FOUNTAIN CODES AND THEIR TYPICAL APPLICATION IN WIRELESS STANDARDS LIKE EDGE

by

Trienko Lups Grobler

Submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Computer Engineering)

in the

Department of Electrical, Electronic and Computer Engineering

Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

July 17, 2008

Supervisor: Professor J. C. Olivier

SUMMARY

FOUNTAIN CODES AND THEIR TYPICAL APPLICATION IN WIRELESS STANDARDS LIKE EDGE

by

Trienko Lups Grobler

Supervisor: Professor J.C. Olivier

Department of Electrical, Electronic and Computer Engineering

Master of Engineering (Computer)

One of the most important technologies used in modern communication systems is channel coding. Channel coding dates back to a paper published by Shannon in 1948 [1] entitled "*A Mathematical Theory of Communication*". The basic idea behind channel coding is to send redundant information (parity) together with a message to make the transmission more error resistant. There are different types of codes that can be used to generate the parity required, including block, convolutional and concatenated codes.

A special subclass of codes consisting of the codes mentioned in the previous paragraph, is sparse graph codes. The structure of sparse graph codes can be depicted via a graphical representation: the factor graph which has sparse connections between its elements. Codes belonging to this subclass include Low-Density-Parity-Check (LDPC) codes, Repeat Accumulate (RA), Turbo and fountain codes. These codes can be decoded by using the belief propagation algorithm, an iterative algorithm where probabilistic information is passed to the nodes of the graph.

This dissertation focuses on noisy decoding of fountain codes using belief propagation decoding. Fountain codes were originally developed for erasure channels, but since any factor graph can be decoded using belief propagation, noisy decoding of fountain codes can easily be accomplished. Three fountain codes namely Tornado, Luby Transform (LT) and Raptor codes were investigated during this dissertation. The following results were obtained:

1. The Tornado graph structure is unsuitable for noisy decoding since the code structure protects the first layer of parity instead of the original message bits (a Tornado graph consists of more than one layer).

2. The successful decoding of systematic LT codes were verified.

3. A systematic Raptor code was introduced and successfully decoded. The simulation results show that the Raptor graph structure can improve on its constituent codes (a Raptor code consists of more than one code).

Lastly an LT code was used to replace the convolutional incremental redundancy scheme used by the 2G mobile standard Enhanced Data Rates for GSM Evolution (EDGE). The results show that a fountain incremental redundancy scheme outperforms a convolutional approach if the frame lengths are long enough. For the EDGE platform the results also showed that the fountain incremental redundancy scheme outperforms the convolutional approach after the second transmission is received. Although EDGE is an older technology, it still remains a good platform for testing different incremental redundancy schemes, since it was one of the first platforms to use incremental redundancy.

Keywords: articulation principle, belief propagation, Enhanced Data Rates for GSM Evolution(EDGE), factor graph, fountain code, incremental redundancy, Luby-Transform codes(LT), low-density parity-check codes(LDPC), noisy decoding, Raptor codes, sparse graph, Tornado codes.

SAMEVATTING

FONTEIN KODES EN HUL TIPIESE GEBRUIK IN DRAADLOSE STANDAARDE SOOS VDGE
deur
Trienko Lups Grobler
Studieleier: Professor J.C. Olivier
Departement Elektriese, Elektroniese en Rekenaar Ingenieurswese
Meester in Ingenieurswese (Rekenaar)

Een van die belangrikste tegnologieë wat gebruik word in kommunikasie stelsels is kanaalkodering. Kanaalkodering dateer terug na die publikasie van Shannon in 1948 [1] getiteld "*A Mathematical Theory of Communication*". Die basiese idee agter kanaalkodering is om addisionele inligting met 'n boodskap te stuur (pariteit), sodat die boodskap meer bestand sal wees teen foute wat gemaak word as gevolg van geraas. Daar is verskillende tipes kodes wat gebruik kan word om die ekstra inligting te genereer, naamlik blok-, konvolusie- en aangeskakelde kodes.

'n Spesiale subklas van kodes wat bestaan uit die kodes genoem in die vorige paragraaf, is yl grafiekkodes. Die struktuur van yl grafiekkodes kan grafies voorgestel word deur 'n faktoriserings-grafiek (met min verbindingslyne tussen elemente binne die grafiek). Die kodes wat deel is van hierdie subklas sluit in Lae-Digtheid Pariteit-Kodes (LDPK), Herhaal-Akkumuleer (HA) kodes, Turbo kodes asook fonteinkodes. Yl grafiekkodes kan gedekodeer word deur die gebruik van sekerheids-propagasie (iteratiewe algoritme waar waarskynlikhede na die nodes van die grafiek aangestuur word).

Hierdie verhandeling fokus op die raserige dekodering van fonteinkodes deur sekerheids-propagasie te gebruik. Fonteinkodes was oorspronklik ontwerp vir uitwis-kanale. Omdat enige yl faktoriserings-grafiek gedekodeer kan word met behulp van sekerheids-propagasie, was dit eenvoudig om raserige dekodering toe te pas op fontein faktoriserings-grafieke. Drie hoof fonteinkodes, naamlik Warrelwind, Luby Transform (LT) en Toege-

draaide kodes is ondersoek in die verhandeling. Die volgende gevolgtrekkings kan gemaak word:

1. Die Warrelwind grafiek struktuur kan nie gebruik word vir raserige dekodeering nie weens die feit dat die struktuur van Warrelwind kodes die eerste laag pariteit beskerm in plaas van die oorspronklike boodskaplaag ('n Warrelwind grafiek bestaan uit meer as een laag).

2. Die feit dat raserige dekodering van sistematiese LT kodes moontlik is, is bevestig.

3. 'n Sistematies Toegedraaide kode is ontwerp en ook gedekodeer (raserig). Die simulasie resultate wys dat die Toegedraaide kode se grafiek beter resultate kan lewer as die grafieke van die kodes waaruit die Toegedraaide grafiek bestaan ('n Toegedraaide kode is saamgestel uit meer as een kode).

'n Sistematiese LT kode is ook gebruik om die inkrementele-pariteit konvolusie skema van Verbeterde Data Tempo GSM Evolusie (VDGE) te vervang. Die simulasie resultate wys dat 'n fontein inkrementele-pariteit skema beter vaar as 'n konvolusie benadering as die blok lengtes lank genoeg is. Die resultate wys ook dat die fontein benadering beter vaar as die konvolusie benadering na die tweede trasnmissie ontvang is. Alhoewel VDGE 'n ouer tegnologie is, bly dit 'n goeie platform om inkrementele-pariteit skemas op te toets, omdat VDGE een van die eerste platforms is waarop inkrementele-pariteit gebruik is.

Sleutelwoorde: artikulasie-beginsel, sekerheids-propagasie, Verbeterde Data Tempo GSM Evolusie(VDGE), faktoriserings-grafiek, fontein kode, inkrementele-pariteit, Luby-Transform kodes, lae-digtheid pariteit-kodes(LDPK), raserige dekodering, Toegedraaide kodes, yl grafieke, Warrelwind kodes.

Ons weet dat God alles ten goede laat meewerk vir die wat Hom liefhet, die wat volgens sy besluit geroep is.

Romeine 8:28, Bybel Nuwe Vertaling, 1983

This dissertation is dedicated to my parents.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Prof J.C. Olivier, Jacobus Vlok and Leon Staphorst for their technical contributions. Secondly I would like to thank Pieter Grobler, Cornelis Vlok, Franz Erasmus and Paula Wessels for their support. I would also like to thank Tiaan du Preez and Riaan Visser for the many coffees and lunches respectively. Lastly I would like to thank God for helping me with this year.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AMPS	Advanced Mobile Phone System
AWGN	Additive White Gaussian Noise
BCH	Bose, Ray-Chaudhuri, Hocquenchem
BCS	Block Check Sequence
BER	Bit Error Rate
BLER	Block Error Rate
BP	Belief Propagation
CC	Convolutional Code
CDMA	Code Division Multiple Access
E	Extension Bit
ECC	Error Correcting Coding
EDGE	Enhanced Data Rates for GSM Evolution
FBI	Final Block Indicator
FDD	Frequency Division Duplexing
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
FM	Frequency Modulation
GFB	Generalized Forward/Backward
GMSK	Gaussian Minimum-Shift Keying
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication

HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
IBP	Iterative Belief Propagation
IP	Internet Protocol
IR	Incremental Redundancy
IS-95	Interim Standard 95
LA	Link Adaptation
LBC	Linear Block Code
LDPC	Low-Density Parity-Check
LLR	Log Likelihood Ratio
LOS	Line-Of-Sight
LT	Luby Transform
MAC	Medium Access Control
MAP	Maximum A-Posteriori
MC-CDMA	Multi Carrier-CDMA
MCS	Modulation and Coding Scheme
NLOS	Non-Line-Of-Sight
NMT	Nordic Mobile Telephony
OFDM	Orthogonal Frequency Division Multiplexing
PSK	Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
RA	Repeat-Accumulate
RLC	Radio Link Control
RS	Reed-Solomon
RSC	Recursive Systematic Convolutional

SATNAC	Southern African Telecommunication Networks and Applications Conference
SMS	Short Message Service
SNR	Signal-to-Noise-Ratio
STC	Space Time Codes
TACS	Total Access Communication System
TB	Tail Bits
TC	Turbo Code
TCM	Trellis Coded Modulation
TDMA	Time Division Multiple Access
TX	Transmission
USF	Uplink State Flag
VA	Viterbi Algorithm
WCDMA	Wideband Code Division Multiple Access

NOMENCLATURE

α_n	Instantaneous average fading amplitude corresponding to the n^{th} transmitted bit
B	A bipartite graph
B_D	Maximum Doppler spread
$\mathcal{C}(B)$	The code associated with B
$\mathcal{C}_{LT}$	A weakened LT code
ϵ_b or E_b	Energy per uncoded bit
$1+\epsilon$	Decoding inefficiency
f_b	Baseband data bit rate
f_c	Carrier frequency in Hertz
f_s	Sampling frequency
$f_{w\to v}(X_{w\to v})$	Product of local functions in $F_{v\to w}$
F	Factor graph F
$F_{v\to w}$	Sub graph of F containing node w but not node v
$\mathbf{G}$	Generator matrix of channel code
$G(D)$	Generator polynomial of CC
$\tilde{g}$	Marginalized function of g
$\hat{g}$	Normalized marginalized function of g
$\mathbf{H}$	Parity check matrix of channel code
K	Rician factor measured in dB
$\kappa(d)$	Improved robust soliton distribution

$\lambda(x)$	Likelihood ratio of x
λ_i	Fraction of edges on the left of degree i
$\Lambda(x)$	LLR of x
$\Lambda(m_n)$	Channel LLR associated with bit n
$\Lambda(m_o)$	Output LLR
$\Lambda(m_i)$	Input LLR
$\mu_{x\to f}(x)$	Message passed from node x to node f
$\mu(d)$	Robust soliton distribution
$m(t_i)$	BPSK modulated symbol
N_0	Single-sided noise PSD
$n(x)$	Neighbours of node x
$\mathbf{P}(\Lambda(x), n, r)$	Ensemble of graphs
$\Pr(x)$	Probability of variable x
$\Pr(\mathbf{t}\|\mathbf{r})$	Posterior probability of $\mathbf{t}$
$\Pr(t_n\|\mathbf{r})$	Posterior probability of n^{th} received bit
$\Pr(r_i\|t_i)$	Probability of the i^{th} code bit after transmission through channel
$P_e(bit)$	Probability of bit error
ρ_i	Fraction of edges on the right of degree i
$\rho(d)$	Soliton distribution
$q^t_{m,n}$	Probability that the transmitted bit is t, given the information obtained from all check nodes, except node m
$q^L_{m,n}$	LLR value of $q^t_{m,n}$
q_n	The n^{th} code node in factor graph
$\mathbf{r}$	Channel-corrupted received code word
$r^t_{m,n}$	Probability that check node m is satisfied when n has value t, (other bits independent)
$r^L_{m,n}$	LLR value of $r^t_{m,n}$
r_m	The m^{th} check node in factor graph
r_n	n^{th} received bit of $\mathbf{r}$

σ^2	Variance of noise variable η
$\mathbf{t}$	Transmitted code word
t_n	n^{th} transmitted bit of $\mathbf{t}$
X_A	Variable index set A
$X_{v\to w}$	Variable set containing $x_{\{v,w\}}$ and the variables of $F_{v\to w}$
$x_{\{v,w\}}$	Variable associated with edge $\{v, w\}$
$\mathbf{x}$	Information message word before transmission
$\mathbf{y}$	Received and decoded message word
$\Omega(x)$	Right degree distribution of weakened LT code

CHAPTER 1

INTRODUCTION

1.1 CHAPTER INTRODUCTION

The need to communicate is as old as mankind itself. Communication has applications in peace keeping, is vital to world economic stability, and is a key to sustained economic growth. Modern communication tools like e-mail and mobile phones are developments towards instantaneous global communication and interaction.

The development of mobile telephony happened in phases, each phase is known as a Generation. Mobile development is currently in the third generation (3G), which enables the use of multimedia applications on our mobile phones. Simple communication tools are simply taken for granted, but are in fact complicated devices, that include state of the art probabilistic inference calculations (such as the mobile telephone).

This thesis aims to make wireless communication more reliable. This chapter begins by explaining how channel coding can be used to improve communication, followed by the general history of channel coding and mobile telephony. In short, this chapter summarizes the theory and application of error correcting codes in communication systems. The chapter ends with the contribution made by this thesis as well as a layout of the dissertation.

1.2 CHANNEL CODING

One of the most important elements of a modern communication system is error correction coding. In error correcting coding redundant information (parity) is added to a message to make the message more resilient against noise induced errors during transmission. The field of error correcting coding dates back to a paper published by Shannon [1]. Shannon predicted that reliable communication is achievable, if redundancy is added to the message across a memoryless channel, as long as the communication rate does not exceed the capacity of the channel. Shannon however did not propose any

coding scheme to achieve this. Since 1948 effort has been put into the ongoing search for codes able to achieve these theoretical bounds. The codes developed over the years can be classified into mainly three types, namely block codes, convolutional codes and concatenated codes.

The main block codes developed during the past 60 years include Hamming [2], Bose-Chaudhuri-Hocqhuenghem (BCH) [3, 4, 5], Reed-Solomon (RS) [6] and Low Density Parity Check (LDPC) codes [7, 8, 9]. Some of the most important developments in decoding algorithms of block codes include the Berlekamp-Massey hard decision decoding algorithm of RS codes [10], the construction of block code trellis diagrams to enable soft decision Viterbi decoding [11] of linear block codes [12] and the development of BP (belief propagation) decoding of sparse graph (LDPC) codes [13, 8, 14].

CC (Convolutional codes) were first introduced by Elias in 1955 [15]. Two important decoding algorithms were developed to decode these codes namely Viterbi trellis decoding [11] and the MAP (Maximum A-Posteriori) algorithm [16].

The first concatenated codes were developed by Forney [17] and much later Turbo codes (very good concatenated codes) were developed by Berrou *et al.* [18].

Two other advances in coding theory is worth mentioning as well. The development of TCM (Trellis Coded Modulation) (combining modulation and coding) [19] and STC (Space Time Codes) (using multiple antennas) [20].

The target set by Shannon has now almost been reached by using Turbo and LDPC codes. These codes can perform arbitrarily close to the Shannon bound when used on the AWGN channel and the binary erasure channel if large block (code) sizes are used. In MacKay [14] Turbo and LDPC codes are discussed in terms of their factor graph representations, and are therefore also part of a family of codes known as sparse graph codes, that also includes Repeat Accumulate (RA) and fountain codes. These codes are called sparse due to the low density factor graphs that represent them. BP is the main algorithm used to decode these codes (BP was originally developed by Pearl and Gallager [13, 7, 8]).

1.3 MOBILE TELEPHONY

The development of cellular networks can be classified in different classes (grouped together by technology), each class is known as a generation:

0G : Half-duplex radio phone technology which operated without a cellular network

(1946).

1G : Voice communication through analog FM transmission. Frequency Division Multiple Access/Frequency Division Duplexing (FDMA/FDD) was used to accommodate multiple users and channels (1980s). Advanced Mobile Phone System (AMPS), Total Access Communication System (TACS) and Nordic Mobile Telephony (NMT) were examples of 1G networks.

2G : Purely digital networks using digital modulation and Time Division Multiple Access (TDMA) or Code Division Multiple Access (CDMA) to support voice, text messaging (SMS) and circuit-switched data communication (1990s). The two best examples of 2G networks are Global System for Mobile communication (GSM) and Interim Standard 95 (IS-95). Forward error correction (FEC) and encryption are also supported by this generation. Other special services like caller identification are also available to users. Some additional standards were developed to increase the data speeds of 2G networks, including 2.5G General Packet Radio Service (GPRS) and 2.75G Enhanced Data Rates for GSM Evolution (EDGE).

3G : Provides data rates up to 2 Mbps using wideband modulation techniques with increased user capacity (2000's). Services like Internet, e-mail, multimedia streaming, video telephony and instant messaging are all services available to 3G devices (this includes mobile handsets and computers). The two main 3G standards are Wideband Code Division Multiple Access (WCDMA) and Multi Carrier CDMA (MC-CDMA). Improvements on the 3G standard include 3.5G High Speed Downlink Packet Access (HSDPA) and 3.75G High Speed Uplink Packet Access (HSUPA).

4G : Higher data rates (10Mbps to 1 Gbps) will be supported through the use of an all-IP futuristic network. The multiple access technology used by this generation will in all likelihood be OFDM (Orthogonal Frequency Division Multiplexing). The application layer services like mobile video will be provided. There will be seamless integration between all generation networks.

1.4 MOTIVATION AND CONTRIBUTION

The research reported in this dissertation builds on the results found in [21, 22]. In [21] different codes were decoded using soft decision Viterbi decoding. In [22] these results were extended to the BP decoding of sparse graph codes like LDPC, Repeat Accumulate and Turbo codes. These codes are discussed in the book by MacKay [14]. This dissertation investigates the BP decoding of the final class of sparse graph codes from [14], which is fountain codes, and also builds on the idea of systematic fountain codes [23]. Fountain codes are also used as an alternative Incremental Redundancy (IR) scheme [24, 25], and tested on the EDGE platform. Although EDGE is an older technology (2G) it still remains a good platform to test the performance of different IR schemes. The main contributions of this thesis are summarized below:

1. **BP decoding of simple block and convolutional codes**: A Hamming (7,4,3) and a RSC(2,1,2) (Recursive Systematic Convolutional) code are decoded using BP decoding. The main aim of this is to show how different codes can be decoded using BP and factor graphs. This supports the idea of movement away from trellis decoding towards factor graphs. It will become clear that many different codes can be combined and then be decoded using a single algorithm and graph structure. For instance, block and convolutional codes can be combined in one graph and then be decoded using BP.

2. **Noisy decoding of the systematic Tornado graph structure using BP**: The factor graph of a systematic Tornado code has more than one layer. The BP algorithm was applied to this unique factor graph structure, since BP can be applied to any factor graph as shown in Chapter 2. The Tornado (80,40) code from Section 3.3.4 was used for simulation purposes. From the simulation results it was concluded that the graph structure of a Tornado code does not lower the BER when compared to the BER performance of its individual layers (the Tornado(80,40) code actually performs worse). The first parity layer, however, has a very low BER. This shows that the Tornado code actually protects its first parity layer rather than the original message bits. This makes the graph structure of a Tornado code unsuitable for noisy decoding purposes [26]. The results from Section 5.2.7 replaces the simulation results obtained in Section IV of [26].

3. **Verification of noisy decoding of systematic LT codes** : Systematic LT codes are introduced in [23]; a decoding algorithm for these codes are also given

in the same publication. This algorithm was derived using the same principles discussed in Chapter 2, approximately the same time as this dissertation. The simulations (in this dissertation) were performed on a systematic LT (80,40) code that uses an improved robust soliton distribution with parameters (40,0.05,0.05) (see Section 5.2.7 for more details). The simulation results confirm the results obtained in [23], showing that the decoding algorithm works. A smaller code is used in this dissertation, since the decoding algorithm is tested and not the BER performance of systematic LT codes.

4. **Noisy decoding of the systematic Raptor graph structure**: Systematic Raptor codes are introduced in this dissertation. The Raptor code uses a pre-code as well as a weakened LT code in the construction of its factor graph. But since the two graphs (when combined) form one new graph, a Raptor code can be decoded using BP as discussed in Chapter 2 and 3. This is demonstrated by using a low complexity example; a Raptor (80,20) code consisting of a (40,20) regular (3,6) LDPC code as pre-code followed by a LT (80,40) code (see Sections 3.5 and 5.2.7 for greater detail). The simulation results, show that the new graph structure does perform better under flat fading conditions (no line of sight), but performs worse than its constituent codes under AWGN channel conditions.

5. **Introducing fountain codes as an IR scheme**: Currently punctured convolutional codes are used as incremental redundancy scheme in EDGE [24, 25]. This dissertation studies the performance obtained when fountain codes replace the current scheme. The fountain code used for this scheme can be found in Section 5.3. The simulation results show that the convolutional approach outperforms the fountain code for the first transmission. The fountain code however outperforms the convolutional code from the second transmission onwards. In summary, the fountain code has potential as an IR scheme, and depends on the specific environment it is used in. If large frame lengths are used it is the most likely approach to use [26].

1.5 LAYOUT OF DISSERTATION

The main aim of this thesis is to test a fountain incremental redundancy scheme on the EDGE platform. The secondary aim is to test the different fountain factor graphs

on a standard QPSK transmission system. The results of the two aims can be obtained in Chapter 5 and 6.
Chapter 5 and 6 are preceded by theoretical chapters that enabled the creation of the simulations responsible for the results found in these chapters. In the first case BP is explained in detail in Chapter 2; this is required by Chapter 3 to enable the noisy decoding of fountain codes. The EDGE platform is introduced in Chapter 4. Combining Chapters 3 and 4 makes it possible to implement a fountain IR scheme. The basic content of each chapter is summarized below:

Chapter 2: The chapter introduces factor graphs followed by the sum-product algorithm. The sum-product algorithm is shown to calculate the marginal function by using the articulation principle. Finally belief propagation is derived from the sum-product algorithm by substituting the factor graph with a Tanner graph. The chapter ends with an example of how belief propagation can be applied to decode a simple Hamming (7,4,3) block code.

Chapter 3: The chapter begins by giving a general overview of fountain code development over the past decade. After this section, each of the three fountain codes (Tornado, LT, Raptor) are explained under the following general headings: encoding, decoding on erasure channel, design, noisy decoding and factor graph representation.

Chapter 4: A global overview of EDGE is given (including data rates obtainable) followed by the structure of the TX (Transmission) blocks used by the standard. The current encoder used by EDGE follows the TX block section. The chapter ends with the concept of IR, how the current encoder implements IR and how fountain codes could replace the current scheme.

Chapter 5: The chapter is divided into two main sections. In the first part of the chapter the three fountain codes are investigated on a standard flat fading channel. Some simple non-fountain codes are also investigated in this section. The second section investigates the use of fountain codes on the EDGE platform to implement IR.

Chapter 6: The chapter contains all the conclusions drawn from the simulations results presented in Chapter 5.

CHAPTER 2

SPARSE GRAPH CODES AND BELIEF PROPAGATION

2.1 CHAPTER INTRODUCTION

The chapter introduces factor graphs followed by the sum-product algorithm. The sum-product algorithm is shown to calculate the marginal function by using the articulation principle. Next belief propagation is derived from the sum-product algorithm by substituting the factor graph with a Tanner graph. An example is given to show how to construct factor graphs (Tanner) for simple block codes. The chapter ends with an example of how belief propagation can be applied to decode a simple Hamming (7,4,3) block code (the second last section discusses classic LDPC BP).

2.2 FACTOR GRAPH DEFINITION

A factor graph is a bipartite graph that shows how a function can be factored into the product of local functions. As an example $g(x, y) = x \cdot y + x$ can be factored in to $g(x, y) = f_1(x) \cdot f_2(y)$ where $f_1(x) = x$ and $f_2(y) = y + 1$. The corresponding factor graph is depicted in Figure 2.1.

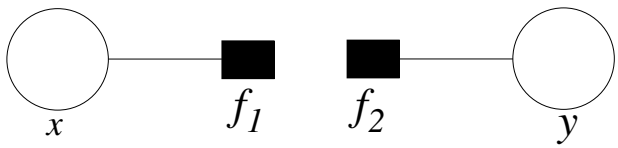

Figure 2.1: A simple factor graph representing $g(x, y) = x \cdot y + x$

A bipartite graph is a graph that can only consist of two types of nodes. In the case of a factor graph these two types are the variable (depicted as circles in Figure 2.1) and function nodes (depicted as squares in Figure 2.1) respectively. A variable node can only be connected to a function node if that variable is an argument of that specific

function. Nodes of the same type are never connected with each other and the edges are not directed.

Assume the function g can be factored as [27, 28]

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) \tag{2.1}$$

its factor graph is shown in Figure 2.2

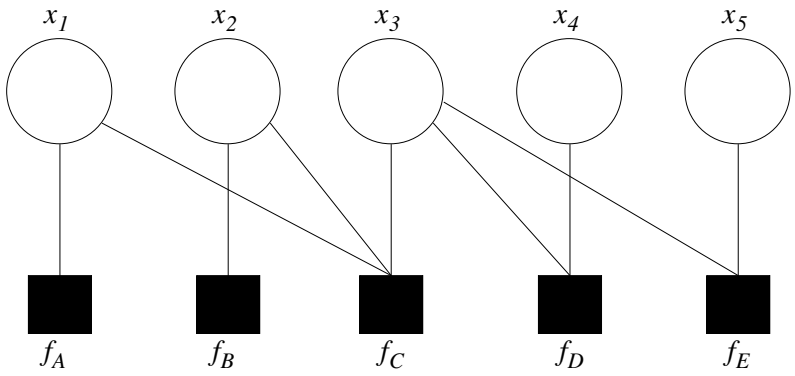

Figure 2.2: The factor graph of $g(x_1, x_2, x_3, x_4, x_5)$

Suppose one is interested in calculating the marginal function [14] $\tilde{g}$ in terms of the variable x_1, which in this case is equal to

$$\begin{aligned}\tilde{g}(x_1) &= \sum_{x_2, x_3, x_4, x_5} g(x_1, x_2, x_3, x_4, x_5) \\ &= \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5)\end{aligned} \tag{2.2}$$

where $\sum_{x_i}$ denotes a summation over all the values the variable x_i can take on. Using equation (2.2) it is easy to define the general marginal function $\tilde{g}(x_j)$ as being the function g summed over all possible combinations of values that the arguments of g can have other than the argument x_j. Since g is factored (as shown in equation (2.1)) and the local functions are not dependent on all $x_k, k \in \{2, \cdots, 5\}$ it is possible to rewrite equation (2.2) as

$$\tilde{g}(x_1) = f_A(x_1) \sum_{x_2} f_B(x_2) \sum_{x_3} f_C(x_1, x_2, x_3) \sum_{x_4} f_D(x_3, x_4) \sum_{x_5} f_E(x_3, x_5) \tag{2.3}$$

Now simplify the calculation of equation (2.3) as

$$f_I(x_3) = \sum_{x_5} f_E(x_3, x_5) \qquad \text{and} \tag{2.4}$$

$$f_{II}(x_3) = \sum_{x_4} f_D(x_3, x_4) \tag{2.5}$$

Assuming $f_E(x_3, x_5)$ is defined as

$$f_E(x_3, x_5) = \begin{cases} 1 & \text{if } (x_3, x_5) = (0,0) \text{ or } (x_3, x_5) = (1,1) \\ 0 & \text{if } (x_3, x_5) = (1,0) \text{ or } (x_3, x_5) = (0,1) \end{cases} \tag{2.6}$$

then it is easy to see that $f_I(x_3)$ is equal to $f_E(x_3, 0) + f_E(x_3, 1)$. The value of $f_{II}(x_3)$ can be calculated similarly. Then using $f_{III}(x_3) = f_I(x_3) \cdot f_{II}(x_3)$ one can calculate

$$f_{IV}(x_1, x_2) = \sum_{x_3} f_C(x_1, x_2, x_3) \cdot f_{III}(x_3) \qquad \text{and} \tag{2.7}$$

$$f_V(x_1) = \sum_{x_2} f_B(x_2) \cdot f_{IV}(x_1, x_2) \tag{2.8}$$

Finally $\tilde{g}$ can be calculated with $\tilde{g}(x_1) = f_A(x_1) \cdot f_V(x_1)$, assuming all local functions are defined as in the case of f_E.
From the above it is clear that due to the structure of the factor graph some calculations could be performed locally at a function node without any other knowledge of the factor graph. The results of the locally calculated functions can be passed on to other variable nodes to help the remaining function nodes higher up in the hierarchy perform their calculations. This message passing approach forms the basis of the sum-product algorithm.

2.3 SUM-PRODUCT ALGORITHM

If the sum-product algorithm is applied to the factor graph of a function $g(x_1, x_2, \ldots, x_N)$ it calculates the marginal function [27, 28]

$$\tilde{g}(x_i) = \sum_{\substack{x_j \\ \forall j \in \{1,2,\cdots,N\} \setminus \{i\}}} g(x_1, x_2, \cdots, x_i, \cdots, x_N) \tag{2.9}$$

where $i \in \{1, 2, \cdots, N\}$. To make the notation more compact, we define

$$f(x_1, x_2, \cdots, x_L) \downarrow x_i = \sum_{\substack{x_j \\ \forall j \in \{1,2,\cdots,L\} \setminus \{i\}}} f(x_1, x_2, \cdots, x_L) \tag{2.10}$$

where $i \in \{1, 2, \cdots, L\}$ and $\downarrow$ is known as the *summary operator*. This can also be written in the following notation

$$f(x_1, x_2, \cdots, x_L) \downarrow x_i = f(X_A) \downarrow x_i \tag{2.11}$$

where i is the same as before and $A \equiv \{1, 2, \cdots, L\}$ making $X_A \equiv \{x_1, x_2, \cdots, x_L\}$. In words the notation X_A represents the set of all variables with subscripts contained in *variable index set* A. Take note that the marginal function does not have to be calculated only in terms of x_i but can also be calculated in terms of a set of variables $X_B \equiv \{x_1, \cdots\} : f(X_A) \downarrow X_B$ (sum over all variables except over those contained in variable set X_B).

The sum-product algorithm can be best understood by interpreting each node of a factor graph as a processor [27, 28]. The factor graph consists of two different types of processors, which are the "variable" and the "function" processors respectively. Two processors can only communicate if they are connected via an "edge" (wire/connections). Each of the two different types of processors uses a different formula to process the information it receives on its connections. So the basic idea is that each variable processor begins by sending the message 1 to their adjacent function processors. The function processors use a unique update rule for each received message (by using all the other received messages except the one being updated). After each function processor has finished updating its messages it sends them back to the variable processor. The variable processor performs the same action using a different update rule. It is very important to keep in mind that the messages being passed between processors are entire functions. This passing of messages between processors continue until no new message (do not change) can be generated. Now the update rules of each processor can be given. The update rules used by the two different processors can be merged into the following general rule if the internal function of a variable processor is assumed to be equal to the unity function.

Rule 1 (The Sum-Product Update Rule) *The message sent from a node v on an edge e is the product of the local function at v (in the case of a variable node this is equal to the unit function) with all messages received at v on edges other than e, summed over all variables other than the variables associated with e (except one which is either the variable of the considered node or the variable of the node where the messages are sent to) [27, 28].*

This rule can be represented mathematically by two update equations (see also Figure 2.3). These update equations for the variable and function nodes are respectively given as [27, 28]:

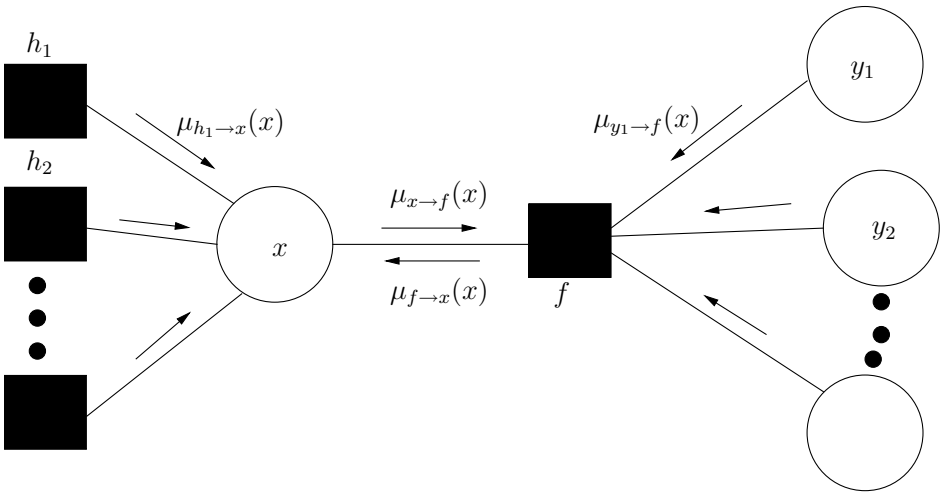

Figure 2.3: A fragment of a factor graph showing the update rules for the sum-product algorithm

$$\mu_{x \to f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \to x}(x) \tag{2.12}$$

and

$$\mu_{f \to x}(x) = \left(f(x, y_1, y_2, \cdots) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \to f}(y) \right) \downarrow x \tag{2.13}$$

where $\mu_{v \to w}$ represents the message sent from node v to node w and $n(v)$ denotes all the neighbors of node v. It is important to mention here that the messages $\mu_{h \to x}(x)$ depend only on x and for this reason the $\downarrow$ operator is omitted in the calculation of $\mu_{x \to f}(x)$. In the special case when a variable node has only two neighbors (see also Figures 2.4 and 2.5)

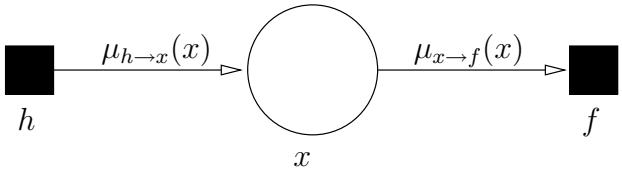

Figure 2.4: A variable node with only two edges

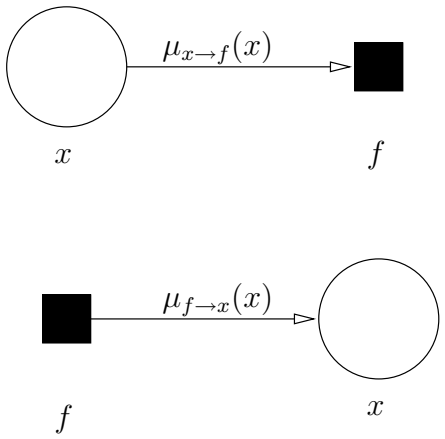

Figure 2.5: Leaf nodes

the message is simply passed on as $\mu_{x\to f}(x) = \mu_{h\to x}(x)$. Leaf nodes send the following messages $\mu_{x\to f}(x) = 1$ or $\mu_{f\to x}(x) = f(x)$.

2.4 BASIC CHARACTERISTICS OF THE SUMMARY OPERATOR

The following characteristics of the summary operator are important. In the first case it is easy to see that [27]

$$g(X_C) \downarrow X_B = g(X_C) \downarrow X_{B\cap C} \tag{2.14}$$

since X_C can only be summed over elements contained in it. The above can easily be understood by looking at the following example,

$$g(x_1, x_2, x_3) \downarrow \{x_2, x_3, x_4\} = g(x_1, x_2, x_3) \downarrow \{x_2, x_3\} \tag{2.15}$$

since x_4 is not part of g.

2.4.1 AXIOMS

The operator $\downarrow$ can only be called a summary operator if, for all R-valued functions f and g and for all variable index sets B, C and D, the following axioms are satisfied [27]:

1. $B \subset C \Rightarrow f(X_D) \downarrow X_B = [f(X_D) \downarrow X_C] \downarrow X_B$

2. $B \subset C \Rightarrow [f(X_B) \cdot g(X_D)] \downarrow X_C = f(X_B) \cdot [g(X_D) \downarrow X_C]$

3. $B \cap C = \emptyset \Rightarrow [f(X_B) \cdot g(X_C)] \downarrow X_D = [f(X_B) \downarrow X_D] \cdot [g(X_C) \downarrow X_D]$

Axiom 1 implies that a marginal can be obtained by evaluating a sequence of ever more restrictive summaries, and that the evaluation of these summaries can be done in any order. In the example,

$$\begin{aligned} g(x_1, x_2, x_3) \downarrow x_1 &= (g(x_1, x_2, x_3) \downarrow \{x_1, x_2\}) \downarrow x_1 \\ &= (g(x_1, x_2, x_3) \downarrow \{x_1, x_3\}) \downarrow x_1 \end{aligned} \tag{2.16}$$

x_3 can be summarized followed by x_2 or vice versa, it doesn't matter since the same result is obtained. Axiom 2 shows that when $B \subset C$ then $f(X_B)$ is a constant. This is illustrated by the example below,

$$(f(x_1) \cdot g(x_1, x_2, x_3)) \downarrow \{x_1, x_2\} = f(x_1) \cdot (g(x_1, x_2, x_3)) \downarrow \{x_1, x_2\} \tag{2.17}$$

Axiom 3 implies that if two functions have no arguments in common, the marginal of the product is the product of the marginals. The last axiom is also made clear with an example

$$\begin{aligned} [f(x_1, x_2) \cdot g(x_3, x_4)] \downarrow \{x_1, x_3\} &= \sum_{x_2} \sum_{x_4} f(x_1, x_2) \cdot g(x_3, x_4) \\ &= \sum_{x_2} f(x_1, x_2) \cdot \sum_{x_4} g(x_3, x_4) \\ &= (f(x_1, x_2) \downarrow \{x_1, x_3\}) \cdot \\ &\quad (g(x_3, x_4) \downarrow \{x_1, x_3\}) \end{aligned} \tag{2.18}$$

The following lemma is easily derived from the above axioms (and can be extended to products consisting of more than two functions) as shown in [27],

Lemma 1 *If X_A and X_B are disjoint and do not contain x, then*

$$[f(x, X_A)g(x, X_B)] \downarrow x = [f(x, X_A) \downarrow x] \cdot [g(x, X_B) \downarrow x] \tag{2.19}$$

2.5 MESSAGE PASSING SCHEDULES

This section closely follows the notation of [27, 28]. Until now only the way messages are updated has been defined, the order (schedule) in which this should occur has not. The assumption is made that one message is passed every clock cycle (assuming the message passing is synchronized by a discrete time clock).

2.5.1 "NOWHERE IDLE" SCHEDULES

A schedule in which at least one message (only pending messages may be updated) is updated in every clock cycle (prevents schedule from becoming "lazy") is known as a "*nowhere idle*" schedule. A node v has a message pending at en edge e only if node v can send a different message than the one sent previously on e. Generally whenever a new message arrives it will cause pending messages at each edge of that node. It follows naturally that if a "nowhere idle" message passing schedule is applied to a factor graph that has no loops and is a finite tree, the sum-product algorithm will terminate (no pending messages will remain due to the absorption of messages by leaf nodes). The following theorem shows that the final result of the sum-product algorithm (using a "nowhere idle" schedule) is $\tilde{g}$ as defined in equation (2.9) [27, 28].

Theorem 1 *Let F be a factor graph for a function $g(x_1, x_2, \cdots, x_N)$, and suppose F is a finite tree. Let x_i be any variable of F, i.e. $i \in \{1, 2, \cdots\}$. Every nowhere idle schedule for the sum-product algorithm will eventually result in a state with no pending messages. Then it is true that*

$$g(x_1, x_2, \cdots, x_N) \downarrow x_i = \prod_{f \in n(x_i)} \mu_{f \to x_i}(x_i) \tag{2.20}$$

where $n(x_i)$ denotes the set of neighbors of x_i, i.e. the set of function nodes that are connected by an edge to x_i.

The above theorem is shown to be true in Section 2.6 by using the articulation principle. This is done by first showing the validity of the above theorem in the case of a special subclass of "nowhere idle" schedules known as generalized forward/backward (GFB) schedules and then generalizing the result for all "nowhere idle" schedules.

2.5.2 GFB SCHEDULES

After a GFB schedule finishes, exactly one message was passed (updated) in each direction on every edge of a factor graph. To ensure the above the restrictions of the "nowhere idle" schedule needs to be tightened. For a GFB a node v may only update a message on edge e if it has received messages on all edges except e. Using the GFB only leaf nodes of a graph F can send messages in the beginning of the algorithm (only nodes with pending messages). See Figure 2.5 for the values of these messages. After the leaf nodes have sent their messages they are deleted from the tree (including their edges). Some of the nodes that received messages from the original leaf nodes will now

be leaf nodes and therefore be able to send messages themselves. Clearly this process can continue until a single node remains. Since at each stage the graph is a tree of more than one node, there are always at least two leaf nodes in a position to send a message, so this process will not stall. When only a single node v remains a message will have been passed in one direction over each edge in the original graph. Now the original graph gets reconstructed adding any neighbor of any retrieved node (starting with v) randomly (sending a message). After the original graph F is reconstructed a message will have been passed in both directions on every edge of F. At this stage the algorithm terminates and $\tilde{g}$ can be calculated using equation (2.20) for any variable in F. This is proven in Section 2.6.

2.6 ARTICULATION PRINCIPLE

Assume that F is a finite tree. If any edge $\{v, w\}$ of F is cut two sub graphs $F_{w\to v}$ (containing v, but not w) and $F_{v\to w}$ (containing w, but not v) are obtained and the variable associated with edge $\{v, w\}$ is denoted as $x_{\{v,w\}}$

The articulation principle [27] refers to the idea that the message $\mu_{v\to w}$ sent from v to w must articulate (encapsulate or summarize) for the variables attached to $F_{v\to w}$ the product of the local functions in $F_{w\to v}$. This is expressed as

$$\mu_{v\to w} = f_{w\to v}(X_{w\to v}) \downarrow X_{v\to w} \tag{2.21}$$

where $X_{x\to y}$ refers to the variable set containing $x_{\{x,y\}}$ and the variables of $F_{x\to y}$ and $f_{x\to y}(X_{x\to y})$ refers to the product of the local functions in $F_{x\to y}$ (see also Figure 2.6).

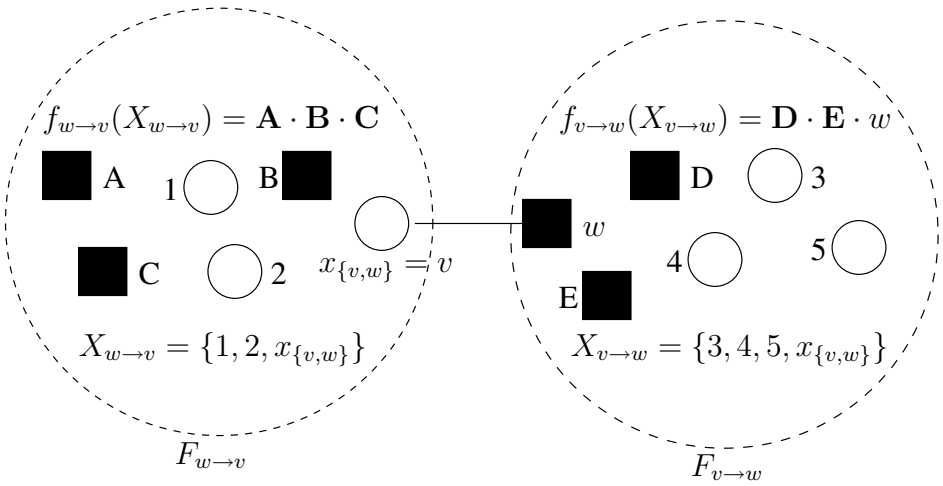

Figure 2.6: Articulation Principle notation represented graphically

When F is a tree, it can be observed that only $x_{\{v,w\}}$ is present in variable sets $X_{v\to w}$ and $X_{w\to v}$ formed when F is cut at $x_{\{v,w\}}$ ($X_{v\to w} \cap X_{w\to v} = x_{\{v,w\}}$). Now equation (2.21) can be written as [27]

$$\mu_{v\to w} = f_{w\to v}(X_{w\to v}) \downarrow x_{\{v,w\}} \tag{2.22}$$

due to equation (2.14).
The articulation principle has now been defined but it must now be proven to hold in a finite tree graph F that employs the message-passing algorithm (using a GFB schedule) for all messages sent in F. The messages sent by leaf nodes (see Figure 2.5 for values) clearly satisfies equation (2.22). If it can be shown that equation (2.22) is true for all outgoing messages whenever it is true for incoming messages, then by induction it will be true for all messages. Assuming $\{v, w\}$ is the outgoing edge of v and suppose that the equivalent of equation (2.22) is satisfied for all incoming messages at v, i.e.,

$$(\forall u \in n(v)\backslash w)\mu_{u\to v} = f_{v\to u}(X_{v\to u}) \downarrow x_{\{u,v\}} \tag{2.23}$$

If v is a variable node, $x_{\{u,v\}} = v$. For distinct $u_1, u_2 \in n(v), X_{v\to u_1} \cap X_{v\to u_2} = \{v\}$; hence, from Lemma 1 , equation (2.12) and equation (2.22), the following is obtained [27]

$$\begin{aligned}\mu_{v\to w}(v) &= \prod_{u\in n(v)\backslash w} (f_{v\to u}(X_{v\to u}) \downarrow v) \\ &= \left(1 \cdot \prod_{u\in n(v)\backslash w} f_{v\to u}(X_{v\to u})\right) \downarrow v \\ &= f_{w\to v}(X_{w\to v}) \downarrow v \end{aligned} \tag{2.24}$$

as desired.
If v is a local function node, then for each $u \in n(v)\backslash w, x_{\{u,v\}} = u$. Assuming the equivalent of equation (2.22) holds and using $u \in n(v)\backslash w, x_{\{u,v\}} = u$ it becomes clear that the incoming messages of v can be expressed as

$$u_{u\to v} = f_{v\to u}(X_{v\to u}) \downarrow u \tag{2.25}$$

unless u is a leaf node, in which case $u_{u\to v} = 1$. Now applying the axioms defining the

summary operator and Lemma 1, the following is obtained [27]

$$\begin{aligned} f_{w\to v}(X_{w\to v}) \downarrow w &\overset{(a)}{=} \left[v(X_{n(v)}) \prod_{u\in n(v)\backslash\{w\}} (f_{v\to u}(X_{v\to u}) \right] \downarrow w \\ &\overset{(b)}{=} \left[\left(v(X_{n(v)}) \prod f_{v\to u}(X_{v\to u}) \right) \downarrow n(v) \right] \downarrow w \\ &\overset{(c)}{=} \left[v(X_{n(v)}) \cdot \left(\left(\prod f_{v\to u}(X_{v\to u}) \right) \downarrow n(v) \right) \right] \downarrow w \\ &\overset{(d)}{=} \left[v(X_{n(v)}) \cdot \prod (f_{v\to u}(X_{v\to u}) \downarrow n(v)) \right] \downarrow w \\ &\overset{(e)}{=} \left[v(X_{n(v)}) \cdot \prod (f_{v\to u}(X_{v\to u}) \downarrow u) \right] \downarrow w \\ &\overset{(f)}{=} \left[v(X_{n(v)}) \cdot \prod \mu_{u\to v}(u) \right] \downarrow w \\ &\overset{(g)}{=} \mu_{v\to w}(w) \end{aligned} \tag{2.26}$$

In all cases, the product is over $u \in n(v)\backslash\{w\}$ as shown in equality (a), which follows directly from the definition of $f_{w\to v}(X_{w\to v}) \downarrow w$. Equality (b) is a consequence of Axiom 1, since $\{w\} \subset n(v)$, and (c) is due to Axiom 2. Equality (d) is due to Axiom 3, since $X_{v\to u_1}$ and $X_{v\to u_2}$ are disjoint for distinct u_1 and u_2 in $n(v)\backslash\{w\}$. Then (e) follows since $X_{v\to u} \cap n(v) = \{u\}$, while (f) is due to equation (2.25) and (g) follows by definition (2.13).

It has now been shown that the articulation principle holds for a finite tree that employs the sum-product algorithm (GFB schedule). It still needs to be shown that when the articulation principle is valid $\tilde{g}$ can be calculated by equation (2.20). This is shown by [27]

$$\begin{aligned} g(X_S) \downarrow x &= \left(\prod_{f\in n(x)} f_{x\to f}(X_{x\to f}) \right) \downarrow x \\ &= \prod_{f\in n(x)} (f_{x\to f}(X_{x\to f}) \downarrow x) \\ &= \prod_{f\in n(x)} \mu_{f\to x}(x) \end{aligned} \tag{2.27}$$

with S being the subscript index of all variables contained in a graph F (the marginal function is calculated in terms of variable x and f is any neighbor of x). The first equality follows from the fact that $g(X_S)$ can be written as the product of the local functions contained in the disjoint sub trees obtained by removing node x from F. The second equality follows from Lemma 1 (since for distinct f_1 and f_2 in $n(x)$, we have

$X_{x\to f_1} \cap X_{x\to f_1} = \{x\}$ and the third equality follows from the articulation principle (2.22).

In the case of the "nowhere idle" schedule every node will have received messages (that conforms to the articulation principle) sent from every other node (directly or indirectly), at termination (because F is a finite tree) and therefore the message on each edge $\{v, w\}$ of F will be equal to the product of the local functions in $F_{w\to v}$ (from [27]). From here the same result can be obtained for the "nowhere idle" schedule as for the GFB schedule (using similar arguments as in the case of the GFB) [27].

2.7 NORMALIZED MARGINALS

The normalized marginal is defined as

$$\hat{g}(x_i) = \frac{\tilde{g}(x_i)}{\sum_{x_i} \tilde{g}(x_i)} \tag{2.28}$$

If only the normalized marginal is required, then an alternative version of the sum-product algorithm is used [14]. This alternative form is obtained by changing equations (2.12) and (2.20) to

$$\mu_{x\to f}(x) = \alpha_1 \prod_{h\in n(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{2.29}$$

and

$$\hat{g}(x_i) = \alpha_2 \prod_{f\in n(x_i)} \mu_{f\to x_i}(x_i) \tag{2.30}$$

where $\alpha_1 = \frac{1}{\sum_x \mu_{x\to f}(x)}$ and $\alpha_2 = \frac{1}{\sum_{x_i} \prod_{f\in n(x_i)} \mu_{f\to x_i}(x_i)}$.

2.8 BELIEF PROPAGATION AND SPARSE GRAPH CODES

For the factor graph of a sparse graph code the variable nodes are replaced by code nodes and the function nodes are replaced by check nodes. See Figure 2.9 for an example of a factor graph. When the sum-product algorithm is applied on a factor graph of a sparse graph code it is called belief propagation.

In the case of a sparse graph code the messages being passed between nodes are either probability vectors or log-likelihoods (beliefs). More precisely in the case of an LDPC code, the messages being passed from a code node q to a check node r is the probability

that q is either a 1 or a 0, given the observed value of q and all the values received from the neighbors of q except from r during the previous round. The message sent from a check node r to a code node q is the probability that q is equal to either a 1 or a 0 given all the values received by r from its neighbors during the previous round except from q. Due to the unique structure of the factor graph and the type of messages used by belief propagation the update rules simplify dramatically. These update rules are derived in this section directly from the sum-product algorithm.

The link between belief propagation and the sum-product algorithm can easily be shown by remembering that belief propagation is a bit wise decoding algorithm. This means that each decoding iteration of belief propagation improves the probability value of each bit. The final probability that a certain bit t_n (posterior probability) was a 1 or 0 is (obtained through bit wise decoding) equal to the marginalized probability (of being a 1 or a 0) of all the other bits in the received codeword [22]. This can be represented mathematically as

$$\Pr(t_n|\boldsymbol{r}) = \sum_{\{t_{n'}:n'\neq n\}} \Pr(\boldsymbol{t}|\boldsymbol{r}) \tag{2.31}$$

with $\boldsymbol{t}$ representing the sent codeword and $\boldsymbol{r}$ the noisy received codeword. Now equation (2.20) can be used to calculate equation (2.31), since equation (2.20) is a marginalization problem. This is in effect belief propagation decoding as will be shown below. Since the factor graph of a sparse graph code contains loops, equation (2.20) can only calculate equation (2.31) approximately.

The scheduling algorithm used in this dissertation entails updating all the messages of the code nodes first followed by all the messages of all the function nodes. This approach is a "nowhere idle" schedule making equation (2.20) applicable.

2.8.1 PROBABILITIES, LIKELIHOODS AND LOG-LIKELIHOODS

The probability that a random binary variable X is equal to either 1 or 0 is denoted mathematically as $\Pr[X = 1]$ or $\Pr[X = 0]$. The likelihood ratio λ of X is then defined as

$$\lambda(X) = \frac{\Pr[X = 0]}{\Pr[X = 1]} \tag{2.32}$$

The log-likelihood ratio Λ of X will then be equal to

$$\begin{aligned}\Lambda(X) &= \ln \lambda(X) \\ &= \ln \left[\frac{\Pr[X=0]}{\Pr[X=1]}\right]\end{aligned} \tag{2.33}$$

The ratios λ and Λ can also consist of conditional probabilities by substituting $\Pr[X = 1]$ and $\Pr[X = 0]$ with $\Pr[X = 1|Y]$ and $\Pr[X = 0|Y]$ respectively in the above formulas. This will then give $\lambda(X|Y)$ and $\Lambda(X|Y)$.

2.8.2 UPDATE RULE DERIVATION FOR BELIEF PROPAGATION

Assuming the messages consist of probability vectors, let $(\Pr[X_k = 0|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}]$, $\Pr[X_k = 1|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}])$ be the output probability vector of the k^{th} variable (code) node of a factor graph consisting of v variable nodes (assume that the k^{th} variable node has degree n). Where $[\cdot|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}]$ means "given all C_i except C_k" with $i \in \{1, 2, \cdots, n\}$ and $k \in \{1, 2, \cdots, v\}$. Also let $(\Pr[X_k = 0|C_i], \Pr[X_k = 1|C_i])$ be an input probability vector of the same variable node. Using equation (2.29) it is easy to show that the output probability vector of a message node can be updated using (see also Figure 2.7)

$$\begin{aligned}\Pr[X_k = 0|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}] &= \alpha \prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} \Pr[X_k = 0|C_i] \\ \Pr[X_k = 1|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}] &= \alpha \prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} \Pr[X_k = 1|C_i]\end{aligned} \tag{2.34}$$

where $\alpha = \frac{1}{\Pr[X_k=0|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}]+\Pr[X_k=1|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}]}$. When the messages are log-likelihood ratios instead the update rule changes to

$$\Lambda(X_k|C_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}) = \sum_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} \Lambda(X_k|C_i) \tag{2.35}$$

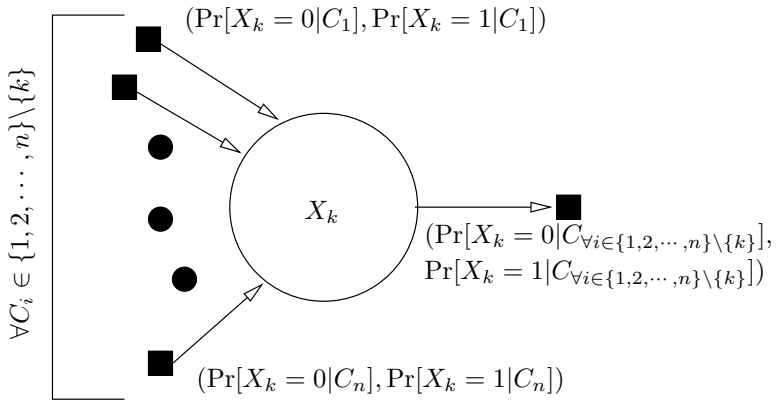

Figure 2.7: A variable(code) node from a factor graph of a sparse graph code with probability vectors as messages

The update rule derivation for the check node is a bit more involved and is done by first investigating what happens at a 3-port check node of a sparse graph code as depicted in Figure 2.8.

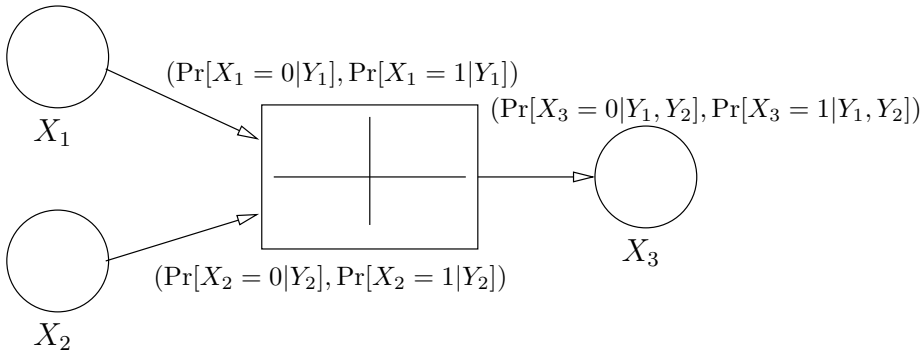

Figure 2.8: A check node from a factor graph of a sparse graph code with probability vectors as messages

The function of a 3 port check node is defined as

$$f_{\text{XOR}}(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } x_1 \oplus x_2 = x_3 \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

where $x_1, x_2, x_3 \in \{0, 1\}$ and $\oplus$ donates the binary xor operation. Again assuming the messages are probability vectors, then the update rule for $\Pr[X_3 = x_3|Y_1, Y_2]$ can be easily derived from equation (2.13) and is equal to

$$\Pr[X_3 = x_3|Y_1, Y_2] = \sum_{x_1,x_2} f_{\text{XOR}}(x_1, x_2, x_3) \Pr[X_1 = x_1|Y_1] \Pr[X_2 = x_2|Y_2] \tag{2.37}$$

which implies that the output probability vector $(\Pr[X_3 = 0|Y_1, Y_2], \Pr[X_3 = 1|Y_1, Y_2])$ can be calculated as

$$\begin{aligned}\Pr[X_3 = 0|Y_1, Y_2] &= \sum_{(x_1,x_2) \text{ so that } x_1 \oplus x_2 = 0} \Pr[X_1 = x_1|Y_1] \Pr[X_2 = x_2|Y_2] \\ &= \Pr[X_1 = 0|Y_1] \Pr[X_2 = 0|Y_2] + \\ &\quad \Pr[X_1 = 1|Y_1] \Pr[X_2 = 1|Y_2] \end{aligned} \tag{2.38}$$

and

$$\begin{aligned}\Pr[X_3 = 1|Y_1, Y_2] &= \sum_{(x_1,x_2) \text{ so that } x_1 \oplus x_2 = 1} \Pr[X_1 = x_1|Y_1] \Pr[X_2 = x_2|Y_2] \\ &= \Pr[X_1 = 0|Y_1] \Pr[X_2 = 1|Y_2] + \\ &\quad \Pr[X_1 = 1|Y_1] \Pr[X_2 = 0|Y_2] \end{aligned} \tag{2.39}$$

Note that if $2\Pr[X_1 = 0|Y_1] - 1 = p$ and $2\Pr[X_2 = 0|Y_2] - 1 = q$, then $2\Pr[X_1 \oplus X_2 = 0|Y_1, Y_2] - 1 = 2\Pr[X_3 = 0|Y_1, Y_2] - 1 = pq$ due to the fact that equation (2.40) is equal to equation (2.41) (shown by using equation (2.38))

$$\begin{aligned} pq &= (2\Pr[X_1 = 0|Y_1] - 1)(2\Pr[X_2 = 0|Y_2] - 1) \\ &= 4\Pr[X_1 = 0|Y_1] \Pr[X_2 = 0|Y_2] - \\ &\quad 2\Pr[X_1 = 0|Y_1] - 2\Pr[X_2 = 0|Y_2] + 1 \end{aligned} \tag{2.40}$$

$$\begin{aligned} 2\Pr[X_1 \oplus X_2 = 0|Y_1, Y_2] - 1 &= 2(\Pr[X_1 = 0|Y_1, Y_2] \Pr[X_2 = 0|Y_1, Y_2] + \\ &\quad \Pr[X_1 = 1|Y_1, Y_2] \Pr[X_2 = 1|Y_1, Y_2]) - 1 \\ &= 2(\Pr[X_1 = 0|Y_1] \Pr[X_2 = 0|Y_2]) + \\ &\quad 2((1 - \Pr[X_1 = 0|Y_1])(1 - \Pr[X_2 = 0|Y_2])) - 1 \\ &= pq \end{aligned} \tag{2.41}$$

Also note that if another edge is added to the check node then $2\Pr[X_1 \oplus X_2 = 0|Y_1, Y_2] - 1 = r$ and $2\Pr[X_4 = 0|Y_4] - 1 = s$ then $2\Pr[(X_1 \oplus X_2) \oplus X_4 = 0|Y_1, Y_2, Y_4] - 1 = rs = pqs$ using the same reasoning as before and therefore for an n-port node

$$\begin{aligned} 2\Pr[X_k = 0|Y_{\forall i \in \{1,2,\cdots,n\} \setminus \{k\}}] - 1 &= 2\Pr[X_1 \oplus X_2 \oplus \cdots \oplus X_{k-1} \oplus X_{k+1} \oplus \cdots \oplus X_n = 0 \\ &\quad |Y_{\forall i \in \{1,2,\cdots,n\} \setminus \{k\}}] - 1 \\ &= \prod_{\forall i \in \{1,2,\cdots,n\} \setminus \{k\}} (2\Pr[X_i = 0|Y_i] - 1) \\ &= \prod_{\forall i \in \{1,2,\cdots,n\} \setminus \{k\}} (\Pr[X_i = 0|Y_i] - \Pr[X_i = 1|Y_i]) \end{aligned} \tag{2.42}$$

Showing that the output probability vectors can be updated with

$$\begin{aligned}\Pr[X_k = 0|Y_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}] &= \frac{1}{2} + \frac{1}{2}\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} (\Pr[X_i = 0|Y_i] - \\ &\quad \Pr[X_i = 1|Y_i]) \\ \Pr[X_k = 1|Y_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}] &= \frac{1}{2} - \frac{1}{2}\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} (\Pr[X_i = 0|Y_i] - \\ &\quad \Pr[X_i = 1|Y_i]) \end{aligned} \tag{2.43}$$

Now by equation (2.32)

$$P[X_i = 0|Y_i] = \frac{\lambda(X_i|Y_i)}{1+\lambda(X_i|Y_i)} \tag{2.44}$$

leading to

$$\begin{aligned} 2\Pr[X_i = 0|Y_i] - 1 &= \frac{\lambda(X_i|Y_i) - 1}{\lambda(X_i|Y_i) + 1} \\ &= \frac{e^{\Lambda(X_i|Y_i)} - 1}{e^{\Lambda(X_i|Y_i)} + 1} \\ &= \tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right) \end{aligned} \tag{2.45}$$

which means that

$$\Pr[X_k = 0|Y_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}] = \frac{1}{2} + \frac{1}{2}\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} \tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right) \tag{2.46}$$

$$\Pr[X_k = 1|Y_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}] = \frac{1}{2} - \frac{1}{2}\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} \tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right). \tag{2.47}$$

Using equation (2.46) and equation (2.47) it is possible to calculate the log-likelihood message sent out from the k^{th} edge as:

$$\begin{aligned}\Lambda(X_k|Y_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}) &= \ln\frac{1+\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}\tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right)}{1-\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}\tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right)} \\ &= 2\tanh^{-1}\left(\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}\tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right)\right). \end{aligned} \tag{2.48}$$

This concludes the derivation of the update rules for belief propagation.
It is also clear that equation (2.30) changes to

$$\begin{aligned}\Pr[X_k = 0|C_{\forall i\in\{1,2,\cdots,n\}}] &= \alpha\prod_{\forall i\in\{1,2,\cdots,n\}}\Pr[X_k = 0|C_i] \\ \Pr[X_k = 1|C_{\forall i\in\{1,2,\cdots,n\}}] &= \alpha\prod_{\forall i\in\{1,2,\cdots,n\}}\Pr[X_k = 1|C_i] \end{aligned} \tag{2.49}$$

with

$$\alpha = \frac{1}{\Pr[X_k = 0|C_{\forall i\in\{1,2,\cdots,n\}}] + \Pr[X_k = 1|C_{\forall i\in\{1,2,\cdots,n\}}]}$$

and

$$\Lambda(X_k|C_{\forall i\in\{1,2,\cdots,n\}}) = \sum_{\forall i\in\{1,2,\cdots,n\}} \Lambda(X_k|C_i) \tag{2.50}$$

when probability vectors and LLR's are used for messages.

2.8.3 FACTOR GRAPH CONSTRUCTION

A factor or Tanner graph [29] can be constructed for any parity check matrix $\mathbf{H}(M\times N)$ or generator matrix $\mathbf{G}(K\times L)$. Loosely speaking a Tanner graph is a factor graph that has probability vectors as messages (set member indicator function [27]) and factors as a product of checks. The Hamming (7,4,3) code will be used as an example to demonstrate how a factor graph is constructed [30].
The $\mathbf{H}(3\times 7)$ matrix of the Hamming (7,4,3) code is expressed as:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{2.51}$$

The graph constructed from $\mathbf{H}$ contains 3 check nodes $r_m; m = 1,\cdots,3$ and 7 code nodes $q_n; n = 1,\cdots,7$. The way code nodes connect to check nodes is dictated by the rows of $\mathbf{H}$. The first row gives the parity check equation $q_1 \oplus q_2 \oplus q_3 \oplus q_5 = r_1$. This in effect means code nodes $\{1,2,3,5\}$ connect to check node 1. This is repeated for each row. The resulting factor graph can be seen in Figure 2.9. Similarly the columns of $\mathbf{H}$, when interpreted as incidence vectors, indicate in which parity-check equations code symbols appear or participate in. The leftmost column of $\mathbf{H}$ indicates that code node 1 is connected to check nodes $\{1,3\}$

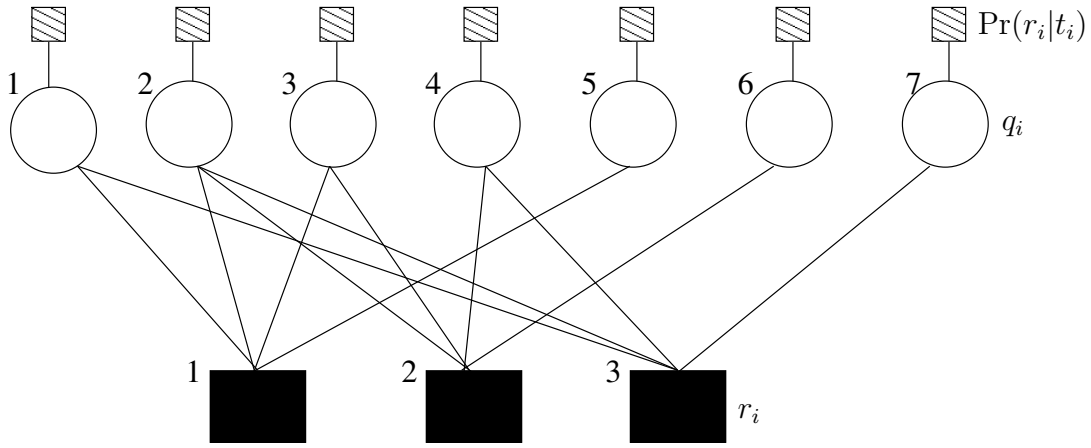

Figure 2.9: Factor graph representation of the parity-check matrix of the Hamming (7,4,3) code

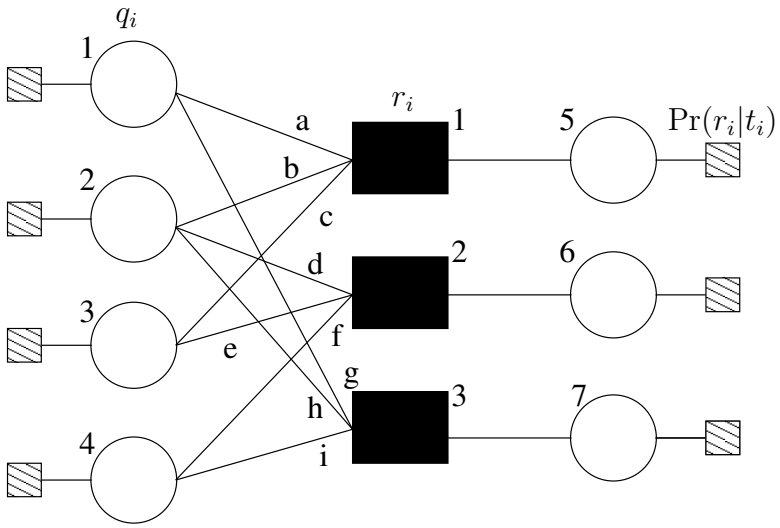

Figure 2.10: Factor graph representation of the generator matrix of the Hamming (7,4,3) code

The $\mathbf{G}(4 \times 7)$ matrix of the Hamming (7,4,3) code is equal to

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{2.52}$$

The graph constructed from $\mathbf{G}$ contains 3 check nodes $r_m; m = 1, \cdots 3$ and 7 code nodes $q_n; n = 1, \cdots, 7$. The graph contains three layers. The first layer consists of the code

nodes representing the original message bits (this is the identity matrix that takes up to 4 rows and 4 columns of $\mathbf{G}$) in this case code nodes $\{1, 2, 3, 4\}$. These nodes are also known as message nodes. The second layer consists of 3 check nodes (the r layer in Figure 2.10). The third layer consists of the remaining code nodes $\{5, 6, 7\}$ and are also known as parity nodes. The $\mathbf{G}$ matrix actually defines which message nodes are connected to which parity nodes (indirectly). This is true since the second and third layer are connected one on one meaning check node 1 is connected to code node 5 and so on. So the remaining columns of $\mathbf{G}$ determine which message nodes connect to which check nodes. For instance column 5 shows that code nodes $\{1, 2, 3\}$ connect to check node 1. This is repeated for the remaining columns. The resulting factor graph can be seen in Figure 2.10. The rows of $\mathbf{G}$ (the non identity part-from column 5 onwards) determines which check nodes are connected to a specific message node. For instance row 1 shows that check nodes $\{1, 3\}$ connect to code node 1. If no Identity matrix is contained in $\mathbf{G}$, a fake Identity matrix is inserted with dimensions $K \times K$ and the graph is constructed as described above. The nodes constructed by the fake Identity matrix is not transmitted leading to a non-systematic code. Take note that all code nodes are connected to special function nodes that can only send the observed value $\Pr(r_i|t_i)$ (see also equation (2.55), take note that r_i represents a soft received value and not check node i). If the graph was generated by a non-systematic $\mathbf{G}$ matrix the special function nodes are all set to 0 (if messages are LLR) for all message nodes.

2.8.4 BP FOR LDPC CODES

The complete belief propagation algorithm can now be given for decoding sparse graph codes using the $\mathbf{H}$ matrix. This is also known as LDPC decoding [14, 30]. By using this example it is possible to construct a decoding algorithm for any factor graph whether constructed from $\mathbf{H}$ or $\mathbf{G}$.

The following notation is introduced to make the description of the belief propagation algorithm clearer. Let $h_{i,j}$ denote the entry of $\mathbf{H}$ in the i^{th} row and j^{th} column. Let

$$\mathcal{L}(m) = \{l : h_{m,l} = 1\} \tag{2.53}$$

denote the set of code positions that participate in the m^{th} parity-check equation, and let

$$\mathcal{M}(l) = \{m : h_{m,l} = 1\} \tag{2.54}$$

denote the set of check equations in which code position l participates. The algorithm iteratively computes two types of conditional probabilities:

- $q^x_{m,l}$: The probability that the l-th bit of the code word $\mathbf{t}$ has the value x, given the information obtained via the check nodes other than check node m. This quantity can also be expressed as an LLR value, $q^{\mathrm{L}}_{m,l}$.

- $r^x_{m,l}$: The probability that a check node m is satisfied when bit l is fixed to a value x and the other bits are independent with probabilities $q^x_{m,l'}, l' \in \mathcal{L}(m)\backslash l$. This quantity can also be expressed as an LLR value, $r^{\mathrm{L}}_{m,l}$.

Further assuming that the modulated symbols $m(t_l)$ (BPSK modulation) are sent over an AWGN channel with flat fading added and received as r_l. Also assume that the AWGN channel has zero mean and variance $N_0/2$ and the instantaneous average fading amplitude of r_l is α_l. With the above in mind it is possible to calculate the probability $\Pr(r_l|t_l)$ of the l-th code bit after transmission through an AWGN channel with flat fading added as [22]

$$\Pr(r_l|t_l) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(r_l - \alpha_l t_l)^2}{2\sigma^2}\right] \tag{2.55}$$

with $2\sigma^2 = N_0/2$. Note that the aim of the algorithm is to calculate equation (2.31), which can only be approximated due to loops contained in the factor graphs of sparse graph codes. Now the BP algorithm can be introduced [14, 30].

Initialization

For $l \in \{1, 2, \cdots, N\}$, initialize the a-priori probabilities of the code nodes (these values form the special function nodes of the factor graph discussed in Section 2.8.3),

$$p^0_l = \Pr(r_l|t_l = 0[-1]) = \frac{1}{1 + \exp(r_l \alpha_l \frac{4}{N_0})} \tag{2.56}$$

$$p^{\mathrm{L}}_l = \ln\left[\frac{\Pr(r_l|t_l = 0[-1])}{\Pr(r_l|t_l = 1)}\right] = \Lambda(r_l|t_l) = -r_l\alpha_l\frac{4}{N_0} \tag{2.57}$$

and is calculated by using equation (2.55) and assigning $t_l = -1, 1$ (BPSK modulation). Also $p^0_l = 1 - p^1_l$.

For every (l, m) such that $h_{m,l} = 1$,

$$q^0_{m,l} = p^0_l, \qquad q^1_{m,l} = p^1_l, \qquad q^{\mathrm{L}}_{m,l} = p^{\mathrm{L}}_l \tag{2.58}$$

Message Passing

Step1: Bottom-up (horizontal)

For each l, m compute

$$\delta r_{m,l} = \prod_{l' \in \mathcal{L}(m)\backslash l} (q^0_{m,l'} - q^1_{m,l'}) \tag{2.59}$$

and

$$r^0_{m,l} = (1 + \delta r_{m,l})/2 \qquad r^1_{m,l} = (1 - \delta r_{m,l})/2 \tag{2.60}$$

$$r^{\mathrm{L}}_{m,l} = 2\tanh^{-1}\left(\prod_{l' \in \mathcal{L}(m)\backslash l} \tanh(q^{\mathrm{L}}_{m,l'}/2)\right) \tag{2.61}$$

which is derived from equations (2.43) and (2.48).

Step2: Top-down (vertical)

For each l, m compute

$$q^0_{m,l} = p^0_l \prod_{m' \in \mathcal{M}(l)\backslash m} r^0_{m',l} \qquad q^1_{m,l} = p^1_l \prod_{m' \in \mathcal{M}(l)\backslash m} r^1_{m',l} \tag{2.62}$$

and normalize, with $\alpha = 1/(q^0_{m,l} + q^1_{m,l})$,

$$q^0_{m,l} = \alpha q^0_{m,l} \qquad q^1_{m,l} = \alpha q^1_{m,l} \tag{2.63}$$

$$q^{\mathrm{L}}_{m,l} = p^{\mathrm{L}}_l + \sum_{m' \in \mathcal{M}(l)\backslash m} r^{\mathrm{L}}_{m',l} \tag{2.64}$$

which is derived from equations (2.34) and (2.35). The reason for adding the p^k_l term is to incorporate the special function nodes discussed in Section 2.8.3. For each l compute the a-posteriori probabilities,

$$q^0_l = p^0_l \prod_{m \in \mathcal{M}(l)} r^0_{m,l} \qquad q^1_l = p^1_l \prod_{m \in \mathcal{M}(l)} r^1_{m,l} \tag{2.65}$$

and normalize with, $\alpha = 1/(q^0_l + q^1_l)$,

$$q^0_l = \alpha q^0_l \qquad q^1_l = \alpha q^1_l \tag{2.66}$$

$$q^{\mathrm{L}}_l = p^{\mathrm{L}}_l + \sum_{m \in \mathcal{M}(l)} r^{\mathrm{L}}_{m,l} \tag{2.67}$$

which is derived from equations (2.49) and (2.50).

For $n = 1, 2, \cdots N$ compute

$$\Pr(t_n = 0|\boldsymbol{r}) \approx q^0_n \qquad \Pr(t_n = 1|\boldsymbol{r}) \approx q^1_n \qquad \Lambda(t_n|\boldsymbol{r}) \approx q^{\mathrm{L}}_n \tag{2.68}$$

due to equation (2.20).

Decoding

For $n = 1, 2, \cdots N$ compute

$$y_n = \begin{cases} 0 & \text{if } q^0_n > 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{2.69}$$

or

$$y_n = \begin{cases} 0 & \text{if } q_n^{\mathrm{L}} > 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.70}$$

where y_n is the n-th bit of the decoded codeword $\boldsymbol{y}$. If $\boldsymbol{yH} = \mathbf{0}$, then $\boldsymbol{y}$ is the estimated codeword. Otherwise return to step 1. If the number of iterations exceed a predetermined threshold, a decoding failure is declared and the hard received values are given directly as the output.

2.8.5 BP DECODING EXAMPLE

The BP algorithm for LDPC codes was formally introduced in Section 2.8.4. In this section a more general approach will be followed to show how BP decoding can be applied to any factor graph using the update rules of Section 2.8.2 directly on a factor graph of a Hamming (7,4,3) **G** matrix (see Section 2.8.3).

The belief propagation algorithm will now be applied to the factor graph found in Figure 2.10. Assuming the following corrupted codeword $\mathbf{r} = \big($-0.3505, 0.2181, 0.6464, -0.3264, -0.5954, 0.1677, -0.0654$\big)$ was received by the receiver (very low SNR) in the form of $-\Lambda(r_i|t_i)$ (see equation (2.55)).

Firstly the initialization phase of the belief propagation algorithm needs to be implemented. Since only the special function nodes can send messages (equal to the observed soft value $-\Lambda(r_i|t_i)$) in the beginning of the algorithm these messages will propagate down each edge that is indirectly connected to them through a code node (because all other messages are equal to 0). Hence the edges connected to q_1 will all have the value of $-\Lambda(r_1|t_1) = -0.3505$ after initialization as can be seen for edges a and g in Table 2.1. It becomes clear from the above that the initialization phase is actually a complete iteration of belief propagation (one update of all check and code nodes). The remaining messages for the initialization phase can be found in Table 2.1 and was obtained in the same manner.

The next step is to update all the edges of the check nodes using equation (2.48). But since the received values are $-\Lambda$ equation (2.48) changes to

$$\Lambda(X_k|Y_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}}) = -2\tanh^{-1}\left(\prod_{\forall i\in\{1,2,\cdots,n\}\setminus\{k\}} -\tanh\left(\frac{\Lambda(X_i|Y_i)}{2}\right)\right) \tag{2.71}$$

To update message a at check node r_1 messages $\{\mathrm{b}, \mathrm{c}, -\Lambda(r_5|t_5)\}$ are needed.

$$\begin{aligned}\mathrm{a} &= \mu_{r_1 \to q_1} \\ &= -2\tanh^{-1}\left((-1)^3 \cdot \tanh\left(\frac{0.2181}{2}\right) \cdot \tanh\left(\frac{0.6464}{2}\right) \cdot \right. \\ &\quad \left. \tanh\left(\frac{-0.5954}{2}\right)\right) \\ &= -0.0196 \end{aligned} \tag{2.72}$$

The remaining updated messages can be found in Table 2.1 and were obtained in the same manner.

The next step is to update all the edges of the check nodes using equation (2.35). To update message a at code node q_1 messages $\{\mathrm{g}, -\Lambda(r_1|t_1)\}$ are needed

$$\begin{aligned}\mathrm{a} &= \mu_{q_1 \to r_1} \\ &= (0.0011) + (-0.3505) \\ &= -0.3494 \end{aligned} \tag{2.73}$$

The remaining updated messages can be found in Table 2.1 and were obtained in the same manner.

Table 2.1: One iteration of belief propagation on a Hamming (7,4,3) code

Edge	Initialization	Update Check	Update Code
a	-0.3505	-0.0196	-0.3494
b	0.2181	0.0313	0.2078
c	0.6464	0.0109	0.6434
d	0.2181	-0.0085	0.2476
e	0.6464	-0.0029	0.6573
f	-0.3264	0.0057	-0.3252
g	-0.3505	0.0011	-0.3701
h	0.2181	-0.0018	0.2410
i	-0.3264	0.0012	-0.3207

Now one iteration of belief propagation has been completed (actually two if the initialization phase is included) and $-\Lambda(t_i|\boldsymbol{r})$ can be approximated by using equation (2.50). For q_1 this will be equal to

$$\begin{aligned}-\Lambda(t_1|\boldsymbol{r}) &\approx (0.0011) + (-0.3505) + (-0.0196) \\ &= -0.3690 \end{aligned} \tag{2.74}$$

This can be continued for all $i \in N$ and the result of this can be found in Table 2.2.

Table 2.2: $-\Lambda(t_i|\boldsymbol{r})$ after 1 iteration of BP (Hamming (7,4,3))

$-\Lambda(t_1\|\boldsymbol{r})$	$-\Lambda(t_2\|\boldsymbol{r})$	$-\Lambda(t_3\|\boldsymbol{r})$	$-\Lambda(t_4\|\boldsymbol{r})$	$-\Lambda(t_5\|\boldsymbol{r})$	$-\Lambda(t_6\|\boldsymbol{r})$	$-\Lambda(t_7\|\boldsymbol{r})$
-0.3690	0.2392	0.6543	-0.3195	-0.6072	0.1568	-0.0593

From Table 2.2 the decoded codeword $\boldsymbol{y}$ can be obtained and is equal to $\boldsymbol{y} = \big(0,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0\big)$. If this is not a valid codeword the algorithm needs to be repeated, otherwise $\boldsymbol{y}$ is the output. If a predetermined number of iterations have been performed without success the hard limited original received codeword $\boldsymbol{r}$ is given as output.

CHAPTER 3

FOUNTAIN CODES

3.1 CHAPTER SUMMARY

The chapter provides a general overview of fountain code development. After the overview each of the three fountain codes (Tornado, LT, Raptor) are analyzed under the following general headings: encoding, decoding on erasure channel, design, noisy decoding, factor graph representation.

3.2 OVERVIEW OF FOUNTAIN CODES

Digital fountain codes use the concept of a sparse-graph at their core. The first sparse-graph code was introduced by R.G. Gallager in [7, 8] in the early 1960s and is known as LDPC codes. One of the code families derived from the sparse-graph concept was digital fountain codes. Digital fountain codes are designed for the binary erasure channel [31], in which each codeword symbol is lost with a fixed constant probability p in transit independent of all the other symbols.

The digital fountain concept [32, 33] started as a data carousel (broadcast disk) [34]. In this approach the source loops through all transmission packets continuously; the receivers may log onto this stream at any time and download packets until they have received the entire message. A data carousel can be seen as an imperfect approximation of an ideal solution, which is referred to as a digital fountain. The difference between a data carousel and a digital fountain is that a digital fountain can reconstruct the message from any subset of encoding packets equal in length to the original message (note that the packets are encoded using some sort of FEC), while the decoding properties of a data carousel is not as strict (the packets used does not have to be any subset or a specific length). A digital fountain can be compared to a running tap. When a cup is filled it is not important which droplets land in the cup, but only that enough water is required to fill the cup. The above metaphor also highlights another important property of digital fountain codes namely its ability to generate an infinite amount of encoded packets from the original source.

One of the first coding schemes chosen to create digital fountains is Reed-Solomon (RS) codes [6, 35, 36]. The reason RS codes were considered is the fact that the original message can be constructed from k RS symbols, where k represents the original message size (in packets) and n represents the code size (in packets). The two main drawbacks of RS codes are the limited amount of distinct decoding symbols and the quadratic algorithm used to decode it [32, 37].

The next coding scheme proposed to approximate a digital fountain is a sparse-graph erasure code called Tornado codes [38, 39]. The sparse property of Tornado codes solved the quadratic time problem of RS codes (made directly proportional to n). This does come with a price; the receivers have to receive a little bit more than k packets to correctly decode the original message. Note that a Tornado code is not a true fountain code (rather a predecessor), due to the fact that it can not produce an unlimited supply of unique encoded bits.

The next logical evolution was to develop a rateless code which is a true fountain code. Luby Transform (LT) codes were the first rate less codes and were discovered by Luby [40, 41, 14]. LT codes have a similar graph structure than Tornado codes, though its graph is not predefined giving LT codes their rateless property. To make this structure possible each encoded symbol must store a list of its neighbors in real time, which can be accomplished in practice if the receiver and transmitter share the same random seed and degree distribution. They can agree on this before transmission begins. The underlying degree distribution is a very important design choice and the robust soliton distribution is a popular choice [40]. Using this degree distribution leads to encoding symbols with average degree $O(\ln k)$. Encoding symbols with average degree $O(\ln k)$ leads to codes that can encode and decode in time proportional to $O(k \ln k)$ [40]. These codes also require that the receiver receive a little more than k packets before accurate decoding can be accomplished. A detailed comparison of the implementation of RS, Tornado and LT codes as digital fountains can be found in [32, 33].

The only improvement over LT codes is to lower the average degree of encoding symbols to a constant which would lead to decoding in time proportional to $O(k)$. Raptor codes [42, 43, 41, 44] accomplish just this by using pre-coding. An average degree distribution of $O(\ln k)$ is required to cover each message node with high probability. To remove this restriction, a message can be pre-coded using an erasure code like Tornado. If an encoded message M' is treated as the message then it is not necessary to recover every packet of M' in order to recover M (original message), but instead just a constant fraction of the packets of M'. This reduces the decoding time complexity to $O(k)$.

Research on using digital fountain codes on non-erasure channels were presented in [43, 45, 23, 46, 26]. Take note that fountain codes consist of at least 10000 encoded bits (it becomes efficient at such large lengths). The above is summarized in Table 3.1

Table 3.1: The different coding schemes used in the past to approximate a digital fountain code

Code	Negatives	Positives
RS	Limited distinct decoding symbols	Can decode with k encoded packets
	Quadratic decoding algorithm	
Tornado	Need more than k encoded packets to decode	Decoding time proportional to n
	Limited distinct decoding symbols	
LT	Need more than k encoded packets to decode	Decoding time proportional to $k.\ln k$
		Unlimited distinct decoded symbols
Raptor	Need more than k encoded packets to decode	Decoding time proportional to k (due to pre-coding)
		Unlimited distinct decoded symbols

3.3 TORNADO CODES

The Tornado code was one of the first codes used to approximate a fountain. Its basic structure, encoding and decoding algorithms are presented in the following sections. The most popular design approach is also discussed below. This section closely follows the notation of [38, 26].

3.3.1 BASIC STRUCTURE

The basic structure of a Tornado code consists of layered bipartite graphs. Each bipartite graph B is associated with a code $\mathcal{C}(B)$ with k message bits and βk redundant bits (check, parity bits), where $0 < \beta < 1$. Each graph B thus consists of k left nodes and βk right nodes.

The encoding of $\mathcal{C}(\beta)$ is accomplished by setting each check bit equal to the $\oplus$ (mod-2 sum) of its neighboring message bits. The codes used are systematic and sparse. Due

to a predefined graph structure Tornado codes are not rateless like LT codes.
The decoding of a layer on an erasure channel is accomplished by substituting the correctly received bits into the graph structure and then solving the unknown bits iteratively using the following decoding operation.

Given the value of a check bit and all but one of the message bits on which it depends, set the missing message bit (or check bit depending on which layer) to be the $\oplus$ of the check bit and its known message bits (check bits).

This procedure only works if enough bits are received correctly. The above is explained in more detail in [38] (an example is given in section 3.4.2).
Now that a single bipartite graph was introduced it can be cascaded to form a Tornado code. First $\mathcal{C}(B)$ is used to produce βk check bits for the original k message bits. Then a similar code is used to produce $\beta^2 k$ check bits for the βk check bits of $\mathcal{C}(B)$, and so on. The last level of a Tornado code may use a conventional erasure correcting code like RS (Reed-Solomon) [38]. Implementing Tornado codes on non-erasure channels using belief propagation will prohibit this approach since Reed-Solomon codes are not sparse codes.
To circumvent this problem the last layer consists of a unique bipartite graph that specializes in maximizing error recovery of left as well as right hand side nodes. The other layers are only designed to maximize error recovery of left hand nodes. The reason being that these layers assume that the conventional error correcting code has performed all the necessary corrections of the left nodes, at the last layer of the graph.
The Tornado graph structure can now be formally defined as follows [38]. A family of codes $\mathcal{C}(B_0), \cdots, \mathcal{C}(B_m)$ can be constructed from a family of graphs $B_0, \cdots, B_m$, where B_i has $\beta^i k$ left nodes and $\beta^{i+1} k$ right nodes. The variable m is chosen that $\beta^{m+1} k$ is roughly $\sqrt{k}$. The cascade ends with an erasure correcting code C of rate $1 - \beta$ with $\beta^{m+1} k$ message bits. The code $\mathcal{C}(B_1, \cdots, B_m, C)$ has k message bits and

$$\sum_{i=1}^{m+1} \beta^i k + \beta^{m+2} k/(1-\beta) = k\beta(1-\beta) \tag{3.1}$$

check bits formed by using $\mathcal{C}(B_0)$ to produce βk check bits for the k message bits, using $\mathcal{C}(B_i)$ to form $\beta^{i+1} k$ check bits for the $\beta^i k$ bits produced by $\mathcal{C}(B_{i-1})$ and finally using

C to produce an additional $k\beta^{m+2}/(1-\beta)$ check bits for the $\beta^{m+1}k$ bits output by $\mathcal{C}(B_m)$. As $\mathcal{C}(B_0, B_1, \cdots, B_m, C)$ has k message bits and $k\beta/(1-\beta)$ check bits, it is a code of rate $1-\beta$.

The decoding of a complete Tornado code on an erasure channel is done as follows. The conventional erasure code recovers all the missing left and right check nodes of the last layer, making all the check bits of $\mathcal{C}(B_m)$ known. This, with all the other received values can now be substituted into the remaining graph. Now the graph can be decoded (bit by bit), if enough bits were received correctly, by using the decoding operation described earlier in this section. The code is called a Tornado since if enough bits are received the decoding happens at once (quickly), one after the other. This has the appearance of a Tornado storm [32, 38].

A Tornado code has encoding and decoding times proportional to $n\ln(1/\epsilon)$, where $1+\epsilon$ $(0 < \epsilon < 1)$ is known as the *decoding inefficiency*. In other words a Tornado code requires at least $(1+\epsilon)k$ packets to decode. The encoding and decoding time dependence come from the fact that for a Tornado code the speed of encoding and decoding depend on the amount of edges in the graph. Since the average degree of a node is equal to $\ln(1/\epsilon)$ and the conventional error correcting code C has about $\sqrt{k}$ left nodes, encoding and decoding becomes linearly proportional to the code length n [38].

3.3.2 DEGREE SEQUENCES

An edge is a line that connects any two nodes, one from the left with one on the right. Edges that are adjacent to a node of degree i on the left (or right) is referred to as edges of degree i on the left (or right), where the degree of a node is determined by the amount of edges connected to it. Each of the degree sequences is specified by a pair of vectors $(\lambda_1, \cdots, \lambda_m)$ and $(\rho_1, \cdots, \rho_m)$, where λ_i is the initial fraction of edges on the left of degree i and ρ_j is the initial fraction of edges on the right of degree j. Note that the graphs are specified in terms of fractions of edges, and not nodes, of each degree. The sequence λ give rise to a generating polynomial $\lambda(x) = \sum_i \lambda_i x^{i-1}$. The average left degree of the graph is thus equal to

$$a_l = \left[\sum_i \lambda_i / i\right]^{-1} \tag{3.2}$$

If E is the number of edges in the graph, then the number of left nodes of degree i is

$$E\lambda_i / i \tag{3.3}$$

and hence the number of left nodes is

$$E \sum_i \lambda_i / i \tag{3.4}$$

The above can be repeated for the right side nodes, as long as E remains constant [38]. For an effective erasure correcting code the degree sequences have to satisfy the following theorem from [38].

Theorem 2 *Let k be an integer, and suppose that $\mathcal{C} = \mathcal{C}(B_1, \cdots, B_m, C)$ is a cascade of bipartite graphs as explained in section 3.3.1, where B_1 has k left nodes. Suppose that each B_i is chosen at random with edge degrees specified by $\lambda(x)$ and $\rho(x)$, such that $\lambda(x)$ has $\lambda_1 = \lambda_2 = 0$, and suppose δ is such that*

$$\rho(1 - \delta \cdot \lambda(x)) > 1 - x \tag{3.5}$$

for $x \in (0, 1]$. Then, if at most a δ-fraction of the coordinates of an encoded word in $\mathcal{C}$ are erased independently at random, the erasure decoding algorithm of section 3.3.1 terminates successfully with probability $1 - O(k^{3/4})$, and does so in $O(k)$ steps.

3.3.3 DESIGNING A TORNADO CODE FOR THE ERASURE CHANNEL

The Tornado code consists of multiple layers of bipartite graphs. A popular design for these layers is discussed in this section and is summarized in figure 3.1.

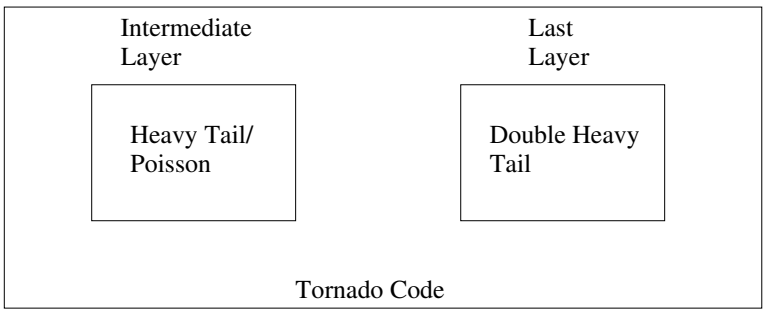

Figure 3.1: A popular design of a Tornado code

All of the intermediate layers are designed using the heavy tail distribution for the left nodes (discussed below). The last layer is designed using the double heavy tail distribution for the left nodes (discussed below). This is done due to the fact that the

double heavy tail distribution is more effective than the heavy tail distribution when correcting erasures on the right and left nodes of the bipartite graph, according to [38]. The code in this section is designed to satisfy the condition represented by equation (3.5).

An intermediate layer is designed in the following manner as discussed in [38]. Let B be an intermediate bipartite graph with k left nodes and βk right nodes. The left degree sequence is described by the following truncated heavy tail distribution. Let $H(D) = \sum_{i=1}^{D} 1/i$ be the truncated harmonic sum truncated at D, where D is an integer used to trade off the average degree with how well the decoding process works. The fraction of edges of degree $i = 2, \cdots, D+1$ on the left is given by

$$\lambda_i = 1/(H(D)(i-1)) \tag{3.6}$$

From equations (3.2) and (3.6) it can be shown that $a_l = H(D)(D+1)/D$. The average right degree a_r needs to satisfy a_l/β. The right degree sequence is defined by the Poisson distribution with mean a_r : for all $i \geq 1$ the fraction of edges of degree i on the right equals

$$\rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!} \tag{3.7}$$

where α is chosen such that the average degree on the right is equal to a_r. In other words, α satisfies

$$\alpha e^{\alpha}/(e^{\alpha}-1) = a_r \tag{3.8}$$

This approach however does not work due to the fact that there are nodes of degree two on the left. To overcome this problem a small change is made in the structure of B. Let $\gamma = \beta/D^2$. The βk right nodes of B can be split into two distinct sets, the first set consisting of $(\beta-\gamma)k$ nodes and the second set consisting of γk nodes. The graph B is then formed by taking the union of the two graphs B_1 and B_2. B_1 is formed between the k left nodes and $(\beta-\gamma)k$ right nodes as described above. B_2 is formed between the k left nodes and the second set of γk right nodes, where each of the k left nodes has degree three and the $3k$ edges are connected randomly to the γk right nodes.

For this graph $\delta = \beta(1-1/D)$ and is defined by equation (3.5).

The last layer is designed using a different technique than the other layers [38]. The edge distribution on the left is now a double heavy tail. In other words $\lambda(x) = \bar{\lambda}(x^2)$, where $\bar{\lambda}$ is the edge distribution function of the heavy tail distribution. The right edge distribution is calculated using linear programming (simplex method [47]). Assuming that the right side only needs to satisfy equation (3.5). From $\lambda(x)$ and δ (choosing

it at least equal to $\delta = \beta(1 - 1/D))$ $\rho(x)$ can be calculated. The objective is to find $\rho_m; m \in M$, where M is a fixed set of positive integers and has a size of at least N. Let $x_i = 1/N$ for $i = 1, 2, \cdots, N$. The simplex method can now be used to minimize

$$\sum_i (\rho(1 - \delta\lambda(x_i)) + x_i - 1) \tag{3.9}$$

subject to $\rho_i > 0$, $\rho(1 - \delta\lambda(x_i)) > 1 - x_i$, $\sum_i \rho_i = 1$ and $\sum_k \lambda_k/k = \sum_i \rho_i/i$ (if the amount of left and right hand nodes are equal). This solution is only feasible if $\rho(1 - \delta\lambda(x)) > 1 - x$ for all $x \in (0, 1]$.

3.3.4 AN EXAMPLE DESIGN

This example is based on the example given in [26]. The first step in designing a Tornado code is choosing the dimensions of the code, and the amount of layers. A $(n, k) = (80, 40)$ code with 2 layers is considered here. Although a typical Tornado code consists of at least 10000 encoded bits, the example considered here is sufficient to highlight a flaw in Tornado codes when used on an AWGN channel with flat fading added. The first layer will consist of the heavy tail distribution as described in Section 3.3.3 with $D = 3$ and $\beta = 0.5$. The first layer will thus consist of 40 right nodes and 20 left nodes. The first layer consists of the union between two graphs B_1 and B_2 (see Section 3.3.3). The nodes on the right needs to be divided into two unique sets. First $\gamma = \beta/D^2 = 1/18$ needs to be calculated. Now B_1 will use $\lfloor(\beta - \gamma)k\rfloor = 17$ and B_2 will use 3 nodes on the right. Both graphs will use all of the left hand nodes. First B_1 is designed and then B_2 is added to form B. The polynomials $\lambda(x)$ and $\rho(x)$ of B_1 is designed by using the values calculated in Table 3.2 and can be found in Table 3.3. Note from Table 3.2 that a new β is used to calculate the average right degree. With $\lambda(x)$, $\rho(x)$ and equation (3.3) the degrees of each node can be determined. Since equation (3.3) does not divide the edges of B_1 perfectly, seven edges remain (only 92 edges are used). The remaining edges are donated to B_2. To complete B, B_2 needs to be added to B_1. This is easily accomplished by adding three edges to each left node and assigning these edges (including the seven donated ones) randomly to the three nodes of B_2. One such assignment is shown in Table 3.4.

The last layer of the graph is designed by using linear programming. It consists of the 20 parity bits of the first layer on the left and 20 parity bits on the right. Assuming D is also equal to three for the last layer and that δ remains the same as the previous layer $\lambda(x)$ and $\rho(x)$ can be calculated as shown in Table 3.5. With $\lambda(x)$ and equation

(3.4) one can calculate E, which is equal to 78 in this case. With E, $\lambda(x)$, $\rho(x)$, and equation (3.3) Table 3.6 can be constructed. The reason for a node of degree 3 on the right is that two edges remained after dividing the edges on the right. These two edges were assigned to a random node of degree one.

A random (80,40) Tornado code is shown in Figure 3.2. The first layer of nodes represent the original 40 message bits. The second 20 nodes represent the first layer of parity bits. The last 20 nodes represent the second layer of parity bits. Take note that the check nodes are omitted to make the factor graph representation more simple.

Table 3.2: Values needed for the design of B_1

Unknown	Value	Equation
E_{B_1}	99	(3.4)
β_{new}	0.425	17/40
a_l	2.444	(3.2)
a_r	5.5	$a_r = a_l/\beta_{new}$
α	5.477	(3.8)
δ	0.333	$\delta = \beta(1 - 1/D)$

Table 3.3: Polynomial degree distributions of B_1

Unknown	Equation
$\lambda(x)$	(3.6)
$\lambda(x) = 0.5455x + 0.2727x^2 + 0.1818x^3$	
$\rho(x)$	(3.7)
$\rho(x) \approx 0.0042 + 0.0229x + 0.0627x^2 + 0.1145x^3 + 0.1568x^4$ $0.17185x^5 + 0.1568x^6 + 0.1227x^7 + 0.084x^8$	
$E_{B_{1,new}} = 92$	$E_{B_2} = 127$
$E = 219$	

Table 3.4: Complete design of first layer

Left Nodes		Right Nodes	
Degree	**Amount**	**Degree**	**Amount**
5	26	2	1
6	9	3	2
7	5	4	3
		5	3
		6	3
		7	2
		8	2
		9	1
		35	$1(B_2)$
		46	$2(B_2)$

Table 3.5: Polynomial degree distributions of second layer

Unknown	**Equation**
$\lambda(x)$	$\lambda(x) = \bar{\lambda}(x^2)$
$\lambda(x) = 0.5455x^2 + 0.2727x^4 + 0.1818x^6$	
$\rho(x)$	(3.9)
$\rho(x) = 0.079 + 0.9921x^4$	
$E = 78$	

Table 3.6: Complete design of second layer

Left Nodes		Right Nodes	
Degree	**Amount**	**Degree**	**Amount**
3	13	1	5
5	5	3	1
7	2	5	14

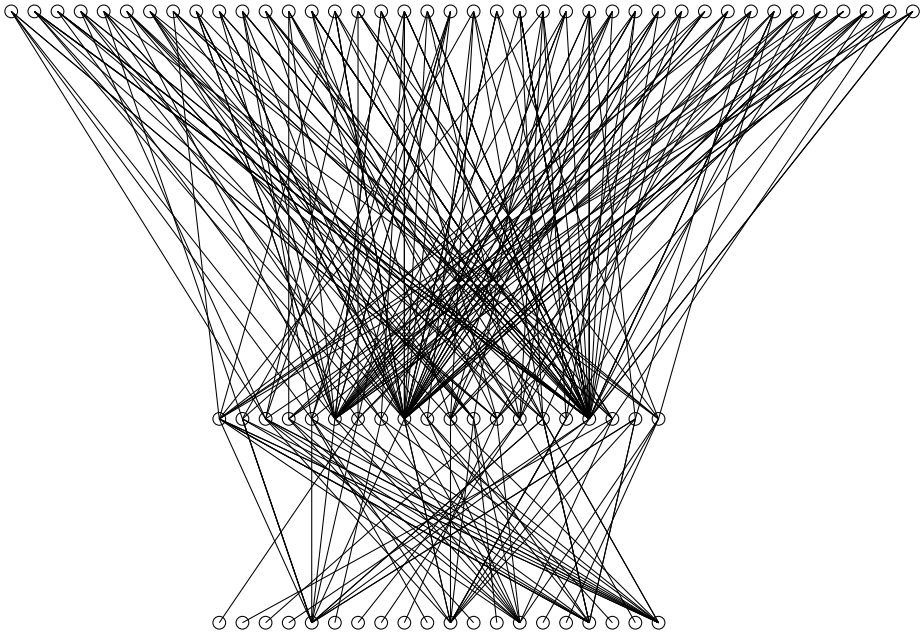

Figure 3.2: A random (80,40) Tornado code

3.3.5 FACTOR GRAPH REPRESENTATION

A factor graph of a simple Tornado code is given in Figure 3.3.

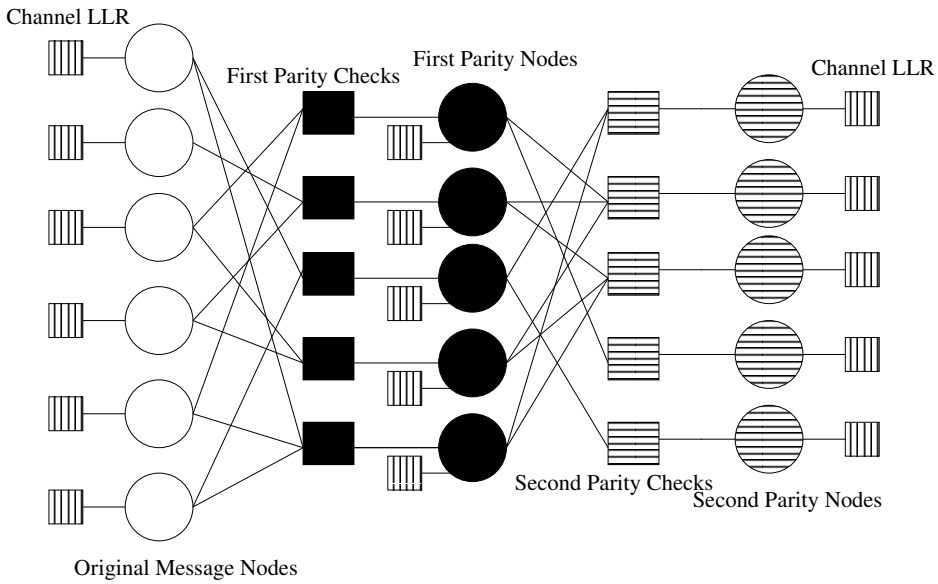

Figure 3.3: A simple factor graph of a two layer Tornado code

It is easy to see that a Tornado code can be decoded on a noisy channel by using the algorithm of Section 3.4.3.
The white circles represent the original message bits. The black circles represent the first check bits, while the circles filled with a horizontal pattern represent the check bits of the first parity layer. The check nodes of these bits are colored similarly. The vertically filled function blocks contain the channel LLR as defined by equation (3.10). Noisy decoding of Tornado codes is discussed in greater detail in [26].

3.4 LT CODE

The LT code was the first rate less code used to approximate a fountain. Its basic structure, encoding and decoding algorithms are presented in the following sections. The most popular design approach is also discussed below. This section closely follows the notation of [41].

3.4.1 ENCODER

An LT code is actually a dynamic LDGM (low-density generator matrix) code. It generates codewords through a random sparse **G** matrix. It is also rate less, for it can keep on deriving parity bits without end. The encoder can be described in simpler terms [41]:
Each encoded parity bit p_n can be generated from the message bits $m_1, m_2, \cdots, m_k$ through the following two steps:

1. Choose the degree d_n of p_n randomly from a degree distribution $\rho(d)$.

2. Choose at random d_n message bits, and set p_n equal to the bit wise sum, modulo 2, of the chosen message bits.

This procedure is exactly the same as multiplying a message with a dynamic random **G** matrix. It is important to note here that a standard LT code is non-systematic [40], since only the parity bits that were generated is sent through the erasure channel. To create a systematic LT code [23] is quite simple; send the message bits before the parity bits. This code comes in handy for decoding noisy codewords.

3.4.2 ERASURE DECODER

The decoding process on the erasure channel can be thought of as a simplification of the sum-product algorithm [41]. Since the messages being passed are either completely certain or unknown the algorithm simplifies greatly and can be summarized in the following steps [41]:

1. Find a parity bit p_n that is connected to only one message bit m_k (if there is no such parity bit, the decoding algorithm stops and fails to recover all message bits).
   (a) Set $m_k = p_n$.
   (b) Add m_k to all parity bits $p_{n'}$ that are connected to m_k:
   $p_{n'} := p_{n'} + m_k$ for all n' such that $G_{n'k} = 1$.
   (c) Remove all the edges connected to m_k.
2. Repeat 1 until all m_k has been recovered.

This algorithm is demonstrated for a toy case in Figure 3.4 [41]. The white circles represent message bits, while the black circles represent parity bits. A parity bit is the $\oplus$ of all the message bits connected to it. This is a simplification of the factor graph representation of Chapter 2 (where the code and check nodes were drawn separately). There are three message bits and four parity bits, which have values $p_1, p_2, p_3, p_4 = 1011$. During the first iteration, the only parity bit that is connected to one message bit is the first parity bit (see Figure 3.4(a)). This value is copied to m_1 (see Figure 3.4(b)), delete the parity bit, and then the new value of m_1 gets added to p_2 and p_4 (see Figure 3.4(c)). This disconnects m_1 from the graph. At the start of the second iteration (see Figure 3.4(c)), p_4 is connected to a single message bit m_2. Now one sets m_2 equal to p_4 (see Figure 3.4(d)), and then add this value to p_2 and p_3 (see Figure 3.4(e)). Finally one sees that the parity bits connected to m_3 are equal as expected and can be used to restore m_3 (see Figure 3.4(f)).
What is clear from the above is that this procedure works nicely on non-systematic LT codes.

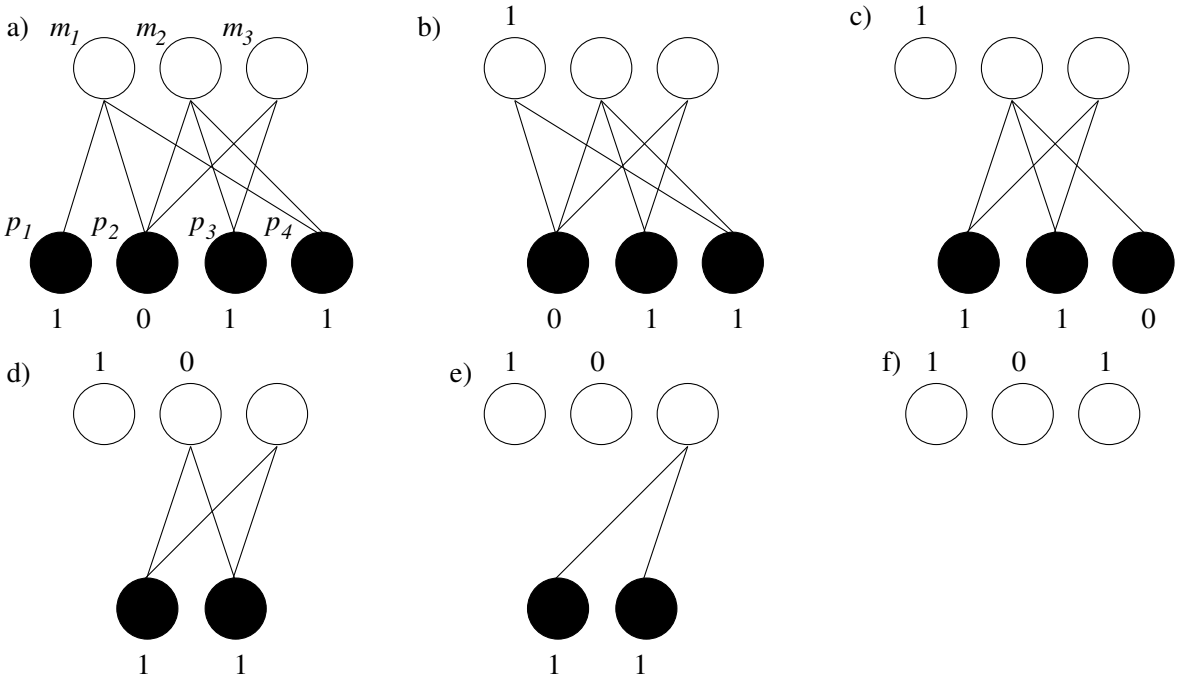

Figure 3.4: BP decoding of a simple LT code on an erasure channel

3.4.3 NOISY DECODER

To decode an LT code from a noisy channel it is important to construct the factor graph in the notation of Chapter 2. The code from Figure 3.4 can also be drawn as in Figure 3.5.

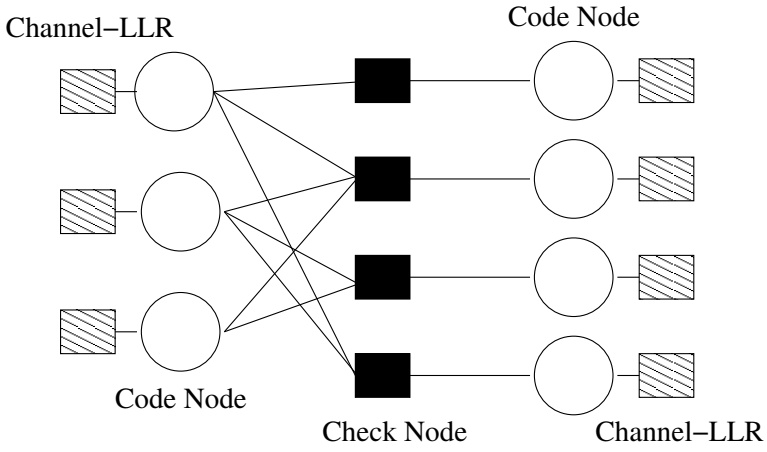

Figure 3.5: A simple factor graph of an LT code

Section 2.8.5 shows how to decode any factor graph (see that same section for an example). For clarity the algorithm will be given here using simpler and clearer equations. A code node usually represents a received bit and is assigned a channel LLR $\Lambda(m_n)$ (see equation (2.57)).

$$\Lambda(m_n) = -\frac{4}{N_0}\alpha_n r_n \tag{3.10}$$

where r_n is the soft value of the n^{th} received bit, α_n is the average fading amplitude of r_n and N_0 is the single-sided noise power spectral density.

As can be seen from Figure 3.5 an LT factor graph consists of two node types, check and code nodes. Each of these types of nodes use different formula's to update its branches. The input LLR's of a code node are summed together to produce the output LLR (including channel LLR), and is described as [22, 26, 23, 46]:

$$\Lambda(m_o) = \sum_{i \neq o} \Lambda(m_i) + \Lambda(m_n) \tag{3.11}$$

For a check node

$$\Lambda(m_o) = 2\tanh^{-1}\left[\prod_{i \neq o} \tanh(0.5\Lambda(m_i))\right] \tag{3.12}$$

is used. Take note that the output branch is not used in the calculation. To update a node each branch of a node needs to be updated with the above formula's. All branches are usually set to zero for the first iteration.

The order in which the nodes are updated can be chosen uniquely for each factor graph. Standard LDPC iterative belief propagation involves updating the code nodes first and then the check nodes. After such an iteration has been completed, the algorithm can stop or continue with another cycle. Usually the algorithm stops if a codeword is found or a certain predetermined amount of cycles are reached. See Sections 2.8.5 and 2.8.4 for exit calculations (determining the decoded bit values).

What is clear from the above is that this algorithm will work better on systematic than non-systematic codes. If the code is non-systematic the message nodes will all have initial LLR values equal to 0. The 0 values will hamper efficient belief propagation, making almost all messages equal to zero.

3.4.4 DEGREE DISTRIBUTION

The design of a good degree distribution of an LT code for an erasure channel is closely related to the classical problem of throwing k balls into K bins [40, 41]. From classical probability theory one knows that $k = K \cdot \ln(K/\delta)$ balls are required to cover each of

the K bins with probability at least $1 - \delta$. When performing the same analysis on a LT code the balls are analogous to the edges (in the factor graph) and the bins are the same as the message bits. The amount of edges required makes sense, all of the message bits must at least be connected to the graph to obtain successful decoding. The LT decoding process will only work if at the end all message bits are covered (meaning that the decoding process from Section 3.4.2 will not halt). This will only occur when at least one parity bit has degree one after each iteration [40]. Using the above it is easy to see that a good LT code will have $O(K \cdot \ln(K/\delta))$ edges, average left degree of $O(\ln(K/\delta))$ and encoding and decoding times of $O(K \cdot \ln(K/\delta))$. Take note that in this section k represents the amount of edges in a graph and K the amount of message nodes, this is done since in classic LT notation the amount of message bits is always referred to with a capital letter K.
The ideal soliton distribution [40],

$$\rho(d) = \begin{cases} 1/K & \text{for } d = 1 \\ \frac{1}{d(d-1)} & \text{for } d = 2, 3, \cdots, K \end{cases} \tag{3.13}$$

conforms to the above requirements. One of its primary characteristics is that after every iteration of the decoding process (from Section 3.4.2) exactly one parity bit has degree one. This however works poorly in practice since minor fluctuations in decoding will lead to termination of the decoding prematurely [40].
The robust soliton distribution has two extra parameters c and δ; it is designed to keep the expected number of degree-one parity bits equal to approximately:

$$S \equiv c \ln(K/\delta) \cdot \sqrt{K} \tag{3.14}$$

rather than one, throughout the decoding process. The parameter δ is a bound on the probability that the decoding fails to run to completion after a certain number K' of bits were received. The parameter c is a constant of order one. A positive function can now be defined [40]

$$\tau(d) = \begin{cases} \frac{S}{Kd} & \text{for } d = 1, 2, \cdots, (K/S) - 1 \\ \frac{S}{K} \ln(S/\delta) & \text{for } d = K/S \\ 0 & \text{for } d > K/S \end{cases} \tag{3.15}$$

To obtain the robust soliton distribution μ [40], ρ and τ are added and normalized:

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z} \tag{3.16}$$

where $Z = \sum_d \rho(d) + \tau(d)$. The number of encoded bits required at the receiving end to ensure that the decoding runs to completion, with probability at least 1 - δ, is $K' = KZ$.

Some degrees have such low probabilities p_d that parity bits of degree d are absent. Instead these probabilities can be used to reinforce the amount of degree-one parity bits. This is done by introducing an extra factor [48]

$$v(d) = \begin{cases} \sum \mu(d_i) & \text{for } d = 1 \\ 0 & \text{for } d > 1 \end{cases} \tag{3.17}$$

where d_i represents the degree-i term of the distribution, satisfying the following inequalities

$$\begin{cases} \frac{\frac{1}{d_i(d_i-1)} + \frac{S}{Kd_i}}{Z} K < 1 & \text{for } 2 \leq d_i \leq \frac{K}{S} - 1 \\ \frac{1}{d_i(d_i-1)} \cdot \frac{K}{Z} < 1 & \text{for } (\frac{K}{S} + 1) \leq d_i \leq K \end{cases} \tag{3.18}$$

The improved robust soliton distribution $\kappa(d)$ [48], can now be defined as

$$\kappa(d) = \frac{\rho(d) + \tau(d) + v(d)}{Z} \tag{3.19}$$

where $Z = \sum_d \rho(d) + \tau(d) + v(d)$. The distribution $\kappa(d)$ can be seen in Figure 3.6 for $K = 10000$ with $\delta = 0.05$ and $c = 0.2$ (example of a good LT code).

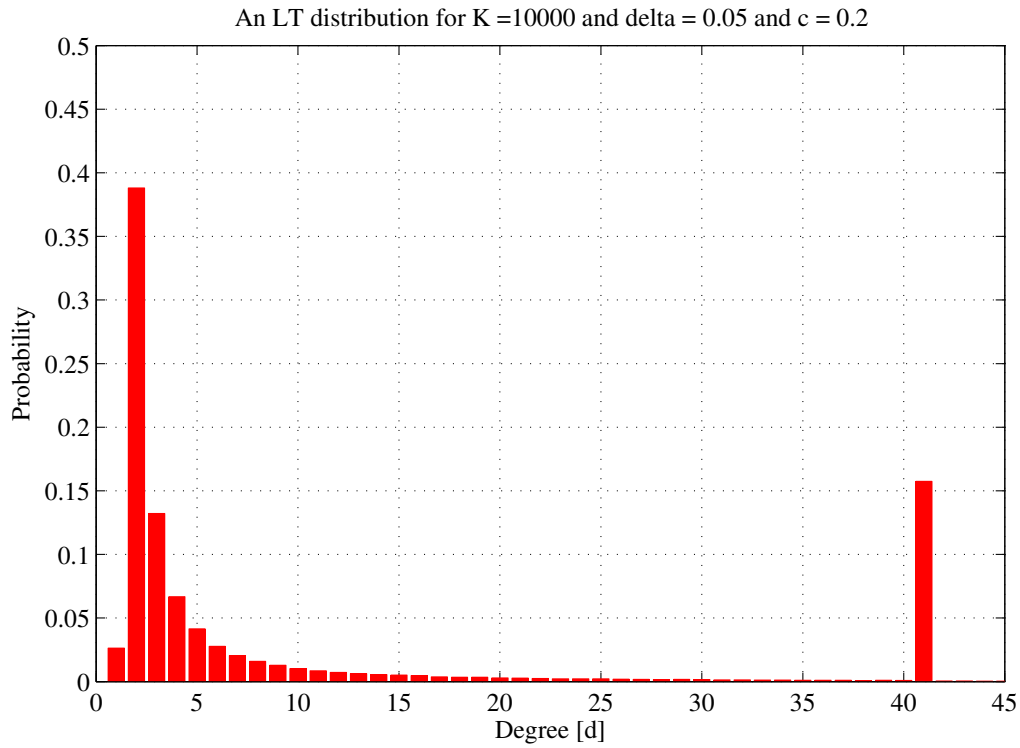

Figure 3.6: The improved robust soliton distribution $\kappa(d)$ for $K = 10000$ with $\delta = 0.05$; $c = 0.2$

3.5 RAPTOR CODES

The main idea behind Raptor codes [42] is to use a pre-code $\mathcal{C}$ to lower the average degree of each encoding symbol to a constant. This makes the decoding time needed proportional to k. A Raptor codeword of length n is constructed from a message of length k by first encoding it with $\mathcal{C}$ to produce an intermediate codeword of size $\tilde{k}$. The intermediate codeword is encoded by a weakened LT code to produce the required codeword of length n. The reason for the faster execution of the encoding and decoding lies in the fact that the pre-code lifts the restriction of the average degree of each encoding symbol. The average degree of each encoding symbol no longer needs to be $O(\ln \tilde{k})$; it may be lowered to a constant. The pre-code will correct all the erasures the LT code failed to correct (due to the fact that it will not be able to cover all the input symbols). Since the pre-code and the weakened LT code can be encoded and decoded in time proportional to k, the entire codeword can be decoded in time proportional to

k ($O(k\ln(1/\epsilon))$). The decoding is performed in exactly the same manner as in Section 3.4.2; the only difference is the algorithm is first deployed on the weakened LT code. If the LT code terminates, the pre-code is activated to decode the remaining message bits that the LT code failed to correct. Since it has been shown in Section 3.4.3 that noisy decoding can be applied to any factor graph it can also be applied to Raptor codes. The design of a Raptor code, for the erasure channel, comes down to the design of an efficient pre-code $\mathcal{C}$ and a weakened LT code $\mathcal{C}_{LT}$. A popular choice for the pre-code is a right regular LDPC code [42]. Figure 3.7 illustrates a popular design of Raptor codes [42].

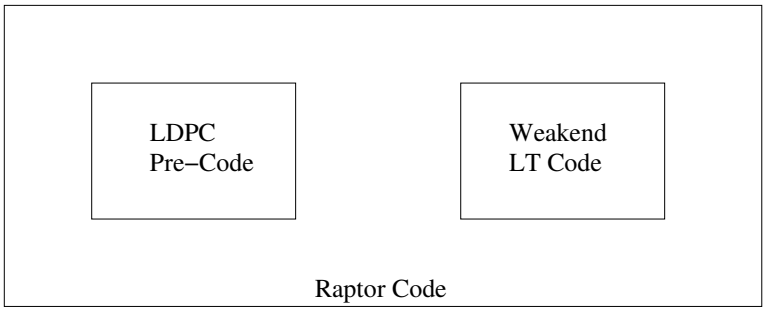

Figure 3.7: A popular design of a Raptor code

The pre-code need not be an LDPC code and can be many codes; even a Tornado code can be used [42]. What is interesting to note is that $\mathcal{C}_{LT}$ also requires more than $\tilde{k}$ symbols in order to be to decoded (it also has a decoding inefficiency). A Raptor code is also rate less since it uses an LT code.

3.5.1 A GOOD PRE-CODE

An LDPC code can be decoded in exactly the same manner as explained in Section 3.4.2 as long as the code contains a minimum amount of stopping sets. A stopping set is a set of code nodes such that their induced graph has the property that all the check nodes have degree greater than one. Meaning the decoding will fail since there are no more degree one check nodes left. Since the union of two stopping sets is again a stopping set, a bipartite graph contains a unique maximal stopping set (which may be the empty set).

The LDPC codes considered here can be described as the ensemble of graphs denoted by $\mathbf{P}(\Lambda(x), n, r)$, where Λ refers to the right degree distribution, n is the length of the codeword and r is the amount of check nodes used by the LDPC code [42]. The

probability bound for a maximal stopping set of size s for $\mathbf{P}(\Lambda(x), n, r)$ is calculated in the following paragraph.

Let r be a positive integer. For $n \geq 1$, z, $o \in \mathbb{Z}$, and $d \geq 1$ let $A_n(z, o)$ be recursively defined by

$$\begin{aligned} A_0(r,0) &:= 1 \\ A_0(z,o) &:= 0, \qquad \text{for } (z,o) \neq (r,0) \\ A_{n+1}(z,o) &:= \sum_{l,k} A_n(l,k) \cdot \\ &\quad \sum_d \Lambda_d \frac{\binom{l}{l-z}\binom{k}{k+l-z-o}\binom{r-l-k}{d-k-2l+2z+o}}{\binom{r}{d}}, \\ &\quad \text{for } n \geq 0 \end{aligned} \tag{3.20}$$

then the upper bound for the probability that the random graph $\mathcal{G}$, from the ensemble $\mathbf{P}(\Lambda(x), n, r)$, has a maximal stopping set of size s can be calculated by using the following [42]

$$\binom{n}{s} \sum_{z=0}^{r} A_s(z,0) \left(1 - \sum_d \Lambda_d \frac{\binom{r-z}{d}}{\binom{r}{d}}\right)^{n-s} \tag{3.21}$$

Since the maximal stopping set size s determines the fraction of bits a code can correct effectively, s can be used to design a LDPC code. If the probability of s is high then decoding of $\mathcal{C}$ will fail regularly, if at least s bits were erased. The art of designing a good LDPC pre-code is to make the probability of $s = \tilde{k}\delta$ as small as possible, where δ is the fraction of unrecovered input symbols of the weakened LT code.

A good LDPC code designed by using equation (3.21) is the right regular code with $\Lambda(x) = x^4$ [42]. It is also advised to use an extended Hamming code before the right regular code, to remove small stopping sets [42].

3.5.2 THE WEAKENED LT CODE

A weakened LT code is designed by using the following inequality [42]

$$\Omega'(x) \geq \frac{-\ln\left(1 - x - c\sqrt{\frac{1-x}{\tilde{k}}}\right)}{1+\epsilon} \tag{3.22}$$

for $x \in [0, 1-\delta]$ and $\Omega(x)$ is the degree distribution of the right hand side of the weakened LT code $\mathcal{C}_{LT}$, $\tilde{k}$ is the amount of input symbols, c is a positive constant, δ is

the fraction of input symbols that can not be recovered by $\mathcal{C}_{LT}$ and $1+\epsilon$ is the decoding inefficiency of $\mathcal{C}_{LT}$. For equation (3.22) to be solvable, δ needs to be larger than $c/\sqrt{\tilde{k}}$. Given δ, ϵ, c and $\tilde{k}$ the interval $[0, 1-\delta]$ can be discretized and the above inequality can be required to hold on these discretization points. From these linear inequalities the unknown coefficients of $\Omega(x)$ are obtained. Linear programming can now be used to solve the minimization problem created by the objective function $\Omega'(1)$ to obtain a degree distribution with minimum possible average degree (using the simplex method [47]).

When a Raptor code is designed for the erasure channel it is easy to see that the pre-code needs to be systematic, and the weakened LT code non-systematic. For a noisy channel both codes need to be systematic for the same reason as discussed in Section 3.4.3.

3.5.3 FACTOR GRAPH REPRESENTATION

A factor graph representation of a Raptor code is given in Figure 3.8. This is not the only possible factor graph for a Raptor code, but it is a popular design.

The white circles represent the original codeword code nodes of length k. The circles filled with a horizontal pattern represent the parity bits generated by the pre-code $\mathcal{C}$ (they are also code nodes). The black circles represent the parity bits generated by the weakened LT code $\mathcal{C}_{LT}$ (they are also code nodes). All the circles together form all the code nodes of a systematic Raptor code. In the non-systematic case only the bits represented by the black circles are sent over the erasure channel. The horizontally filled check nodes represent the check nodes of the pre-code. The black checks represent the check nodes of the LT code. As mentioned before noisy decoding can easily be applied to this factor graph, since any factor graph can be decoded using the algorithm given in Section 3.4.3. The vertically filled function blocks contain the channel LLR as defined by equation (3.10). Designing Raptor codes for non-erasure channels is discussed in [43].

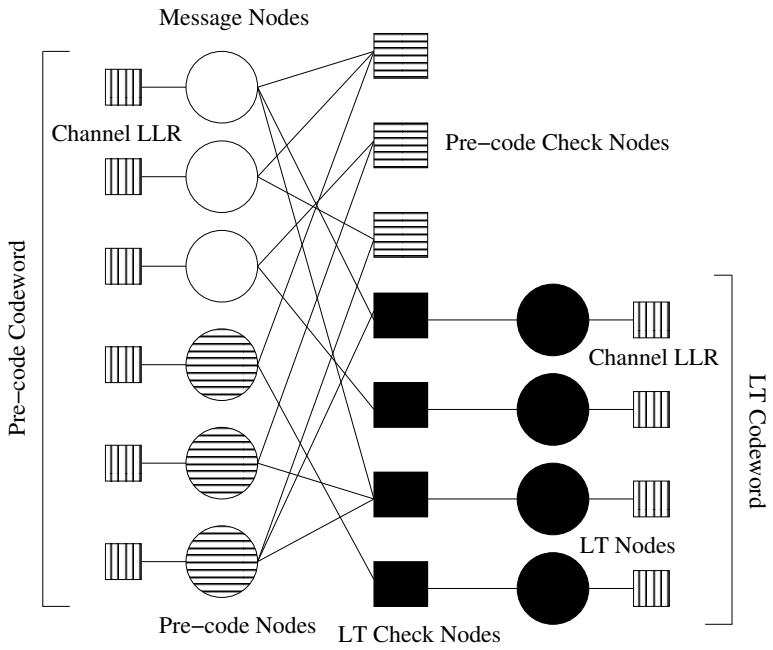

Figure 3.8: A simple factor graph of a Raptor code with an LDPC code as pre-code

CHAPTER 4

ENHANCED DATA RATES FOR GSM EVOLUTION (EDGE)

4.1 CHAPTER INTRODUCTION

The digital mobile phone technology Enhanced Data rates for GSM Evolution (EDGE) is discussed briefly in this chapter. The chapter starts with a general overview of EDGE. The overview includes the coding schemes, data rates obtainable and other techniques used by the standard. The next section gives the structure of the transmission (TX) blocks used by EDGE for each coding scheme. The information in the transmission section include data payload sizes as well as puncturing rates (again for each coding scheme). Next the chapter introduces the physical convolutional encoder used by EDGE. Each coding scheme uses the same encoder, but the puncturing schemes differ. The different puncturing schemes are also discussed. Lastly the use of punctured convolutional codes as incremental redundancy scheme is explained in detail. The use of fountain codes as alternative incremental redundancy scheme is also discussed in the last section.

4.2 OVERVIEW OF EDGE

EDGE is a digital mobile phone technology that allows increased data transmission rates and improved data transmission reliability when compared to older technologies like GPRS (General Packet Radio Service) available to users of GSM (Global System for Mobile communications). Although EDGE is technically a 3G network technology, it is generally classified as the unofficial standard 2.75G, due to its slower network speed. EDGE helps in bridging the gap between 2G and 3G, and enables the use of multimedia services (it is more a transition technology) [49]. EDGE uses a bandwidth of 270.833 kHz [50] and uses nine modulation and coding schemes (MCS) to vary its data rates. Gaussian minimum-shift keying (GMSK) [51] is used by the lower four coding

schemes (similar to GPRS) and 8 phase shift keying (8PSK) [51] for the upper five of its nine coding schemes. The fact that 8PSK can modulate 3 bits per symbol is the reason for the increase in data rate (when compared with GMSK that can modulate only 1 bit per symbol). One frame block (1392 bits) is transmitted over 4 bursts in a total time of 20 ms making the bit rate 278.4 kbit/s and yielding a bit rate of 69.6 kbit/s per time slot (compared with the current 23.2 kbit/s (GPRS)) (this is true for uncoded using no headers) . The receiver is now more complex and employs sophisticated equalization techniques. The actual maximum bit rate of EDGE is achieved by MCS9 and equals 59.2 kbit/s, the difference is due to EDGE implementation issues like headers and parity bits [24]. The improvement of 8PSK modulation is summarized in Table 4.1 [49]

Table 4.1: Comparison between EDGE and GPRS

	EDGE	**GPRS**
Modulation	8PSK, 3 bits per symbol	GMSK, 1 bit per symbol
Payload/burst	348 bits	116 bits
Rate/time slot	69.6 kbit/s	23.2 kbit/s

EDGE also uses Link Adaptation (LA) [49, 52], which adapts the MCS scheme to fit the channel conditions. Otherwise one uses a too strong or weak code for the channel, hampering throughput. EDGE introduces Incremental Redundancy (IR) [49, 25], which sends additional redundancy information to the receiver after a decoding failure. Both transmissions can now be combined for better decoding. EDGE combines these two techniques to maximize throughput. Each MCS scheme has different throughput rates to enable the use of LA. The weaker the channel the stronger the coding scheme, the less the throughput. The speed of all nine MCS schemes is given in Table 4.2 [50, 53]. The code rates obtainable by each MCS scheme when employing IR is given in Table 4.3. The * indicates that some bits have been sent more than once.

Only the upper five coding schemes are discussed in greater detail in the remainder of this chapter since they attribute to a higher bit rate. The data is encoded by a punctured convolutional code [51, 30] which can be decoded using the Viterbi algorithm [11].

Table 4.2: Comparison between the EDGE modulation and coding schemes (MCS)

MCS	Speed	Modulation
MCS1	8.8 kbit/s	GMSK
MCS2	11.2 kbit/s	GMSK
MCS3	14.8 kbit/s	GMSK
MCS4	17.6 kbit/s	GMSK
MCS5	22.4 kbit/s	8PSK
MCS6	29.6 kbit/s	8PSK
MCS7	44.8 kbit/s	8PSK
MCS8	54.4 kbit/s	8PSK
MCS9	59.2 kbit/s	8PSK

Table 4.3: Code rates obtainable with convolutional code

MCS Scheme	Transmission 1	Transmission 2	Transmission 3
MCS 5	0.37	0.33*	-
MCS 6	0.49	0.33*	-
MCS 7	0.76	0.38	0.33*
MCS 8	0.92	0.46	0.33*
MCS 9	1	0.5	0.33*

4.3 STRUCTURE: EDGE TX BLOCKS

As mentioned in Section 4.2 each transmission block of EDGE consists of 1392 bits and is sent over 4 bursts in a total time of 20 ms. The structure of each transmission block depends on the MCS scheme currently employed. MCS5 and MCS6 have only one data payload in each transmission block, while MCS7, MCS8, MCS9 have two data payloads contained in each transmission block. This is illustrated in Figures 4.1 and 4.2 [24].

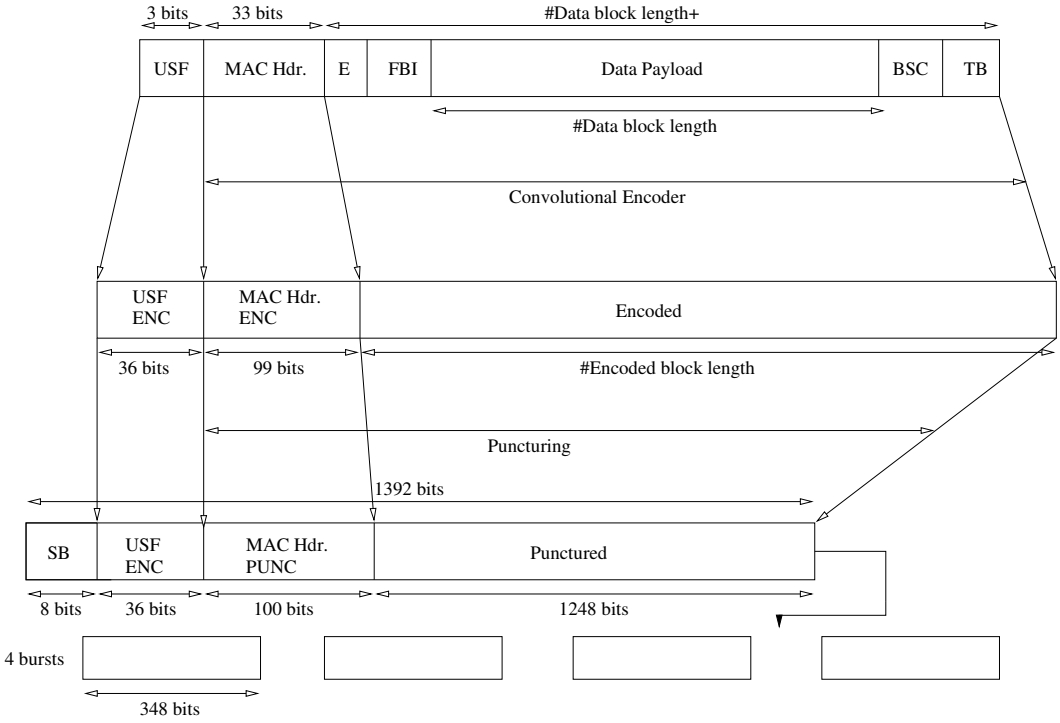

Figure 4.1: TX Block structures for MCS5 and MCS6

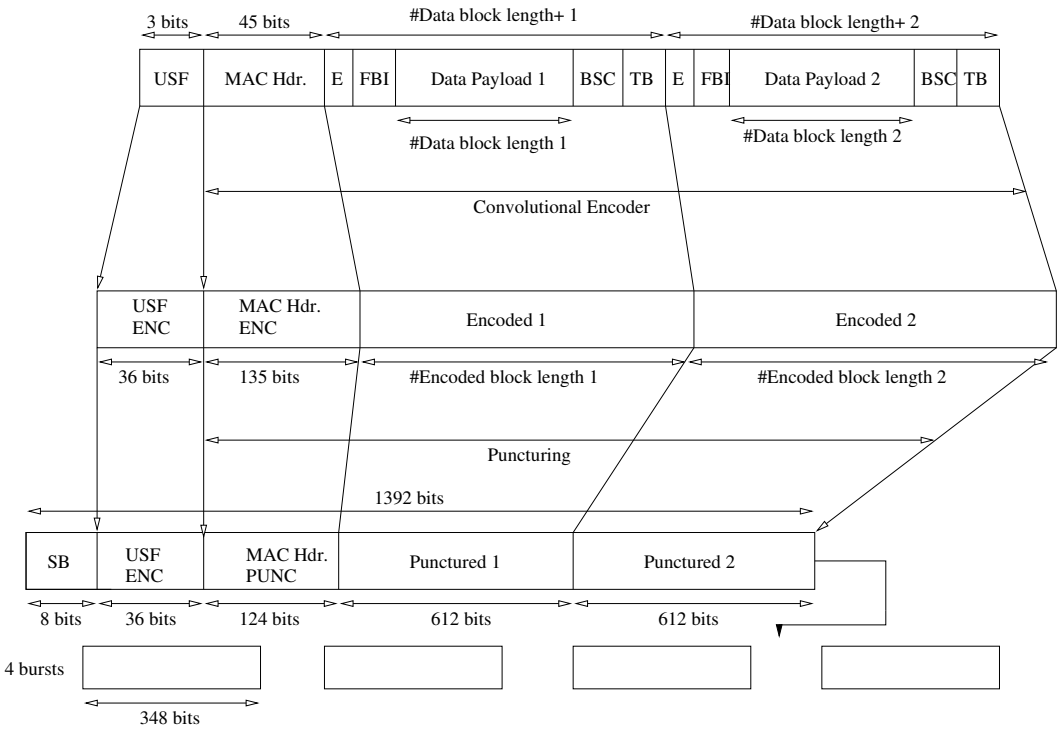

Figure 4.2: TX Block structures for MCS7, MCS8 and MCS9

The basic procedure in creating a transmission block [24] is that the E (Extension bit), FBI (Final Block Indicator), BCS (Block Check Sequence), TB (Tail bits), USF (Uplink State Flag) and MAC/RLC (Medium Access Control/Radio Link Control) header bits are appended to the data payload to form a MAC/RLC block. The length of the data payload and the MAC/RLC block depends on the MCS scheme used. The USF and MAC/RLC header are encoded separately. The remaining bits of the MAC/RLC block are encoded with a rate 1/3 convolutional encoder. The length of this encoded block also depends on the MCS scheme used. The encoded USF field is not punctured, but the other fields are punctured. In the case of MCS6 and MCS7 the encoded MAC/RLC header is only padded with a zero and not punctured. Finally SB (Stealing bits) are added to complete the 1392 bit transmission block. After puncturing the TX block is also interleaved. The above procedure for encoding the data payload is done twice for MCS7, MCS8 and MCS9, due to two data payloads. The decoding of a TX block is done by following the above steps in reverse [24]. The general procedure for creating a TX block is depicted in Figure 4.3. The sizes of each variable field is found in Tables 4.4 and 4.5 [24].

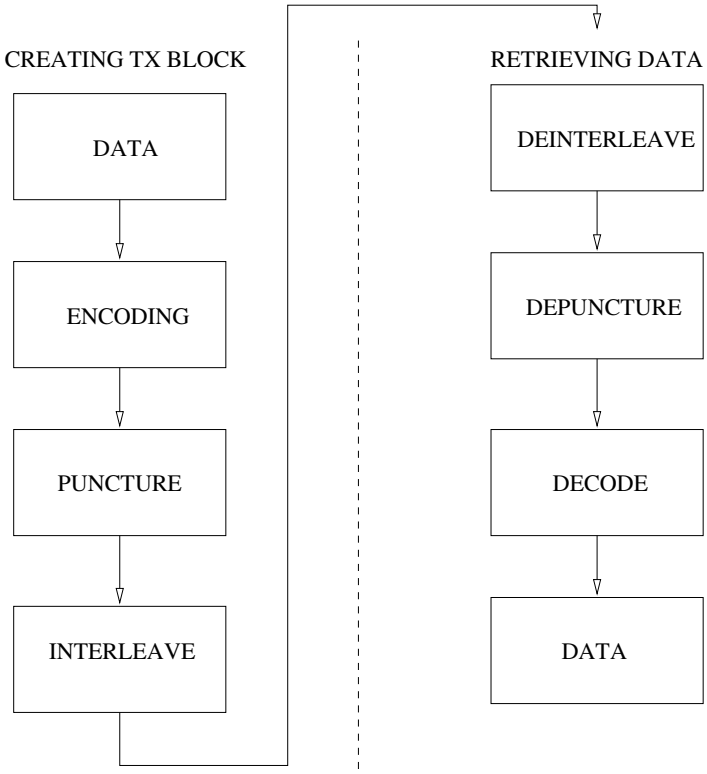

Figure 4.3: General procedure for creating a TX block

Table 4.4: Different field lengths considered in TX block of EDGE (MCS5 and MCS6)

	MCS5	MCS6
#Data block length	448	592
#Data block length+	468	612
#MAC/RLC block length	504	648
#Encoded block length	1404	1836

Table 4.5: Different field lengths considered in TX block of EDGE (MCS7, MCS8 and MCS9)

	MCS7	MCS8	MCS9
#Data block length 1&2	448	554	592
#Data block length+ 1&2	468	564	612
#MAC/RLC block length	984	1176	1272
#Encoded block length 1&2	1404	1692	1836

For detailed information on the function of each field, the interleaving scheme used and more detail see [24]. For the remainder of this chapter the focus will be on the convolutional encoder and its use by the IR scheme.

4.4 CONVOLUTIONAL ENCODER

The encoder used is a (3,1,7) convolutional encoder (constraint length 7) with generator polynomials [49, 24]:

$$\begin{aligned} G_0(D) &= 1 + D^2 + D^3 + D^5 + D^6 \\ G_1(D) &= 1 + D + D^2 + D^3 + D^6 \\ G_2(D) &= 1 + D + D^4 + D^6 \end{aligned} \tag{4.1}$$

The exact same encoder is used for each coding scheme, though the puncturing differs. The puncturing schemes for each MCS scheme are given in Tables 4.6, 4.7 and 4.8 [24].

Table 4.6: Puncture pattern 1 for each MCS scheme

MCS Scheme	Pattern 1
MCS5	$b_1 = [110111111]$ $b_2 = [110]$ $p_1 = \emptyset$ $p_1 = [p_1 b_1]$ for $k = 1 \cdots 154$ $p_1 = [p_1 b_2]$ for $i = 1 \cdots 6$ $p_1[48\ 372\ 696\ 1020] = 1$
MCS6	$b_1 = [110]$ $p_1 = \emptyset$ $p_1 = [p_1 b_1]$ for $k = 1 \cdots 612$ $p_1[33\ 99\ 165\ 231\ 297\ 429\ 495\ 561\ 627\ 693\ 825\ 891\ 957 \cdots$ $1023\ 1089\ 1221\ 1287\ 1353\ 1419\ 1485\ 1617\ 1683\ 1749\ 1815] = 1$
MCS7	$b_1 = [110010001001110100]$ $p_1 = \emptyset$ $p_1 = [p_1 b_1]$ for $k = 1 \cdots 78$ $p_1[2\ 20\ 38\ 236\ 416\ 596\ 776\ 956\ 1136\ 1352\ 1370\ 1388] = 0$
MCS8	$b_1 = [10100110001001001000100110010001 0001]$ $p_1 = \emptyset$ $p_1 = [p_1 b_1]$ for $k = 1 \cdots 47$ $p_1[846] = 1$
MCS9	$b_1 = [100]$ $p_1 = \emptyset$ $p_1 = [p_1 b_1]$ for $k = 1 \cdots 612$

Table 4.7: Puncture pattern 2 for each MCS scheme

MCS Scheme	Pattern 2
MCS5	$b_1 = [101111111]$ $b_2 = [101]$ $p_2 = \emptyset$ $p_2 = [p_2 b_1]$ for $k = 1 \cdots 154$ $p_2 = [p_2 b_2]$ for $i = 1 \cdots 6$ $p_2[137\ 461\ 785\ 1109] = 1$
MCS6	$b_1 = [101]$ $p_2 = \emptyset$ $p_2 = [p_2 b_1]$ for $k = 1 \cdots 612$ $p_2[17\ 83\ 149\ 215\ 281\ 413\ 479\ 545\ 611\ 677\ 809\ 875\ 941 \cdots$ $1007\ 1073\ 1205\ 1271\ 1337\ 1403\ 1469\ 1601\ 1667\ 1733\ 1799] = 1$
MCS7	$b_1 = [001101100010001011]$ $p_2 = \emptyset$ $p_2 = [p_2 b_1]$ for $k = 1 \cdots 78$ $p_2[17\ 35\ 53\ 197\ 377\ 557\ 737\ 917\ 1097\ 1367\ 1385\ 1403] = 0$
MCS8	$b_1 = [010010001001100101010010010010100100]$ $p_2 = \emptyset$ $p_2 = [p_2 b_1]$ for $k = 1 \cdots 47$ $p_2[583] = 1$
MCS9	$b_1 = [010]$ $p_2 = \emptyset$ $p_2 = [p_2 b_1]$ for $k = 1 \cdots 612$

Table 4.8: Puncture pattern 3 for each MCS scheme

MCS Scheme	Pattern 3
MCS5	-
MCS6	-
MCS7	$b_1 = [001001110100110010]$ $p_3 = \emptyset$ $p_3 = [p_3 b_1]$ for $k = 1 \cdots 78$ $p_3[14\ 32\ 50\ 302\ 482\ 662\ 842\ 1022\ 1202\ 1364\ 1382\ 1400] = 0$
MCS8	$b_1 = [00110001010000100110010000110100 1010]$ $p_3 = \emptyset$ $p_3 = [p_3 b_1]$ for $k = 1 \cdots 47$ $p_3[1157] = 1$
MCS9	$b_1 = [001]$ $p_3 = \emptyset$ $p_3 = [p_3 b_1]$ for $k = 1 \cdots 612$

4.5 INCREMENTAL REDUNDANCY

Incremental Redundancy (IR) [49, 25] works on the following principle. As soon as a received block fails to decode, the block is retransmitted using different parity. Now the information from both transmissions can be combined to decode the original message. Effectively the code rate is decreased after the second transmission, adding more parity to the transmitted codeword. This can continue for a third transmission depending on the coding scheme used.

IR is accomplished in EDGE by using a punctured convolutional code, see Figure 4.4. During the first transmission the data (1) is encoded by a rate 1/3 convolutional encoder and punctured using puncture pattern 1 (2). Since only one transmission was made the combination stage is skipped (3). If the decoding (4) fails the loop starts again. The second transmission punctures the encoded data by using pattern 2 (2). Due to differences in the puncturing patterns the receiver will be able to fill in some punctures of the first transmission. If some bits (in the same bit positions) were sent twice, they are averaged (since they are LLR) to obtain a new soft value for that bit. These calculations are performed at the combination stage (3). Decoding takes place

again, if decoding fails again the procedure needs to be repeated. The amount of cycles depend on the MCS scheme used. For MCS5 and 6, only two transmissions are possible. For MCS7 to 9 three transmissions are possible. If the maximum puncturing pattern is reached a decoding failure is declared (2). IR is illustrated for a toy case in Figure 4.5.

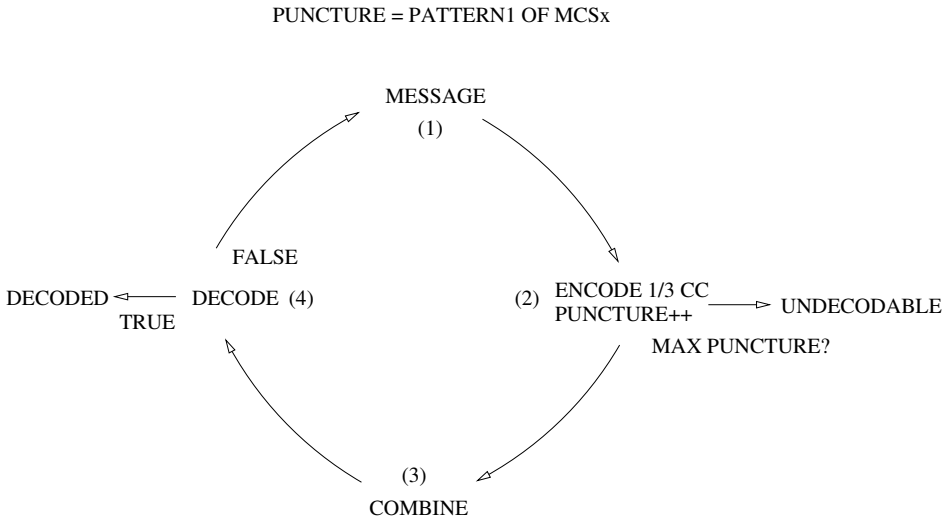

Figure 4.4: IR implemented with punctured convolutional codes

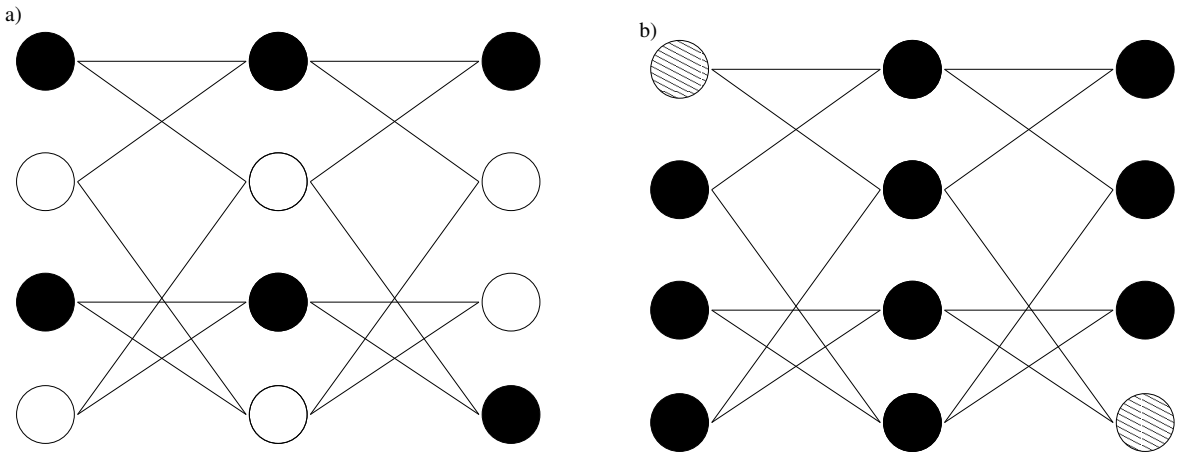

Figure 4.5: Segment of a Trellis displaying IR

In Figure 4.5(a) the white circles represent the bits that were punctured and the black circles represent the bits that were sent during transmission 1. The trellis in Figure 4.5(b) represent what happened to Figure 4.5(a) after a second transmission was made. As one can see all the bits have values now because the puncture patterns of both

transmissions were different. Some bits were sent twice; these bits are indicated by the circles containing patterns.

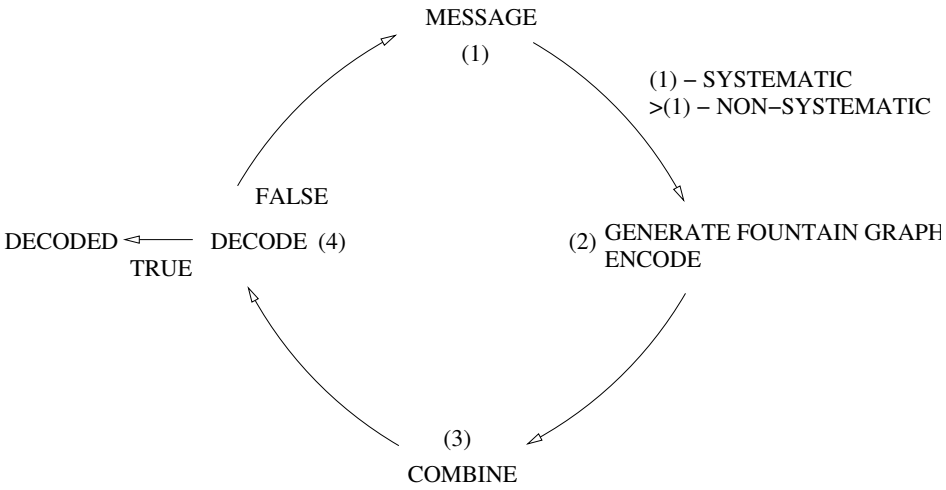

Figure 4.6: IR implemented with fountain codes

This is however not the only approach that can be followed. EDGE can also use a fountain code to implement an IR scheme (see Figure 4.6). Raptor or LT codes can be used to replace the puncturing scheme [26], e.g. a message can be encoded using a systematic LT code [23] (2) and if the frame at the decoder (4) is invalid the transmitter retransmits the message using a different non-systematic LT structure (2). Now the receiver can use both transmissions for successful decoding (3). The cycle can be continued until a valid frame is received. In contrast to punctured convolutional codes, fountain codes have the ability to utilize all the retransmitted information for decoding and can have an endless amount of retransmissions. The reason being it grows dynamically (rate less), and does not fill up punctures. A Raptor code can also be used, with the first transmission being a pre-code, such as an LDPC code. From there on the scheme is exactly the same. A systematic fountain code is used on the first transmission since noisy decoding of non-systematic codes performs poorly. For the second transmission a non-systematic LT code (only parity) is used, because the graphs can be combined at the receiver (BP decoding [22, 26, 23, 46], see Section 3.4.2). No puncturing is required for this scheme. The above is illustrated for a toy case in Figure 4.7.
In Figure 4.7 the white circles represent the bits sent. In Figure 4.7(a) we have the factor graph after the first transmission, in Figure 4.7(b) we have the factor graph after the second transmission. As one can see the graph grows dynamically, decreasing the code rate. Since fountain codes never have retransmissions it can be a very powerful

code to use as an IR scheme.

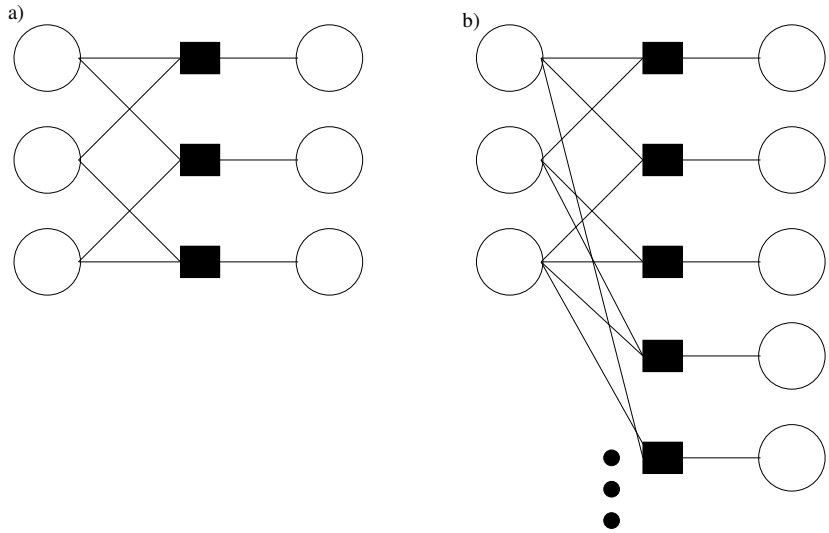

Figure 4.7: Segment of a Factor graph (fountain code) displaying IR

The code rates obtainable by using these schemes to implement IR are indicated in Tables 4.3 and 4.9.

Table 4.9: Code rates obtainable with fountain code

MCS Scheme	Transmission 1	Transmission 2	Transmission 3
MCS 5	0.37	0.19	-
MCS 6	0.49	0.24	-
MCS 7	0.76	0.38	0.25
MCS 8	0.92	0.46	0.31
MCS 9	1	0.5	0.33

CHAPTER 5

SIMULATION RESULTS

5.1 CHAPTER INTRODUCTION

This chapter contains all the results from the simulations performed for this thesis. The simulations can be divided into two main sections. Simulations were performed on a i) standard QPSK platform and on the ii) EDGE platform. The effect of the different fountain factor graphs were evaluated on the QPSK platform, while fountain codes were used as the incremental redundancy scheme of choice on the EDGE platform. Some simple codes were also implemented on the QPSK platform.

Note the importance of the QPSK simulations, even though the main contribution of this thesis is the implementation of fountain codes as incremental redundancy scheme. The QPSK platform was used in the first case to test some simple codes. The testing of simple codes were performed to verify the working of the belief propagation algorithm as presented in Chapter 2 and to illustrate the wider application of belief propagation on any factor graph representation of any code whether block or convolutional (depending on block size and density), which helps in understanding how belief propagation can be applied to the strange factor graph representations of fountain codes. A feasibility study on fountain codes (when used on the QPSK channel) was also conducted so that the correct fountain code could be selected as incremental redundancy scheme. The results of the noisy decoding of fountain codes were also included to show how these codes performed on noisy channels since this has not been completely discussed in literature. In short the QPSK simulations are included to show how belief propagation can be applied on a multitude of different factor graph structures.

5.2 SIMULATION: QPSK PLATFORM

In this section the QPSK platform is introduced, the performance of an uncoded QPSK platform is discussed and the performance of some simple codes are presented. Lastly the performance of different fountain codes are given.

5.2.1 QPSK PLATFORM

Most of the codes in this chapter were tested on a standard QPSK [51] communication system. A standard QPSK communication system is depicted in Figure 5.1.

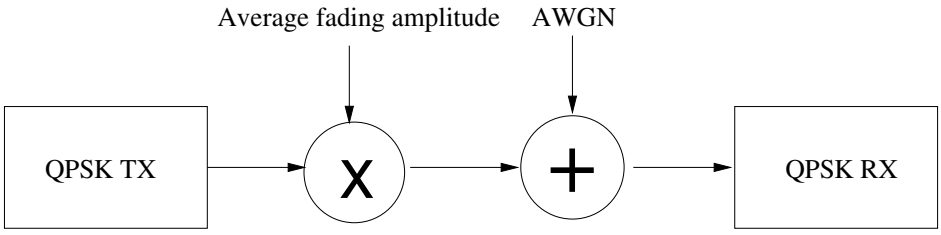

Figure 5.1: A simple QPSK communication system

In this channel the source output is modulated using QPSK modulation, sent over a flat fading channel and demodulated at the receiver. The channel consists of a product and a sum. First the signal is multiplied by the average fading amplitude and then AWGN is added. The channel is described as flat since its transfer function in the frequency domain is constant over the signal bandwidth. The noise that is added to the signal can be attributed to a variety of noise sources, including galactic noise (for example radiation), terrestrial noise, amplifier noise, interference from other communication systems, and last but not least, thermal noise caused by the motion of electrons in conducting media (the noise is Gaussian distributed with constant power spectral density (white)). Fading is due to the constructive/destructive interference of many electromagnetic waves (signals) at the receiver caused by reflection, diffraction and scattering of the original transmitted signal by objects in the vicinity of the transmission system. The fading can either be slow or fast (fade slower or more rapidly). This is determined by the maximum Doppler spread B_D of the channel. This phenomenon of fast and slow fading can be attributed to the Doppler effect (change in frequency due to relative movement between transmitter and receiver). In the case of the channel used in this chapter slower fading is simulated by setting $B_D = 33$ Hz, while faster fading is simulated by setting $B_D = 100$ Hz (this is only true for this simulator). Another important characteristic of the channel is the amount of line of sight (LOS) between the transmitter and receiver. This is determined by the Rician factor K of the channel. For no LOS (NLOS) $K = -100$ dB and for some LOS $K = 9$ dB. All codes are simulated for all combinations of these four parameters. These parameters completely describes the channel simulator for this specific case, but more detail is needed to approximate

a real channel (for example multipath delay spread and signal bandwidth). The implementation details of this channel is omitted and is described in detail in [21, 22]. The settings of the QPSK transmitter are shown in Table 5.1

Table 5.1: Settings of QPSK transmitter

Variable	Name	Value
f_c	Carrier frequency	2 kHz
f_s	Sampling frequency	16 kHz
f_b	Bit rate	1 kbps

The different channel parameters that describe the flat fading channel is summarized in Table 5.2

Table 5.2: Flat fading channel parameters

Variable	Values	Reason
C_{norm}	4.76(100Hz) and 8.27(33Hz)	Ensure unity power of filtered output signal
B_D	33Hz and 100Hz	Simulates slow and fast fading respectively
K	-100dB and 9 dB	Simulates NLOS and LOS respectively

The variable C_{norm} is a scaling factor used by the flat fading channel to ensure unity power of the filtered output signal (used by the Doppler filters to create Doppler spread). For an in depth description of this variable see [21, 22]. The QPSK receiver assumes perfect carrier recovery, phase synchronization and channel state information. The demodulation approach used is matched filtering implemented by an integrating dump circuit [51, 21, 22].

5.2.2 THEORETICAL BER CURVES: UNCODED QPSK

Two theoretical BER curves for an uncoded QPSK transmission system are discussed. Firstly when sent through an AWGN only channel the theoretical BER output is equal to [51]:

$$P_b(e) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{5.1}$$

where $Q(x)$ equals

$$Q(x) = \int_x^\infty e^{-z^2/2} dz \tag{5.2}$$

Secondly the theoretical BER curve of a flat fading channel is (NLOS, slow fading) equal to [51]:

$$P_b(e) = 1/2 - 1/2\sqrt{\frac{E_b/N_0}{1 + E_b/N_0}} \tag{5.3}$$

Fading without any LOS is known as Rayleigh fading. These theoretical BER curves are drawn in Figure 5.2.

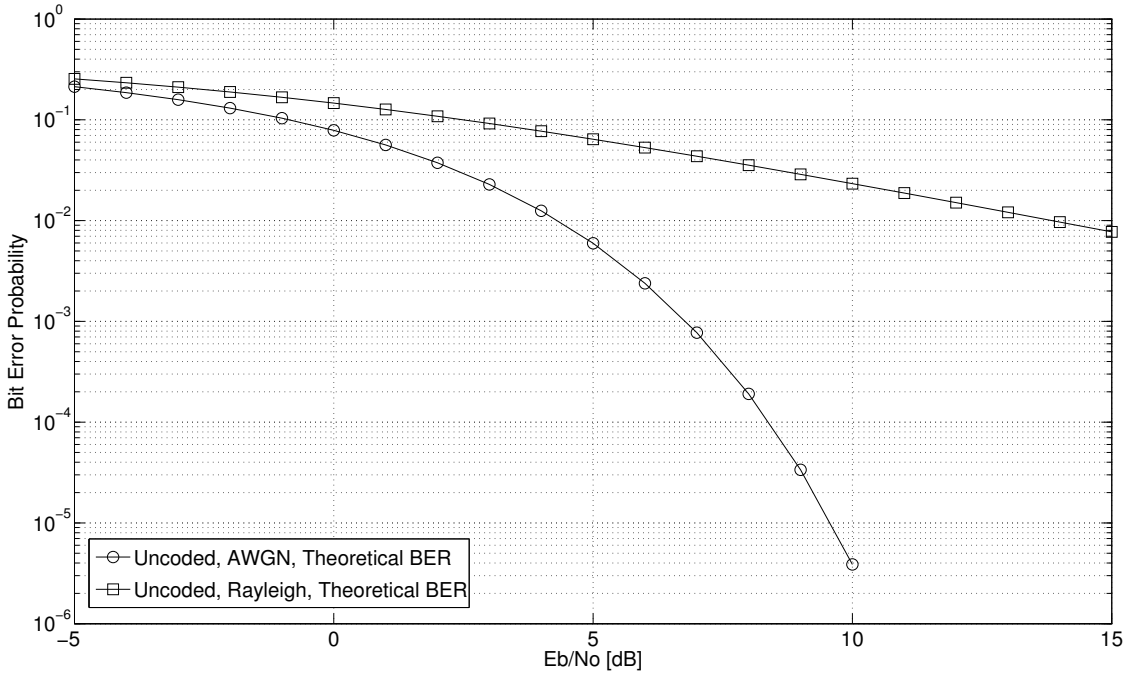

Figure 5.2: Theoretical BER performance of an uncoded QPSK communication system on a flat fading channel

5.2.3 SIMULATED BER CURVES: UNCODED QPSK

The simulated BER curves for all combinations of fading channel parameters for the uncoded QPSK system can be found in Figures 5.3 ($B_D = 33$Hz and $K = -100$dB,9dB) and 5.4 ($B_D = 100$Hz and $K = -100$dB,9dB). In Figure 5.5 the uncoded QPSK system is simulated for AWGN conditions only.

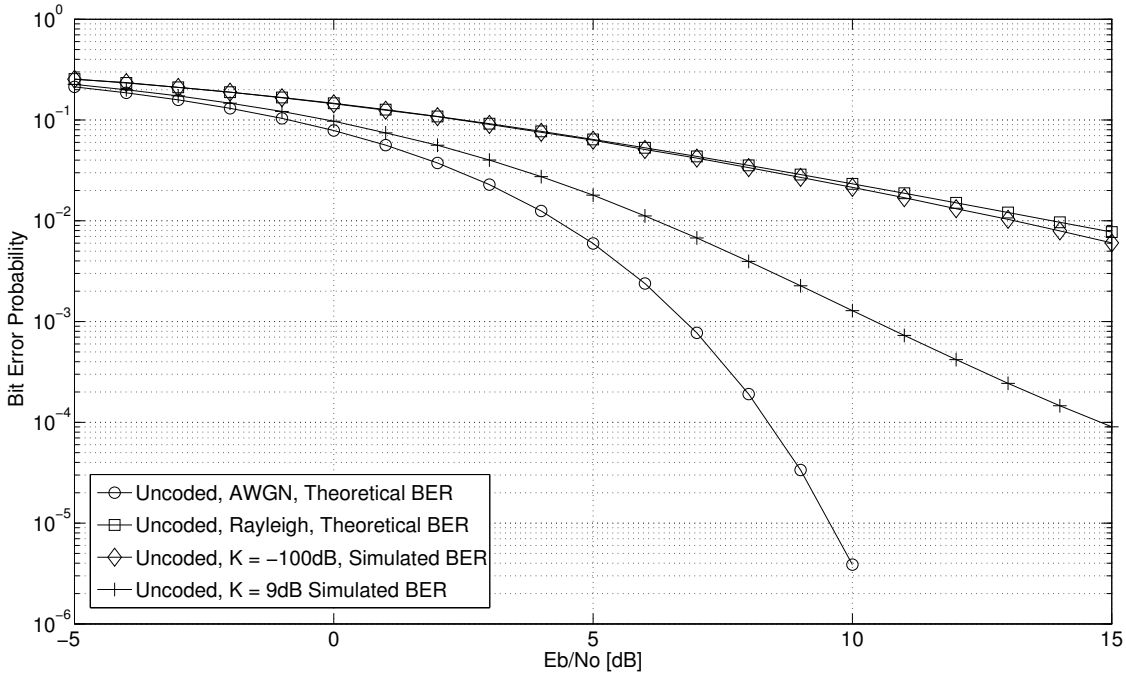

Figure 5.3: BER performance of an uncoded QPSK communication system on a flat fading channel, $B_D = 33Hz$

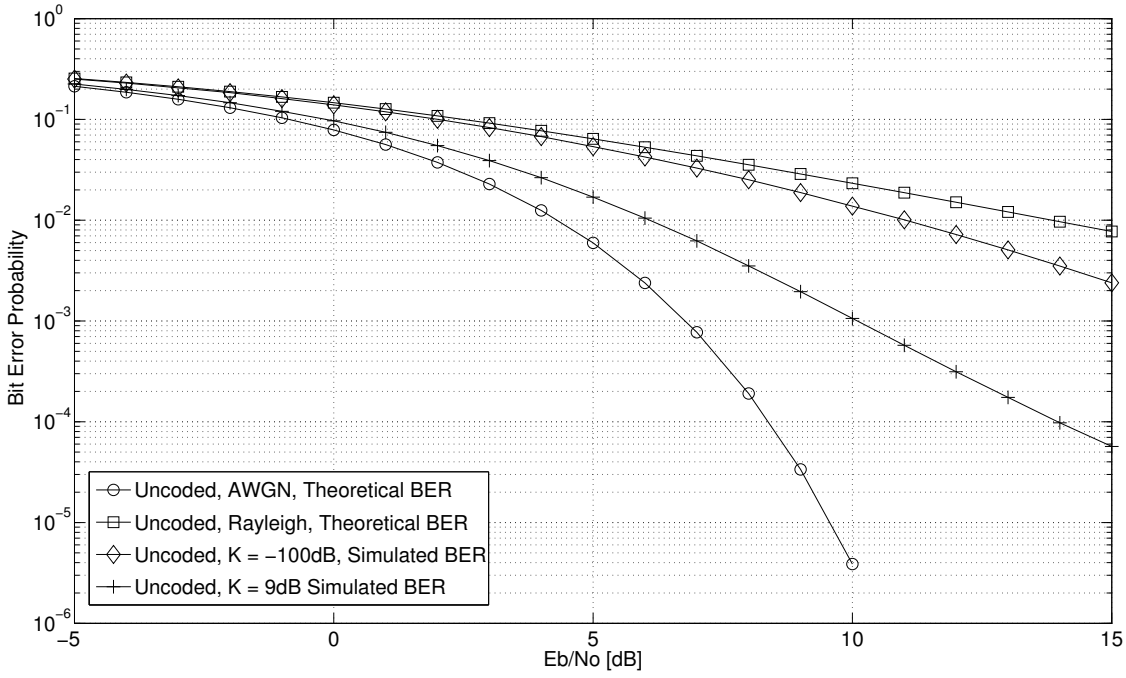

Figure 5.4: BER performance of an uncoded QPSK communication system on a flat fading channel, $B_D = 100Hz$

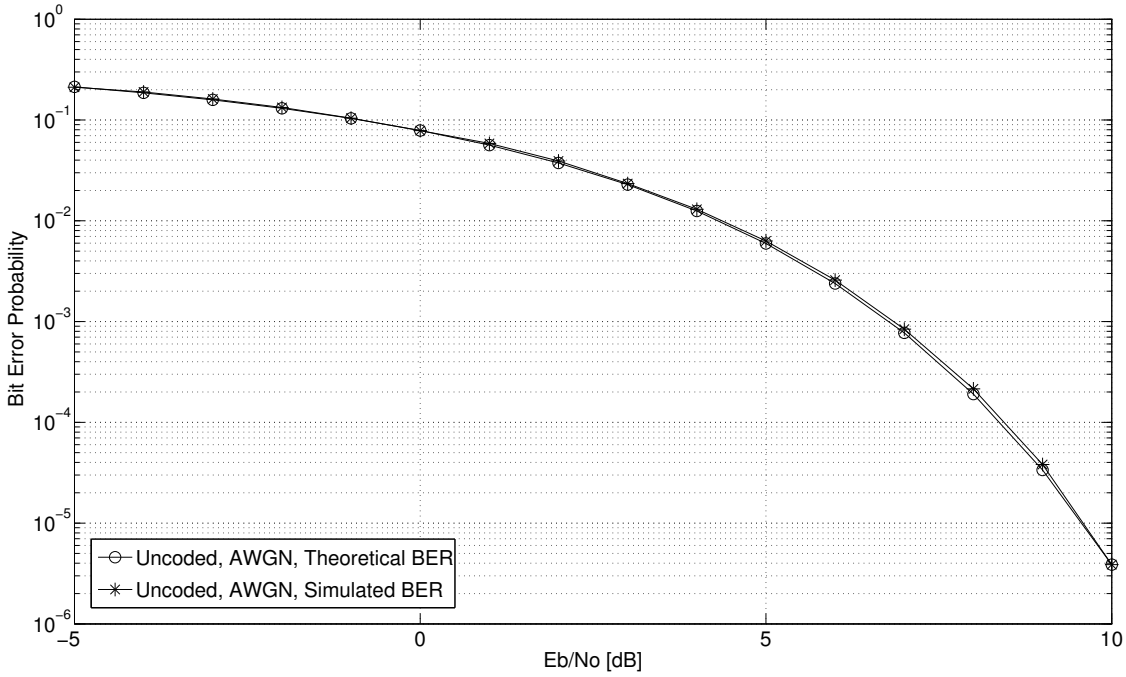

Figure 5.5: BER performance of an uncoded QPSK communication system on a AWGN channel

These results can be compared with [21] to verify their correctness. As shown in Figures 5.3 and 5.4 the simulated fading BER curves perform better than the theoretical upper bound due to the correct implementation of Doppler spread and LOS. As these factors can determine the severity of the fading a signal experiences. If more LOS is added the BER curve improves as expected. The simulated fading curves can not do worse than the theoretical upper bound since this curve represents the worst possible fading a signal can experience, with NLOS as well as the fact that the fading that is experienced is extremely slow.

5.2.4 SIMPLE CODES

In this section some simple channel codes are introduced. Channel coding is used to decrease the bit error rate of a transmission system by adding parity to a message. An encoder is placed in front of the transmitter, and a decoder is placed after the receiver in Figure 5.1. The encoder is usually implemented by matrix multiplication or shift registers. The decoder can consist of different decoding techniques including Viterbi [11] and BP [30, 14]. The codes mentioned below were tested on the QPSK transmission system to show how BP can be applied to different codes (having diverse factor graph

representations). The codes tested for this purpose include a simple Hamming, RSC and LDPC code. All simulations only perform BP decoding, 10 iterations of BP. The resulting BER graphs of these simple codes are given in Section 5.2.5 and discussed in Section 5.2.6.

5.2.4.1 HAMMING (7,4,3)

The Hamming (7,4,3) [2, 51, 30] code is a block code and has a **G** and **H** matrix equal to equation (2.52) and equation (2.51) respectively. Block codes are presented using the notation (n, k, d_{min}), where n represents the length of a code word, k is the length of the original message and d_{min} is the minimum hamming weight between any two codewords. Encoding is accomplished by multiplying the message with **G**. From the **G** and **H** matrix one can construct a factor graph as done in Section 2.8.3. From the factor graph the code can be decoded using BP as shown in Section 2.8.5. Figures 5.7 and 5.8 present the simulated BER results obtained when decoding Hamming (7,4,3) with BP under slow and fast fading channel conditions respectively. On each of these graphs the simulated LOS and NLOS results are displayed. For completeness the simulated BER for AWGN channel conditions is shown in Figure 5.9. These graphs are discussed in detail in Section 5.2.6.

5.2.4.2 GALLAGER (20,5,6)

The Gallager (20,5,6) (the original LDPC code [7]) is a block code and has **G** and **H** matrices equal to

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{5.4}$$

$$\mathbf{H} = \begin{pmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix} \tag{5.5}$$

The encoding and decoding are done using the same principles as discussed for the Hamming code (since **G** and **H** are available). Figures 5.10 and 5.11 show the simulated BER results obtained when decoding Gallager (20,5,6) with BP under slow and fast fading channel conditions respectively. On each of these graphs the simulated LOS and NLOS results are displayed. For completeness the simulated BER for AWGN channel conditions is shown in Figure 5.12. These graphs are discussed in detail in Section 5.2.6.

5.2.4.3 RSC (2,1,2)

In general a CC (convolutional code) [15] can be encoded via linear shift registers (state machine). The amount of shift registers a CC uses is known as the constraint length K of the code. The memory of a CC is equal to $K + 1$. The state machine (encoder) works by receiving k information bits (symbol) and outputting n encoded bits after performing some modular (in the case of binary symbols) operations on the information symbol and the current register symbols. After the calculation the registers get shifted up (right), and in so doing changing the state of the state machine for the next information symbol leading to a code with memory. A CC is thus defined via the following parameters (n, k, K). The encoder of a systematic RSC (2,1,2) (1,5/7)

[51, 30] code is shown in Figure 5.6, where (1,5/7) refers to the way the registers are connected.

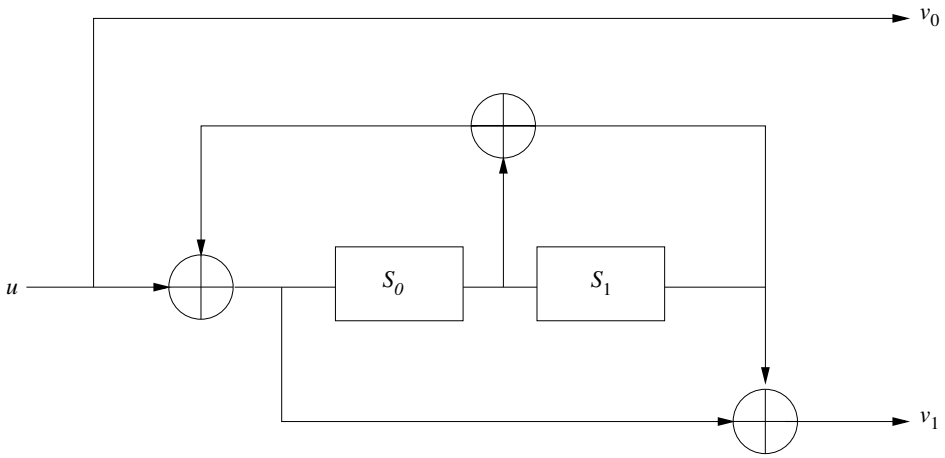

Figure 5.6: The RSC (2,1,2) (1,5/7) encoder

Convolutional codes are usually decoded using a trellis and the Viterbi algorithm [11]. To be able to decode this code using BP one needs to construct a factor graph representation of this code. The **H** matrix of this code is equal to [54]

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{5.6}$$

From **H** the factor graph can be constructed. This **H** matrix has no predefined length and depends on the code segment length that must be decoded at a time. Making the code segments (200,100) (98 information bits, 2 tail bits) an appropriate **H** matrix with dimensions (100×200) can be constructed. Two bits are needed for the convolutional encoder to end in the zero state, and take up the last four columns of the **H** matrix. The last row of **H** must end with the sequence ...1 1 0 1 1 1, to support termination in the zero state. The even columns represent information bits, while the odd columns represent parity bits. With the above the RSC can be decoded using BP. The results from this code were obtained by decoding code segments of length (200,100). Figures 5.13 and 5.14 present the simulated BER results obtained when decoding RSC (2,1,2) with BP under slow and fast fading channel conditions respectively. On each of these

graphs the simulated LOS and NLOS results are displayed. For completeness the simulated BER for AWGN channel conditions is shown in Figure 5.15. These graphs are discussed in detail in Section 5.2.6.

5.2.5 SIMULATED BER CURVES: SIMPLE CODES

All of the simulated BER curves of the simple codes are presented in this section.

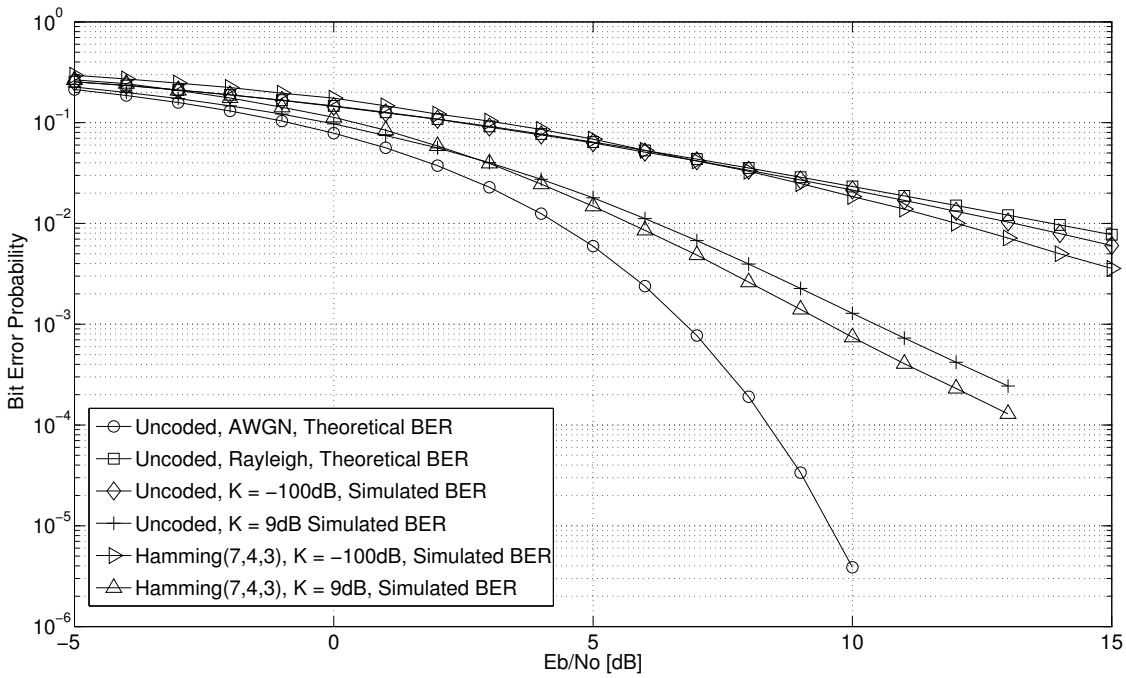

Figure 5.7: BER performance of a Hamming(7,4,3) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

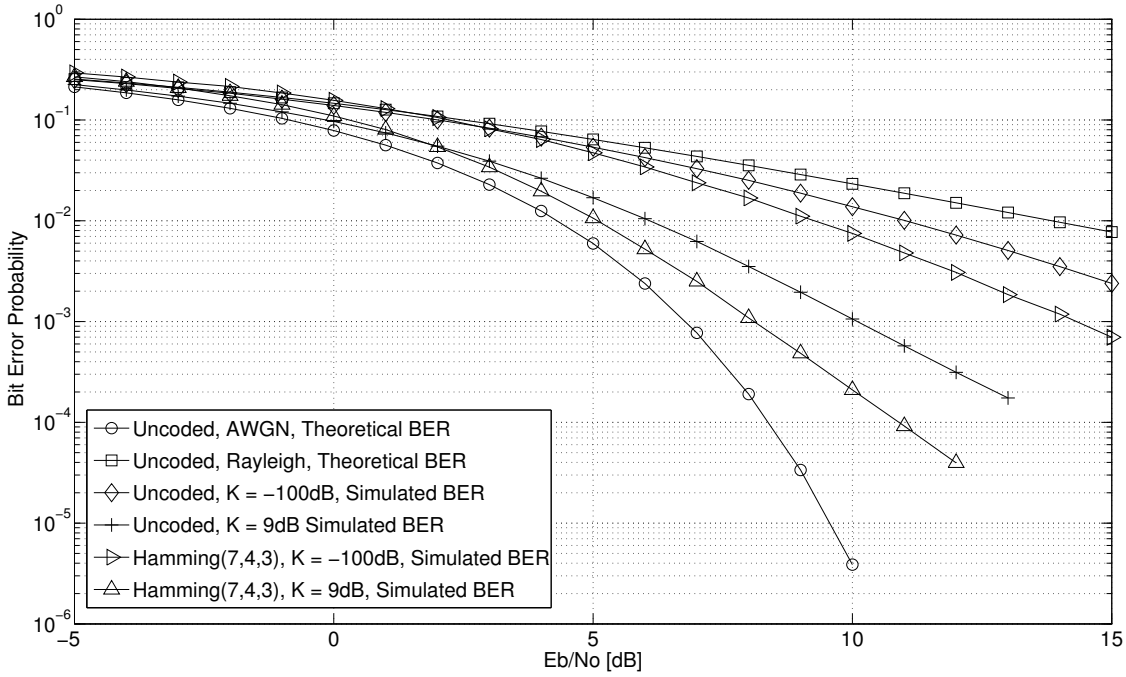

Figure 5.8: BER performance of a Hamming(7,4,3) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

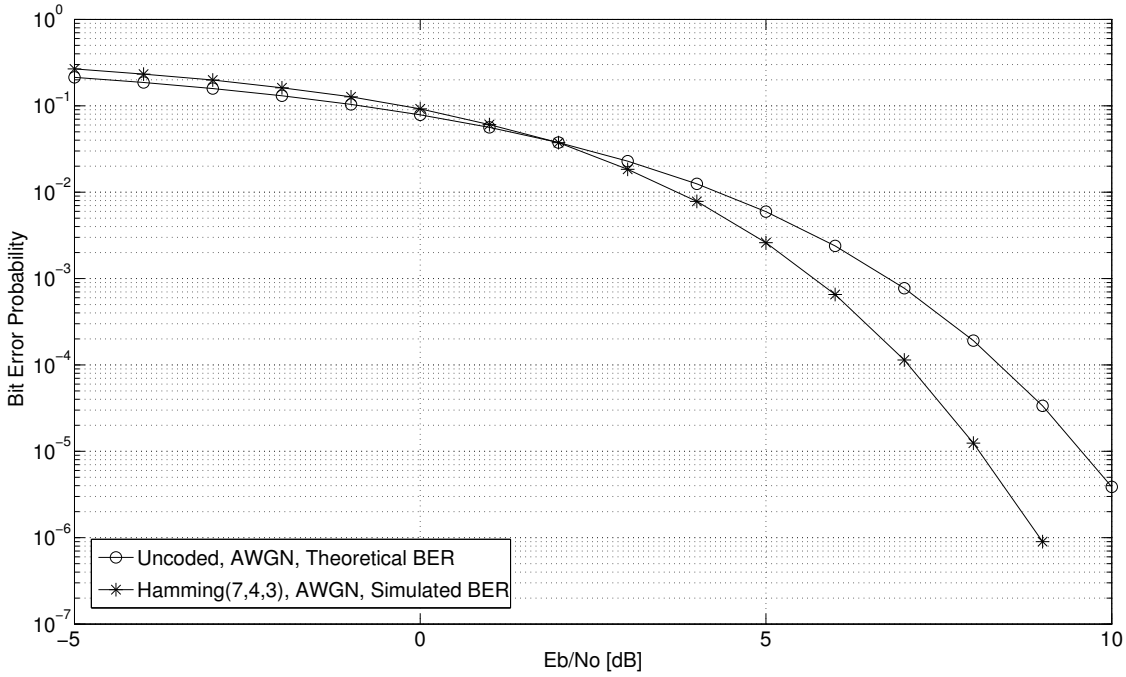

Figure 5.9: BER performance of a Hamming(7,4,3) coded QPSK communication system on a AWGN channel [BP decoding]

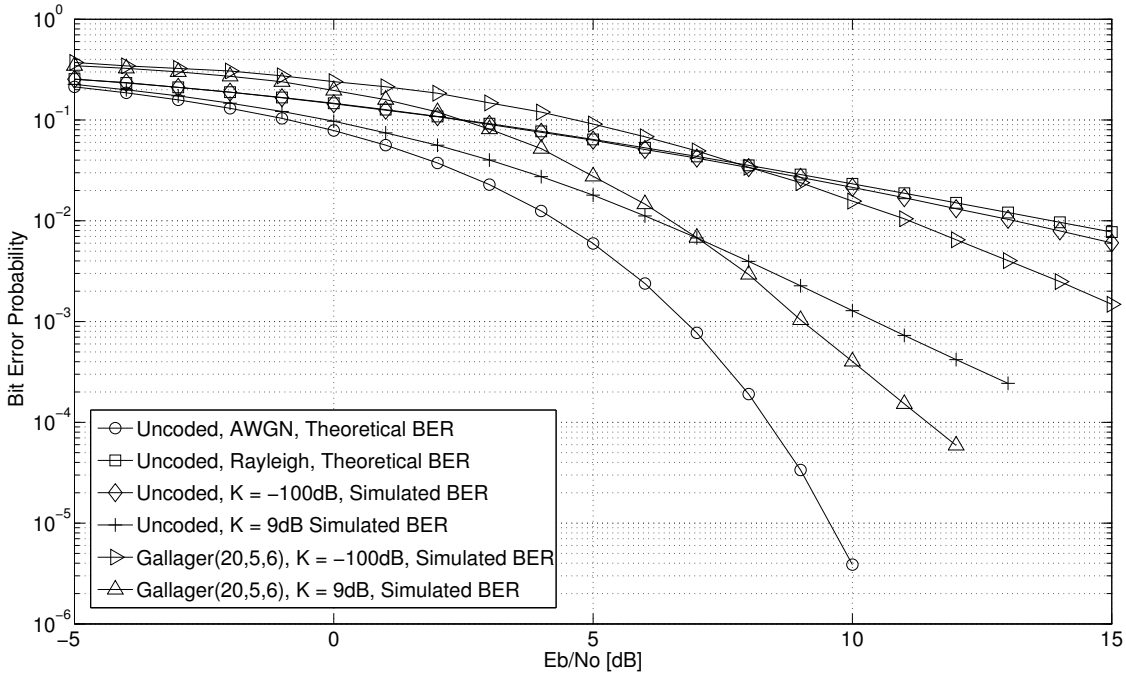

Figure 5.10: BER performance of a Gallager(20,5,6) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

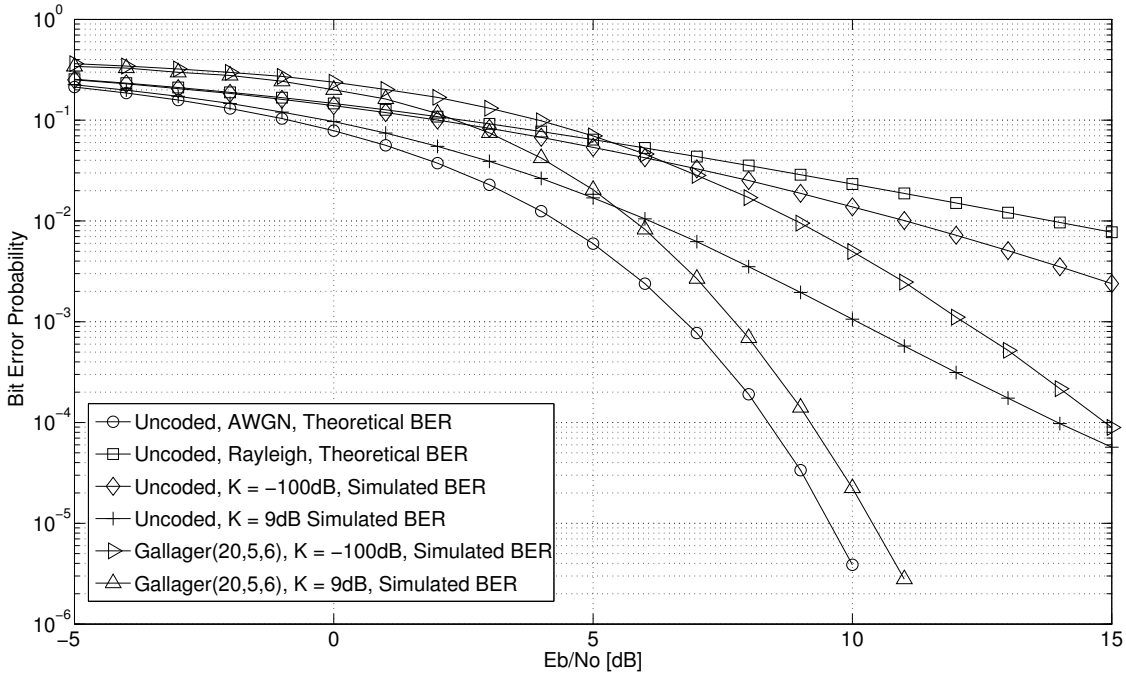

Figure 5.11: BER performance of a Gallager(20,5,6) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

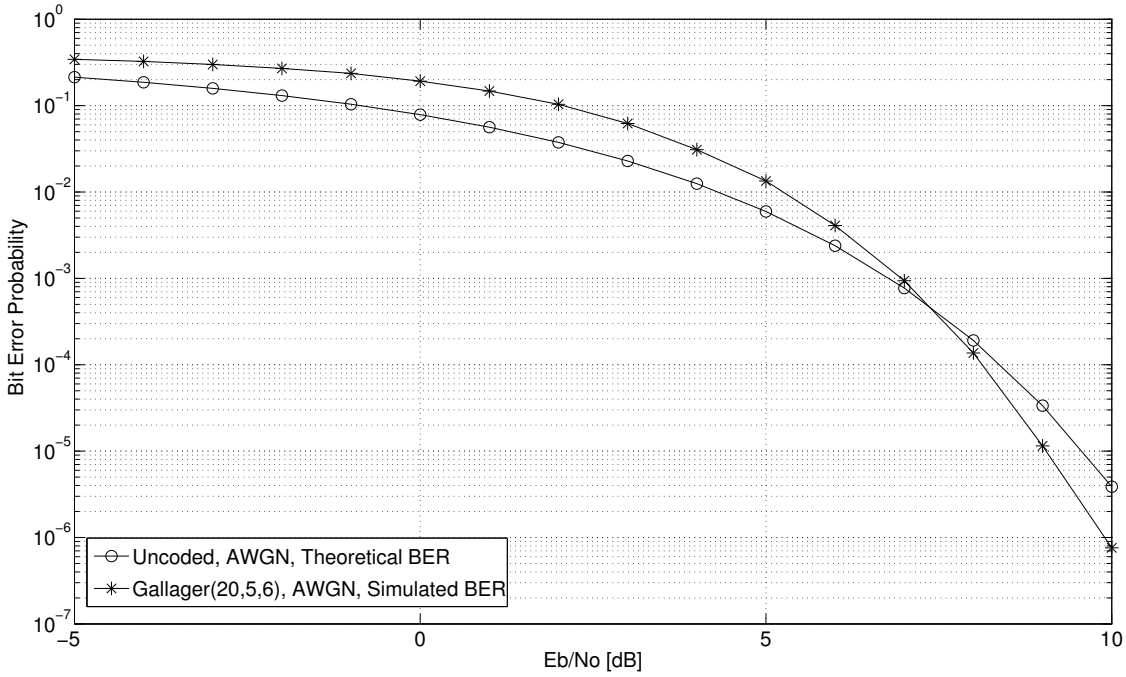

Figure 5.12: BER performance of a Gallager(20,5,6) coded QPSK communication system on a AWGN channel [BP decoding]

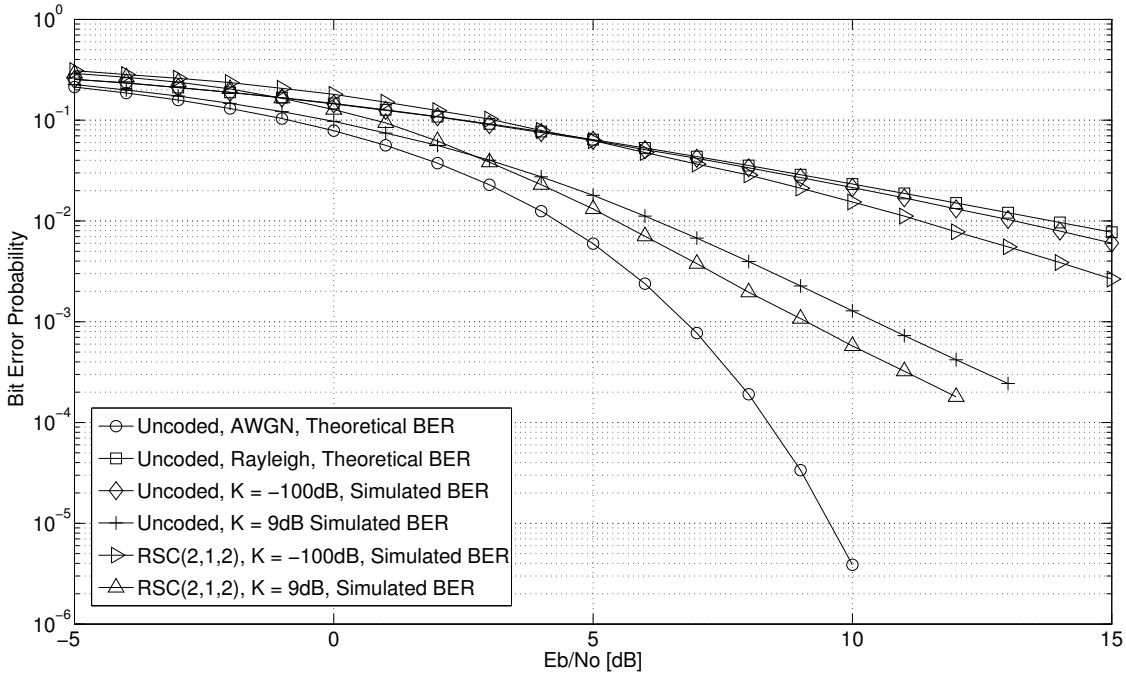

Figure 5.13: BER performance of a RSC(2,1,2) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

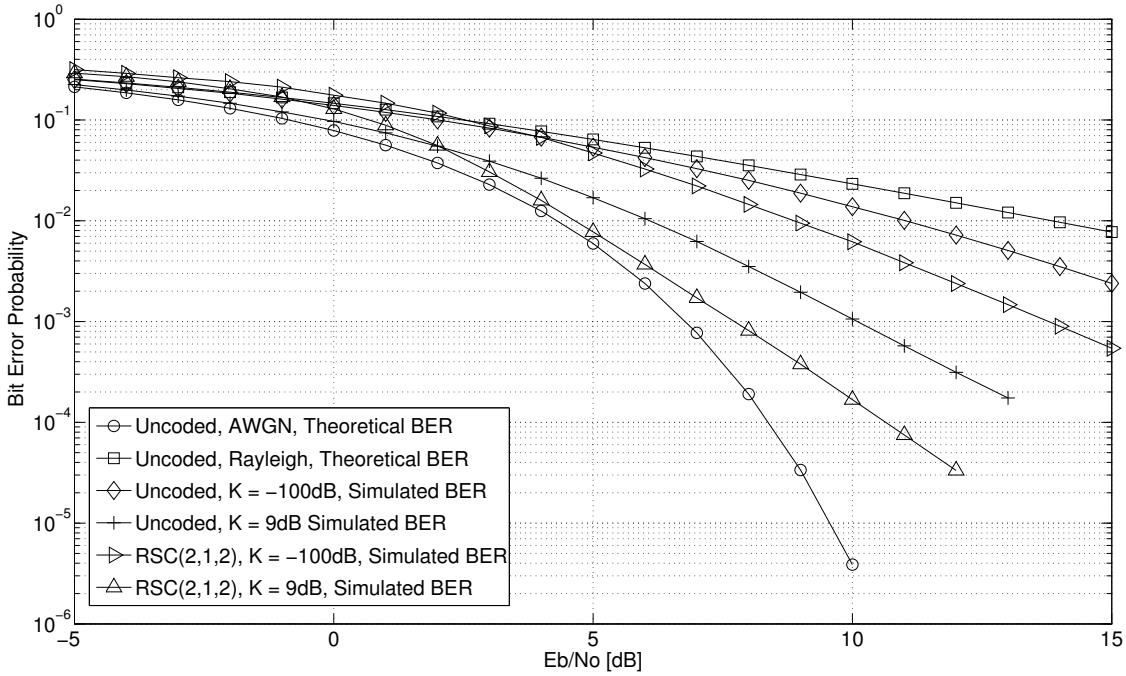

Figure 5.14: BER performance of a RSC(2,1,2) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

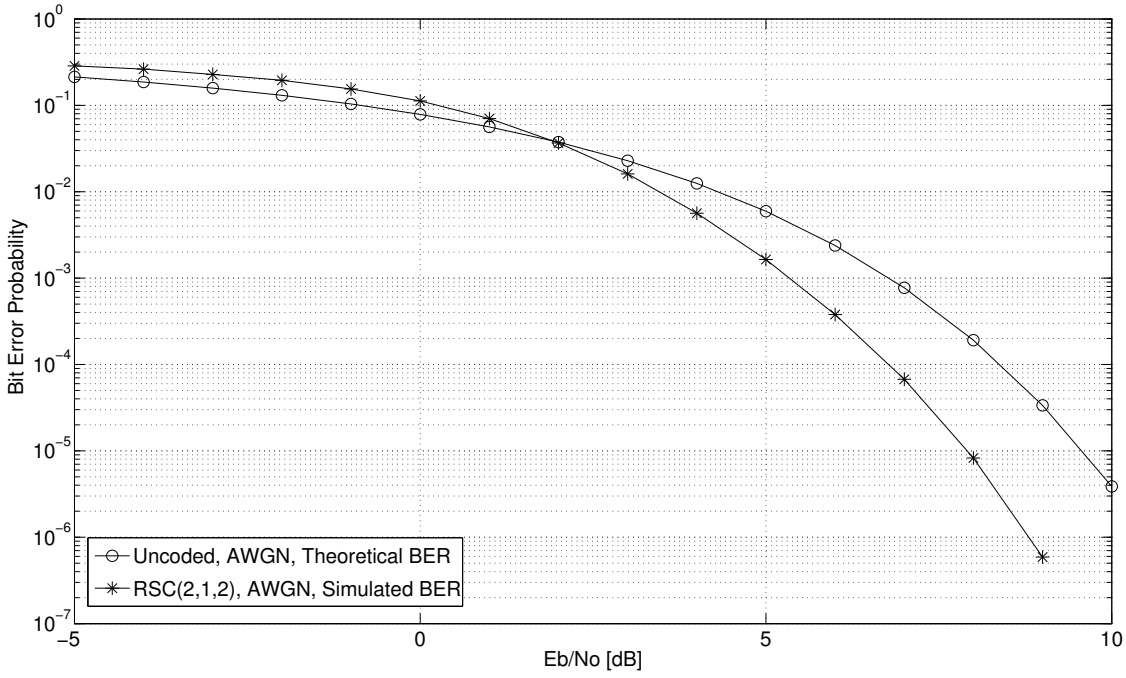

Figure 5.15: BER performance of a RSC(2,1,2) coded QPSK communication system on a AWGN channel [BP decoding]

5.2.6 DISCUSSION OF RESULTS: SIMPLE CODES

The maximum coding gain and cross over point (in dB) of each simple code (compared to uncoded QPSK) is displayed in Tables 5.3 and 5.4 and also in Figures 5.16 and 5.17.

Table 5.3: Coding gain obtained by simple codes when compared to uncoded QPSK

Simple Code	-100dB @33Hz	9dB @33Hz	-100dB @100Hz	9dB @100Hz	AWGN
Hamming [dB]	1.75	1.2	2.7	2.8	1.6
Gallager [dB]	3	2.5	4	5.5	0.6
RSC [dB]	2.2	1.5	3	3	1.8

Table 5.4: Cross over point of simple codes, compared to uncoded QPSK

Simple Code	-100dB @33Hz	9dB @33Hz	-100dB @100Hz	9dB @100Hz	AWGN
Hamming [dB]	6	3	3	2	2
Gallager [dB]	8	7	6.2	5.5	7.5
RSC [dB]	5	3	4	2	2

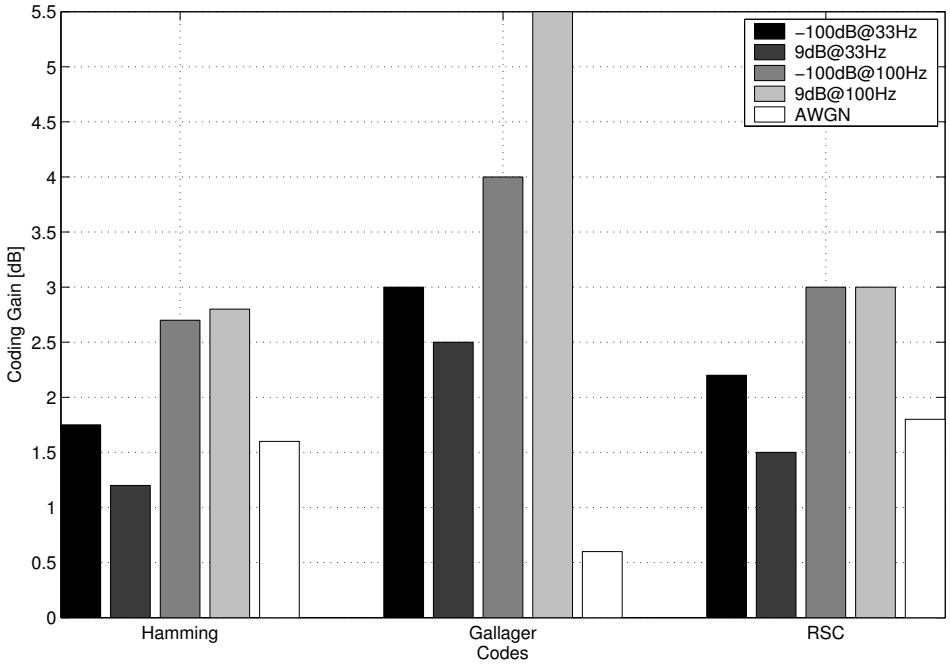

Figure 5.16: Table 5.3 represented graphically

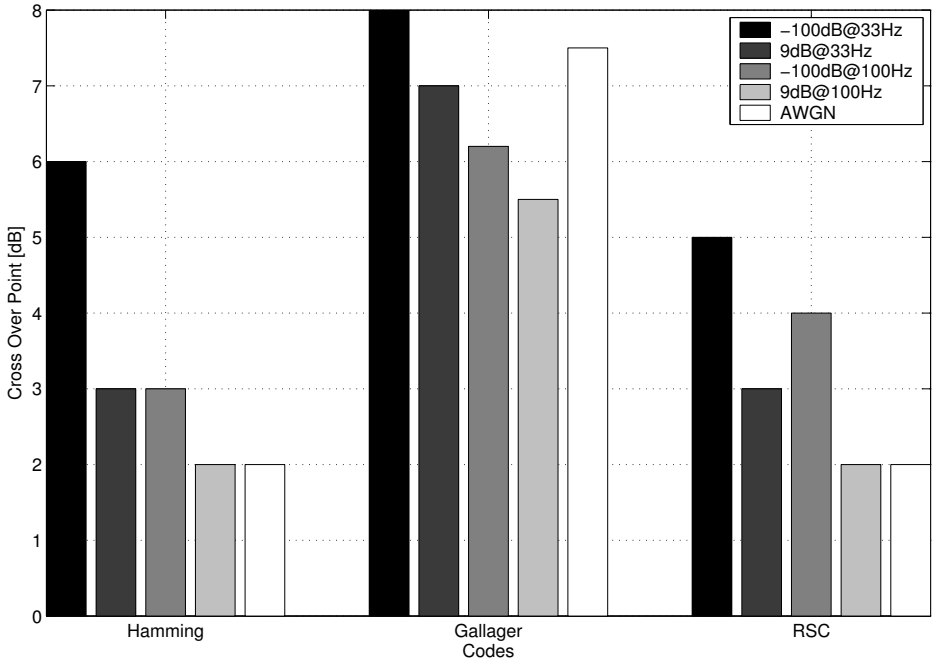

Figure 5.17: Table 5.4 represented graphically

The most important results include:

1. The RSC (2,1,2) code performs on average about 0.3 dB better than the Hamming (7,4,3) code. They have similar cross over profiles. The Hamming (7,4,3) performs on average 2 dB better than an uncoded system.

2. The Gallager (20,5,6) code performs on average 3.1 dB better than uncoded. The price for the improvement is a very high cross over profile (cross at high E_b/N_0 values). The high cross over profile makes the code impractical for practical implementation. The code also performs only marginally better under AWGN conditions due to a high cross-over profile. The Gallager (20,5,6) code is included for historical value.

3. The simple codes studied can not be compared directly to the fountain codes, since only 10 iterations of BP were used for decoding. The strength of larger codes lie in the fact that the amount of belief propagation iterations is large. For instance the Hamming (7,4,3) code will not show much improvement if the amount of iterations is increased from 10 to 100 (due to a small factor graph). On the other hand the LDPC (40,20) code will show large improvements if the amount of decoding cycles is increased. That is why only certain codes are compared with each other. For this study only 10 iterations were considered. See Chapter 6 for more detailed conclusions.

5.2.7 FOUNTAIN CODES

One of the main contributions of this thesis is the soft decoding of fountain codes (using BP). The three example fountain codes used to illustrate how this can be accomplished are presented here. The basic theory of how each fountain code is constructed and how to decode these codes were discussed in Chapter 3 (see Section 3.4.3 for soft decoding). All of the codes from this section were decoded using 10 iterations of BP.

5.2.7.1 TORNADO (80,40)

The Tornado code used for the simulations is exactly the same as the code designed in Section 3.3.4, except for the fact that B_2 (second graph used to create first layer) no longer consists of $3k$ edges. The amount of edges is reduced to k to lower the chances of creating parallel assignments. The code is just too small for so many edges. Other derivations of this basic Tornado code are also simulated on the QPSK transmission model. In the first case the code consisting of only the first layer of the Tornado (80,40)

code is called Tornado $(60,40)_f$. The last layer of the Tornado (80,40) code is called Tornado $(40,20)_l$. The simulated BER performance of the first layer of parity bits (of Tornado (80,40)) is also investigated (the 20 parity bits produced by Tornado $(60,40)_f$) and referred to using the notation Tornado $(80,40)_p$. The encoding of Tornado codes and the noisy decoding of Tornado are discussed in Sections 3.3 and 3.4.3 respectively. The figures associated with the simulated BER performance of each Tornado code on the QPSK transmission model is given by Table 5.5 (see Section 5.2.7.2). These graphs are discussed in detail in Section 5.2.7.3.

Table 5.5: Figures representing the simulated BER performance of a Tornado code

Code	Slow fading	Fast fading	AWGN
Tornado(80,40)	Figure 5.18	Figure 5.19	Figure 5.20
Tornado(60,40)$_f$	Figure 5.21	Figure 5.22	Figure 5.23
Tornado(40,20)$_l$	Figure 5.24	Figure 5.25	Figure 5.26
Tornado(80,40)$_p$	Figure 5.27	Figure 5.28	Figure 5.29

5.2.7.2 SIMULATED BER CURVES: TORNADO

The simulated BER performance of each Tornado variation is given in this section.

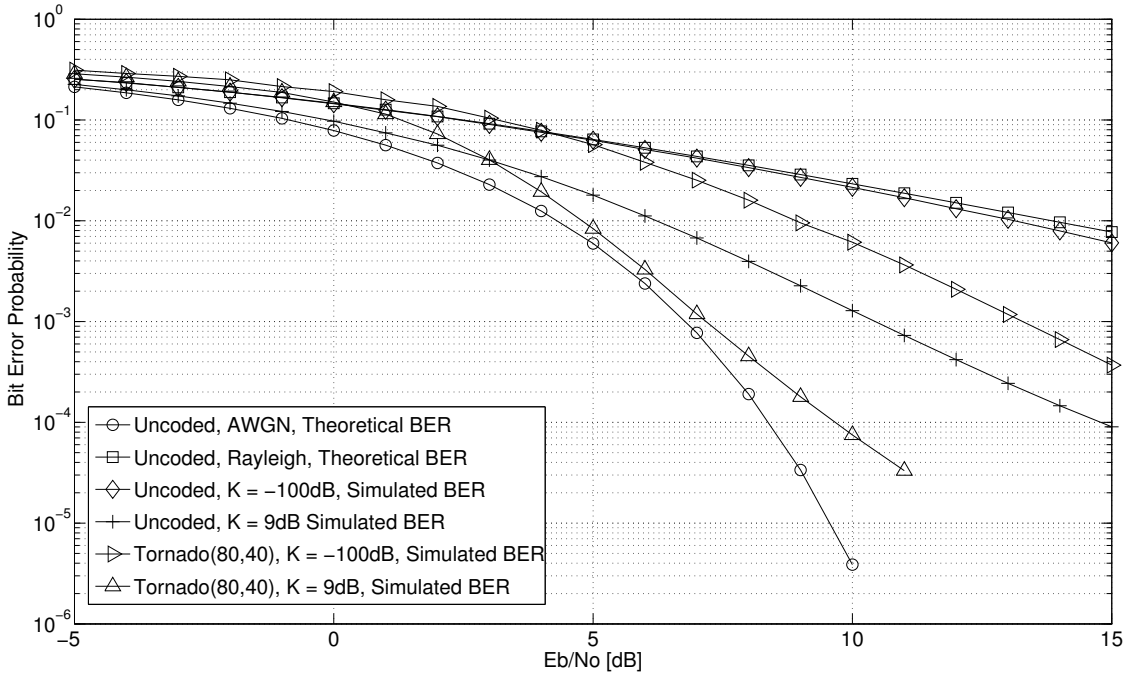

Figure 5.18: BER performance of a Tornado(80,40) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

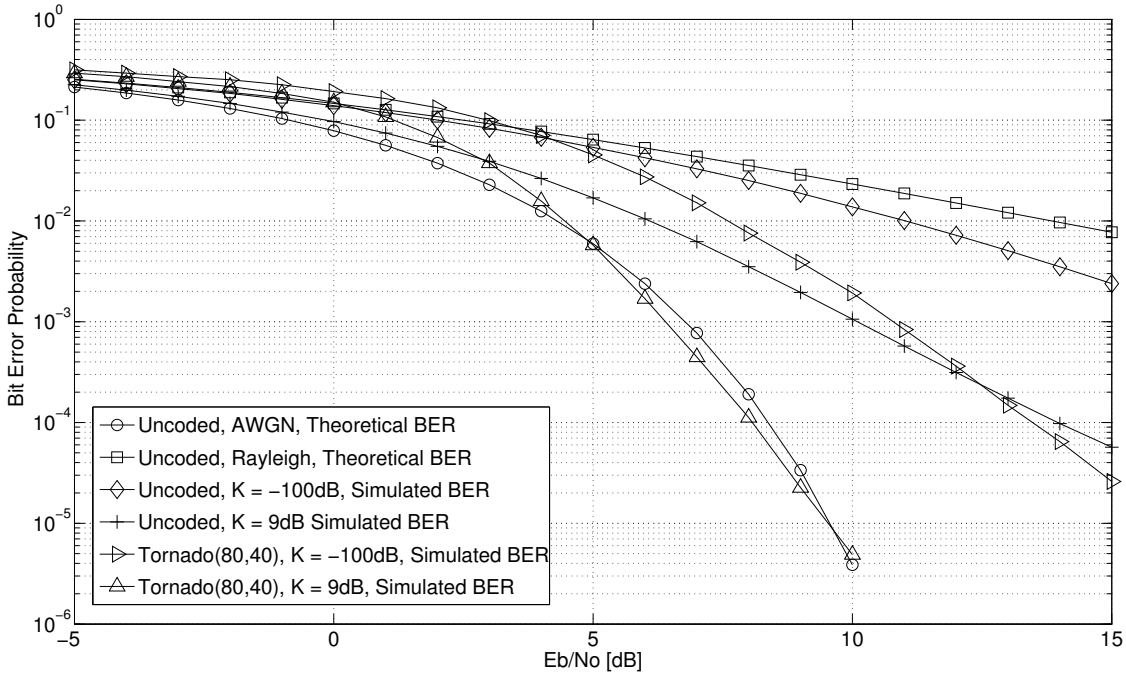

Figure 5.19: BER performance of a Tornado(80,40) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

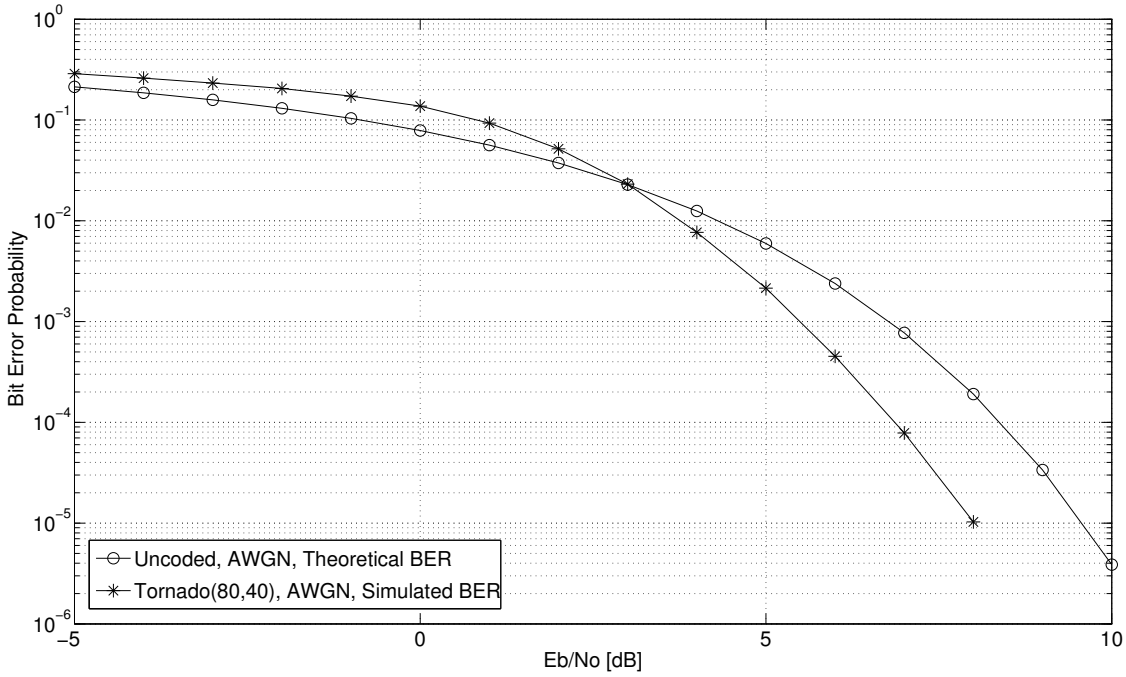

Figure 5.20: BER performance of a Tornado(80,40) coded QPSK communication system on a AWGN channel [BP decoding]

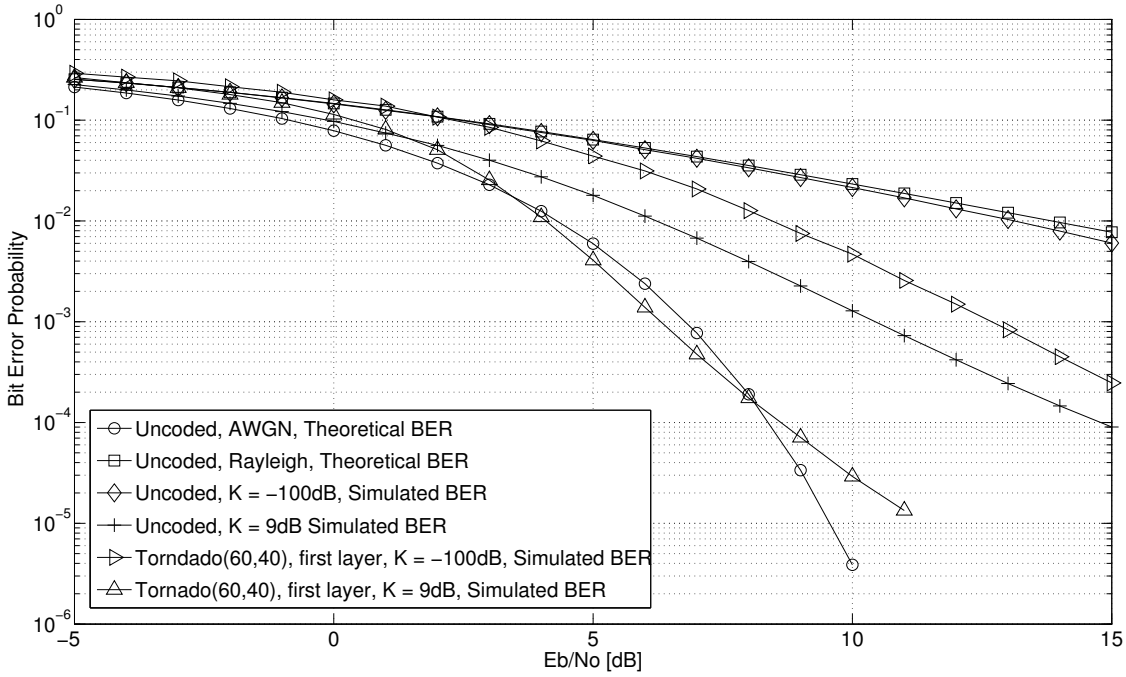

Figure 5.21: BER performance of a Tornado(60,40)$_f$ coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

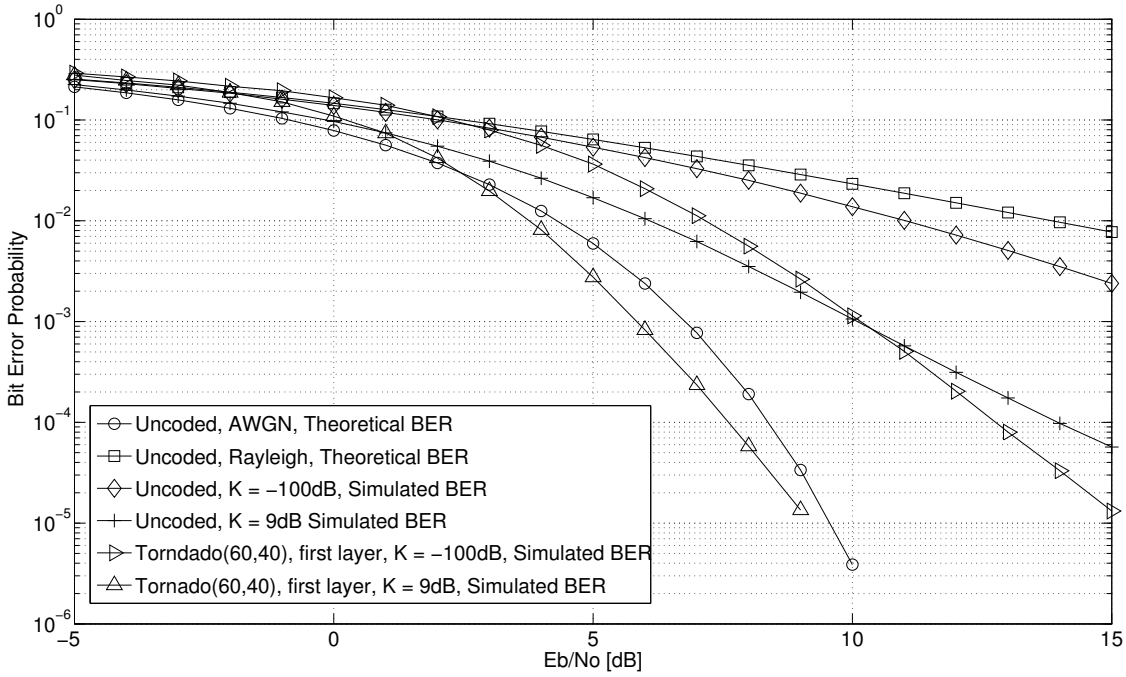

Figure 5.22: BER performance of a Tornado(60,40)$_f$ coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

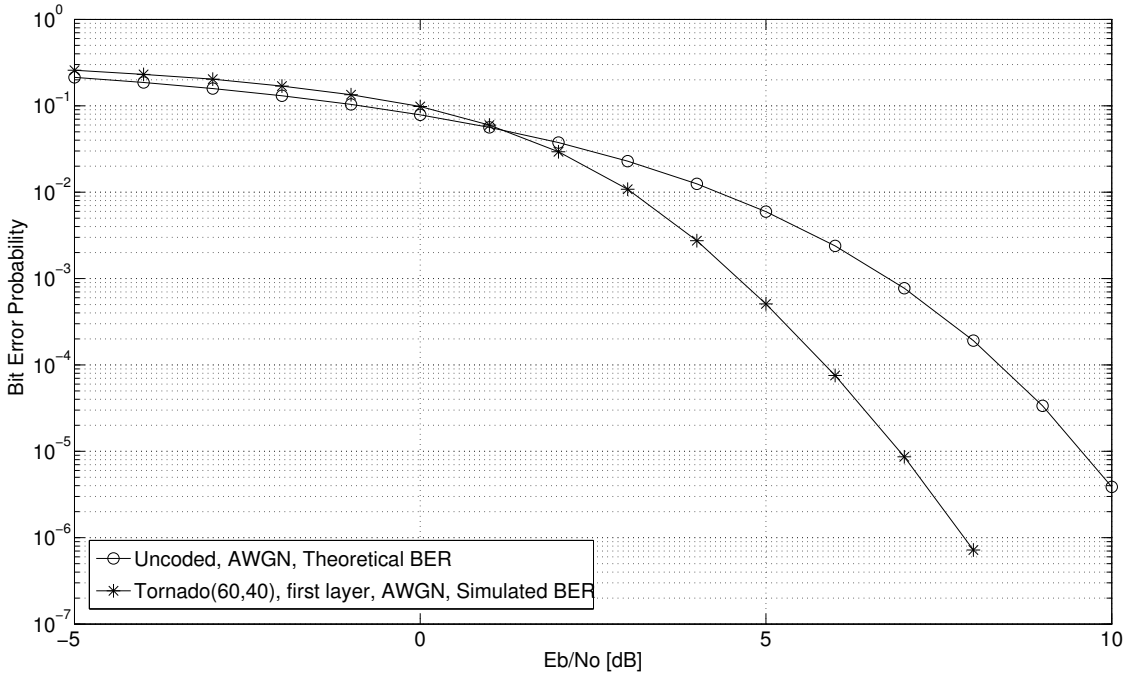

Figure 5.23: BER performance of a Tornado(60,40)$_f$ coded QPSK communication system on a AWGN channel [BP decoding]

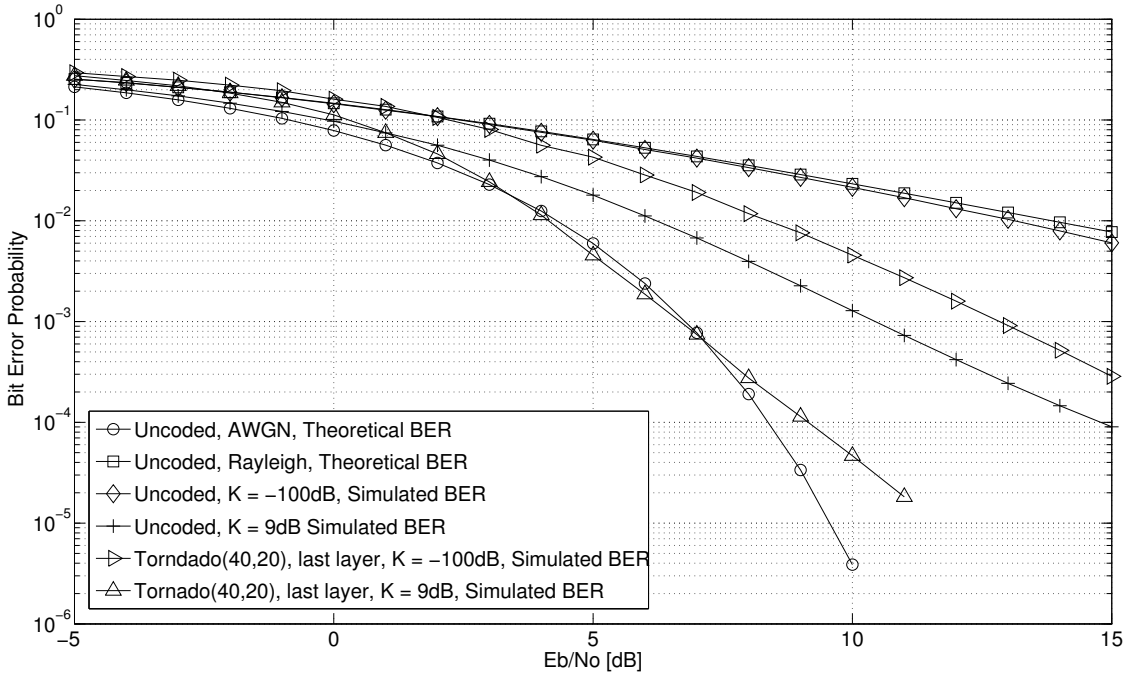

Figure 5.24: BER performance of a Tornado(40,20)$_l$ coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

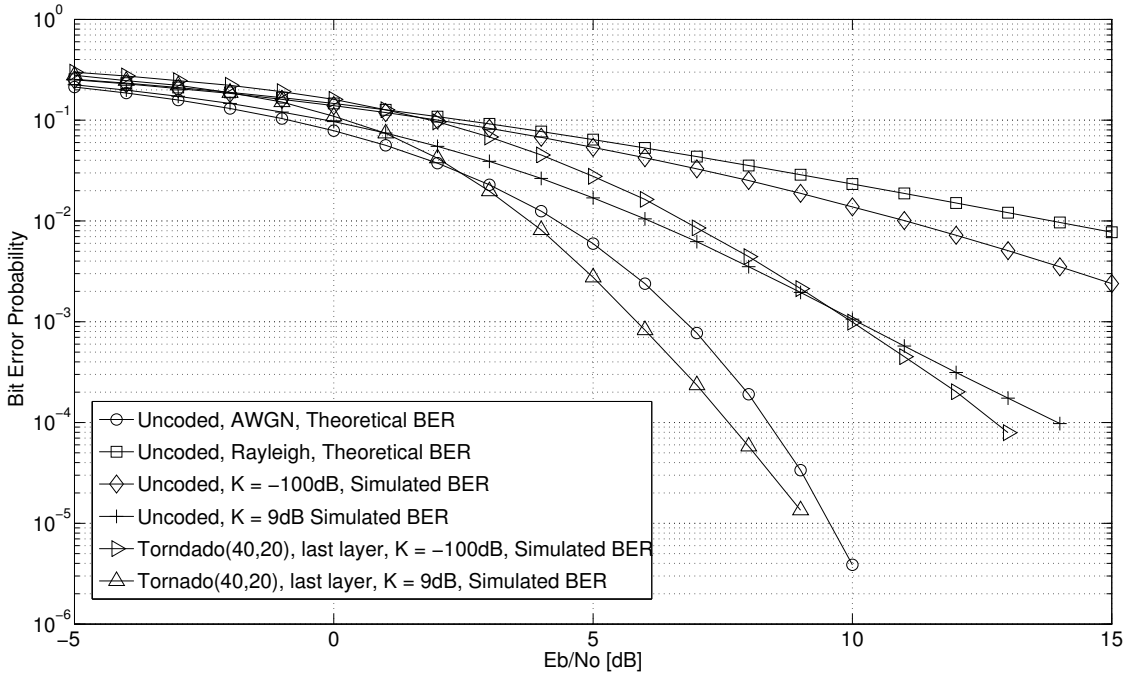

Figure 5.25: BER performance of a Tornado(40,20)$_l$ coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

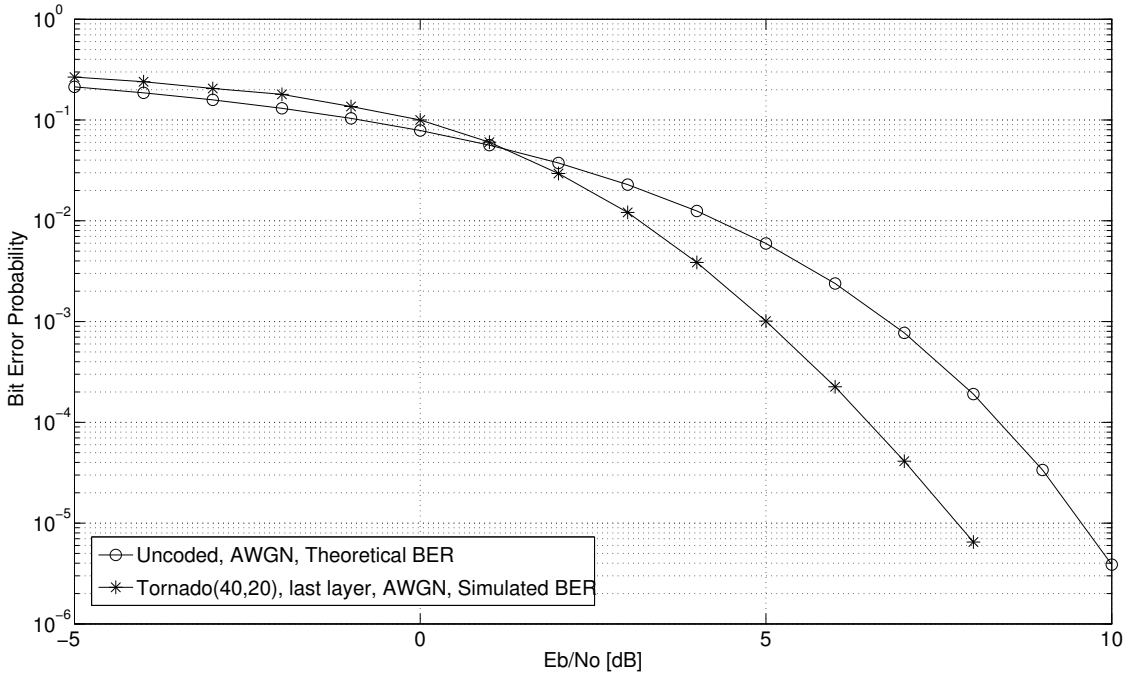

Figure 5.26: BER performance of a Tornado(40,20)$_l$ coded QPSK communication system on a AWGN channel [BP decoding]

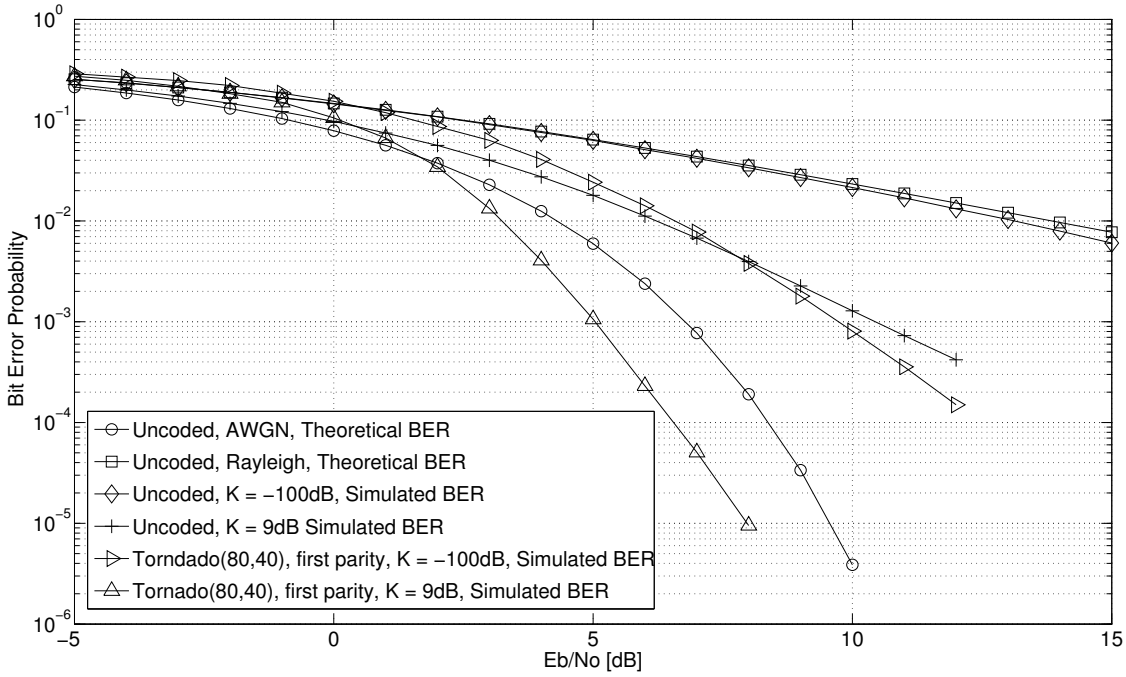

Figure 5.27: BER performance of a Tornado(80,40)$_p$ coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

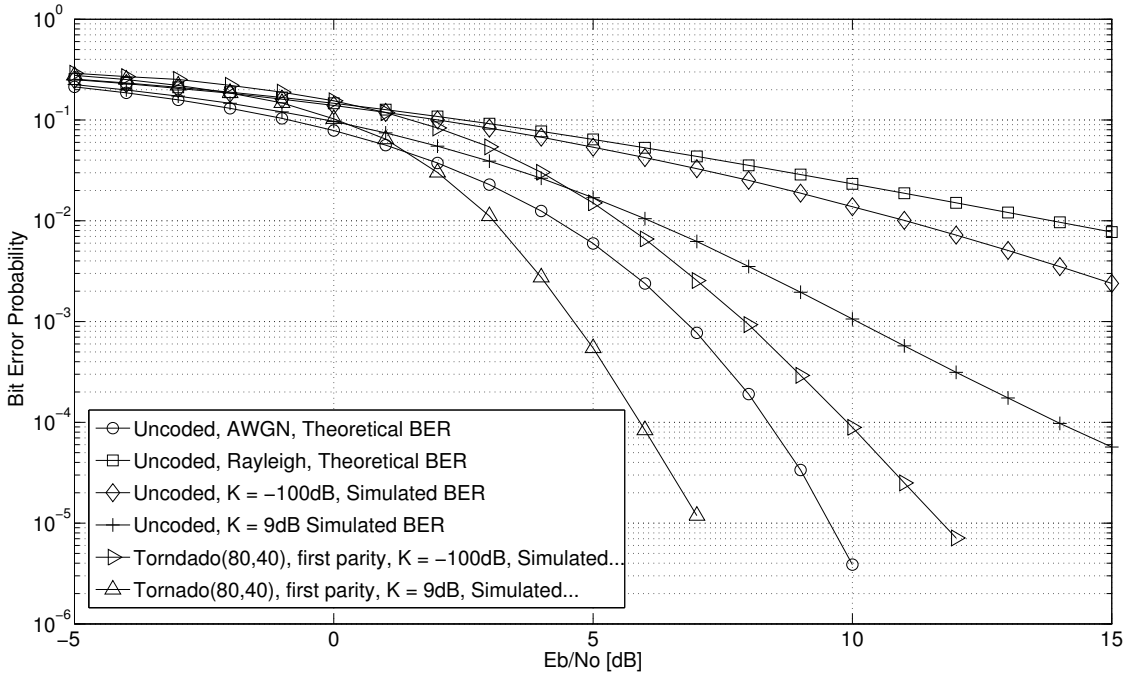

Figure 5.28: BER performance of a Tornado$(80,40)_p$ coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

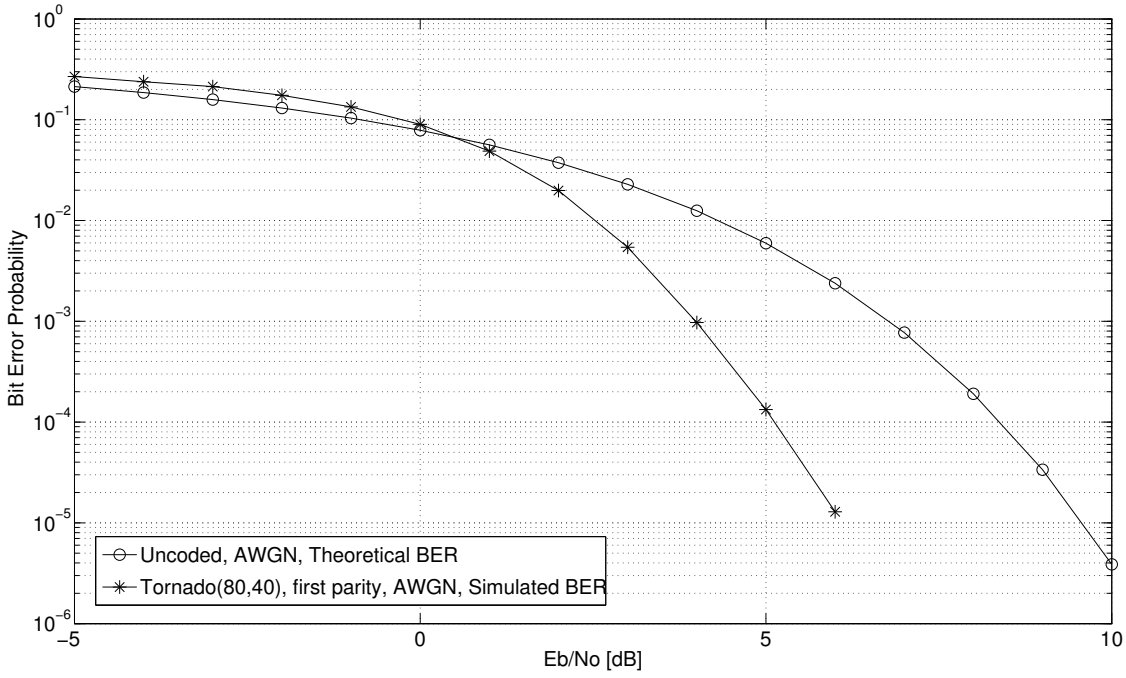

Figure 5.29: BER performance of a Tornado$(80,40)_p$ coded QPSK communication system on a AWGN channel [BP decoding]

5.2.7.3 DISCUSSION OF TORNADO CODE RESULTS

The maximum coding gain and cross over point of each Tornado variation (compared with uncoded QPSK) are displayed in Tables 5.6 and 5.7 and also in Figures 5.30 and 5.31

Table 5.6: Coding gain obtained by Tornado codes when compared to uncoded QPSK

Tornado Code	-100dB @33Hz	9dB @33Hz	-100dB @100Hz	9dB @100Hz	AWGN
Tornado [dB]	5	5.1	5.2	6.5	1.5
Tornado$_l$ [dB]	5.5	5.8	6.2	6.7	1.8
Tornado$_f$ [dB]	6	6.2	6	7	2.8
Tornado$_p$ [dB]	7.8	6.5	7.8	8.8	3.5

Table 5.7: Cross over point of Tornado codes, compared to uncoded QPSK

Tornado Code	-100dB @33Hz	9dB @33Hz	-100dB @100Hz	9dB @100Hz	AWGN
Tornado [dB]	4	3	4	3	3
Tornado$_l$ [dB]	2	1	1	1	1
Tornado$_f$ [dB]	2	1	2	1	1
Tornado$_p$ [dB]	1	0	1	0	0.5

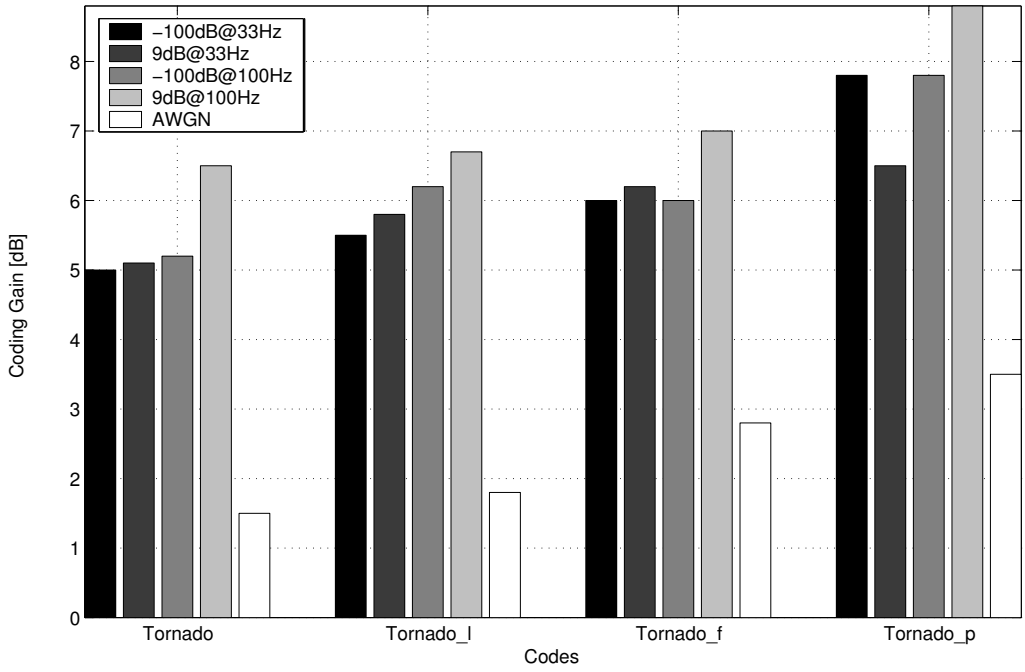

Figure 5.30: Table 5.6 represented graphically

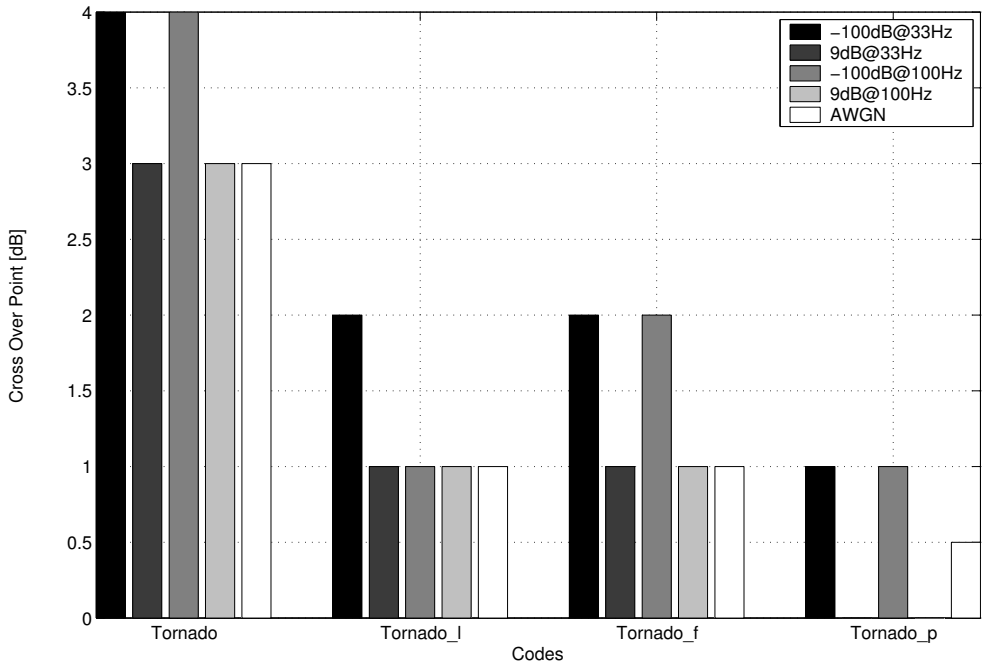

Figure 5.31: Table 5.7 represented graphically

The most important results pertaining to the Tornado code include:

1. The Tornado (80,40) code is outperformed by its two layers Tornado $(40,20)_l$ and Tornado $(60,40)_f$. Tornado $(60,40)_f$ also outperforms Tornado $(40,20)_l$ on average. The Tornado (80,40) code has a higher cross over profile than its constituent codes. The two one layer Tornado variations have similar cross over profiles.

2. The result of only testing the first parity layer of the Tornado (80,40) code is that these bits are extremely well protected. Tornado $(80,40)_p$ has a lower BER and cross over profile than all Tornado variations tested. See Chapter 6 for more detailed conclusions.

5.2.7.4 REGULAR LDPC (40,20)

The regular LDPC [7, 14, 30] code of this section is used as the pre-code of the Raptor (80,20) code presented later on in this main section. The LDPC code studied here is a (3,6) uniformly distributed regular code. This means that its **H** matrix has 3 and 6 uniformly distributed ones in each of its columns and rows respectively. The **H** matrix of the LDPC (40,20) code was generated randomly. The easiest way of encoding LDPC codes is to multiply the message with the generator matrix. The generator matrix **G** is obtained from **H** by using Gauss-Jordan elimination. Decoding is performed by using the algorithm presented in Section 2.8.4. Figures 5.32 and 5.33 present the simulated BER results obtained when decoding LDPC (40,20) with BP under slow and fast fading channel conditions respectively. On each of these graphs the simulated LOS and NLOS results are displayed. For completeness the simulated BER for AWGN channel conditions is shown in Figure 5.34 (see Section 5.2.7.7). These graphs are discussed in Section 5.2.7.8.

5.2.7.5 SYSTEMATIC LT (80,40)

LT codes were discussed in Section 3.4. An LT code is uniquely defined by its distribution, generally the improved robust soliton distribution with parameters (K, δ, c) is used. The LT (80,40) code studied here has the following values for these parameters (40,0.05,0.05). From (K, δ, c) the LT code can be generated. The LT encoder was discussed in Section 3.4.1 and its noisy decoder in Section 3.4.3. Take note the LT code used was systematic [23]. Non-systematic LT codes do not perform well with the decoding algorithm from Section 3.4.3. Figures 5.35 and 5.36 present the simulated BER results obtained when decoding LT (80,40) with BP under slow and fast fading channel conditions respectively. On each of these graphs the simulated LOS and NLOS results

are displayed. For completeness the simulated BER for AWGN channel conditions is shown in Figure 5.37 (see Section 5.2.7.7). These graphs are discussed in Section 5.2.7.8.

5.2.7.6 SYSTEMATIC RAPTOR (80,20)

The Raptor code is discussed in detail in Section 3.5, where it was illustrated that a Raptor code consists of a pre-code (usually an LDPC code) and a weakened LT code. The Raptor code constructed in this section however does not conform to the standard design of a Raptor code. The LT code used is not weakened, and the LDPC code used is regular. Deviations from the standard design is allowed, since decoding is done on a noisy channel and the improved speed of Raptor codes are no longer the primary design consideration. The Raptor code of this section is systematic since the decoding algorithm from Section 3.4.3 only performs well on systematic codes. So the Raptor code used here consists of the LDPC (40,20) (pre-code) and LT (80,40) code discussed earlier. Encoding was discussed in Section 3.5 and decoding in 3.4.3 (since the new code is one new factor graph decoding algorithm from Section 3.4.3 may be applied). Figures 5.38 and 5.39 present the simulated BER results obtained when decoding Raptor (80,20) with BP under slow and fast fading channel conditions respectively. On each of these graphs the simulated LOS and NLOS results are displayed. For completeness the simulated BER for AWGN channel conditions is shown in Figure 5.40 (see Section 5.2.7.7). These graphs are discussed in Section 5.2.7.8.

5.2.7.7 SIMULATED BER CURVES: LDPC, LT AND RAPTOR

The simulated BER performance of the remaining fountain codes are given in this section.

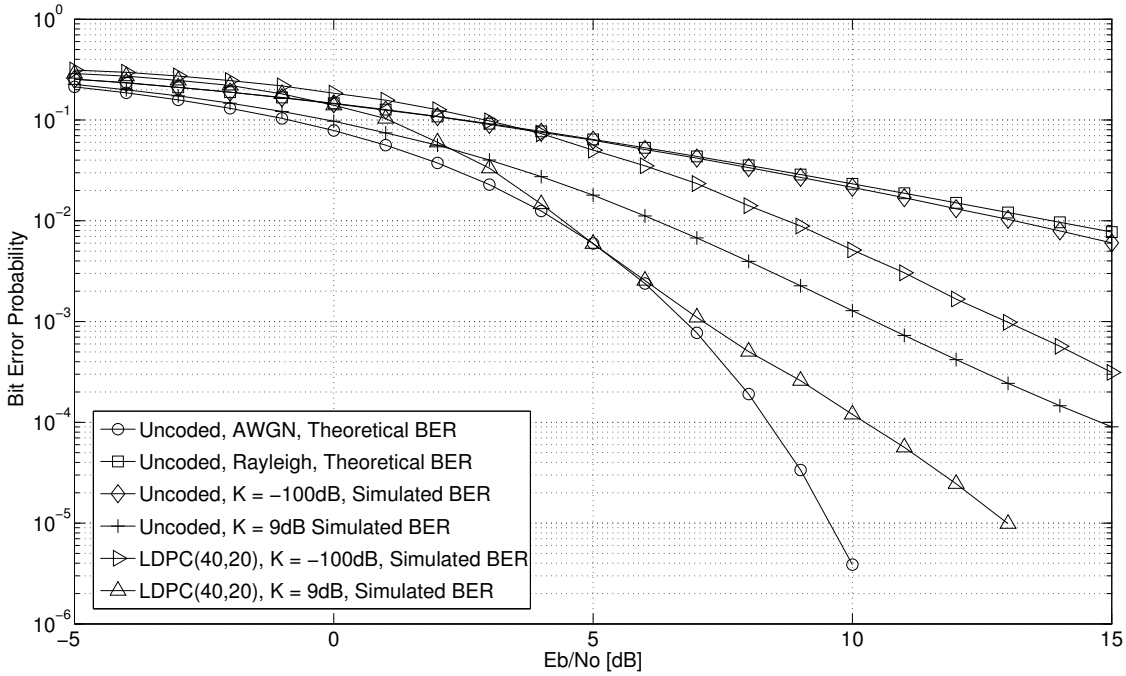

Figure 5.32: BER performance of a LDPC(40,20) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

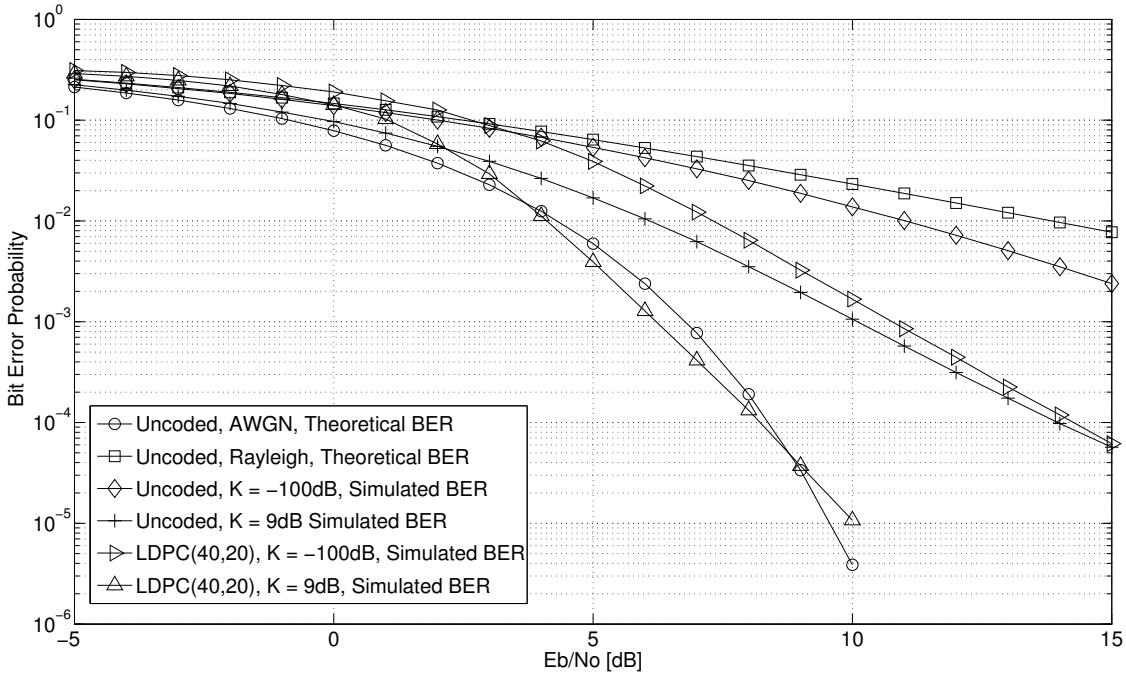

Figure 5.33: BER performance of a LDPC(40,20) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

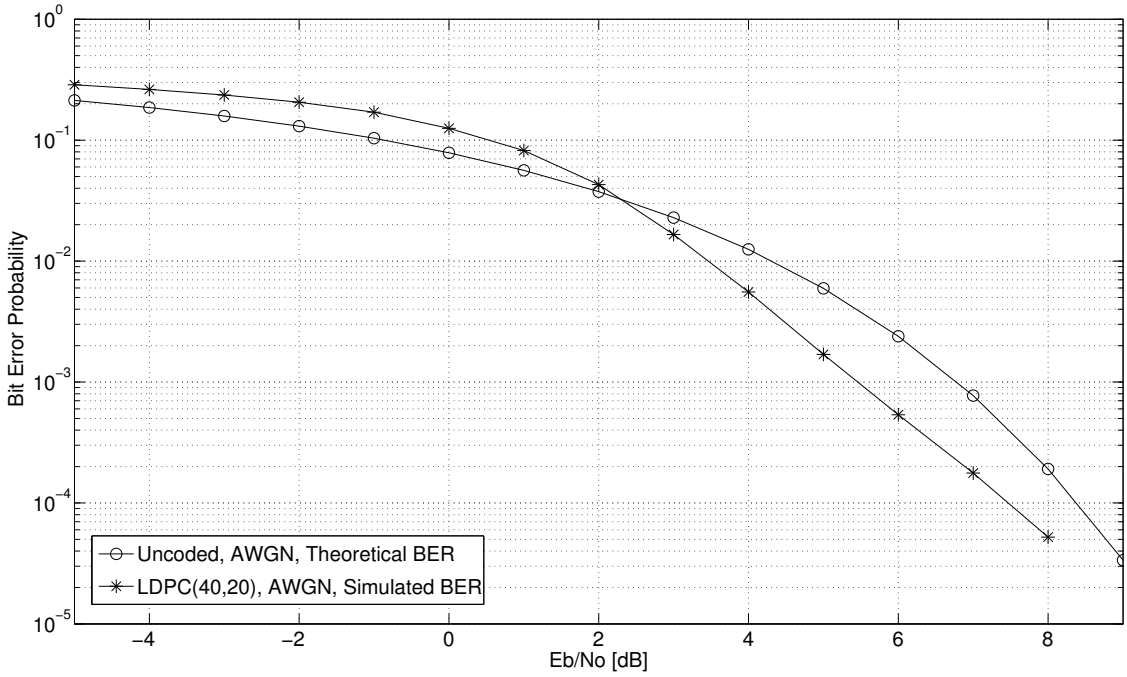

Figure 5.34: BER performance of a LDPC(40,20) coded QPSK communication system on a AWGN channel [BP decoding]

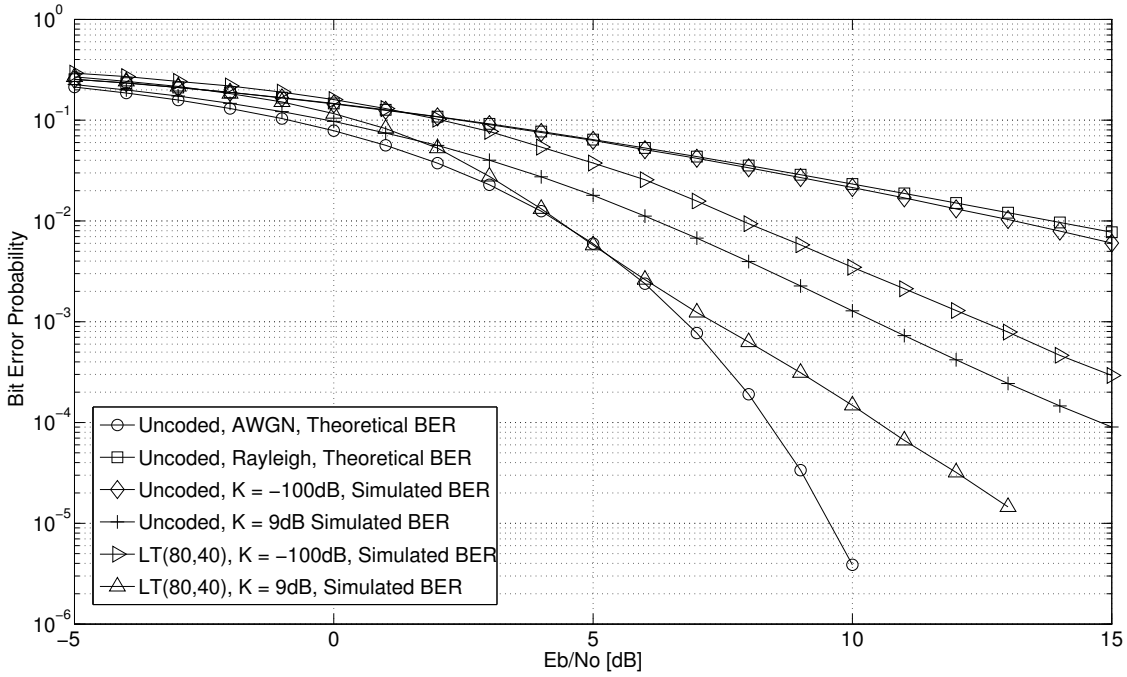

Figure 5.35: BER performance of a LT(80,40) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

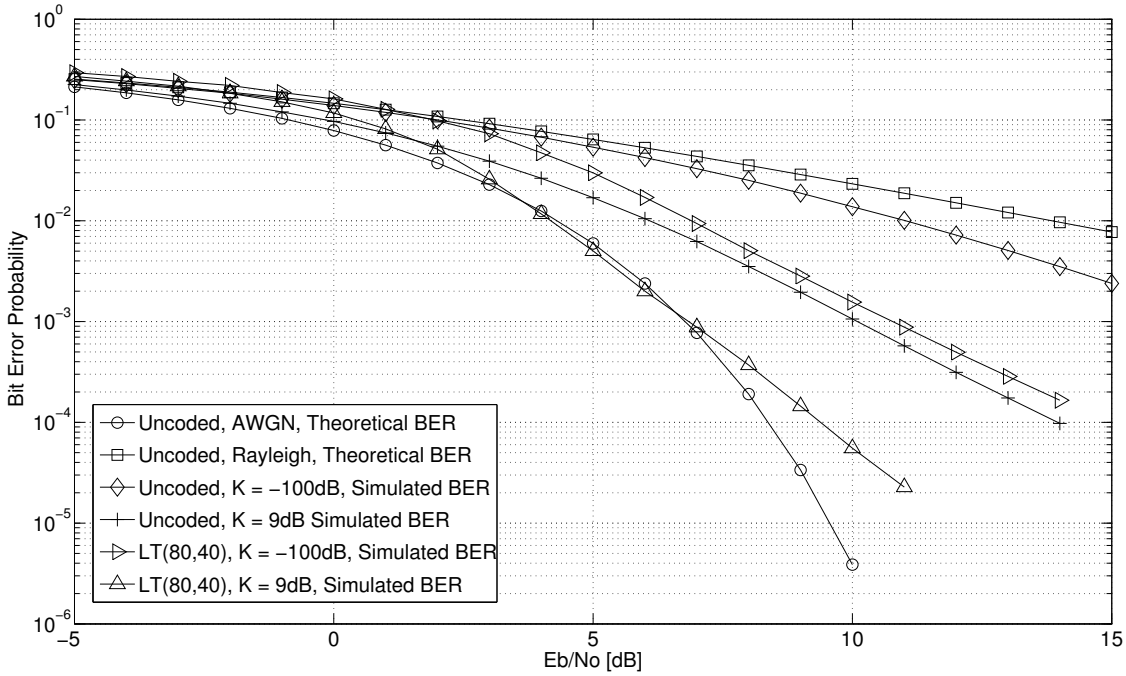

Figure 5.36: BER performance of a LT(80,40) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

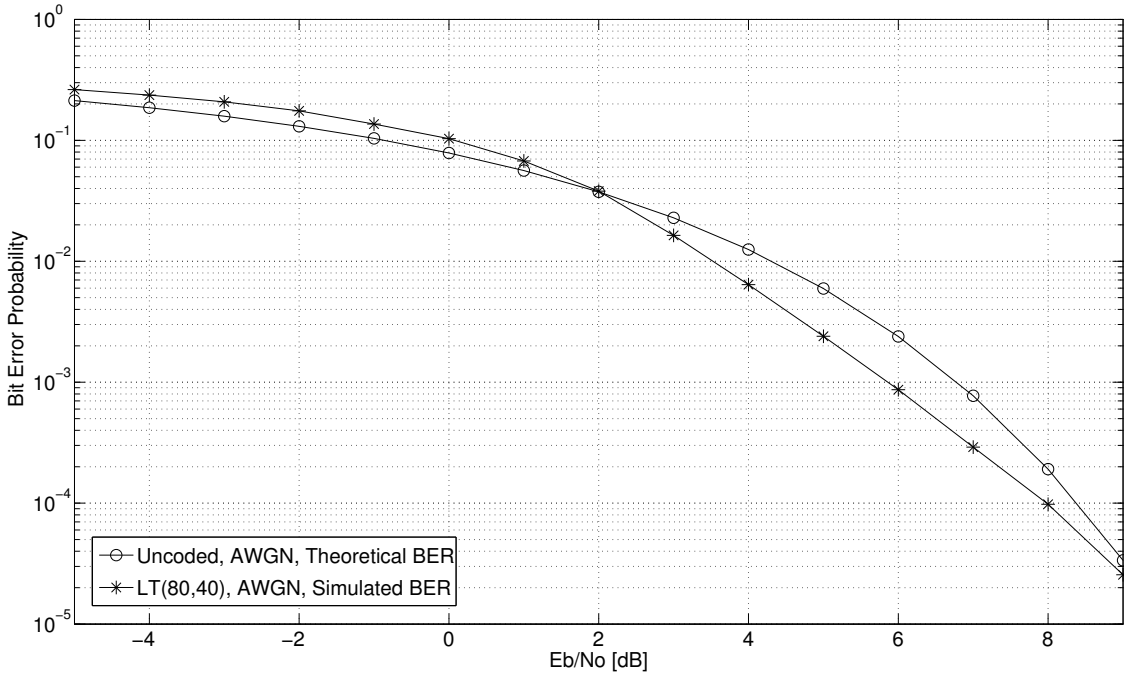

Figure 5.37: BER performance of a LT(80,40) coded QPSK communication system on a AWGN channel [BP decoding]

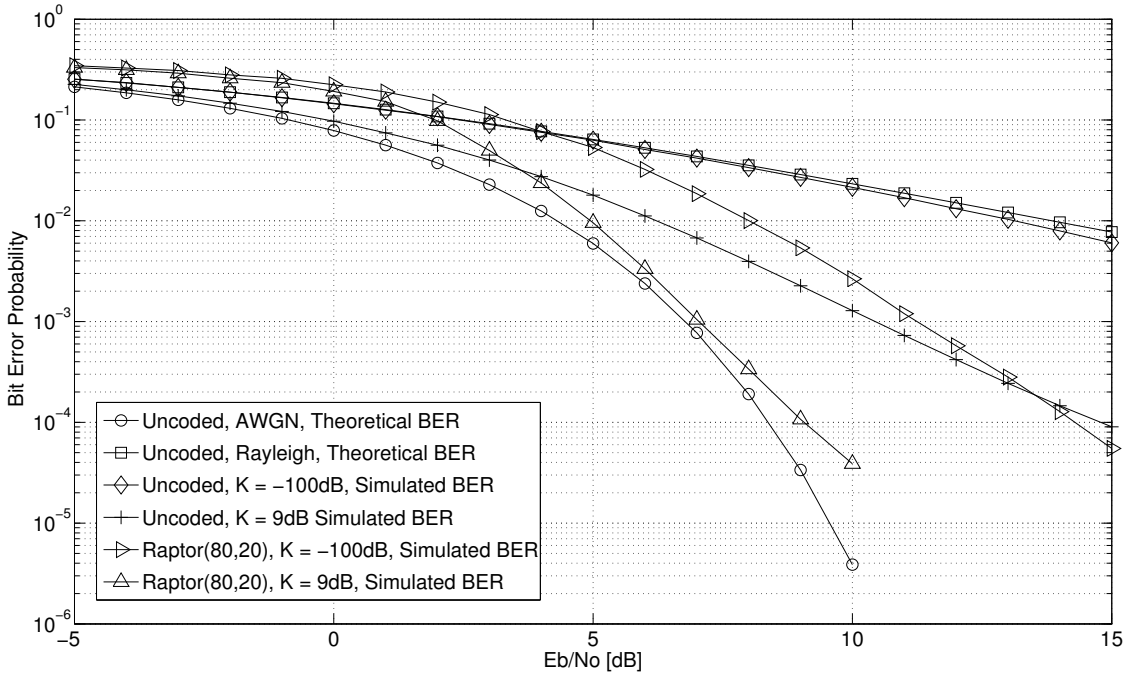

Figure 5.38: BER performance of a Raptor(80,20) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 33Hz$

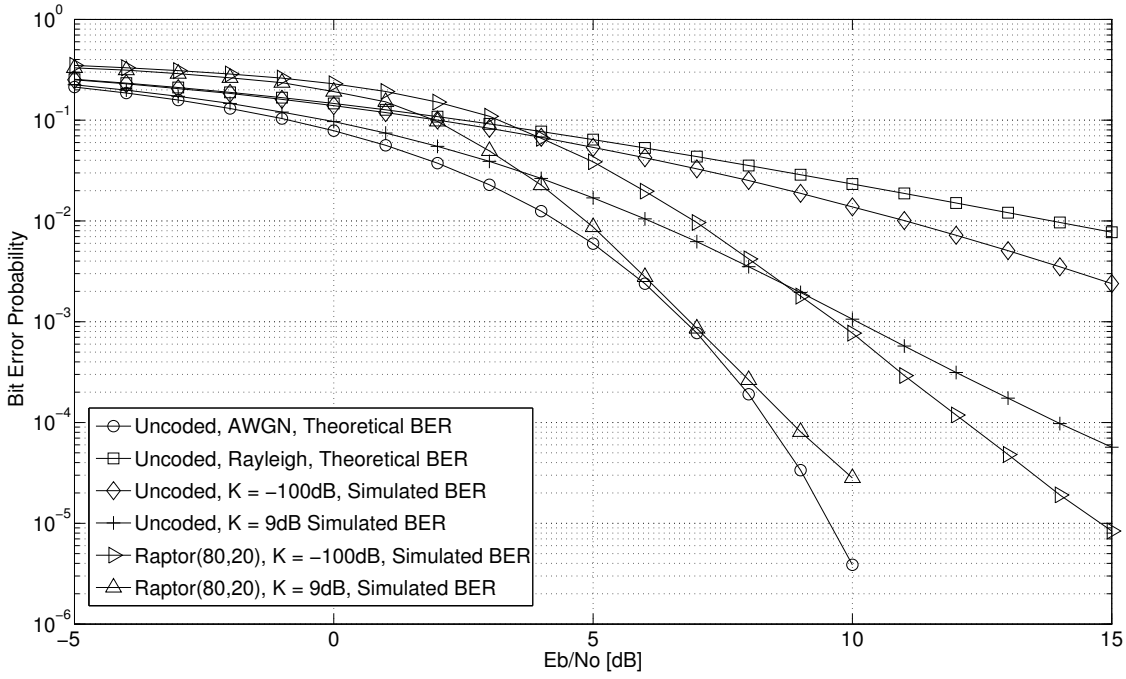

Figure 5.39: BER performance of a Raptor(80,20) coded QPSK communication system on a flat fading channel [BP decoding], $B_D = 100Hz$

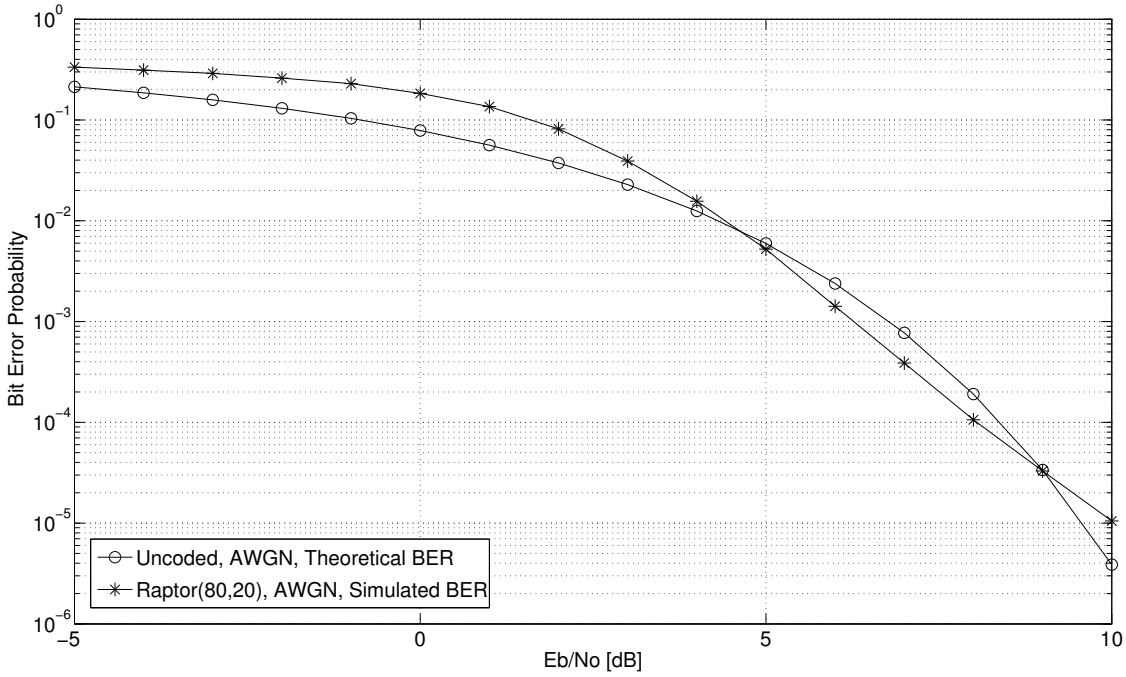

Figure 5.40: BER performance of a Raptor(80,20) coded QPSK communication system on a AWGN channel [BP decoding]

5.2.7.8 DISCUSSION OF RESULTS: LDPC, LT AND RAPTOR

The maximum coding gain and cross over point of each fountain code (compared with uncoded QPSK) are displayed in Tables 5.8 and 5.9 and also in Figures 5.41 and 5.42

Table 5.8: Coding gain obtained by fountain codes when compared to uncoded QPSK

Fountain Code	-100dB @33Hz	9dB @33Hz	-100dB @100Hz	9dB @100Hz	AWGN
LDPC [dB]	5.2	4.8	5.8	6.4	1.2
LT [dB]	6	4.5	5.5	4.5	1
Raptor [dB]	6.5	6	6.5	5.6	0.5
Tornado$_f$ [dB]	6	6.2	6	7	2.8

Table 5.9: Cross over point of fountain codes, compared to uncoded QPSK

Fountain Code	-100dB @33Hz	9dB @33Hz	-100dB @100Hz	9dB @100Hz	AWGN
LDPC [dB]	4	2	3	2	2.3
LT [dB]	1	1.6	2	1.6	2
Raptor [dB]	4	3.5	4	3.5	4.5
Tornado$_f$ [dB]	2	1	2	1	1

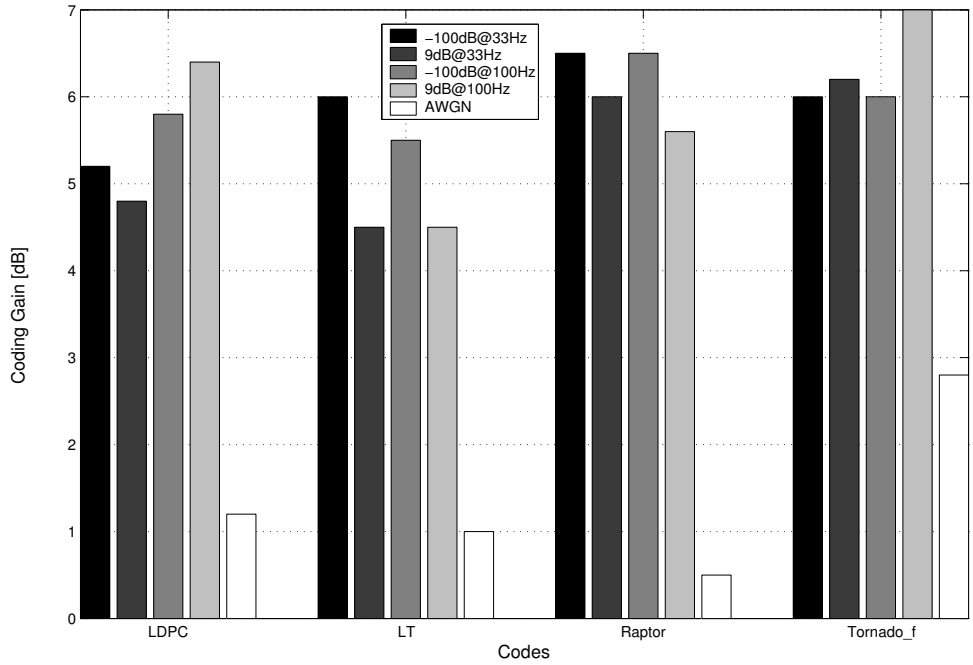

Figure 5.41: Table 5.8 represented graphically

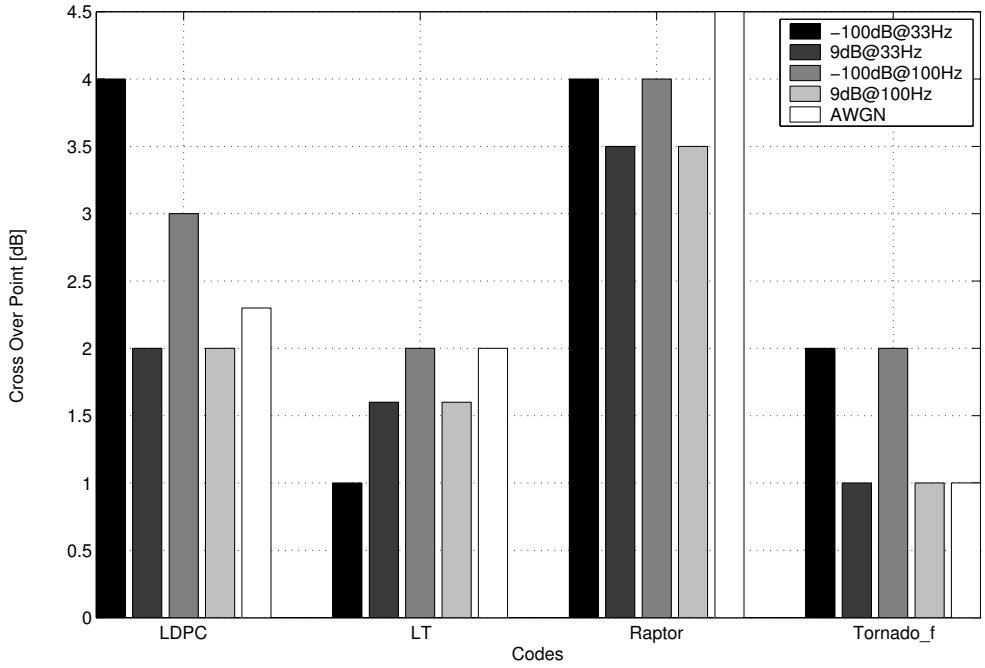

Figure 5.42: Table 5.9 represented graphically

The most important results pertaining to fountain codes include:

1. The LDPC (40,20) code performs better than LT (80,40) except for extreme fading conditions (-100 dB @ 33 Hz). The LT (80,40) performs marginally worse than LDPC (40,20) for the other channel conditions, but begins to deteriorate under moderate fading conditions due to an error floor. The cross over profile of the LT (80,40) code does however perform better than the LDPC (40,20) code by almost 1 dB on average. The LT (80,40) and the LDPC (40,20) codes don't perform well under AWGN conditions if only 10 iterations of BP are applied.

2. The Raptor (80,20) code outperforms its constituent codes under extreme fading conditions but deteriorates quickly under moderate fading conditions. It starts performing worse under fading conditions 9 dB @ 100 Hz. The performance increase under extreme fading conditions do come with a drawback; a higher cross over profile (1.25 dB worse than the LDPC (40,20) code). The Raptor (80,20) code performs extremely poor under AWGN conditions (only a maximum of 0.5 dB improvement over an uncoded scheme is achieved).

3. The three codes mentioned were only compared to the best Tornado code. All codes performed worse than Tornado $(60,40)_f$ except for the Raptor (80,20) code

that outperforms the Tornado code under extreme fading conditions. The cross over profile of the Tornado code is better than all three codes mentioned previously.

4. Although the Tornado code outperforms the other codes, especially on the AWGN channel, it can not be generalized. The LT code used has a bad distribution since $K << 10000$ and therefore will not perform as proficient as it should, hampering the Raptor code as well. The LDPC code used is regular, while the Tornado code is irregular. See Chapter 6 for more detailed conclusions.

5.3 SIMULATION: EDGE PLATFORM

In this section the second simulation platform is introduced followed by the results of implementing a fountain IR scheme on this platform.

5.3.1 EDGE PLATFORM

The EDGE platform was discussed in Chapter 4. Some simulation specific parameters need to be defined so the results can be repeated; these parameters are given in Table 5.10.

Table 5.10: Parameters defining EDGE platform used for simulations

Parameter	**Value**
Maximum Doppler Frequency	41.6Hz
Fading	Frequency selective
Channel Model	Typically Urban
Co channel interference	None
Channel state information	Estimated (real world receiver)

The main idea simulated on the EDGE platform is the use of fountain codes as incremental redundancy (IR) scheme (see Section 4.4). As mentioned in Chapter 4 the current EDGE scheme uses convolutional codes. For the details of each MCS scheme see Sections 4.4 and 4.5. For the fountain approach a systematic LT code was used to implement the IR scheme. The LT codes used for each MCS scheme are different. As

mentioned in Section 5.2.7.5 each LT code can be uniquely defined by the distribution (K, δ, c). The LT codes used for this simulation can all be described by $(x, 0.01, 0.06)$, where x differs for each MCS scheme. The value of x for each coding scheme can be found in Table 5.11

Table 5.11: The value of K for each MCS scheme

MCS	x
5	468
6	612
7	468
8	564
9	612

The first transmission consists of x message bits and $1248 - x$ parity bits in the case of MCS5 and MCS6. All other retransmissions send 1248 parity bits. In the case of MCS7, MCS8 and MCS9 two data payloads are sent, each consisting of 612 bits. For the first transmission one data payload consists of x message bits and $612 - x$ parity bits. During the second transmission only parity bits are sent. See Section 4.5 for the code rates obtainable. Belief propagation was used as decoding algorithm, 200 iterations.

5.3.2 SIMULATED BLER CURVES: EDGE

The results obtained via computer simulation of IR (EDGE system) using fountain and convolutional codes for each MCS scheme can be found in Figures 5.43 to 5.47.

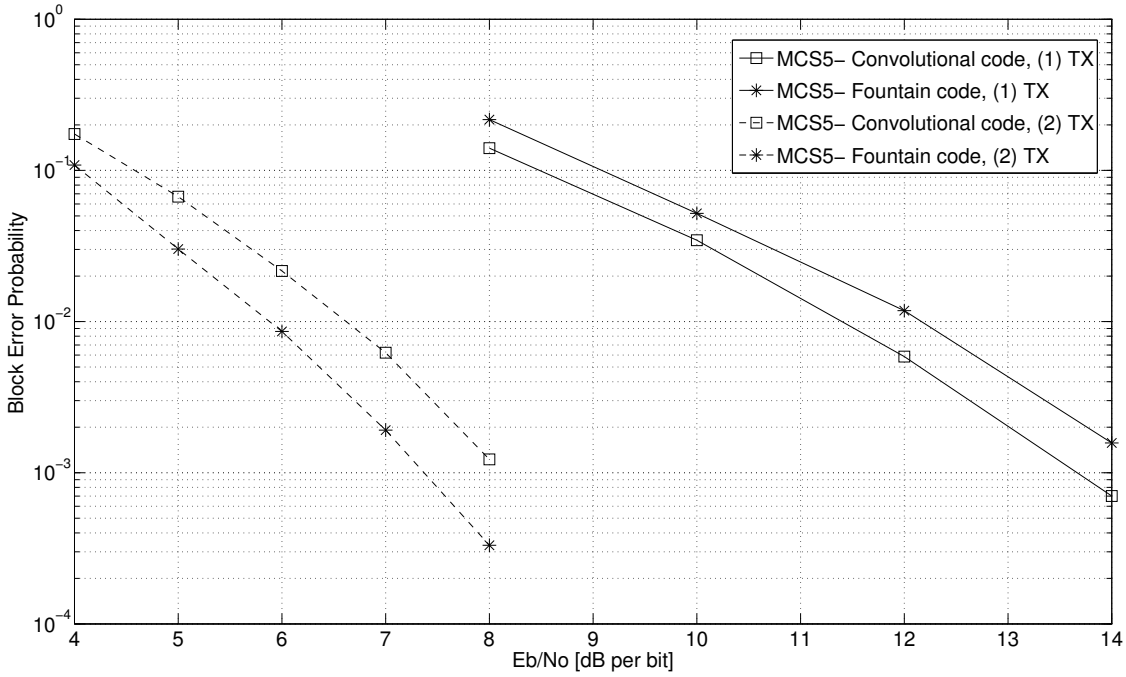

Figure 5.43: BLER performance of IR implemented on EDGE (MCS5) using convolutional and fountain codes

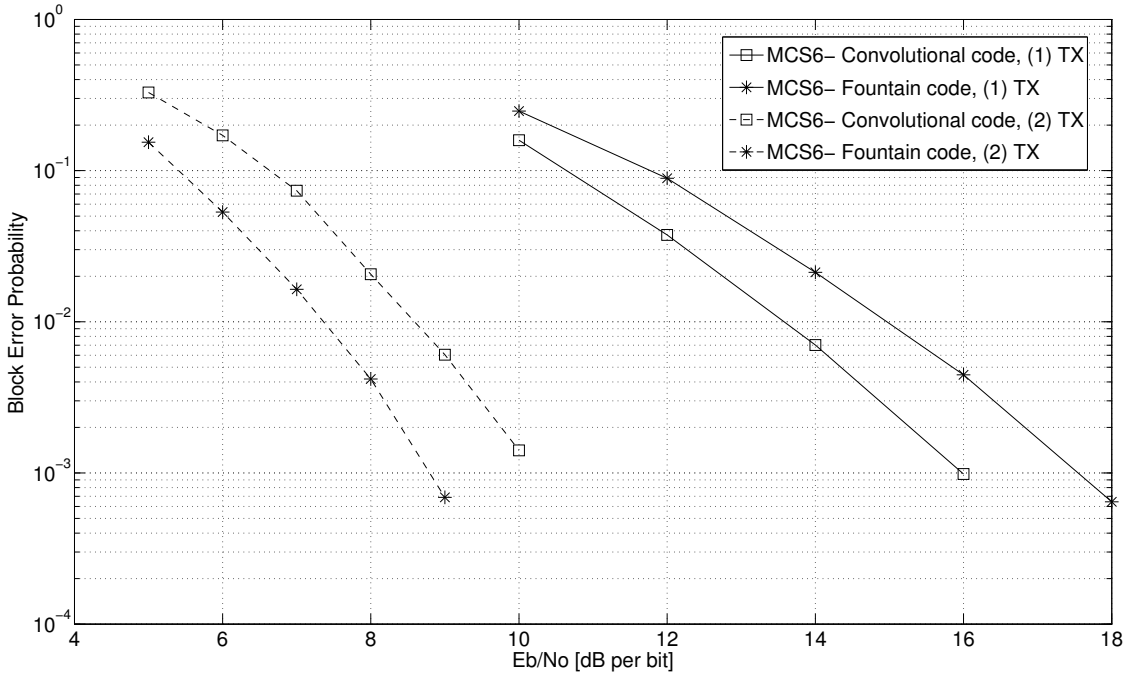

Figure 5.44: BLER performance of IR implemented on EDGE (MCS6) using convolutional and fountain codes

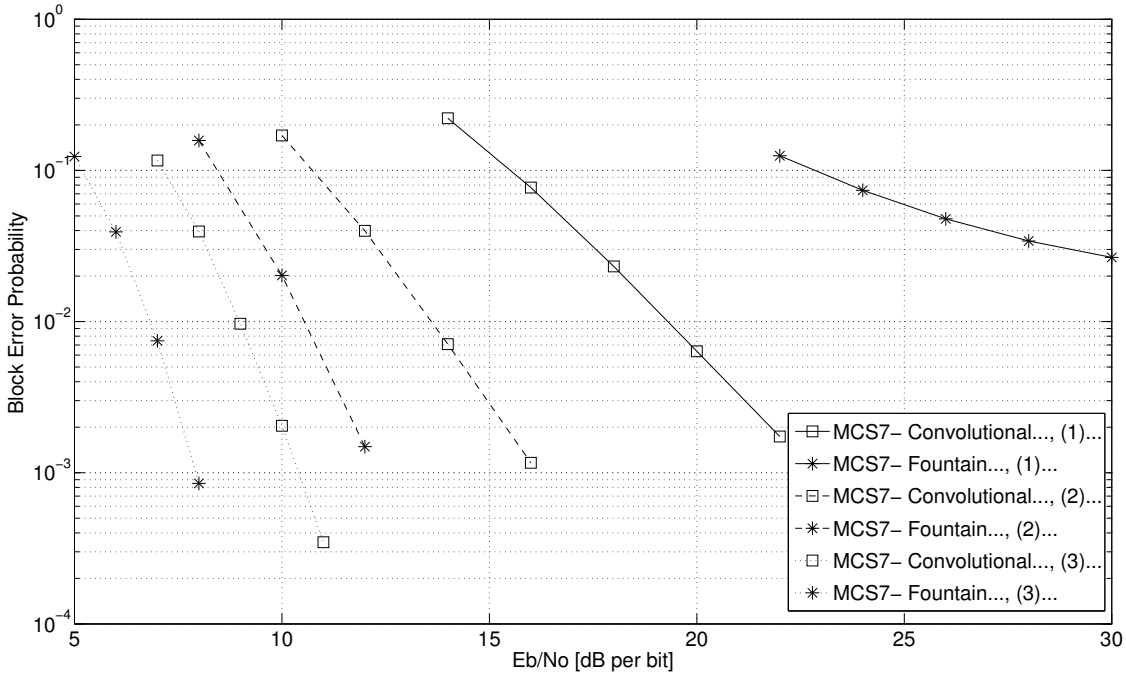

Figure 5.45: BLER performance of IR implemented on EDGE (MCS7) using convolutional and fountain codes

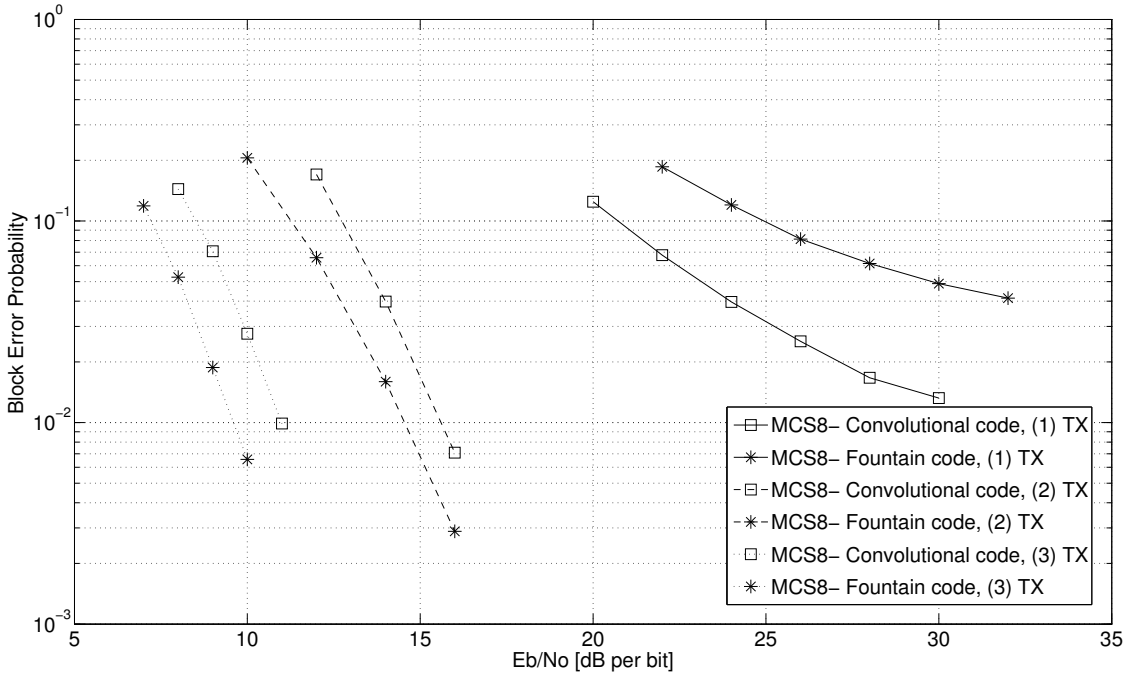

Figure 5.46: BLER performance of IR implemented on EDGE (MCS8) using convolutional and fountain codes

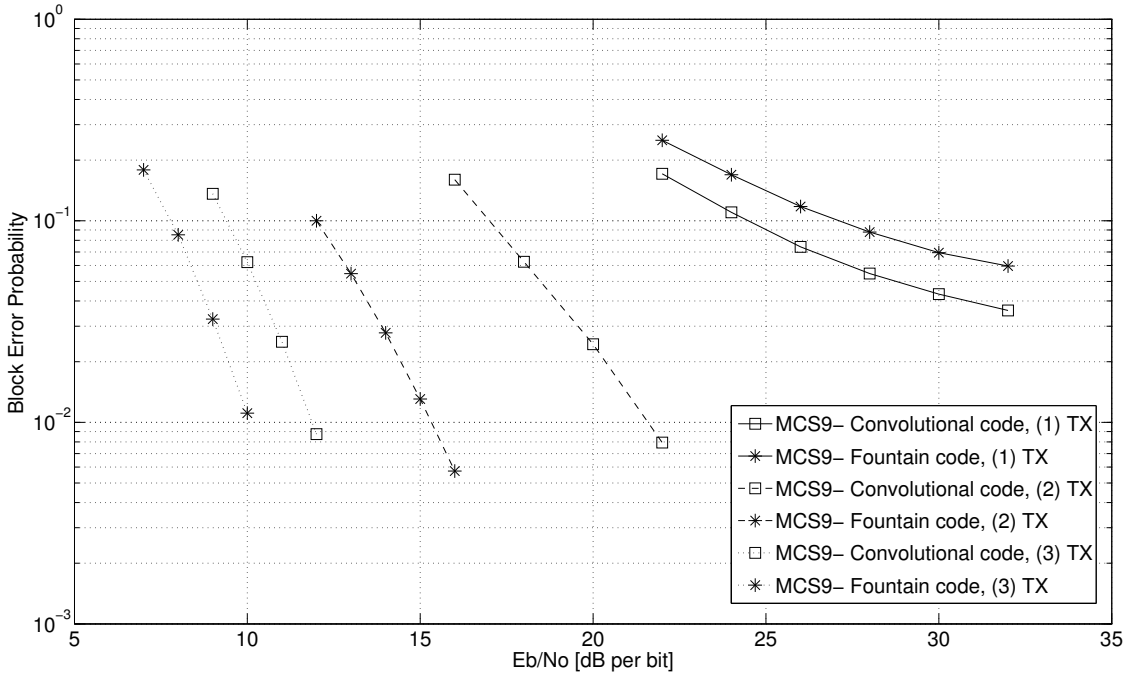

Figure 5.47: BLER performance of IR implemented on EDGE (MCS9) using convolutional and fountain codes

5.3.3 DISCUSSION OF RESULTS: EDGE

The actual dB improvement due to the fountain approach on each transmission for each MCS scheme can be found in Table 5.12.

Table 5.12: Improvement of fountain approach when compared to convolutional approach

MCS	Transmission 1 [dB]	Transmission 2 [dB]	Transmission 3 [dB]
5	-0.6	0.75	-
6	-1.75	1.5	-
7	-8	4	3
8	-4	1	1.2
9	-1.8	6	1.8

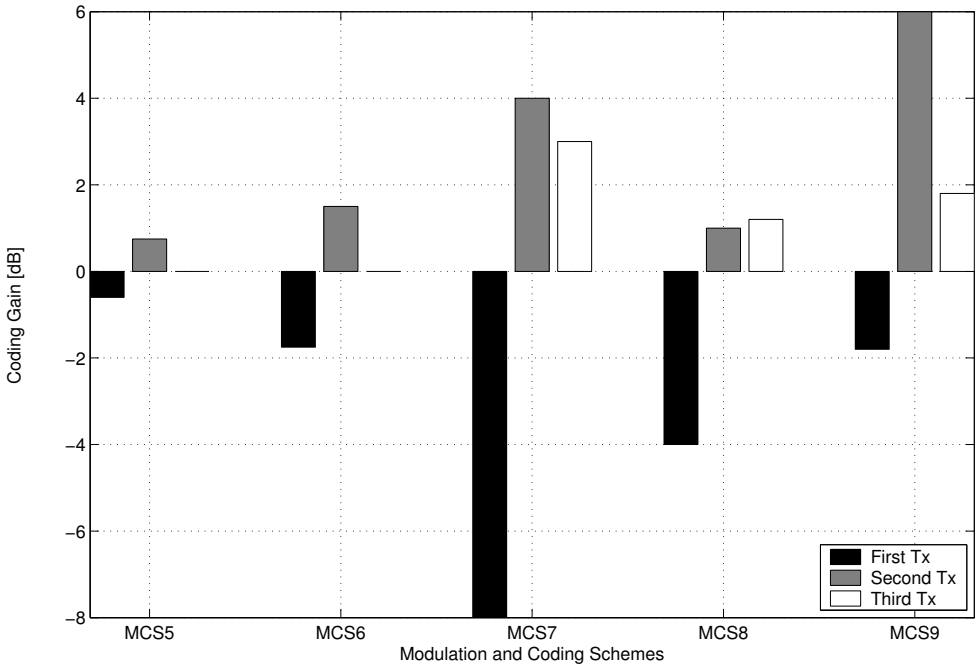

Figure 5.48: Table 5.12 represented graphically

The most important general observations that can be made from the simulations are:

1. Implementing IR on EDGE using the convolutional approach outperforms the fountain approach on the first transmission. This is true for all MCS schemes.

2. The fountain approach starts gaining over the convolutional approach during the second transmission, for all MCS schemes.

3. In the case of MCS5, MCS6 and MCS9 the fountain approach only performs marginally worse than the convolutional IR scheme during the first transmission.

CHAPTER 6

CONCLUSIONS

6.1 CHAPTER INTRODUCTION

This chapter briefly summarizes the main conclusions drawn from Chapter 5 in two sections namely Section 6.2 that discusses the conclusions obtained from the QPSK platform simulations and Section 6.3 that discusses the conclusions obtained from the EDGE platform simulations.

6.2 CONCLUSIONS: QPSK PLATFORM

The most important findings of the QPSK simulations include the following:

1. The most important result from decoding simple codes (see Section 5.2.4) is that different codes (block and convolutional) can be decoded with one type of decoder, namely a BP decoder (depending on the size and density of the factor graph used). Further merging this result with the results from the Raptor code (see below) implies that many codes can be combined into one factor graph and decoded using one graph structure. This is even true if a block and convolutional code are combined. As can be seen from Section 5.2.4.3 a convolutional code can be represented by a factor graph. The factor graph of a convolutional code can thus easily be combined with an LT code (the same procedure will be used as in the case of a Raptor code, see Section 5.2.7.6).

2. The main result from the simulations performed on the Tornado variations (see Section 5.2.7.1) is that the Tornado graph structure is not suitable for noisy decoding. What happens is that the individual layers perform better than the Tornado (80,40) code. This is probably true due to the fact that the second layer doesn't really protect the original message bits, but the first layer of parity. This is supported by the fact that the first parity layer has an extremely low BER. The entire graph works to protect the first parity layer, instead of protecting the original message bits (see Sections 5.2.7.2 and 5.2.7.3).

3. The most important results obtained by simulating these fountain codes (see Section 5.2.7) are the improvement of the systematic Raptor code (under extreme fading conditions) over its constituent codes and the verification of systematic LT decoding (see Sections 5.2.7.4, 5.2.7.5, 5.2.7.6, 5.2.7.7 and 5.2.7.8). Improved Raptor decoding implies that combining codes into one graph and decoding can lead to improved codes. The fact that the Raptor code performs worse under moderate fading conditions implies that the weaknesses of the constituent codes may also be amplified by code combination. However the main result is that with research (density evolution) and specific codes a stronger code may be created from weaker codes.

6.3 CONCLUSIONS: EDGE PLATFORM

The main conclusions and improvements regarding the EDGE platform simulations are discussed below (see Section 5.3).

6.3.1 MAIN CONCLUSIONS

The following conclusions can be drawn from the observations gathered from the simulation results of the EDGE platform (See Sections 5.3.2 and 5.3.3):

1. The reason the first transmission of MCS5 and MCS6 using the fountain IR approach performs only marginally worse, is because these schemes only have one data payload per MAC/RLC block (see Section 4.3). This implies longer codewords than MCS7, MCS8 and MCS9. Fountain codes perform better with longer codewords. They perform especially well when their code rate $R_c < 0.5$. This is why MCS5 is closer to the convolutional approach than MCS6, due to an original lower code rate (see Section 4.5).

2. MCS6 shows a larger improvement than MCS5 during the second transmission probably due to a larger K (see Section 3.4). LT codes perform better for larger values of K, as long as the code rate $R_c < 0.5$. It can also be due to the fact that the convolutional code performs worse in the case of MCS6.

3. MCS7 and MCS8 perform poorly on the first transmission if fountain codes are used due to very small code lengths. MCS7, MCS8 and MCS9 have two data payloads per RLC/MAC block implying extremely small code lengths (see Section

4.3). Most sparse graph codes perform poorly if the code length is small. In the case of the systematic LT code, the code can not cover all of its message bits with so few parity bits.

4. For MCS9 the code rate is 1 (see Section 4.3), which makes the uncoded scheme almost as good as using a punctured convolutional code. This is the reason for the fountain approach only performing marginally worse (since the code rate is 1 only message bits are sent during the first transmission if the fountain approach is used).

5. The improvements of MCS7 and MCS9 on retransmissions can be compared with the improvements of MCS5 and MCS6 respectively.

6.3.2 IMPROVEMENTS

The following improvements are possible:

1. The systematic LT code used, can be redesigned by using density evolution.

2. The systematic LT code can be replaced with a systematic Raptor code. Now for the first transmission one can use a specially designed LDPC code (using density evolution). The LT codes used for retransmissions can also be redesigned using density evolution (for a specific channel). This approach ought to work, when compared to the results obtained from the systematic Raptor code.

3. The frame lengths can be made larger. If larger frame lengths are used the fountain code will always outperform the convolutional approach.

4. The above improvements will only work on MCS5 and MCS6. For MCS7, MCS8 and MCS9 the only possible way to improve the results is to use some sort of CC as pre-code and an LT code as inner code. These two graphs need to be combined for decoding.

6.3.3 FINAL NOTES

The final conclusions are:

1. The convolutional IR scheme has good coding gain for smaller frame lengths.

2. It can be shown that the fountain approach will outperform the CC scheme for larger block sizes.

3. The fountain approach has an endless amount of retransmissions and uses all of the resent bits. CC has some repetitions and can not have an endless amount of retransmissions.

4. The fountain decoding structure grows larger after each transmission, slowing decoding.

5. The fountain approach has potential, and if the frame lengths are large enough it may be the default choice for an IR scheme.

6.4 SUMMARY OF WORK

This dissertation focused on noisy decoding of fountain codes using belief propagation decoding. Fountain codes were originally developed for erasure channels, but since any factor graph can be decoded using belief propagation, noisy decoding of fountain codes can easily be accomplished. Three fountain codes namely Tornado, Luby Transform (LT) and Raptor codes were investigated during this dissertation. The following results were obtained:

1. The Tornado graph structure is unsuitable for noisy decoding since the code structure protects the first layer of parity instead of the original message bits (a Tornado graph consists of more than one layer).

2. The successful decoding of systematic LT codes were verified.

3. A systematic Raptor code was introduced and successfully decoded. The simulation results show that the Raptor graph structure can improve on its constituent codes (a Raptor code consists of more than one code).

Lastly an LT code was used to replace the convolutional incremental redundancy scheme used by the 2G mobile standard Enhanced Data Rates for GSM Evolution (EDGE). The results show that a fountain incremental redundancy scheme outperforms a convolutional approach if the frame lengths are long enough. For the EDGE platform the results also showed that the fountain incremental redundancy scheme outperforms the convolutional approach after the second transmission is received. Although EDGE

is an older technology, it still remains a good platform for testing different incremental redundancy schemes, since it was one of the first platforms to use incremental redundancy.

6.5 CRITICAL EVALUATION AND FUTURE WORK

The results of this dissertation show that fountain codes have great potential as incremental redundancy scheme. Unfortunately for EDGE the frame sizes are too small to use effectively as incremental redundancy scheme. For other platforms using larger frame lengths this approach will work well. Further coding gains can also be obtained by redesigning the LT code for the specific channel using density evolution.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, July and October 1948.

[2] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, April 1950.

[3] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, September 1959.

[4] R. Bose and D. Ray-Chaudhari, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, March 1960.

[5] R. Bose and D. Ray-Chaudhari, "Further results on error correcting binary group codes," *Information and Control*, vol. 3, pp. 279–290, September 1960.

[6] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society of Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, June 1960.

[7] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[8] R. G. Gallager, "Low-density parity-check codes," Ph.D. thesis, M.I.T., Cambridge, Massachusetts, 1963.

[9] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, August 1996.

[10] E. R. Berlekamp, "Nonbinary BCH decoding," *IEEE Transactions on Information Theory*, vol. 14, no. 2, pp. 242–242, March 1968.

[11] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.

[12] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes using a trellis," *IEEE Transactions on Information Theory*, vol. 24, pp. 76–80, January 1978.

[13] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchial approach," in *Proceedings American Association of Artificial Intelligence National Conference on AI*, Pittsburg, USA, August 1982, pp. 133–146.

[14] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*, 1st ed. Cambridge, UK: Cambridge University Press, 2005.

[15] P. Elias, "Coding of noisy channels," *IRE Convention Record*, vol. 4, pp. 37–47, March 1955.

[16] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, March 1974.

[17] G. D. Forney, *Concatenated Codes*, 1st ed. Cambridge, Massachusets, USA: MIT Press, 1966.

[18] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proceedings of 1993 IEEE International Conference on Communications*, vol. 2, Geneva, Switzerland, May 1993, pp. 1064–1070.

[19] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Transactions on Information Theory*, vol. 28, no. 1, pp. 55–67, January 1982.

[20] S. M. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 8, pp. 1451–1458, October 1998.

[21] L. Staphorst, "Viterbi decoded linear block codes for narrowband and wideband communication over mobile fading channels," Master's dissertation, University of Pretoria, South Africa, 2005.

[22] J. D. Vlok, "Sparse graph codes on a multi-dimensional WCDMA platform," Master's dissertation, University of Pretoria, South Africa, 2007.

[23] T. Nguyen, L. Yang, and L. Hanzo, "Systematic Luby Transform codes and their soft decoding," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Shanghai, China, October 2007.

[24] M. Hakaste, E. Nikula, and S. Hamiti, *GSM, GPRS and EDGE Performance*, 2nd ed. John Wiley & Sons, Ltd., 2003.

[25] R. V. Nobelen and N. Seshadri, "Incremental redundancy transmission for EDGE," *ETSI SMG 2*, August 1998.

[26] T. L. Grobler, J. C. Olivier, and J. D. Vlok, "Fountain codes and their possible application in standards like GSM," in *Southern African Telecommunication Networks and Applications Conference*, Mauritius, September 2007.

[27] F. R. Kschischang, B. J. Frey, and H. Loeliger, "Factor graphs and the sum product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[28] S. M. Moser, "Investigation of algebraic codes of small length using factor graphs," Master's dissertation, Labaratory of Signal and Information Processing, 1999.

[29] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, September 1981.

[30] R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, 1st ed. Chichester, UK: John Wiley & Sons Limited, 2004.

[31] P. Elias, "Coding for two noisy channels," *Information Theory, 3rd London Symp*, pp. 61–67, 1955.

[32] J. Byers, M. Luby, and M. Mitzenmacher, "A Digital Fountain Approach to Asynchronous Reliable Multicast," *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 8, pp. 1528–1540, October 2002.

[33] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of the ACM SIGCOMM'98 Conference*, Vancouver, Canada, August 1998, pp. 56–67.

[34] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination based data delivery using broadcast disks," *IEEE Personal Communications Magazine*, vol. 2, pp. 50–60, December 1995.

[35] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, April 1997.

[36] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An xor-based erasure-resilient coding scheme," International Computer Science Institute, Berkeley, California, Tech. Rep., 1995.

[37] M. Mitzenmacher, "Digital fountains: A survey and look forward," in *IEEE Information Theory Workshop*, San-Antonio, Texas, October 2004, pp. 271–276.

[38] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 569–584, February 2001.

[39] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proceedings of 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, USA, January 1998, pp. 364–373.

[40] M. Luby, "LT codes," in *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Vancouver, Canada, 2002, pp. 271–282.

[41] D. J. C. MacKay, "Fountain codes," *IEE Proceedings - Communication*, vol. 152, no. 6, pp. 1062–1068, December 2005.

[42] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, June 2006.

[43] O. Etesami and A. Shokrollahi, "Raptor codes on binary memoryless symmetric channels," *IEEE Transactions on Information Theory*, vol. 52, no. 5, pp. 2033–2051, May 2006.

[44] R. Karp, M. Luby, and A. Shokrollahi, "Verfication decoding of raptor codes," in *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, Adelaide, Australia, September 2005, pp. 1310–1314.

[45] R. Palanki and J. Yedidia, "Rateless codes on noisy channels," in *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, Chicago, USA, 2004, p. 37.

[46] T. Stockgammer, H. Jenkac, T. Mayer, and W. Xu, "Soft decoding of LT-codes for wireless broadcast," in *Proceedings of the IST Mobile and Wireless Communications Summit*, Dresden, Germany, 2005.

[47] K. Borgwardt, *The Simplex Method: A Probabilistic Analysis*. Springer, January 1987.

[48] R. Tee, T. Nguyen, L. Yang, and L. Hanzo, "Serially concatenated luby transform coding and bit-interleaved coded modulation using iterative decoding for the wireless internet," in *Proceedings of IEEE Vehicular Technology Conference*, Montreal, Canada, 2006, pp. 22–26.

[49] M. H. Ismail, H. M. Mourad, and M. El-Soudani, "A newproposal for Link Adaptation in EDGE system based on the use of Turbo codes," in *Proceedings of the Twentieth National Radio Science Conference*, Cairo, Egypt, March 2003.

[50] Anon., "EDGE," Vocal Technologies, Ltd., Tech. Rep. EDGE-0004A-1, 2001. [Online]. Available: http://www.vocal.com/data_sheets/full/edge.pdf

[51] J. G. Proakis, *Digital Communications*, 4th ed. Boston, Massachusetts, USA: McGraw-Hill, 2001.

[52] S. Nanda, K. Balachandran, and S. Kumar, "Adaptation Techniques in Wireless Packet Data Services," *IEEE Communications Magazine*, January 2000.

[53] Anon., "The evolution of EDGE," Erricson, Tech. Rep. 285 23-3107 Uen Rev A, February 2007. [Online]. Available: http://www.erricson.com/technology/whitepapers/3107_The_evolution_of_EDGE_A.pdf

[54] S. Vialle and J. Boutres, "A Gallager-Tanner construction based on convolutional codes," Motorola Research Center, Paris, ENST, Communications and Electronic Department, Tech. Rep., December 1998.

LIST OF FIGURES

LIST OF TABLES