	0		0	0
	1	1		0	0	1				1	1		1	0	1		
			1	1	1		0	0				0	0	1		1	1
$S=2$	0	0		0	1	0			$S=6$	0	0		1	1	0		
			0	0	1		0	1				1	1	1		1	0
	0	1		0	1	0				0	1		1	1	0		
010			1	1	0		1	0	110			0	0	0		0	1
	1	0		0	1	0				1	0		1	1	0		
			1	0	1		1	0				0	1	1		0	1
	1	1		0	1	0				1	1		1	1	0		
			0	1	0		0	1				1	0	0		1	0
$S=3$	0	0		0	1	1			$S=7$	0	0		1	1	1		
			1	0	1		0	1				0	1	1		1	0
	0	1		0	1	1				0	1		1	1	1		
011			0	1	0		1	0	111			1	0	0		0	1
	1	0		0	1	1				1	0		1	1	1		
			0	0	1		1	0				1	1	1		0	1
	1	1		0	1	1				1	1		1	1	1		
			1	1	0		0	1				0	0	0		1	0

Table 5.8: Encoding Function of DVB-RCS Turbo Codes (11, 13/15)

Then we have the interleaver index i

$$i = (P_0 \times j + P + 1) \mod N \qquad (5.8)$$

The pre-defined interleaver parameters [59] for each information frame size in terms of P_q, where $q \in \{0, 1, 2, 3\}$, are shown in Table 5.9.

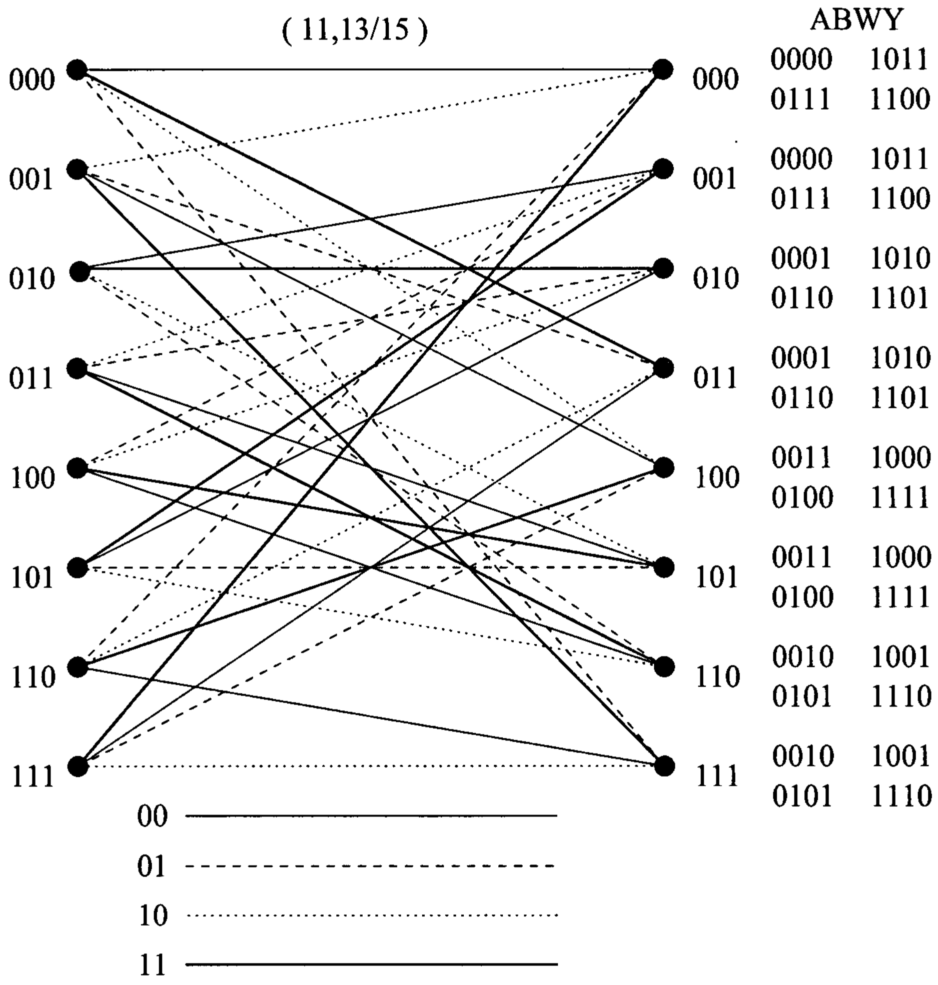


Figure 5.2: Trellis Diagram of DVB-RCS Turbo Codes (11, 13/15)

5.3.2 Bit-based Interleaver

For bit-based interleaver, it only includes one level permutation, which permutes the k information bits into different positions, where ($k = 2 \times N$) is the length of information bits. The input sequence is arranged in form of $\{(a_0, b_0), (a_1, b_1), ...,$ $(a_{N-1}, b_{N-1})\}$, and bit-based interleaver is functional to permute every bit into a new position, then we have the permuted input sequences $\{(a'_0, b'_0), (a'_1, b'_1), ...,$ $(a'_{N-1}, b'_{N-1})\}$.

Frame size in bit	P_o	$\{P_1, P_2, P_3\}$
$N = 96$	11	$\{24, 0, 24\}$
$N = 128$	7	$\{34, 32, 2\}$
$N = 424$	13	$\{106, 108, 2\}$
$N = 440$	23	$\{112, 4, 116\}$
$N = 456$	17	$\{116, 72, 188\}$
$N = 848$	11	$\{6, 8, 2\}$
$N = 864$	13	$\{0, 4, 8\}$
$N = 880$	13	$\{10, 4, 2\}$
$N = 1504$	19	$\{376, 224, 600\}$
$N = 1696$	19	$\{2, 16, 6\}$
$N = 1712$	19	$\{428, 224, 652\}$
$N = 1728$	19	$\{2, 16, 6\}$

Table 5.9: Symbol-based DVB-RCS Turbo Codes' Permutation Parameters

5.3.3 Remark on Bit-based Interleaver

For DVB-RCS turbo codes, the iterative decoder operates by passing two received data bits and two received parity bits to the code trellis to compute the conditional probability $Pr(d_t = l|(A = a_t, B = b_t))$ and extrinsic information $Ex(d_t = l|(A = a_t, B = b_t))$, $l \in \{00, 01, 10, 11\}$. In symbol-based interleaver, the extrinsic information can be directly used for the next decoder, since in any pair of data bits, a_t and b_t are always together after two level permutations. But in bit-based interleaver, a_t and b_t can not be guaranteed that they will be in a pair after permutation. Thus the extrinsic information in condition of $A = a_t$ and $B = b_t$ can not be directly used for the next decoder, since $A = a_i$ and $B = b_i$. In order to pass the most reliable extrinsic information to the next decoder, we have to break the extrinsic information in pair into extrinsic information in bit. During the transform, some of the extrinsic information are erased and can not provide help for the other decoder. For instance, there are a series of extrinsic information in pair for (a_t, b_t) as shown in Table 5.10. In the table, for the first four sets of extrinsic values, the split extrinsic information for bits are still helpful to identify one correct data bit between a_t and b_t. But for the last two sets of extrinsic information, since the non-zero extrinsic values are opposite to each other, for example, when $Ex(d_t = 00) = 0.5$, $Ex(d_t = 11) = 0.5$ and when $Ex(d_t = 10) = 0.5$, $Ex(d_t = 01) = 0.5$. For getting the bit extrinsic information, only 0.5 can be assigned to data bits a_t and b_t, which means both of those bits are unknown. Hence it is not possible to provide any helpful information knowledge for the other decoder.

$Ex(d_t = 00)$	$Ex(d_t = 01)$	$Ex(d_t = 10)$	$Ex(d_t = 11)$
0.5	0.5	0	0
0.5	0	0.5	0
0	0.5	0	0.5
0	0	0.5	0.5
0.5	0	0	0.5
0.0	0.5	0.5	0

Table 5.10: Sample of Extrinsic Information for $Ex(d_t = l|(A = a_t, B = b_t))$

In symbol-based interleaver, since a_t and b_t are always together in one pair, the extrinsic information as shown in Table 5.10 still can be directly passed to the next decoder, which might help provide useful information knowledge during the decoding process. Regarding the discussed observation, we implement the simulation on same codeword with same interleaver in symbol-based and bit-based decoded by iterative decoder at same number of iterations. As the numerical results shown in Figure 5.3, at any erasure probability ϵ, there exists a performance gap the same codes between bit-based interleaver and symbol-based interleaver, which is an obvious evidence that the bit-based interleaver is affected by the loss of extrinsic information exchange in pair than symbol-based interleaver. Meanwhile, the result of symbol-based interleaver is also suffered by the stopping sets, as compared to the decoding performance of turbo codes (15/13) in the similar code rate and codeword length. Then our research analysis is emphasised on if a well designed symbol-based interleaver with larger hamming distance could break the stopping sets as shown in Figure 5.3; or if a bit-based interleaver is able to achieve an improved performance than the symbol-based interleaver. Thus the well designed DRP interleaver and S-random interleaver are applied on the DVB-RCS turbo codes design.

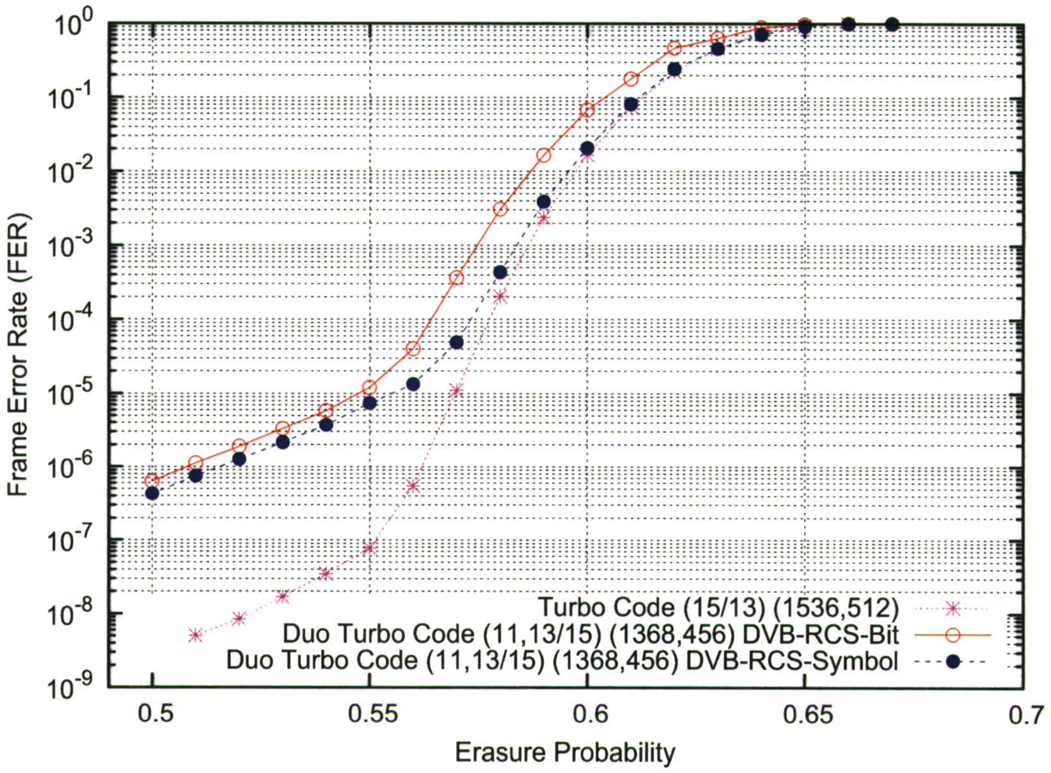


Figure 5.3: Results of DVB-RCS Turbo Codes by BCJR Decoder

5.3.4 Numerical Results

The BCJR iterative decoding results for DVB-RCS turbo codes in size of ($n = 1368$, $k = 456$) with 50 iterations over the erasure channel are shown in Figure 5.4. At range of $\epsilon = 0.56$ to $\epsilon = 0.66$, DVB-RCS turbo codes with symbol-based interleaver achieve the best performance, but the error floor starts from $\epsilon = 0.55$ due to the stopping sets. The DRP-B interleaver is based on the DRP symbol-based interleaver by doubling the index number of each symbol in terms of N to construct the DRP bit-based interleaver. It is noted that such interleaver has the same feature as DVB-RCS symbol-based interleaver has, that each paired bits stay together after permutation. The other two bit-based interleavers are S-random bit-based interleaver with $d_{min} = 32$, and DRP-A bit-based interleaver with $d_{min} = 38$. The curves clearly show that both codes with bit-based interleavers successfully lower the error floors. But at the same range of $\epsilon = 0.56$ to $\epsilon = 0.66$, their performances are still affected by the stopping sets, which

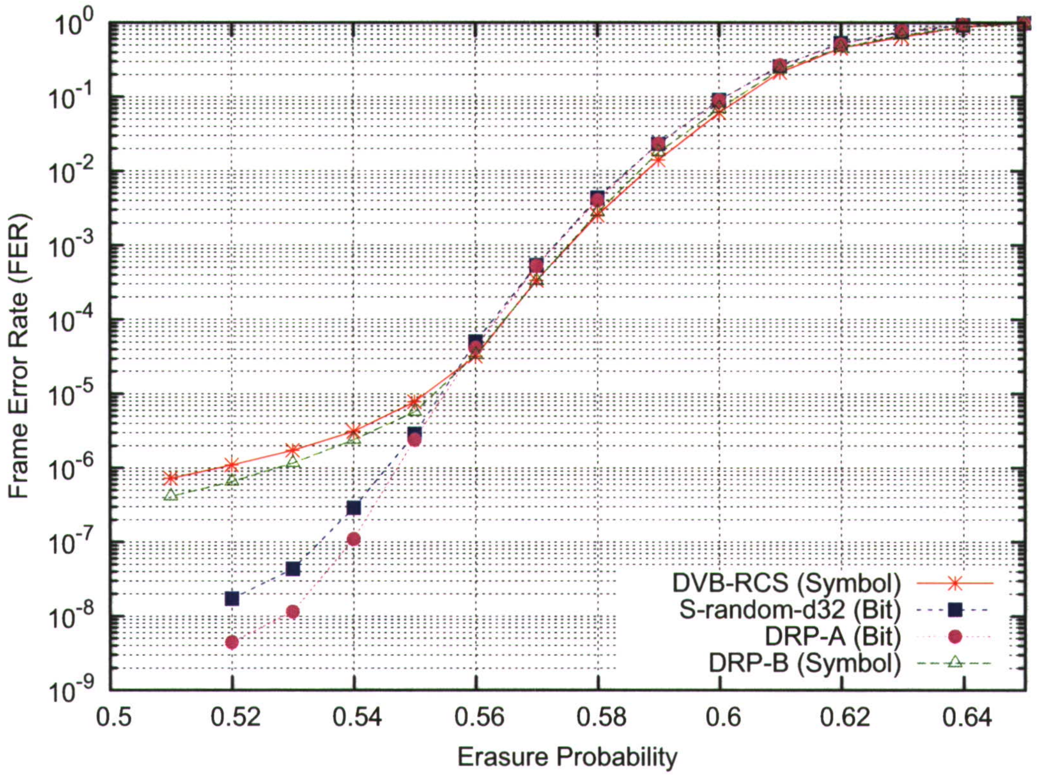


Figure 5.4: Iterative Numerical Results of DVB-RCS Turbo Codes

could be partially due to the discovered observation of the bit-based interleaver as discussed in Section 5.3.3. Furthermore, the DRP-B interleaver does make a small change on lowering the error floor compared to the error floor caused by DVB-RCS interleaver, but the performance still significantly get degraded from $\epsilon = 0.55$ and flattens out.

5.4 Probabilistic based Guessing Algorithm

Since DVB-RCS turbo codes with either symbol-based interleaver or bit-based interleaver are suffered by the existing stopping sets, a probabilistic based guessing method might be realised to help bit-based interleaver to retrieve the helpful extrinsic information from previous decoder.

For a pair of bits, if both of bits are erased, we call the event with $Ex(d_t = 00) = Ex(d_t = 11) = 0.5$ as event α, and event with $Ex(d_t = 01) = Ex(d_t =$

10) = 0.5 as event β. The different schedules by adopting the guessing algorithm to help solve event α are proposed and implemented.

- During iterative decoding, if event α occurs, guess either $Ex(d_t = 00) = 1$ or $Ex(d_t = 11) = 1$, then pass the guessed extrinsic information value to the other decoder. After certain iterations, less than 4, the decoder could easily blow out due to the incorrect extrinsic value, or the decoder could successfully decode all or most erasures due to its correctness.

- During iterative decoding, if event α occurs, guess either the first bit as 0 or 1, Then pass the new value to next decoder to check if the corresponding second bit is same as 0 or 1. If they are same and it does not corrupt the decoder, then the guess is correct , otherwise it is wrong.

By observing the different schedules during the guessing algorithm, the first method of guessing the paired bits does not fit, since the incorrect extrinsic information may mess up the entire iterative decoder, and it is unable to tell its correctness. The appropriate schedule for the guessing algorithm based on the second schedule may be realised:

- By guessing the first bit with 0, if it helps decode the entire erasures, then the guess is definitely correct with value 0.

- By guessing the first bit with 0, if the number of erasures gets progressively increased to the number of information bits as the decoding iterates, it means the bits should be guessed as 1.

- By guessing the first bit with 0, if the second bit is not equal to 0, and the final number of erasures after certain number of iterations is not equal to the number of information bits, it is necessary to guess the first bit with 1, and check the final number of erasures after certain iterations. If both final number of erasures are same and not equal to the number of information bits, it means the paired bits can not be guessed, it is advised to proceed to the next event.

According to the above analysis and observation on the trails, the special features of DVB-RCS turbo codes could be concluded as follows:

- For the bits which are well connected during the permutation, if they are erasures and correctly guessed, they must be able to help correct all or most erasures. If those erasures are incorrectly guessed, it may cause the number of erasures increased to the number of information bits after certain iterations, which helps to confirm the opposite values are the correct values and they should be able to help clear off the erasures.
- For the bits which are not well connected during the permutation, no matter what values are guessed, there is no clear sign to tell the values' correctness.

Consequently for event β, the similar criteria for the guessing algorithm is also applicable by changing the paired values from $(00, 11)$ to $(10, 01)$.

5.4.1 Numerical Results by applying the Guessing Algorithm

Since event α and event β represent different extrinsic information exchange during the bit-based interleaver, the corresponding solutions could lead more computational complexity. Hence the guessing schedule for single event α, we consider it as "Guess 1"; and the guessing schedules for both events α and β, we consider it as "Guess 2".

The simulation results by adopting "Guess 1" and "Guess 2" for DVB-RCS turbo codes in DVB-RCS bit-based interleaver are shown in Figure 5.5. For codes with S-random bit-based interleaver, the simulation results by using "Guess 1" and "Guess 2" are shown in Figure 5.6. And the results for codes with DRP-A bit-based interleaver by using "Guess 1" and "Guess 2" are shown in Figure 5.7. From the results, it is clear that the proposed guessing algorithm with either schedule arrangement is able to help improve the decoding performance, which also exposes the existing mentioned observation in the bit-based interleaver. The results using "Guess 2" achieve slightly better performance than results using "Guess 1", which shows that events α and β do exist during the iterative decoding for DVB-RCS turbo codes with bit-based interleavers. And the bit-based interleaver does suffer from such stopping sets to result in a poor decoding performance between $\epsilon = 0.66$ to $\epsilon = 0.56$. Due to the failure of guessing on the

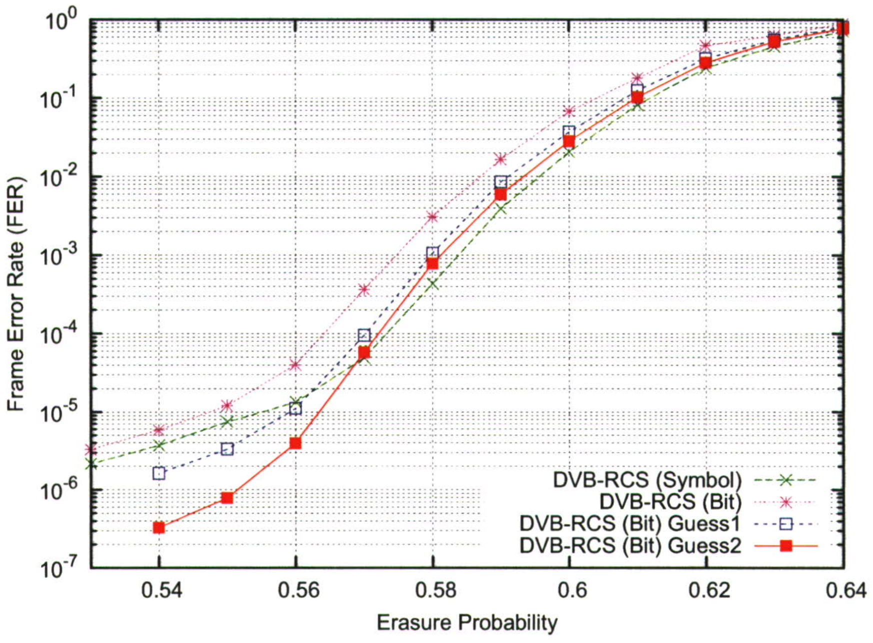


Figure 5.5: Results of DVB-RCS Turbo Codes with DVB-RCS Bit-based Interleaver

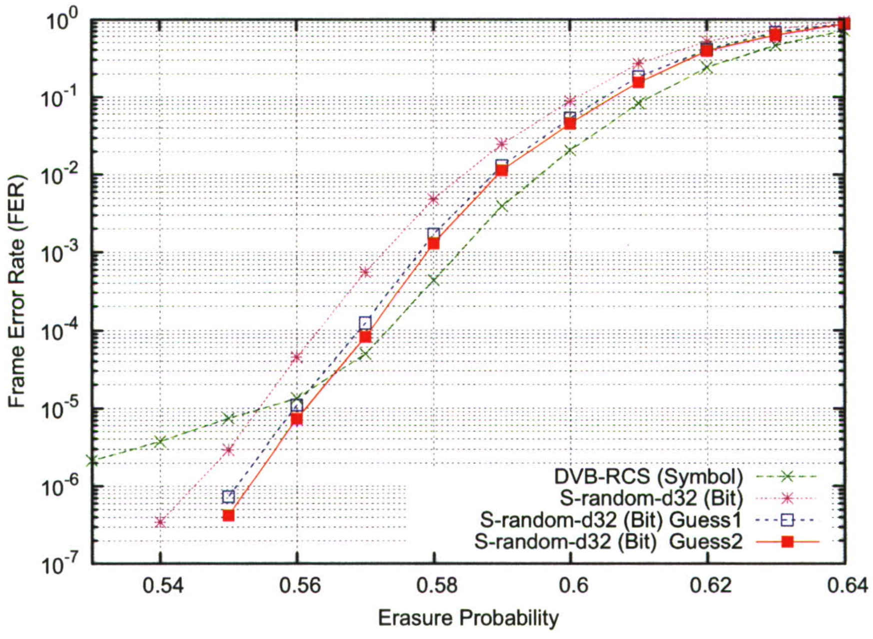


Figure 5.6: Results of DVB-RCS Turbo Codes with S-random Bit-based Interleaver

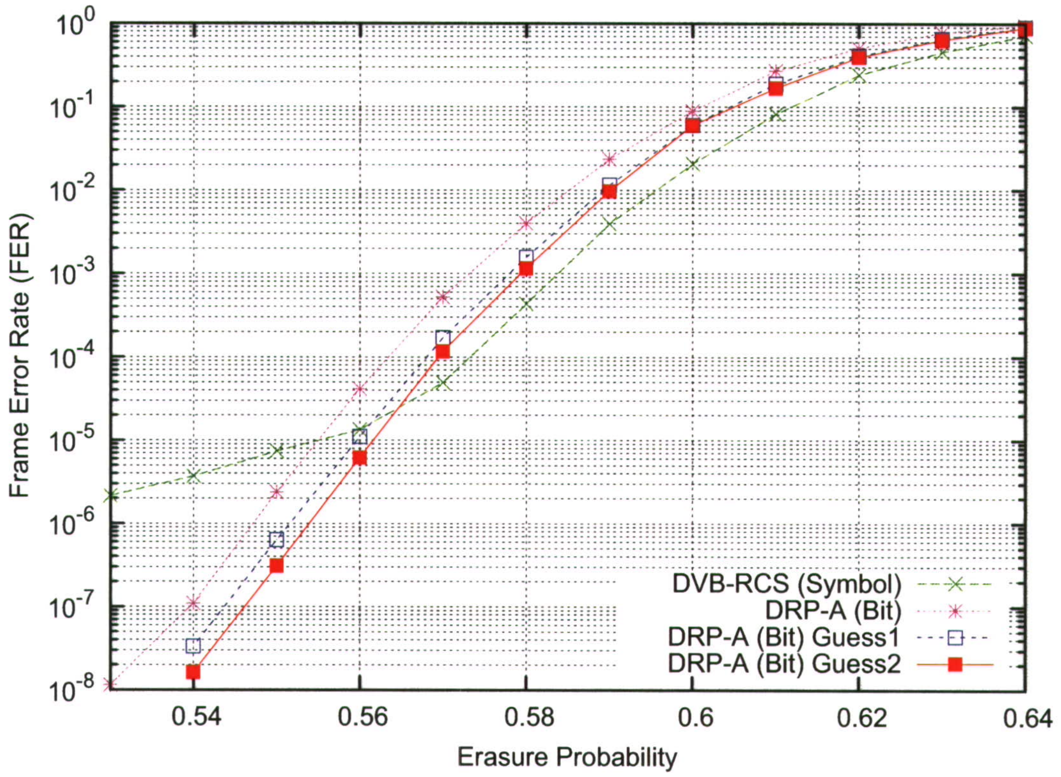


Figure 5.7: Results of DVB-RCS Turbo Codes with DRP Bit-interleaver

erasures, which are not well connected after the permutation, the best decoding performance for bit-based interleaver by using "Guess 2" still can not achieve a better performance than symbol-based interleaver at waterfall range between $\epsilon = 0.64$ and $\epsilon = 0.57$. Furthermore, beyond those bit-based interleavers, the DVB-RCS symbol-based interleaver, which ensures paired bits to stay together, still provides the best performance between $\epsilon = 0.66$ to $\epsilon = 0.56$. And such property of unchanged bits in pair in interleaver design still benefits at earlier stage of erasure probability before the error floor occurrence. Although the proposed algorithm with bit-based interleaver reasonably helps the decoder to break the stopping sets and lower the error floor, it still has not offered a comprehensive candidate to completely replace the symbol-based interleaver during the DVB-RCS turbo codes design. But the proposed guessing algorithm with either "Guess 1" or "Guess 2" does lower the error floor than symbol-based interleaver and provide better performance than using the bit-based interleaver without guessing.

The trade off between the improvement by using "Guess 2" than "Guess 1" and the need of additional computational complexity of "Guess 2" could be another convergent problem.

A sample of stopping set, which has Hamming weight of 16 for DVB-RCS turbo codes ($n = 384, k = 128$), is shown in Appendix B. As the sample shows, although the total number of parity-check equations is 256, the total number of intersected parity-check equations is only 27. Moreover for most erasures, the corresponding number of intersected parity-check equations is 3. There only exist two erasures intersected by more than 10 parity-check equations. Such erasure positions could be the key to break the stopping sets, also the ideal guessing position for the discussed event α and β. A fact may be realised that the computational complexity for the erasure decoding method to break the stopping sets, which is worked on the parity-check equations, could be largely reduced due to the limited number of parity-check equations intersected by the erasures.

5.5 Summary

In this chapter, the DVB-RCS turbo codes have been described and the construction of their parity-check matrix has been introduced. The issue between symbol-based interleaver and bit-based interleaver has been observed and analysed, the corresponding solution has been introduced by using a probabilistic based guessing algorithm, which partially helps solve the performance degradation for codes with bit-interleaver. The DVB-RCS turbo codes stopping sets over the erasure channel has been analysed and a new representation between the stopping sets and the parity-check matrix has been depicted.

Part II

Iterative Coding in Optimal Decoding Arrangement

Chapter 6

Optimum Decoding of Iterative Decodable Codes in the Erasure Channel

6.1 Introduction

Based on the analysis on iterative decoding algorithms over the erasure channel in Chapter 4, this chapter presents a hybrid decoding scheme to evaluate different arrangements of iterative decoders with a so-called “In-Place” ML decoder in terms of the computational complexity and decoding performance. Section 6.2 describes the details of the “In-Place” decoding algorithm and its reduced computational complexity over the erasure channel. The ML decoding results for codes in varied codeword lengths are presented in Section 6.2.2. The hybrid decoding arrangement scheme is described in Section 6.3. The analysis on the output from BCJR decoder at different erasure probabilities is shown in Section 6.3.1. The computational complexity by adopting different iterative decoders for the hybrid decoding arrangement is discussed in Section 6.3.2. And the optimum results by using the hybrid arrangement are evaluated and presented in Section 6.3.3. The computational complexity analysis for the “In-Place” algorithm and the complexity analysis for the hybrid decoding arrangements were presented in [131]

6.2 "In-Place" Decoding in the Erasure Channel

The "*In-Place*" algorithm was first introduced by Cai *et al.* [16], which is a Gaussian reduction algorithm avoiding the need of column-permutations over the parity-check matrix. Such decoding algorithm does not only provide a complexity reduced decoding algorithm operating the on parity-check matrix **H**, it also achieves a similar asymptotic performance as the ML decoder. It was also patented for erasure codes by Tomlinson *et al.* [120].

Let **H** be the parity-check matrix, and $\mathbf{h_i}$, $i \in \{0, 1, ..., m-1\}$ be the row vector (the parity-check equation) of **H** with a subspace of n, where m is the total number of parity-check equations, $m \geq n-k$. Let $\mathbf{C} = \{c_0, c_1, ...c_{n-1}\}$ be a codeword. Then we have for each parity-check equation $\mathbf{h_i}$

$$\left(\sum_{j=0}^{n-1}(c_j \times h_{ij})\right) \mod 2 = 0 \tag{6.1}$$

The general steps of implementing the "In-Place" algorithm are described as follows:

1. The partially erased code vector is received, and the set of erasures ϵ are substituted in the positions of erased bits in the parity-check matrix **H**.

2. First of all, an erasure ϵ_j is picked, the first picked equation containing index j is flagged and subtracted with all the other unflagged equations which contain erasure ϵ_j. Hence a new set of equations is produced.

3. The next step is to start with another erasure ϵ_l, the first picked unflagged equation containing ϵ_l is flagged and subtracted with all the other unflagged equations which contain erasure ϵ_l.

4. The procedure continues until either all the equations have been flagged or none of the unflagged equations contain the corresponding erasure ϵ as the picked equation has.

5. According to (6.1), if ϵ_j is the only erasure at single parity-check equation, then e_j can be easily decoded as "0" or "1" for c_j.

6. The decoded erasure c_j is substituted back to the equations, where $h_{ij} = 1$, then it might cause more equations containing single erasure.

7. The procedure continues until all the decoded erasures are substituted back as decoding success or there only exist equations with at least 2 erasures, which leads to decoding failure.

6.2.1 Complexity Analysis

The parity-check matrix for turbo codes or turbo Gallager codes is depicted in Figure 6.1, where k is the size of information bits. The parity-check matrix is

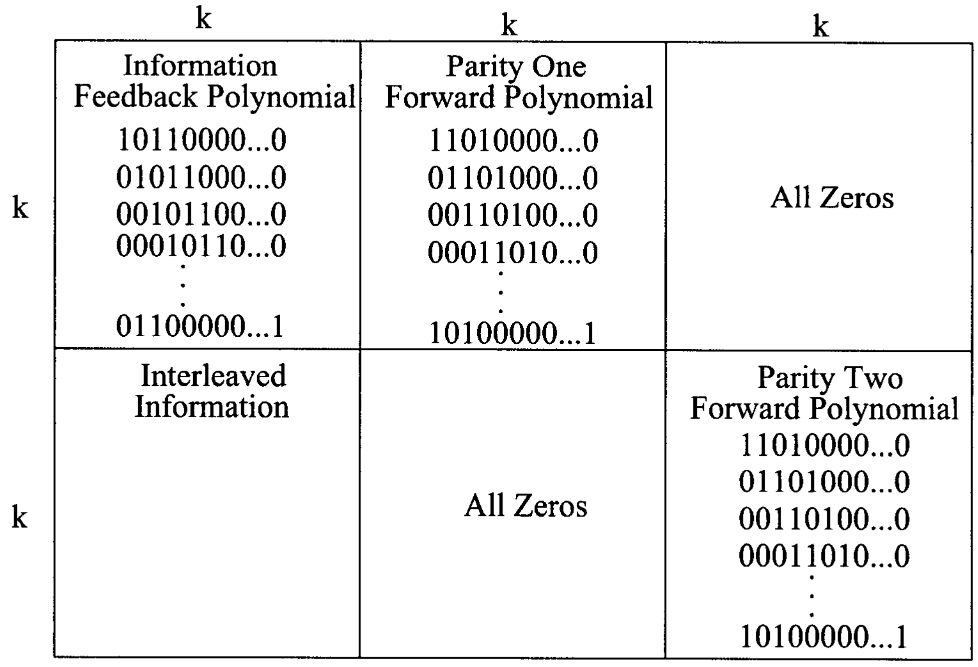


Figure 6.1: The Parity-Check Matrix for Turbo Codes (15/13)

divided into two separate sections connected by the interleaver π, since the two corresponding parity bits are generated independently by an information bit and a permuted information bit respectively. In the upper section, since the matrix is constructed in diagonal order, there are a maximum M equation additions for one equation. Thus the required number of equation additions N_o^{up} is defined as:

$$N_o^{up}(n_e) = M \times n_e \tag{6.2}$$

In the lower section, since the columns of information bits are permuted by the interleaver, we assume the maximum number of equation additions for one equation as $k-1$. Then the number of equation additions N_o^{low} is obtained as:

$$N_o^{low}(n_e) = \sum_{i=1}^{n_e-1}(k-1-i) = \frac{(n_e-1)(2k-n_e)}{2} \tag{6.3}$$

Each equation addition requires n ADDs, if using 32-bit integers to store the equation information, then there are $n/32$ sub-blocks in one single equation. Thus each equation addition requires $n/32$ ADDs. The hard decision of each decoded bit requires $n-1$ MULs and $n-2$ ADDs. In decoding one frame, the required number of operations N_o^{IP} in equivalent additions in terms of n_e for the "In-Place" decoding algorithm is computed as:

$$N_o^{IP}(n_e) = \frac{3k(n_e-1)(2k-n_e)}{64} + \frac{3kMn_e}{32} + n_e(6k-3) \tag{6.4}$$

6.2.2 Numerical Results

The ML decoding performances for turbo code (15/13) in different codeword lengths are shown in Figure 6.2, the codes interleavers all are in DRP [29] design. It is clearly shown that the ML decoding performance is improved by increasing the codeword length at same code rate, which enables the enlarged error correction ability of $n-k$ [122].

6.3 Optimum Decoding Arrangements

It is well known that for the erasure channel, any decoder that is able to solve the parity-check matrix for the channel erasures achieves maximum likelihood decoding, for example see [98]. Thus the hybrid decoding scheme includes an inner

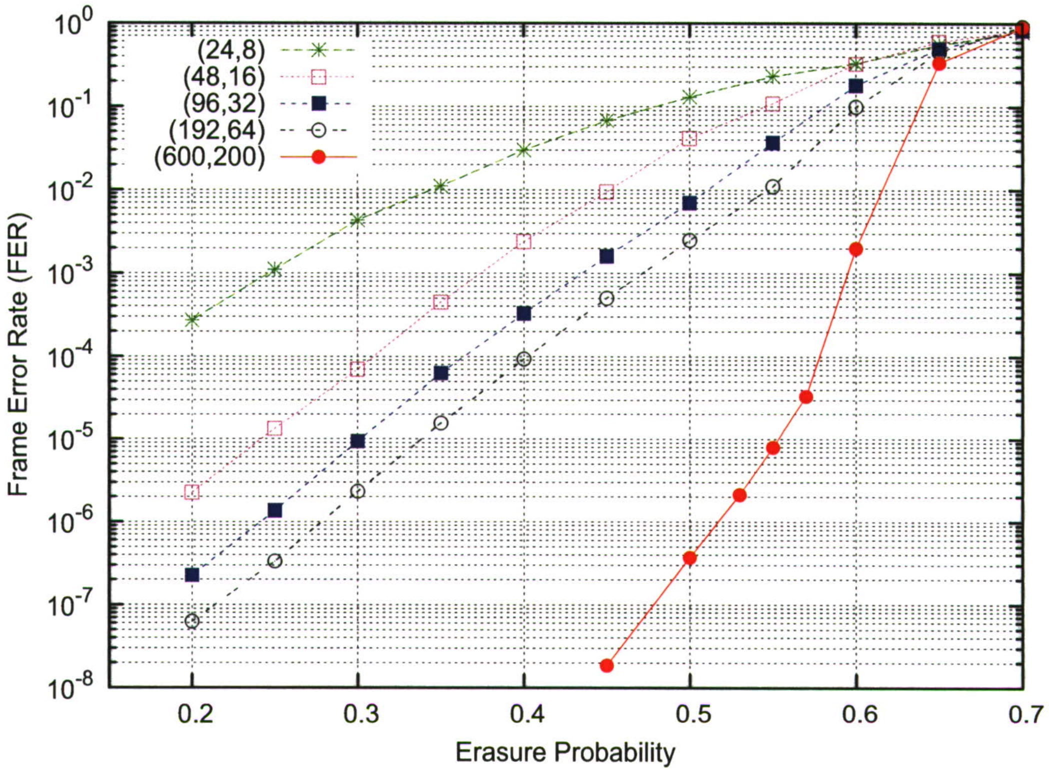


Figure 6.2: ML Decoding for Turbo Codes (15/13)

iterative decoder which is focused on either optimised LUT decoder *belief propagation* (BP) decoder. The "In-Place" decoder becomes optional and is applied to break the corresponding stopping sets. The structure of the hybrid decoding arrangement is shown in Figure 6.3, where u' and p' represent the received information bit and parity bits respectively, which correspond to the inputs to the iterative decoder. $L_e(u')$ and $L_e(p')$ represent the corresponding extrinsic information for information u' and parity p'. $D_{IT}(u')$ and $D_{IT}(p')$ are the decoded information bit and parity bit from the iterative decoder. $D_{IP}(D_{IT}(u'))$ is the decoded information bit from the "In-Place" decoder. The decoding process is iterated by passing the extrinsic information between iterative decoders, until it reaches a stopping set or it corrects all the information erasures. If a stopping set has been reached, $D_{IT}(u')$ and $D_{IT}(p')$ are passed to the "In-Place" decoder, otherwise it is unnecessary to trigger the "In-Place" decoder.

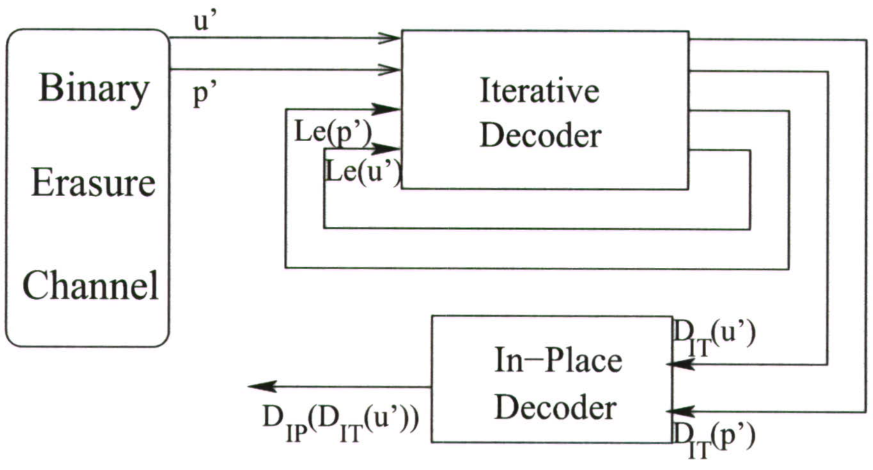


Figure 6.3: Hybrid Decoding Scheme

6.3.1 Analysis on BCJR Decoding Output

The frequency number of erasures distribution between input and output of turbo decoder for turbo code (15/13) in size of $(600, 200)$ with 50 iterations at $\epsilon = 0.6$ is shown in Figure 6.4. The highest erasure number for information bits after the erasure channel is nearly the same as $\epsilon \times k$. And the highest erasure number for information bits after turbo decoder is about 87. Thus most of the information erasures can be easily decoded by the "In-Place" decoder with nearly $(1 - R_c) \times k$ decoding ability [122].

The frequency number of erasure distribution between input and output of the turbo decoder for the same code at $\epsilon = 0.52$ is shown in Figure 6.5. The highest erasure number for information bits after the erasure channel is similar as Figure 6.4, which satisfied $0.52 \times k$. And the highest erasure number for information bits after turbo decoder is reduced to about 68. Thus more number of the information erasures could be easily decoded by the "In-Place" decoder, which leads a lower error probability.

6.3.2 Complexity Analysis of Hybrid Decoding Arrangements

Since the hybrid decoding arrangement includes an iterative decoder, the entire complexity is equal to the complexity of the iterative decoder plus the complexity

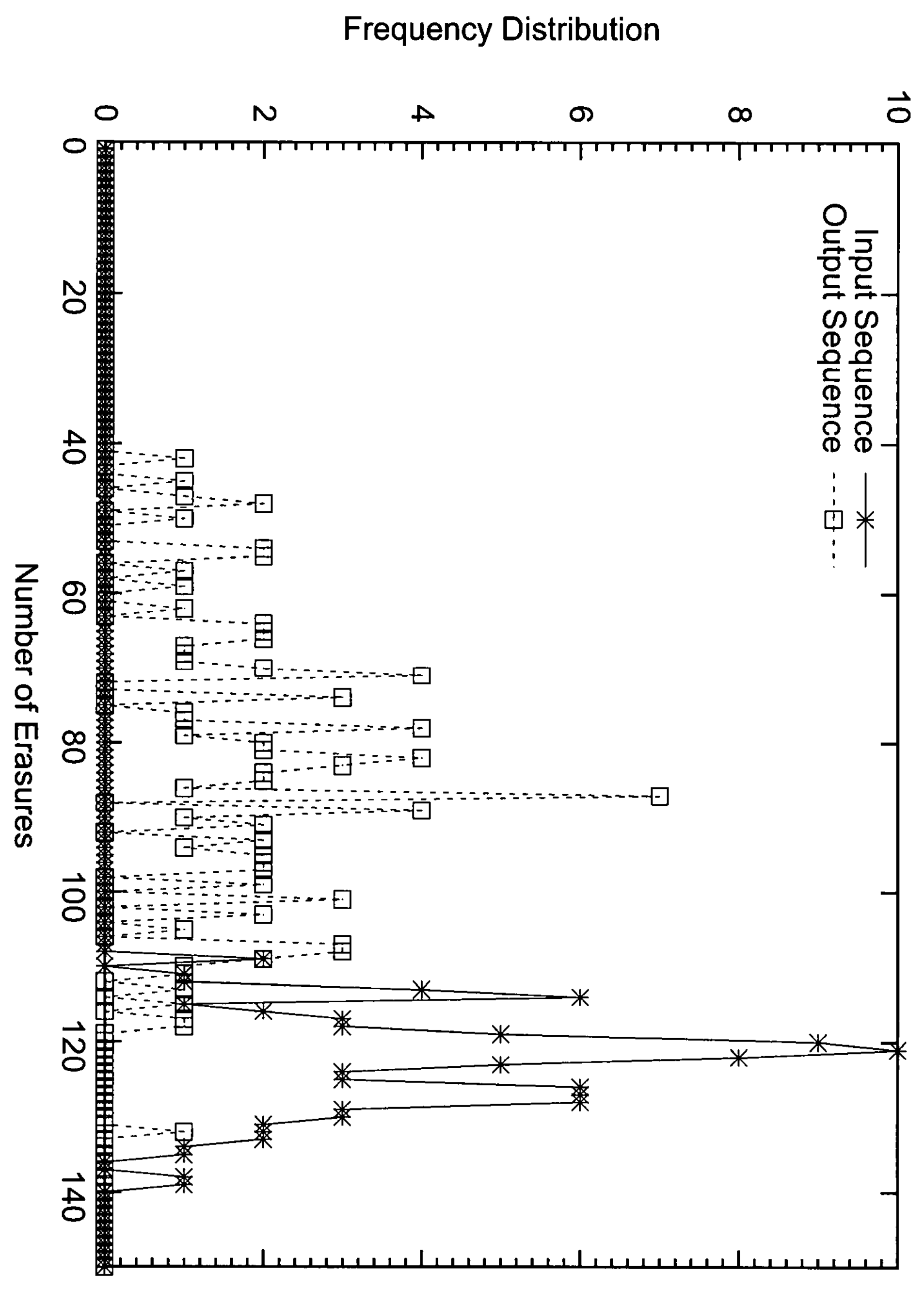


Figure 6.4: Frequency Distribution of Erasures between Input and Output of Turbo Decoder, $\epsilon = 0.60$

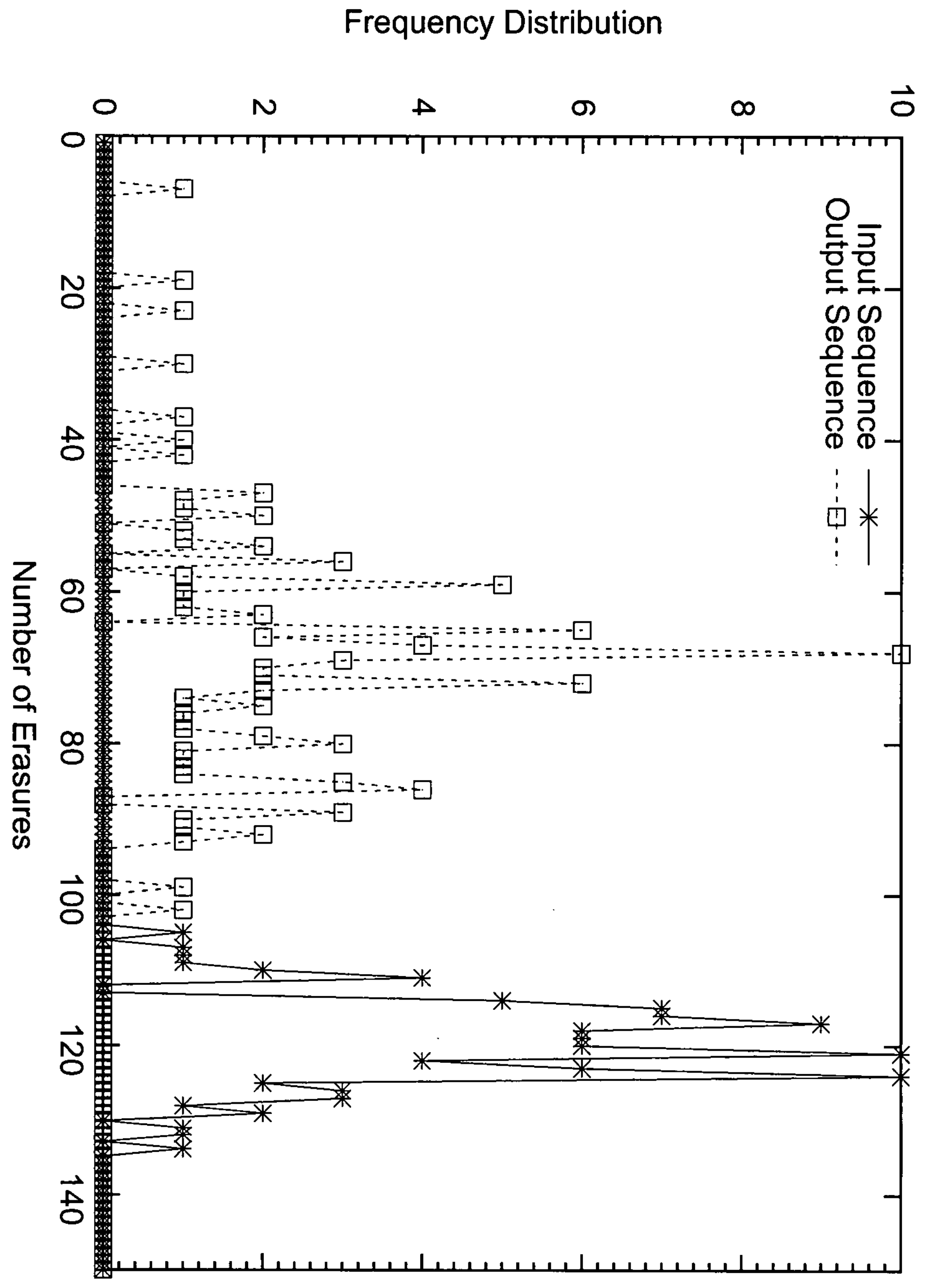


Figure 6.5: Frequency Distribution of Erasures between Input and Output of Turbo Decoder, $\epsilon = 0.52$

of the "In-Place" algorithm to solve the stopping sets, when stopping sets exist. Thus the blocks decoded correctly by the iterative decoder only include the iterative decoding computational complexity. And the blocks, which can not be decoded due to the residual erasures, include the iterative decoding complexity and the "In-place" decoding computational complexity. Then the required number of operations for each hybrid decoding arrangement is shown in Table 6.1, where L is the maximum number of iterations.

Hybrid Decoders	Iteratively Decoded Blocks	Maximum Likelihood Decoded Blocks
Hybrid (Turbo Decoder)	$N_o^T(k)$	$N_o^T(k) + N_o^{IP}(n_e^L)$
Hybrid (LUT Decoder)	$N_o^{LUT}(n_e)$	$N_o^{LUT}(n_e) + N_o^{IP}(n_e^L)$
Hybrid (BP Decoder)	$N_o^{BP}(n_e)$	$N_o^{BP}(n_e) + N_o^{IP}(n_e^L)$

Table 6.1: Required Number of Operations for Different Hybrid Decoding Arrangements

Since n_e^L is less than or equal to n_e^0, which is the input number of erasures to the iterative decoder, each hybrid decoding arrangement provides a reduced complexity decoding scheme than the "In-Place decoder alone. The convergent decoding performance of each arrangement in terms of computational complexity only depends on the number of erasures remaining after the iterative decoder, and also the iterative decoding performance. It should be noted that as all of the decoders are maximum likelihood decoders, they all achieve the same optimum performance. The differences are the computational complexity and the impact on the decoding speed. This is primarily determined by the effectiveness of the first stage iterative decoder: the better the performance of the iterative decoder achieves, the less the computational complexity of the hybrid decoding arrangement requires.

The computational complexity comparison between hybrid decoding arrangements with different iterative decoders followed by an "In-Place" decoder and ML decoder alone is shown in Figure 6.6. It is clear that both hybrid decoding arrangements provide reduced complexity algorithm by combining either optimised iterative LUT decoder or BP decoder with the "In-Place" decoder. The

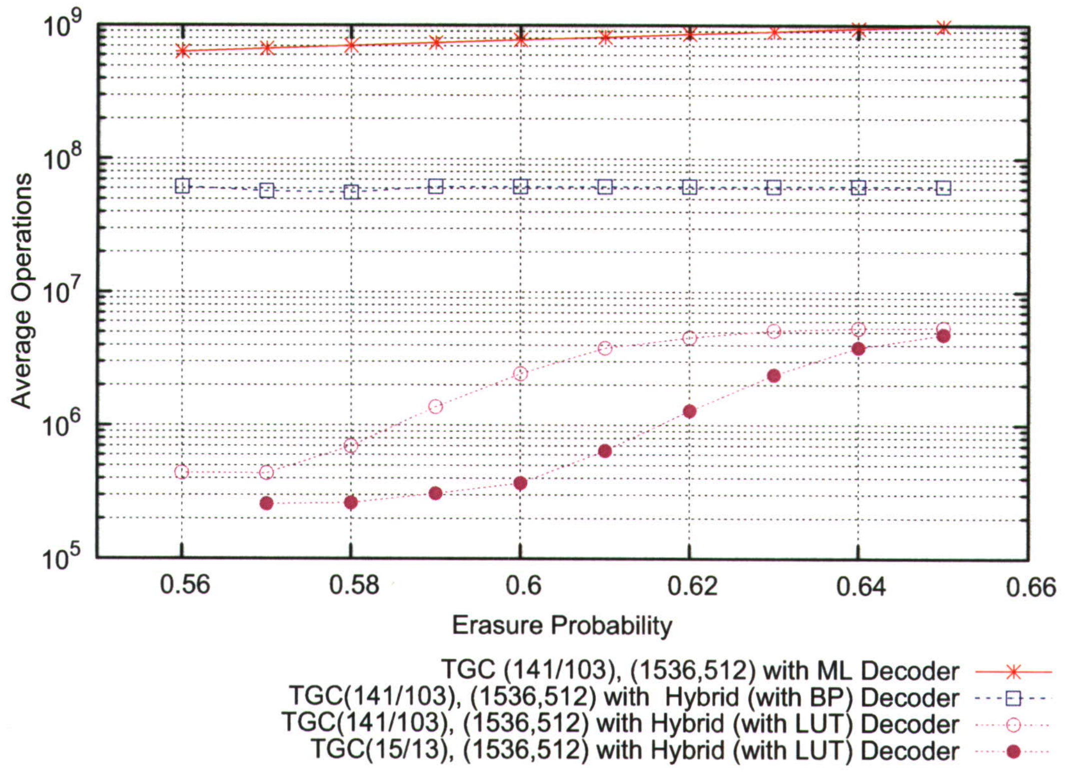


Figure 6.6: Complexity Comparison between Hybrid Decoding and ML Decoding

optimised iterative LUT decoding is capable of converging better with the "In-Place" decoder for TGC (15/13) or TGC (141/103) in terms of reduced decoding complexity.

6.3.3 Numerical Results

Computer simulations have been carried out to assess the performance of TGCs by adopting different decoding arrangements. First of all, we compare the results of TGC (15/13) with codeword length of $(600, 200)$ with DRP [29] interleaver in the arrangement of optimised iterative LUT decoder and an "In-Place" decoder as shown in Figure 6.7. From the results, we can see that optimised iterative LUT decoder paired with "In-Place" decoder achieves the optimum performance as we expect. Furthermore, at $\epsilon = 0.5$, optimised LUT decoder nearly achieves the similar performance as the "In-Place" decoder achieves differentiated by a

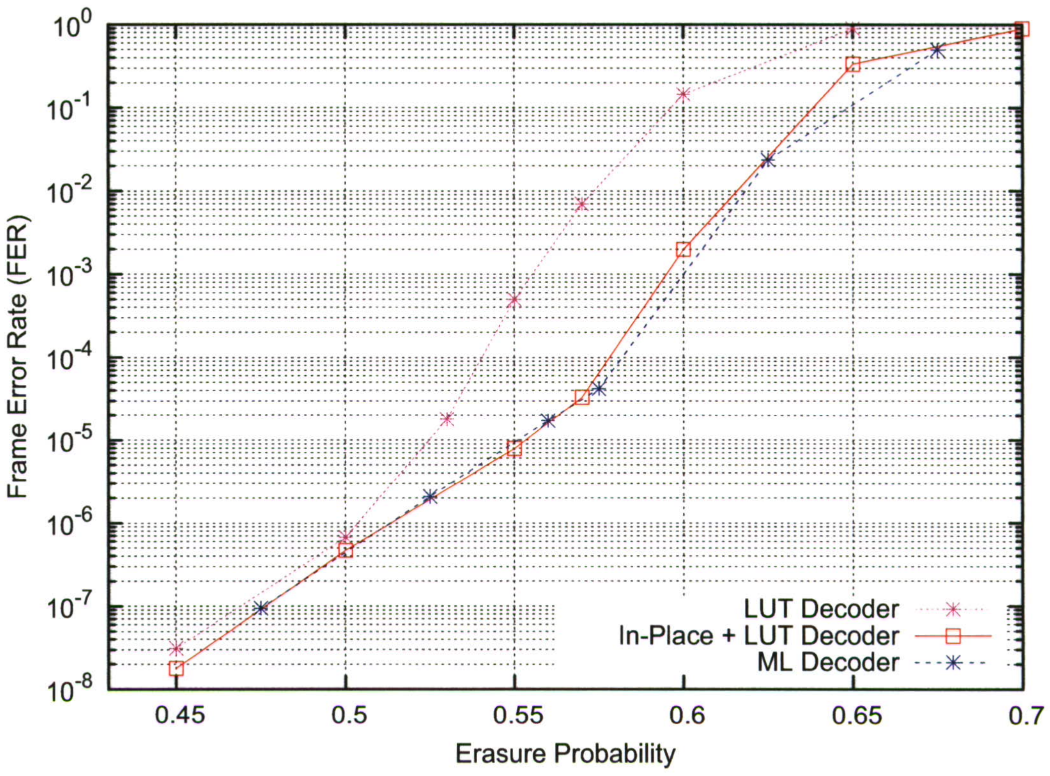


Figure 6.7: Results of TGC (15/13) ($n = 600, k = 200$)

small gap, which means most blocks should be able to be successfully decoded solely by the optimised iterative LUT decoder.

Results of TGC (15/13) in size of (1536, 512) permuted by DRP [29] interleaver with optimised iterative LUT decoder, hybrid decoder and ML decoder are shown in Figure 6.8. From the results, we can see that hybrid decoder is able to achieve the exactly same performance as ML decoder does. Furthermore, the performance differences between ML decoder and optimised iterative LUT decoder get smaller as the erasure probability decreases. By combining with an "In-Place" algorithm, the hybrid decoder provides a solution to achieve ML performance with reduced complexity due to the well performing optimised iterative LUT decoder.

Since the computational complexity of BP decoding does not directly depend on the code constraint length of the shift register, it will enable the belief propagation decoding algorithm capable of decoding turbo codes with constituent convolutional codes in long constraint length, and hence be potentially charac-

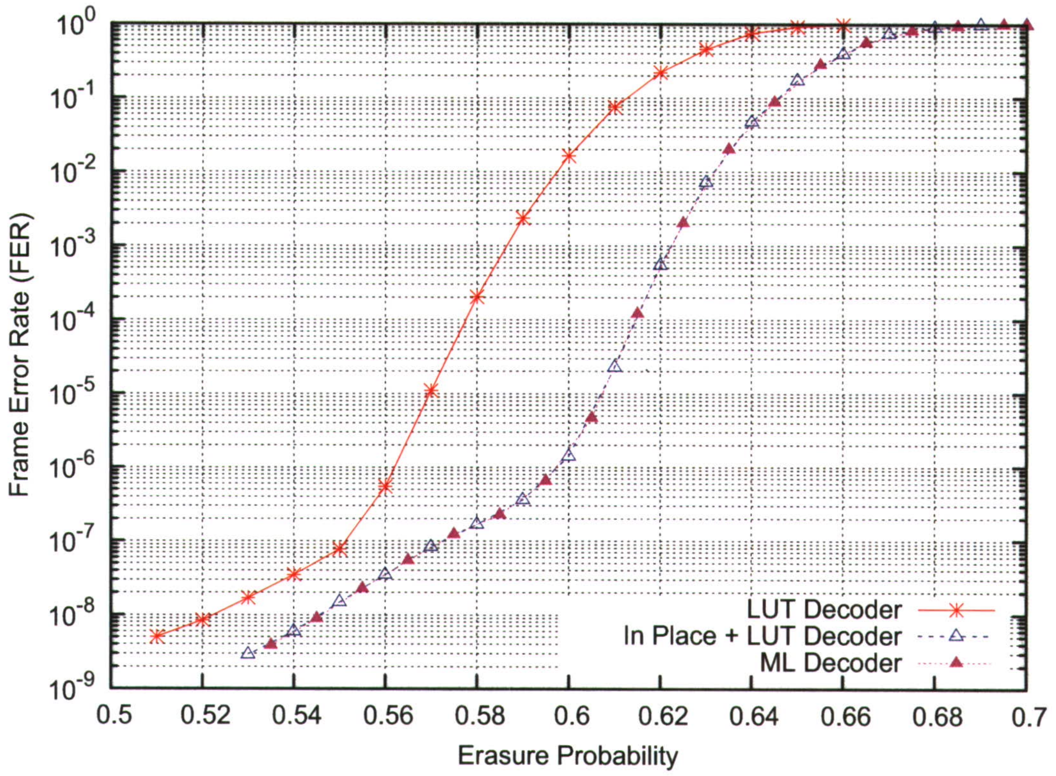


Figure 6.8: Results of TGC (15/13) ($n = 1536, k = 512$)

terised by a large free distance. The decoding complexity of the BCJR algorithm grows exponentially with the code constraint length, the alternative iterative decoder cannot be practically used for codes with long constraint lengths. In order to compare and identify the performances of different iterative decoders based on the same code, or code structure, we consider the proposed code TGC $(0, 3, 4/0, 14, 34)$ with DRP [29] interleaver as one of the reference code, which can only be decoded by BP decoder due to its long constraint length. Since the BP decoding performance of TGC (141/103) nearly reaches the best performance in the class of UMTS codes, and its constraint length is still manageable by the optimised LUT decoder, we consider this code as another key code for comparison. Furthermore TGC (15/13) with optimised iterative LUT decoder achieves better performance than TGC (141/103), it is also considered during the comparison. The comparison between TGC (15/13), TGC (141/103) and TGC $(0, 3, 4/0, 14, 34)$ is shown in Figure 6.9. For all the testing codes in Fig-

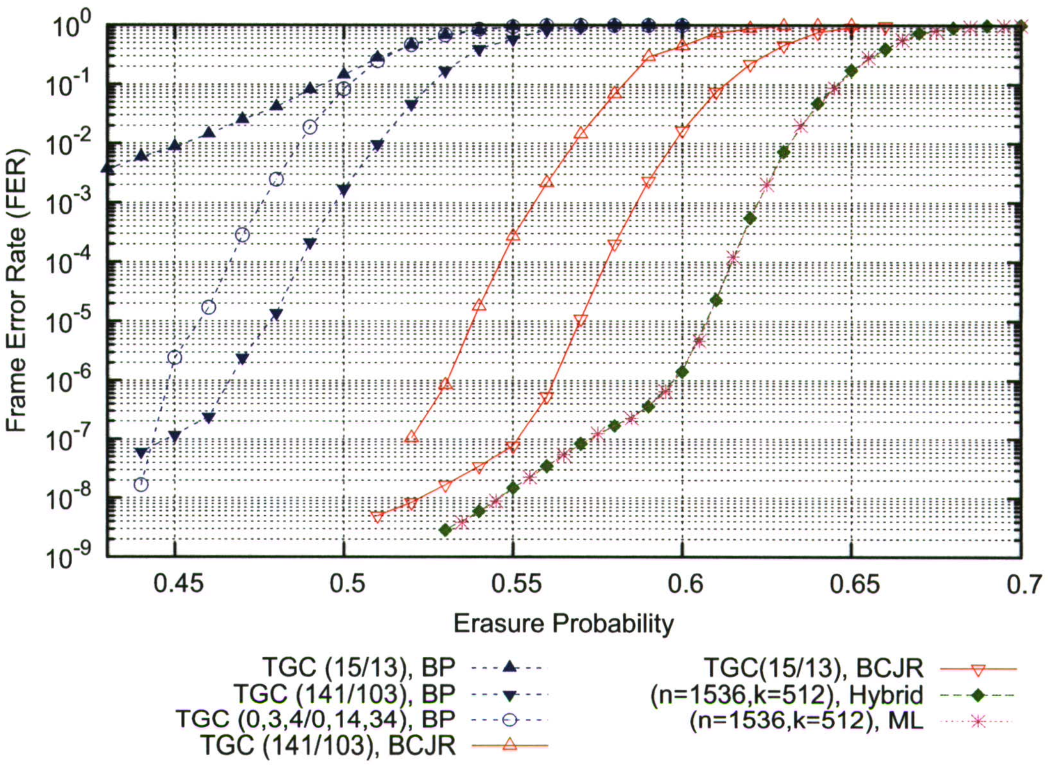


Figure 6.9: FER Performances of Turbo Gallager Codes

ure 6.9, they are constructed by using the same DRP interleaver with code size $(1536, 512)$. From the results, it is apparent that optimised iterative LUT decoder provides significant improvement over the BP decoder due to its better convergence over the erasure channel. There exists a coding gain difference of 0.1 in erasure probability for same TGC (141/103) with different iterative decoders. TGC (15/13) achieves the best iterative decoders' performance with less computational complexity. Especially, at erasure probability $\epsilon = 0.55$, there exists the smallest performance difference between optimised iterative LUT decoder and ML decoder, which means most of blocks could be successfully decoded by the optimised iterative LUT decoder. The hybrid decoder achieves the exactly same performance as ML decoding does. Since the performance of hybrid decoder only depends on the length of the codeword, thus for same code with the same code rate, the design of turbo Gallager codes does not affect the ML-like performance achieved by the hybrid decoder. Hence TGC (15/13) with the best iterative performance and the least decoding complexity becomes an ideal candidate to be considered for the proposed encoding-decoding scheme in the erasure channel in terms of optimum decoding achievement with less computational complexity.

We also analyse the relationship between the percentage of packets successfully decoded by the iterative decoders and the percentage of packets passed to the hybrid decoder to achieve the ML performance. The comparison between the different iterative decoders for TGC $(15/13)$, $(141/103)$ and $(0,3,4/0,14,34)$ is shown in Figure 6.10. It is clear that the optimised iterative LUT decoder

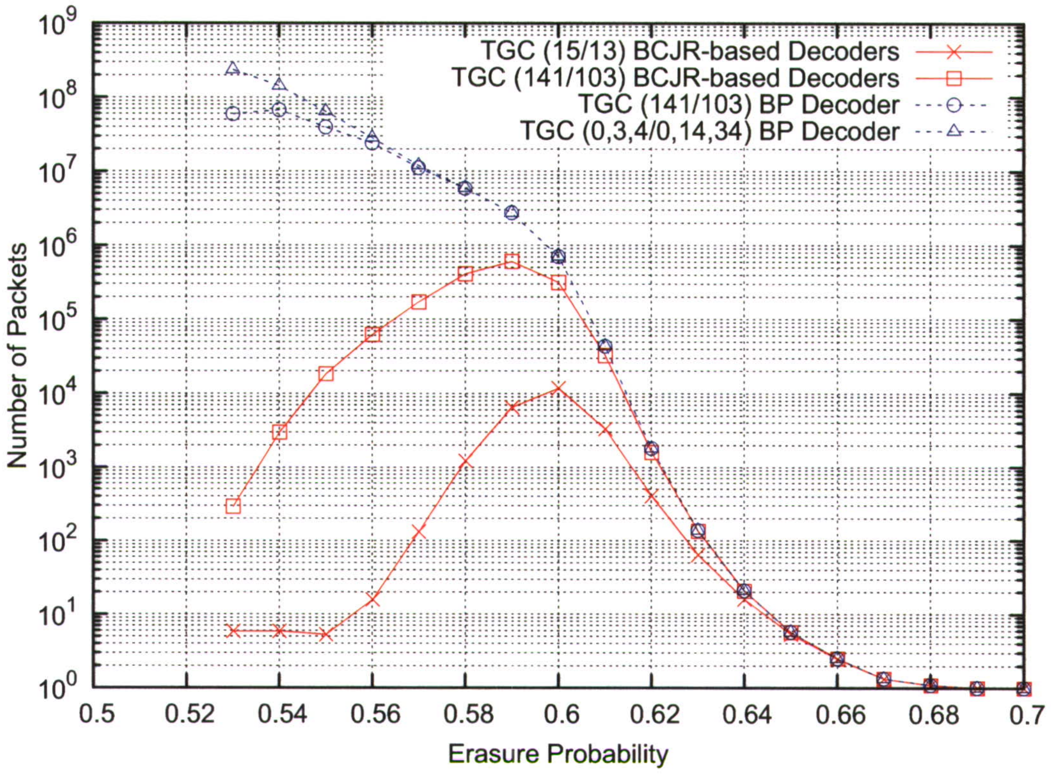


Figure 6.10: Percentage of Packets for ML Decoder

requires less packets with residual erasures to be passed to the ML decoder. And comparing to TGC (141/103), TGC (15/13) achieves the best convergence between code and the decoder. Especially at $\epsilon = 0.55$, when the other decoders need to pass from 10^4 to 10^8 blocks to the ML decoder, TGC (15/13) with optimised iterative LUT decoder only needs to pass less than 10 packets to the "In-Place" ML decoder.

6.4 Summary

In this chapter, two different iterative decoding algorithms have been compared by coupling with an "In-Place" ML decoder over the erasure channel for turbo Gallager codes. Both hybrid decoding arrangements have provided reduced complexity decoding compared to a stand alone "In-Place" decoder, and all decoders have achieved ML performance. Due to the existing performance difference between the two iterative decoders, the BCJR-based decoders using low memory order of turbo Gallager codes has provided the best trade-off between convergent performance and computational complexity. Good performance has been obtained for turbo Gallager code (15/13) with small memory, which could be a good candidate for a hybrid decoding scheme using the LUT decoder with significantly reduced complexity.

Chapter 7

Optimal Decoding for Iterative Codes in the AWGN Channel

7.1 Background

In terms of the soft decision based optimal decoding performance for the AWGN channel (also called *maximum-likelihood*, ML), LDPC codes with ML decoding in general is not feasible. In 1995, Fossorier proposed the sub-optimal decoder, called *ordered statistics decoder* (OSD) [44], which aimed at searching the re-ordered most-reliable k information bits for the maximised codeword with a constraint of order i. Better performance was later achieved with the same order of i in 2002 by Fossorier [43]. Valembois & Fossorier [123] based on Fossorier's previous work proposed the "box and match techniques" to help further improve the decoding performance. By adding the CRC check into the code to construct a concatenated code with OSD decoder to help lower the error floor was proposed by Gounai & Ohtsuki [47]. An extended Dorsch decoder [34] was proposed by Tomlinson *et al.* [121] in 2007, aimed at achieving near optimal ML decoding for linear block codes by searching the k information bits for error patterns, leading to differential low weight codewords.

By following the hybrid decoding arrangements for turbo Gallager codes over the erasure channel as discussed in Chapter 6, which combines an optimised iterative decoder using (BP or BCJR) and an ML "In-Place" decoder [16]. In this chapter, a new decoding arrangement is considered for the AWGN channel using linear block codes with sparse parity-check matrices, like LDPC codes. The corresponding iterative BP decoder is used as the initial decoder, the iterative output is conditioned so that it is best suitable as the input to the OSD-i instead of using the traditional soft iterative output. As a consequence improved results are obtained and successfully break the corresponding error floors caused by BP stopping sets. The basis of the conditioning of the output from the iterative decoder is explained with supporting analysis in Section 7.2. The difference between using the standard iterative output from the BP decoder and the conditioned output is analysed and compared in Section 7.3. Section 7.4 gives results showing the relative performances of the OSD-i, the new proposed decoding arrangement and the BP decoder for some well known LDPC codes and a class of cyclic codes. The related decoding algorithm and optimal performances will be presented in [132].

7.2 Decoding beyond Iterative Decoding for the AWGN Channel

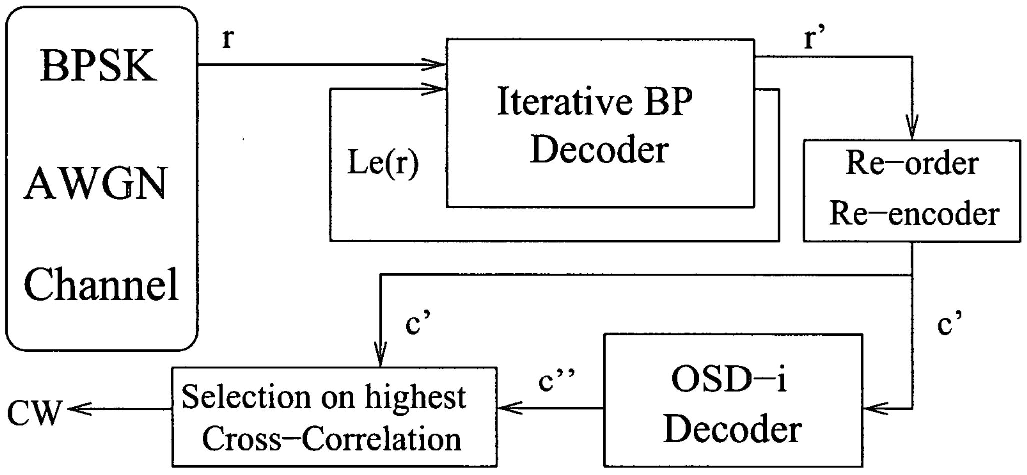


Figure 7.1: The Proposed Decoding Structure

Let $\mathcal{C}$ be a linear block code (n, k, d_{min}), the proposed decoding structure for linear block codes over the AWGN channel is shown in Figure 7.1, where $\mathbf{r}$ is the received signal vector plus the AWGN channel noise vector of *variance* σ^2. The BP iterative decoder produces the output $\mathbf{r}'$ after a given number of iterations.

Then $\mathbf{r}'$ is conditioned to become $\hat{\mathbf{r}}'$ which is permuted and re-encoded by the corresponding generator matrix $\mathbf{G}^{\pi}$, where π is the index interleaver determined primarily by the bit log likelihood ratios, and secondly by the column swaps to achieve full rank. The new generated codeword $\mathbf{c}'$ is passed to OSD-i to search for the derived codewords $\mathbf{c}''$, which achieves the highest cross-correlation with the received vector, under the constrained order i. During the OSD-i decoding, the cross-correlation of $\mathbf{c}'$ based on $\hat{\mathbf{r}}'$ is used as a lower bound to limit the search size of the OSD-i. If OSD-i is unable to find any codeword with higher cross-correlation than $\mathbf{c}'$, constrained by the codeword search size $\binom{k}{i}$, then $\mathbf{c}'$ is selected as the output codeword "cw".

7.2.1 Iterative Decoding

In the earlier approach introduced by Fossorier [42], OSD-i decoding was attempted with each iteration switching back and forth between OSD decoding and iterative decoding. Here we show that better results may be achieved by the simpler approach of carrying out BP decoding at a fixed number of iterations L, before invoking OSD-i decoding without switching back and forth. For LDPC codes like linear block codes, whose parity-check matrices are sparse having few, if any, cycles of length 4, the iterative BP decoder during most of the time is able to improve the extrinsic information at each iteration. After L iterations, instead of passing the output vector $\mathbf{r}'$ directly to the OSD-i as in [42], the output vector is conditioned to become $\hat{\mathbf{r}}'$ and then passed to the OSD-i. This is done to avoid occasions when the iterative decoder destroys some of the received information prior to passing it to the OSD-i.

7.2.2 OSD-i Decoding and Construction of Equivalent Generator Matrices

As in conventional OSD-i decoding, the decoding is based on determining information sets, vectors of length k bits which can be used to generate codewords of the code. We want the generated n-bit codewords to be close in *Euclidean distance* to the input vector of length n. Correspondingly, the input vector $\hat{\mathbf{r}}'$

from the iterative decoder is permuted in order of reliability based on the *log likelihood ratio* given by $|\hat{\mathbf{r}}'|$ to become $\mathbf{x}$ plus second order considerations based on the need of full rank achievement for the constructed generator matrix. The permuted input is $x_m = \pi(\hat{r}'_m)$, where π denotes the required permutation. The permuted $\mathbf{x}$ vector consists of almost the *most reliable bits* (MRB) from most reliable, extreme left and the *least reliable bits* (LRB) from extreme right, where $\{|x_0| > |x_1| >, ..., > |x_{n-1}|\}$. In order to obtain full rank of the new generator matrix $\mathbf{G}$, *Gaussian elimination* is performed by starting from the most reliable bit in LRB x_k and progressing towards x_{n-1}. At some point, there will be one bit, $x_{k+\delta}$ which is not independent of the previously solved for bits and thus cannot be solved for. It is not possible for this bit to be a parity bit, given the previous choices for parity bits. This is indicated by $x_{k+\delta}$, not being present in any of the remaining, uncommitted parity-check equations. The procedure in this circumstance is to skip bit $x_{k+\delta}$ and try to solve for the next more reliable bit $x_{k+\delta-1}$, solving if possible, skipping if not, and continue in this way. In practice, very few bits have to be skipped in this way and skipped bits have almost the same reliability [121] as the bits that replace them in the LRB. Thus the ordered received vector $\mathbf{x}$ is interleaved by index factor $\bar{\pi}$ as $\bar{\mathbf{x}}$, due to the column swaps. Then we have the updated MRB from extreme left as $\{|\bar{x}_0| > |\bar{x}_1|, ..., > |\bar{x}_{k-1}|\}$ and the LRB from extreme right as $\{|\bar{x}_k| > |\bar{x}_{k+1}|, ..., > |\bar{x}_{n-1}|\}$.

7.3 BP Output Impact on OSD-i

For the AWGN channel, an ML decoder searches for the codeword out of all the possible codewords which has the highest cross-correlation, Y_{max} with the received vector $\mathbf{r}$.

$$Y_{max} = \sum_{j=0}^{n-1} |r_j| \tag{7.1}$$

Let $\hat{\mathbf{c}} = (\hat{c}_0, \hat{c}_1, ..., \hat{c}_{n-1})$ be the hard-decision binary code vector from $\mathbf{r}$. It is noted that the Y_{max} is achievable, if and only if the code $\hat{\mathbf{c}}$ is a valid codeword. Let $\mathcal{C}_1$ be the transmitted codeword resulting in the received vector $\mathbf{r}$, and $Y(\mathcal{C}_1)$ its cross-correlation with $\mathbf{r}$.

$$Y(\mathcal{C}_1) = \sum_{j=0}^{n-1} |r_j| \cdot (1 - \mathcal{C}_{1_j} \oplus \hat{c}_j) \tag{7.2}$$

For the decoder there exist a set of codewords $\mathcal{C}_\mathbf{2}$, where $Y_{max} \geq Y(\mathcal{C}_2) \geq Y(\mathcal{C}_1)$.

- **Case A:** If there exists any $\mathcal{C}_2$ with $Y(\mathcal{C}_2) > Y(\mathcal{C}_1)$, an ML type decoding error will occur.

- **Case B:** If the best codeword $\mathcal{C}_2$ with $Y(\mathcal{C}_2) = Y(\mathcal{C}_1)$, the codeword with maximum correlation with $\mathbf{r}$ is the transmitted codeword and the decoder has achieved successful decoding.

For constrained codeword search, aiming to achieve convergent output performance, there exist the following situations

- For case A, if there exists any $\mathcal{C}_2$ with $Y(\mathcal{C}_2) > Y(\mathcal{C}_1)$ during the constrained search, an ML decoding error is obtained. Otherwise if there exists $\mathcal{C}_2$ with $Y(\mathcal{C}_2) = Y(\mathcal{C}_1)$, the transmitted codeword is found. If no codeword is found that satisfied all of the parity-check equations, a non-ML decoding error occurs.

- For case B, there only exists that the transmitted codeword with $Y(\mathcal{C}_1) = Y(\mathcal{C}_2)$ corresponding to $\mathbf{r}$ is found or a non-ML decoding error is indicated.

Most codeword-search algorithms adopt various constraints aimed at achieving near-optimum performance with smaller search size. With this aim, let $\mathbf{c}'$ be the re-encoded codeword based on re-ordered k information bits from $\mathbf{r}'$. The maximum attainable cross-correlation $Y(\mathbf{c}')$ provides an upper bound which helps limit the size of search. The re-encoded codeword may contain a smaller number of errors in the MRB than the MRB corresponding to the received vector $\mathbf{r}$. Thus using input vector $\hat{\mathbf{r}}'$ leads to more successful decoding by an OSD with smaller order i than using input vector $\mathbf{r}$.

7.3.1 Soft Iterative Output $\check{\mathbf{r}}'$

In general, the *a posteriori probability* (APP) value of L in logarithm for bit m during the iterative decoding process is expressed as the sum of three terms [92]

$$L_m = L_m^c + L_m^a + L_m^e \tag{7.3}$$

where L_m^c denotes the *channel measurement*, which is the effect of channel output corresponding to bit m. L_m^a represents the a *priori* value in logarithm, it is the function of the a priori probability of bit m. The final term is the independent *extrinsic information* about bit m. In terms of reliability, the relationship between received bit r_m and updated $\check{r}'_m$ according to the soft output r'_m from the iterative decoder is expressed as

$$\check{r}'_m = r_m \times \frac{\log\left(\frac{p(c_m=0|r'_m)}{p(c_m=1|r'_m)}\right)}{\log\left(\frac{p(c_m=0|r_m)}{p(c_m=1|r_m)}\right)} = r_m \cdot \left(\frac{L_m}{L_m^c + L_m^a}\right) \tag{7.4}$$

which is derived in Appendix A. Thus as $\check{Y}_{max}$ is limited to $\sum_{j=0}^{n-1} |\check{r}'_j|$, and there may exist $\check{Y}_{max} \neq Y_{max}$ after a certain number of iterations. The cross correlation of this vector with the transmitted codeword $\check{Y}(\mathcal{C}_1)$ might result in a reduction compared to the original received vector. Correspondingly the set of codewords $\mathcal{C}_2$, which produce ML decoding errors, will be increased in size as a new set $\check{\mathcal{C}}_2$. The following possibilities exist

- If $\check{Y}(\mathcal{C}_1)$ is reduced, $|\check{\mathcal{C}}_2| > |\mathcal{C}_2|$, where $|\mathcal{C}_2|$ denotes the size of the codeword set $\mathcal{C}_2$.
- Or if $\check{Y}(\mathcal{C}_1)$ increases, $|\check{\mathcal{C}}_2| < |\mathcal{C}_2|$.

Let $\mathbf{c}$ be a codeword generated by the corresponding generator matrix $\check{\mathbf{G}}$, $\mathbf{c} \in \{\check{\mathcal{C}}_2 \backslash \mathcal{C}_2\}$. Then $\check{Y}(\mathcal{C}_1) < \check{Y}(\mathbf{c}) < Y(\mathcal{C}_1) < Y(\mathcal{C}_2)$, where $\check{Y}$ denotes the cross-correlation of a codeword based on $\check{\mathbf{r}}'$, and Y is the cross-correlation of the same codeword based on $\mathbf{r}$. Once a codeword $\mathbf{c}$ is found, with higher cross-correlation than the transmitted codeword then there is an ML decoding error based on $\check{\mathbf{r}}'$. However this is not a real ML decoding error because if the received vector

r is used the cross correlation of codeword **c** is less than the cross correlation of the transmitted codeword. In practice, it has been observed that this is a common event and the decoding performance is significantly degraded through this mechanism.

Besides the above issue, there is another reason that the unconditioned soft iterative output is not an ideal input for decoding. Stopping sets and trapping sets, considered in the LDPC codes design has become a major issue due to the degraded performance in the AWGN channel at high SNR [95]. These are not only due to cycles of length 4, but also cycles of length 6 and 8. If there exists a set of bits, which form a cycle, then the extrinsic information of these bits can be destructive as the BP decoder iterates. Thus the sign and magnitudes of bit log likelihood ratios can change for the worse as the BP decoder iterates leading to some bits in the LRB of the received vector swapping for bits in the MRB of the received vector leading to an increase in the number of bit errors in the MRB. This will seriously affect the performance of OSD-i due to its order limitation. The similar phenomenon was also observed in [121].

The frequency distributions of the number of bit errors in the MRB as a result of iterative decoding after different iterations are shown in Figure 7.2. These were obtained by evaluating 10^6 received vectors for Tanner codes $(155, 64, 20)$ [118] at 4dB $\frac{E_b}{N_o}$. The x-axis shows the number of errors in the MRB input to the OSD-i, and the y-axis denotes the number of input vectors having the same number of errors in MRB. First of all, as the iteration is increased, the number of blocks with null error in MRB is increased. It clearly shows the better performance could be achieved as more iterations are called. After the first iteration, the maximum number of errors in MRB is only 3, but after 5 iterations, the maximum number of errors in MRB is increased to 6. Furthermore at 10 iterations, it produces the maximum number of errors as 8 in MRB, even after 50 iterations, the maximum number of errors in MRB is still 7. Thus for the 10^6 blocks, all of them could be successfully decoded after 1 iteration by OSD decoder up to 3, but part of them might have to be decoded by OSD decoder up to 7 after 50 iterations. It clearly shows that the discussed issue occurs more frequently as the BP decoder iterates. And more error bits with higher magnitudes are shifted to the MRB.

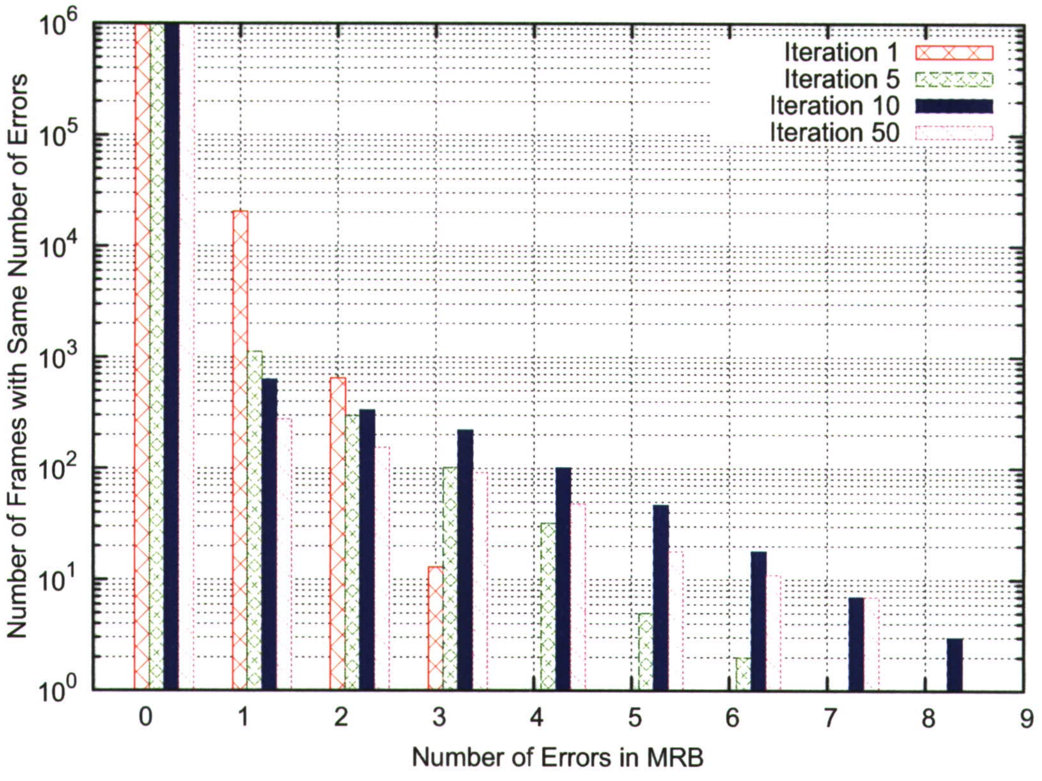


Figure 7.2: Comparison of Soft Output for Tanner Codes $(155, 64, 20)$ at $\frac{E_b}{N_o}$=4dB with Different Iterations

7.3.2 Conditioned iterative Output $\hat{\mathbf{r}}'$

In order to avoid the above issues, we need to maintain Y_{max} and yet use the iterative decoder output to assist OSD-i decoding, the conditioned output $\hat{\mathbf{r}}'$ of the iterative decoder is used for OSD-i decoding. The conditioned output $\hat{r}'_m$ is defined as

$$\hat{r}'_m = \begin{cases} r_m & , if \frac{r'_m}{r_m} \geq 0 \\ -r_m & , if \frac{r'_m}{r_m} < 0 \end{cases} \tag{7.5}$$

Thus

$$\hat{Y}_{max} = \sum_{j=0}^{n-1} |\hat{r}'_j| = Y_{max} \tag{7.6}$$

According to (7.3), it is noted that the log likelihood ratio $\check{L}_m$ from $\check{\mathbf{r}}'$ is equivalent to L_m. In this case, the extrinsic knowledge about bit m is evaluated as either "1" or "-1". Thus the conditioned iterative APP value $\hat{L}_m$ log likelihood ratio is described as

$$\hat{L}_m = L^c_m + L^a_m + \log(-1)^{\hat{b}'_m} \tag{7.7}$$

where $\hat{b}'_m$ is the hard-decision binary bit, $\hat{b}'_m \in \{0,1\}$, according to $\hat{r}'_m$.

Referring to (7.7), the magnitudes of $\check{L}_m$ and $\hat{L}_m$ can be significantly different. The factor of $\check{L}^e_m$ for $\check{L}_m$ is ranged in the entire real field $\mathbb{R}$. On the other hand, the factor of $\hat{L}^e_m$ for $\hat{L}_m$ is only ranged in either $-2(L^c_m + L^a_m)$ or 0, which just changes the sign of the reliability. Thus the conditioned iterative output not only ensures the constancy of maximum attainable cross-correlation, it also provides an initial tight range for the OSD-i with the same MRB to help reduce the codeword search size and computational complexity. Furthermore, the iterative output help reduce the number of errors in MRB part, which also could shorten the codeword search size.

The frequency distributions of the number of bit errors in the MRB as a result of iterative decoding by using conditioned output after different iterations are shown in Figure 7.3. These were obtained by evaluating 10^6 received vectors for Tanner codes $(155, 64, 20)$ [118] at 4dB $\frac{E_b}{N_o}$. As the iteration is increased, the number of blocks with null error in MRB is increased. The number of frames with same number of errors in MRB is reduced as the decoding iterates. Furthermore, as the number of iterations increased, the number of frames with larger number of errors in MRB progressively decreases. Especially, OSD-5 is required to get error free after 1 iteration, on the other hand, OSD-4 is rarely required after 10 more iterations. It is a good evidence that the conditioned output helps to reduce the number of frames with larger number of errors in MRB.

The comparisons between soft output and conditioned output from BP decoder for number of blocks with different number of errors in MRB are shown in Figure 7.4 with different iterations. The simulated code is Tanner code $(155, 64, 20)$ at Eb/No=4dB. First of all, the iterative decoder helps significantly reduce the

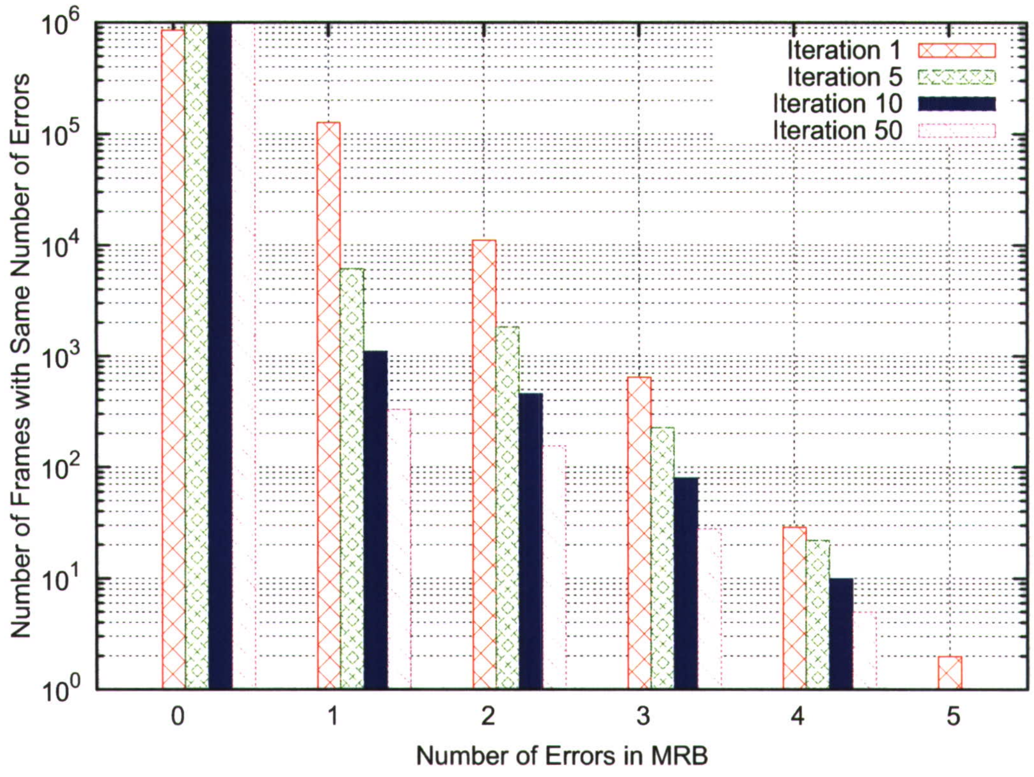


Figure 7.3: Comparison of Conditioned Output for Tanner Codes $(155, 64, 20)$ at Eb/No=4dB with Different Iterations

number of errors among the MRB as the iterations are increased. And the number of blocks with conditioned output is reduced in the number of errors in MRB as the iterations are increased. On the other hand, the number of errors in MRB for soft output is spread and increased to maximum 7.

The frequency distributions of the number of bit errors in the MRB as a result of iterative decoding, conditioned iterative decoding and no iterative decoding are shown in Figure 7.5. These were obtained by evaluating 10^6 received vectors for the *Euclidean geometry* (EG) LDPC codes $(255, 175, 17)$ [119] at 4.5dB $\frac{E_b}{N_o}$ with 50 iterations of the iterative decoder. The number of errors j in the MRB dictates the order i of the OSD-i that is necessary for successful decoding. If j is greater than i, then decoding errors are certain. Thus Figure 7.5 may be used to estimate the probability of decoding error for the different decoding arrangements. It can be seen that the worst input is the output from the iterative decoder with the

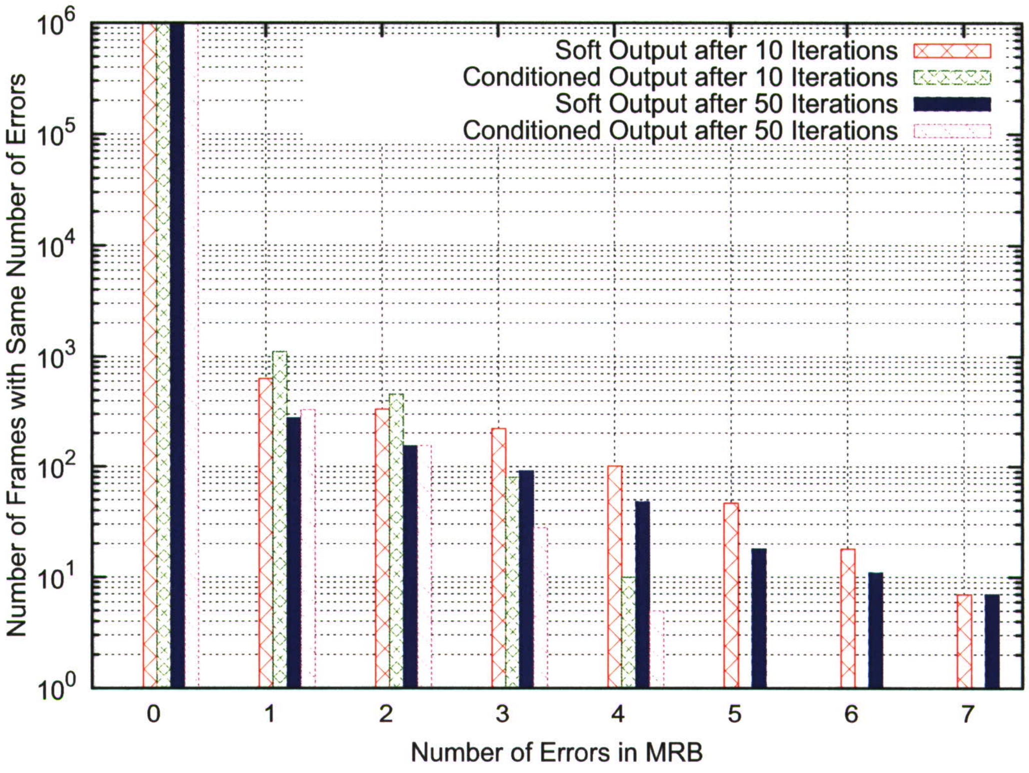


Figure 7.4: Comparison between Soft Output and Conditioned Output for Tanner Codes $(155, 64, 20)$ at Eb/No=4dB with Different Iterations

highest number of errors in the MRB from 4 to 8. The best input with the smallest number of errors in the MRB is the conditioned output from the iterative decoder. There is another benefit of using the conditioned output in that the number of blocks decreases faster as the number of errors in the MRB increases, in comparison to the other inputs. It obviously provides a better convergent performance in terms of decoding computational complexity and performance, which means more error-blocks could be successfully decoded by using a smaller OSD-i decoder with less search size.

7.4 Numerical Results

In this section, different decoding arrangements for some well known LDPC codes are presented. For all the related iterative decoding, the maximum number of

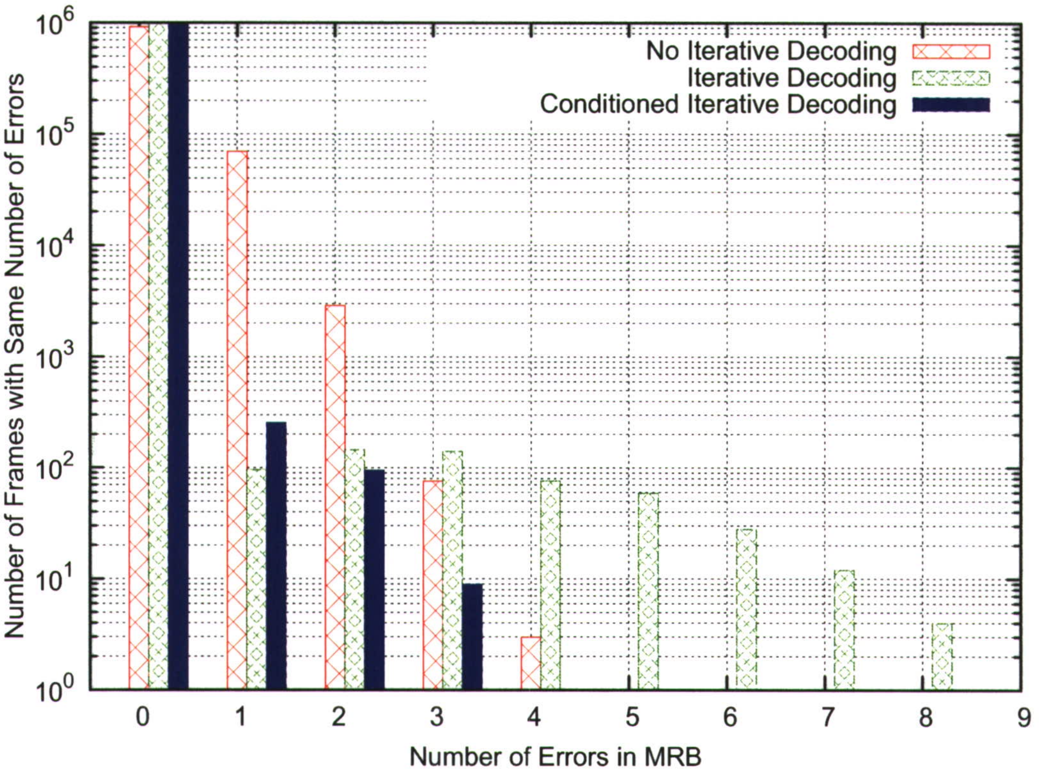


Figure 7.5: Comparisons between Different Inputs to OSD-i for EG LDPC Codes $(255, 175, 17)$ at Eb/No=4.5dB, Iteration=50

iterations is set as 50. The BP decoder paired with OSD-i decoder and the proposed decoder paired with OSD-i decoder and OSD-i decoder alone are separately illustrated. The partial stopping sets distribution for the well known LDPC codes in the following section could be referred to Table 8.2 and Table 8.3.

7.4.1 Well Known LDPC Codes

The results achieved by different decoding arrangements for Tanner codes $(155, 64, 20)$ [118] are shown in Figure 7.6 and Figure 7.7. Due to the good d_{min} and small multiplicity of low weight codewords for Tanner codes, BP decoding performs quite well with more than 1dB coding gain compared to the OSD-1 decoder. The proposed decoder using OSD-i achieves significant improvements in performance than the OSD-i decoder alone and has a lower error floor than the BP decoder with OSD-i decoding. There nearly exists a 2dB coding gain between

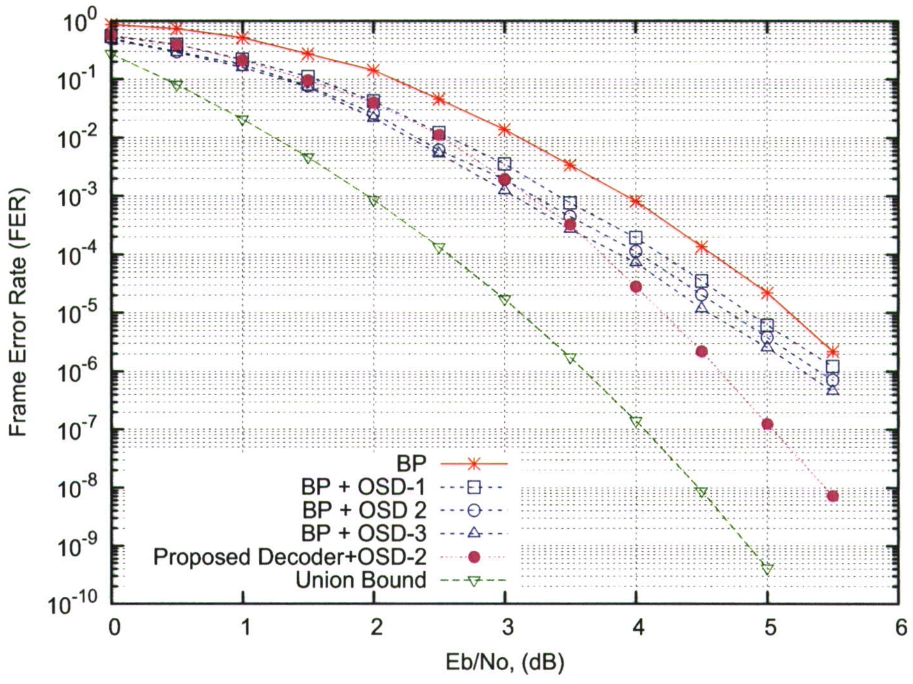


Figure 7.6: Comparison Results for Tanner Codes $(155, 64, 20)$ (Part 1)

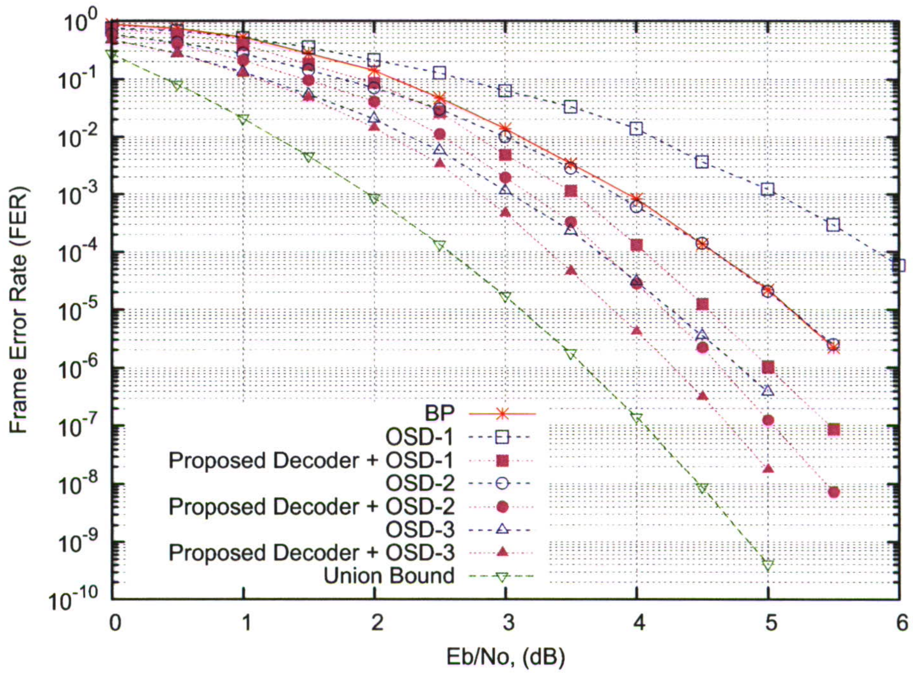


Figure 7.7: Comparison Results for Tanner Codes $(155, 64, 20)$ (Part 2)

the proposed decoder with OSD-1 and the OSD-1 decoder alone.

The results achieved by the different decoding arrangements for regular Gallager codes $(204, 102, 8)$ [80] are shown in Figure 7.8 and Figure 7.9. Although

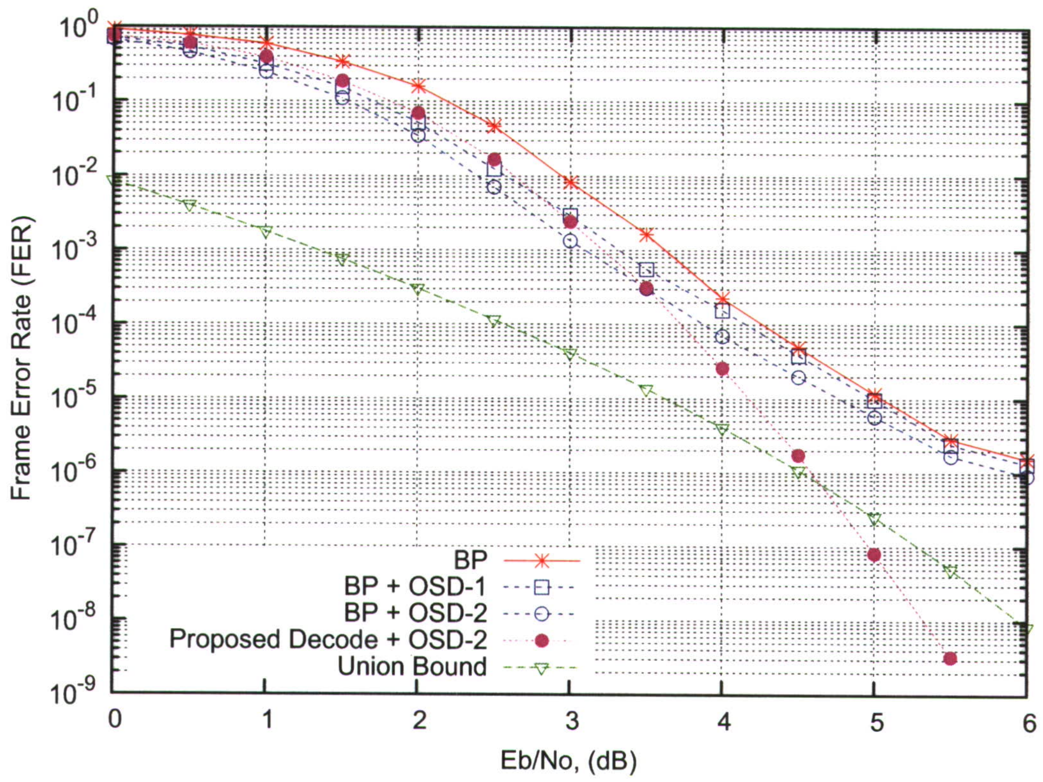


Figure 7.8: Comparison Results for Regular Gallager Codes $(204, 102, 8)$ (Part 1)

the simulated code has a small d_{min}, it has totally 50 multiplicity in low weight codewords up to hamming weight 15. Thus BP decoding performs quite well with more than 1.5dB coding gain compared to the OSD-1 decoder and better performance than the OSD-2 decoder before the error floor region. The proposed decoder using OSD-1 achieves significant improvement in performance than the OSD-1 decoder alone and guarantees the free error-floor performance than BP decoder and BP decoder with OSD-2 decoding. There exists more than 1.5dB coding gain between the proposed decoder with OSD-1 and the OSD-1 decoder alone since 3dB $\frac{E_b}{N_o}$.

The results achieved by different decoding arrangements for regular Gallager codes $(200, 100, 9)$ are shown in Figure 7.10 and Figure 7.11. Such a code has

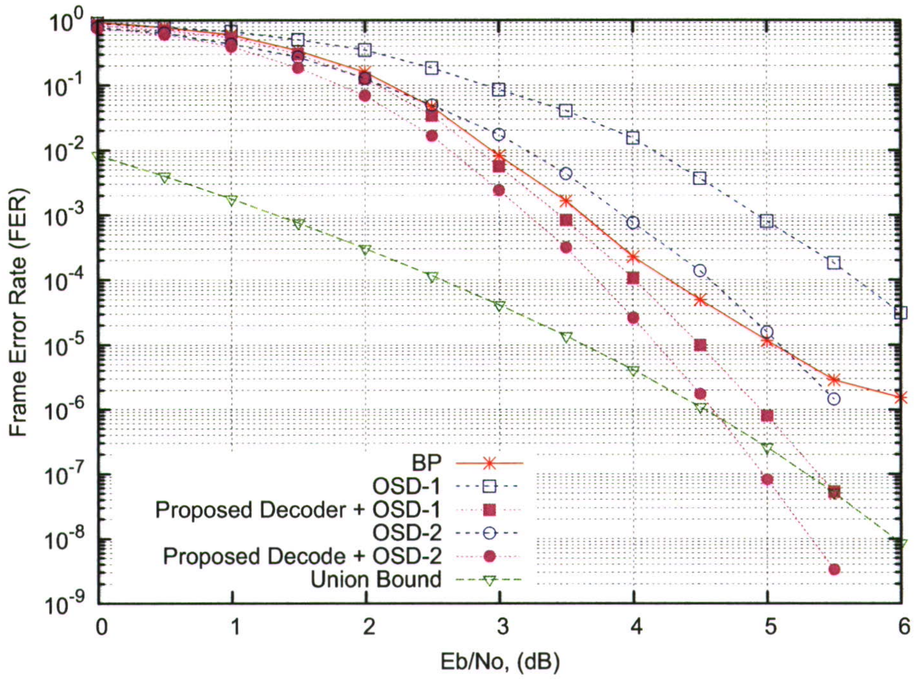


Figure 7.9: Comparison Results for Regular Gallager Codes $(204, 102, 8)$ (Part 2)

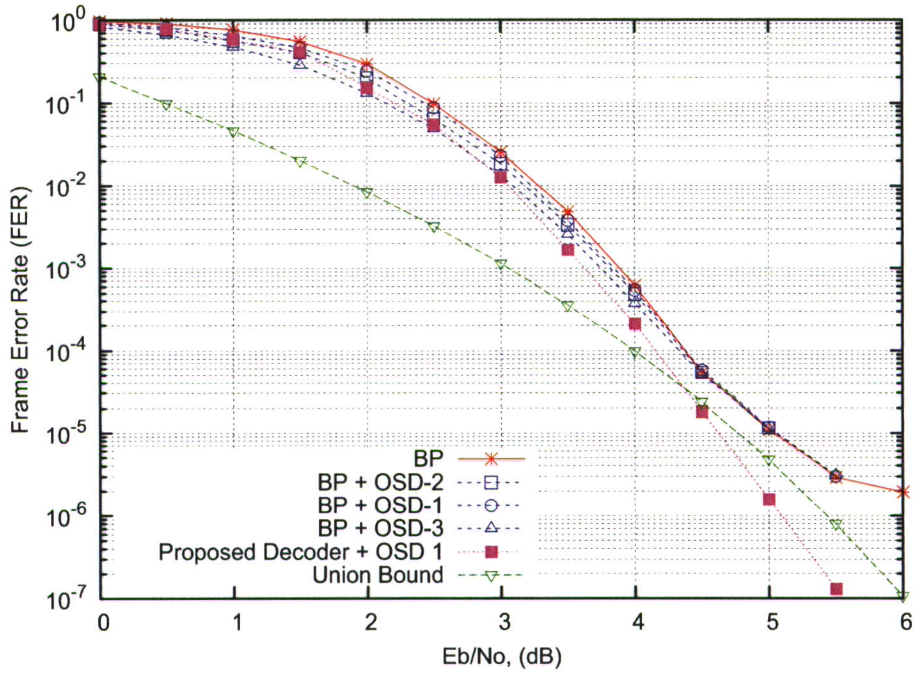


Figure 7.10: Comparison Results for Regular Gallager Codes $(200, 100, 9)$ (Part 1)

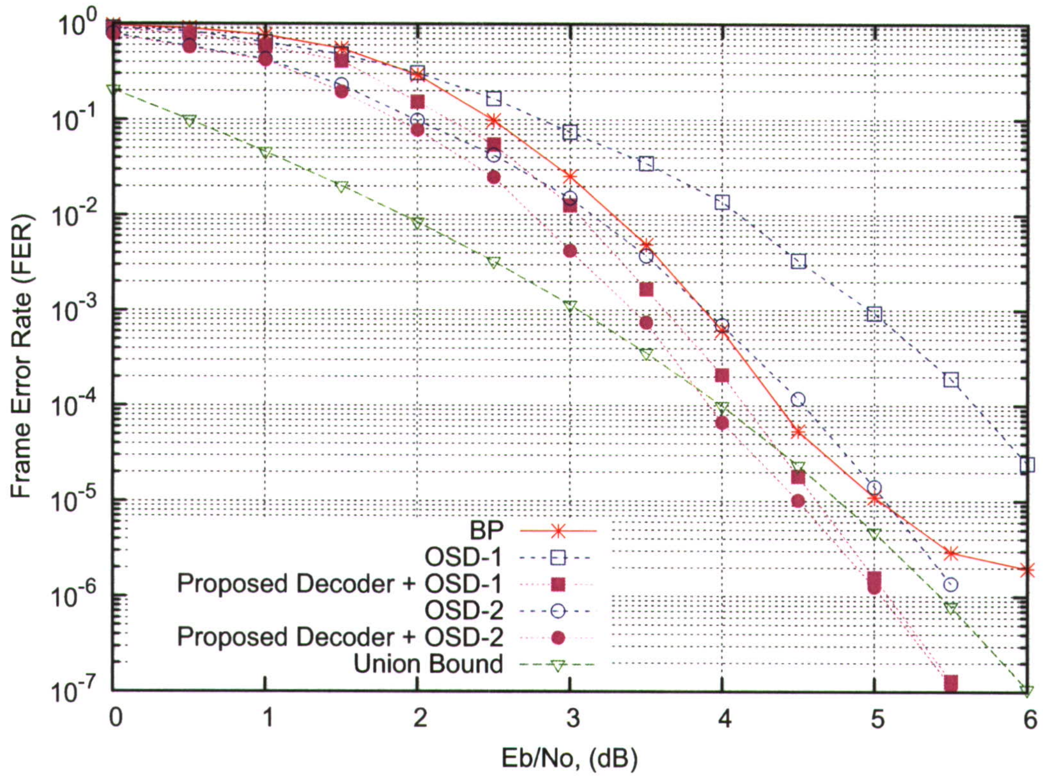


Figure 7.11: Comparison Results for Regular LDPC Codes (200, 100, 9) (Part 2)

similar d_{min} of 9 as MacKay's LDPC codes (204, 102, 8), but the high multiplicity of its low weight codewords and stopping sets causes the BP decoder performing similar as OSD-2 decoding performance, but start to reach the error-floor region since 5db $\frac{E_b}{N_o}$. But BP decoding still achieves nearly 1.5dB coding gain at 4.5dB $\frac{E_b}{N_o}$ than OSD-1. Furthermore, the output, which has been terribly affected, causes the OSD-i to have the identical performance with the BP decoder. The proposed decoder using OSD-1 achieves significant improvement in performance over the OSD-1 alone and guarantees the free error-floor performance than BP decoder and BP decoder with OSD-2 decoding. There exists more than a 1dB coding gain between the proposed decoder with OSD-1 and the OSD-1 decoder alone since 3dB $\frac{E_b}{N_o}$.

The *progressive edge-growth* (PEG) irregular LDPC code proposed by Hu [58], provides a better performance with the similar codeword lengths due to its strong d_{min}. The results for the PEG LDPC codes (256, 128, 17) are shown in Figure 7.12

and Figure 7.13. The BP decoder achieves about 1dB coding gain at high SNR

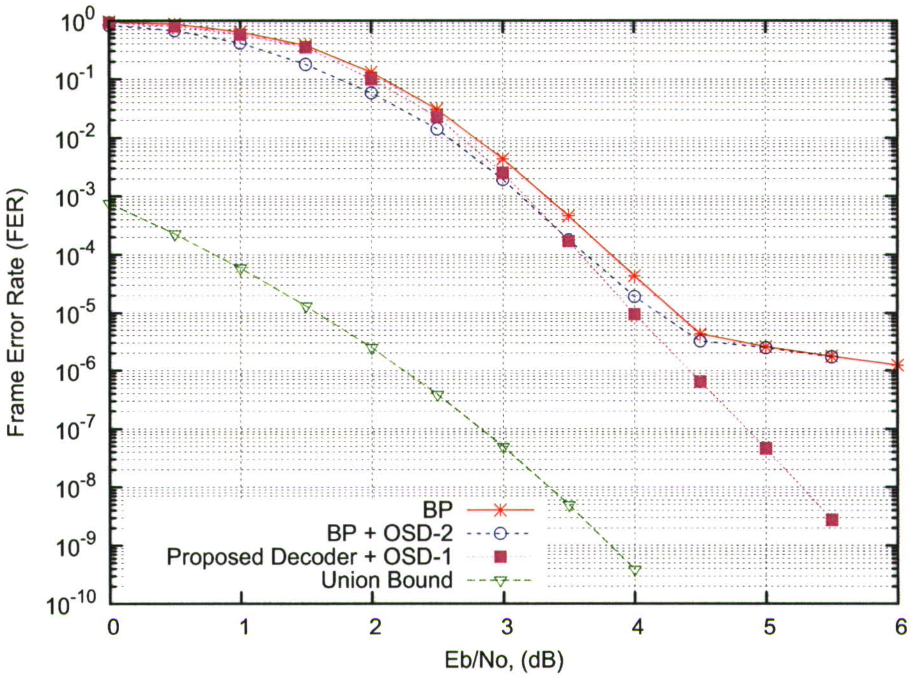


Figure 7.12: Comparison Results for PEG LDPC Codes $(256, 128, 17)$ (Part 1)

from 3.5dB $\frac{E_b}{N_o}$ than the OSD-2 decoder, but degrades due to the stopping sets from 4.5dB $\frac{E_b}{N_o}$, where the error floor occurs. The proposed decoder paired with OSD-1 produces significantly better performance than the OSD-2 decoder at any SNR and successfully breaks the error floor caused by the BP decoder and BP decoder paired with OSD-2.

7.4.2 Cyclic LDPC Codes

The stopping sets distributions for the cyclic LDPC codes constructed based on [119] are shown in Table 7.1

LDPC code	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
EG LDPC $(255, 175)$ [119]	8	2 (0)	0 (0)	0 (0)	5 (0)	52 (0)
EG LDPC $(273, 191)$ [119]	11	4 (0)	30 (0)	444 (0)	6537 (0)	129402 (0)
PG LDPC $(341, 205)$ [119]	15	4 (0)	− (−)	− (−)	− (−)	− (−)

Table 7.1: Stopping Set Distribution up to $s_{min} + 4$

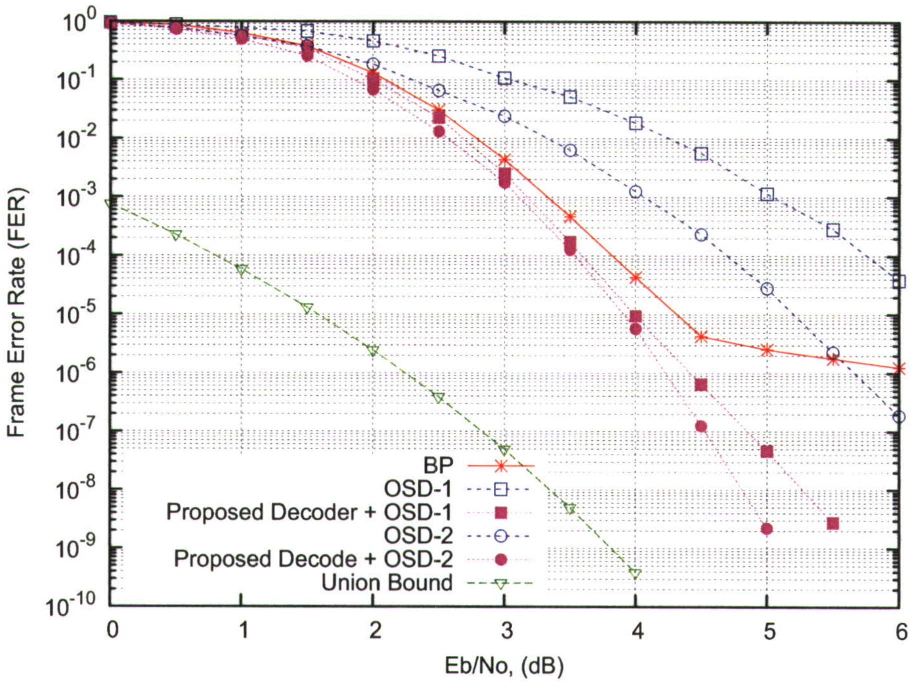


Figure 7.13: Comparison Results for PEG LDPC Codes (256, 128, 17) (Part 2)

The finite geometry codes make good LDPC codes using belief propagation decoding [62]. The results for the EG LDPC codes (255, 175, 17) based on cyclotomic idempotents are shown in Figure 7.14 and Figure 7.15. The BP decoder achieves about 0.5dB coding gain at high SNR from 4.5dB $\frac{E_b}{N_o}$ than the OSD-1 decoder, and the error floor occurs from 5.5dB $\frac{E_b}{N_o}$. The proposed decoder paired with OSD-1 produces significantly better performance than the OSD-2 decoder after 5dB $\frac{E_b}{N_o}$ and has a lower error floor than the BP decoder with OSD-2 and OSD-3 decoder.

The results for the *projective geometry* (PG) cyclic LDPC codes (273, 191, 18) are shown in Figure 7.16 and Figure 7.17. The iterative BP decoder achieves more than 1dB coding gain over the OSD-1 decoder, and approaches closely to the OSD-2 decoder, then the error floor occurs at 5.5dB $\frac{E_b}{N_o}$. The proposed decoder with OSD-1 performs better than all of the other decoding arrangements at $\frac{E_b}{N_o}$ values above 5dB, except for the proposed decoder with larger order OSD-i.

The results for the PG cyclic LDPC codes (341, 205, 16) are shown in Figure 7.18 and Figure 7.19. The iterative BP decoder achieves more than 1dB

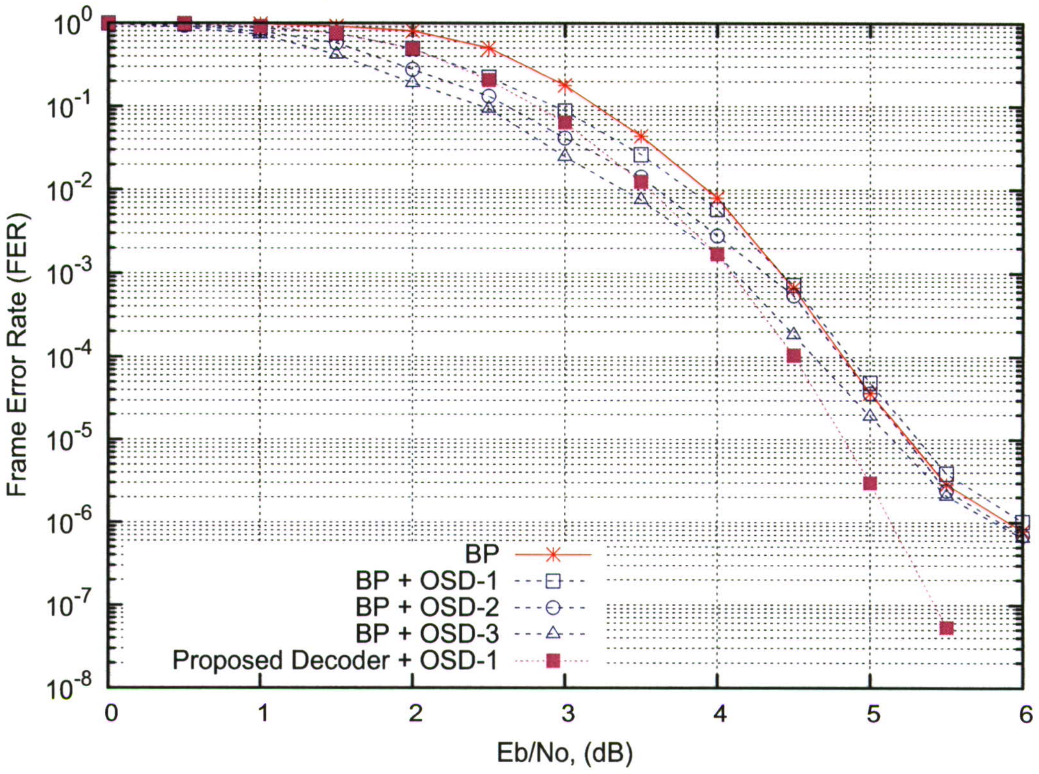


Figure 7.14: Comparison Results for EG LDPC Codes $(255, 175, 17)$ (Part 1)

coding gain than the OSD-1 decoder at 4.5dB $\frac{E_b}{N_o}$, and closely approaches the OSD-2 decoder at 5dB $\frac{E_b}{N_o}$. At $\frac{E_b}{N_o}$ value of 5.5dB, there is an evidence of an error floor using BP decoding. The BP decoder with OSD-3 decoder also exhibits an error floor. The proposed decoder with OSD-1 performs better than most of the other decoding arrangements at $\frac{E_b}{N_o}$ values above 4dB.

Those results provide good evidence that the proposed decoder with OSD-i achieves better performance than OSD-i decoding alone and shows no sign of an error floor unlike BP decoding coupled with OSD-i decoding or stand-alone BP decoding. The proposed decoder with OSD-i crosses the OSD-$(i+1)$ decoding performance at high SNR and is much less decoding complex.

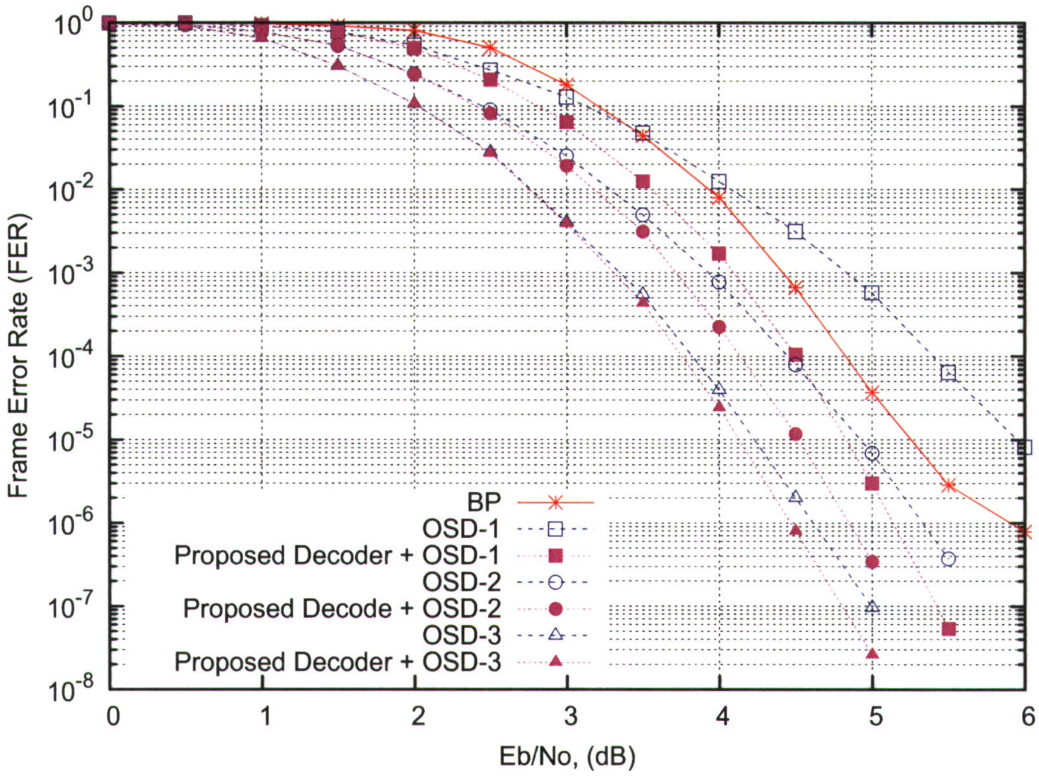


Figure 7.15: Comparison Results for EG LDPC Codes (255, 175, 17) (Part 2)

7.4.3 Simulation Remark on Cyclic LDPC Codes

For those cyclic codes, whose parity-check matrix could have n parity-check equations, which includes $(n-k)$ independent parity-check equations and k dependent parity-check equations. The BP decoding algorithm could evaluate their received vector through the entire n parity-check equations instead of $n-k$ at each iteration. Then the minimum stopping set weight might be improved due to the additional equations' support. Thus the performance of cyclic LDPC codes decoded by BP decoder could make a significant improvement.

The improved results for the EG cyclic (255, 175, 17) LDPC codes by following the revised decoding process are depicted in Figure 7.20. It is clear that the revised BP decoding performance approaches between OSD-2 and OSD-3 rather than the standard BP performance between OSD-1 and OSD-2 as shown in Figure 7.15. But the error-floor occurs at 5dB $\frac{E_b}{N_o}$, then the increase in performance flattens out. Moreover, the proposed decoder on n parity-check equations paired

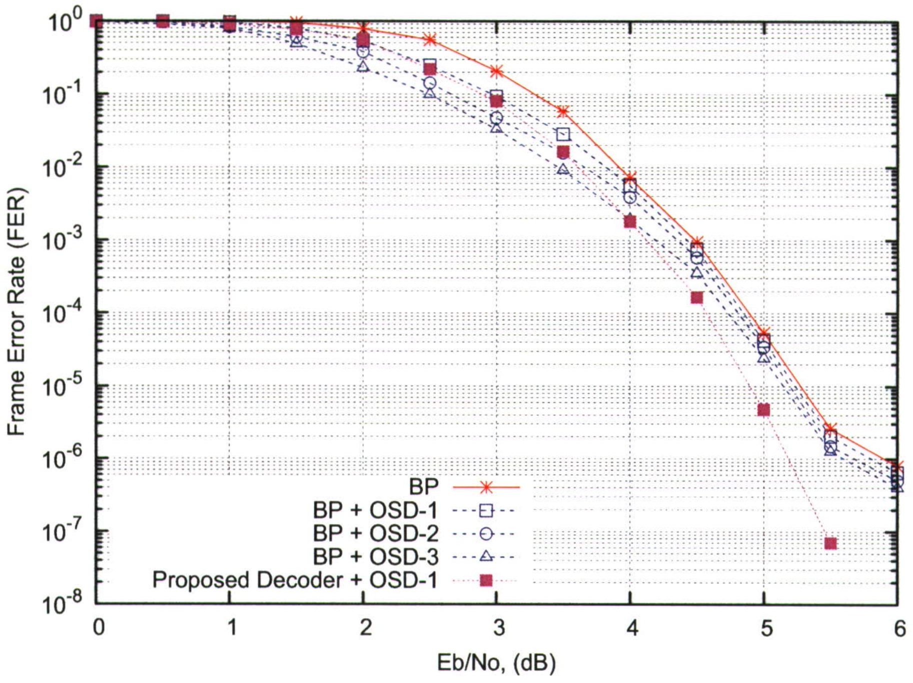


Figure 7.16: Comparison Results for PG LDPC Codes (273, 191, 18) (Part 1)

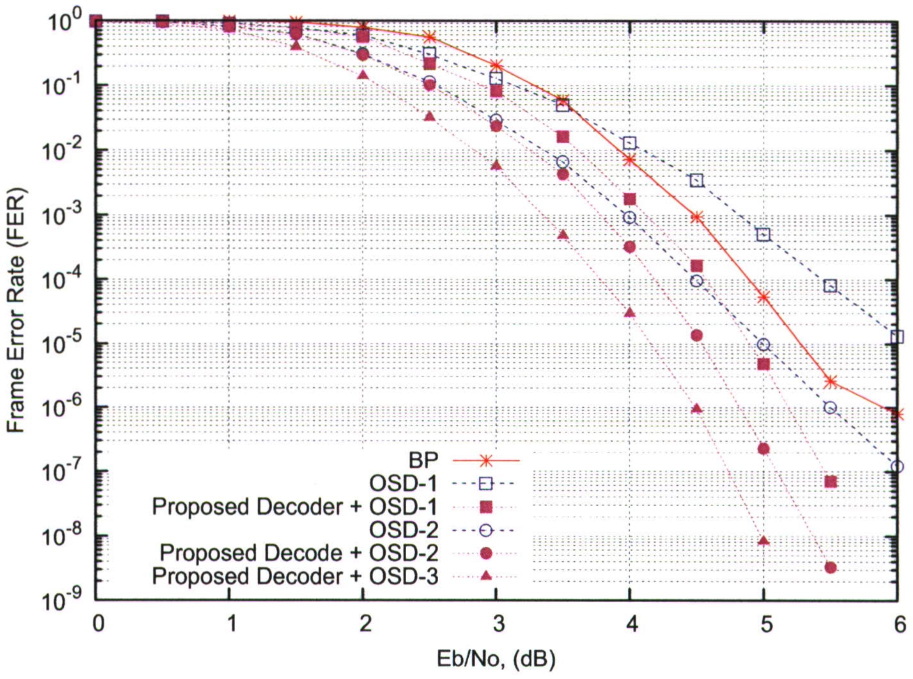


Figure 7.17: Comparison Results for PG LDPC Codes (273, 191, 18) (Part 2)

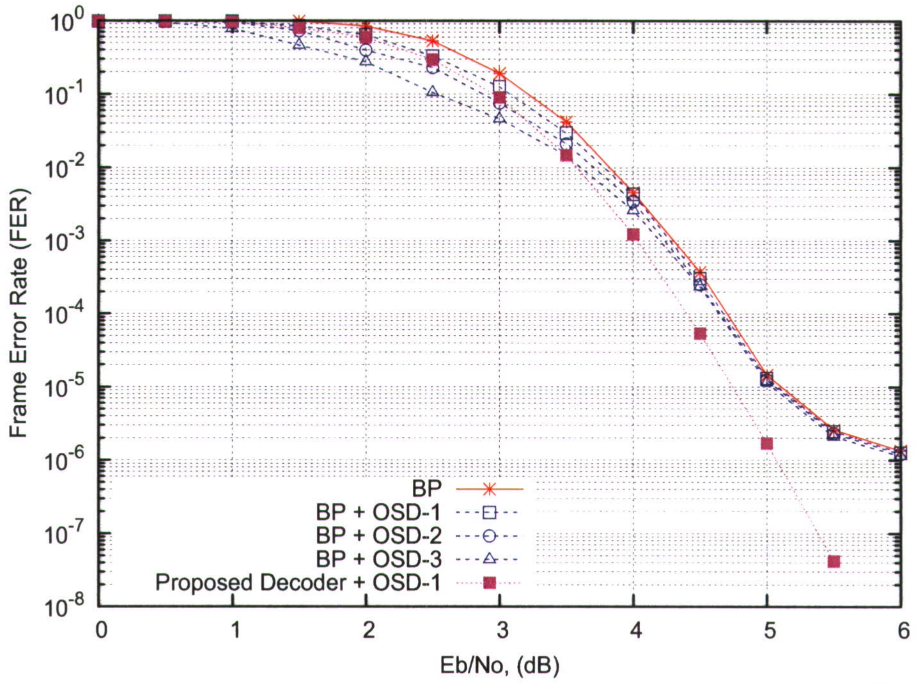


Figure 7.18: Comparison Results for PG LDPC Codes (341, 205, 16) (Part 1)

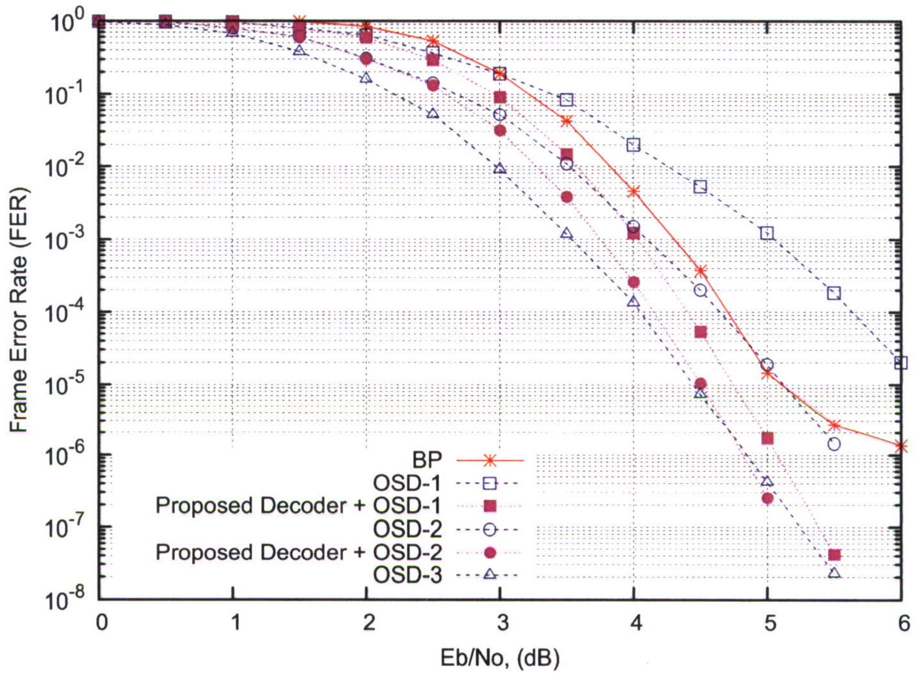


Figure 7.19: Comparison Results for PG LDPC Codes (341, 205, 16) (Part 2)

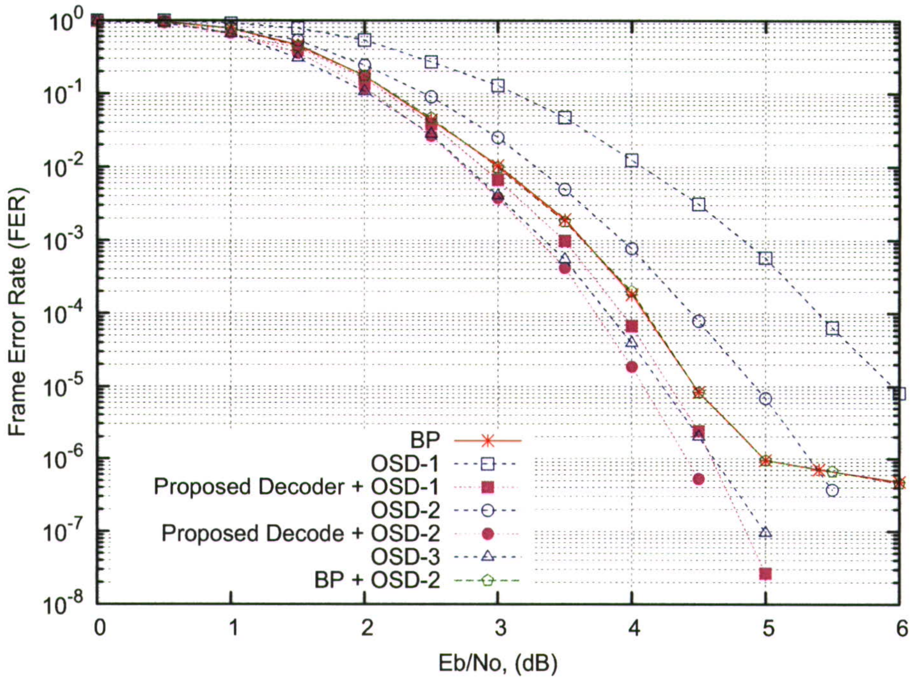


Figure 7.20: Comparison Results for EG LDPC Codes (255, 175, 17) (Revised)

with the OSD-1 decoder makes an increasingly obvious improvement as the SNR increases, and nearly approaches OSD-3 decoding performance at 4.5db $\frac{E_b}{N_o}$. On the other hand, the BP with OSD-i by exploring n parity-check equations seems have more trouble on handling the soft output, resulting in nearly the identical BP decoding performance.

The remarked results for the EG cyclic (273, 191, 18) LDPC codes are depicted in Figure 7.21. Since the computational complexity could be exponentially increased by the higher order number i, thus for this code, it involves $\binom{191}{3}$ searching nodes, which might be achieved by time-polynomial consumption. Then the proposed decoder with OSD-1 actually provides a reduced complexity algorithm to help achieve the similar results, which is a definite improvement over OSD-2 due to the excellent BP decoding performance on n parity-check equations. And the BP paired with OSD-i still suffers by the highly scaled magnitudes in MRB, which hardly makes any improvement based on BP decoding output. Furthermore, the proposed decoder with OSD-1 successfully breaks the error-floor issue as BP decoder has beyond 4.5dB $\frac{E_b}{N_o}$.

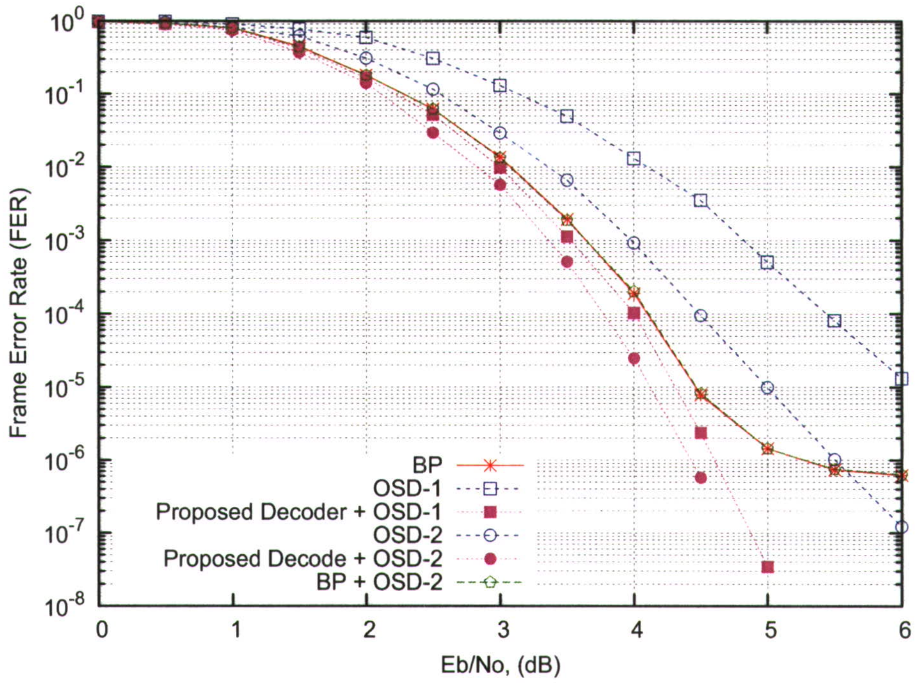


Figure 7.21: Comparison Results for EG LDPC Codes $(273, 191, 18)$ (Revised)

The other revised results for the PG cyclic $(341, 205, 16)$ LDPC codes are depicted in Figure 7.22. First of all, the proposed decoder with OSD-i outperform than OSD-3 since 3.5dB $\frac{E_b}{N_o}$. The BP decoder nearly achieves the similar results as OSD-3 does, although such code has smaller d_{min} than other two cyclic codes. The error-floor still occurs on BP decoder after 5dB $\frac{E_b}{N_o}$, which could be simply solved by the proposed decoder with OSD-1.

7.5 Summary

In this chapter, a new decoding arrangement for linear block codes with sparse parity-check matrices over the AWGN channel has been introduced, which uses a conditioned output from a BP decoder instead of the standard soft output. Simulation Results have been presented that the proposed decoder using OSD-1 completely solves the error floor problem associated with BP decoding of LDPC codes. Also, compared to stand-alone OSD-i decoding, the proposed decoder using OSD-$(i-1)$ decoding performs better at high SNR and involves much less

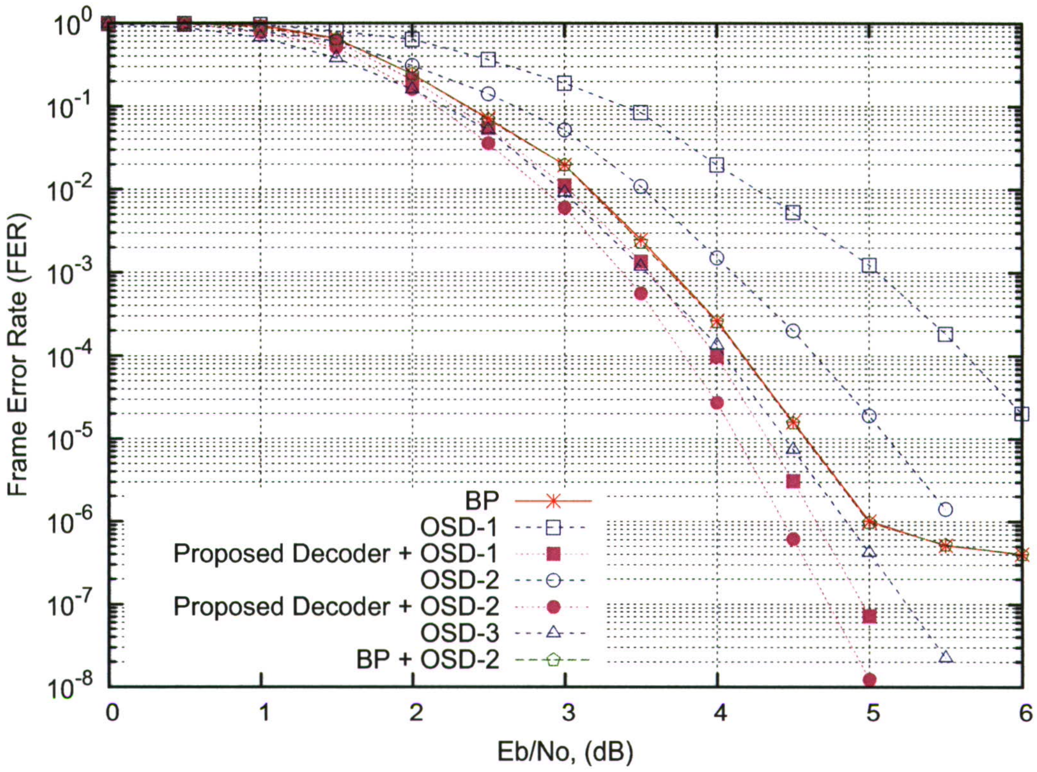


Figure 7.22: Comparison Results for PG LDPC Codes $(341, 205, 16)$ (Revised)

decoder complexity.

Part III

Exhaustive Tree Search in Code Spectra and Decoding

Chapter 8

Tree based Exhaustive Search

8.1 Background

For a linear block code, one of the fundamental properties is the *minimum distance*, denoted as d_{min}. The *minimum distance* generally decides the codes capability of detecting or correcting a number of errors. For instance, the minimum distance has a major impact on the decoding starting from moderate SNR. There is another parameter which is based on the d_{min} and helps define a further characteristic of the codes, it is the so-called *weight enumerators* or *weight spectrum.* Let $\mathcal{C}$ be a linear code with length n, and A_i be the number of codewords in weight of i. Then the *weight enumerator* of $\mathcal{C}$ is defined as

$$\mathcal{W}_{\mathcal{C}}(z) = \sum_{i=0}^{n} A_i z^i \tag{8.1}$$

where z^i represents the number of distinct non-zero distances between codewords, and the sequence of $\{A_0, A_1, ..., A_n\}$ is referred to as the *weight distribution* of $\mathcal{C}$. For codeword's spectrum, d_{min} is referred to be the smallest number of z_i with distinct non-zero distances between codewords.

In terms of code performance evaluation, the *minimum distance* might be the first step to help know the property of the code. This could be one of the reasons that it has attracted a great amount of interest in coding theory. In 1978, the problem of computing the minimum distance of an arbitrary binary linear code was conjectured to be NP-hard by Berlekamp *et al.* [8] due to the existence of polynomial-time consuming algorithm. Such conjecture as a mystic puzzle was decrypted in an affirmative way by Vardy [124] in 1997. Since then, different effective methods have been introduced to find the small size of codewords for a binary linear code. The probabilistic algorithm was introduced by Leon [68] and Stern [115]. Based on Stern's work, Canteaut & Chabaud [17] proposed an efficient algorithm by operating on the subspace of the parity-check matrix instead of the entire matrix. Then a specific probabilistic based algorithm to estimate the small weight distribution for LDPC codes was proposed by Hirotomo *et al.* [52]. Meanwhile, another approach to compute the minimum distance of linear block codes called *error impulse* (EI) was introduced by Berrou *et al.* [12]. By applying the EI technique, an effective modification algorithm for LDPC codes was proposed by Daneshgaran *et al.* [31]. Furthermore Hu *et al.* [57] proposed a new approach called the *nearest non-zero codeword search* (NNCS) by combining the idea of EI method and the reliability-based list decoding from [42].

The iterative decoding performance for LDPC codes is constrained by the small size of stopping sets, especially the minimal stopping sets cause the error floor occurring at early stage, which is the moderate SNR in the AWGN channel or the high erasure probability ϵ over the erasure channel. The problem of finding the minimum size of stopping sets was focused on as an interesting topic. Thus we have a new notable symbol s_{min}, which represents the minimum distance of distinct stopping sets. Let $\mathcal{S}$ be a stopping set with length of n, and B_i be the number of stopping sets in weight of i. Then the corresponding *weight enumerator* of $\mathcal{S}$ is designated

$$\mathcal{W}_{\mathcal{S}}(s) = \sum_{i=0}^{n} B_i s^i \tag{8.2}$$

where s^i represents the number of distinct non-zero distances between stopping sets. The stopping sets' weight distribution sequence starts from s_{min} to help analyse the stopping sets impact over the corresponding iterative decoding performance; for instance, the BP stopping sets of LDPC codes to the BP iterative decoding performance, which could help explain the occurrence of error floor caused by stopping sets. On the other hand, the codeword weight enumerator (8.1) could be used in bound estimation, like *Union Bound*, to get the optimum error performance over the AWGN channel or the erasure channel. The problem of computing the minimum size of stopping sets for an arbitrary Tanner graph of LDPC codes was also proved to be NP-hard by Krishnan & Shankar [63]. But based on the knowledge of computing the minimum distance of LDPC codes, some efficient algorithms for finding the minimum-size of stopping sets for LDPC codes were introduced by applying different approaches. An EI based algorithm to find the small size stopping sets was proposed by Richter [96]. Hirotomo *et al.* [53] proposed a probabilistic based algorithm to find the minimum-size stopping sets of LDPC codes. On the other hand, Wang *et al.* [127] proposed the first exhaustive search based algorithm to find all the small size error-patterns, which include codewords, stopping sets and k-out trapping sets, for LDPC codes. Furthermore, based on Rosnes' previous work [100, 101] of exhaustive search on turbo codes stopping sets, he [102] proposed an efficient algorithm to find all the stopping sets and codewords by a given threshold for LDPC codes. We investigate the idea of considering qualified active rows in the tree representation structure to propose our optimised algorithm by using a reduced complexity based upper bound to achieve the same goal of finding all the stopping sets and codewords in small size for LDPC codes. This chapter describes the detail of the related algorithms and explanations. The code representation and related definitions are described in Section 8.2. The tree search algorithm and the simple bound algorithm are explained in Section 8.3. A series of weight spectrum results about some well known LDPC codes and WiMax LDPC codes are illustrated in Section 8.4. The proposed search algorithm and the corresponding bound were presented in [2].

8.2 Preliminaries

8.2.1 Code Representation

Let $\mathcal{C}$ be a binary linear block code with length n, $n \in \mathbb{N}$, and k-dimensional subspace of $\{0,1\}^n$ be the information bits with length k. The linear code $\mathcal{C}$ can be represented by a $m \times n$ binary parity-check matrix $\mathbf{H}$, where $m \geq n - k$. Let $\mathbf{x} = \{x_0, ..., x_{n-1}\} \in \{0,1\}^n$ be the transmitted vector, then bits x have to be satisfied with $\mathbf{H} \cdot \mathbf{x} = 0$. In equivalent graphical representation, the code $\mathcal{C}$ is represented by a bipartite graph, called a Tanner graph [117]. A bipartite graph comprises a vertex set of *variable nodes* $\mathbf{V} = \{v_0, ..., v_{n-1}\}$, and a vertex set of *check nodes* $\mathbf{C} = \{c_0, ..., c_{m-1}\}$. The variable nodes correspond to the columns of $\mathbf{H}$, and the check nodes correspond to the rows of $\mathbf{H}$. H_{ji} represents the connection relationship between the variable node v_i and check node c_j. It is defined that there exists a connected *edge* between variable node v_i and check node c_j, if $H_{ji} = 1$.

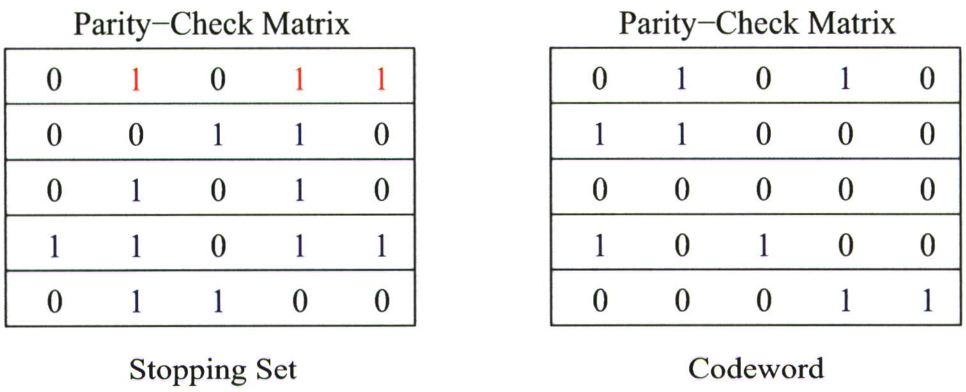
For LDPC codes, the degree sequences of columns and rows of $\mathbf{H}$ specify the property of the codes. Let λ_i be the degree of variable nodes, λ_{d_v} is the maximal degree of variable nodes, ρ_j be the degree of check nodes and ρ_{d_c} is the maximal degree of check nodes. If λ_i is a fixed value with $i = d_v$, and ρ_j is a fixed value with $j = d_c$, then the codes are said to be regular (d_v, d_c) LDPC codes, otherwise they are irregular LDPC codes.

8.2.2 Codeword Set and Stopping Set

- *Codeword Set:* A codeword set is a subset of $\{x_0, ..., x_{n-1}\}$ with Hamming weight d. Since $\mathbf{Hx} = 0$ for all valid codeword, such that the set is said to be a codeword if, and only if, the induced subgraph contains no check node with odd degree. The minimal Hamming distance d_{min} is defined as the minimal weight of the non-empty codeword sets.

- *Stopping Set:* A stopping set is a subset of $\{b_0, ..., b_{n-1}\}$ with Hamming weight s, where $\mathbf{b} = \{b_0, ..., b_{n-1}\} \in \{0,1\}^n$. A set is defined as a stopping set, if there exists no check node with degree one on the induced subgraph. The minimal stopping distance s_{min} is defined as the minimal size of the non-empty stopping sets. By definition, a valid codeword set is also a stopping set, but a stopping set might not be a valid codeword set, thus $s_{min} \leq d_{min}$ is applied for all parity-check codes. By adding redundant independent parity-check equations to the parity-check matrix $\mathbf{H}$, it might help improve the code structure. Thus some of the small size stopping sets become valid codeword sets and the minimal stopping distance gradually increases or reaches the minimal Hamming distance [107]. For instance, by adding two more redundant parity-check equations, the minimal stopping distance of the Tanner code $(155, 64)$ [118] is improved from 12 to 18.

The comparison example between a codeword set and a stopping set over the parity-check matrix is shown in Figure 8.1. The left figure of a stopping set shows



0	1	0	1	1
0	0	1	1	0
0	1	0	1	0
1	1	0	1	1
0	1	1	0	0

0	1	0	1	0
1	1	0	0	0
0	0	0	0	0
1	0	1	0	0
0	0	0	1	1

Figure 8.1: Comparison of Codeword Set and Stopping Set

that there exists at least one check node with odd degree excluding one on the induced parity-check matrix. Meanwhile, the right figure of a codeword set shows that there only exist check nodes with even degree including zero. According to the definition of a stopping set and a codeword set, the codeword set could be viewed as a subset of the stopping set, which only contains check nodes with even degree.

8.3 Tree-Search based Stopping Set Enumeration Algorithm

8.3.1 Bounded Tree Search

A binary linear code could be viewed as a tree starting from the first index bit with branches of value “0” and “1”. Thus a tree is capable of displaying the entire combinations of the code up to length n, as resulting in 2^n combinations.

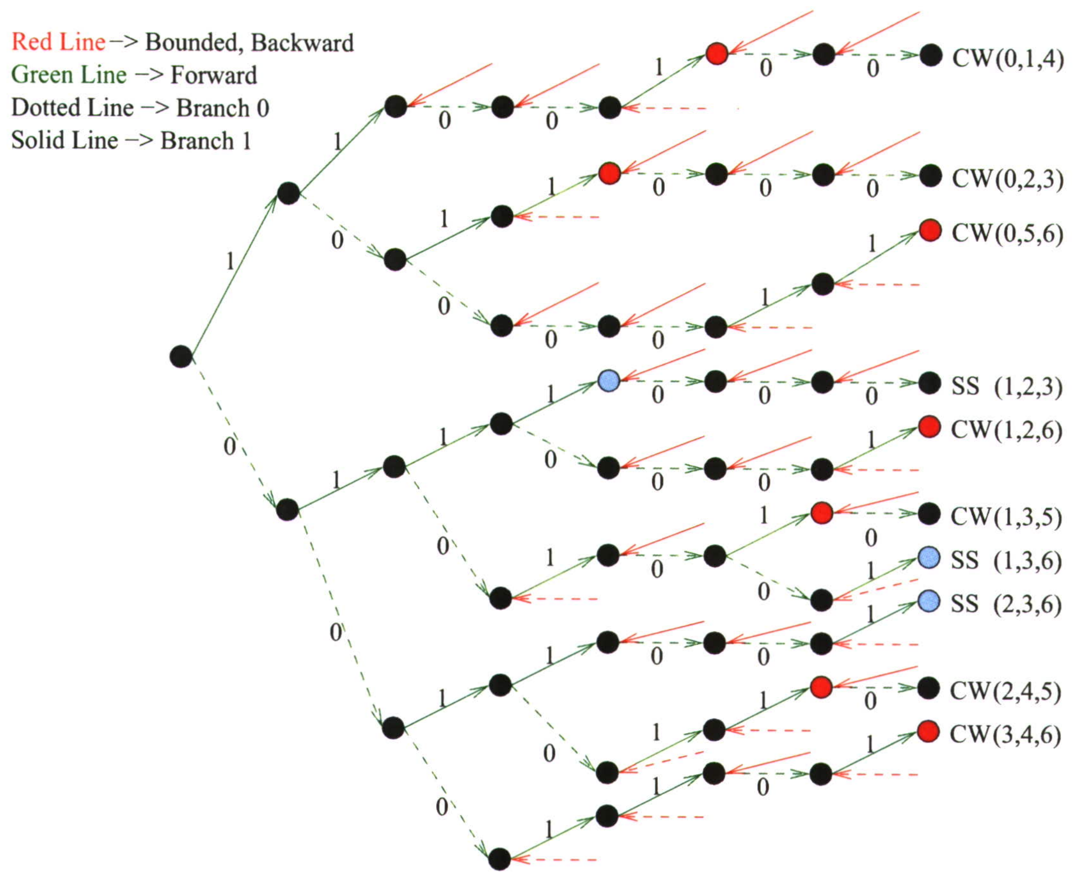


Figure 8.2: Bounded Tree Search Example

By following the different routes of branches, some of the routes indicate the valid codewords or stopping sets. The search complexity on the valid branches exponentially increases with the increment of the code length. Thus the bounded tree search algorithm is required to help constrain the set of searching branches by a certain lower bound. The lower bound is to detect which branches are capable of passing the threshold. Here an example is given of a bounded tree search for Hamming codes $(7, 4)$ with threshold 3 to find all the codewords and stopping sets. As the example is shown in Figure 8.2, each connected line from left to right through eight nodes represents a codeword or stopping sets. The red node is the node where the bound helps determine a codeword; the blue node is the node which indicates a stopping set based on the bound. It is obviously shown that the actual number of combinations, 10, which has been gone through, is much smaller than 2^7, 128.

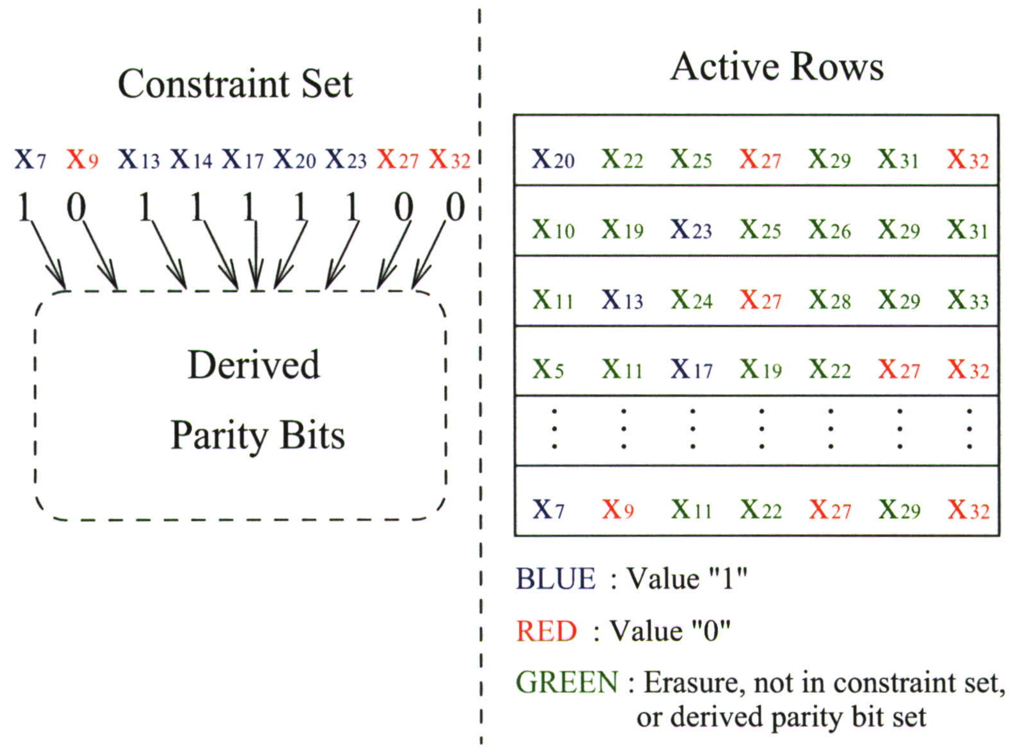

Constraint Set
X7 X9 X13 X14 X17 X20 X23 X27 X32
1 0 1 1 1 1 1 0 0
Derived Parity Bits
Active Rows
X20 X22 X25 X27 X29 X31 X32
X10 X19 X23 X25 X26 X29 X31
X11 X13 X24 X27 X28 X29 X33
X5 X11 X17 X19 X22 X27 X32
X7 X9 X11 X22 X27 X29 X32
BLUE : Value "1"
RED : Value "0"
GREEN : Erasure, not in constraint set, or derived parity bit set

Figure 8.3: Snapshot of Bounded Tree Search

The Figure 8.3 shows a snapshot of the bounded tree search. The left of the figure is a constraint set with values of "0" or "1", each bit in the constraint set may be linked with a set of derived parity bits according to the parity-check matrix. The right of the figure shows how the constraint set is induced with subgraph of parity-check matrix to construct the set of active rows. The involved bit x_i with *blue* represents value of "1", and colour *red* represents value of "0", colour *green* denotes bit as erasure excluded in constraint set or the derived parity bit. According to the rule of $\mathbf{Hx} = 0$, those active row structures could help determine the potential bound or the decodable bits.

8.3.2 Tree-Search based Stopping Sets Enumeration

Inspired by the SSE algorithm from Rosnes & Ytrehus [102] and the exhaustive algorithm by Wang *et al.* [127], we present an efficient algorithm, which is a tree-search based algorithm with reduced complexity in computing the lower bound, to search all the belief-propagation (BP) stopping sets and codeword sets up to size of threshold τ for any parity-check code $\mathcal{C}$.

Algorithm 1 Tree-search based Stopping Set Enumeration (TSSE)

```
repeat
  Pick one untouched branch as a constraint set $\mathcal{F}$.
  if $|\mathcal{F}| = n$ and $w(\mathcal{F}) \leq \tau$ then
    Constraint set $\mathcal{F}$ is saved, if $\mathcal{F}$ is valid
  else
    1). Pass $\mathcal{F}$ to the modified iterative decoder (*) with erasures in the unconstrained positions.
    2). Construct a new constraint set $\mathcal{F}'$ with new decoded positions, which is the extended branch.
    if $|\mathcal{F}'| = n$ and $w(\mathcal{F}') \leq \tau$ then
      Constraint set $\mathcal{F}'$ is saved, if $\mathcal{F}'$ is valid
    else if No contradiction is found in $\mathbf{H}^{(\mathcal{F}')}$, and $w'(\mathcal{F}') \leq \tau$ then
      a). Pick an unconstrained position $p$.
      b). Extending branch $\mathcal{F}'$ to position $p$ to get new branch $\mathcal{F}'' = \mathcal{F}' \bigcup \{(p,1)\}$ and branch $\mathcal{F}''' = \mathcal{F}' \bigcup \{(p,0)\}$.
    end if
  end if
until Tree has been fully explored
```

In the algorithm description, a *constraint set* $\mathcal{F}$ is used to represent the set of searched known bits of a code $\mathcal{C}$, which forms a *branch* of the tree during the tree search. $\mathcal{F}$ is a set $\{(p_i, s_{p_i}) : p_i \in \Gamma\}$, where $\Gamma \subseteq \{0, ..., n-1\}$ and $s_{p_i} \subset \{0, 1\}$. p_i is the bit position with value 0 or 1. In the parity-check matrix $\mathbf{H}$, a parity-check equation is said to be an *active* row if, and only if, the row weight is exactly one. Then the set of active rows in $\mathbf{H}$ is denoted by $\{\mathfrak{h}_0, ..., \mathfrak{h}_{\phi-1}\}$, where ϕ is the total number of active rows. A constraint set $\mathcal{F}$ with size n is said to be *valid* if, and only if, there exists no active row in $\mathbf{H}^{(\mathcal{F})}$. The pseudo-code of the algorithm to find all the stopping sets and codeword sets by threshold τ is given in Algorithm 1. When the whole tree has been searched, constrained by the lower bound, the list with the saved constraint sets with full size n is the whole set of stopping sets and codeword sets up to size of τ.

The modified iterative decoding is operated on the Tanner graph, which receives a n-bit binary input vector with erasures in some of the positions. Let $r_j(\mathcal{F})$ be the rank (ones) of row j, $j \in \{0, ..., m-1\}$ for the constrained position $\{p_i : (p_i, 1) \in \mathcal{F}\}$ intersected by row j on $\mathbf{H}$. And let $r'_j(\mathcal{F})$ be the rank of row j for the unconstrained position $\{p_i : (p_i, 1) \in \{0, ..., n-1\} \backslash \mathcal{F}\}$ intersected by row j on $\mathbf{H}$. The modified iterative decoding algorithm based on belief-propagation decoding algorithm over the binary erasure channel [76] is shown in Algorithm 2. As noted in the line with marked (*), the modified iterative decoder is not necessary to call, if the condition of $r_j \leq 1$ and $r'_j = 1$ is not met; or the branch with constraint set $\mathcal{F}$ can be ignored, if condition of $r_j = 1$ and $r'_j = 0$ occurs. Thus the computing complexity can be significantly reduced or neglected rather than calling it for every new branch with the corresponding constraint set $\mathcal{F}$.

Algorithm 2 Modified Iterative Decoding

```
Get rank $\mathbf{r}(\mathcal{F})$ and $\mathbf{r}'(\mathcal{F})$ for all the equation rows on $\mathbf{H}$.
repeat
  if $r_j > 1$ then
    Row $j$ is flagged
  else if $r_j = 1$ and $r'_j = 0$ then
    Contradiction $\rightarrow$ Quit decoder
  else if $r_j \leq 1$ and $r'_j = 1$ then
    1). Row $j$ is flagged
    2). The variable bit $i$ is decoded as the XOR of the value of $r_j$.
    3). Update the value of $r_j$ and $r'_j$, if $H_{ji} = 1$.
  end if
until No new unconstrained bit is decoded
```

Thus, we conclude the following three actions based on the different situations.

- **Contradiction**: The parity-check equation has weight 1, one more bit needed to complete the parity-check equation.

In Constraint $\mathcal{F}$					
1	0	0	0	0	0

- **Decodable Bits**: For constraint set $\mathcal{F}$, equation only has weight 0 or 1, one more bit needed to complete the parity-check equation.

In Constraint $\mathcal{F}$	In Erasures
0 0 0 0 0	0
0 0 0 0 1	1

- **No Action**: For constraint set $\mathcal{F}$, equation has weight larger than 1, there is no action for one more bit or more than one bits needed to complete the parity-check equation, since it aims to find the stopping sets.

In Constraint $\mathcal{F}$	In Erasures
1 0 1 0 1	No
1 0 1 1 1	Action

8.3.3 Simple Method of Computing the Lower Bound

In Rosnes' algorithm [102], the complexity of computing the lower bound and the position selection occupy the majority algorithm complexity. The simplex method from *linear programming* (LP) is applied and provides the solution to get the lower bound based on the constraint set in [102]. The complexity of the simplex method depends on the number of involved unknown bits and the number of active rows, since there exists a number of inner loops to process the "pivot" operations. Although the LP is able to provide a tighter lower bound, which helps reduce the total number of searching branches, there still exists a big trade off between the complexity of computing the tighter bound and the complexity of searching more constraint sets in terms of the time-domain. Thus we propose a simple method with negligible computing complexity to get a reasonable tight lower bound. The objective of getting the lower bound is changed to find a smaller possible collection of active rows $\mathcal{I}(\mathcal{F}) = \{I_{i_0}(\mathcal{F}), ..., I_{i_{q-1}}(\mathcal{F})\}$, where $I_{i_0}(\mathcal{F})$ is the set of active rows with constraint set $\mathcal{F}$ corresponding to the i_0th column $\mathbf{h}_{i_0}$ of $\mathbf{H}$, and q is the number of intersected unknown bits. Let $w(\mathbf{h}_j^{I_j(\mathcal{F})})$ be the weight of ones on the jth column of $\mathbf{H}$, which is the number of active rows intersected by the jth column. The worst situation of active rows collection is considered that the $I_j(\mathcal{F})$ with larger column weight of ones on the jth column is always with value 1, then the active rows can be compensated by $I_j(\mathcal{F})$ and the total number of active rows ϕ is reduced by $w(\mathbf{h}_j^{I_j(\mathcal{F})})$ until $\phi \leq 0$. Algorithm 3 shows the pseudo-code of computing the smaller number of intersected unknown bits q, which help compute the lower bound $w'(\mathcal{F}) = w(\mathcal{F}) + q$.

Algorithm 3 Simple Method to Find the Smallest Collection Set of Active Rows

```
1. Arrange the set of $\mathcal{I}(\mathcal{F})$ in descending order, where $\mathbf{h}_{i'_0}$ is the column with the maximal column weight corresponding to constraint $\mathcal{F}$.
2. $q$ is initialised as 0.
while $\phi > 0$ do
  1). $\phi$ is subtracted by $w(\mathbf{h}_{i'_0})$.
  2). $q$ is accumulated by 1.
end while
```

Since the lower bound is the key to constrain the total number of necessary searching nodes, the lower bound algorithm is focused on the balance between the efficiency and tightness. The aim of the lower bound algorithm is to find a combination of the unknown bits corresponding to the active rows with either value of 0 or 1, which produces the minimal weight of the set of unknown bits. Table 8.1 shows an example of active rows over the parity-check matrix corresponding to a constraint set.

Row \ Column	j_5	j_{10}	j_{15}	j_{16}	j_{17}	j_{20}	j_{22}
h_1	1	0	1	0	1	1	0
h_{12}	1	1	1	0	1	0	0
h_{15}	0	1	0	0	0	1	0
h_{26}	1	0	0	1	0	0	0
h_{30}	0	1	0	1	0	0	1

Table 8.1: Example of Computing Lower Bound

According to the example, we can simply conclude the following inequalities:

$$\begin{aligned}
j_{1,5} \times x_5 + j_{1,15} \times x_{15} + j_{1,17} \times x_{17} + j_{1,20} \times x_{20} &\geq 1 \\
j_{12,5} \times x_5 + j_{12,10} \times x_{10} + j_{12,15} \times x_{15} + j_{12,17} \times x_{17} &\geq 1 \\
j_{15,10} \times x_{10} + j_{15,20} \times x_{20} &\geq 1 \\
j_{26,5} \times x_5 + j_{26,16} \times x_{16} &\geq 1 \\
j_{30,10} \times x_{10} + j_{30,16} \times x_{16} + j_{30,22} \times x_{22} &\geq 1
\end{aligned}$$

The above problem aims to minimise the sum of all the involved $(j_i \times x_i)$, where x_i is the bit value at position i, $x_i \in \{0, 1\}$. Thus we have

$$\min(j_5x_5 + j_{10}x_{10} + j_{15}x_{15} + j_{16}x_{16} + j_{17}x_{17} + j_{20}x_{20} + j_{22}x_{22})$$

By applying the Algorithm 3, the minimised integer value indicates the lower bound of the specific constraint set.

8.3.4 Position Selection

Since in the proposed bound algorithm, all the selected unconstrained positions are assumed with value 1, then the first position in the index list with maximal column weight becomes an ideal selection. By comparing to the selection criteria from [102], which is based on the simplex method, the position selection rule based on the proposed simple method becomes much simpler with negligible computation complexity. In practice, a more sophisticated method has been found that choosing the position j at maximal column weight in information bit rather than parity bit works even better.

8.4 Numerical Results

8.4.1 Well Known LDPC Codes

The algorithms above have been used to evaluate all of the low weight stopping sets for some well known LDPC codes. The codes weight spectrum results of up to $(s_{min}+6)$ are given in Table 8.2 and Table 8.3. The total number of stopping sets

Code Name	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$
Tanner $(155, 64)$ [118]	18	465 (0)	2015 (0)	9548 (1023)	23715 (0)
QC LDPC $(1024, 512)$ [67]	15	1 (1)	1 (0)	0 (0)	1 (1)
PEG Reg $(256, 128)$ [58, 79]	11	1 (0)	11 (7)	22 (12)	51 (28)
PEG Reg $(504, 252)$ [58, 79]	19	2 (0)	5 (2)	8 (0)	27 (5)
PEG iReg $(504, 252)$ [58, 79]	13	2 (1)	1 (1)	5 (5)	13 (11)
PEG iReg $(1008, 504)$ [58, 79]	13	1 (1)	0 (0)	0 (0)	3 (3)
MacKay $(504, 252)$ [79]	16	1 (0)	3 (0)	3 (0)	12 (0)
MacKay $(204, 102)$ [79]	8	1 (1)	2 (0)	3 (1)	6 (0)
LDPC $(200, 100)$	9	100 (100)	0 (0)	100 (0)	0 (0)

Table 8.2: Low Weight Stopping Sets and Codewords of LDPC Codes (Part 1)

Code Name	s_{min}	$N_{s_{min}+4}$	$N_{s_{min}+5}$	$N_{s_{min}+6}$
Tanner $(155, 64)$ [118]	18	106175 (6200)	359290 (0)	1473585 (43865)
QC LDPC $(1024, 512)$ [67]	15	6 (1)	6 (2)	12 (4)
PEG Reg $(256, 128)$ [58, 79]	11	116 (46)	329 (113)	945 (239)
PEG Reg $(504, 252)$ [58, 79]	19	78 (0)	199 (26)	– (–)
PEG iReg $(504, 252)$ [58, 79]	13	31 (16)	52 (28)	124 (60)
PEG iReg $(1008, 504)$ [58, 79]	13	3 (3)	4 (4)	5 (3)
MacKay $(504, 252)$ [79]	16	36 (2)	106 (0)	320 (22)
MacKay $(204, 102)$ [79]	8	14 (5)	59 (0)	229 (43)
LDPC $(200, 100)$	9	100 (0)	0 (0)	200 (0)

Table 8.3: Low Weight Stopping Sets and Codewords of LDPC Codes (Part 2)

are shown for a given weight with the number of codewords in parentheses. The minimum Hamming weights of stopping sets for Tanner coeds [118], MacKay codes [79] and one of the PEG codes [58] exactly match the results shown on [56, 102, 127]. Furthermore, more weight spectrum results are provided to help analyse those codes properties. Interestingly, for Tanner code $(155, 64)$ with d_{min} 20, it has 93 parity check equations, 2 more than the 91 parity-check equations are required to encode the code. The reason has been found that if only 91 parity-check equations are used in the iterative decoder, then there is a stopping set of weight 12 degrading the decoder performance, and s_{min} is found with 18 by decoding with all 93 parity-check equations.

8.4.2 WiMax LDPC Codes

8.4.2.1 Standard Construction

n	576	672	768	864	960	1056	1152	1248	1344	1440
Code	Minimum Codeword Weight d_{min}									
1/2	13	**19**	**20**	**19**	**19**	**21**	**19**	**22**	23	**27**
2/3A	*10*	*9*	*8*	*11*	*13*	*10*	*14*	*13*	*14*	*13*
2/3B	**12**	**11**	**14**	**16**	**15**	**15**	**16**	*15*	**16**	**17**
3/4A	**10**	**10**	*10*	**12**	**12**	*13*	**13**	**13**	**14**	12
3/4B	**8**	*8*	**9**	**11**	**11**	*9*	**11**	9	**12**	10
5/6	**5**	7	**7**	**7**	**7**	*7*	**7**	*7*	*7*	**7**
	Minimum Stopping Set Weight s_{min}									
1/2	13	**18**	**18**	**19**	**19**	**19**	**19**	**19**	**23**	*24*
2/3A	*10*	*9*	*8*	*9*	*12*	*10*	*13*	*13*	*14*	*13*
2/3B	*10*	**11**	**13**	**15**	**14**	**15**	**16**	*15*	**16**	**17**
3/4A	**9**	*8*	**10**	**11**	**12**	**12**	**10**	**12**	**12**	12
3/4B	**8**	8	**9**	**10**	**11**	9	**11**	*9*	**12**	*10*
5/6	**5**	**6**	**7**	**7**	**7**	*7*	**7**	*7*	*7*	**7**

n	1536	1632	1728	1824	1920	2016	2112	2208	2304
Code	Minimum Codeword Weight d_{min}								
1/2	20	**27**	**21**	19	25	**27**	**28**	23	*31*
2/3A	*12*	*13*	*15*	*15*	*15*	*15*	*15*	*15*	*15*
2/3B	*15*	**18**	*15*	*15*	**16**	*15*	**16**	**20**	*15*
3/4A	**14**	13	*17*	**13**	**17**	**17**	*15*	**20**	*19*
3/4B	**11**	**13**	**13**	**12**	*10*	*12*	**14**	**13**	*12*
5/6	**7**	7	**8**	*8*	*7*	7	*8*	*8*	*9*
	Minimum Stopping Set Weight s_{min}								
1/2	20	**27**	**21**	19	**25**	**27**	**28**	23	*28*
2/3A	*12*	*12*	*14*	*15*	*14*	*15*	*15*	*15*	*15*
2/3B	*15*	**18**	*15*	*15*	**16**	*15*	**16**	**20**	*15*
3/4A	*12*	*12*	12	**12**	12	12	12	12	*12*
3/4B	**11**	**13**	**12**	**12**	**10**	11	**14**	**13**	*12*
5/6	**7**	7	*7*	*8*	**7**	7	*8*	*8*	*9*

Table 8.4: WiMax LDPC Codes' d_{min} and s_{min} (Standard)

WiMax LDPC codes [1] , as the IEEE 802.16e standardised LDPC codes, have been fully explored at the low weight stopping sets in variety of code rates and codeword lengths. Such LDPC codes are designed using a binary base matrix by shifting the $z \times z$ identity matrix with different factors, which are based on the size of the codeword length. Such WiMax LDPC codes in a variety of code rates and codeword lengths are fully explored at the low weight of stopping sets up to $s_{min} + 4$ as shown in Appendix C. Then the overall d_{min} and s_{min} for all range of WiMax LDPC codes in variant code-rates are given in Table 8.4.

8.4.2.2 Modulo Construction

In this section, we present another class of WiMax LDPC codes based on the modulo function, according to [1]. Such class of LDPC codes in a variety of code rates and codeword lengths is fully explored at the low weight of stopping sets up to $s_{min} + 4$ as shown in Appendix D, which provide more sufficient weight spectrum about such WiMax LDPC class of codes as shown in [103]. The overall d_{min} and s_{min} for all range of WiMax LDPC codes with modulo construction in variant code-rates are given in Table 8.5.

In comparison between standard construction and modulo construction, the value in **bold** indicates larger Hamming distance at the corresponding codeword length and code-rate. And value in *italic* indicates the identical Hamming distance at the same codeword length and code-rate. In general, the codes in standard construction provide better or identical Hamming distances. In standard construction, the code construction for code-rate 2/3A also follows the modulo construction. Thus for all codes in code-rate of 2/3A with variant codeword lengths, the d_{min} and s_{min} are identical between standard construction and modulo construction. Furthermore, for codes in codeword length 2304, both construction methods are equivalent, thus their Hamming distances of d_{min} and s_{min} are also identical between standard construction and modulo construction.

8.5 Summary

In this chapter, an efficient algorithm has been presented to evaluate all of the low weight stopping sets and codeword sets for some well known LDPC codes, WiMax standardised LDPC codes and WiMax LDPC codes under modulo construction.

n	576	672	768	864	960	1056	1152	1248	1344	1440
Code	Minimum Codeword Weight d_{min}									
1/2	**16**	13	14	17	14	20	16	17	**24**	24
2/3A	*10*	*9*	*8*	*11*	*13*	*10*	*14*	*13*	*14*	*13*
2/3B	10	8	12	14	13	14	15	*15*	15	15
3/4A	9	8	*10*	10	11	*13*	10	11	10	**15**
3/4B	6	*8*	4	9	5	**10**	6	*9*	9	**11**
5/6	3	**8**	6	4	6	*7*	6	*7*	*7*	6
	Minimum Stopping Set Weight s_{min}									
1/2	**15**	13	14	15	14	18	16	17	22	*24*
2/3A	*10*	*9*	*8*	*9*	*12*	*10*	*13*	*13*	*14*	*13*
2/3B	*10*	8	12	13	13	14	10	*15*	15	15
3/4A	6	*8*	9	10	10	11	6	11	9	**14**
3/4B	6	6	4	9	5	**10**	6	*9*	9	*10*
5/6	3	5	6	4	4	*7*	6	*7*	*7*	6

n	1536	1632	1728	1824	1920	2016	2112	2208	2304
Code	Minimum Codeword Weight d_{min}								
1/2	**24**	24	19	**22**	*25*	26	20	**24**	*31*
2/3A	*12*	*13*	*15*	*15*	*15*	*15*	*15*	*15*	*15*
2/3B	*15*	15	*15*	*15*	15	*15*	15	15	*15*
3/4A	12	**15**	*17*	9	15	14	*15*	15	*19*
3/4B	8	12	12	11	*10*	*12*	13	12	*12*
5/6	6	**8**	*7*	*8*	*7*	**8**	*8*	*8*	*9*
	Minimum Stopping Set Weight s_{min}								
1/2	**23**	24	19	**20**	23	26	20	**24**	*28*
2/3A	*12*	*12*	*14*	*15*	*14*	*15*	*15*	*15*	*15*
2/3B	*15*	15	*15*	*15*	15	*15*	15	15	*15*
3/4A	*12*	*12*	**14**	9	**15**	**14**	**15**	**15**	*12*
3/4B	8	12	11	11	6	**12**	12	12	*12*
5/6	6	**8**	*7*	*8*	4	**8**	*8*	*8*	*9*

Table 8.5: WiMax LDPC Codes' d_{min} and s_{min} (Modulo)

The related searching algorithm has been detailed with different graphical examples. And the proposed bound algorithm has been explained and indicated its low computational complexity on computing the lower bound. Due to the efficiency of the proposed exhaustive tree search algorithm, a potential research direction has been realised, which could be to help decode for such linear block codes with sparse parity-check matrices over the AWGN channel. Such a decoding method by adopting the tree search based algorithm will be explained and detailed in Chapter 9.

Chapter 9

Exhaustive Tree Search based Optimal Decoding

9.1 Introduction

The searched error vector introduced by Tomlinson [121] consists of the hard-decided received vector bit-wised by a codeword derived from a re-encoded codeword according to the k nearly ordered most-reliable information bits and a codeword with low-weight information. Processing this error vector produces the most likelihood codeword with the closest Euclidean distance compared to the received vector. In this chapter, based on the knowledge of finding small weight of stopping sets and codeword sets introduced in Chapter 8, an algorithm for searching all low-weight error patterns on a received vector for LDPC codes is proposed. The codeword set and related representation for the modified codeword search algorithm are described in Section 9.2. Section 9.3 introduces the modified expanded Dorsch decoding algorithm coupled with branch and bound, exhaustive search for finding low-weight codewords. Some considerations to optimise the search algorithm are discussed. In Section 9.4, the extended algorithm for finding low weight error vectors is described in terms of a new lower bound algorithm. Section 9.5 compares results of iterative decoding performance, sub-optimal decoding performance and optimal decoding performance for some linear codes with sparse structure and reasonable codeword length.

9.2 Tree-Search based Codeword Set Enumeration Algorithm

In common with Chapter 8, the similar notations and preliminaries are used in this chapter as described in Section 8.2.1. Let $\mathcal{S}$ denote a subset of $\{0,1\}^n$, the set of all binary vectors of length n. At any point during the tree search process, a *constraint set*, $\mathcal{F}$ is defined to consist of bit positions p_i and the states of these bit positions s_{p_i}, $s_{p_i} \in \{0,1\}^n$. The support set $\mathcal{X}(\mathcal{F})$ of the constraint set $\mathcal{F}$, is the set of positions where $s_{p_i} = 1$ and the Hamming weight $w(\mathcal{F})$ of $\mathcal{F}$ is the total number of such positions. The sub-matrix $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$ is defined by consisting of all the columns of $\mathbf{H}$ where $s_{pi} = 1$, and the row weight of $h_i^{\mathcal{X}(\mathcal{F})}$ is the total number of $1's$ at row i. A row of h_i is considered as an *active* row of $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$, if $w(h_i^{\mathcal{X}(\mathcal{F})})$ has odd row weight. If a constraint set $\mathcal{F}$ with size of n contains no active row of $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$, thus the set $\mathcal{F}$ forms a codeword with weight $w(\mathcal{X}(\mathcal{F}))$. Thus a *codeword* set is a subset of $\{x_0, ..., x_{n-1}\}$ with Hamming weight d, $d = w(\mathcal{X}(\mathcal{F}))$, where the induced subgraph contains no check node with odd degree. The minimal Hamming distance d_{min} is defined as the minimal weight of the non-empty codeword set. If there exist active rows, then $\mathcal{F}$ has either to be appended with additional bit positions if $|\mathcal{F}| < n$, or one or more states s_{p_i} need to be changed to compensate the active rows with odd row weight.

Since this research work only aims to find codewords excluding stopping sets, the criteria might be simplified based on Algorithm 1. The Tree-search based Codeword Set Enumeration (TCSE) algorithm is designed to search all the codeword sets up to size of threshold τ for any parity-check code $\mathcal{C}$.

In the equivalent graphical tree view representation, the constraint set can be considered as a *branch* of the tree, which represents the set of searched known bit-positions of a code $\mathcal{C}$. $|\mathcal{F}|$ denote the size of $\mathcal{F}$, representing the known depth of the tree. Since the active row is defined as the row with odd weight, thus the set of active rows in $\mathbf{H}$ is denoted by $\{\mathfrak{h}_0, ..., \mathfrak{h}_{\phi-1}\}$, where ϕ is the total number of active rows. A constraint set $\mathcal{F}$ with size n is said to be *valid* if and only if there exists no active row in $\mathbf{H}^{(\mathcal{F})}$, in other words, a codeword is found. The pseudocode of the algorithm to find all the codeword sets by threshold size τ is given in Algorithm 4. When the whole tree has been explored, constrained by the lower bound, the list of the saved constraint sets in full size n is the whole set of codewords up to size of τ. The contradiction is considered as that if $w(h_i^{\mathcal{X}(\mathcal{F})}) = 1$

Algorithm 4 Tree-search based Codeword Set Enumeration (TCSE)

```
repeat
  Pick one untouched branch as a constraint set $\mathcal{F}$.
  if $|\mathcal{F}| = n$ and $w(\mathcal{X}(\mathcal{F})) \leq \tau$ then
    Constraint set $\mathcal{F}$ is saved, if $\mathcal{F}$ is valid
  else
    1). Pass $\mathcal{F}$ to the iterative decoder with erasures in the unconstrained positions.
    2). Construct a new constraint set $\mathcal{F}'$ with new decoded positions, which is the extended branch.
    if $|\mathcal{F}'| = n$ and $w(\mathcal{X}(\mathcal{F}')) \leq \tau$ then
      Constraint set $\mathcal{F}'$ is saved, if $\mathcal{F}'$ is valid
    else if No contradiction is found in $\mathbf{H}_{\mathcal{X}(\mathcal{F}')}$, and $w'(\mathcal{X}(\mathcal{F}')) \leq \tau$ then
      a). Pick an unconstrained position $p$.
      b). Extend the branch $\mathcal{F}'$ to position $p$ to get new branch $\mathcal{F}'' = \mathcal{F}' \bigcup \{(p, 1)\}$ and branch $\mathcal{F}''' = \mathcal{F}' \bigcup \{(p, 0)\}$.
    end if
  end if
until Tree has been fully explored by threshold $\tau$
```

and $w(h_i'^{\mathcal{X}(\mathcal{F})}) = 0$, where $h_i'^{\mathcal{X}(\mathcal{F})}$ is the row weight of row i for the unconstrained positions $\{p_i : (p_i, 1)\} \in \{0, 1, ..., n-1\} \backslash \mathcal{F}\}$ intersected by row i on $\mathbf{H}$. The iterative BP decoder over the erasure channel [76] is considered as the candidate iterative decoder. It should be noted that it is not necessary to call the iterative decoder unless the condition of $w(h_i'^{\mathcal{X}(\mathcal{F})}) = 1$ is met. Thus it helps reduce the computational complexity to call the decoder for every new branch $\mathcal{F}'$. The lower bound algorithm $w'(\mathcal{F}')$ and position selection criteria follow the same rule as in Section 8.3.4.

9.3 Algorithm Inspired by Dorsch Decoding

Since the exhaustive tree-search based algorithm uses the parity-check matrix $\mathbf{H}$, then the dual codes $\mathcal{C}'$ generated by the parity-check equations $\mathbf{h}$ corresponding to linear block codes $\mathcal{C}$ are explored and limited by a threshold τ. Thus the Dorsch decoding algorithm is extended to search the bounded dual codes $\mathcal{C}'$ instead of the codes $\mathcal{C}$, which is generated by the re-ordered nearly most-reliable k information bits. The BPSK-modulated codeword $\mathbf{x}$ is transmitted through the AWGN channel and received as the vector $\mathbf{r} = \{r_0, ..., r_{n-1}\} \in \mathbb{R}$, which is affected by noise with variance σ^2. A hard-decided received vector $\mathbf{b} \in \{0, 1\}^n$ is derived from the received vector $\mathbf{r}$ using bit-wise decisions. Usually, the binary vector $\mathbf{b}$ is not a codeword. The re-encoded codeword $\hat{\mathbf{x}}$ is produced by $\mathbf{H}$ according to the binary information set of $\mathbf{b}$. The maximum attainable correlation Y_{max} with the received vector is given by

$$Y_{max} = \sum_{j=0}^{n-1} |r_j| \tag{9.1}$$

According to the extended Dorsch decoding algorithm [121], the codeword $\tilde{\mathbf{x}}_i$ with low weight is used to find the maximised codeword $\mathbf{x}_i$ corresponding to the received vector $\mathbf{r}$, is given by

$$\mathbf{x}_i = \hat{\mathbf{x}} \oplus \tilde{\mathbf{x}}_i \tag{9.2}$$

Then the corresponding binary error vector $\mathbf{z}_i$ is given by

$$\mathbf{z}_i = \mathbf{b} \oplus \hat{\mathbf{x}} \oplus \tilde{\mathbf{x}}_i \tag{9.3}$$

The first error vector $\hat{\mathbf{z}}$ is defined as

$$\hat{\mathbf{z}} = \mathbf{b} \oplus \hat{\mathbf{x}} \tag{9.4}$$

Thus the cross-correlation cost $Y(\mathbf{x}_i)$ is given by

$$Y(\mathbf{x}_i) = Y_{max} - Y_{\triangle}(\mathbf{x}_i) \tag{9.5}$$

where $Y_{\triangle}(\mathbf{x}_i)$ is defined by

$$Y_{\triangle}(\mathbf{x}_i) = 2 \sum_{j=0}^{n-1} (\hat{z}_j \oplus \tilde{x}_{i_j}) \times |r_j| \tag{9.6}$$

In order to find the codeword $\mathbf{x}_i$ with smallest cross-correlation reduction from Y_{max}, which is the smallest difference between Y_{max} and $Y(\mathbf{x}_i)$, then we find the smallest $Y_\triangle(\mathbf{x}_{min})$, starting with $Y_\triangle(\hat{\mathbf{x}})$

$$Y_\triangle(\mathbf{x}_{min}) = \min(Y_\triangle(\mathbf{x}_i)) \tag{9.7}$$

This may be achieved by searching for the low weight codeword $\mathbf{x}_i$ using the TCSE algorithm with the following modifications.

- Each new explored branch $\mathcal{F}$ will be checked with its $Y(\mathcal{F})$ comparing to $Y_\triangle(\mathbf{x}_{min})$, where its position value s_{p_i} is different from $\hat{z}_{p_i}$, before checking its size.
- A constraint set $\mathcal{F}$ is *valid*, if and only if there exists no active row with $w(\mathcal{X}(\mathcal{F}))$ less than τ and $Y_\triangle(\mathcal{F}) < Y_\triangle(\mathbf{x}_{min})$. Then $Y_\triangle(\mathbf{x}_{min})$ is updated as $Y_\triangle(\mathcal{F})$.

The position criteria is further constrained by

- The new position p with value $s_p \in \{0,1\}$ will be selected when $(Y(\mathcal{F}) + |r_p|) < Y_\triangle(\mathbf{x}_{min})$, thus new branch $\mathcal{F}' = \mathcal{F}\bigcup\{(p,1)\}$ and branch $\mathcal{F}'' = \mathcal{F}\bigcup\{(p,0)\}$ are expanded.
- If $(Y(\mathcal{F}) + |r_p|) \geq Y_\triangle(\mathbf{x}_{min})$, then a new branch $\mathcal{F}' = \mathcal{F}\bigcup\{(p,\hat{z}_p)\}$ is expanded. It should be noted that it is different from [2], such that position p can not be ignored during the codeword search, since it may miss a subset of potential codeword $\mathbf{x}_i$.

In the simpler lower bound algorithm, for the set of active rows $\mathfrak{h}$, the unconstrained bit p_j, $(j \in \{0,...,n-1\}\backslash\mathcal{F})$, is considered intersected by $\mathfrak{h}_i$, where $H_{ij} = 1$. A further consideration can be revised that the bit position p_j with $(Y(\mathcal{F}) + |r_{p_j}|) < Y_\triangle(\mathbf{x}_{min})$ might be prior considered in computing the lower weight. In order to achieve a tighter bound, the integer linear programming algorithm can also be used, but this involves an increase in complexity, requiring iterative computations of *pivot* operations, even though $\mathbf{H}$ is a sparse parity-check matrix of LDPC codes.

9.4 Simplified Approach based on TCSE Algorithm

At low SNR, the low weight codeword search has to proceed until the threshold is reached, and the simple method bound does not provide a sufficiently tight lower bound to limit the size of codeword search. Another approach based on the basic algorithm is proposed. It is to use TCSE algorithm to search for an error vector $\tilde{\mathbf{e}}_i$, which contains an index set $\mathcal{I}_{\tilde{\mathbf{e}}_i}$ consisting of positions, where their values are different from received binary vector $\mathbf{b}$. Thus the error vector $\tilde{\mathbf{e}}_i$ is given by

$$\tilde{e}_{i_j} = \begin{cases} |b_j - 1|, & \text{if } j \in \mathcal{I}_{\tilde{\mathbf{e}}_i} \\ 0, & \text{if } j \bigcap \mathcal{I}_{\tilde{\mathbf{e}}_i} = \emptyset \end{cases} \tag{9.8}$$

The cross-correlation difference $Y_\triangle(\tilde{\mathbf{e}}_i)$ corresponding to Y_{max} is given by

$$Y_\triangle(\tilde{\mathbf{e}}_i) = 2\sum_{j=0}^{n-1}(\tilde{e}_{i_j}) \times |r_j| \tag{9.9}$$

The smallest cross-correlation difference $Y_\triangle(\mathbf{e}_{min})$ is given by

$$Y_\triangle(\mathbf{e}_{min}) = \min(Y_\triangle(\tilde{\mathbf{e}}_i)) \tag{9.10}$$

Then the constraint set $\mathcal{F}$ is redefined as a set with $s_{p_j} = |b_{p_j} - 1|$, which produces extra cross-correlation cost $\mathcal{R}(\mathcal{F}) = \sum_{j=0}^{|\mathcal{F}|}(|r_{p_j}|)$. $\mathcal{F}$ is said to be *valid* if and only if $\mathbf{H}(\hat{\mathbf{e}}_\mathcal{F} \oplus \mathbf{b}) = 0$, where $\hat{\mathbf{e}}_\mathcal{F}$ is the error vector corresponding to constraint set $\mathcal{F}$ with extra cost less than threshold $Y_\triangle(\mathbf{e}_{min})$. In other words, there is no active row for $(\hat{\mathbf{e}}_\mathcal{F} \oplus \mathbf{b})$ on $\mathbf{H}$. The simplified tree-search based error vector enumeration (STESE) is depicted in Algorithm 5 to search all the error vectors with extra cost up to $Y_\triangle(\mathbf{e}_{min})$, which might be initialised as $Y_\triangle(\hat{\mathbf{z}})$.

The simplified algorithm is designed to find all the potential error-vector sets with extra cost less than threshold. Once a smaller error vector is found, then the threshold is updated. Thus the search size of the error vectors is reduced as the threshold decreases. $\mathcal{R}'(\mathcal{F})$ is designated to compute the lower bound of potential cost corresponding to $\mathcal{F}$.

Algorithm 5 Simplified Tree-search based Error-vector Set Enumeration (STESE)

```
repeat
  Pick one untouched branch as a constraint set $\mathcal{F}$.
  if $\mathcal{R}(\mathcal{F}) < Y_{\triangle}(\mathbf{e}_{min})$ then
    1). Constraint set $\mathcal{F}$ is picked, if $\mathcal{F}$ is valid.
    2). And $Y_{\triangle}(\mathbf{e}_{min})$ is updated as $\mathcal{R}(\mathcal{F})$
  else
    Active rows set $\mathfrak{h}_{\mathcal{F}}$ is collected
    if $\mathcal{R}'(\mathcal{F}) \leq Y_{\triangle}(\mathbf{e}_{min})$ then
      a). Pick an unconstrained position $p$.
      b). Extend the branch $\mathcal{F}$ to position $p$ to get new branch $\mathcal{F}' = \mathcal{F} \bigcup \{(p, |b_p - 1|)\}$.
    end if
  end if
until Tree has been fully explored by updated $Y_{\triangle}(\mathbf{e}_{min})$
```

9.4.1 Computing $\mathcal{R}'(\mathcal{F})$ on $\mathcal{F}$

The simple method bound given in Algorithm 3 and [2] computes the intersected positions by active rows ordered by column-weight. Each ordered position with its column-weight is deducted from the total number of active rows ϕ until all active rows are compensated. Thus the number of deducted positions is the least possible bound, which guarantees that no potential p missed. The proposed bound algorithm is to estimate the potential coordinates where their column weights could compensate the active rows, meanwhile their total cost has to be less than the difference between threshold and $\mathcal{R}(\mathcal{F})$. Let $\mathcal{I}(\mathcal{F}) = \{I_{i_0}(\mathcal{F}), ..., I_{i_{q-1}}(\mathcal{F})\}$ be the active rows index set, where $I_{i_0}(\mathcal{F})$ is the set of active rows on $\mathcal{F}$ corresponding to the i_0th column of $\mathbf{H}$, and q is the total number of intersected positions. First of all, the bound is designed to order the intersected positions by their normalised cost, which is its cost $|r_{i_0}|$ divided by the column weight $w(I_{i_0})$. Then it follows the same strategy, the least ordered position is picked. Its column weight is deducted and the threshold is checked, it is repeated until all active rows are compensated or threshold is exceeded. During observations, the lower bound algorithm can not provide the least probable bound, as the normalised position can not guarantee to attain all the potential error patterns. Such an issue could have arisen because the involved positions actually produce different combinations to satisfy the check criteria without in any arranged order.

Thus a lower bound algorithm is realised to find a possible position combination to satisfy the requirement of all active rows' compensations and extra allowed cross-correlation cost. Let $\mathfrak{L}$ be a list set $\in \{0, ..., n-1\}$ to store positions' indices. $\eta(\mathfrak{L})$ is denoted as a flag set to flag the positions on $\mathfrak{L}$, it consists of the positions selected in $\mathfrak{L}$ and the excluded positions. The excluded position is the position p with $(|r_p| + \mathcal{R}(\mathcal{F})) \geq Y_{\triangle}(\mathbf{e}_{min})$. The position combination search algorithm is inspired by the codeword search algorithm, which follows the similar strategy. The proposed position-combination search algorithm as a lower bound is depicted in Algorithm 6, where $w(\mathfrak{L})$ represents the total number of column weight corresponding to the positions in $\mathfrak{L}$. The allowed cross-correlation cost based on $\mathcal{F}$

Algorithm 6 Lower Bound Algorithm to Find a Satisfied Position-Combination

```
repeat
    Pick one unflagged position as a list set $\mathfrak{L}$.
    if $(\mathcal{R}(\mathfrak{L}) + \mathcal{R}(\mathcal{F})) < Y_{\triangle}(\mathbf{e}_{min})$ and $w(\mathfrak{L}) \geq \phi$ then
        1). The satisfied position-combination is found.
        2). Exit.
    else if $w(\mathfrak{L}) < \phi$ then
        a). Pick an unflagged position $p$ based on $\eta(\mathfrak{L})$, where $(\mathcal{R}(\mathfrak{L}) + \mathcal{R}(\mathcal{F}) + |r_p|) < Y_{\triangle}(\mathbf{e}_{min})$.
        b). Extend the list $\mathfrak{L}$ with position $p$ to get new $\mathfrak{L}'$ and $\eta(\mathfrak{L}') = (\eta(\mathfrak{L}) \bigcup p)$.
    end if
until Combination is found or maximum iteration number is reached
```

decreases, as $\mathcal{R}(\mathcal{F})$ reaches the threshold. Once the qualified number of intersected positions gets smaller, then the iterations in finding a position-combination gets faster and easier to determine. At the beginning stage of extending the search on the tree, there might exist a relatively large number of positions based on the active rows, even though $\mathbf{H}$ is sparse. With the increase of the codeword length, the complexity of computing the lower bound is significantly increased, a limitation on the iteration is required to stay within a reasonable computational complexity. The position for the extended tree search from computing on the lower bound is ideal to pick the position in the approved combination with highest column-weight and least cost at the same column weight.

9.5 Numerical Results

The related codeword weight spectra for some short length LDPC codes are shown in Table 9.1, which are computed by Algorithm 4. The Union bounds based on the weight spectrum for the codes are illustrated against the corresponding simulation results.

LDPC code	d_{min}	$N_{d_{min}}$	$N_{d_{min}+1}$	$N_{d_{min}+2}$	$N_{d_{min}+3}$	$N_{d_{min}+4}$
Tanner code $(155, 64)$ [118]	20	1023	0	6200	0	43865
MacKay code $(120, 56, 10)$ [79]	10	3	0	51	0	399
MacKay code $(96, 32, 14)$ [79]	14	7	0	24	0	127
EG code $(63, 37, 9)$ [119]	9	1960	10584	9702	42042	179928

Table 9.1: Codeword Set Weight Spectrum up to $d_{min} + 4$

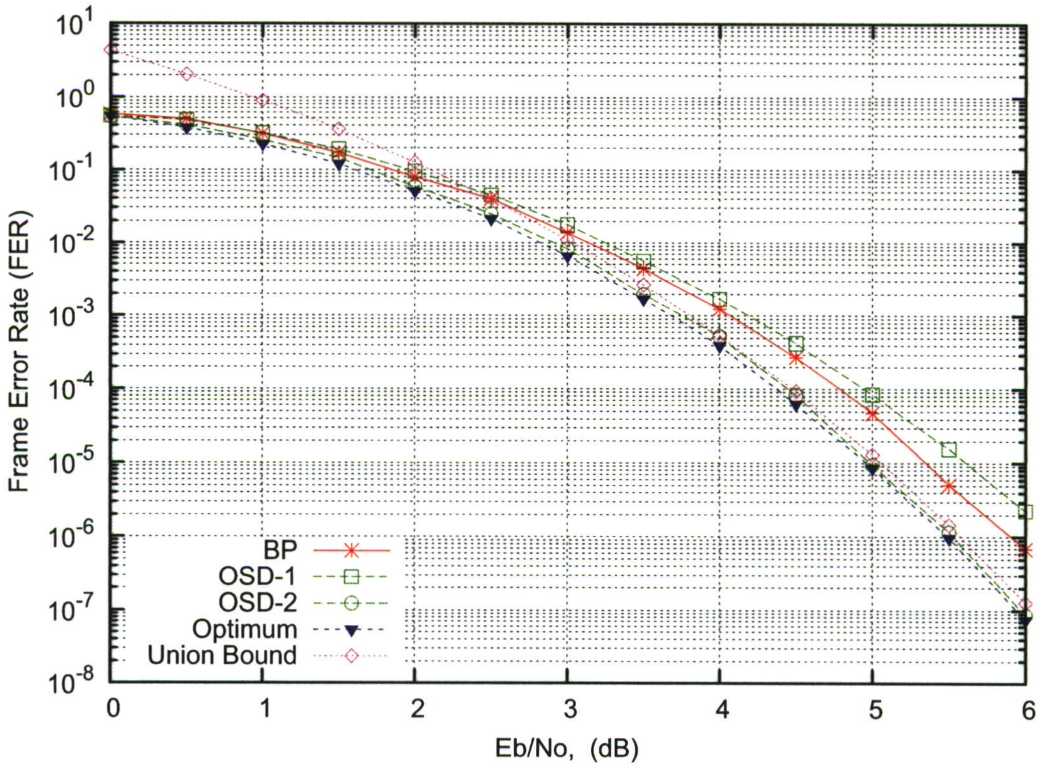


Figure 9.1: FER Performance of EG LDPC Codes $(63, 37, 9)$

Euclidean Geometry (EG) LDPC code $(63, 37, 9)$ based on cyclotomic idempotents proposed by Tjhai *et al.* [119] is a short code with reasonable d_{min} of 9 and sparse parity-check matrix. The corresponding results of the code with iterative output, optimum output and sub-optimal OSD output are shown in Figure 9.1. The iterative decoder output is with BP decoder after 50 iterations, which shows EG codes with cyclic structure over the parity-check matrix can be well decoded by BP decoder as shown in [62]. The OSD-2 decoder has nearly achieved the optimum performance as the proposed decoder achieves. The optimum decoder is with the proposed algorithm with limitation of 1×10^4 nodes search. Furthermore, the lower bound is limited to 50 iterations during the position combination search, which is sufficient to find the maximised codeword.

MacKay LDPC code $(96, 32, 14)$ [80] is a regular LDPC code with $(3, 5)$ and $(2, 4)$ mixture, which indicate the column weight with 5 or 4 and the row weight with 3 or 2. Thus its parity-check matrix guarantees a very sparse structure at $\frac{1}{3}$ code rate with a good d_{min} of 14. The corresponding results of the code with iterative output, optimum output and sub-optimal OSD output are shown in Figure 9.2. The iterative decoder output is with BP decoder after 50 iterations, and it achieves a better result than OSD-1, with 0.5dB coding gain since 4dB $\frac{E_b}{N_o}$. The optimum decoder is with the proposed algorithm with limitation of 1×10^6 nodes search and 200 iterations of the position combination search for lower bound. Such decoding configuration successfully achieves the optimum decoding performance against the Union bound, especially after $\frac{E_b}{N_o}$ 4db. Furthermore, it is clearly shown that OSD-i with small order number, where $i \leq 3$, could not achieve the optimum decoding performance under the maximum searching size of $\binom{32}{3}$. On the other hand, the proposed decoder shows its benefit and becomes the ideal candidate for such sparse linear block codes.

MacKay LDPC code $(120, 56, 10)$ [80] is a $\frac{1}{2}$ rate assorted regular Gallager code with mixture of $(3, 6)$ and $(3, 5)$, which provides a sparse parity-check matrix with row weight 3 and column weight 6 or 5 and reasonable d_{min} of 10. The corresponding results of the code with iterative output, optimum output and sub-optimal OSD output are shown in Figure 9.3. The BP iterative decoding output is achieved with 50 iterations and has better performance than OSD-1 with 0.5dB additional coding gain at 4.5dB $\frac{E_b}{N_o}$. The decoder uses the algorithm described above with a limitation of 10^7 nodes search and 500 iterations of position combination search for the lower bound. With such decoding arrangement,

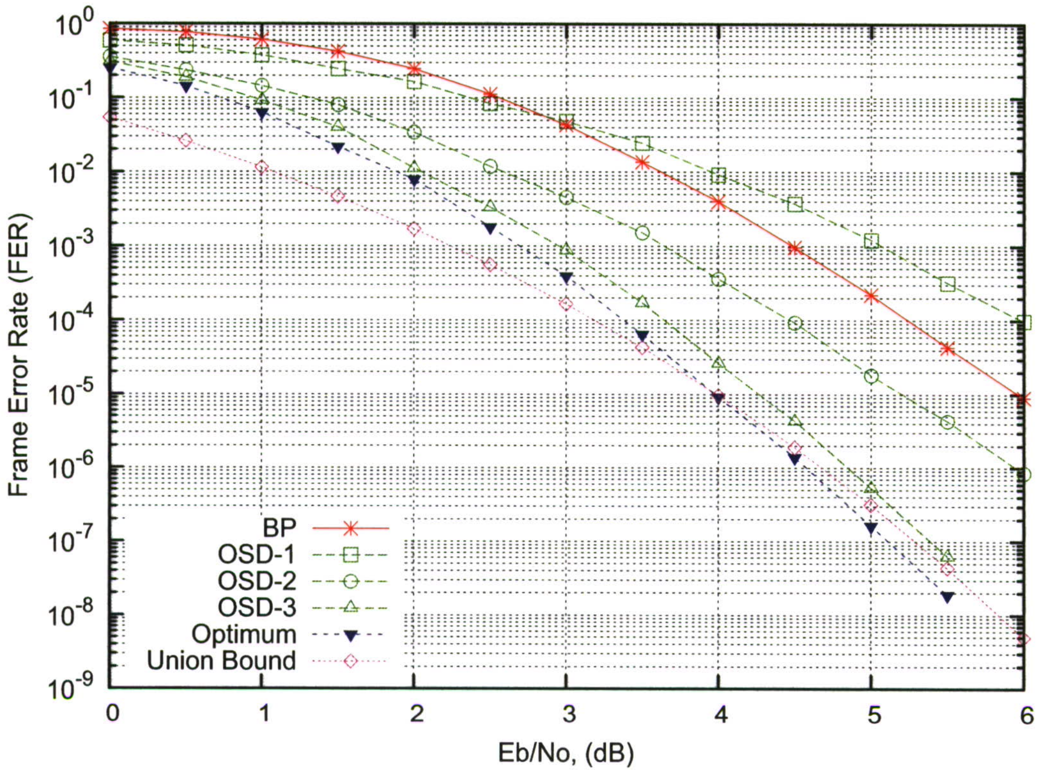


Figure 9.2: FER Performance of MacKay LDPC Codes $(96, 32, 14)$

the decoder successfully achieves near-optimum performance compared to the computed union bound.

Tanner code $(155, 64)$ is a $(3, 5)$-regular LDPC code constructed by Tanner *et al.* [118]. The underlying Tanner graph has girth 8 and its relatively large minimum distance of 20 makes the code an excellent candidate for iterative decoding. The results of the code with iterative output, optimum output and sub-optimal OSD output are shown in Figure 9.4. The optimum decoder is with the proposed algorithm with limitation of 1×10^8 nodes search. Furthermore, the lower bound is limited to 500 iterations during the position combination search, which is sufficient to find the maximised codeword. From the results it can be seen that the iterative decoding performance achieves similar results to the OSD-2 decoder. As the order number i increases, the sub-optimal decoding performance is significantly improved by more than 1dB additional coding gain for each increased order number. And the near-optimum decoding performance is successfully achieved

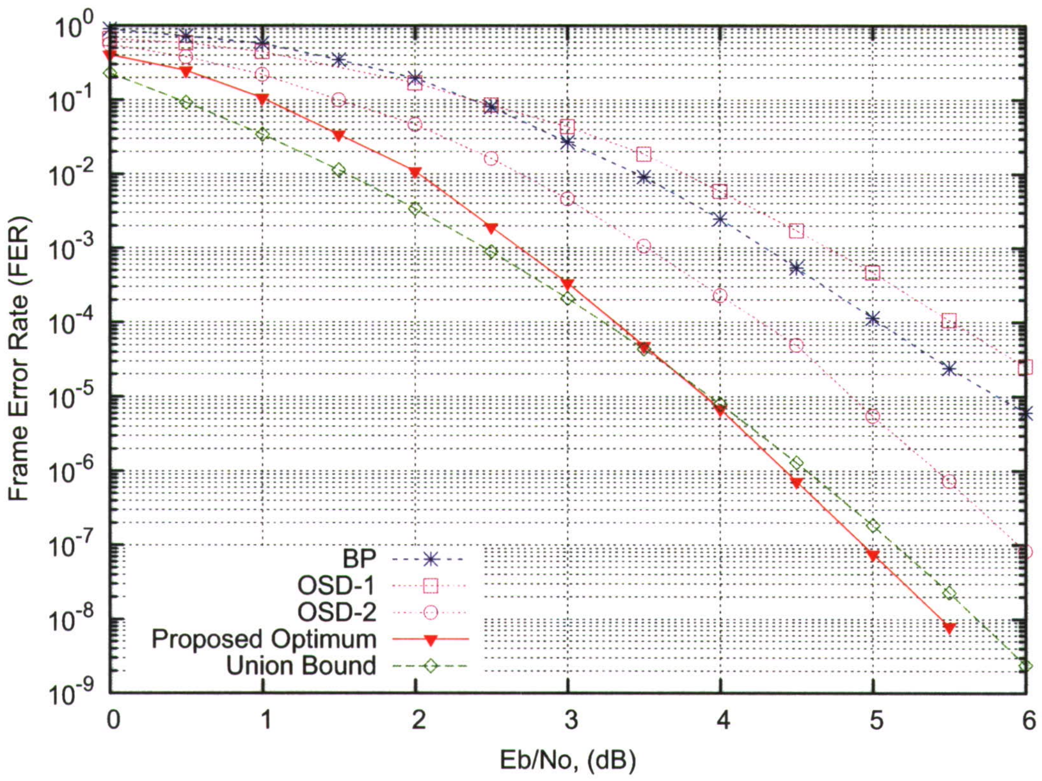


Figure 9.3: FER Performance of MacKay LDPC Codes (120, 56, 10)

by the proposed tree-search based error-vector searching decoder using 500 iterations of position combination search and 10^8 maximum nodes search. The near-optimum decoding performance approaches closely to the computed union bound at 4dB $\frac{E_b}{N_o}$.

9.6 Summary

In this chapter, a new approach of achieving maximum-likelihood achievement based on the Dorsch decoder has been introduced. The approach has utilised an efficient exhaustive tree search algorithm to find a low-weight codeword, which is used to find the closest codeword to the received vector. A further optimisation has been described, which is based on the basic algorithm to exhaustively search for a low weight error vector, in order to determine the closest codeword. The practical simulation results have demonstrated that the proposed decoder algo-

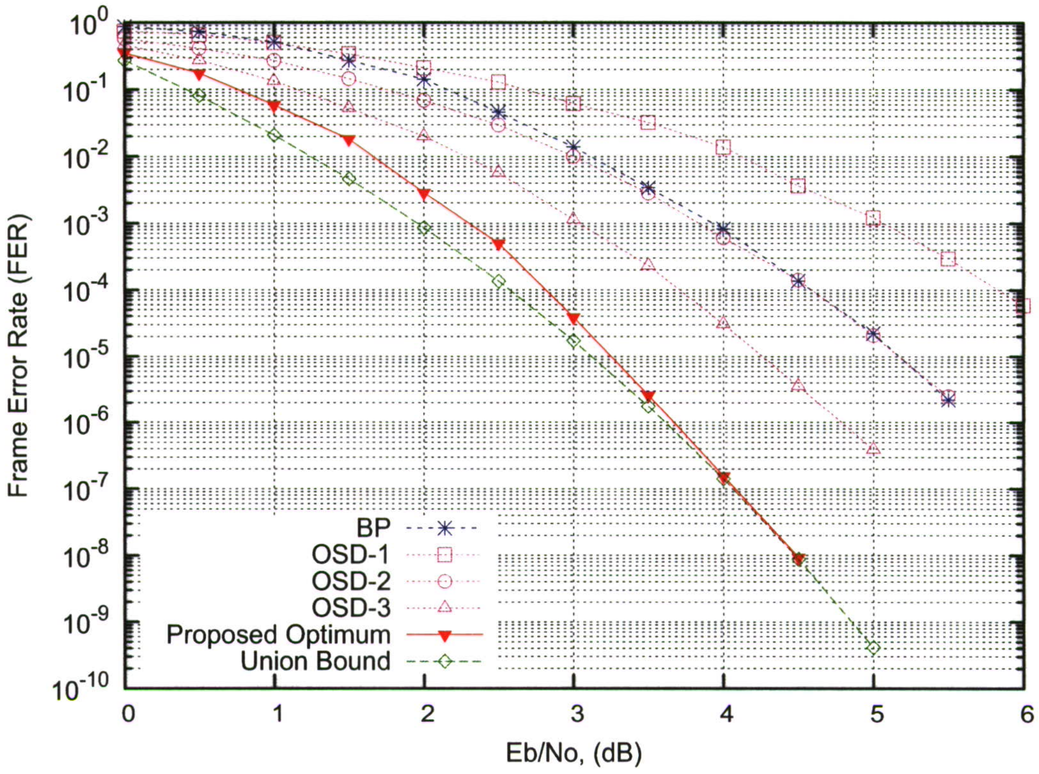


Figure 9.4: FER Performance of Tanner Codes $(155, 64, 20)$

rithm is able to approach the near-optimum decoding using a bounded search algorithm. Since the search has exploited the sparseness of the parity-check matrix of the code, it is more suitable for linear block codes having very sparse parity-check matrices, such as some range of LDPC codes.

Part IV

Conclusions and Future Research Work

Chapter 10

Conclusions and Future Work

10.1 Conclusions

This research work is dedicated to the designated decoding algorithms in terms of sub-optimal or optimum decoding achievement with optimised computational complexity for the codes, which can be decoded iteratively by updating the additional confidence knowledge. And these codes evaluations and stopping sets analysis are extensively analysed. According to the study of turbo codes and LDPC codes, their iterative decoding performances have been evaluated and compared for the AWGN channel. Such iterative decoding algorithms have been analysed and optimised based on the characteristic of the erasure channel. Hence a further optimised iterative decoding algorithm, called look-up table decoder, has been introduced and investigated. The computational complexity of the proposed algorithm and iteration comparison have been identified and analysed between the standard turbo decoder and the LUT decoder.

As turbo Gallager codes named, such codes can be decoded either by a BCJR decoder or a BP decoder. We consider a class of turbo Gallager codes, which are based on the UMTS standard turbo codes structure as the testing codes for the hybrid decoding arrangements in the erasure channel. Such hybrid decoder including an iterative decoder followed by an "In-Place" decoder has been evaluated and compared in using different iterative decoders. The computational complexity in terms of optimum decoding for the proposed arrangements and an ML decoder has been analysed and compared. The comparison has shown that the proposed decoding arrangements have not only achieved the identical optimum performance, but also converged well with much reduced computational complexity. For the hybrid arrangement, under the affordable encoding complexity, the optimised iterative LUT decoder with TGC (15/13) provides an significantly improved performance comparing to other codes with iterative decoders. Thus the two new decoding schemes using turbo Gallager codes have been proposed for the erasure channel.

By following the turbo codes implementation over the erasure channel, the DVB-RCS turbo codes (11, 13/15) have also been evaluated over the erasure channel. The issue between symbol-based interleaver and bit-based interleaver has been realised that the probabilities of having state "11" and "00" are identical for bit-based interleaver due to the pairing property of each DVB-RCS turbo codes symbol. Since a BCJR decoder could not determine such mentioned output in pair, then a probabilistic based guessing algorithm has been introduced to help break such type of stopping sets. The improvement has been achieved by adopting the proposed algorithm for codes with bit-based interleaver. Furthermore, the stopping sets of DVB-RCS are analysed and represented in a new form of induced parity-check matrix.

Due to the existing fact that the iterative decoding performance of LDPC codes are seriously interfered by the BP stopping sets resulting in an error-floor, we have analysed the soft output from the BP decoder and proposed a new decoding arrangement to help lower the error-floor. The proposed decoding algorithm is conditioned on the soft output and an OSD-i decoder becomes optional. A series of analysis has been established, which support the benefit of the proposed decoding arrangement. And the numerical results have clearly shown its advantage and breakthrough of the lowered error-floor. Moreover, a class of cyclic LDPC codes has been evaluated and remarked by utilising n parity-check equations instead of $n - k$ independent parity-check equations, the significant improvements have shown the suitability of using BP decoding, especially with error-floor free. Due to the excellent performance of BP output, the proposed decoder with OSD-1 could easily make significantly better performance than an OSD decoder with higher order.

Inspired by Rosnes' remarkable paper on finding all the low weight stopping sets of LDPC codes, the modified fast algorithm with ideal trade off between the tightness of the lower bound and the number of nodes search has been introduced. The proposed lower bound features with low complexity computation on the intersected active rows, has provided a reasonable estimation to help limit the exhaustive tree search size. Some well known LDPC codes have been evaluated by adopting the efficient algorithm, the evaluated stopping sets and codeword weight spectra correctly match the previous published results. Moreover the WiMax LDPC codes in variety of code-rates and codeword lengths have been extensively evaluated.

By adapting the efficient exhaustive tree search algorithm, an extended Dorsch decoding algorithm has been realised and introduced by generating low weight codeword to help find the error vector based on the noisy received vector from the AWGN channel. An improvement to the proposed decoding algorithm to search the error-vector with lowest cross-correlation cost in stead of the codeword with highest cross-correlation has been devised. The simulation results have shown that the proposed decoding algorithm with certain constraints of the tree search size could achieve nearly optimum decoding performance for a range of linear block codes, which have very sparse parity-check matrix, in a moderate codeword length.

The major contributions of this research work include:

- A UMTS structure based class of turbo Gallager codes has been evaluated by using both iterative decoding algorithms including BCJR algorithm and BP algorithm for the erasure channel.

- The BCJR algorithm has been investigated and a look-up table based decoder algorithm has been introduced with much less computational complexity for the erasure channel.

- The hybrid decoding arrangements for turbo Gallager codes on the erasure channel have shown the optimum decoding performance with much less computational complexity, especially for TGC (15/13) with LUT decoder.

- DVB-RCS turbo codes have been analysed on the erasure channel. The issue between the symbol-based and bit-based interleavers has been raised and analysed. A solution to such issue by using bit-based interleavers has been proposed to help lower the error floor.

- By analysing the soft output from iterative BP decoder, the conditioned output has been devised to couple with OSD-i. Such converging arrangement has shown its great significance that the compound performances for a range of LDPC codes have successfully lowered the error floor, even error-floor free in some of the LDPC codes in moderate codeword length. Such significant improvement could have been achieved by simply applying the OSD-1 with k more codeword search. Moreover, because of the excellent iterative performance of LDPC codes, the arrangement of iterative decoder coupled by OSD-1 could achieve much better performance than OSD-2 alone, even OSD-3 for some of the LDPC codes, with much less computational complexity.

- Based on Rosnes' outstanding work in the stopping sets search, we have followed his novelty and devised a new bound algorithm with negligible computational complexity. Some well known LDPC codes have been evaluated and extended the results from previous work. Especially, the entire range of WiMax LDPC codes have also been explored, which could be a supplement for the use of WiMax standard.

- An extended decoding algorithm has been proposed by taking the stopping sets search algorithm as the basis. The new proposed decoding algorithm of searching the error vector with least cross correlation cost has been evaluated for a range of LDPC codes, which have very sparse parity-check matrix and reasonably moderate codeword lengths.

10.2 Future Work

A few immediate research directions might be realised according to this research contributions of the iterative decoding algorithm for turbo codes in erasure channel, DVB-RCS turbo codes stopping sets analysis, tree search based weight spectrum search and optimum decoding algorithm as follows.

- As the look-up table based decoding algorithm introduced in Section 4.5, the majority issue is the memory of the look-up table could be out of usage as the memory size of the shift register gets increased. Thus, enabling of using larger memory size of the shift register for LUT decoder might be a research direction to store multi-trellis sections as one unit instead of one trellis section as one unit in the look-up table. Thus the LUT decoder could process turbo codes in larger memory order to decode the erasure by looking up the shortened table, which is created in the main memory of the system. Such algorithm might require the help from the interleaver which could ideally rearrange a set of parity bits into a redundant area during the decoding process. The criteria of designing punctured turbo codes might be an ideal reference start.

- For DVB-RCS turbo codes, the symbol-based DVB-RCS interleaver code still achieves the best decoding performance at high erasure probability, thus a more advanced bit-interleaver by using DRP algorithm or S-random algorithm paired with the probabilistic algorithm might be a good direction to help achieve a better performance over the erasure channel at any erasure probability ϵ.

- According to the evaluations of a range of well known LDPC codes and a set of cyclic LDPC codes as shown in Chapter 7, the simulation results for most of the evaluated codes have shown significant improvements by applying the proposed decoder paired with OSD-1. Since most of these codes are in a relative short to moderate length, a further evaluation for a range of LDPC codes in larger size will be an ideal implementation in order to achieve a better convergent performance, the WiMax LDPC codes [1] with size of at least 576 and the well structured quasi-cyclic LDPC codes [3, 55], which have girths at least 6 and less encoding complexity, will be considered as the candidates.

- As Chapter 8 introduces a fast algorithm based on exhaustive tree search, the efficiency of the algorithm in terms of speed is mainly relied on the lower bound algorithm and the bit selection. Comparing to Rosnes' bound algorithm [102] by using "*simple method*" of linear programming, which could involve "pivot" inner loop operations, Algorithm 3 could not provide similar tight enough bound to reduce the search size, but it guarantees all the potential branches allowed to proceed over the entire code tree, especially with negligible computing complexity. Thus a research direction could be led to design a moderate bound algorithm, which could provide much tighter bound with small further computational complexity. Furthermore, the position selection criteria could be considered in the history of such position's activity over the different set of active rows.

- According to the research work in Chapter 9, a more sophisticated lower bound algorithm could be realised as an immediate research direction to provide tighter lower bound with less limitation. The intersected bits involved by the set of active rows are exponentially increased as the parity-check matrix gets slightly dense, which could be the increased weight of row. Thus the bit-combination search size to satisfy the threshold could easily be enlarged. The modified or specified linear programming, or inter programming, algorithm might be another appropriate candidate to help solve the multi-inequality problem. A faster algorithm is expected to help determine the optimum decoding performance for LDPC codes with much longer codeword length, for instance, the WiMax [1] LDPC codes with least 576 of 1/2 code-rate and the well structured quasi-cyclic LDPC codes [3, 55], which have girths at least 6 and less encoding complexity, will be considered as the candidates. .

References

[1] 802.16E, I.S. (2005). Wimax ldpc codes, air interface for fixed and mobile broadband wireless access systems, ieee std 802.16e-2005. Available from `http://standards.ieee.org/getieee802/download/802.16e-2005.pdf`. 18, 162, 163, 187, 188

[2] AMBROZE, M., TOMLINSON, M. & YANG, L. (2009). Exhaustive weight spectrum analysis of some well known ldpc codes. In *the 10th IEEE International Communication Theory and Applications (ISCTA 09)*, Lake District, UK. xix, 53, 151, 171, 173

[3] AMMAR, B., HONARY, B., KOU, Y., XU, J. & LIN, S. (2004). Construction of low-density parity-check codes based on balanced incomplete block designs. *IEEE Transactions on Information Theory*, **IT-50**, 1257–1269. 51, 187, 188

[4] ATKINSON & KENDAL.A (1989). *An Introduction to Numerical Analysis.* John Wiley & Sons, Inc., New York, USA, 2nd edn. 53

[5] BAHL, L.R., COCKE, J., JELINEK, F. & RAVIV, J. (1974). Optimal decoding of linear codes for minimising symbol error rate. *IEEE Transactions on Information Theory*, **IT-20**, 284–287. 3, 30, 41

[6] BAYES, T. & PRICE, R. (1763). An essay towards solving a problem in the doctrine of chances. *Transactions of Royal Society of London*, **53**, 370–418. 31

[7] BERLEKAMP, E.R. (1968). *Algebraic Coding Theory.* McGraw-Hill, New York, United States of America. 2

[8] BERLEKAMP, E.R., MCELIECE, R.J. & VAN TILBORG, H.C.A. (1978). On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, **24**, 384–386. 17, 150

[9] BERROU, C. & M.JEZEQUEL (1999). Non binary convolutional codes for turbo coding. *Electronics Letters*, **35**, 39–40. 85

[10] BERROU, C., GLAVIEUX, A. & THITIMAJSHIMA, P. (1993). Near shannon limit error-correcting coding: Turbo codes. In *Proc. IEEE International Conference on Communications*, 1064–1070, Geneva, Switzerland. 3, 37, 41, 43, 49, 76

[11] BERROU, C., DOUILLARD, C. & JEZEQUEL, M. (1999). Multiple parallel concatenation of circular recursive convolutional (crsc) codes. *Annals of Telecommun.*, **54**, 166–172. 16, 85

[12] BERROU, C., VATON, S., JEZEQUEL, M. & DOUILLARD, C. (2002). Computing the minimum distance of linear codes by the error impulse method. In *Proc of IEEE Globecom*, 1017–1020, Taipei, Taiwan. 18, 150

[13] BLAKE, I.F. & MULLIN, R. (1975). *The mathematical theory of coding*. Academic, New York. 51

[14] BOSE, R.C. & RAY-CHAUDHURI, D.K. (1960). On a class of error-correcting binary group codes. *Information and Control*, **3**, 68–79. 2

[15] BOUTILLON, E. & GNAEDIG, D. (2005). Maximum spread of d-dimentional multiple turbo codes. *IEEE Transactions on Communications*, **53**, 1237–1242. 3

[16] CAI, J., TJHAI, C., TOMLINSON, M., AMBROZE, M. & AHMED, M.Z. (2005). An efficient solution to packet loss: Erasure correcting codes. In *4^{th} IASTED, Communication Systems and Networks*, Spain. 16, 106, 121

[17] CANTEAUT, A. & CHABAUD, F. (1998). A new algorithm for finding minimum weight words in a linear code: Application to mceliece's cryptosystem and to narrow-sense bch codes of length 511. *IEEE Transactions on Information Theory*, **IT-44**, 367–378. 18, 150

[18] C.Berrou, Jezequel, M., Douillard, C. & Kerouedan, S. (2001). The advantages of non-binary turbo codes. In *IEEE Information Theory Workshop*, Caims, Australia. 16, 85

[19] Chase, D. (1972). A class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, **IT-18**, 170–182. 2

[20] Chatzigeorgiou, I., Rodrigues, M.R.D., Wassel, I.J. & Carrasco, R. (2005). A comparison of convolutional and turbo coding schemes for broadband fwa systems. In *Proc. 12th International Conference on Telecommunications*, Cape Town, South Africa. 64

[21] Chen, L., Djurdjevic, I., Xu, J., Lin, S. & Ghaffar, K.A. (2004). Construction of qc-ldpc codes based on the minimum-weight codewords of rs codes. In *IEEE International Symposium on Information Theory*, 239, Chicago, IL, USA. 51

[22] Chen, L., Xu, J., Djurdjevic, I. & Lin, S. (2004). Near-shannon-limit quasi-cyclic low-density parity-check codes. *IEEE Transactions on Communications*, **52**, 1038–1042. 55

[23] Chung, S.Y., Jr., G.D.F., Richardson, T.J. & Urbanke, R. (2001). On the design of low-density parity-check codes within 0.0045 db from the shannon limit. *IEEE Communication Letters.*, **5**, 58–60. 4, 17, 49, 50

[24] Colavolpe, G. (2004). Design and performance of turbo gallager codes. *IEEE Transactions on Communications*, **52**, 1901–1908. 4, 57, 58, 60, 61

[25] Costello, D.J. & Forney, G.D. (2007). Channel coding: The road to channel capacity. *Proceedings of the IEEE*, **95**, 1150–1177. 38

[26] Crozier, S. (2000). New high-spread high-distance interleavers for turbo codes. In *in Proc. of the 20th Biennial Symposium on Communications*, 3–7, Queens' University, Kingston, Ontario, Canada. 16

[27] CROZIER, S. (2000). New high-spread high-distance interleavers for turbo-codes. In *in Proc. 20th Biennial Symposium on Communications*, 3–7, Kingston, Ontario, Canada. 40

[28] CROZIER, S. & GUINAND, P. (2001). High-performance low-memory interleaver banks for turbo codes. In *in Proc. IEEE Vehicular Technology Conf.*, 2394–2398, Atlantic city, NJ, USA. 41, 44, 45

[29] CROZIER, S. & GUINAND, P. (2003). Distance upper bounds and true minimum distance results for turbo codes designed with drp interleaver. In *in Proc. 3rd Int. Symp. Turbo Codes Related Topics*, Brest, France. 41, 60, 68, 82, 108, 114, 115, 116

[30] CROZIER, S., LODGE, J., GUINAND, P. & HUNT, A. (1999). Performance of turbo codes with relative prime and golden interleaving strategies. In *in Proc. sixth International Mobile Satellite Conference (IMSC'99)*, 268–275, Ottawa, Canada. 40

[31] DANESHGARAN, F., LADDOMADA, M. & MONDIN, M. (2006). An algorithm for the computation of the minimum distance of ldpc codes. *European Transactions on Telecommunications*, **17**, 57–62. 18, 150

[32] DI, C., PROIETTI, D., TELATAR, I.E., RICHARDSON, T. & URBANKE, R. (2002). Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Transactions on Information Theory*, **48**, 1570–1579. 4, 17

[33] DOLINAR, S. & DIVSALAR, D. (1995). Weight distributions for turbo codes using random and nonrandom permutations. *TDA Progress Report*, **42**, 122. 16, 40

[34] DORSCH, B. (1974). A decoding algorithm for binary block codes and j -ary output channels. *IEEE Transactions on Information Theory*, **20**, 391–394. 18, 121

[35] ELIAS, P. (1955). Coding for noisy channels. *IRE Conv. Rec.*, **4**, 37–46. 2

[36] ELIAS, P. (1956). Coding for two noisy channels. In *Third London Symposium on Information Theory*, Academic Press, New York. 4, 10, 25

[37] FANO, R.M. (1963). A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, **IT-9**, 64–74. 2

[38] FORNEY, G.D. (1973). The viterbi algorithm. *Proceeding of the IEEE*, **61**, 268–278. 30

[39] FORNEY, G.D. (2001). Codes on graphs: normal realizations. *IEEE Transactions on Information Theory*, **47**, 520–548. 57

[40] FORNEY, JR., G.D. (1966). *Concatenated Codes*. MIT Press, Cambridge, MA, United States of America. 3

[41] FOSSORIER, M. (2004). Quasi-cyclic low-density parity-check codes from circulant permutation matrices. *IEEE Transactions on Information Theory*, **50**, 2346–2348. 51

[42] FOSSORIER, M.P.C. (2001). Iterative reliability-based decoding of low-density parity-check codes. *IEEE Journal on Select. Areas in Commun*, **19**, 908–917. 17, 123, 150

[43] FOSSORIER, M.P.C. (2002). Reliability-based soft-decision decoding with iterative information set reduction. *IEEE Transactions on Information Theory*, **48**, 3101–3106. 121

[44] FOSSORIER, M.P.C. & LIN, S. (1995). soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, **41**, 1379–1396. 121

[45] GALLAGER, R.G. (1963). *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press. 2, 3, 4, 17, 49, 50, 51, 53

[46] GOLAY, M.J.E. (1949). Notes on digital coding. *Proc. IRE*, **37**, 657. 1

[47] GOUNAI, S. & OHTSUKI, T. (2006). Lowering error floor of irregular ldpc codes by crc and osd algorithm. *Trans. of IEICE*, **E89-B**, 1–10. 121

[48] HAGENAUER, J. & HOEHER, P. (1989). A viterbi algorithm with soft-decision outputs and its applications. In *IEEE Globecom*, vol. 3, 1680–1686. 3

[49] HAMADA, N. (1973). On the p-rank of the incidence matrix of a balance or partial balanced incomplete block designs and its applications to error correcting codes. *Hiroshima Math. J.*, **3**, 153–226. 51

[50] HAMMING, R.W. (1950). Error detecting and error correcting codes. *Bell Syst. Tech. J.*, **29**, 147–160. 1

[51] HARTMANN, C.R.P. & RUDOLPH, L.D. (1976). An optimum symbol by symbol decoding rule for linear codes. *IEEE Transactions on Information Theory*, **IT-22**, 514–517. 18

[52] HIROTOMO, M., MOHRI, M. & MORII, M. (2005). A probabilistic computation method for the weight distribution of low-density parity-check codes. In *IEEE International Symposium on Information Theory*, 2166–2170, Adelaide, Australia. 18, 150

[53] HIROTOMO, M., KONISHI, Y. & MORII, M. (2008). A probabilistic algorithm for finding the minimum-size stopping sets of ldpc codes. In *Proc. IEEE Information Theory Workshop*, 66–70, Porto, Portugal. 18, 151

[54] HOCQUENGHEM, A. (1959). Codes correcteurs d'erreurs. *Chiffres*, **2**, 147–156. 2

[55] HONARY, B., MOINIAN, A. & AMMAR, B. (2005). Construction of well-structured quasi-cyclic low-density parity check codes. *IEE Proceedings*, **152**, 1081–1085. 187, 188

[56] HU, X.Y. & ELEFTHERIOU, E. (2006). A probabilistic subspace approach to the minimal stopping set problem. In *Proc. 4th Int. Symp. Turbo Codes and Related Topics*, Munich, Germanay. 161

[57] Hu, X.Y., Fossorier, M.P.C. & Eleftheriou, E. (2004). On the computation of the minimum distance of low-density parity-check codes. In *Proc of IEEE Int. Conf. Commun. (ICC)*, vol. 2, 776–771, Paris, France. 18, 150

[58] Hu, X.Y., Eleftheriou, E. & Arnold, D.M. (2005). Regular and irregular progressive edge-growth tanner graphs. *IEEE Transactions on Information Theory*, **51**, 386–398. 136, 161

[59] Institute, E.T.S. (2003). Digital video broadcasting (dvb); interaction channel for satellite distribution systems. ETSI EN 301 790 v1.3.1 (2003-03). 16, 85, 91

[60] Johnson, S. & Weller, S. (2001). Regular low-density parity-check codes from combinatorial designs. In *IEEE Information Theory Workshop*, 90–92. 51

[61] Kou, Y., Lin, S. & Fossorier, M. (2000). Low density parity check codes based on finite geometries: a rediscovery. In *IEEE International Symposium on Information Theory*, Sorrento, Italy. 51

[62] Kou, Y., Lin, S. & Fossorier, M. (2001). Low density parity check codes based on finite geometries: a rediscovery and new results. *IEEE Transactions on Information Theory*, **47**, 2711–2736. 4, 138, 176

[63] Krishnan, K.M. & Shankar, P. (2007). Computing the stopping distance of a tanner graph is np-hard. *IEEE Transactions on Information Theory*, **53**, 2278–2280. 18, 151

[64] Kschischang, F.R. & Sorokine, V. (1994). On the trellis structure of block codes. In *IEEE International Symposium on Information Theory*, Trondheim, Norway. 8

[65] Kurkoski, B.M. (2004). *Algorithms and Schedules for Turbo Equlization.* Ph.D dissertation, Department of Electrical Engineering (Communication Theory and Systems), University of California, San Diego, United States of America. 16

[66] Kurkoski, B.M., Siegel, P.H. & Wolf, J.K. (2003). Exact probability of erasure and a decoding algorithm for convolutional codes on the binary erasure channel. In *Proc. IEEE GLOBECOM*, vol. 3, 1741–1745, San Francisco, CA. 16, 77

[67] Lan, L., Zeng, L., Lei, Y.Y., Chen, L., Lin, S. & Abdel-Ghaffar, K. (2007). Construction of quasi-cyclic ldpc codes for awgn and binary erasure channels: A finite field approach. *IEEE Transactions on Information Theory*, **53**, 2429–2458. 161

[68] Leon, J.S. (1988). A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, **IT-34**, 1354–1359. 18, 150

[69] Li, Z., Chen, L., Zeng, L., Lin, S. & Fong, W. (2005). Efficient encoding of quasi-cyclic low-density parity-check codes. In *IEEE GLOBECOM Conf.*. 51

[70] Lin, S. (1970). *An introduction to ERROR-CORRECTING CODES*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 9

[71] Lin, S. & Costello, Jr., D.J. (2004). *Error Control Coding: Fundamentals and Applications*. Pearson Education, Inc, 2nd edn. 14, 40

[72] Lin, S., Xu, J., Djurdjevic, I. & Tang, H. (2002). Hybrid construction of ldpc codes. In *in Proc. 40th Annu. Conf. Communication, Control, and Computing*, Monticello, IL, USA. 4, 51

[73] Lin, S., Chen, L., Xu, J. & Djurdjevic, I. (2003). Near shannon limit quasicyclic low-density parity-check codes. In *in Proc. 2003 IEEE GLOBECOM Conf.*, San Francisco, IL, USA. 4, 51

[74] Lodge, J., Hoeher, P. & Hagenauer, J. (1992). The decoding of multidimensional codes using separable map 'filters'. In *in Proc. 16th Biennial Symposium on Communications*, 343–346, Kingston, ON, Canada. 3

[75] Luby, M., Mitzenmacher, M., Shokrollahi, A., D.Spielman & V.Stemann (1997). Practical loss-resilient codes. *29th annual ACM Symp.Theory of Computing*, 150–159. 4, 69

[76] Luby, M.G., Mitzenmacher, M., Shokrollahi, M.A. & Spielman, D.A. (2001). Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, **47**, 569–584. 4, 15, 157, 169

[77] 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**, 585–598. 4

[78] Ma, H.H. & Wolf, J.K. (1986). On tail biting convolutional codes. *IEEE Transactions on Communications*, **34**, 104–111. 8, 36, 85

[79] MacKay, D.J.C. (1999). Encyclopedia of sparse graph codes. Available: `http://www.inference.phy.cam.ac.uk//codes/data.html`. 161, 175

[80] MacKay, D.J.C. (1999). Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, **45**, 399–431. 49, 50, 51, 55, 134, 176

[81] MacKay, D.J.C. & Neal, R.M. (1997). Near shannon limit performance of low-density parity-check codes. *Electronic Letters*, **33**, 457–458. 3, 4, 50

[82] Massey, J.L. (1963). *Threshold Decoding*. MIT Press, Cambridge, MA, United States of America. 2, 3, 18

[83] Massey, J.L. (1969). Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, **IT-15**, 122–127. 2

[84] McEliece, R.J., Mackay, D.J. & Cheng, J. (1998). Turbo decoding as an instance of pearl's belief propagation algorithm. *IEEE Journal on Selected Areas in Communications*, **16**, 140–152. 57

[85] Michelson, A.M. & Levesque, A.H. (1985). *Error-Control Techniques for Digital Communication*. John Wiley & Sons, Inc., United States of America. 6, 10

[86] Morelos-Zaragoza, R.H. (2002). *The Art of Error Correcting Coding.* John Wiley & Sons, Inc., Chichester, England. 7, 8, 36

[87] Muller, D.E. (1954). Application of boolean algebra to switching circuit design and to error detection. *IRE Trans. Electron. Comput.*, **EC-3**, 6–12. 1

[88] Pearl, J. (1988). *Probabilisitic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers Inc., Morgam Kaufmann, San Mateo, CA. 49

[89] Perotti, A. & S.Benedetto (2004). A new upper bound on the minimum distance of turbo codes. *IEEE Transactions on Information Theory*, **50**, 2985–2997. 4

[90] Peterson, W.W. (1960). Encoding and error-correction procedures for the bose-chaudhuri codes. *IRE Trans. Inform. Theory*, **IT-6**, 459–470. 2

[91] Peterson, W.W. (1961). *Error-Correcting Codes.* MIT Press, Cambridge, MA. 2

[92] Proakis, J.G. (2000). *Digital Communications.* McGraw-Hill, New York, 4th edn. 126

[93] Reed, I.S. (1954). A class of multiple-error-correcting codes and the decoding scheme. *IRE Trans. Inform. Theory*, **IT-4**, 38–49. 1

[94] Reed, I.S. & Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, **8**, 300–304. 2

[95] Richardson, T.J., Shokrollahi, A. & Urbanke, R.L. (2001). Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, **47**, 619–637. 4, 49, 50, 127

[96] Richter, G. (2006). Finding small stopping sets in the tanner graph of ldpc codes. In *Proc. 4^{th} Int. Symp. on Turbo Codes & Related Topics*, Munich, Germany. 18, 151

[97] Robertson, P., Villebrun, E. & Hoeher, P. (1995). A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain. In *IEEE Int. Conf. Comm.*, vol. 2, 1009–1013, Seattle, WA, USA. 64

[98] Roman, S. (2000). *Introduction to Coding and Information Theory*. Springer. 108

[99] Rosnes, E. & O.Ythehus (2005). Improved algorithms for the determination of turbo-code weight distributions. *IEEE Transactions on Communications*, **53**, 20–26. 4, 16

[100] Rosnes, E. & O.Ythehus (2007). Turbo decoding on the binary erasure channel: finite-length analysis and turbo stopping sets. *IEEE Transactions on Information Theory*, **53**, 4059–4075. 4, 16, 18, 151

[101] Rosnes, E. & Ytrehus, O. (2005). Turbo stopping sets: The uniform interleaver and efficient enumeration. In *IEEE International Symposium on Information Theory*, 1251–1255, Adelaide, SA, Australia. 18, 151

[102] Rosnes, E. & Ytrehus, O. (2007). An algorithm to find all small-size stopping sets of low-density parity-check matrices. In *IEEE International Symposium on Information Theory*, Nice, France. 18, 151, 155, 158, 160, 161, 188

[103] Rosnes, E. & Ytrehus, O. (2009). An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices. *IEEE Transactions on Information Theory*, **55**, 4167–4178. 18, 163

[104] S.Benedetto & Montorsi, G. (1996). Unveilling turbo codes: some results on parallel concatenated coding schemes. *IEEE Transactions on Information Theory*, **42**, 409–428. 40

[105] S.Benedetto, D. Divsalar, G.M. & Pollara, F. (1998). Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding. *IEEE Transactions on Information Theory*, **44**, 909–926. 3

[106] SCHALKWIJK, J.P.M. & VINCK, A.J. (1975). Syndrome decoding of convolutional codes. *IEEE Transactions on Communications*, **23**, 789–792. 77

[107] SCHWARTZ, M. & VARDY, A. (2006). On the stopping distance and the stopping redundancy of codes. *IEEE Transactions on Information Theory*, **52**, 922–932. 153

[108] SHANNON, C.E. (1948). A mathematical theory of communication. *Bell Syst. Tech. J.*, **27**, 379–423, 623–656. 1, 9

[109] SILVERMAN, R.A. & BALSER, M. (1954). Coding for constant-data-rate systems. *IRE Trans. Inform. Theory*, **PGIT-4**, 50–63. 2

[110] SIPSER, M. & SPIELMAN, D.A. (1996). Expander codes. *IEEE Transactions on Information Theory*, **42**, 1710–1722. 50

[111] SKLAR, B. (2001). *Digital Communications: Fundamentals and Applications*. Prentice Hall, Upper Saddle River, New Jersey, 2^{nd} edn. 30

[112] SOLJANIN, E. & OFFER, E. (2001). Ldpc codes: a group algebra formulation. In *In Proc. Internat. Workshop on Coding and Cryptography WCC*. 17

[113] SOLJANIN, E. & OFFER, E. (2003). Bit-optimal decoding of codes whose tanner graphs are trees. *Discrete Applied Mathematics*, **128**, 293–303. 17

[114] SPIELMAN, D.A. (1996). Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, **42**, 1723–1731. 50

[115] STERN, J. (1988). A method for finding codewords of small weight. *Proceedings of the 3rd International Colloquium on Coding Theory and Applications in Lecture Notes in Computer Science; Vol. 388*, 106–113. 18, 150

[116] SWEENY, P. (2002). *Error Control Coding: from theory to practice*. John Wiley & Sons, Inc. 52

[117] TANNER, R.M. (1981). A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, **IT-27**, 533–547. 3, 58, 152

[118] TANNER, R.M., SRIDHARA, D. & FUJA, T. (2001). A class of group-structured ldpc codes. In *Proc. Int. Symp. on Commun. Theroy and Appl. (ISCTA)*, Ambleside, England. 53, 127, 129, 132, 153, 161, 175, 177

[119] TJHAI, C., TOMLINSON, M., AMBROZE, M. & AHMED, M. (2005). Cyclotomic idempotent-based binary cyclic codes. *Electron. Lett.*, **41**, 341–343. 130, 137, 175

[120] TOMLINSON, M., CAI, J., TJHAI, C., AMBROZE, M. & AHMED, M. (2004). System for Correcting Deleted or Missing Symbols. UK Patent Application 0428042.6. 106

[121] TOMLINSON, M., TJHAI, C. & AMBROZE, M. (2007). Extending the dorsch decoder towards achieving maximum-likelihood decoding for linear codes. *IET Communications*, **1**, 479–488. 18, 121, 124, 127, 167, 170

[122] TOMLINSON, M., TJHAI, C., CAI, J. & AMBROZE, M. (2007). Analysis of the distribution of the number of erasures correctable by a binary linear code and the link to low-weight codewords. *IET Communications*, **1**, 539–548. 18, 108, 110

[123] VALEMBOIS, A. & FOSSORIER, M.P.C. (2004). Box and match techniques applied to soft-decision decoding. *IEEE Transactions on Information Theory*, **50**, 796–810. 121

[124] VARDY, A. (1997). The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, **43**, 1757–1766. 17, 150

[125] VASIC, B. & O.MILENKOVIC (2004). Combinatorial constructions of low-density parity-check codes for iterative decoding. *IEEE Transactions on Information Theory*, **50**, 1156–1176. 51

[126] VITERBI, A.J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, **IT-13**, 260–269. 2, 29

[127] 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**, 1976–1999. 18, 151, 155, 161

[128] WELLS, R.B. (1999). *Applied Coding and Information Theory for Engineers*. Prentice Hall. 36

[129] WOZENCRAFT, J.M. & REIFFEN, B. (1961). *Sequential Decoding*. MIT Press, Cambridge, MA, United States of America. 2

[130] WU, P.Y. (2001). On the complexity of turbo decoding algorithms. In *Proc. 53rd Vehicular Technology Conference,*, vol. 2, 1439–1443. 16, 64

[131] YANG, L., AMBROZE, M. & TOMLINSON, M. (2008). Comparison of decoding turbo gallager codes in hybrid decoding arrangements with different iterative decoders over the erasure channel. In *the 11th IEEE International Conference on Communications Systems (ICCS)*, Guangzhou, China. xix, 63, 105

[132] YANG, L., TOMLINSON, M. & AMBROZE, M. (2010). Decoding low-density parity-check codes with error-floor free over the awgn channel. In *the 2010 IEEE International Conference on Wireless Communication, Networking and Information Security (WCNIS2010)*, Beijing, China. xix, 122

[133] YANG, L., TOMLINSON, M. & AMBROZE, M. (2010). Extended optimum decoding for ldpc codes based on exhuastive tree search algorithm. In *the 12th IEEE International Conference on Communications Systems (ICCS)*, Singapore. xix

[134] YUAN, J., VUCETIC, B. & FENG, W. (1999). Combined turbo codes and interleaver design. *IEEE Transactions on Communications*, **47**, 484–487. 16

Appendix A

Proof on Relationship between Received Bit and Soft Output

The relationship between received bit r_m affected by the noise factor σ in the AWGN channel and the probability in ratio of $\frac{p(c_m=0)}{p(c_m=1)}$ can be simply expressed as

$$\frac{p(c_m = 0|r_m)}{p(c_m = 1|r_m)} = \frac{\epsilon^{-(r_m+1)^2/2\sigma^2}}{\epsilon^{-(r_m-1)^2/2\sigma^2}} \tag{A.1}$$

By adding logarithm on both sides,

$$\log \frac{p(c_m = 0|r_m)}{p(c_m = 1|r_m)} = \frac{-(r_m + 1)^2 + (r_m - 1)^2}{2\sigma^2} = \frac{-2r_m}{\sigma^2} \tag{A.2}$$

Since σ is a constant, then r_m is proportional to $\log \frac{p(c_m=0|r_m)}{p(c_m=1|r_m)}$.

Appendix B

Sample of Stopping Set Representation over Parity-check Matrix

Erasure Pattern: Hamming weight = 16, Code = DVB-RCS Turbo Code (384, 128, 26), Interleaver=DRP-rp13-rs17

The set of erasure indices: 51 52 53 54 55 57 115 116 119 120 121 179 243 244 313 371

The related parity-check equations:
(R=1, R'=48) 48 50 $\underline{51}$ 304 307 369 $\underline{371}$ −1 (2/7)
(R=2, R'=49) 49 $\underline{51}$ $\underline{52}$ 305 308 370 372 −1 (2/7)
(R=3, R'=50) 50 $\underline{52}$ $\underline{53}$ 306 309 $\underline{371}$ 373 −1 (3/7)
(R=4, R'=51) $\underline{51}$ $\underline{53}$ $\underline{54}$ 307 310 372 374 −1 (3/7)
(R=5, R'=52) $\underline{52}$ $\underline{54}$ $\underline{55}$ 308 311 373 375 −1 (3/7)
(R=6, R'=53) $\underline{53}$ $\underline{55}$ 56 309 312 374 376 −1 (2/7)
(R=7, R'=54) $\underline{54}$ 56 $\underline{57}$ 310 $\underline{313}$ 375 377 −1 (3/7)
(R=8, R'=55) $\underline{55}$ $\underline{57}$ 58 311 314 376 378 −1 (2/7)
(R=9, R'=57) $\underline{57}$ 59 60 $\underline{313}$ 316 378 380 −1 (2/7)
(R=10, R'=112) 112 114 $\underline{115}$ 304 305 307 368 369 370 $\underline{371}$ −1 (2/10)
(R=11, R'=113) 113 $\underline{115}$ $\underline{116}$ 305 306 308 369 370 $\underline{371}$ 372 −1 (3/10)
(R=12, R'=114) 114 $\underline{116}$ 117 306 307 309 370 $\underline{371}$ 372 373 −1 (2/10)
(R=13, R'=115) $\underline{115}$ 117 118 307 308 310 $\underline{371}$ 372 373 374 −1 (2/10)
(R=14, R'=116) $\underline{116}$ 118 $\underline{119}$ 308 309 311 372 373 374 375 −1 (2/10)
(R=15, R'=117) 117 $\underline{119}$ $\underline{120}$ 309 310 312 373 374 375 376 −1 (2/10)
(R=16, R'=118) 118 $\underline{120}$ $\underline{121}$ 310 311 $\underline{313}$ 374 375 376 377 −1 (3/10)

B. SAMPLE OF STOPPING SET REPRESENTATION OVER PARITY-CHECK MATRIX

(R=17, R'=119) $\underline{119}$ $\underline{121}$ 122 311 312 314 375 376 377 378 –1 (2/10)
(R=18, R'=120) $\underline{120}$ 122 123 312 $\underline{313}$ 315 376 377 378 379 –1 (2/10)
(R=19, R'=121) $\underline{121}$ 123 124 $\underline{313}$ 314 316 377 378 379 380 –1 (2/10)
(R=20, R'=176) 176 178 $\underline{179}$ 262 304 $\underline{313}$ 332 –1 (2/7)
(R=21, R'=177) 177 $\underline{179}$ 180 275 314 354 $\underline{371}$ –1 (2/7)
(R=22, R'=179) $\underline{179}$ 181 182 $\underline{313}$ 333 342 $\underline{371}$ –1 (3/7)
(R=23, R'=240) 240 242 $\underline{243}$ 262 275 304 $\underline{313}$ 332 354 383 –1 (2/10)
(R=24, R'=241) 241 $\underline{243}$ $\underline{244}$ 275 304 314 332 345 354 $\underline{371}$ –1 (3/10)
(R=25, R'=242) 242 $\underline{244}$ 245 263 292 $\underline{313}$ 332 345 354 $\underline{371}$ –1 (3/10)
(R=26, R'=243) $\underline{243}$ 245 246 292 $\underline{313}$ 314 332 333 342 $\underline{371}$ –1 (3/10)
(R=27, R'=244) $\underline{244}$ 246 247 263 292 301 314 320 342 $\underline{371}$ –1 (2/10)

The number of H-Matrix equations involved with each erasure in format of (Index of Erasure: Number of Equations) are shown as follows:
51: 3
52: 3
53: 3
54: 3
55: 3
57: 3
115: 3
116: 3
119: 3
120: 3
121: 3
179: 3
243: 3
244: 3
313: 10
371: 12

Appendix C

WiMax LDPC Codes Weight Spectrum (Standard)

The results of WiMax LDPC codes in code rate of 1/2 are given in Table C.1 with threshold up to $s_{min} + 4$. The results of WiMax LDPC codes in code rates of 2/3A and 2/3B are given in Table C.2 and Table C.3 with threshold up to $s_{min} + 4$. The results of WiMax LDPC codes in code rates of 3/4A and 3/4B are given in Table C.4 and Table C.5 with threshold up to $s_{min} + 4$.

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	13	24(24)	0(0)	0(0)	24(24)	0(0)
672	18	56(0)	140(56)	56(56)	308(84)	420(168)
768	18	32(0)	0(0)	96(64)	128(32)	192(96)
864	19	36(36)	36(36)	144(0)	324(36)	828(180)
960	19	120(80)	120(40)	160(0)	280(160)	400(120)
1056	19	44(0)	0(0)	44(44)	132(0)	220(88)
1152	19	48(48)	0(0)	0(0)	0(0)	0(0)
1248	19	52(0)	0(0)	0(0)	52(52)	52(0)
1344	23	112(112)	56(0)	280(224)	560(224)	1008(280)
1440	24	60(0)	60(0)	60(0)	180(60)	720(300)
1536	20	64(64)	0(0)	0(0)	64(64)	64(0)
1632	27	68(68)	408(0)	476(136)	748(272)	1836(816)
1728	21	72(72)	0(0)	0(0)	0(0)	0(0)
1824	19	76(76)	0(0)	0(0)	0(0)	0(0)
1920	25	160(80)	240(80)	240(240)	400(160)	1040(160)
2016	27	84(84)	84(84)	756(168)	518(182)	1260(336)
2112	28	264(264)	88(0)	440(264)	616(264)	1144(440)
2208	23	92(92)	0(0)	0(0)	0(0)	0(0)
2304	28	96(0)	96(0)	288(0)	288(96)	624(336)

Table C.1: WiMax LDPC Codes $R_c = 1/2$

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	10	72(48)	24(24)	192(120)	792(360)	2724(1118)
672	9	28(28)	28(0)	84(28)	336(84)	1512(588)
768	8	64(64)	0(0)	0(0)	128(128)	160(128)
864	9	36(0)	36(0)	72(72)	72(0)	504(216)
960	12	80(0)	440(280)	1400(400)	3080(840)	8440(2320)
1056	10	44(44)	0(0)	44(44)	44(0)	352(88)
1152	13	48(0)	48(48)	144(144)	768(240)	1968(1056)
1248	13	104(104)	208(0)	572(312)	1352(572)	3484(1404)
1344	14	224(168)	168(112)	336(112)	1736(1008)	5012(1764)
1440	13	180(180)	60(0)	300(120)	240(0)	1380(780)
1536	12	64(64)	0(0)	0(0)	192(64)	448(320)
1632	12	68(0)	136(68)	0(0)	204(136)	340(204)
1728	14	72(0)	216(144)	144(72)	432(216)	2196(1008)
1824	15	76(76)	228(152)	228(152)	988(760)	2660(1064)
1920	14	80(0)	80(80)	160(160)	320(240)	960(640)
2016	15	84(84)	252(0)	168(168)	504(336)	1680(840)
2112	15	88(88)	0(0)	176(88)	352(176)	1144(352)
2208	15	92(92)	0(0)	92(92)	460(276)	1012(644)
2304	15	96(96)	0(0)	96(96)	480(384)	768(384)

Table C.2: WiMax LDPC Codes $R_c = 2/3$A

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	10	24(0)	0(0)	48(48)	264(24)	1056(168)
672	11	28(28)	56(28)	112(56)	196(0)	1540(252)
768	13	32(0)	64(64)	256(64)	760(168)	2176(256)
864	15	72(0)	315(45)	648(72)	2434(400)	7776(828)
960	14	40(0)	40(40)	295(95)	880(80)	2460(440)
1056	15	44(44)	88(0)	0(0)	440(176)	2640(792)
1152	16	96(48)	432(48)	1008(336)	1824(240)	4896(480)
1248	15	52(52)	0	104(104)	156(104)	728(312)
1344	16	63(63)	56(56)	196(56)	560(168)	1568(196)
1440	17	120(60)	120(0)	300(240)	975(195)	1200(300)
1536	15	64(64)	0(0)	0(0)	0(0)	128(0)
1632	18	204(68)	272(68)	289(221)	952(136)	1224(136)
1728	15	72(72)	0(0)	0(0)	72(0)	0(0)
1824	15	76(76)	0(0)	0(0)	0(0)	0(0)
1920	16	80(80)	80(0)	160(0)	0(0)	240(0)
2016	15	84(84)	0(0)	0(0)	0(0)	84(84)
2112	16	88(88)	88(0)	0(0)	0(0)	264(0)
2208	20	92(92)	92(0)	92(0)	276(92)	1012(276)
2304	15	96(96)	0(0)	0(0)	0(0)	0(0)

Table C.3: WiMax LDPC Codes $R_c = 2/3$B

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	9	72(0)	228(84)	1488(336)	7032(1416)	35808(4512)
672	8	28(0)	0(0)	112(56)	476(168)	2352(602)
768	10	128(64)	512(256)	1744(512)	6336(1152)	32592(4032)
864	11	180(0)	468(234)	2376(324)	9360(1134)	38232(3888)
960	12	140(80)	360(120)	2180(540)	8840(2080)	37440(5440)
1056	12	22(0)	308(88)	814(242)	3300(792)	13926(2376)
1152	10	48(0)	0(0)	24(0)	240(48)	528(288)
1248	12	26(0)	156(52)	260(104)	2132(416)	6500(1092)
1344	12	28(0)	112(0)	224(168)	952(280)	1960(336)
1440	12	90(60)	60(0)	180(60)	372(192)	960(420)
1536	12	32(0)	0(0)	128(128)	192(64)	1152(192)
1632	12	34(0)	68(68)	0(0)	0(0)	612(68)
1728	12	36(0)	0(0)	0(0)	0(0)	72(0)
1824	12	38(0)	76(76)	0(0)	76(76)	228(0)
1920	12	40(0)	80(0)	160(0)	240(0)	240(0)
2016	12	42(0)	0(0)	0(0)	0(0)	0(0)
2112	12	44(0)	0(0)	0(0)	88(88)	0(0)
2208	12	46(0)	0(0)	0(0)	0(0)	0(0)
2304	12	48(0)	0(0)	0(0)	0(0)	0(0)

Table C.4: WiMax LDPC Codes $R_c = 3/4$A

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	8	48(48)	96(48)	408(240)	1942(816)	8252(3076)
672	8	14(14)	0(0)	140(112)	588(252)	3626(1568)
768	9	64(64)	96(64)	416(224)	2613(1232)	9625(2624)
864	10	36(0)	252(144)	1188(612)	4752(1944)	20064(6642)
960	11	400(280)	760(320)	4000(1840)	13653(4120)	55141(14360)
1056	9	44(44)	0(0)	220(176)	616(352)	1760(748)
1152	11	144(48)	528(240)	1440(624)	5532(2064)	19016(7344)
1248	9	52(52)	52(52)	52(52)	312(156)	988(416)
1344	12	560(392)	616(224)	1736(616)	7553(2968)	28043(11172)
1440	10	60(60)	60(60)	130(10)	540(240)	2190(810)
1536	11	64(64)	128(128)	128(64)	960(640)	3648(1408)
1632	13	272(204)	748(544)	2992(1564)	8730(3536)	27731(9248)
1728	12	72(0)	576(432)	576(216)	2520(936)	7200(3528)
1824	12	228(228)	380(304)	988(836)	2812(836)	9500(3724)
1920	10	80(80)	0(0)	0(0)	0(0)	640(480)
2016	11	84(0)	84(84)	336(168)	546(294)	1260(588)
2112	12	44(0)	0(0)	0(0)	88(88)	0(0)
2208	13	184(92)	92(92)	1012(644)	1610(874)	6072(2300)
2304	12	16(16)	96(96)	0(0)	672(480)	1824(768)

Table C.5: WiMax LDPC Codes $R_c = 3/4$B

The results of WiMax LDPC codes in code rate of 5/6 are given in Table C.6 up to s_{min}+4.

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	5	24(24)	24(0)	600(312)	3276(1248)	31320(10272)
672	6	28(0)	308(168)	2170(924)	15484(5040)	146566(37590)
768	7	160(128)	1200(688)	6816(3264)	55408(16048)	500000(108960)
864	7	324(144)	856(360)	5148(1944)	32634(10332)	264528(64332)
960	7	120(80)	500(300)	2880(1440)	18120(6360)	142840(38760)
1056	7	132(44)	286(242)	2200(836)	13156(4620)	87208(24024)
1152	7	192(96)	384(240)	2400(1200)	15768(5616)	93312(27264)
1248	7	156(156)	338(338)	1820(676)	9880(4056)	59852(18252)
1344	7	168(112)	252(84)	1512(840)	9744(4844)	51296(16688)
1440	7	180(180)	600(360)	2520(1020)	10350(3510)	54720(15840)
1536	7	128(128)	352(224)	704(384)	3520(1344)	19008(6016)
1632	7	68(68)	34(34)	204(68)	816(680)	5372(2040)
1728	7	72(0)	72(72)	216(144)	2088(720)	9936(3960)
1824	8	38(38)	152(0)	874(570)	5016(2280)	24453(8721)
1920	7	160(80)	160(160)	240(80)	1040(160)	6720(2640)
2016	7	84(84)	168(0)	252(84)	588(336)	3612(1596)
2112	8	132(132)	176(88)	880(264)	3344(1848)	20306(7062)
2208	8	138(46)	0(0)	460(0)	1564(736)	7337(2415)
2304	9	192(192)	288(0)	1920(672)	8616(2424)	43296(13632)

Table C.6: WiMax LDPC Codes $R_c = 5/6$

Appendix D

WiMax LDPC Codes Weight Spectra (Modulo)

The results of WiMax LDPC codes in code rate of 1/2 are given in Table D.1 with threshold up to $s_{min} + 4$. The results of WiMax LDPC codes in code rates of 2/3A and 2/3B are given in Table D.2 and Table D.3 up to $s_{min} + 4$. The results of WiMax LDPC codes in code rates of 3/4A and 3/4B are given in Table D.4 and Table D.5 up to $s_{min} + 4$.

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	15	8(0)	288(192)	72(0)	288(72)	816(216)
672	13	28(28)	0(0)	0(0)	0(0)	28(0)
768	14	32(32)	32(0)	0(0)	32(0)	32(0)
864	15	36(0)	36(0)	36(36)	126(36)	216(36)
960	14	40(40)	0(0)	0(0)	0(0)	0(0)
1056	18	44(0)	0(0)	44(44)	44(0)	44(44)
1152	16	48(48)	0(0)	0(0)	0(0)	96(0)
1248	17	156(52)	208(52)	364(104)	156(156)	104(52)
1344	22	112(0)	112(0)	560(168)	392(168)	1092(252)
1440	24	120(60)	60(0)	180(120)	480(60)	780(120)
1536	23	64(0)	64(64)	128(0)	192(192)	320(128)
1632	24	68(68)	68(68)	136(136)	544(204)	544(204)
1728	19	72(72)	0(0)	0(0)	0(0)	0(0)
1824	20	76(0)	0(0)	76(76)	0(0)	114(114)
1920	23	160(0)	0(0)	80(80)	80(0)	320(240)
2016	26	84(84)	0(0)	0(0)	252(84)	252(0)
2112	20	88(88)	0(0)	0(0)	88(0)	0(0)
2208	24	92(92)	0(0)	184(92)	184(92)	184(92)
2304	28	96(0)	96(0)	288(0)	288(96)	624(336)

Table D.1: WiMax LDPC Codes $R_c = 1/2$ (Modulo)

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	10	72(48)	24(24)	192(120)	792(360)	2748(1188)
672	9	28(28)	28(0)	84(28)	336(84)	1512(588)
768	8	64(64)	0(0)	0(0)	160(128)	160(128)
864	9	36(0)	36(0)	108(72)	108(0)	576(216)
960	12	80(0)	440(280)	1400(400)	3080(840)	8440(2320)
1056	10	44(44)	0(0)	44(44)	44(0)	352(88)
1152	13	48(0)	48(48)	144(144)	768(240)	1968(1056)
1248	13	104(104)	208(0)	572(312)	1352(572)	3484(1404)
1344	14	224(168)	168(112)	336(112)	1736(1008)	5012(1764)
1440	13	180(180)	60(0)	300(120)	240(0)	1380(780)
1536	12	64(64)	0(0)	0(0)	192(64)	448(320)
1632	12	68(0)	136(68)	0(0)	204(136)	340(204)
1728	14	72(0)	216(144)	144(72)	432(216)	2196(1008)
1824	15	76(76)	228(152)	228(152)	988(760)	2660(1064)
1920	14	80(0)	80(80)	160(160)	320(240)	960(640)
2016	15	84(84)	252(0)	168(168)	504(336)	1680(840)
2112	15	88(88)	0(0)	176(88)	352(176)	1144(352)
2208	15	92(92)	0(0)	92(92)	460(276)	1012(644)
2304	15	96(96)	0(0)	96(96)	480(384)	768(384)

Table D.2: WiMax LDPC Codes $R_c = 2/3$A (Modulo)

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	10	36(12)	72(0)	176(56)	864(240)	3336(672)
672	8	7(7)	0(0)	0(0)	140(28)	294(126)
768	12	64(64)	128(64)	384(96)	1120(224)	3128(408)
864	13	72(0)	270(144)	732(156)	1449(261)	5400(504)
960	13	40(40)	80(40)	120(40)	400(160)	1280(320)
1056	14	132(44)	264(220)	924(132)	1848(352)	6908(704)
1152	10	48(0)	0(0)	0(0)	0(0)	48(0)
1248	15	52(52)	104(52)	260(52)	208(0)	1092(104)
1344	15	112(112)	7(7)	224(168)	504(224)	2016(168)
1440	15	60(60)	0(0)	0(0)	150(90)	540(240)
1536	15	64(64)	64(64)	64(64)	64(64)	192(0)
1632	15	204(136)	136(0)	408(136)	340(136)	1292(408)
1728	15	72(72)	0(0)	0(0)	72(72)	432(72)
1824	15	76(76)	76(76)	76(0)	152(76)	228(228)
1920	15	80(80)	0(0)	80(0)	80(80)	80(0)
2016	15	84(84)	0(0)	0(0)	84(84)	84(0)
2112	15	88(88)	0(0)	0(0)	88(0)	176(0)
2208	15	92(92)	0(0)	0(0)	0(0)	0(0)
2304	15	96(96)	0(0)	0(0)	0(0)	0(0)

Table D.3: WiMax LDPC Codes $R_c = 2/3$B (Modulo)

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	6	24(0)	0(0)	0(0)	24(24)	192(96)
672	8	28(28)	84(28)	336(224)	1204(280)	6048(980)
768	9	32(0)	128(32)	576(128)	2192(384)	9472(1024)
864	10	72(36)	360(36)	612(144)	2232(576)	9576(1044)
960	10	40(0)	280(160)	560(120)	2160(280)	8500(1420)
1056	11	88(0)	132(0)	396(176)	1122(132)	4356(792)
1152	6	48(0)	0(0)	0(0)	0(0)	48(48)
1248	11	52(52)	104(52)	156(0)	520(156)	2028(520)
1344	9	56(0)	56(56)	56(0)	448(168)	840(168)
1440	14	180(0)	660(180)	1860(360)	6000(900)	22200(2100)
1536	12	64(64)	128(0)	128(0)	704(128)	2240(192)
1632	12	68(0)	0(0)	0(0)	272(204)	476(136)
1728	14	144(0)	144(0)	216(0)	1008(288)	3696(1080)
1824	9	76(76)	0(0)	0(0)	0(0)	76(0)
1920	15	240(80)	80(0)	1040(320)	2880(320)	11600(1040)
2016	14	84(84)	0(0)	84(0)	252(0)	1260(168)
2112	15	88(88)	176(0)	616(88)	968(0)	1760(352)
2208	15	92(92)	184(184)	1012(184)	1196(368)	4048(460)
2304	12	48(0)	0(0)	0(0)	0(0)	0(0)

Table D.4: WiMax LDPC Codes $R_c = 3/4$A (Modulo)

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	6	8(8)	72(48)	72(72)	504(264)	1992(648)
672	6	28(0)	0(0)	84(28)	784(168)	2408(616)
768	4	16(16)	0(0)	0(0)	32(0)	216(184)
864	9	72(36)	144(36)	720(504)	2639(1278)	12769(4140)
960	5	40(40)	160(40)	120(40)	360(160)	909(320)
1056	10	132(44)	352(264)	1628(550)	4180(1364)	15324(4488)
1152	6	16(16)	0(0)	0(0)	144(48)	192(96)
1248	9	104(52)	0(0)	156(0)	1092(520)	3368(1300)
1344	9	112(56)	168(56)	448(168)	644(112)	3111(672)
1440	10	60(0)	120(60)	300(180)	960(360)	3480(1740)
1536	8	16(16)	64(0)	64(64)	64(64)	384(256)
1632	12	340(204)	544(272)	2176(1326)	6114(2584)	19530(6664)
1728	11	72(0)	216(72)	216(0)	1800(1080)	3187(1368)
1824	11	76(76)	171(171)	76(76)	836(608)	2508(1140)
1920	6	80(0)	7(0)	8(0)	0(0)	56(56)
2016	12	84(84)	336(84)	504(168)	1848(1176)	6048(2604)
2112	12	88(0)	176(88)	440(176)	1584(1056)	3432(1760)
2208	12	92(92)	276(184)	184(92)	1380(644)	2944(1288)
2304	12	16(16)	96(96)	0(0)	672(480)	1824(768)

Table D.5: WiMax LDPC Codes $R_c = 3/4$B (Modulo)

D. WIMAX LDPC CODES WEIGHT SPECTRA (MODULO)

The results of WiMax LDPC codes in code rate of 5/6 are given in Table D.6 up to $s_{min}+4$.

n	s_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$
576	3	24(24)	12(12)	96(96)	972(564)	4200(1224)
672	5	28(0)	112(56)	1288(560)	8876(2632)	71176(14336)
768	6	96(96)	672(352)	5312(1552)	36192(8832)	274624(51184)
864	4	54(54)	72(0)	66(48)	756(324)	5634(2574)
960	4	40(0)	0(0)	80(40)	400(320)	4440(1580)
1056	7	44(44)	572(260)	3588(1508)	19344(6032)	125580(28548)
1152	6	72(24)	480(240)	1788(780)	9360(2976)	56976(13776)
1248	7	156(104)	572(260)	3588(1508)	19344(6032)	125580(28548)
1344	7	224(168)	224(112)	1064(448)	9380(3304)	54880(14560)
1440	6	60(60)	120(0)	420(120)	1860(420)	10812(3132)
1536	6	64(64)	64(64)	64(0)	256(64)	3328(1344)
1632	8	136(68)	408(204)	4624(1496)	22780(6120)	123012(29886)
1728	7	72(72)	126(54)	1464(600)	4680(1944)	24408(6912)
1824	8	152(76)	9(532)	10(1368)	9196(3116)	50274(12502)
1920	4	80(0)	0(0)	0(0)	80(80)	320(240)
2016	8	84(84)	336(336)	504(84)	5880(2436)	18018(6132)
2112	8	88(88)	176(88)	616(264)	2552(1232)	13728(3784)
2208	8	92(92)	92(92)	460(92)	2852(828)	9660(3128)
2304	9	192(192)	288(0)	1920(672)	8616(2424)	43296(13632)

Table D.6: WiMax LDPC Codes $R_c = 5/6$ (Modulo)

Appendix E

Publications

The submitted and published papers for IEEE conferences are presented in the followings:

- *L. Yang*, M. Ambroze and M. Tomlinson, "Comparison of Decoding Turbo Gallager Codes in Hybrid Decoding Arrangements with Different Iterative Decoders over the Erasure Channel", *the 11th IEEE International Conference on Communications Systems (ICCS)*, Nov, 2008, Guangzhou, China.
- M.Ambroze, M. Tomlinson and *L. Yang*, "Exhaustive Weight Spectrum Analysis of some well known LDPC Codes", *the 10th International Symposium on Communication Theory and Applications (ISCTA)*, July, 2009, Ambleside, Lake District, UK.
- *L. Yang*, M. Tomlinson and M. Ambroze, "Decoding Low-Density Parity-Check Codes with Error-Floor Free over the AWGN Channel", *the 2010 IEEE International Conference on Wireless Communication, Networking and Information Security (WCNIS2010)*, June, 2010, Beijing, China.
- *L. Yang*, M. Tomlinson and M. Ambroze, "Extended Optimum Decoding for LDPC Codes based on Exhaustive Tree Search Algorithm", submitted to *t*he 12th IEEE International Conference on Communications Systems (ICCS), November, 2010, Singapore.

Comparison of Decoding Turbo Gallager Codes in Hybrid Decoding Arrangements with Different Iterative Decoders over the Erasure Channel

Li Yang, Marcel Ambroze, Martin Tomlinson
Fixed and Mobile Communications Research
University of Plymouth
PL4 8AA, United Kingdom
Email: li.yang, marcel.ambroze, martin.tomlinson@plymouth.ac.uk

Abstract—**In this paper, different iterative decoders for turbo Gallager codes are optimised and compared for the binary erasure channel. The complexity and performance differences between turbo decoder, BCJR-based Look-Up Table decoder and belief propagation decoder are analysed and evaluated. A hybrid decoding arrangement, which uses an iterative decoder followed by a maximum likelihood "In-Place" matrix inversion algorithm, is compared for the different iterative decoders. Results are presented which show that the BCJR-based iterative decoders achieve better performance than using the belief propagation decoder for turbo Gallager codes in the erasure channel. When small encoder memory is selected, the optimised Look-Up Table decoder provides a good balance between convergence performance and complexity.**

I. Introduction

In recent years, the Binary Erasure Channel (BEC) [1], has been shown to be useful in evaluating an error correcting code's performance. The erasure channel is characterised by the bit erasure probability ϵ, where $0 \leq \epsilon \leq 1$.

In 1993, turbo codes, which have been shown to be capable of achieving near-Shannon limit performance with affordable decoding complexity, were first introduced by Berrou [2]. The turbo encoder is formed by a parallel concatenation of two convolutional encoders interconnected through an interleaver. The corresponding decoding process is based on an *iterative* decoding algorithm in which each component decoder passes the *extrinsic* information to the other to realise successful decoding. Another class of codes which were able to exhibit similar performance and characteristics was called low-density parity-check (LDPC) codes [3]. The first iterative decoding algorithm for LDPC codes over the erasure channel was proposed by Luby [4], who showed that channel capacity can be approached arbitrarily closely. The Finite-length analysis of LDPC codes over the BEC was described in [5], which showed that the performance of iterative belief-propagation decoding of LDPC codes over BEC can be characterised in terms of *stopping sets*. From [6], it was shown that stopping sets also existed for turbo codes and that they characterised exactly the performance of turbo decoding on the erasure channel. Turbo codes are easier to analyse due to the existence of efficient weight spectrum algorithms [7], however it is thought that LDPC codes could potentially have lower error floors [3].

In order to achieve low encoding and decoding complexity, Colavolpe proposed a new class of codes which combined the advantages of turbo codes with those of LDPC codes, now known as turbo Gallager codes (TGC) [8]. An optimised hybrid decoding arrangement, which uses an iterative decoder followed by a maximum likelihood "In-Place" decoder for turbo Gallager codes over the erasure channel was proposed by Yang *et al* [9]. Since turbo Gallager codes can be decoded by either BCJR-based decoders, which include turbo decoder and optimised Look-Up Table (LUT) decoder, or belief propagation (BP) decoder, the complexity and performance of each type of iterative decoder for turbo Gallager codes is compared and analysed together with the complexity and performance of the hybrid decoding arrangement.

The rest of the paper is organised as follows: in Section II, we give the background of turbo Gallager codes, and discuss code constraints necessary for the absence of cycles of length 4. General complexity considerations are given. Section III describes the iterative turbo decoder, LUT decoder and the simplified belief-propagation decoder in the erasure channel. Their complexities in terms of equivalent additions are analysed and compared. In Section IV, the complexity of the hybrid decoding arrangement combining each iterative decoder is analysed and discussed. Section V presents the numerical results of some turbo Gallager codes showing the differences between different iterative decoders and hybrid decoding arrangements over the erasure channel. Section VI presents the conclusions of the paper.

II. Preliminaries

A. Turbo Gallager Codes

Turbo Gallager codes are a special type of turbo codes which can be successfully decoded by means of the decoding algorithm used for LDPC codes by properly choosing the component convolutional codes. According to the algebraic rules of selecting the proper turbo codes as TGCs [8], we consider different constituent encoders which are recursive convolutional codes, the codes' polynomial exponent differences are constrainted to be distinct in order to ensure

TABLE I
NUMBER OF EQUIVALENT ADDITIONS PER OPERATION

Operations	Number of Equivalent Additions
Addition, Subtraction	1
Multiplication, Division	1
Comparison	1
Assignment	1
Table Look-up	3

TABLE II
NUMBER OF COMPUTATIONS FOR STANDARD TURBO DECODING ALGORITHM

Procedure	ADD	MUL	DIV	CP	ASSI
Proc. A	–	$1+2\times 2^M$	–	$4\times 2^M+2$	2
Proc. B	$2\times 2^M-1$	2×2^M	2^M	4×2^M	–
Proc. C	$2\times 2^M-1$	2×2^M	2^M	4×2^M	–
Proc. D	$2\times 2^M-1$	4×2^M	2	6×2^M	–

that message-passing decoding can work successfully on the Tanner graph of the overall code due to the absence of cycles of length four. As an example, the standard UMTS turbo codes $(15/13)$, corresponding to the polynomials $1+x+x^3$ and $1+x^2+x^3$, are suitable as turbo Gallager codes, because for each polynomial the exponent differences are distinct and form the perfect difference sets (0,1,3), (0,2,3) of size 1.

B. *Complexity Analysis Considerations*

The purpose of the complexity analysis is to determine the relative speed of the different decoders. The decoding algorithms considered for TGC decoding are optimised LUT decoder, standard turbo decoder, belief propagation decoder and "In-Place" decoder. The LUT decoder requires the look-up table to be constructed once, and this has no impact on decoding speed. Consequently the look-up table construction complexity is not included, when comparing the different decoders. A MAP decoding algorithm complexity analysis was presented in [10]. A more thorough investigation on turbo decoding algorithms was performed in [11], where each operation is quantified as a number of "equivalent additions" and logical and mathematical operations have similar complexity. In our analysis, the basic operations performed by the various decoding algorithms in erasure channel include addition (ADD), subtraction (SUB), multiplication (MUL), division (DIV), comparison (CP), assignment (ASSI) and table look-up (LKUP). The ASSI is to assign a relative value to variable. The LKUP is used in the LUT decoder, it corresponds to three equivalent additions, since 3 CPs are used to map the input parameters to the decoded value stored in the look-up table. The number of equivalent additions for the various operations are shown in Table I.

III. COMPARISON OF ITERATIVE DECODERS

A. *Turbo Decoder*

Due to the properties of the erasure channel, the MAP algorithm is sufficient to provide the accurate conditional probability for each data bit. Thus MAP algorithm is considered as the turbo decoding algorithm. Let n be the length of codeword, n_e be the number of data erasures at the input to the decoder, where $\bar{n_e} = (k\cdot\epsilon)$, k is the length of information bits. With each iteration of the iterative decoder, the number of erasures is reduced. Let n_e^i be the number of erasures at iteration i, and 2^M be the number of states in the trellis, where M is the encoder memory, N_o represents the required number of operations in equivalent additions, n_i is the maximum number of iterations.

For the turbo decoding algorithm, the required procedures are classified as follows:

1) Branch Metrics Calculation (Proc. A), requires 2 CPs and 2 ASSIs for each data bit and parity bit, 1 MUL for updating data bit estimation. For each valid branch, it consists of 2 CPs and 1 MUL. There are 2×2^M branches in total.
2) Forward Metrics Calculation (Proc. B), each state consists of 4 CPs, 2 MULs and 1 ADD for both branches. There are 2^M states in total. Normalisation requires 2^M-1 ADDs and 2^M DIVs.
3) Backward Metrics Calculation (Proc. C), the computation is the same as that in the forward metric calculation.
4) Soft Decision of the decoded bit (Proc. D), computation for $d_t=1$ includes 3 CPs and 2 MULs for each state, where d_t is the data bit at time t; there are 2^M states in total. Computation for $d_t=0$ is same as $d_t=1$. The soft decision is summed by 2^M-1 ADDs for $d_t=\pm 1$. Normalisation includes 1 ADD and 2 DIVs.

The required computations for the turbo decoding algorithm are shown in Table II. In a $1/3$ rate turbo decoding scheme, decoding one frame, k trellis sections and 2 corresponding MAP decoders are required. The number of operations N_o^T for k trellis sections with n_i iterations is obtained as:

$$N_o^T(k) = \sum_{i=1}^{n_i}(4k + 74k2^M) \qquad (1)$$

B. *Optimised LUT Decoder*

Based on the earlier invention of the table look-up based approach for decoding on trellis for convolutional codes [12] over the erasure channel, a further simplified LUT decoding algorithm for turbo Gallager codes over the erasure channel was proposed by Yang [9]. Since non-zero values of α or β are always identical, where α is the forward metric, and β is the backward metric. Consequently the values of α_i and β_i may be represented by a single bit with a "1" representing a non-zero value state and a "0" representing a zero value state. Thus two binary vectors can be used to represent the values of α and β at each trellis section and a more efficient decoding arrangement may be realised by constructing a Look-Up table, which includes two vectors of binary numbers to represent the final trellis transitions, and one vector to represent the conditions of received bits. In the decoding process, the trellis section containing erased information is processed by looking up the table to directly obtain the corrected information bit, or to indicate decoding failure. For the optimised iterative

TABLE III
NUMBER OF COMPUTATIONS FOR OPTIMISED LUT DECODING ALGORITHM

Procedure	ADD	MUL	CP	ASSI	LKUP
Proc. F	–	–	$9 \times 2^{M+1}$	–	–
Proc. G	–	–	2^{M+1}	2^M	–
Proc. H	–	–	2^{M+1}	2^M	–
Proc. I	2^M	–	–	–	–
Proc. J	$2^{M+1}-2$	2^{M+1}	2	2	1

LUT decoder, the look-up table is constructed once only. The decoding procedures including look-up table construction are classified as follows:

1) Look-Up Table Construction (Proc. F), for each trellis section, there are three possible conditions to be considered. The condition includes a) both data and parity bits are erased; b) data is erased and parity is not erased, parity is either 0 or 1. Each branch requires 2 CPs for its validation, there are 2^{M+1} branches in total. One trellis section includes 2^{M+1} CPs to decide the decoding success or failure. There are $2 \times \sum_{i=0}^{M} \binom{2^M}{2^i}$ possible trellis sections.
2) α Metric (Proc. G), each state requires 2 CPs and 1 ASSI for its validation, there are 2^M states in total.
3) β Metric (Proc. H), its computation is as same as the Proc. G.
4) Valid Trellis Construction (Proc. I), it consists of 1 ADD for each state.
5) Hard Decision of the decoded bit (Proc. J), it requires $2^{M+1} - 2$ ADDs and 2^{M+1} MULs for getting starting states and ending states; 2 CPs and 1 ASSI for getting decoding options, and 1 LKUP and 1 ASSI to assign the decoded bit.

The required computations for the optimised LUT decoding algorithm are shown in Table III. Thus in a $1/3$ rate turbo decoding scheme, decoding one frame requires k trellis sections, 2 corresponding LUT decoders and n_e^i decoding operations at each iteration. The required number of operations N_o^{LUT} with n_i iterations is obtained as:

$$N_o^{LUT}(n_e) = \sum_{i=1}^{n_i} (12k2^M + (2^{M+2} + 5)n_e^i) \qquad (2)$$

For comparison, we assume k as the maximum number of data erasures decoded by LUT decoder after n_i iterations. Thus there exist maximum k Proc. J for decoding one block, then the required number of operations $N_o^{LUT_s}$ in term of k is obtained as:

$$N_o^{LUT_s}(k) = 12k2^M n^i + (2^{M+2} + 5)k \qquad (3)$$

C. Simplified Message-Passing Decoding

Due to the simplicity of the computation of conditional probabilities over the erasure channel, the belief propagation decoding algorithm may be simplified based on [4]. If at row i, there exists more than one erasure, then there is insufficient information to decode each erasure. Thus these rows are not calculated during the complexity computation excluding checking the subset. Hence, only if subset $\mathbf{c}_i$ contains one erasure, these computations are counted regarding the decoding complexity. Then the algorithm is classified as follows:

1) Equation Check (Proc. K), each equation consists of n CPs, there are $n - k$ equations in total.
2) Decoding Erasure (Proc. L), for equations containing only one erasure, each of these requires $n - 1$ MULs and $n - 2$ ADDs.

TABLE IV
COMPUTATION REQUIREMENT OF BP DECODING ALGORITHM

Procedure	ADD	MUL	CP
Proc. K	–	–	$n-k$
Proc. L	$n-2$	$n-1$	–

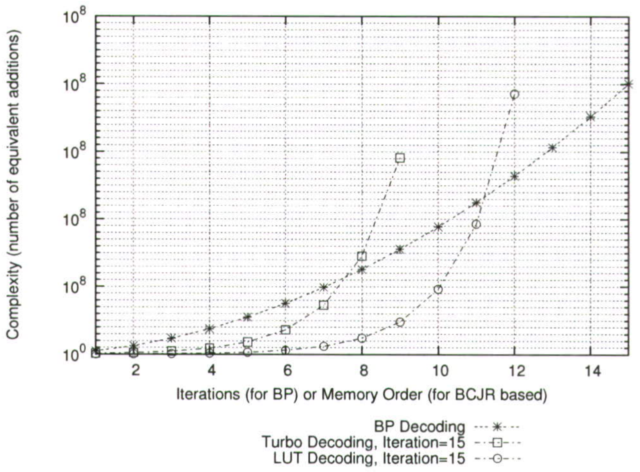


Fig. 1. Complexity Comparison between BCJR-based Decoding and BP Decoding

The required computations for the BP decoding algorithm is shown in Table IV. For $1/3$ rate turbo Gallager codes, $n - k = 2k$. Thus, to decode one frame, the required number of operations N_o^{BP} in terms of n_e with n_i iterations for BP algorithm is obtained as:

$$N_o^{BP}(n_e) = \sum_{i=1}^{n_i} (6k^2 + (n_e^{i-1} - n_e^i)(6k - 3)) \qquad (4)$$

For comparison purpose, again we assume $n - k$ as the maximum number of erasures decoded by BP decoder after n_i iterations, thus there exist maximum $n - k$ Proc. L for decoding one block, then the required number of operations $N_o^{BP_s}$ in term of k is obtained as:

$$N_o^{BP_s}(k) = 6k^2 n_i + 12k^2 - 6k \qquad (5)$$

The comparison between different iterative decoders for decoding one block, which is $n = 1536, k = 512$, is shown in Fig. 1. From the figure, it is clear that the complexity of the BCJR-based decoding algorithms is exponential as a function of memory order, meanwhile, the complexity of BP decoding is only increased by the number of iterations. Since the turbo decoder and the LUT decoder both use the BCJR algorithm, their performance will be identical. Ignoring the look-up table construction computation, the LUT algorithm provides reduced complexity compared to the turbo decoder.

IV. Maximum Likelihood Hybrid Decoder

As it is well known that the Maximum Likelihood decoder (ML) for the erasure channel is practically realisable but with complexity proportional to n^3, we consider the "In-Place" algorithm as the maximum likelihood decoding algorithm with reduced complexity [13]. The hybrid decoder scheme includes an inner iterative decoder based on either BCJR based decoding or BP decoding. The iterative decoder is followed by the "In-Place" matrix inversion algorithm, which works on the residual erasures left after iterative decoding.

"In-Place" Algorithm: The "In-Place" algorithm, which is able to achieve maximum likelihood performance [13], is a Gaussian reduction algorithm avoiding the need for column-permutations over the parity-check matrix. Since the parity-check matrix structure of turbo Gallager codes is exactly the same as turbo codes, each encoder is independent from each other. The corresponding parity bits are independent, and the parity-check matrix is divided into two independent sections. For the upper section, since the matrix is constructed in diagonal order, there are maximum M equation additions for one equation. For the lower section, since the information columns are permuted by the interleaver, we assume the maximum number of equation additions for one equation is $k-1$. Each equation addition requires n ADDs, and by using 32 bit integers to store the equation information, there are $n/32$ sub-blocks in one single equation. Hence each equation addition requires $n/32$ ADDs. Hard decision of each decoded bit requires $n-1$ MULs and $n-2$ ADDs. In decoding one frame, the required number of operations N_o^{IP} for "In-Place" algorithm is computed as:

$$N_o^{IP}(n_e) = \frac{3k(n_e-1)(2k-n_e)}{64} + \frac{3kMn_e}{32} + n_e(6k-3) \qquad (6)$$

Complexity of Hybrid Decoding Arrangements: Since the hybrid decoder includes the iterative decoder, the entire complexity is equal to the iterative decoder plus the complexity of the "In-Place" algorithm to solve the stopping sets, when stopping sets exist. The blocks decoded correctly by iterative decoder only include the iterative decoding complexity; and the blocks, which are unable to be decoded due to the residual erasures, include the iterative decoding computations and the "In-place" decoding computations. The required number of operations for each arrangement by adapting each iterative decoder is shown in Table V. Since $n_e^{n_i}$ is less than or equal to n_e^0, which is the input number of erasures to the decoder, each hybrid decoding arrangement provides a reduced complexity decoding scheme than the "In-Place"decoder on its own. The convergence performance of each hybrid decoder in term of complexity only depends on the number of erasures remaining after the iterative decoder, and depends on the iterative decoder's performance. It should be noted that as all of the decoders are maximum likelihood decoders, they all achieve the same performance. The only difference is the complexity and how this affects the decoding speed. This is primarily determined by the effectiveness of the first stage iterative decoder: the better the performance of the iterative decoder, the less the complexity of the hybrid decoder arrangement.

TABLE V
Required Number of Operations for Different Hybrid Decoding Arrangements

Hybrid Decoders	Iteratively Decoded Blocks	Maximum Likelihood Decoded Blocks
Hybrid (Turbo Decoder)	$N_o^T(k)$	$N_o^T(k) + N_o^{IP}(n_e^{n_i})$
Hybrid (LUT Decoder)	$N_o^{LUT}(n_e)$	$N_o^{LUT}(n_e) + N_o^{IP}(n_e^{n_i})$
Hybrid (BP Decoder)	$N_o^{BP}(n_e)$	$N_o^{BP}(n_e) + N_o^{IP}(n_e^{n_i})$

V. Numerical Results

Computer simulations have been carried out to assess the performance of some example turbo Gallager codes using different decoders on the erasure channel. Since the complexity of BP decoding does not directly depend on the code constraint length, it is capable of decoding turbo Gallager codes having constituent convolutional codes with long constraint lengths, and hence potentially characterised by a large free distance. Meanwhile the decoding complexity of the BCJR-based algorithm grows exponentially with the constituent convolutional code's constraint length, the alternative iterative decoder cannot be practically used for codes with long constraint lengths. In order to compare the different iterative decoders under the same conditions, we consider the UMTS turbo codes' structure as the basic code structure, where the code constraint length is increased by adding more "0" in the middle of the generator polynomials. For comparison purposes, we consider the (0,3,4/0,14,34) [8] component code as the reference code, but it should be noted that this code may only be decoded by BP decoder, due to the codes' constraint length.

The simulation is implemented for turbo Gallager codes, $n = 1536, k = 512$, in structure of (11..1/1..11) with DRP interleaver, the results of BP decoder are shown in Fig. 2. From the results, it may be seen that the decoding performance is improved by increasing the code constraint length with the runs of "0" inside. When the longer code constraint length is selected, some of the stopping sets caused by the shorter constraint length code are broken and a lower error floor is realised. When the constraint length is chosen to obtain a certain minimum Hamming distance, there only exists the difference of error floors, for instance the difference between TGC (61/43) and TGC (141/103).

Since the performance of TGC (141/103) decoded by BP decoder nearly reaches the best performance in the UMTS structure, and its constraint length is still manageable by turbo decoder, the TGC (141/103) is considered for comparison. The turbo decoding results of codes TGC (15/13) and (141/103) and their BP performances are compared and their hybrid decoding performance and ML performance are shown in Fig. 3. Since turbo decoder and LUT decoder achieve the same performance as they both are based on the BCJR algorithm,

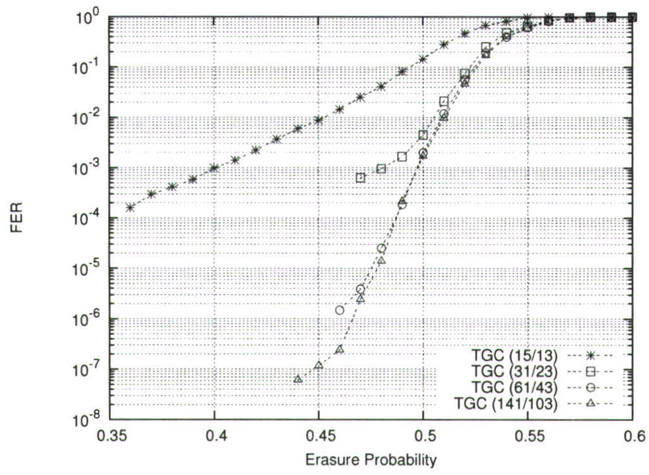


Fig. 2. Results for TGC in (11..1/1..11) Structure with BP Decoder

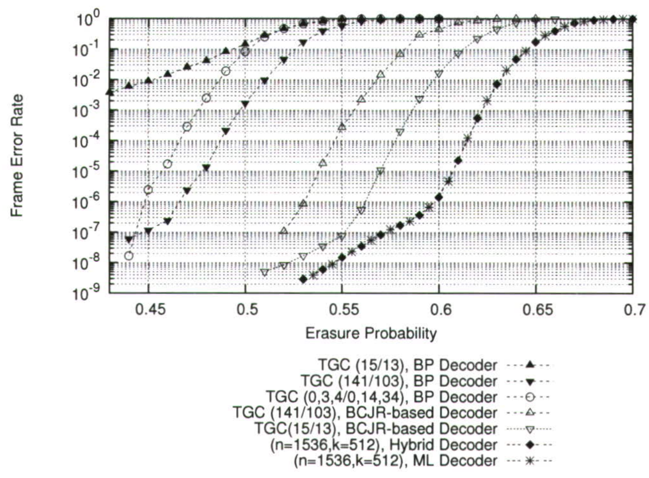


Fig. 3. Results Comparison between Iterative Decoders and Hybrid Decoders

we use BCJR-based decoder performance to describe both decoders' performances. According to the results, it is realised that TGC (141/103) may be a better code compared with TGC (0,3,4/0,14,34) due to its better performance with BP decoder, although TGC (0,3,4/0,14,34) has a lower error floor due to its long constraint length. At the other side, the BCJR-based decoders achieve impressive coding gain improvements for both TGC (15/13) and TGC (141/103) compared to the BP decoder. The TGC (15/13) with small encoder memory achieves the best iterative decoder performance. It is clearly seen that for turbo Gallager codes on the erasure channel, BCJR-based decoders converge much better than the BP decoder. This is in contrast to the AWGN channel where turbo Gallager codes with BCJR-based decoding or BP decoding produce similar results.

VI. Conclusion

In this paper, we have compared different iterative decoding algorithms for examples of turbo Gallager codes. Unlike the AWGN channel, we have shown that there exists a performance difference for turbo Gallager codes between BCJR-based decoding and BP decoding on the erasure channel. The iterative decoding algorithms' complexity have been compared between MAP decoding, LUT decoding and belief propagation decoding in terms of number of equivalent additions. Ignoring the complexity of look-up table construction, the LUT decoder is able to achieve a reduced complexity than MAP decoder and should operate faster in practical decoder implementations. Although both BCJR-based iterative decoders suffer from an exponential increase in complexity with encoder memory, the numerical results have shown that BCJR-based decoding algorithms are able to achieve better performance than BP decoder in erasure channel for same turbo Gallager codes. Both hybrid decoding arrangements using the "In-Place" decoder provide reduced complexity decoding compared to a stand alone "In-Place" decoder, and all decoders achieve ML performance. The BCJR-based decoders using low memory turbo Gallager codes provide the best trade-off between convergence performance and complexity. Good performance was obtained for the TGC (15/13) with small memory and is a good candidate for a hybrid decoding scheme using the LUT decoder with reduced complexity.

References

[1] P. Elias, "Coding for two noisy channels," in *Proc. 3rd London Symposium on Information Theory*, London, England, 1955, pp. 61–76, Academic Press, New York.

[2] C. Berrou A. Glavieux and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE International Conference on Communications*, Geneva, Switzerland, May 1993, pp. 1064–1070.

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

[4] M. Luby M. Mitzenmacher M. Shokrollahi and D. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, Feb 2001.

[5] D. Proietti I. E. Telatar T. Richardson C. Di and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, pp. 1570–1579, Jun 2002.

[6] E. Rosnes and O. Ytrehus, "Turbo decoding on the binary erasure channel: Finite-length analysis and turbo stopping sets," in *IEEE International Symposium on Information Theory*, Feb 2006.

[7] E. Rosnes and O. Ytrehus, "Improved algorithms for the determination of turbo-code weight distributions," *IEEE Transactions on Communications*, 2005.

[8] G. Colavolpe, "Design and performance of turbo gallager codes," *IEEE Transactions on Communications*, vol. 52, no. 11, Nov 2004.

[9] L. Yang M. Ambroze and M. Tomlinson, "Decoding turbo gallager codes for the erasure channel," *Submitted to IEEE Communication Letter*, Feb 2008.

[10] P. Robertson E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *IEEE Int. Conf. Comm.*, Seattle, WA, USA, June 1995, vol. 2, pp. 1009–1013.

[11] P.H.-Y. Wu, "On the complexity of turbo decoding algorithms," in *Proc. 53rd Vehicular Technology Conference,*, Spring 2001, vol. 2, pp. 1439–1443.

[12] B. M. Kurkoski P. H. Siegel and J. K. Wolf, "Exact prbability of erasure and a decoding algorithm for convolutional codes on the binary erasure channel," in *Proc. IEEE GLOBECOM*, San Francisco, CA, Dec 2003, vol. 3, pp. 1741–1745.

[13] J. Cai C. Tjhai M. Tomlinson M. Ambroze and M. Z. Ahmed, "An efficient solution to packet loss: Erasure correcting codes," in *Proc. 4th IASTED International Conference Communication Systems and Networks*, Spain, 2005, pp. 224–229, ACTA Press.

Exhaustive Weight Spectrum Analysis of some well known LDPC Codes

Marcel Ambroze, Martin Tomlinson, Li Yang
Fixed and Mobile Communications Research
University of Plymouth
PL4 8AA, United Kingdom
Email: marcel.ambroze, martin.tomlinson, li.yang@plymouth.ac.uk

Abstract—**The indicative performance of an LDPC code may be determined from exhaustive analysis of the low weight spectral terms of the code's stopping sets which by definition includes the low weight codewords. In a landmark paper in 2007, Rosnes and Ytrehus showed that exhaustive, low weight stopping set analysis of codes whose parity check matrix is sparse is feasible using a bounded tree search over the length of the code with no distinction between information and parity bits. For an (n, k) code the potential total search space is of size 2^n but a good choice of bound dramatically reduces this search space to a practical size. Indeed the choice of bound is critical to the success of the algorithm. It is shown in this paper that an improved algorithm can be obtained if the bounded tree search is applied to a set of k information bits since the potential total search space is initially reduced to size 2^k. Since such a restriction will only find codewords and not all stopping sets a class of bits is defined as unsolved parity bits and these are also searched as appended bits in order to find all low weight stopping sets. Weight spectrum results are presented for a commonly used WiMax LDPC code plus some other well known LDPC codes.**

I. Introduction and Preliminaries

In a landmark paper in 2007, Rosnes and Ytrehus showed that exhaustive, low weight stopping set analysis of codes whose parity check matrix is sparse is feasible using a bounded tree search over the length of the code with no distinction between information and parity bits [1]. A previous paper on the same topic of exhaustive search of stopping sets of LDPC codes by Wang et al, [2] used a different and much less efficient algorithm. In common with these two papers we use similar notation and preliminaries.

The code C is defined to be binary and linear of length n and dimension k and is a k-dimensional subspace of $\{0,1\}^n$, and may be specified as the null space of a $m \times n$ binary parity check matrix $\mathbf{H}$ of rank $n-k$. The number of parity check equations, m of $\mathbf{H}$ satisfies $m \geq (n-k)$. It should be noted , as illustrated in the results below, that the number of parity check equations m in excess of $n-k$ can have a dramatic effect on the stopping set weight spectrum, excluding codewords of course as these are not affected.

As in [1], $\mathcal{S}$ is used to denote a subset of $\{0,1\}^n$, the set of all binary vectors of length n. At any point in the tree search, a constraint set, $\mathcal{F}$ is defined consisting of bit positions p_i and the states of these bit positions s_{p_i}, $s_{p_i} \in \{0,1\}^n$. The support set $\chi(\mathcal{F})$ of the constraint set, $\mathcal{F}$, is the set of positions where $s_{p_i} = 1$, and the Hamming weight of $\mathcal{F}$ is the number of such positions. The sub matrix $\mathbf{H}_{\chi(\mathcal{F})}$ consists of all the columns of $\mathbf{H}$ where $s_{p_i} = 1$, and the row weight of $\mathbb{H}_{\chi(\mathcal{F})}$ is the number of $1's$ in that row. An active row of $\mathbf{H}_{\chi(\mathcal{F})}$ is a row with unity row weight. It is obvious that if all rows of $\mathbf{H}_{\chi(\mathcal{F})}$ have even row weight then $\mathcal{F}$ is a codeword, noting that for an iterative decoder codewords are also stopping sets. If at least one row has odd weight, 3 or higher and there are no active rows then $\mathcal{F}$ is a stopping set but not a codeword. If there are active rows then $\mathcal{F}$ has either to be appended with additional bit positions or one or more states s_{p_i} need to be changed to form a stopping set. With this set of basic definitions, tree search algorithms may be described which carry out an exhaustive search of $\{0,1\}^n$ using a sequence of constraints $\mathcal{F}$ to find all stopping sets whose Hamming weight is $\leq \tau$.

II. An efficient Tree Search Algorithm

The *constraint set* F is used to represent the set of searched known bits of a code C, which forms a *branch* of the tree in the tree search. The set of active rows in $\mathbf{H}$ is denoted by $\{\mathfrak{h}_0, ..., \mathfrak{h}_{\phi-1}\}$, where ϕ is the total number of active rows. A constraint set F with size n is said to be *valid* if and only if there exists no active rows in $\mathbf{H}^{(F)}$. The pseudocode of a particularly efficient algorithm to find all the stopping sets including codeword sets with weight equal to or less than τ is given in Algorithm 1 below. The found stopping sets are stored as the algorithm progresses.

The modified iterative decoding is carried out on a n-length binary input vector containing in some of the positions. Let $r_j(F)$ be the rank (ones) of row j, $j \in \{0, ..., m-1\}$ for the constrained position $\{p_i : (p_i, 1) \in F\}$ intersected by row j on $\mathbf{H}$. And let $r'_j(F)$ be the rank of row j for the unconstrained position $\{p_i : (p_i, 1) \in \{0, ..., n-1\} \backslash F\}$ intersected by

Algorithm 1 Tree-search based Stopping Set Enumeration (TSSE)

```
repeat
  Pick one untouched branch as a constraint set $F$.
  if $|F| = n$ and $w(F) \leq \tau$ then
    Constraint set $F$ is saved, if $F$ is valid
  else
    1). Pass $F$ to the modified iterative decoder (*) with erasures in the unconstrained positions.
    2). Construct a new constraint set $F'$ with new decoded positions, which is the extended branch.
    if $|F'| = n$ and $w(F') \leq \tau$ then
      Constraint set $F'$ is saved, if $F'$ is valid
    else if No contradiction is found in $\mathbf{H}^{(F')}$, and $w'(F') \leq \tau$ then
      a). Pick an unconstrained position $p$.
      b). Extending branch $F'$ to position $p$ to get new branch $F'' = F' \bigcup \{(p, 1)\}$ and branch $F''' = F' \bigcup \{(p, 0)\}$.
    end if
  end if
until Tree has been fully explored
```

row j on $\mathbf{H}$. The modified iterative decoding algorithm based on belief-propagation decoding algorithm over the binary erasure channel is shown in Algorithm 2. As noted in the line with marked (*), the modified iterative decoder is not invoked if the condition of $r_j \leq 1$ and $r'_j = 1$ is not met; or the branch with constraint set F has condition of $r_j = 1$ and $r'_j = 0$. This significantly speeds up the tree search. As noted in

Algorithm 2 Modified Iterative Decoding

```
Get rank $\mathbf{r}(F)$ and $\mathbf{r}'(F)$ for all the equation rows on $\mathbf{H}$.
repeat
  if $r_j > 1$ then
    Row $j$ is flagged
  else if $r_j = 1$ and $r'_j = 0$ then
    Contradiction $\rightarrow$ Quit decoder
  else if $r_j \leq 1$ and $r'_j = 1$ then
    1). Row $j$ is flagged
    2). The variable bit $i$ is decoded as the XOR of the value of $r_j$.
    3). Update the value of $r_j$ and $r'_j$, if $H_{ji} = 1$.
  end if
until No new unconstrained bit is decoded
```

the line with marked (*), the modified iterative decoder is not necessary to call, if the condition of $r_j \leq 1$ and $r'_j = 1$ is not met; or the branch with constraint set F can be ignored, if condition of $r_j = 1$ and $r'_j = 0$ occurs. Thus the computing complexity can be significantly reduced than calling it for every new branch with the corresponding constraint set F.

A. Efficient lower bound

The tree search along the current branch may be terminated if the weight necessary for additional bits to produce a stopping set plus the weight of the current constraint set F exceeds τ. Instead of actually evaluating these bits it is more effective to calculate a lower bound on the weight of the additional bits. The bound uses the active rows $\mathcal{I}(F) = \{I_{i_0}(F), ..., I_{i_{q-1}}(F)\}$, where $I_{i_0}(F)$ is the set of active rows with constraint set F corresponding to the i_0th column $\mathbf{h}_{i_0}$ of $\mathbf{H}$, and q is the number of intersected unknown bits. Let $w(\mathbf{h}_j^{I_j(F)})$ be the weight of ones on jth column of $\mathbf{H}$, which is the number of active rows intersected with jth column. Under a worst case assumption, the $I_j(F)$ with larger column weight of ones on jth column is always with value 1, then the active rows can be compensated by $I_j(F)$ and the total number of active rows ϕ is reduced by $w(\mathbf{h}_j^{I_j(F)})$ until $\phi \leq 0$. Algorithm 3 shows the pseudocode of computing the smallest number of intersected unknown bits q in order to produce no active rows. The lower bound $w'(F) = w(F) + q$ is the result.

Algorithm 3 Simple method to find the smallest collection set of active rows

```
1. Arrange the set of $\mathcal{I}(F)$ in descending order, where $\mathbf{h}_{i'_0}$ is the column with the maximal column weight corresponding to constraint $F$.
2. $q$ is initialised as 0.
while $\phi > 0$ do
  1). $\phi$ is subtracted by $w(\mathbf{h}_{i'_0})$.
  2). $q$ is accumulated by 1.
end while
```

B. Best coordinate position selection

In the evaluation of the lower bound above, the selected unconstrained positions are assumed to have value 1. Correspondingly, the first position in the index list has maximal column weight and is the best choice for the coordinate to add to the constraint set F.

III. Results

The algorithms above have been used to evaluate all of the low weight stopping sets for some well known LDPC codes. The results are given in Table I. The total number of stopping sets are shown for a given weight with the number of codewords in parentheses. Interestingly the Tanner code has 93 parity check equations, 2 more than the 91 parity check equations needed to encode the code. If only 91 parity check equations are used in the iterative decoder there is a stopping set of weight 12 degrading the decoder performance.

A. WiMax LDPC codes

WiMax LDPC codes [4] , as the IEEE 802.16e standard LDPC codes, have been fully explored and the low weight stopping sets for all combinations of code rates and lengths have been found. Detailed results for WiMax LDPC codes of code rates $1/2$, $2/3A$, $2/3B$, $3/4A$, $3/4B$ are given in Table II, III, IV, V, VI. In these tables, the code index i is linked to the code length N by the formula $N = 576 + 96i$. The minimum weight of non-codeword stopping sets (s_m) and codeword stopping sets (d_m) for all WiMax LDPC codes is given in Table VII.

IV. Conclusions

An efficient algorithm has been presented which enables all of the low weight stopping sets to be evaluated for some common LDPC codes. Future work will explore the determination of efficient algorithms for use with multiple computers operating in parallel in order to evaluate all low weight stopping sets for codes several thousand bits long. Future work will also explore the performance improvements obtainable by using a number of additional parity check equations over $n - k$.

References

[1] E. Rosnes and O. Ytrehus, " An algorithm to find all small-size stopping sets of Low-Density Parity-Check Matrices", *ISIT 2007. IEEE International Symposium on Information Theory*, pp. 2936 - 2940, June 2007.

[2] C. C. Wang, S.R.Kulkarni and H.V.Poor, "Exhausting Error-Prone Patterns in LDPC Codes", submitted to *IEEE Transactions on Information Theory*, available from http://arxiv.org/abs/cs.IT/0609046.

[3] R. M. Tanner, D. Sridhara and T. Fuja, "A class of group-structured ldpc codes", in *Proc. Int. Symp. on Commun. Theroy and Appl. (ISCTA)*, Ambleside, England, July 2001.

[4] "WiMax LDPC codes, Air interface for fixed and mobile broadband wireless access systems, IEEE Std 802.16e-2005", available from http://standards.ieee.org/getieee802/download/802.16e-2005.pdf.

[5] Lan Lan, Lingqi Zeng, Y.Y.Tai, Lei Chen, Shu Lin and K. Abdel-Ghaffar, "Construction of Quasi-Cyclic LDPC Codes for AWGN and Binary Erasure Channels: A Finite Field Approach", *IEEE Transactions on Information Theory*, vol. 53, Issue 7, July 2007 Page(s):2429 - 2458.

[6] D.J.K. MacKay, "Encyclopedia of sparse graph codes [Online]", Available: http//www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[7] X. Y. Hu, E. Eleftheriou and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs", *IEEE Transactions on Information Theory*, vol. 51, pp. 386-398, Jan 2005.

Code Name	s_m	N_{s_m}	$N_{s_m}+1$	$N_{s_m}+2$	$N_{s_m}+3$	$N_{s_m}+4$	$N_{s_m}+5$	$N_{s_m}+6$
Tanner (155, 64) [3]	18	465 (0)	2015 (0)	9548 (1023)	23715 (0)	106175 (6200)	359290 (0)	1473585 (43865)
QC LDPC (1024, 512) [5]	15	1 (1)	1 (0)	0 (0)	1 (1)	6 (1)	6 (2)	12 (4)
PEG Reg (256, 128) [6], [7]	11	1 (0)	11 (7)	22 (12)	51 (28)	116 (46)	329 (113)	945 (239)
PEG Reg (504, 252) [6], [7]	19	2 (0)	5 (2)	8 (0)	27 (5)	78 (0)	199 (26)	()
PEG iReg (504, 252) [6], [7]	13	2 (1)	1 (1)	5 (5)	13 (11)	31 (16)	52 (28)	124 (60)
PEG iReg (1008, 504) [6], [7]	13	1 (1)	0 (0)	0 (0)	3 (3)	3 (3)	4 (4)	5 (3)
MacKay (504, 252) [6]	16	1 (0)	3 (0)	3 (0)	12 (0)	36 (2)	106 (0)	320 (22)

TABLE I
LOW WEIGHT STOPPING SETS AND CODEWORDS OF KNOWN CODES.

i	S_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$	$N_{s_{min}+5}$	$N_{s_{min}+6}$	$N_{s_{min}+7}$	$N_{s_{min}+8}$
0	13	24(24)	0(0)	0(0)	24(24)	0(0)	24(0)	120(72)	312(96)	()
1	18	56(0)	140(56)	56(56)	308(84)	420(168)	756(224)	2296(476)	5460(1288)	()
2	18	32(0)	0(0)	96(64)	128(32)	192(96)	704(352)	992(224)	1888(672)	()
3	19	36(36)	36(36)	144(0)	324(36)	828(180)	810(162)	2304(576)	()	()
4	19	120(80)	120(40)	160(0)	280(160)	400(120)	880(120)	1760(560)	()	()
5	19	44(0)	0(0)	44(44)	132(0)	220(88)	176(44)	176(132)	()	()
6	19	48(48)	0(0)	0(0)	0(0)	0(0)	48(0)	144(144)	()	()
7	19	52(0)	0(0)	0(0)	52(52)	()	()	()	()	()
8	23	112(112)	56(0)	280(224)	560(224)	1008(280)	()	()	()	()
9	24	60(0)	60(0)	60(0)	180(60)	720(300)	()	()	()	()
10	20	64(64)	0(0)	0(0)	64(64)	64(0)	0(0)	96(96)	256(128)	()
11	27	68(68)	408(0)	()	()	()	()	()	()	()
12	21	72(72)	0(0)	0(0)	0(0)	0(0)	0(0)	216(216)	144(0)	()
13	19	76(76)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	76(76)	76(76)
14	25	160(80)	240(80)	240(240)	400(160)	()	()	()	()	()
15	27	84(84)	84(84)	756(168)	518(182)	()	()	()	()	()
16	28	264(264)	88(0)	440(264)	()	()	()	()	()	()
17	23	92(92)	0(0)	0(0)	0(0)	0(0)	276(92)	()	()	()
18	28	96(0)	96(0)	288(0)	288(96)	624(336)	()	()	()	()

TABLE II
WIMAX 1/2 LDPC CODES

i	S_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$	$N_{s_{min}+5}$	$N_{s_{min}+6}$
13	15	76(76)	228(152)	()	()	()	()	()
14	14	80(0)	80(80)	160(0)	()	()	()	()
15	15	84(84)	252(0)	()	()	()	()	()
16	15	88(88)	0(0)	()	()	()	()	()
17	15	92(92)	0(0)	92(92)	460(276)	()	()	()
18	15	96(96)	0(0)	96(96)	480(384)	()	()	()

TABLE III
WIMAX 2/3A LDPC CODES

i	S_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$	$N_{s_{min}+5}$	$N_{s_{min}+6}$
6	16	96(48)	432(48)	()	()	()	()	()
7	15	52(52)	0(0)	104(104)	156(104)	728(312)	2041(533)	()
8	16	63(63)	56(56)	196(56)	560(168)	1568(196)	()	()
9	17	120(60)	()	()	()	()	()	()
10	15	64(64)	0(0)	0(0)	0(0)	128(0)	384(64)	()
11	18	204(68)	()	()	()	()	()	()
12	15	72(72)	0(0)	0(0)	72(0)	()	()	()
13	15	76(76)	0(0)	0(0)	0(0)	0(0)	76(0)	()
14	16	80(80)	80(0)	()	()	()	()	()
15	15	84(84)	0(0)	0(0)	0(0)	84(84)	294(168)	()
16	16	88(88)	88(0)	()	()	()	()	()
17	20	92(92)	92(0)	92(0)	()	()	()	()
18	15	96(96)	0(0)	0(0)	0(0)	0(0)	144(96)	()

TABLE IV
WIMAX 2/3B LDPC CODES

i	S_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$	$N_{s_{min}+5}$	$N_{s_{min}+6}$
6	10	48(0)	0(0)	24(0)	240(48)	624(288)	()	()
7	12	26(0)	156(52)	260(104)	2184(416)	()	()	()
8	12	28(0)	112(0)	224(168)	952(280)	()	()	()
9	12	90(60)	60(0)	180(60)	372(192)	()	()	()
11	12	34(0)	68(68)	0(0)	0(0)	()	()	()
12	12	36(0)	0(0)	0(0)	0(0)	72(0)	504(144)	()
13	12	38(0)	76(76)	0(0)	76(76)	()	()	()
14	12	40(0)	80(0)	160(0)	240(0)	240(0)	800(160)	()
15	12	42(0)	0(0)	0(0)	0(0)	0(0)	168(84)	()
16	12	44(0)	0(0)	0(0)	88(88)	()	()	()
17	12	46(0)	0(0)	0(0)	0(0)	0(0)	0(0)	()
18	12	48(0)	0(0)	0(0)	0(0)	0(0)	0(0)	96(0)

TABLE V
WiMAX 3/4A LDPC Codes

i	S_{min}	$N_{s_{min}}$	$N_{s_{min}+1}$	$N_{s_{min}+2}$	$N_{s_{min}+3}$	$N_{s_{min}+4}$	$N_{s_{min}+5}$	$N_{s_{min}+6}$
7	9	52(52)	52(52)	52(52)	312(156)	988(416)	3094(1274)	11180(3952)
8	12	560(392)	616(224)	1792(616)	7784(2968)	()	()	()
9	10	60(60)	60(60)	130(10)	540(240)	2190(810)	7440(2940)	()
10	11	64(64)	128(128)	128(64)	960(640)	3648(1408)	()	()
11	13	272(204)	748(544)	2992(1564)	()	()	()	()
12	12	72(0)	576(432)	576(216)	2520(936)	()	()	()
13	12	228(228)	380(304)	988(836)	2888(836)	()	()	()
14	10	80(80)	0(0)	0(0)	0(0)	640(480)	2416(1216)	()
15	11	84(0)	84(84)	336(168)	546(294)	1260(588)	()	()
16	14	176(88)	968(792)	()	()	()	()	()
17	13	184(92)	92(92)	1012(644)	()	()	()	()
18	12	16(16)	96(96)	672(480)	()	()	()	()

TABLE VI
WiMAX 3/4B LDPC Codes

	Code Length $N = 576 + 96i$									
i	0	1	2	3	4	5	6	7	8	9
N	576	672	768	864	960	1056	1152	1248	1344	1440
Code Rate	Minimum Codeword Weight d_m									
1/2	13	19	20	19	19	21	19	22	23	27
2/3A	10	9	8	11	13	10	14	13	14	13
2/3B	12	11	14	16	15	15	16	15	16	17
3/4A	10	10	10	12	12	13	13	13	14	12
3/4B	8	8	9	11	11	9	11	9	12	10
5/6	5	7	7	7	7	7	7	7	7	7
	Minimum Stopping Set Weight s_m									
1/2	18	18	18	21	19	19	24	19	24	24
2/3A	10	10	11	9	12	13	13	14	14	14
2/3B	10	12	13	15	14	16	16	18	18	17
3/4A	9	8	10	11	12	12	10	12	12	12
3/4B	9	10	10	10	11	11	11	12	12	12
5/6	6	6	7	7	7	7	7	9	7	8

	Code Length $N = 576 + 96i$								
i	10	11	12	13	14	15	16	17	18
N	1536	1632	1728	1824	1920	2016	2112	2208	2304
Code Rate	Minimum Codeword Weight d_m								
1/2	20	27	21	19	25	27	28	23	31
2/3A	12	13	15	15	15	15	15	15	15
2/3B	15	18	15	15	16	15	16	20	15
3/4A	14	13	17	13	17	17	15	20	19
3/4B	11	13	13	12	10	12	14	13	12
5/6	7	7	8	8	7	7	8	8	9
	Minimum Stopping Set Weight s_m								
1/2	24	28	28	28	25	29	29	28	28
2/3A	15	12	14	16	14	16	17	18	18
2/3B	19	18	18	20	17	20	17	21	20
3/4A	12	12	12	12	12	12	12	12	12
3/4B	13	13	12	13	14	11	14	13	15
5/6	8	9	7	9	7	8	9	8	10

TABLE VII
WiMax Codes [4] Weight Spectrum

Decoding Low-Density Parity-Check Codes with Error-Floor Free over the AWGN Channel

Li Yang, Martin Tomlinson, Marcel Ambroze

Abstract—We propose a new soft decision decoding arrangement for LDPC codes over the AWGN channel with error-floor free. The iterative belief propagation decoder is used as the initial decoder with the iterative output conditioned prior to OSD decoding. Improved results are obtained to break the corresponding error floors caused by the stopping sets. The basis of the conditioning of the iterative output is explained with supporting analysis. Some practical examples of performance are presented for some well known LDPC codes and it is shown that the proposed decoder with OSD-i does not only produce better results than a stand-alone OSD-$(i+1)$ decoder with considerable reduction in decoder complexity, but also guarantees the error-floor free.

Keywords—LDPC, error-floor, OSD, conditioned

I. Introduction

LOw-density parity-check (LDPC) codes are one of the traditional types of linear block codes since their first introduction by Gallager [1] in 1963 and have attracted a great deal of interest in recent years, following the rediscovery by MacKay [2] in 1999. The corresponding message passing decoding algorithm (also known as *belief propagation*, BP) as an iterative decoding algorithm has successfully brought traditional codes back into the modern digital communications research area after the invention of turbo codes [3] in 1993. The competition between LDPC codes and turbo codes has probably peaked although the outcome is by no means clear. Following the introduction of irregular LDPC codes [4], the asymptotic approach to the Shannon limit by LDPC codes coupled with iterative decoding has proven the benefits of the approach [5].

In terms of the optimal soft decision decoding performance for the AWGN channel (also called *maximum-likelihood*, ML), LDPC codes with ML decoding in general is not feasible. In 1995, Fossorier proposed the sub-optimum decoder, called *ordered statistics decoder* (OSD) [6], which aimed to search the re-ordered most-reliable k information bits for the maximized codeword with constraint of order i. Better performance was later achieved with the same order of i in 2002 [7]. Valembois [8] based on Fossorier's previous work proposed the "box and match techniques' to help further improve the decoder performance. An extended Dorsch decoder [9] was proposed by Tomlinson [10] in 2007, aimed at achieving near optimal ML decoding for linear block codes by searching the k information bits for error patterns, leading to differential low weight codewords.

Li Yang is the PhD student with the Fixed and Mobile Communications Research, University of Plymouth, PL4 8AA, United Kingdom, email: li.yang@plymouth.ac.uk

Professor M. Tomlinson is the leader of the Fixed and Mobile Communications Research, University of Plymouth, PL4 8AA, United Kingdom, email: martin.tomlinson@plymouth.ac.uk

Dr. M. Ambroze is the lecturer with the Fixed and Mobile Communications Research, University of Plymouth, PL4 8AA, United Kingdom, email: marcel.amborze@plymouth.ac.uk

A hybrid decoding arrangement for the erasure channel using turbo Gallager codes was proposed by Yang [11], which combines an optimized iterative decoder using (BP or MAP) with a ML decoder. In this paper, a new decoding arrangement is considered for the AWGN channel using LDPC codes with sparse parity-check matrices. The iterative belief propagation decoder is used as the initial decoder, the iterative output is conditioned so that it is best suitable as the input to OSD decoder instead of using the traditional soft iterative output. As a consequence improved results are obtained and successfully break the corresponding error floors.

The basis of the conditioning of the output of the iterative decoder is explained with supporting analysis in Section II. The difference between using the standard iterative output from the BP decoder and the conditioned output is analyzed and compared in Section III. Section IV gives results showing the relative performances of the OSD decoder, the new decoder and the BP decoder for some well known LDPC codes. Section V gives the conclusions.

II. Decoding beyond iterative decoding for the AWGN Channel

The proposed decoding structure for linear, (n, k), block codes over the AWGN channel is shown in Figure 1, where $\mathbf{r}$ is the received signal vector plus AWGN with variance σ^2. The BP iterative decoder produces the output $\mathbf{r}'$ after a given number of iterations. Then $\mathbf{r}'$ is conditioned to become $\hat{\mathbf{r}}'$ which is permuted and re-encoded by the corresponding generator matrix $\mathbf{G}^{\pi}$, where π is the index interleaver determined primarily by the bit log likelihood ratios, and secondly by column swaps to achieve full rank. The new generated codeword $\mathbf{c}'$ is passed to OSD-i decoder to search for derived codewords $\mathbf{c}''$ achieving highest cross-correlation with the received vector, constrained by order i. During the OSD-i decoding, the cross-correlation of $\mathbf{c}'$ based on $\hat{\mathbf{r}}'$ is used as a lower bound to limit the search size of the OSD-i decoder. If OSD-i is unable to find any codeword with higher cross-correlation than $\mathbf{c}'$, constrained by the search size $\binom{k}{i}$, then $\mathbf{c}'$ is selected as the output codeword.

A. Iterative Decoding

In the earlier approach [12], OSD-i decoding was attempted with each iteration switching back and forth between OSD

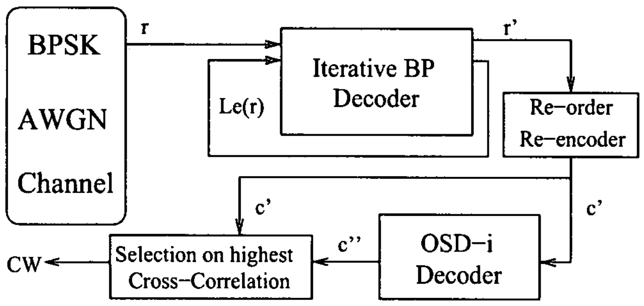


Fig. 1. The Proposed Decoding Structure

decoding and iterative decoding. Here we show that better results may be achieved by the simpler approach of carrying out a fixed number of iterations N, before invoking OSD-i decoding with no switching back and forth. For LDPC codes, whose parity-check matrices are sparse having few, if any, cycles of length 4, the iterative BP decoder most of the time is able to improve the extrinsic information with each iteration. After N iterations, instead of passing the output vector, $\mathbf{r}'$, directly to the OSD-i decoder as in [12], the output vector is conditioned to become $\hat{\mathbf{r}}'$ and then passed to the OSD-i decoder. This is to avoid occasions when the iterative decoder destroys some of the received information prior to passing it to the OSD-i decoder.

B. OSD-i decoding and construction of equivalent generator matrices

As in conventional OSD-i decoding, decoding is based on determining information sets, vectors of length k bits which can be used to generate codewords of the code. We want the generated n bit codewords to be close in Euclidean distance to the input vector of length n. Correspondingly, the input vector $\hat{\mathbf{r}}'$ from the iterative decoder is permuted in order of reliability based on the log likelihood ratios given by $|\hat{\mathbf{r}}'|$ to become $\mathbf{x}$ plus second order considerations based on needing to achieve full rank for the constructed generator matrix. The permuted input is $x_m = \pi(\hat{r}'_m)$, where π denotes the required permutation. The permuted $\mathbf{x}$ vector consists of almost the *most reliable bits* (MRB) from most reliable, extreme left and the *least reliable bits* (LRB) from extreme right, where $\{|x_0| > |x_1| >, ..., > |x_{n-1}|\}$. In order to obtain full rank of the new generator matrix $\mathbf{G}$, Gaussian elimination is performed by starting from the least reliable bit x_{n-1} and progressing towards x_k. At some point, there will be one bit, $x_{k+\delta}$ which is not independent of the previously solved for bits and thus cannot be solved for. It is not possible for this bit to be a parity bit, given the previous choices for parity bits. This is indicated by $x_{k+\delta}$, not being present in any of the remaining, uncommitted parity check equations. The procedure in this circumstance is to skip bit $x_{k+\delta}$ and try to solve for the next more reliable bit $x_{k+\delta-1}$, solve if possible, skipping if not, and continue in this way. In practice, very few bits have to be skipped in this way and skipped bits have almost the same reliability [10] as the bits that replace them in the LRB. Thus the ordered received vector $\mathbf{x}$ is interleaved by index factor $\bar{\pi}$ as $\bar{\mathbf{x}}$, due to the column swaps. Then we have the updated MRB from extreme left as $\{|\bar{x}_0| > |\bar{x}_1|, ..., > |\bar{x}_{k-1}|\}$ and the LRB from extreme right as $\{|\bar{x}_k| > |\bar{x}_{k+1}|, ..., > |\bar{x}_{n-1}|\}$.

III. BP OUTPUT IMPACT ON OSD DECODER WITH CONDITIONED SOFT DECISIONS

For the AWGN channel, the ML decoder searches for the codeword out of all the possible codewords which has the highest cross-correlation, Y_{max} with the received vector $\mathbf{r}$.

$$Y_{max} = \sum_{j=0}^{n-1} |r_j| \tag{1}$$

Let $\hat{\mathbf{c}}$ be the hard-decided binary code vector from $\mathbf{r}$. Let $\mathcal{C}_1$ be the transmitted codeword resulting in the received vector $\mathbf{r}$, and $Y(\mathcal{C}_1)$ its cross-correlation with $\mathbf{r}$.

$$Y(\mathcal{C}_1) = \sum_{j=0}^{n-1} |r_j| \cdot (1 - \mathcal{C}_{1_j} \oplus \hat{c}_j) \tag{2}$$

For the decoder there exist a set of codewords $\mathcal{C}_2$, where $Y_{max} \geq Y(\mathcal{C}_2) \geq Y(\mathcal{C}_1)$.

- **Case A:** If there exists any $\mathcal{C}_2$ with $Y(\mathcal{C}_2) > Y(\mathcal{C}_1)$, a ML type decoding error will occur.
- **Case B:** If the best codeword $\mathcal{C}_2$ with $Y(\mathcal{C}_2) = Y(\mathcal{C}_1)$, the codeword with maximum correlation with $\mathbf{r}$ is the transmitted codeword and the decoder has achieved successful decoding.

For constrained codeword search, aiming to achieve convergent output performance, there exists the following situations

- For case A, if there exists any $\mathcal{C}_2$ with $Y(\mathcal{C}_2) > Y(\mathcal{C}_1)$ during the constrained search, a ML decoding error is obtained. Otherwise if there exists $\mathcal{C}_2$ with $Y(\mathcal{C}_2) = Y(\mathcal{C}_1)$, the transmitted codeword is found. If no codeword is found satisfied all parity check equations, a non-ML decoding error occurs.
- For case B, there only exists that the transmitted codeword with $\mathcal{C}_1 = \mathcal{C}_2$ corresponding to $\mathbf{r}$ is found or a non-ML decoding error is indicated.

Most codeword-search algorithms adopt various constraints aimed at achieving near-optimum performance with smaller search size. With this aim, let $\mathbf{c}'$ be the re-encoded codeword based on re-ordered k information bits from $\mathbf{r}'$. The maximum attainable cross-correlation $Y(\mathbf{c}')$ provides an upper bound which helps limit the size of search. The re-encoded codeword may contain a smaller number of errors in the MRB bits than the MRB bits corresponding to the received vector $\mathbf{r}$. Thus using input vector $\hat{\mathbf{r}}'$ leads to more successful decoding by an OSD decoder with smaller order i than using input vector $\mathbf{r}$.

A. Soft Iterative Output $\hat{\mathbf{r}}'$

In general, the a *posteriori* probability (APP) value of L in logarithm for bit m during the iterative decoding process is expressed as the sum of three terms.

$$L_m = L_m^c + L_m^a + L_m^e \tag{3}$$

where L_m^c denotes the *channel measurement*, which is the effect of channel output corresponding to bit m. L_m^a represents the a *priori* value in logarithm, it is the function of the a priori probability of bit m. The final term is the independent *extrinsic knowledge* about bit m. In terms of reliability, the relationship between received bit r_m and updated $\check{r}'_m$ according to the soft output r'_m from the iterative decoder is expressed as

$$\check{r}'_m = r_m \cdot \frac{\log\left(\frac{p(c_m=0|r'_m)}{p(c_m=1|r'_m)}\right)}{\log\left(\frac{p(c_m=0|r_m)}{p(c_m=1|r_m)}\right)} = r_m \cdot \frac{L_m}{L^c_m + L^a_m} \tag{4}$$

Thus as $\check{Y}_{max}$ is limited to $\sum_{j=0}^{n-1} |\check{r}'_j|$, and there may exist $\check{Y}_{max} \neq Y_{max}$ after a certain number of iterations. The cross correlation of this vector with the transmitted codeword $\check{Y}(\mathcal{C}_1)$ might result in a reduction compared to the original received vector. Correspondingly the set of codewords $\mathcal{C}_2$, which produce ML decoding errors, will be increased in size as a new set $\check{\mathcal{C}}_2$. The following possibilities exist

- If $\check{Y}(\mathcal{C}_1)$ is reduced, $|\check{\mathcal{C}}_2| > |\mathcal{C}_2|$, where $|\mathcal{C}_2|$ denotes the size of the codeword set $\mathcal{C}_2$.
- Or if $\check{Y}(\mathcal{C}_1)$ increases, $|\check{\mathcal{C}}_2| < |\mathcal{C}_2|$.

Let $\mathbf{c}$ be a codeword generated by the corresponding generator matrix $\check{\mathbf{G}}$, $\mathbf{c} \in \{\check{\mathcal{C}}_2 \backslash \mathcal{C}_2\}$. Then $\check{Y}(\mathcal{C}_1) < \check{Y}(\mathbf{c}) < Y(\mathcal{C}_1) < Y(\mathcal{C}_2)$, where $\check{Y}$ denotes the cross-correlation of a codeword based on $\check{\mathbf{r}}'$, and Y is the cross-correlation of the same codeword based on $\mathbf{r}$. Once a codeword $\mathbf{c}$ is found, with higher cross-correlation than the transmitted codeword then there is a ML decoding error based on $\check{\mathbf{r}}'$. However this is not a real ML decoding error because if the received vector $\mathbf{r}$ is used the cross correlation of codeword $\mathbf{c}$ is less than the cross correlation of the transmitted codeword. In practice, it has been observed that this is a common event and the decoding performance is significantly degraded through this mechanism.

Besides the above issue, there is another reason that the unconditioned soft iterative output is not an ideal input for decoding. Stopping and trapping sets, considered in the LDPC codes' design has become a major issue due to the degraded performance in AWGN channel at high SNR [4]. These are not only due to cycles of length 4, but also cycles of length 6 and 8. If there exists a set of bits, which form a cycle, then the extrinsic information of these bits can be destructive as the BP decoder iterates. Thus the sign and magnitudes of bit log likelihood ratios can change for the worse as the BP decoder iterates leading to some bits in the LRB of the received vector swapping for bits in the MRB of the received vector leading to an increase in the number of bit errors in the MRB. This will seriously affect the performance of OSD-i due to its order limitation. The similar phenomenon was also observed in [10].

The frequency distributions of the number of bit errors in the MRB as a result of iterative decoding after different iterations are shown in Figure 2. These were obtained by evaluating 10^6 received vectors for Tanner codes $(155, 64, 20)$ [13] at 4dB $\frac{E_b}{N_o}$. The x-axis shows the number of errors in the MRB input to the OSD-i decoder, and the y-axis denotes the number of input vectors having the same number of errors in MRB. First of all, as the iteration is increased, the number of blocks with null error in MRB is increased. It clearly shows the better performance could be achieved as more iterations are called.

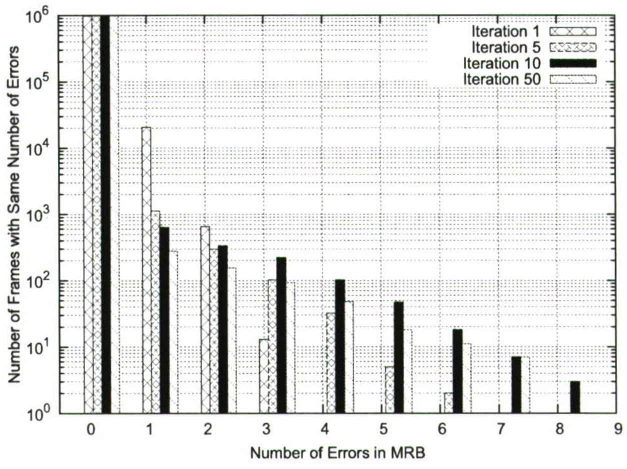


Fig. 2. Soft iterative output from BP decoder for Tanner codes $(155, 64, 20)$ at Eb/No=4dB

After the first iteration, the maximum number of errors in MRB is only 3, but after 5 iterations, the maximum number of errors in MRB is increased to 6. Furthermore at 10 iterations, it produces the maximum number of errors as 8 in MRB, even after 50 iterations, the maximum number of errors in MRB is still 7. Thus for the 10^6 blocks, all of them could be successfully decoded after 1 iteration by OSD decoder up to 3, but part of them might have to be decoded by OSD decoder up to 7 after 50 iterations. It clearly shows that the discussed issue occurs more frequently as the BP decoder iterates. And more error bits with higher magnitudes are shifted to the MRB.

B. Conditioned iterative Output $\hat{\mathbf{r}}'$

In order to avoid the above issues, we need to maintain Y_{max} and yet use the iterative decoder output to assist OSD-i decoding, the conditioned output $\hat{\mathbf{r}}'$ of the iterative decoder is used for OSD-i decoding. The conditioned output $\hat{r}'_m$ is defined as

$$\hat{r}'_m = \begin{cases} r_m & , if \frac{r'_m}{r_m} \geq 0 \\ -r_m & , if \frac{r'_m}{r_m} < 0 \end{cases} \tag{5}$$

Thus

$$\hat{Y}_{max} = \sum_{j=0}^{n-1} |\hat{r}'_j| = Y_{max} \tag{6}$$

According to (3), it is noted that the log likelihood ratio $\check{L}_m$ from $\check{\mathbf{r}}'$ is equivalent to L_m. In this case, the extrinsic knowledge about bit m is evaluated as either "1" or "-1". Thus the conditioned iterative APP value $\hat{L}_m$ log likelihood ratio is depicted as

$$\hat{L}_m = L^c_m + L^a_m + \log(-1)^{\hat{b}'_m} \tag{7}$$

where $\hat{b}'_m$ is the hard-decided binary bit, $\hat{b}'_m \in \{0, 1\}$, according to $\hat{r}'_m$.

Referring to (7), the magnitudes of $\check{L}_m$ and $\hat{L}_m$ can be significantly different. The factor of $\check{L}^e_m$ for $\check{L}_m$ is ranged in the entire real field $\mathbb{R}$. On the other hand, the factor of $\hat{L}^e_m$ for $\hat{L}_m$ is only ranged in $\{-2(L^c_m + L^a_m), 0\}$, which just changes the sign of the reliability. Thus the conditioned

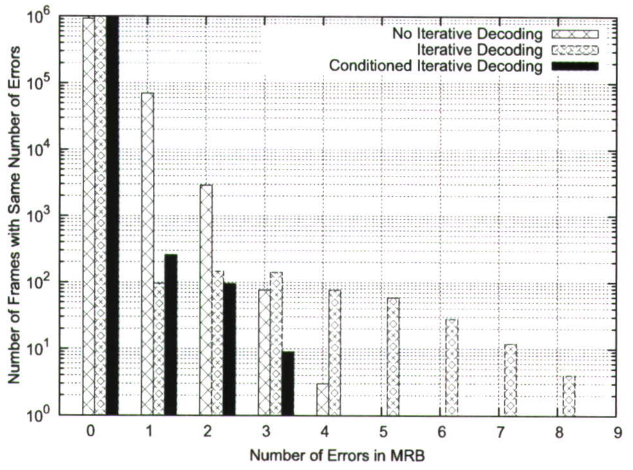


Fig. 3. Comparisons between different inputs to OSD Decoder for EG LDPC codes $(255,175,17)$ at Eb/No=4.5dB, iteration=50

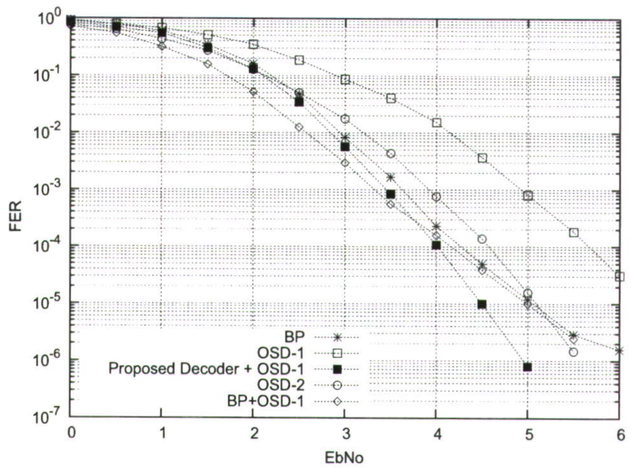


Fig. 4. Comparison results for the regular Gallager codes $(204,102,8)$

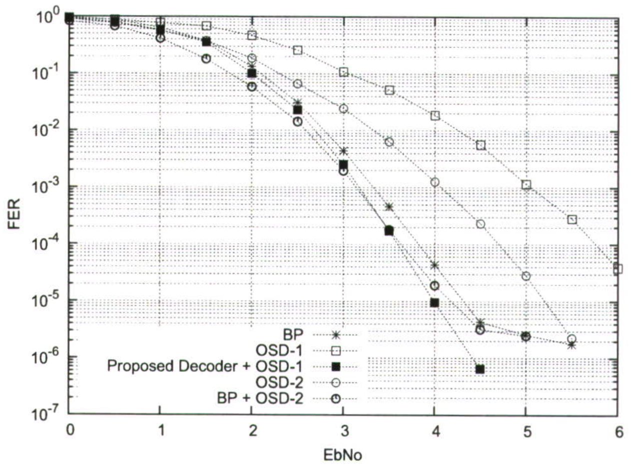


Fig. 5. Comparison results for PEG LDPC codes $(256,128,17)$

iterative output not only ensures the constancy of maximum attainable correlation, it also provides an initial tight range for the OSD decoder with the same MRB to help reduce the search size and computation complexity. Furthermore, the conditioned output reduces the number of errors in MRB, which also reduces the codeword search size.

The frequency distributions of the number of bit errors in the MRB as a result of iterative decoding, conditioned iterative decoding and no iterative decoding are shown in Figure 3. These were obtained by evaluating 10^6 received vectors for the $(255,175,17)$ [14] *Euclidean geometry* (EG) LDPC codes at 4.5dB $\frac{E_b}{N_o}$ with 50 iterations of the iterative decoder. The number of errors j in the MRB dictates the order i of the OSD-i that is necessary for successful decoding. If j is greater than i then decoder errors are certain. Thus Figure 3 may be used to estimate the probability of decoder error for the different decoding arrangements. It can be seen that the worst input is the output from the iterative decoder with the highest number of errors in the MRB from 4 to 8. The best input with the smallest number of errors in the MRB is the conditioned output from the iterative decoder. There is another benefit of using the conditioned output in that the number of blocks decreases faster as the number of errors in the MRB increases, in comparison to the other inputs. It obviously provides a better convergent performance in terms of decoder computation complexity and performance, which means more error-blocks could be successfully decoded by using a smaller OSD-i decoder with less search size.

IV. Results

The results achieved by the different decoder arrangements for regular Gallager codes $(204,102,8)$ [2] are shown in Figure 4. The iterative BP decoder uses 50 iterations. Although the simulated code has a small d_{min}, it has small multiplicity of 50 in total, in low weight codewords up to hamming weight 15. Thus BP decoding performs quite well with more than 1.5dB coding gain compared to the OSD-1 decoder and better performance than the OSD-2 decoder before the error floor region. The proposed decoder using OSD-1 achieves significant improvements in performance than the OSD decoder alone and guarantees the free error-floor performance than BP decoder and BP decoder with OSD-2 decoding. There exists more than 1.5dB coding gain between the proposed decoder with OSD-1 and OSD-1 decoder alone since 3dB $\frac{E_b}{N_o}$.

The *progressive edge-growth* (PEG) LDPC code proposed by Hu [15], provides a better performance within the similar codeword length due to its strong d_{min}. The results for the PEG LDPC code $(256,128,17)$ using 50 iterations for the BP decoder are shown in Figure 5. The BP decoder achieves about 1dB coding gain at high SNR from 3.5dB $\frac{E_b}{N_o}$ than the OSD-2 decoder, but degrades due to the stopping sets since 4.5dB $\frac{E_b}{N_o}$, where the error floor occurs. The proposed decoder paired with OSD-1 produces significantly better performance than the OSD-2 decoder at any signal-noise-ratio and successfully breaks the error floor caused by the BP decoder and BP decoder paired with OSD-2.

The results for the *Projective geometry* (PG) cyclic (341,205,16) LDPC codes [14] are shown in Figure 6. The iterative BP decoder is achieved by exploring the entire n parity-check equations based on the codes' cyclic property, thus the improvement is significant with more than 1dB coding

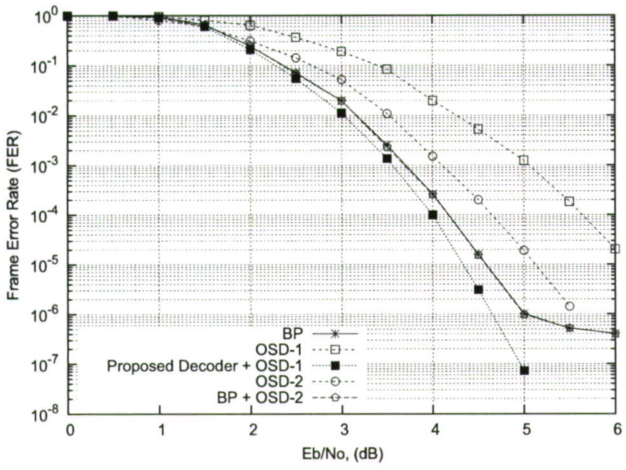


Fig. 6. Comparison results for the PG LDPC code $(341, 205, 16)$

gain than the OSD-1 decoder from 3dB $\frac{E_b}{N_o}$. The error-floors using BP decoding and BP decoding paired with OSD-3 occur at 5dB $\frac{E_b}{N_o}$. The proposed decoder with OSD-1 achieves the best decoding performance with free error-floor.

Those results provide good evidence that the proposed decoder with OSD-i, especially OSD-1, achieves better performance than OSD decoding alone and shows no sign of an error floor unlike BP decoding coupled with OSD decoding or stand-alone BP decoding. The proposed decoder with OSD-i crosses the OSD-$(i+1)$ decoding performance at high SNR and involves much less decoder complexity.

V. Conclusions

A new decoding arrangement for LDPC codes with sparse parity-check matrices over AWGN channel has been introduced which uses a conditioned output from a BP decoder instead of the standard soft output. Results have been presented which show that the proposed decoder using OSD-1 completely solves the error floor problem associated with BP decoding of LDPC codes. Also, compared to stand-alone OSD-i decoding, the proposed decoder using OSD-$(i-1)$ decoding performs better at high SNR and involves much less decoder complexity.

References

[1] R. Gallager, *Low-Density Parity-Check Codes*, Cambridge, MA: MIT Press, 1963.

[2] D. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Feb 1999.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding: Turbo codes," in *Proc. IEEE International Conference on Communications*, Geneva, Switzerland, 23–26 May 1993, pp. 1064–1070.

[4] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, Feb 2001.

[5] S. Y. Chung, G. D. Forney, Jr. T J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045db of the shannon limit," *IEEE Comm. Letters*, vol. 3, pp. 58–60, Feb 2001.

[6] M. P. C. Fossorier and S. Lin, "soft-decision deocidng of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, Sep 1995.

[7] M. P. C. Fossorier, "Reliability-based soft-decision decoding with iteartive information set reduction," *IEEE Transactions on Information Theory*, vol. 48, no. 12, pp. 3101–3106, Dec 2002.

[8] A. Valembois and M. P. C. Fossorier, "Box and match techniques applied to soft-decision decoding," *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 796–810, May 2004.

[9] B. Dorsch, "A decoding algorithm for binary block codes and j -ary output channels," *IEEE Transactions on Information Theory*, vol. 20, pp. 391–394, May 1974.

[10] M. Tomlinson, C. Tjhai, and M. Ambroze, "Extending the dorsch decoder towards achieving maximum-likelihood decoding for linear codes," *IET Communications*, vol. 1, no. 3, pp. 479–488, 2007.

[11] Li Yang, Marcel Ambroze, and Martin Tomlinson, "Comparison of decoding turbo gallager codes in hybrid decoding arrangements with different iterative decoders over the erasure channel," in *the 11th IEEE International Conference on Communications Systems (ICCS)*, Guangzhou, China, Nov 2008.

[12] M. P. C. Fossorier, "Iterative reliability-based decoding of low-density parity-check codes," *IEEE J. Select. Areas of Commun*, vol. 19, pp. 908–917, May 2001.

[13] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured ldpc codes," in *Proc. Int. Symp. on Commun. Theroy and Appl. (ISCTA)*, Ambleside, England, July 2001.

[14] C. Tjhai, M. Tomlinson, M. Ambroze, and M. Ahmed, "Cyclotomic idempotent-based binary cyclic codes," *Electron. Lett.*, vol. 41, no. 3, pp. 341–343, 2005.

[15] X. Y. Hu, E. Eleftheriou, and D. M. Amold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, pp. 386–398, Jan 2005.

Li Yang (S'08) was born in Beijing, China, in 1982. He received his first degree in Computing and master degree in Network System Engineering both from the University of Plymouth, UK, in 2005 and 2006, respectively. Since Oct 2006, he joined the Fixed and Mobile Communications Research Center as a PhD research student. His research interests are in the area of information theory and its applications, which are mainly in turbo codes and LDPC codes design, and their corresponding optimal decoding technique by adapting the iterative decoding algorithms in both binary erasure channel and AWGN channel.

Martin Tomlinson (M'96) is best known for the invention of the Tomlinson Harashima precoding technique. He received his PhD from Loughborough University in 1970 on the subject of adaptive equalisation for data transmission and worked at Plessey Telecommunications Research Ltd until 1975 in digital communication and satellite transmission. He then spent seven years in the Satellite Communications Division of RSRE (now Qinetiq). Professor Tomlinson is currently Head of Fixed and Mobile Communications Research, leading research projects in communications, coding, signal processing, and watermarking. He has published over 200 papers and 50 patents in the fields of Digital Modulation and Coding, Signal Processing, Video Coding and Satellite Communications, as well as contributing to various standards such as the DVB-RCS standard for small terminals.

Marcel A. Ambroze (M'01) received the BEng degree from Technical University of Cluj-Napoca, Romania in 1996 and the PhD in concatenated codes with iterative decoding for data transmission from the University of Plymouth, UK in 2000. He worked as a Research Fellow for two EPSRC projects in Digital Watermarking and Signal Processing for Small Satellite Earth Terminals at the University of Plymouth until 2003. Since 2003 he was employed as a Lecturer in Digital Communications Systems at the University of Plymouth. He is currently a member of the Fixed and Mobile Communications Research at the University of Plymouth, lead by Professor Martin Tomlinson. He has published over 30 papers in the fields of Error Correction Coding, Watermarking and Satellite Communications. His current research interests are in error correction coding and iterative decoding, signal processing and coding for wireless systems and watermarking as communications with side information at the transmitter.

Extended Optimum Decoding for LDPC Codes based on Exhaustive Tree Search Algorithm

Li Yang, Martin Tomlinson, Marcel Ambroze
Fixed and Mobile Communications Research
University of Plymouth
PL4 8AA, United Kingdom
Email: li.yang, martin.tomlinson, marcel.ambroze@plymouth.ac.uk

***Abstract*—A maximum-likelihood decoding algorithm inspired by information set decoding is realised for LDPC codes or linear block codes with sparse parity-check matrices of moderate codeword lengths over the AWGN channel. The extension involves an exhaustive branch and bound tree-search based algorithm for finding all small Euclidean distance error vectors. It provides an approach of searching on the bounded n-bit positions, which involves 2^k combinatorial dual codes based on the corresponding parity-check matrix, to achieve the near-optimum decoding output with attainable computational complexity. Soft decision decoding results are presented for some well-known LDPC codes demonstrating near-optimum maximum-likelihood performance.**

I. Introduction

Low-density parity-check (LDPC) codes as classical linear block codes were first introduced by Gallager [1] in 1963, and such class of codes attracted a great deal of interest in recent years following their rediscovery by MacKay [2] in 1999. Using message passing decoding algorithm (*belief propagation*, BP) in iterative decoding performance, the results are comparable to turbo codes' [3].

In terms of the optimal decoding performance or *maximum likelihood* (ML), LDPC codes with ML decoding in general is not feasible. For the AWGN channel, most of the optimal or near optimal decoding algorithms are only practical for relatively short codes. In 1995, Fossorier rediscovered the information set decoder proposed by Dorsch [4], with the name called "*ordered statistics decoder*" (OSD) algorithm [5], it was to search the ordered most-reliable k information bits for the maximised codeword with constraint of order i. A better arrangement improving the performance with same order of i was proposed in [6]. Valembois [7] based on Fossorier's previous work proposed the "box and match" technique to further improve the decoder performance. An extended Dorsch decoder was proposed by Tomlinson [8] in 2007, it was towards achieving the optimal or near-optimum decoding performance for linear block codes by generating a series of low weight error vectors in the information part to incrementally search the closest codeword according to the cross correlation with the received vector.

Recently different effective methods have been introduced to find low weight codewords for a binary linear block code. Based on the knowledge of computing the minimum distance of LDPC codes, some efficient algorithms for finding the minimum-size stopping sets of LDPC codes were introduced by following different approaches. An "error impulse" (EI) based algorithm to find the small size stopping sets was proposed by Richter [9]. Hirotomo [10] proposed a probabilistic based algorithm to find the minimum-size stopping sets of LDPC codes. The probabilistic methods were superseded by Wang [11] who proposed the first exhaustive search based algorithm to find all low weight vectors, which include codewords, stopping sets and k-out trapping sets, for LDPC codes. Furthermore, based on previous work [12], [13] concerning an exhaustive search for turbo codes' stopping sets, Rosnes proposed an efficient algorithm to find all the low weight stopping sets, including codewords, for LDPC codes [14]. The idea was taken further by Ambroze [15], optimising the algorithm in a tree search branch and bound approach.

The searched error vector introduced by Tomlinson [8] consists of the hard-decided received vector bit-wised by a codeword derived from re-encoded codeword according to the k nearly ordered most-reliable information bits and a codeword with low-weight information. Processing this error vector produces the most likelihood codeword with the closest Euclidean distance compared to the received vector. In this paper an algorithm for searching all low-weight error patterns of a received vector for LDPC codes is proposed.

This paper is organised as follows: general code representations and definitions are described and the exhaustive search algorithm for low weight codewords is described in Section II. Section III introduces the modified expanded Dorsch decoding algorithm coupled with branch and bound, exhaustive search for finding low-weight codewords. Some considerations to optimise the search algorithm are discussed. In Section IV, the extended algorithm for finding low weight error vectors is described in terms of a new lower bound algorithm. Section V compares results of iterative decoding performance, sub-optimum decoding performance and optimal decoding performance for some linear codes with sparse structure and reasonable codeword length. Section VI concludes the research work.

II. Preliminaries

In common with earlier papers [14], [15], similar notations and preliminaries are used in this paper.

A. Code Representation

Let C be a binary linear code with length n, $n \in \mathbb{N}$, and k-dimensional subspace of $\{0,1\}^n$. The linear code C can be represented by a $m \times n$ binary parity-check matrix $\mathbf{H}$, where $m \geq n-k$. Let $\mathbf{x} = \{x_0,...,x_{n-1}\} \in \{0,1\}^n$ be the transmitted vector, then bits $\mathbf{x}$ satisfy $\mathbf{Hx} = 0$. In equivalent graphical representation, the linear block code C, especially for LDPC codes, may be represented by a *bipartite* graph, called a Tanner graph [16]. A bipartite graph comprises a vertex set of *variable* nodes $\mathbf{V} = \{v_0,...,v_{n-1}\}$, and a vertex set of *check* nodes $\mathbf{C} = \{c_0,...,c_{m-1}\}$. The variable nodes correspond to the columns of $\mathbf{H}$, and the check nodes correspond to the rows of $\mathbf{H}$. h_i represent the ith parity-check equation of $\mathbf{H}$, H_{ij} represents the connection relationship between the variable node v_j and check node c_i. It is defined that there exists an connected edge between variable node v_j and check node c_i, if $H_{ij} = 1$.

Let S denote a subset of $\{0,1\}^n$, the set of all binary vectors of length n. At any point during the tree search process, a *constraint set*, $\mathcal{F}$ is defined to consist of bit positions p_i and the states of these bit positions s_{p_i}, $s_{p_i} \in \{0,1\}$. The support set $\mathcal{X}(\mathcal{F})$ of the constraint set $\mathcal{F}$, is the set of positions where $s_{p_i} = 1$ and the Hamming weight $w(\mathcal{F})$ of $\mathcal{F}$ is the total number of such positions. The sub-matrix $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$ is defined by the columns of $\mathbf{H}$ where $s_{pi} = 1$, and the row weight of $h_i^{\mathcal{X}(\mathcal{F})}$ is the total number of $1's$ at row i. A row of h_i is considered as an *active* row of $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$, if $w(h_i^{\mathcal{X}(\mathcal{F})})$ has odd row weight. If a constraint set $\mathcal{F}$ with size of n contains no active rows of $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$, then the set $\mathcal{F}$ forms a codeword with weight $w(\mathcal{F})$. Thus a *codeword* set is a subset of $\{x_0,...,x_{n-1}\}$ with Hamming weight d, $d = w(\mathcal{F})$, where the induced subgraph contains no check node with odd degree. The minimal Hamming distance d_{min} is defined as the minimal weight of the non-empty codeword set. If there exist active rows, then $\mathcal{F}$ has either to be appended with additional bit positions if $|\mathcal{F}| < n$, or one or more states s_{p_i} need to be changed to compensate the active rows with odd row weight.

B. Tree-search based Codeword Set Enumeration Algorithm

Since we aim to find codewords excluding stopping sets, the search criteria can be simplified based on [15]. The algorithm termed, the Tree-search based Codeword Set Enumeration (TCSE) algorithm is designed to search all the codeword set up to size of threshold τ for any parity-check code C.

In the equivalent graphical tree view representation, the constraint set can be considered as a *branch* of the tree, which represents the set of searched known bit-positions of a code C. $|\mathcal{F}|$ denote the size of $\mathcal{F}$, representing the known depth of the tree. Since the active row is defined as the row with odd weight, thus the set of active rows in $\mathbf{H}$ is denoted by $\{\mathfrak{h}_0,...,\mathfrak{h}_{\phi-1}\}$, where ϕ is the total number of active rows. A constraint set $\mathcal{F}$ with size n is said to be *valid* if and only if there exists no active row in $\mathbf{H}_{\mathcal{X}(\mathcal{F})}$, in other word, a codeword is found. The pseudocode of the algorithm to find all the codeword sets by threshold size τ is given in Algorithm 1.

Algorithm 1 Tree-search based Codeword Set Enumeration (TCSE)

```
repeat
  Pick one untouched branch as a constraint set $\mathcal{F}$.
  if $|\mathcal{F}| = n$ and $w(\mathcal{F}) \leq \tau$ then
    Constraint set $\mathcal{F}$ is saved, if $\mathcal{F}$ is valid
  else
    1). Pass $\mathcal{F}$ to the iterative decoder with erasures in the unconstrained positions.
    2). Construct a new constraint set $\mathcal{F}'$ with new decoded positions, which is the extended branch.
    if $|\mathcal{F}'| = n$ and $w(\mathcal{F}') \leq \tau$ then
      Constraint set $\mathcal{F}'$ is saved, if $\mathcal{F}'$ is valid
    else if No contradiction is found in $\mathbf{H}_{\mathcal{X}(\mathcal{F}')}$, and $w'(\mathcal{F}') \leq \tau$ then
      a). Pick an unconstrained position $p$.
      b). Extend the branch $\mathcal{F}'$ to position $p$ to get new branch $\mathcal{F}'' = \mathcal{F}' \bigcup \{(p,1)\}$ and branch $\mathcal{F}''' = \mathcal{F}' \bigcup \{(p,0)\}$.
    end if
  end if
until Tree has been fully explored by threshold $\tau$
```

When the whole tree has been explored constrained by the lower bound, the list of the saved constraint sets in full size n is the whole set of codewords up to size of τ. The contradiction is considered as that if $w(h_i^{\mathcal{X}(\mathcal{F})}) = 1$ and $w(h_i'^{\mathcal{X}(\mathcal{F})}) = 0$, where $h_i'^{\mathcal{X}(\mathcal{F})}$ is the row weight of row i for the unconstrained positions $\{p_i : (p_i, 1)\} \in \{0,1,...,n-1\} \backslash \mathcal{F}$ intersected by row i on $\mathbf{H}$. The iterative BP decoder over the erasure channel is considered as the candidate iterative decoder. It should be noted that it is not necessary to call the iterative decoder unless the condition of $w(h_i'^{\mathcal{X}(\mathcal{F})}) = 1$ is met. Thus it helps reduce the computing complexity to call the decoder for every new branch $\mathcal{F}'$. The lower bound algorithm $w'(\mathcal{F}')$and position selection criteria follow the same rule as in [15].

III. Algorithm inspired by Dorsch algorithm

The codeword search approach of the proposed algorithm is different from the the approaches in [4], [8], a set of codewords satisfying the parity-check matrix $\mathbf{H}$ is explored instead of the codes generated by the re-ordered nearly most-reliable k information bits. Furthermore, the number of search size might be constrained by a updated threshold. The BPSK-modulated codeword $\mathbf{x}$ is transmitted through the AWGN channel and received as the vector $\mathbf{r} = \{r_0,...,r_{n-1}\} \in \mathbb{R}$, which is affected by noise with variance σ^2. A hard-decided received vector $\mathbf{b} \in \{0,1\}^n$ is derived from the received vector $\mathbf{r}$ using bit-wise decisions. Usually, the binary vector $\mathbf{b}$ is not a codeword. The re-encoded codeword $\hat{\mathbf{x}}$ is produced by $\mathbf{H}$ according to the binary information set of $\mathbf{b}$. The maximum attainable correlation Y_{max} with the received vector is given by

$$Y_{max} = \sum_{j=0}^{n-1} |r_j| \quad (1)$$

According to the extended Dorsch decoding algorithm [8], the codeword $\tilde{\mathbf{x}}_i$ with low Hamming weight is used to find the closest codeword $\mathbf{x}_i$, which has the highest cross-correlation corresponding to the received vector $\mathbf{r}$, such codeword is given by

$$\mathbf{x}_i = \hat{\mathbf{x}} \oplus \tilde{\mathbf{x}}_i \quad (2)$$

Then the corresponding binary error vector $\mathbf{z}_i$ is given by

$$\mathbf{z}_i = \mathbf{b} \oplus \hat{\mathbf{x}} \oplus \tilde{\mathbf{x}}_i \quad (3)$$

The first error vector $\hat{\mathbf{z}}$ is defined as

$$\hat{\mathbf{z}} = \mathbf{b} \oplus \hat{\mathbf{x}} \quad (4)$$

Thus the cross-correlation cost $Y(\mathbf{x}_i)$ is given by

$$Y(\mathbf{x}_i) = Y_{max} - Y_{\triangle}(\mathbf{x}_i) \quad (5)$$

where $Y_{\triangle}(\mathbf{x}_i)$ is defined by

$$Y_{\triangle}(\mathbf{x}_i) = 2\sum_{j=0}^{n-1} (\hat{z}_j \oplus \tilde{x}_{i_j}) \times |r_j| \quad (6)$$

In order to find the codeword $\mathbf{x}_i$ with smallest cross-correlation reduction from Y_{max}, which is the smallest difference between Y_{max} and $Y(\mathbf{x}_i)$, then we find the smallest $Y_{\triangle}(\mathbf{x}_{min})$, starting with $Y_{\triangle}(\hat{\mathbf{x}})$

$$Y_{\triangle}(\mathbf{x}_{min}) = \min(Y_{\triangle}(\mathbf{x}_i)) \quad (7)$$

This may be achieved by searching for the low weight codeword $\mathbf{x}_i$ using the TCSE algorithm.

IV. Simplified Approach Based on TCSE Algorithm

At low SNR, the low weight codeword search has to proceed until the threshold is reached, and the simple method bound does not provide a sufficiently tight lower bound to limit the size of codeword search. Another approach based on the basic algorithm is proposed. It is to use TCSE algorithm to search for an error vector $\tilde{\mathbf{e}}_i$, which contains an index set $\mathcal{I}_{\tilde{\mathbf{e}}_i}$ consisting of positions, where their values are different from received binary vector $\mathbf{b}$. Thus the error vector $\tilde{\mathbf{e}}_i$ is given by

$$\tilde{e}_{i_j} = \begin{cases} |b_j - 1|, & \text{if } j \in \mathcal{I}_{\tilde{\mathbf{e}}_i} \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

The cross-correlation difference $Y_{\triangle}(\tilde{\mathbf{e}}_i)$ corresponding to Y_{max} is given by

$$Y_{\triangle}(\tilde{\mathbf{e}}_i) = 2\sum_{j=0}^{n-1} (\tilde{e}_{i_j}) \times |r_j| \quad (9)$$

The smallest correlation difference $Y_{\triangle}(\mathbf{e}_{min})$ is given by

$$Y_{\triangle}(\mathbf{e}_{min}) = \min(Y_{\triangle}(\tilde{\mathbf{e}}_i)) \quad (10)$$

Then the constraint set $\mathcal{F}$ is redefined as a set with $s_{p_j} = |b_{p_j} - 1|$, which produces extra correlation cost $\mathcal{R}(\mathcal{F}) = \sum_{j=0}^{|\mathcal{F}|}(|r_{p_j}|)$. $\mathcal{F}$ is said to be *valid* if and only if $\mathbf{H}(\hat{\mathbf{e}}_{\mathcal{F}} \oplus \mathbf{b}) = 0$, where $\hat{\mathbf{e}}_{\mathcal{F}}$ is the error vector corresponding to constraint set $\mathcal{F}$ with extra cost less than threshold $Y_{\triangle}(\mathbf{e}_{min})$. In other words, there is no active row for $(\hat{\mathbf{e}}_{\mathcal{F}} \oplus \mathbf{b})$ on $\mathbf{H}$. The simplified tree-search based error vector enumeration (STESE) is depicted in Algorithm 2 to search all the error vectors with extra cost up to $Y_{\triangle}(\mathbf{e}_{min})$, which might be initialised as $Y_{\triangle}(\hat{\mathbf{z}})$. The simplified algorithm is designed to find all the potential error-vector sets with extra cost less than threshold. Once a smaller error vector is found, then the threshold is updated. Thus the search size of the error vectors is reduced as the threshold decreases. $\mathcal{R}'(\mathcal{F})$ is designated to compute the lower bound of potential cost corresponding to $\mathcal{F}$.

Algorithm 2 Simplified Tree-search based Error-vector Set Enumeration (STESE)

```
repeat
  Pick one untouched branch as a constraint set $\mathcal{F}$.
  if $\mathcal{R}(\mathcal{F}) < Y_{\triangle}(\mathbf{e}_{min})$ then
    1). Constraint set $\mathcal{F}$ is picked, if $\mathcal{F}$ is valid.
    2). And $Y_{\triangle}(\mathbf{e}_{min})$ is updated as $\mathcal{R}(\mathcal{F})$
  else
    Active rows set $\mathfrak{h}_{\mathcal{F}}$ is collected
    if $\mathcal{R}'(\mathcal{F}) \le Y_{\triangle}(\mathbf{e}_{min})$ then
      a). Pick an unconstrained position $p$.
      b). Extend the branch $\mathcal{F}$ to position $p$ to get new
      branch $\mathcal{F}' = \mathcal{F}\bigcup\{(p, |b_p - 1|)\}$.
    end if
  end if
until Tree has been fully explored by updated $Y_{\triangle}(\mathbf{e}_{min})$
```

A. Computing $\mathcal{R}'(\mathcal{F})$ on $\mathcal{F}$

The simple method bound given in [15] computes the intersected positions by active rows ordered by column-weight. Each ordered position with its column-weight is deducted from the total number of active rows ϕ until all active rows are compensated. Thus the number of deducted positions is the least possible bound, which guarantees that no potential p missed. The proposed bound algorithm is to estimate the potential coordinates where their column weights could compensate the active rows, meanwhile their total cost has to be less than the difference between threshold and $\mathcal{R}(\mathcal{F})$. Let $\mathcal{I}(\mathcal{F}) = \{I_{i_0}(\mathcal{F}), ..., I_{i_{q-1}}(\mathcal{F})\}$ be the active rows index set, where $I_{i_0}(\mathcal{F})$ is the set of active rows on $\mathcal{F}$ corresponding to the i_0th column of $\mathbf{H}$, and q is the total number of intersected positions. First of all, the bound is designed to order the intersected positions by their normalised cost, which is its cost $|r_{i_0}|$ divided by the column weight $w(I_{i_0})$. Then it follows the same strategy, the least ordered position is picked. Its column weight is deducted and the threshold is checked, it is repeated until all active rows are compensated or threshold is exceeded. During observations, the lower bound algorithm can not provide the least probable bound, as the normalised position can not guarantee to attain all the potential error patterns. Such issue could be the involved positions actually

Algorithm 3 Lower bound algorithm to find a satisfied position-combination

```
repeat
  Pick one unflagged position as a list set $\mathcal{L}$.
  if $(\mathcal{R}(\mathcal{L}) + \mathcal{R}(\mathcal{F})) < Y_{\triangle}(\mathbf{e}_{min})$ and $w(\mathcal{L}) \geq \phi$ then
    1). The satisfied position-combination is found.
    2). Break.
  else if $w(\mathcal{L}) < \phi$ then
    a). Pick an unflagged position $p$ based on $\eta(\mathcal{L})$, where
    $(\mathcal{R}(\mathcal{L}) + \mathcal{R}(\mathcal{F}) + |r_p|) < Y_{\triangle}(\mathbf{e}_{min})$.
    b). Extend the list $\mathcal{L}$ with position $p$ to get new $\mathcal{L}'$ and
    $\eta(\mathcal{L}') = (\eta(\mathcal{L}) \bigcup p)$.
  end if
until Combination is found or maximum iteration number is reached
```

could produce different combinations to satisfy the check criteria without in any arranged order.

Thus a lower bound algorithm is realised to find a possible position combination to satisfy the requirement of all active rows' compensation and extra allowed cost. Let $\mathcal{L}$ be a list set $\in \{0, ..., n-1\}$ to store positions' indices. $\eta(\mathcal{L})$ is denoted as a flag set to flag the positions on $\mathcal{L}$, it consists of the positions selected in $\mathcal{L}$ and the excluded positions. The excluded position is the position p with $(|r_p| + \mathcal{R}(\mathcal{F})) \geq Y_{\triangle}(\mathbf{e}_{min})$. The position combination search algorithm is inspired by the codeword search algorithm, which follows the similar strategy. The proposed position-combination search algorithm as a lower bound is depicted in Algorithm 3, where $w(\mathcal{L})$ represents the total number of column weight corresponding to the positions in $\mathcal{L}$. The allowed cost based on $\mathcal{F}$ decreases, as the $\mathcal{R}(\mathcal{F})$ reaches the threshold. Once the qualified number of intersected positions gets smaller, then the iterations in finding a position-combination gets faster and easier to determine. At the beginning stage of extending the search on the tree, there might exist a relative large number of positions based on the active rows, even though $\mathbf{H}$ is sparse. With the increase of the codeword length, the complexity of computing the lower bound is significantly increased, a limitation on the iteration is required to stay within a reasonable computational complexity. The position for the extended tree search from computing on the lower bound is ideal to pick the position in the approved combination with highest column-weight and least cost at the same column weight.

V. Numerical Results

The simulated code is the assorted $(3, 6)$ regular Gallager code $(120, 56, 10)$ [2], [17], which provides a very sparse parity-check matrix. The corresponding results of the code with iterative output, optimum output and sub-optimal OSD-i output are shown in Fig. 1. The BP iterative decoding output is achieved with 50 iterations and has better performance than OSD-1 with 0.5dB additional coding gain at 4.5dB $\frac{E_b}{N_o}$. The decoder uses the algorithm described above with a limitation of 10^7 nodes search and 500 iterations of position combination

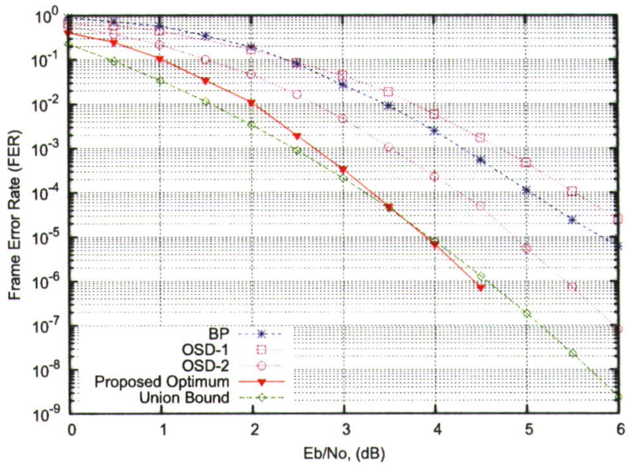


Fig. 1. FER Performance of Assorted Regular Gallager Code $(120, 56, 10)$

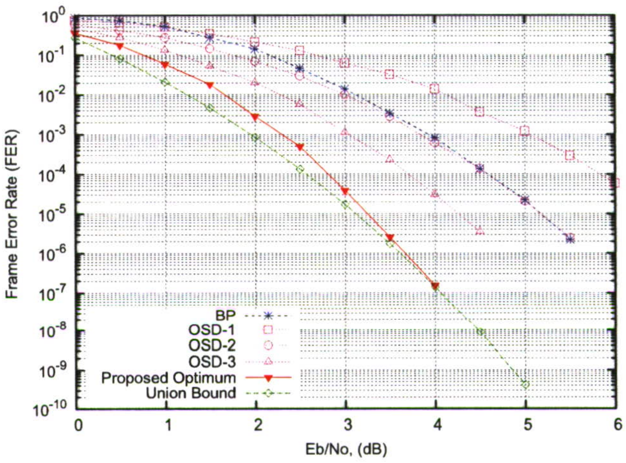


Fig. 2. FER Performance of Tanner Code $(155, 64, 20)$

search for the lower bound. With such decoding arrangement, the decoder successfully achieves near-optimum performance compared with the computed union bound.

The Tanner code $(155, 64)$ is a $(3, 5)$-regular LDPC code constructed by Tanner [18]. The underlying Tanner graph has girth 8 and its relatively large minimum distance of 20 makes the code an excellent candidate for iterative decoding. The results of the code with iterative output, optimum output and sub-optimal OSD-i output are shown in Fig. 2. The optimum decoder is with the proposed algorithm with limitation of 1×10^8 nodes search. Furthermore, the lower bound is limited to 500 iterations during the position combination search, which is sufficient to find the maximised codeword. From the results it can be seen that the iterative decoding performance achieves similar results to the OSD-2 decoder. As the order number i increases, the sub-optimal decoding performance is significantly improved by more than 1dB additional coding gain for each increased order number. And the near-optimum decoding performance is successfully achieved by the proposed tree-search based error-vector searching decoder using 500 iterations of position combination search and 10^8 maximum nodes search.

The near-optimum decoding performance approaches closely to the computed union bound at 4dB $\frac{E_b}{N_o}$.

VI. Conclusion

In this research, a new approach of achieving maximum-likelihood achievement based on the Dorsch decoder is introduced. The approach utilises an efficient exhaustive tree search algorithm to find a low-weight codeword, which is used to find the closest codeword to the received vector. A further optimisation has been described, which is based on the basic algorithm to exhaustively search for a low weight error vector, in order to determine the closest codeword. The practical simulation results have demonstrated that the proposed decoder algorithm is able to approach the near-optimum decoding using a bounded search algorithm. Since the search exploits the sparseness of the parity-check matrix of the code, it is more suitable for linear block codes having very sparse parity-check matrices, such as LDPC codes.

References

[1] R. Gallager, *Low-Density Parity-Check Codes*, Cambridge, MA: MIT Press, 1963.

[2] D. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Feb 1999.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding: Turbo codes," in *Proc. IEEE International Conference on Communications*, Geneva, Switzerland, 23–26 May 1993, pp. 1064–1070.

[4] B. Dorsch, "A decoding algorithm for binary block codes and j -ary output channels," *IEEE Transactions on Information Theory*, vol. 20, pp. 391–394, May 1974.

[5] M. P. C. Fossorier and S. Lin, "soft-decision deocidng of linear block codes based on ordered statistics," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, Sep 1995.

[6] M. P. C. Fossorier, "Reliability-based soft-decision decoding with iteartive information set reduction," *IEEE Transactions on Information Theory*, vol. 48, no. 12, pp. 3101–3106, Dec 2002.

[7] A. Valembois and M. P. C. Fossorier, "Box and match techniques applied to soft-decision decoding," *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 796–810, May 2004.

[8] M. Tomlinson, C. Tjhai, and M. Ambroze, "Extending the dorsch decoder towards achieving maximum-likelihood decoding for linear codes," *IET Communications*, vol. 1, no. 3, pp. 479–488, 2007.

[9] G. Richter, "Finding small stopping sets in the tanner graph of ldpc codes," in *Proc. 4th Int. Symp. on Turbo Codes & Related Topics*, Munich, Germany, Apr 2006.

[10] M. Hirotomo, Y. Konishi, and M. Morii, "A probabilistic algorithm for finding the minimum-size stopping sets of ldpc codes," in *Proc. IEEE Information Theory Workshop*, Porto, Portugal, May 2008, pp. 66–70.

[11] C. C. Wang, S. R. Kulkami, and H. V. Poor, "Finding all small error-prone substructures for ldpc codes," *IEEE Transactions on Information Theory*, vol. 55, pp. 1976–1999, May 2009.

[12] E. Rosnes and O. Ytrehus, "Turbo stopping sets: The uniform interleaver and efficient enumeration," in *IEEE International Symposium on Information Theory*, Adelaide, SA, Australia, Sep 2005, pp. 1251–1255.

[13] E. Rosnes and O.Ythehus, "Turbo decoding on the binary erasure channel: finite-length analysis and turbo stopping sets," *IEEE International Symposium on Information Theory*, Feb 2006.

[14] E. Rosnes and O. Ytrehus, "An algorithm to find all small-size stopping sets of low-density parity-check matrices," in *IEEE International Symposium on Information Theory*, Nice, France, June 2007.

[15] Marcel Ambroze, Martin Tomlinson, and Li Yang, "Exhaustive weight spectrum analysis of some well known ldpc codes," in *the 10th IEEE International Communication Theory and Applications (ISCTA 09)*, Lake District, UK, March 2009.

[16] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. IT-27, pp. 533–547, September 1981.

[17] D. J. C. MacKay, "Encyclopedia of sparse graph codes," Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[18] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured ldpc codes," in *Proc. Int. Symp. on Commun. Theroy and Appl. (ISCTA)*, Ambleside, England, July 2001.