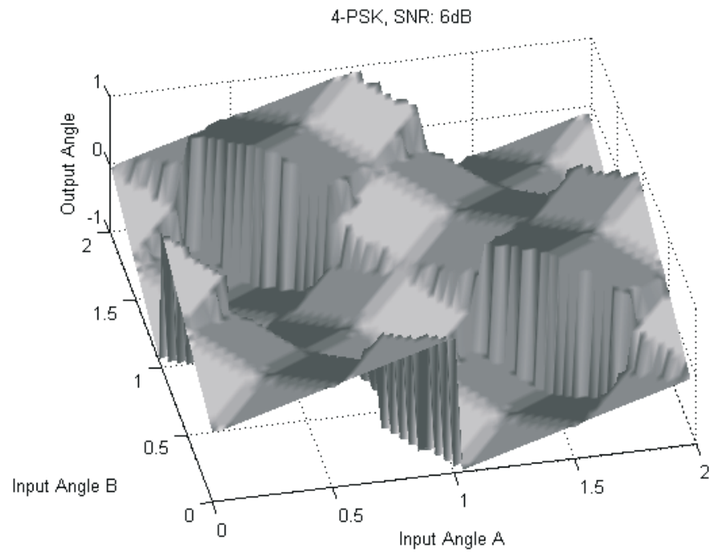
Efficient Message Passing Decoding Using Vector-based Messages

MIKAEL GRIMNELL, MATS TJÄDER



FOI is an assignment-based authority under the Ministry of Defence. The core activities are research, method and technology development, as well as studies for the use of defence and security. The organization employs around 1350 people of whom around 950 are researchers. This makes FOI the largest research institute in Sweden. FOI provides its customers with leading expertise in a large number of fields such as security-policy studies and analyses in defence and security, assessment of different types of threats, systems for control and management of crises, protection against and management of hazardous substances, IT-security an the potential of new sensors.

FOI
Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

Phone: +46 13 37 80 00
Fax: +46 13 37 81 00

www.foi.se

FOI-R--1963--SE
ISSN 1650-1942

Scientific report
February 2006

Command and Control Systems

Mikael Grimnell, Mats Tjäder

Efficient Message Passing Decoding Using Vector-based Messages

FOI-R--1963--SE
ISSN 1650-1942

Scientific report
February 2006

Command and Control Systems

Issuing organization	Report number, ISRN	Report type
FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	FOI-R--1963--SE	Scientific report
	Research area code 4. C4ISTAR	
	Month year February 2006	**Project no.** I7068
	Sub area code 41 C4I	
	Sub area code 2	

Author/s (editor/s)	Project manager
Mikael Grimnell Mats Tjäder	Hugo Tullberg
	Approved by Martin Rantzer
	Sponsoring agency
	Scientifically and technically responsible Hugo Tullberg

Report title

Efficient Message Passing Decoding Using Vector-based Messages

Abstract

Low-Density Parity-Check (LDPC) codes provide strong error correction capability. Early LDPC-work concentrated on the binary Galois Field, GF(2), but here we consider higher order alphabets, GF(q). The code symbols from GF(q) are mapped to M-ary Phase Shift Keying (M-PSK) signals to yield higher spectral efficiency.

LDPC codes are commonly decoded using the Message Passing (MP) algorithm, which is an iterative algorithm that passes messages between nodes in a graph representation of the code. Unfortunately, the computational complexity of the optimal MP decoder, the Belief Propagation (BP) decoder, scales as the square of the order of the used Galois Field.

To reduce complexity, we investigate a number of simplified MP decoders. Since the information of a PSK signal is found in the phase angle, geometrical vectors and angles are used as messages in the decoders.

A promising simplification is the Table Vector Decoder, which approximates the check node operation of a BP decoder by table lookup. The complexity of table-based decoders is unaffected by size of the used Galois Field. For well-designed tables, the table-based decoders suffer only minor losses compared to the optimal Belief Propagation (BP) decoders.

We also investigate the theoretical decoding properties using Density Evolution analysis.

Keywords

LDPC codes, M-PSK modulation, Message Passing decoding, Belief Propagation decoding, geometrical vectors, low-complexity decoding, Density Evolution analysis

Further bibliographic information	Language English

Also published as Master's Thesis Report LiTH-ISY-EX-05/3741--SE at Linköping University

ISSN 1650-1942	Pages 96 p.
	Price acc. to pricelist

Utgivare FOI - Totalförsvarets forskningsinstitut Ledningssystem Box 1165 581 11 Linköping	**Rapportnummer, ISRN** FOI-R--1963--SE	**Klassificering** Vetenskaplig rapport
	Forskningsområde 4. Ledning, informationsteknik och sensorer	
	Månad, år Februari 2006	**Projektnummer** I7068
	Delområde 41 Ledning med samband och telekom och IT-system	
	Delområde 2	
Författare/redaktör Mikael Grimnell Mats Tjäder	**Projektledare** Hugo Tullberg	
	Godkänd av Martin Rantzer	
	Uppdragsgivare/kundbeteckning	
	Tekniskt och/eller vetenskapligt ansvarig Hugo Tullberg	

Rapportens titel

Effektiv meddelandebaserad avkodning med vektormeddelanden

Sammanfattning

Low-Density Parity-Check- (LDPC-) koder har stark felrättningsförmåga. Tidigare arbete fokuserade på den binära talkroppen, GF(2), men här undersöker vi högre ordningens talkroppar, GF(q). Kodade symboler från GF(q) moduleras på M-är Phase Shift Keying (M-PSK) för att få högre spektraleffektivitet.

LDPC-koder avkodas vanligen med Message Passing- (MP-) algoritmen, en iterativ algoritm där meddelande utväxlas mellan noderna i kodens grafrepresentation. Tyvärr beror beräkningskomplexiteten för den optimala MP-avkodaren, Belief Propagation (BP), som kvadraten på ordningen av den använda talkroppen.

För att minska komplexiteten har ett antal förenklade MP-avkodare undersökts. Eftersom informationen hos en PSK-signal överförs i fasvinkeln, har geometriska vektorer och vinklar använts som meddelanden i avkodarna.

Den mest lovande förenklingen är Vektortabellavkodaren, vilken approximerar operationen i check-noden hos en BP-avkodare med en tabellsökning. Komplexiteten hos tabellbaserade avkodare är oberoende av den använda talkroppens ordning. För välberäknade tabeller lider tabellbaserade avkodare endast mindre prestandaförluster jämfört med den optimala BP-avkodaren.

Vi undersöker också kodernas teoretiska egenskaper med hjälp av Density Evolution-analys.

Nyckelord

LDPC-koder, M-PSK-modulation, Meddelandebaserad avkodning, Belief Propagation-avkodning, geometriska vektorer, lågkomplexitetsavkodning, Density Evolution-analys

Övriga bibliografiska uppgifter	**Språk** Engelska

Även publicerad examensarbetesrapport LiTH-ISY-Ex-05/3741--SE vid Linköpings universitet

ISSN 1650-1942	**Antal sidor:** 96 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Abstract

The family of Low Density Parity Check (LDPC) codes is a strong candidate to be used as Forward Error Correction (FEC) in future communication systems due to its strong error correction capability. Most LDPC decoders use the Message Passing algorithm for decoding, which is an iterative algorithm that passes messages between its variable nodes and check nodes. It is not until recently that computation power has become strong enough to make Message Passing on LDPC codes feasible. Although locally simple, the LDPC codes are usually large, which increases the required computation power. Earlier work on LDPC codes has been concentrated on the binary Galois Field, GF(2), but it has been shown that codes from higher order fields have better error correction capability. However, the most efficient LDPC decoder, the Belief Propagation Decoder, has a squared complexity increase when moving to higher order Galois Fields. Transmission over a channel with M-PSK signalling is a common technique to increase spectral efficiency. The information is transmitted as the phase angle of the signal.

The focus in this Master's Thesis is on simplifying the Message Passing decoding when having inputs from M-PSK signals transmitted over an AWGN channel. Symbols from higher order Galois Fields were mapped to M-PSK signals, since M-PSK is very bandwidth efficient and the information can be found in the angle of the signal. Several simplifications of the Belief Propagation has been developed and tested. The most promising is the Table Vector Decoder, which is a Message Passing Decoder that uses a table lookup technique for check node operations and vector summation as variable node operations. The table lookup is used to approximate the check node operation in a Belief Propagation decoder. Vector summation is used as an equivalent operation to the variable node operation. Monte Carlo simulations have shown that the Table Vector Decoder can achieve a performance close to the Belief Propagation. The capability of the Table Vector Decoder depends on the number of reconstruction points and the placement of them. The main advantage of the Table Vector Decoder is that its complexity is unaffected by the Galois Field used. Instead, there will be a memory space requirement which depends on the desired number of reconstruction points.

Table of Contents

List of Symbols

Θ	Channel noise
σ	Noise variance
μ_i	i-th symbol point
B	Channel bandwidth
C	Channel capacity
d_i^2	The geometrical squared Euclidian distance
E	Signal energy
f_{symbol}^{0}	Pdf for the zero symbols from the channel
G	Generator matrix
H	Parity check matrix
M	Alphabet size
$\overline{m}$	Received channel vector
$\mathbf{m_i}$	Message Passing edge message
N	Number of reconstruction points
N_0	Channel noise nower
P_i	Reconstruction probability vector point
R	Rate
r_i	Reconstruction vector point
SNR	Signal to Noise Ratio
$SNR_{Threshold}$	Signal to Noise Ratio threshold
T	Symbol time
u	Inbound check node message
v	Inbound variable node message

1 Introduction

The family of Low Density Parity Check (LDPC) codes is a strong candidate to be used as Forward Error Correction (FEC) in future communication systems due to its strong error correction capability. Most LDPC decoders use the Message Passing algorithm for decoding, which is an iterative algorithm that passes messages between its variable nodes and check nodes. It is not until recently that computation power has become strong enough to make Message Passing on LDPC codes feasible. Although locally simple, the LDPC codes are usually large, which increases the required computation power. Earlier work on LDPC codes has been concentrated on the binary Galois Field, GF(2), but it has been shown that codes from higher order fields have better error correction capability. However, the most efficient LDPC decoder, the Belief Propagation Decoder, has a squared complexity increase when moving to higher order Galois Fields. In this thesis decoding with higher order Galois Fields and M-PSK signalling will be presented. Transmission over a channel with M-PSK signalling is a common technique to increase spectral efficiency. The information is transmitted as the phase angle of the signal.

1.1 Purpose

The focus in this Master's Thesis is on simplifying the Message Passing decoding when having inputs from M-PSK signals transmitted over an AWGN channel. Symbols from higher order Galois Fields were mapped to M-PSK signals, since M-PSK is very bandwidth efficient and the information can be found in the angle of the signal. A special case is the use of only angular information as messages in the Message Passing algorithm. If it is possible to decode a received codeword from the channel using only angular information for its M-PSK symbols with the MP algorithm, then the performance will be compared with the Belief Propagation Decoder using Monte Carlo simulations. The Belief Propagation Decoder will be used as a benchmarking Decoder in this thesis together with a more theoretical Density Evolution analysis of the developed decoders.

1.2 Methods and Sources

The methods used in this thesis are Density Evolution analysis and computer based Monte Carlo simulations. The programs and algorithms for these simulations and analysis have been developed in C++ and Matlab 7.0.

The sources used in the project have mainly been scientific articles and books on the subject. Some web based sources have also been used. A list of references can be found on page 95.

1.3 Structure of the Report

The report will first describe the theoretical background of the used methods and algorithms, both basic telecommunication theory, theory about LDPC codes and

Message Passing decoding. The analysis method Density Evolution will also be described in detail.

Next, the new, developed algorithms will be presented. After that, the simulation and analysis results of the new algorithms will be presented and discussed.

Finally, the results and conclusions of the simulations and analysis will be presented.

2 Theory and Background

A brief overview of the required background will be given in this section. Basic theory in the areas of information theory, telecommunication systems and error detection will be overviewed in 2.1, 2.2, and 2.3. Additional information can be found in many books on the topic, for instance [1], [3], or [4]. LDPC codes will be explained a bit more in-depth in Section 2.4.

2.1 Information Theory Concepts

In order to fully understand LDPC codes, it is necessary to be familiar with two fundamental concepts from information theory, namely Rate and Capacity.

Definition 1. The Rate, or Code Rate, is defined as the ratio between the information data transmitted and the total amount of data transmitted by the code [5]. When the code has a fixed length n and using an alphabet of size M, the rate R is defined as

$$R = \frac{\log M}{n}.$$

The logarithm is usually in base 2, which gives the rate in bits per channel symbol. M is commonly $M=2^k$, so the information symbol is a k-digit binary number [5].

Example 1. 100 information bits will be transmitted. The code adds 25 parity bits for error correction, so a total of 125 bits will be transmitted. The rate of the code is

$$R = \frac{100}{125} = 0.8.$$

The channel is the medium used for information transmission between a sender and a receiver. It does not necessary mean a physical channel, like a cable or a radio channel, but it can also be a channel spanning over time, for example a memory storage facility– it can also be a more abstract channel used for simulations and calculations, such as the Binary Erasure Channel or the Binary Symmetric Channel [4].

In this thesis, the channel will be defined as a discrete, memoryless channel, which can be described by the triple ($\mathcal{X}$, $\mathcal{Y}$, W) where $\mathcal{X}$ and $\mathcal{Y}$ are finite sets defining the input and output alphabets and W is a stochastic transfer matrix.

The channel capacity is explained as the highest possible mutual information between the sender and the receiver. That is, the more information that is known not to be distorted, the higher capacity the channel has. In mathematical terms, this can be described as

$$C(W) = \max_P \{I(P,W) : P \in P(X)\}$$

where W is a known matrix defining the channel, X is a stochastic input variable with the distribution $P(X)$. Before continuing, the entropy function H needs to be defined:

$$H(Y) = \sum_{y \in Y} P(y) \log P(y)$$

$$H(Y \mid X) = \sum_{y \in Y} P(y \mid X) \log P(y \mid X)$$

When the meaning of H is known, it will be possible to define *I(P,W)* as:

$$I(P,W) = I(X;Y) = H(Y) - H(Y \mid X)$$

I(P,W) and *I(X;Y)* are only different notations, one describing the mutual information using the channel matrix and probability distribution, the other using the stochastic input and output variables. For further details on the definition and calculation of channel capacity, see [4] or another book about information theory.

The relationship between the code rate R and channel capacity C is explained in the Channel Coding Theorem [1].

Definition 2. With every channel we can associate a "channel capacity" C. There exist error control codes such that information can be transmitted across the channel at rates less than C with arbitrary low bit error rate.

Bear in mind that this theorem is valid for a sufficiently large code, but does not state how large a code has to be in order to be sufficient. Also, it does not imply how this code may look like.

2.1.1 Additive White Gaussian Noise Channel

The Additive White Gaussian Noise (AWGN) channel is a very common channel used for computer simulations, and is the only channel used in this thesis. It is a memoryless channel that adds white, Gaussian noise with spectral density $N_0/2$ to the sent signal. A definition of white, Gaussian noise can be found in [2]. The received signal is

$$R(t) = s(t) + \Theta(t),$$

where *s(t)* is the sent signal, and *Θ(t)* is the noise.

The capacity of the AWGN channel has been derived in earlier works ([2]) and is only dependent on the bandwidth, B, and Signal to Noise Ratio, P/N_0

$$C_{AWGN} = B \log\left(1 + \frac{P}{N_0 B}\right) \text{ bits per second.}$$

2.2 Telecommunication Concepts

Telecommunication theory is dealing with sending information over a channel. This section will describe how the information is modulated to a signal, and how to detect the received signal. However, the first thing that will be dealt with is a simplified representation of signals, called the Vector Model. This model represents periodic signals in a very convenient way by using coordinates in a vector space instead of time-dependent functions.

2.2.1 Vector Spaces

The first thing that is needed for the Vector Model is an orthonormal (ON) base for the vectors. This base can be derived from a signal set using Gram-Schmidt Orthogonalization [5].

If the signal set has μ signals represented by time dependent functions, $\{s_i(t)\}_{i=1}^{\mu}$, and this signal set is spanning over an N-dimensional function space, then the signals can be linearly dependent. Gram-Schmidth Orthogonalization eliminates the dependencies and returns an independent ON base, $\{\phi_j(t)\}_{j=1}^{N}$, that can be used to represent the signals in the Vector Model.

Assuming $N \leq \mu$, choose N signals from the set

For i = 1,2,...,N, calculate

1) $s_{ij} = (s_i, \phi_j), \quad j \in \{1,2,\ldots,i-1\}.$

2) $g_i(t) = s_i(t) - \sum_{j=1}^{i-1} s_{ij}\phi_j(t)$

3) $\phi_i(t) = \frac{g_i(t)}{\|g_i\|}$

It can be noted that if only sinusoidal signals are used, then each frequency used can be represented as two dimensions, the In-phase (I) and Quadrature (Q) dimension. If only one frequency is used, it is possible to describe the vector space on a two-dimensional plot, the I/Q-plot. An illustration of an I/Q-plot of signals received over a 12dB AWGN channel using QPSK signalling (will be described in Section 2.2.2), is presented in Figure 2.1.

The angle and length of the vector is used often in this thesis. This is a representation equal to the I/Q values, and are the absolute length of the I/Q-vector and the angle counted counter-clockwise starting from the positive I-axis.

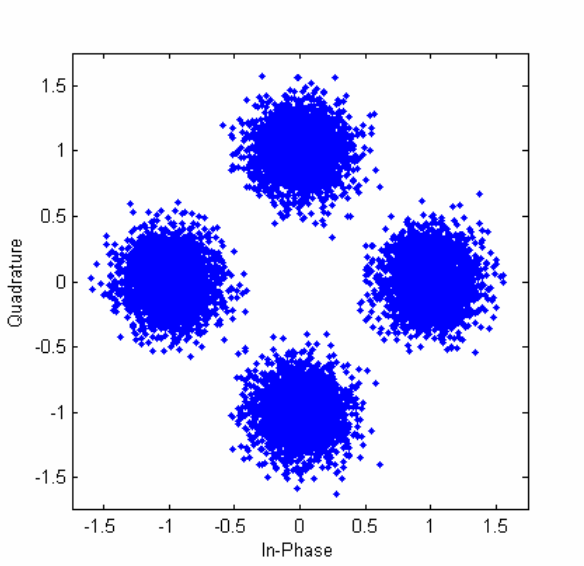


Figure 2.1 I/Q-plot of QPSK signalling over an 12dB AWGN channel.

2.2.2 Digital Modulation Techniques

There are many modulation techniques used in the telecommunications industry, so this introduction will only describe the ones used in this thesis. A comprehensive explanation, along with other modulation techniques, can be found in [5].

2.2.2.1 Binary Phase Shift Keying

Binary Phase Shift Keying (BPSK or 2-PSK) has two possible symbols (0 and 1) to transmit. The two signals representing the two possible symbols are simple sinusoidal signals with the only difference lying in their starting phases. Each symbol has a constant time interval, T (sending time).

$$s_1 = \begin{cases} \sqrt{\frac{2E}{T}}\cos(2\pi f_c T) & 0 \le t < T \\ 0 & elsewhere \end{cases}$$

$$s_2 = \begin{cases} \sqrt{\frac{2E}{T}}\cos(2\pi f_c T + \pi) & 0 \le t < T \\ 0 & elsewhere \end{cases}$$

E is the signal energy and f_c is the carrier frequency, chosen so $2f_cT$ is a positive integer. BPSK does not carry any information in the Quadrature dimension, which can be seen in Figure 2.2, so any noise in that dimension will not affect the detection (unless using a very bad detector).

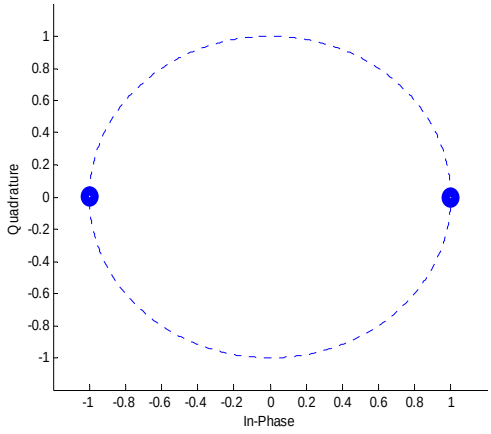


Figure 2.2 Illustration of the BPSK Constellation.

2.2.2.2 M-ary Phase Shift Keying

It is possible to extend BPSK to more than two starting phases. In this case, the signals can be represented as M signals where

$$s_i = \begin{cases} \sqrt{\frac{2E}{T}} \cos\left(2\pi f_c T + \frac{2\pi}{M} i\right) & 0 \le t < T \\ 0 & elsewhere \end{cases} \qquad i = 1 \ldots M$$

This thesis will mainly use Quadriphase Shift Keying (QPSK or 4-PSK), with 4 signals, and 8-PSK, with 8 signals. The I/Q-plots of QPSK and 8-PSK can be found in Figure 2.3 and Figure 2.4 respectively.

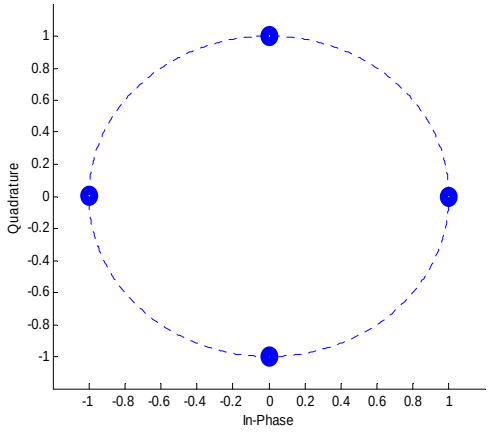


Figure 2.3 The QPSK (4-PSK) constellation.

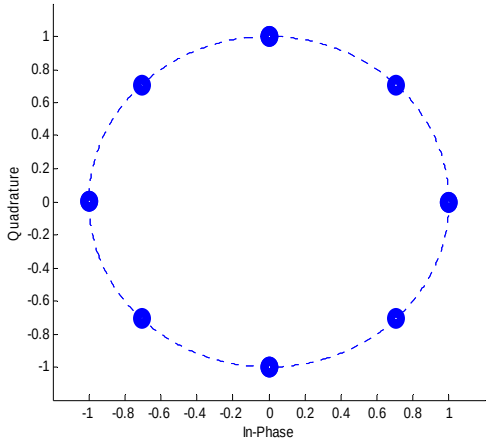


Figure 2.4 The 8-PSK constellation.

2.2.2.3 Additive White Gaussian Noise in the Vector Model

The noise that is added when sending a signal over an AWGN channel, can also be incorporated into the Vector Model. Since the noise is white and Gaussian, it will affect all dimensions equally. In the Vector Model, this can be represented as adding an *N*-dimensional noise vector (assuming that the signal has *N* independent dimensions) to the signal vector. The elements of the noise vector are one-dimensional, independent noise with a variance depending on the SNR for the channel. Figure 2.5 illustrates the differences between a sent signal, *S*, and a received signal, *R*, over an AWGN channel that adds a two-dimensional noise vector, Θ, to the signal. More information can be found in [5].

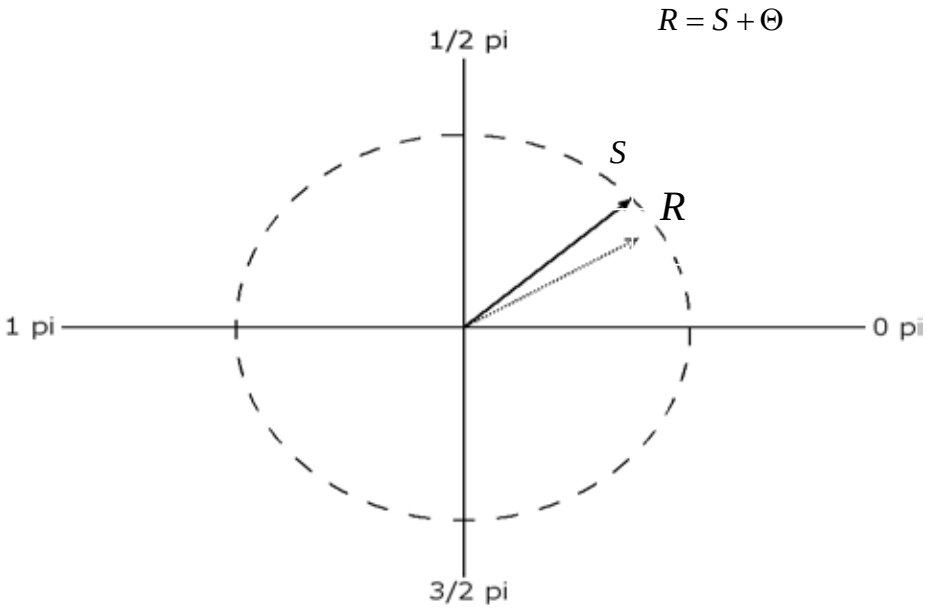


Figure 2.5 The effect of noise on 8-PSK signals.

2.2.3 Detection

This thesis does not consider detection problems, so instead of using the common filter bank detector [3], the detection is performed using the vector model for the received symbol.

According to Bayes' Rule, described in [6], the probability of deciding symbol s_i out of the possible set, $S \in \{s_i\}_{i=1}^{\mu}$, when receiving the signal $\overline{m}$ is

$$P(s_i) = \frac{p(\overline{m}|s_i)P(s_i)}{\sum_{j=1}^{\mu} p(\overline{m}|s_j)P(s_{ji})}.$$

Of the μ different symbols, the Maximum Likelihood Detector will decide the sent symbol/signal to be

$$\hat{s} = \max_{s \in S}\{P(s_i)\}.$$

2.2.3.1 Soft Decisions

If a receiver is capable of deliver some kind of reliability information about its decision, then the decision is called 'soft'. In this thesis, the soft decision used is quite simple: sending directly the I and Q values, or equally length and angle, of the received signal vector.

2.3 Error Correcting Codes

A good method for modulation and demodulation of signals is important to a communication system. However, it does not stop errors to occur. When transmitting over a channel, it is inevitable that sooner or later a symbol will be detected as another symbol than the one sent. Error correcting codes can lower the risk of errors even further. The symbols are coded using an error correcting code before transmitted, and the received symbols can then be decoded and errors can be detected or corrected up to a certain limit.

This thesis is dealing with LDPC codes, a family of very good error correcting codes, so some very rudimentary concepts in the mathematics behind error correction need to be explained. Further reading on the topic can be found in [1].

2.3.1 Group

A set is a collection of objects. A group, *G*, is a set of objects on which a defined binary operation (denoted as "·") is defined. That is, the operation takes two objects from the set and returns a third object in the set. The operation must follow the following rules:

1. It must be associative: $(a \cdot b) \cdot c = a \cdot (b \cdot c) \qquad a, b, c \in G$
2. An identity object, *e*, must exist: $e \cdot a = a \cdot e = a \qquad e, a \in G$
3. An inverse must exist: $a \cdot a^{-1} = a^{-1} \cdot a = e \qquad a, a^{-1} \in G$

If a group also satisfies the following rule 4, it is called a Commutative group:

4. $a \cdot b = b \cdot a \qquad a,b \in G$

The identity element for a commutative group is called the additive identity element.

Example 2. The set of integers form a (infinite) group under addition, but not multiplication since multiplication inverses does not exists in the group.

2.3.2 Ring

A ring is a set of objects, *R*, following the rules:

1. Two binary operations are defined, ”+” and ”·”.
2. *R* is a commutative group under +. The additive identity element is labeled “0”.
3. The operation · is associative.
4. The operation · distributes over +:
   $(a+b) \cdot c = (a \cdot c) + (b \cdot c) \qquad a,b,c \in G$

A Commutative Ring also follows:

5. The operation · commutes: $a \cdot b = b \cdot a \qquad a,b \in G$

A ring with identity follows:

6. The operation · has an identity element, labelled ”1”.

Example 3. A ring is a set of integers under modulo m.

2.3.3 Field

A field is a set of objects, *F*, if [1]:

1. Two operations, + and ·, are defined
2. *F* is a commutative group under +. The additive identity element is labeled *“0”*.
3. *F-{0}*, the field without the additive identity, is a commutative group under ·, with the multiplicative identity element labeled ”*1*”.
4. The operation · distributes over +:
   $(a+b) \cdot c = (a \cdot c) + (b \cdot c) \qquad a,b,c \in F$

Example 4. Infinite fields are the set of all rational numbers and the set of all real numbers.

2.3.4 Galois Fields (Finite Fields)

Finite Fields are usually called Galois Field, which are very important in the error correction research. A Galois Field containing *q* elements is called a Galois Field

of order q and is usually denoted GF(q) [1]. A Galois field must be of order $q=p^k$, where p is a prime integer and k an integer.

2.3.4.1 GF(2)

GF(2) consists of the set {0,1} and the operations, + and · , are described in Table 2.1 and Table 2.2.

+	**0**	**1**
0	0	1
1	1	0

Table 2.1 Additive operation under GF(2).

·	**0**	**1**
0	0	0
1	0	1

Table 2.2 Multiplicative operation under GF(2).

The addition and multiplication operation tables for GF(p) (p is a prime) can be constructed from the set {0, 1, .., p - 1} by performing an addition modulo p and multiplication modulo p.

2.3.4.2 Addition in GF(2)

Many error correcting codes contain addition of multiple elements. Addition in GF(2) has a special property; the additive inverse, a^{-1}, of an element, a, is the element itself, that is, $a=a^{-1}$.

A "proof" can be obtained by inspecting Table 2.1 and finding the cells where the result is *0*, then identify that both elements in the operation must be the same.

Error correcting codes are often determining the value of an element by assuming that all elements should sum to *0* (the parity check). With N elements, the equation would look like:

$$\sum_{k=1}^{N} a_k = 0 .$$

The additive inverse is the element itself, so this equation can be written as

$$\sum_{k=1}^{N} a_k + a_1^{-1} = a_1^{-1} .$$

Since $a+a^{-1}=0$ and $a=a^{-1}$ this is equal to

$$\sum_{k=2}^{N} a_k = a_1$$

2.3.5 Addition in Vector Spaces

Addition on higher order Galois fields (GF(p^k) with $k=2,3,\ldots$), can be seen as a vector addition in k dimensions, each dimension corresponding to a GF(p) addi-

tion. Fields of order p^k is usually called extensions of fields of order p. This thesis will repeatedly use fields of order 2^k, which are k binary symbols collected to one higher order symbol [1].

Example 5. Adding the two symbols from GF(8) corresponding to 5 and 1 results in the symbol 4. The operations in each dimension is an additive operation in GF(2).

$$5 + 1 = 4$$

$$\begin{pmatrix}1\\0\\1\end{pmatrix}+\begin{pmatrix}1\\0\\0\end{pmatrix}=\begin{pmatrix}0\\0\\1\end{pmatrix}$$

2.3.6 Block Codes

A block encoder encodes k consecutive symbols from a data stream into n symbols that is sent through the channel to the block decoder which (hopefully) decodes the received symbols back to the original data [1]. Generally, when working with channel data in GF(q) the k length block to be encoded is in GF(q^k), so the operations possible are described in Section 2.3.3 and Section 2.3.4. Assuming that the encoded data and the symbols sent on the channel are in GF(q), then the code rate of a block code can easily be calculated to

$$R = \frac{\log_q q^k}{n} = \frac{k}{n}.$$

Each message block (the block from the data stream) of length k is encoded to a code word of length n. The set of all possible code words is denoted C, and there are p^k code words in C, but there are p^n possible code words.

A linear block code with a k-sized message block and n-sized code word is usually denoted as a *(n, k)* block code. An encoding/generator matrix for a block code is usually denoted G, and its decoding/parity matrix, H.

Encoding is done by the matrix operation $y = x^T G$, where x is a vector containing k bits and y will be a codeword containing n bits. The channel will add some kind of noise. Assuming additive noise, the received signal will be

$$z = y + \Theta.$$

Decoding is made by finding the syndrome vector,

$$s = zH^T = (y + \Theta)H^T = yH^T + \Theta H^T.$$

The encoder and decoder matrices are created so that a codeword multiplied by the decoder will return 0.

$$yH^T = 0.$$

The syndrome vector will only depend on the errors introduced by the noise. Also, every syndrome vector corresponds to a unique error pattern, so a simple look-up table can be used to find the errors from the channel. A comprehensive explanation can be found in [1].

NOTE: This thesis will use the word frame instead of block, especially when dealing with error rates (Frame Error Rate, *FER*) so that no confusion will be made with Bit Error Rate, *BER*.

2.3.7 Hamming Distance

The definition of a Hamming Distance between two blocks, *v* and *w*, is the number of coordinates in GF(*p*) that are differing. That is, the number of differing elements in a vector space where each dimension is in the base field GF(*p*) [1].

$$d(v,w) = \left|\left\{i \middle| v_i \neq w_i,\ i = 0,0,\ldots,n-1\right\}\right|.$$

The minimum distance, d_{min}, of a block code is the smallest distance between any two possible code words. This measure is very important to a block code since it affects two very important properties of block codes. For a linear code, the all-zero word is a codeword, and therefore the minimum distance is the weight of the codes minimum weight codeword.

1. A decoder can detect all errors, if the number of errors are less than, or equal to $(d_{\min} - 1)$.
2. A decoder can correct all errors, if the number of errors are less than, or equal to $\frac{(d_{\min} - 1)}{2}$.

2.4 Low Density Parity Check Codes

A low density parity check (LDPC) code is usually a very large block code with a sparse decoding matrix (very few non-zero elements), [13] [14]. It is often represented by a Tanner Graph, named after Michael Tanner, a pioneer in iterative decoding. The Tanner Graph has three types of components; the variable nodes, the check nodes, and the edges. The Tanner Graph is implying a possibility to utilize an iterative decoder when decoding LDPC codes.

Each variable is operating wherever its corresponding column in the *H*-matrix has a 1, and each row is corresponding to a parity check function. In a Tanner Graph, each symbol, or variable is represented by a variable node, and each parity check function is represented by a check node. Wherever a row and column is sharing a 1, the variable will be connected to a parity check function, and this is represented by an edge between the corresponding variable node and check node. An illustration of a parity check matrix and its corresponding Tanner Graph can be seen in Figure 2.6. The variable nodes are here depicted by circles and the check nodes by squares.

A very important factor to the performance of a LDPC code is its node degree distribution. The distribution is defined as the distribution of edges for a node type, so each LDPC code has a check node distribution (denoted d_c) and a variable node distribution (d_v). Normally, a set of LDPC codes are distinguished by its node distribution. If all check nodes have the same number of connected edges and all variable nodes also have it (though, not necessary the same), then the code

is called a regular LDPC code. The node distribution for a regular LDPC code is written as (d_v, d_c). A very common regular code ensemble are the *(3, 6)*-LDPC codes.

In the notation of the irregular variable degree, the different variable node degrees are presented, together with how many of the total variable nodes that have that degree. The same notation is used for the check node degree for irregular ensembles. This is illustrated in Example 6, were the node perspective of the degree distribution is used. That is; how many nodes with a specific number of edges are there in the graph. One other way of looking at it is with an edge perspective. Then the number of edges with a specific degree node is taken into account. This is illustrated in Example 7 for the same code as in Example 6. When the weighting of the messages in the DE analysis is performed later on the edge perspective is used.

Example 6. If an irregular ensemble with 3000 variable- and 1000 check nodes is given. There are 600, 900 and 1500 degree 2, 4 and 16 variable nodes, 300 and 700 degree 3 and 10 check nodes. The node perspective notation for this irregular ensemble looks like:

$$\lambda(x) = 0.2\lambda_2 + 0.5\lambda_4 + 0.3\lambda_{16}$$
$$\rho(x) = 0.3\rho_3 + 0.7\rho_{10}$$

Where ρ_3 is the degree three check node and λ_{16} is the degree 16 variable node.

Example 7. The same code as in Example 6 is used. There are a total of (600·2) + (900·4) + (1500·16) = 28800 edges connected to a variable node. (600·2) edges are connected to a degree two variable node, (900·4) to a degree four and (1500·16) edges are connected to a degree 16 variable node. This gives that 0.042, 0.125 and 0.83 ,as a fraction of 1, of the edges is connected to a degree two, four and sixteen variable node respectively. The same types of calculations are performed for the edges connected to different check node degrees.

The code rate is usually not directly calculated from the LDPC code, but a design rate can easily be calculated from the node distribution [8]. For a regular LDPC code, the design rate is

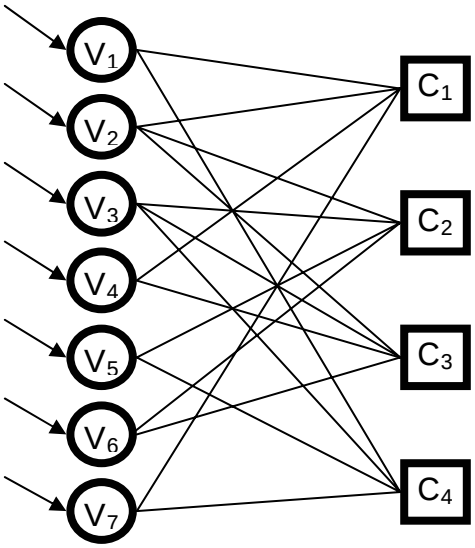
$$r = 1 - \frac{d_v}{d_c}.$$

The actual rate of the code may be higher, since the rows in H may be dependent on each other.

Since a LDPC code is usually large, it will have a good chance of having a very large minimum distance, which makes the codes very good in correcting and detecting errors. In fact, a good LDPC code almost never decodes a codeword that has not been sent; it either corrects the received symbols to the correct codeword or just detects an error in transmission.

Another advantage with a large, well designed, LDPC code is that its corresponding Tanner Graph will have a large girth. The girth is defined as the minimum number of edges that needs to be traversed to return to the starting point

without using the same edge twice. A large girth will make a Message Passing Decoder (will be described in Section 3.1) perform better.



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

Figure 2.6 Example of a Tanner Graph and its corresponding decoding matrix. Note that this matrix is not a LDPC code since it is not sparse.

3 Methods and Algorithms

This Section will present the pre-existing algorithms and evaluation methods that are being used in this thesis to create and evaluate LDPC decoders. First, a common algorithm family for decoding LDPC coded symbols, Message Passing Decoders, is presented along with its most common algorithm, Belief Propagation. Next, Density Evolution, an abstract analysis method for measuring performance on code ensembles, is presented together with an one-dimensional approximate version, the Extrinsic Information Transfer (EXIT) chart algorithm.

3.1 Message Passing Decoding

Message Passing is a very effective decoding algorithm for decoding LDPC codes. It is an iterative and highly parallelizable algorithm, where the idea is to pass messages along the edges between the variable and check nodes in a Tanner Graph. Each node type is processing the incoming messages and a set of outbound messages are created according to a predefined function. The function is naturally different in the variable nodes and the check nodes. An important condition is that the outbound message on an edge may not depend on the incoming message on the same edge. This condition is known to be a trait of good message passing decoders. When the decoder is initialized, the variable node initializes all of its outbound edge messages with the initial message from the channel.

There are three functions that need to be designed:

1. Variable node update function
2. Check node update function
3. Stop rule(s)

The following steps are performed during every Message Passing iteration:

1. Update each check node and its outbound messages with the predefined check node function.
2. Update each variable node and its outbound messages with the predefined variable node function.
3. Check if the stop criterion has been fulfilled. If so, stop the decoder.

Example 8. Assume binary input from the channel, {0, 1}. A very crude message passing algorithm that only passes a binary message between the nodes will be used.

Although the idea is simple, it can be complex when using other alphabets than GF(2). Additional information can be found in [13] and [14]. The following chapters will deal with the design of the node functions and the stop rules.

In order to keep track of the direction of a message, they will be named depending on the direction. Messages going from a variable node to a check node will be denoted u. Messages going from a check node to a variable node will be denoted v. In an implemented version, the direction of the message can most of the time be omitted, and the same (memory) space can be used for both directions. Sometimes it is unclear what the direction the message has, for example in an

analytic calculation. These messages will be denoted *w*. When the calculation has decided on a direction for the message, it will change *w* to the appropriate direction, *u* or *v*. Also, when messages are passed internally inside a node, *w* will be used (Figure 3.1).

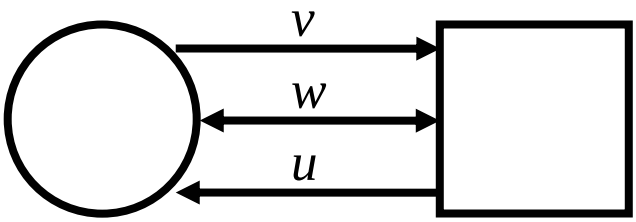
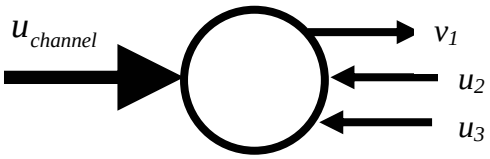


Figure 3.1 Message naming definitions.

3.1.1 Variable Node Update

The update function of a variable node is normally some kind of mean value, or a value that correspond to a symbol that all incoming messages can agree upon. Figure 3.2 is showing an example of a variable node update, where the outbound message is the mean of all incoming messages.

Example 9. Assume the same decoder type as in Example 8. An updated outbound message will be the mean value of the other incoming messages, rounded to either 0 or 1.



$$v_1 = round\left\{\frac{u_{channel} + u_2 + u_3}{3}\right\}$$

Figure 3.2 Example of a variable node and a message update function.

3.1.2 Check Node Update

The check node performs the update according to some kind of error correction scheme.

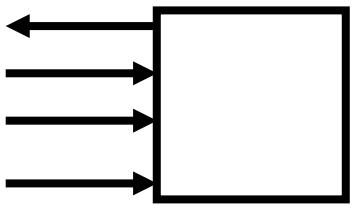
Example 10. Assume the same decoder as in Example 8. The check node needs to perform some kind of correction, but how? When the check node receives only correct messages, the parity check summation will be 0. The summations are made in GF(2).

$$\sum_i w_i = 0\,,$$

But if one or more of the incoming messages are incorrect, there will be no way of knowing which one are wrong. The (probably) best assumption that can be done is:
for each outbound message, u_i, assume that all other incoming mes-

sages, v_j, are correct and assign u_i so that the parity check summation will be 0. Since we are in GF(2), the following function can be used (Figure 3.3)

$u_i = \sum_j v_j$, j all incoming edges except *i*.



$$w_0 + w_1 + w_2 + w_3 = 0$$
$$\Leftrightarrow$$
$$u_0 = v_1 + v_2 + v_3$$

Figure 3.3 Example of a check node and its parity check function.

3.1.3 Stop Rule

There are mainly two types of stop rules that can be used, most of the time both are used in the same system. The first rule is allowing only a certain number of iterations in the decoder, the other rule is to determine whether or not a codeword has been detected and, in that case, make a pre-emptive stop.

Example 11. We can set a maximum number of iterations to 80. As stated in Example 10, a check node with only correct input messages will not change any of its messages. As a preemptive stop function, we can sum all incoming messages in the check node and see if the result is 0. If this is the case for all check nodes, then no corrections will be made and we can assume that we have found a codeword and stop the decoder. The codeword can then be found by calculating all its symbols from every variable node in the Tanner Graph. The symbol out of every variable node is calculated as a mean value out of all the incoming messages from all the edges to that node. But at this time all incoming messages to a variable node should be the same.

3.1.4 Faster Decoding by Serializing Node Operations

Before continuing, remember that it was stated in Section 2.3.1 that an algebraic operation in a group takes two arguments and returns one. When working with message passing, updating each output message requires input from multiple edges, all edges but itself. It is possible to create an algorithm that processes, for each edge, the result of all other edges. However, this algorithm is very slow, since it needs to loop through the messages several times. Assuming there are n edges connected to a node, there are *n* incoming and *n* outbound messages. For each outbound messages, about *n-1* operations needs to be done, so the algorithm would scale $O(n^2)$.

Instead of having one node with *m* edges connected to it, we create *n* nodes with three edges connected to each one, with intermediate edges between these nodes. Each outbound message will now only depend on two incoming messages, but we will have additional messages between the nodes. An example with 5-

edged check node and its serialized version is illustrated in Figure 3.4. The messages, $\boldsymbol{m_i}$, consist of the inbound and outbound message, v_i and u_i, on that edge while the internal messages will be denoted w.

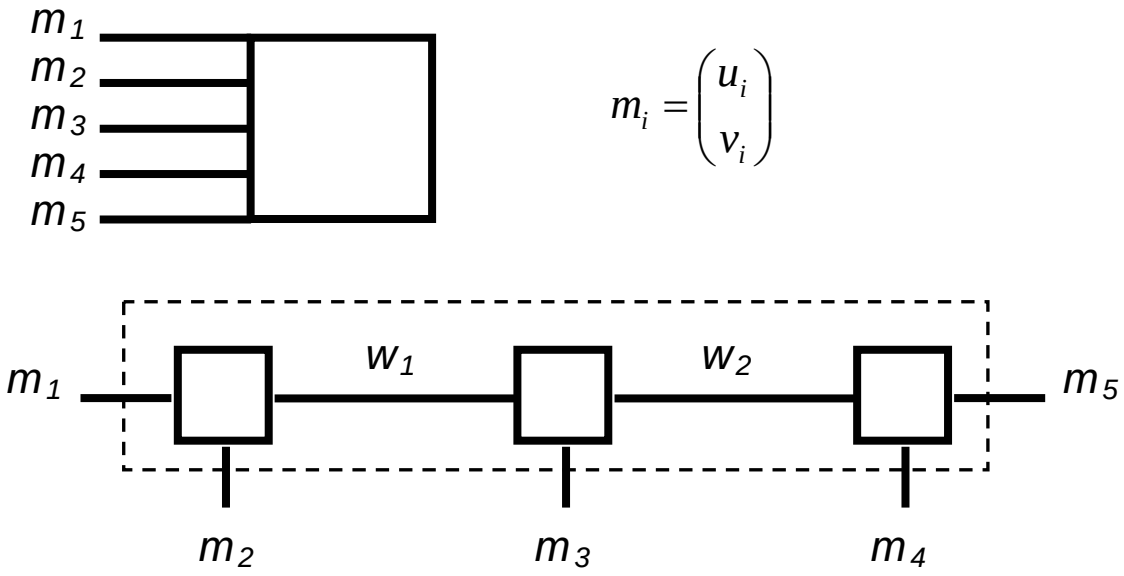


Figure 3.4 Serialization of a degree-5 node.

The algorithm will go from the left side to the right, updating the intermediate messages. When reaching the last node, it will update the outbound messages, turn around and update the outbound messages and the intermediate messages until it reaches the last node.

The algorithm:

1. Start at the node at one side of the chain (here we assume the left side).
2. Update the intermediate message w_1 by operating on v_1 and v_2.
3. Go to the next node.
4. Update the next intermediate message w_2 by operating on the already (incoming) intermediate message w_1 and the inbound message, v_3, attached to the node.

Repeat 3 and 4 for all nodes except the last one.

5. Update u_n by operating on the incoming intermediate message, w_{n-3}, and v_{n-1}.
6. Update u_{n-1} by operating on the incoming intermediate message, w_{n-3}, and v_n.
7. Update the intermediate message, w_{n-3}, by operating on v_n and v_{n-1}.
8. Go back to the node to the left of the current node.
9. Update the outbound message attached to the node, u_{n-2}, by operating on the two intermediate messages attached to the node, w_{n-3} and w_{n-2}.
10. Update the intermediate message, w_{n-2}, attached on the left hand side by operating on the intermediate message on the right hand side, w_{n-3}, and v_{n-2}.

Repeat steps 8-10 for all remaining nodes, except the leftmost node.

11. Update u_2 by operating on v_1 and the incoming intermediate message w_1.
12. Update u_1 by operating on v_2 and the incoming intermediate message, w_1.

This algorithm gives two main advantages; the decoding complexity scales only to O(n) since each incoming message is only used once, and performing the analysis of the check node is possible to by only analysing two incoming messages and one outbound message of a degree-three check node.

3.2 Belief Propagation Decoding

A very powerful Message Passing decoder is the Belief Propagation Decoder. The messages passed along edges are probability-based. There are mainly two types of Belief Propagation decoders, one passes real probabilities for each symbol, and the other one passes logarithmic probabilities. Davey and MacKay have, in [15], described a method of using Belief Propagation in higher order Galois Fields using real probabilities. This thesis will use higher order Galois Fields extensively, so real probabilities will be passed in the Belief Propagation Decoder. Logarithmic Likelihood Ratio will only be used in the DE analysis description part of this thesis.

3.2.1 Check Node Update

The check node update is based on parity checks in GF(q), where the outbound message, u, is the additive operation of v_1 and v_2

$u=v_1+v_2$.

The outbound message in the check node is the probability of the actual outbound message being one of the M possible symbols. In the binary case, the outbound Likelihood Ratio messages are two-dimensional

$$P_u = \begin{pmatrix} P(u=0) \\ P(u=1) \end{pmatrix}.$$

It is possible to keep track of just one of the probabilities as a message since $P(u=0)=1-P(u=1)$, but we will keep both for clarity.

To decide the outgoing message, we need to look at the binary additive operation table (Table 3.1), and decide the probabilities for getting a 0 and 1 respectively

$$P_u = \begin{pmatrix} P(u=0) \\ P(u=1) \end{pmatrix} = \begin{pmatrix} P(v_1=0)P(v_2=0)+P(v_1=1)P(v_2=1) \\ P(v_1=0)P(v_2=1)+P(v_1=1)P(v_2=0) \end{pmatrix}.$$

		v_1	
	+	**0**	**1**
v_2	**0**	0	1
	1	1	0

Table 3.1 Addition in GF(2).

In higher order fields the probability calculations will become more complex, but the same principle applies. The calculations for $q=2^2$ are presented in Table 3.2.

v_2 \ v_1	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Table 3.2 Addition in GF(4).

$$\begin{aligned}
P(u=0) &= P(v_1=0)P(v_2=0)+P(v_1=1)P(v_2=1)+\\
&\quad P(v_1=2)P(v_2=2)+P(v_1=3)P(v_2=3)\\
P(u=1) &= P(v_1=0)P(v_2=1)+P(v_1=1)P(v_2=0)+\\
&\quad P(v_1=2)P(v_2=3)+P(v_1=3)P(v_2=2)\\
P(u=2) &= P(v_1=0)P(v_2=2)+P(v_1=1)P(v_2=3)+\\
&\quad P(v_1=2)P(v_2=0)+P(v_1=3)P(v_2=1)\\
P(u=3) &= P(v_1=0)P(v_2=3)+P(v_1=1)P(v_2=2)+\\
&\quad P(v_1=2)P(v_2=1)+P(v_1=3)P(v_2=0)
\end{aligned}$$

Formally, this can be expressed as:

$$P(u=a)=\sum_{i=0}^{q-1}P(v_1=i)P(v_2=i+a) \qquad i,a\in GF(q).$$

Observe that the addition $v_2=i+a$ is made in GF(q), so the resulting v_2 can be found in the addition table (Table 3.2 for GF(4)).

3.2.2 Variable Node Update

The variable node message update works in a slightly different way than the check node update functions. First of all, the result needs to be normalized to have a proper probability distribution as a result. Second, the channel input needs to be considered at some point.

The basic, binary two input- one output, algorithm can be viewed as

$$P_v=\begin{pmatrix}P(v=0)\\P(v=1)\end{pmatrix}=\frac{1}{\alpha}\begin{pmatrix}P(u_1=0)P(u_2=0)\\P(u_1=1)P(u_2=1)\end{pmatrix}.$$

To ensure that the outbound message is a proper pdf, the message is normalized by α.

$$\alpha=P(v=0)+P(v=1).$$

In a higher order field, each outgoing symbol probability can be expressed as

$$P(v=a)=\frac{1}{\alpha}P(u_1=a)P(u_2=a)$$

where

$$\alpha=\sum_{i=0}^{q-1}P(v=i).$$

The channel input can be treated as an input edge, but without updating it.

A single, outbound, message for a symbol from a variable node on edge k of d_v edges can be fully expressed as

$$P(v_k=a)=\frac{1}{\alpha}P(u_{channel}=a)\prod_r P(u_r=a) \qquad r\in\{1\ldots d_v\neg k\}$$

$$\alpha=\sum_{i=0}^{q-1}P(v=i).$$

3.2.3 Stop Rule Design

Except of having a maximum number of allowed iterations, an additional stop rule may be used. The second stop rule is based on deciding whether or not the most probable code word is a correct code word by finding the most probable symbols from the variable nodes, and see if they fulfil the parity check. If so, stop the decoder and declare success. This can be expressed as stopping the decoder when

$$H\hat{x}=0,$$

where H is the sparse decoding matrix used and $\hat{x}$ is a vector with the most probable symbol from every variable node.

$$\hat{x}=\begin{pmatrix}x_1 & x_2 & \ldots & x_n\end{pmatrix}^T,$$

$$x_i=\underset{a}{\operatorname{argmax}}\{P_k(x=a)\} \qquad a\in\{0\ldots(q-1)\}.$$

$P_k(x=a)$ is the probability that variable node k contains the symbol a and is derived in a similar way as the message updates, but this time messages from all edges are considered.

$$P(x=a)=P(u_{channel}=a)\prod_{i=0}^{q-1}P(u_i=a).$$

In a decoder, this can be solved by determine the most probable symbol at the same time as updating a variable node, then send the most probable symbol along with the actual message to the check nodes, where a GF(q)-parity check is performed before updating the check node.

If the parity check is summed to 0 for all check nodes, then the decoder has found a valid code word and may stop.

3.3 Non-binary LDPC Codes and M-PSK Modulation

When using non-binary LDPC codes together with M-PSK modulation for data transmission, the information data is processed accordingly (Figure 3.5 and Figure 3.6).

1. The n length binary information Data is mapped onto n/log_2M length M-ary symbol information data.
2. The symbol information data is encoded into codewords with the GF(M) generator matrix for the LDPC code used.
3. The symbols of the codewords are M-PSK modulated to I/Q symbol vector representation.
4. The codewords is sent trough the channel and their symbol vectors are distorted by the channel noise.
5. The received distorted vector symbols of the codewords is M-PSK Demodulated into symbol representation.
6. The received codewords is decoded with a Message Passing algorithm for higher order modulation formats (Section 3.2).
7. The M-ary data symbols of the decoded codewords are converted to their binary representation, the binary data is sent to the user.

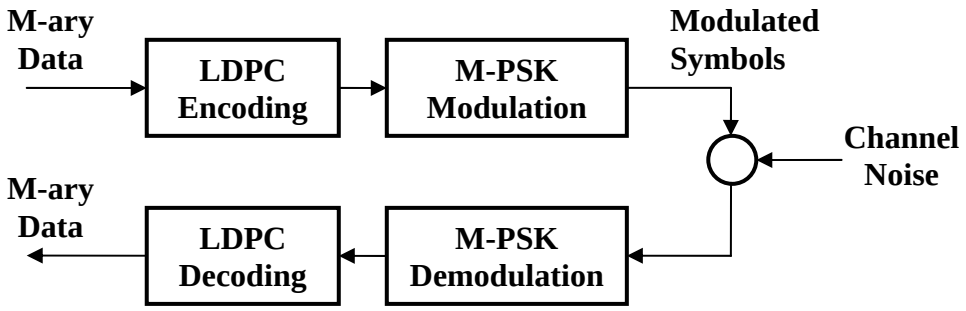


Figure 3.5 Transmission of *M*-ary data when using LDPC codes together with M-PSK modulation.

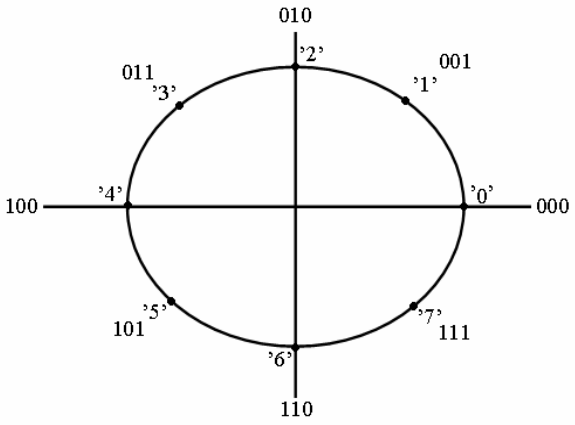


Figure 3.6 Example of 8-PSK modulation of binary data and its corresponding *M*-ary symbol.

3.4 Density Evolution

Density Evolution (DE) is an algorithm used to analyse the performance of an ensemble of LDPC codes with a certain degree distribution, regular or irregular, when using Message Passing decoding. The algorithm only depends on degree distribution of the nodes and the SNR of the channel, it is not performed on a specific LDPC code belonging to that ensemble. So DE does not for example depend on the Tanner Graph representation of a Code. The ensemble of codes contains all codes with the same degree distribution. The result of a Density Evolution calculation is the lowest SNR possible, the $SNR_{Threshold}$, for which successful Message Passing decoding is possible for an ensemble. Density Evolution returns a channel performance measurement constrained to a specific code ensemble.

The results from Density Evolution can be used to compare different degree distributions to find the best code ensemble. Even though different ensembles have the same rate *R*, they will not perform equally. The degree distribution of the regular *(3,6)* ensemble is unique. But the realizations (connections between nodes) are different between different codes in the ensemble. In the case of the regular *(3,*

6) and *(5, 10)* ensembles, they have the same rate, *R=0.5*, but the regular *(3, 6)* ensemble have a smaller $SNR_{Threshold}$ than the regular *(5, 10)* ensemble. In fact the regular *(3, 6)* ensemble have the best $SNR_{Threshold}$ of all the *R=0.5* ensembles. This means that there are codes with a regular *(3, 6)* degree distribution that perform better than any other regular *R=0.5* LDPC code.

3.4.1 Main Idea

Density Evolution is an iterative algorithm performed on an ensemble of LDPC codes with a certain distribution, regular or irregular, for a fixed SNR value. In the DE algorithm the assumption is made that the code length is infinite. The reason for assuming this is that the Tanner Graph for the code can then be assumed to be cycle free. Infinite code lengths give infinite number of nodes. By assuming this, it is possible to assume that the code will have an infinite girth, i.e. it is cycle free. This means that a message has to travel an infinite length before returning to the same node, which basically means that the message is independent from messages previously sent from the node.

By assuming a cycle free Tanner Graph for the code, the calculations of a single DE iteration can be seen as updating the outgoing pdfs from one check node and one variable node representative for regular ensembles. In the case of irregular ensembles one representative for each degree for the check- and variable nodes has to be updated. That is because for regular (d_v, d_c) ensembles of LDPC codes there are only two different node types, the d_v -degree variable node, and the d_c-degree check node. For irregular ensembles there are as many different node types as there are different node degrees for the variable- and check nodes (Section 2.4 and [12]). For irregular codes, the output densities from all the degree representatives for the variable- or check node are added, with the degree distribution (edge perspective) as weight factor. The idea of only updating one node representative for each unique node type in the ensemble is the key approximation of DE, making it a tractable algorithm for analysing the performance of code ensembles.

The main idea with density evolution is to update the probability density function, pdf, of the messages between the nodes representatives, instead of the messages themselves. The outbound densities from the node representatives will then evolve during the iterations, and if the pdfs evolve in such a way that the error goes to zero then the current SNR is not below the $SNR_{Threshold}$.

The update of the outgoing pdf messages f_u^k , f_v^{k+1} and f^0 is illustrated for the regular (3, 6) LDPC code ensemble in Figure 3.7, where k is the iteration index. Here, f^0, f_u^k and f_v^{k+1} are the outbound pdf from the channel and the variable- and check node representative, describing the Log Likelihood Ratio (LLR) of the transferred messages. When LLR probabilities are used the mean of the pdf can have values between $-\infty$ and $+\infty$.

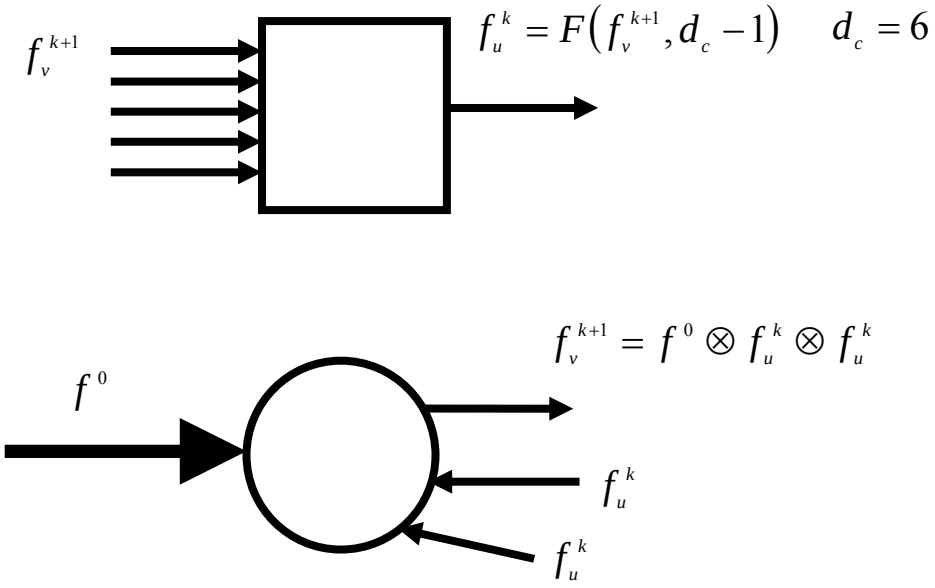


Figure 3.7 Pdf updates on the regular (3, 6) ensemble. Check node on top and variable node below, d_c=6 and d_v=3.

3.4.2 Performing Density Evolution

The initial pdf, f^0 created from the messages received from the channel, depends on the type of channel noise, order of modulation *M*, and the SNR. The channel is assumed to be an AWGN channel, so the DE algorithm only depends on the SNR from the channel and the analyzed node distribution. The initial pdf from the channel is computed for the SNR under consideration. The DE algorithm is then iterated a great number of times with this as the initial input to see if the error probability P_e^k from the pdf of iteration k, f_u^k, from the check node, converges to zero when the number of iterations increases towards infinity ($k \to \infty$). P_e^k is calculated by integrating f_u^k for all LLR values ≤ 0. This is the same as looking at the SNR-values when the mean LLR m_u^k of f_u^k approaches infinity. The initial error probability P_e^0 from the channel is the Log Likelihood Ratio between the wrongly detected messages and the number of transmitted symbols, and the P_e^k is that Log Likelihood Ratio after k Message Passing iterations.

The basic algorithm for Density Evolution works according to the following steps:

Preparations

1. Choose the ensemble to be analyzed, i.e. the node distribution.

2. Set the range of the AWGN $SNR_i = \{SNR_1 \quad SNR_2 \quad ... \quad SNR_{\max}\}$ values to be tested with DE. The SNR_i is chosen as the lowest SNR value, and first to be tested. Then the SNR values should increase to the largest in the range, SNR_{max}.

3. Set the number of iterations n and an acceptable error threshold limit P_e^{max} (biggest acceptable error). This should be 0 but is for practical reason set to a value very close to zero.

Density evolution

4. Calculate the initial f^0 from the channel depending on the first SNR_1 value in the interval and the type of noise.

5. A) If regular ensemble: Iterate the algorithm *n* times. Update and f_u^k and f_v^{k+1} every $k \in \{1 \quad 2 \quad ... \quad n\}$ iteration.

   B) If Irregular ensemble: Iterate the algorithm n times. Update each pdf out from each node representative for all the unique node degrees of the variable- and check nodes. Calculate f_u^k and f_v^{k+1} from the weighted calculations for the pdfs [12]

6. Calculate the error P_e^n out of the final check node output f_u^n. Stop the DE if $P_e^n \leq P_e^{max}$, the $SNR_{Threshold}$ has then been found. Otherwise increment *i* and go back to step 4 (test next SNR)

If $SNR_{Threshold}$ is not found during these steps a different range of SNR_i values has to be defined and tested for the ensemble.

The node degree distribution and pdf update functions for an ensemble of regular *(3,6)* codes is illustrated in Figure 3.7. The notation for node degree distribution for an irregular ensemble of codes looks a little different. For irregular codes there is more than one node degree distribution for the variable- and check nodes.

One example of Density Evolution on a regular *(3, 6)* LDPC ensemble is illustrated in Figure 3.8. The SNR value is here set to 1.73 dB, which appears to be the $SNR_{Threshold}$ for that ensemble. That is because $P_e^k \rightarrow 0$ and $m_u^k \rightarrow \infty$ as $k \rightarrow \infty$. This $SNR_{Threshold}$ calculation concurs with the results calculated by Barry in [13]. The Likelihood ratio (*LR*) for the messages has here been transformed to the Log Likelihood ratio (*LLR*) before calculating the pdf, which gives LLR values on the horizontal axis in the figure.

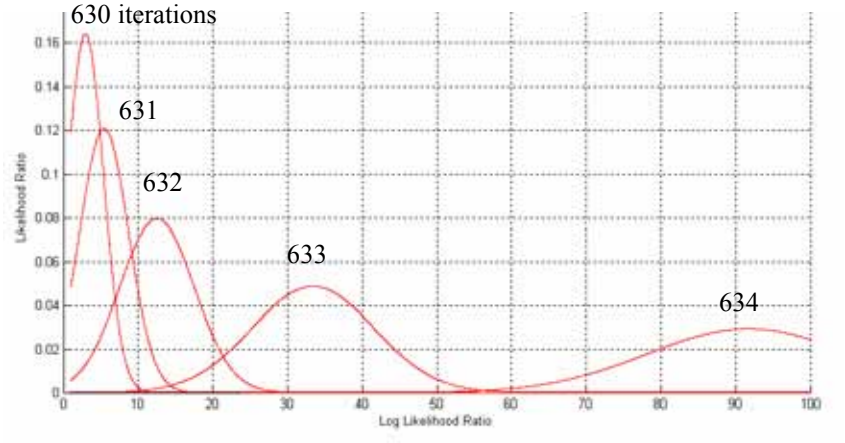


Figure 3.8 The outbound pdf of the messages as a function of SNR for the (3, 6) ensemble. The number above each curve is the number of iterations for that curve.

3.4.3 One-Dimensional Approximation of Density Evolution

Since the update functions for f_u^k and f_v^{k+1} in Section 3.4.1 and 3.4.2 are performed on probability density functions, they are very complex and time consuming when implemented. However, less complex algorithms that approximate DE has been developed. These are one-dimensional analysis of LDPC codes, instead of the multi-dimensional analysis in ordinary DE. Likelihood Ratios (real probabilities) will be used instead of *LLR* for denoting the error probability in the EXIT chart algorithm below.

3.4.3.1 The EXIT Chart Algorithm

A one-dimensional approximation of DE is the Extrinsic Information Transfer chart algorithm, or the EXIT chart algorithm [10]. A one-dimensional variable describing the extrinsic information transfer between the nodes are here sent between the check- and variable node representatives instead of the pdf for the messages. The information used in the EXIT chart calculations is the initial information about the received messages from the channel (intrinsic information) and the information about the messages from the previous iterations (extrinsic information). Other one-dimensional analysis methods for LDPC Codes makes the assumption that all pdfs sent between the representatives are Gaussian, the all Gaussian approach [13][11]. This is not the case in the reality, because the update functions in the check node make its outbound pdf messages non-Gaussian.

For the EXIT Chart algorithm, Gaussian distribution is only assumed for the pdf of the messages sent from the variable node to the check node representative. The pdf for the messages sent from the check node to the variable node representative is not assumed to be Gaussian distributed. So using the semi-Gaussian approach in the EXIT Chart calculations makes the results more accurate and closer to those of the non-approximate DE. The main idea behind the EXIT chart algorithm is to represent the pdf messages f_u^k and f_v^{k+1} sent between the nodes, and f_u^0 from the channel, with a corresponding one-dimensional variable for the intrinsic- and extrinsic information. In other words describe the pdfs with scalars instead of a multidimensional vector (discrete pdf) which otherwise represents the pdf in the

computer calculations. The scalars can be the corresponding error probability for f, its mean or mutual information [10]. This approach gives calculations with scalars instead of multidimensional calculations with discrete probability density functions represented by vectors. From now on, EXIT chart calculations are assumed to be performed with the error P_e corresponding to the pdf f.

The channel used is the AWGN, but the calculations are the same as for the BSC channel. This can be done since the EXIT chart calculations depend on the initial error, P_e^0, for the received messages from the channel, not the shape of their pdf. The incoming (initial) pdf, f^0, from the channel is the probability distribution of the incoming messages from the channel. If all the messages sent through the channel is the '0' M-PSK symbol, then because the distortion in the channel all incoming messages will have different probabilities of being detected as the '0' symbol with the Maximum Likelihood Detection. The greater the distortion of the channel the more symbols will be more likely to be detected as any of the other *M-1* possible symbols rather then the correct '0' with Maximum Likelihood (ML) detection. This ratio between the number of wrongly detected symbols divided with the number of correctly detected symbols is the calculated initial error P_e^0 from the channel for the sent symbols. This is the same as constructing a discrete pdf, f^0, out of a great number of sent and received '0' symbols from the AWGN channel, and then calculate the initial error P_e^0 as the area of the pdf where wrongly ML detection takes place.

The initial input,P_e^0, to the EXIT chart algorithm is calculated for the initial pdf, f^0, from the channel. For each iteration $k \in \{1 \quad 2 \quad \ldots \quad n\}$, a new outbound message P_e^{k+1} from the variable node representative is calculated from P_e^k and P_e^0. The update function for a regular ensemble is Equation 3.1. The error probability information P_e^{k+1} from the variable representative is calculated for each k iteration. As in Density Evolution for an ensemble, a range of SNR_i values are tested with n iterations. The reason and background for using the initial error probability P_e^0 as the extrinsic information for the initial pdf is described in [10].

$$P_e^{k+1} = P_e^0 - P_e^{(0)}\left[\frac{1+\left(1-2P_e^k\right)^{d_c-1}}{2}\right]^{d_v-1} + \left(1-P_e^0\right)\left[\frac{1-\left(1-2P_e^k\right)^{d_c-1}}{2}\right]^{d_v-1} \tag{3.1}$$

where $P_e^{in} = P_e^0$ and $P_e^{Out} = P_e^n$

Here d_c and d_v is the check- and variable node degree for the ensemble being analyzed. A regular *(3,6)* code would for example have $d_c = 6$ and $d_v = 3$.

3.4.3.2 Performing EXIT Chart Analysis

The EXIT chart calculations for a regular ensemble are performed in the following steps:

Preparations

1. Choose the code ensemble to be analyzed, i.e. the node distribution.

2. Define the $SNR_i = \{SNR_1 \quad SNR_2 \quad ... \quad SNR_{\max}\}$ to be tested in the EXIT chart analysis.
3. Define the maximum number of iterations, n, and an acceptable error limit P_e^{max}.

EXIT chart Calculations

4. Calculate the initial P_e^0 corresponding to the pdf f^0 from the channel. f^0 depends on the type of noise and the first SNR_1 value in the SNR_i vector from step 2.
5. Iterate algorithm Equation 3.1 n times with $k \in \{1 \quad 2 \quad ... \quad n\}$.
6. If $P_e^n \leq P_e^{max}$, then $SNR_{Threshold}$ has been found, otherwise increment i and go back to step 4 (test next SNR).

The result of the EXIT chart calculations is close to the result of the ordinary Density Evolution, but the complexity of the calculations is much lower.

The result from EXIT chart calculations on an ensemble a given SNR can be visually illustrated by printing the EXIT chart. One example of a printed EXIT chart is given in Figure 3.9. In the EXIT chart, the incoming error from the algorithm, $P_{in} = P_e^k$, is on the horizontal axis. The outgoing error, $P_{out} = P_e^{k+1}$, is on the vertical axis. The initial P_e^0 from the channel is used as a starting point in the diagram. P_e^{k+1} can be expressed as a function of the incoming error as $P_e^{k+1} = F(P_e^k)$. To make the EXIT chart easier to analyse, the inverse, F^{-1}, of F is also plotted in the chart. F^{-1} is a function of the outgoing error, and returns the incoming error. The space between F and F^{-1} is called the decoding tunnel. The development of P_e^{k+1} depends on the previous P_e^k, and can be viewed in the chart for each iteration k. If $P_e^{k+1} < P_e^k$ for all k, then the error P_e^{k+1} will approach zero as k approaches infinity, then F^{-1} and F will never cross each other. The decoding tunnel is then open for all k [10]. The $SNR_{Threshold}$ for the ensemble is found as the SNR when the decoding tunnel opens up from a closed state, and is open for all iterations. In other words; the $SNR_{Threshold}$ is the SNR value giving an almost closed decoding tunnel, so that just a little smaller value on the SNR will close it. An open decoding tunnel will guarantee that the final decoding error $P_e^n \rightarrow 0$ when $n \rightarrow \infty$. In practical implementations the error becomes very small for only a couple of hundred iterations when the $SNR_{Threshold}$ is found.

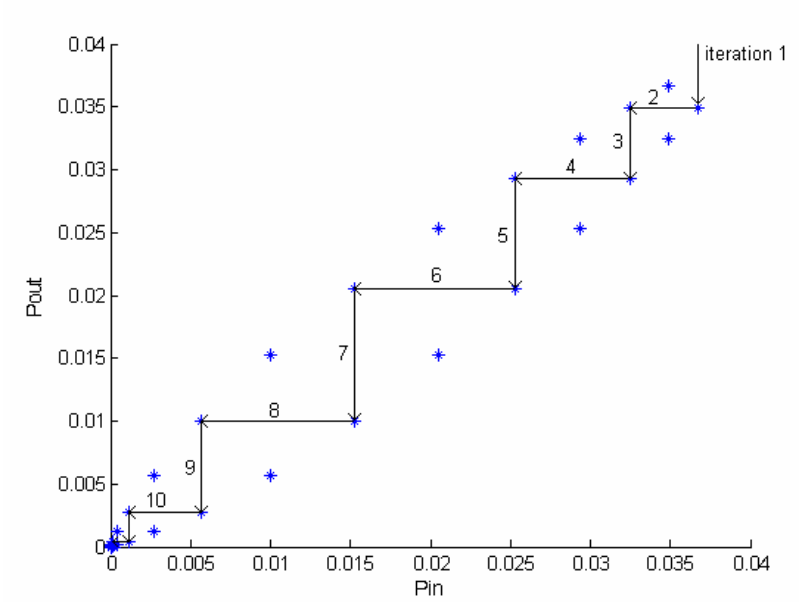


Figure 3.9 EXIT chart for a regular (3, 6) code ensemble. The decoding tunnel is opened so the set SNR value is above $SNR_{Threshold}$.

3.4.4 EXIT Chart with M-PSK Signaling

This section will describe a practical algorithm for performing the EXIT chart calculations on an ensemble of regular non binary LDPC codes using M-PSK symbols. There are several algorithms which perform one-dimensional DE on ensembles of codes, but EXIT calculations have proven to be one of the best methods [10]. The results of performing EXIT chart calculations with the implementation in this Section will be presented in Section 6.

The first part of this section will describe how to calculate the initial pdf, f^0, for the algorithm, and the second part will describe the basic EXIT chart calculations with the corresponding initial Likelihood Ratio, P_s^0. It is the Likelihood Ratio of having detected symbol s as the initial symbol when s has been sent. Detection is made using an ML detector. The possible symbols are $s \in \{0 \quad 1 \quad \ldots \quad M-1\}$.

3.4.4.1 Calculating the Initial pdf for the Zero M-PSK symbols

In M-PSK modulation one can view and process the modulated data in two different ways, either as the $M=2^k$ different vector symbols, or as the M different length k binary representations of the vector symbols. The two different representations for 8-PSK symbols are illustrated in Figure 3.10.

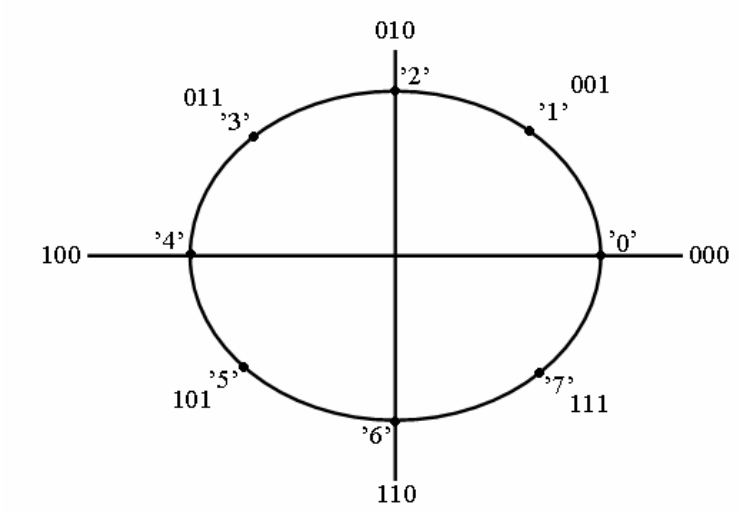


Figure 3.10 The symbols and their binary representation for 8-PSK.

EXIT chart is often calculated using binary representation of the symbols, assuming a Binary Symmetric Channel [9], where a total independence between the incoming bits then can be assumed when calculating the initial pdf from the channel. However, when using non-binary symbols the incoming bits will not be independent from the other *k-1* bits representing the symbol. When, for example, 8-PSK symbols with binary representation are used, every bit is dependent on two other bits. Here some dependency calculations between the incoming bits have to be made, which depends on the behaviour of the binary representations for the *M* possible symbols when distorted by the noise in the channel. When updating the pdfs of the *k* bits in the binary representation of the symbols, assumptions about their dependency can be made by using different approaches. For instance, it is possible to first update the pdfs independently and then create a mean pdf between the bits and add it to the bits pdf.

A slightly different approach for performing EXIT chart on the symbol representation of the M-PSK symbols will here be presented. The focus will be on the symbol representation of the M possible symbols, where the possible symbols are $i \in \{'0' \quad '1' \quad \ldots \quad 'M-1'\}$. First the *M* different initial pdfs, $f^{0}{}_{i}$, from the channel for the M symbols are calculated. But when transmitting over the AWGN channel, all the symbols in the M-PSK signalling constellation used will have the same pdf (Figure 3.10). This gives that only one pdf has to be calculated for one symbol, which is then valid as the pdf for all *M* possible symbol pdfs. This will allow the simplification of only calculating and analysing the code for one of the *M* possible M-PSK symbols that can be sent. Here, the symbol used for the EXIT chart analysis is the zero symbol, '0'. The basic algorithm calculating the initial pdf for the zero symbol, f_0^0 is performed accordingly:

1) Set the SNR value to be tested in the symbol-wise Density evolution.
2) Transmit a large number of Zero M-PSK symbols over an AWGN channel with the SNR from step 1. Store the received distorted zero symbols in a vector $V_{messages}$.

3) Calculate the probability vector P_i^0 from $V_{messages}$ (Appendix A.1), where every element in P_i^0 is the probability that the corresponding element in $V_{messages}$ is the '0' symbol.
4) Create a vector V_{temp} with every column containing the number of each unique zero symbol probabilities in $P_{'0'}$. Sort the columns so the unique zero probabilities lie in ascending order.
5) Divide all the elements in V_{temp} with the total number of zero symbols sent in step 2. The resulting vector can be used as the discrete pdf, $\tilde{f}_{'0'}^{0}$ for the received messages from the AWGN channel. With each element in $\tilde{f}_{'0'}^{0}$ giving the Likelihood Ratio that the received messages have one of the unique zero probabilities of being the zero symbol.

Example 12. After step 3; if the first column in V_{temp} is 13 and the smallest element in $P_{'0'}$ is 0.1, then there are 13 received messages with 0.1 probability of being the '0' symbol. After step 4; if 1000 symbols were sent in step 2 the first element in $\tilde{f}_{'0'}^{0}$ should be 13/1000 = 0.013. Here 1.3 % of the received symbols have 0.1 probability of being the zero symbol.

In the EXIT Chart calculations in Section 3.4.4.2 $\tilde{f}_{'0'}^{0}$ will be used as the approximated discrete pdf for the received distorted zero M-PSK symbols from the channel. An illustrative example of $\tilde{f}_{'0'}^{0}$ for 1000 AWGN distorted zero 8-PSK symbols with channel *SNR=9.5dB* is presented in Figure 3.11.

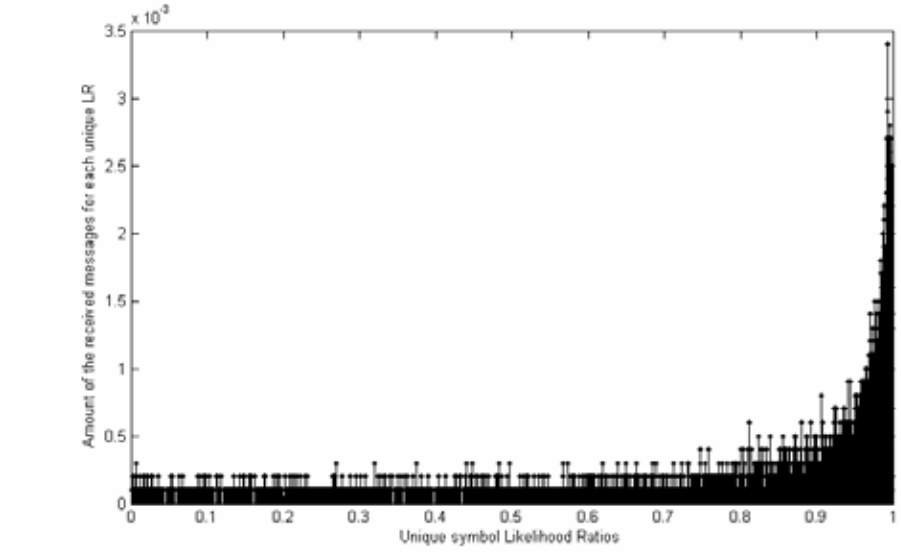


Figure 3.11 The pdf for 1000 8-PSK symbols with SNR =9.5dB. The notation on the axis is Likelihood Ratio.

3.4.4.2 EXIT Chart Calculations for a LDPC Code with M-PSK Symbols

After the discrete probability density function $\widetilde{f}_{'0'}^{\,0}$ for the M-PSK symbols is calculated in Section 3.4.4.1, its corresponding error probability $P_e^{\,0}$ can be calculated with Equation 3.2.

$$P_e^0 = \sum_{0}^{0.5} \widetilde{f}_{'0'}^{\,0} \tag{3.2}$$

In other words; sum all elements of $\widetilde{f}_{'0'}^{\,0}$ with probabilities (LR) value on the horizontal axis equal or below 0.5 (Figure 3.11).

The EXIT chart calculations for a LDPC code with M-PSK symbols are performed according to the following steps:

1) The Exit chart calculations from Section 3.4.3 are executed with the $P_e^{\,0}$ calculated out of Equation 3.2**,** with $P_e^{\,0}$ being the one-dimensional representative for $f_{'0'}^{\,0}$.
2) If the check in step 6 in the EXIT Chart calculation (Section 3.4.3) is not fulfilled, a new SNR_i range is selected and new $\widetilde{f}_{'0'}^{\,0}$ and $P_e^{\,0}$ are calculated.
3) These $P_e^{\,0}$ are again used in step 4-6 in the EXIT Chart algorithm from Section 3.4.3.

These steps are performed until the $SNR_{Threshold}$ for the analysed ensemble is found.

4 Angular Sum Decoding

In this section we present a Message Passing decoder for the vector representation of the *M* possible M-PSK symbols, using the angle and length as information of the M-PSK symbols. When sending M-PSK signals, the symbol information will be represented by the angle, while the length of the received vector can be seen as some kind of reliability information. The parity check will be made by requiring that the sum of all angles in a check node will sum to *0 mod 2π*. This summation requirement is the cause of the name of the decoder, Angular Sum Decoder.

The aim is to keep the messages and decoding simple for the decoder, and to see whether or not this type of decoder is possible to create. Since this approach is a simplification of a Belief Propagation decoder, it will probably need a higher SNR in order to keep the same *BER* as the Belief Propagation decoder. An interesting version of the Angular Sum Decoder is to only pass the angular information as messages in the Message Passing algorithm and assume the length of the vector to be 1. This version will need to send less information and thereby become less complex.

If a working Angular Sum Decoder can be constructed, the messages sent between the nodes in the Message Passing algorithm will contain significantly less information than in the Belief Propagation algorithm. Instead of sending a probability vector of length *M*, only scalars representing the angles (and lengths) are sent as messages between the nodes. Also, the calculations in the nodes will be much less complex than in a Belief Propagation decoder. If a successful decoder can be constructed, the version without length information would probably require a higher SNR to work since it is strapped on the reliability information.

As described in Section 3.1.4, message operations can be described by two input and one output message. Outbound messages from variable nodes and check nodes respectively, will be calculated as:

- Variable node: $v = \arg\left\{e^{iu_1} + e^{iu_2}\right\}$
- Check node: $u = -(v_1 + v_2) \bmod 2\pi$

where *u* and *v* are the angles of the messages. If length information is considered, update operations are performed as:

- Variable node:
  - Angle Update: $v = \arg\{U_1 e^{iu_1} + U_2 e^{u_2}\}$
  - Length Update: $V = \left|U_1 e^{u_1} + U_2 e^{u_2}\right|$
- Check node:
  - Angle Update: $u = -(v_1 + v_2) \bmod 2\pi$
  - Length Update: $U = \min\{V_1, V_2\}$

Where *u* and *v* are the angles of the messages and *U* and *V* are the lengths of the messages. Figure 4.1 and Figure 4.2 illustrate the variable node and check

node operations. $\bar{u}$ and $\bar{v}$ denote the vector messages that have angle *u* and *v* and length *U* and *V* respectively.

In essence, the operation in the variable node is just a vector summation (Figure 4.1), while the operations in the check node (Figure 4.2) are a bit more complex. The angular summation made in the check node is very simple, but the question lies in how to adjust the length. The approach made in this decoder is quite simple: Find the least reliable information (the shortest length) and set the length of output to it.

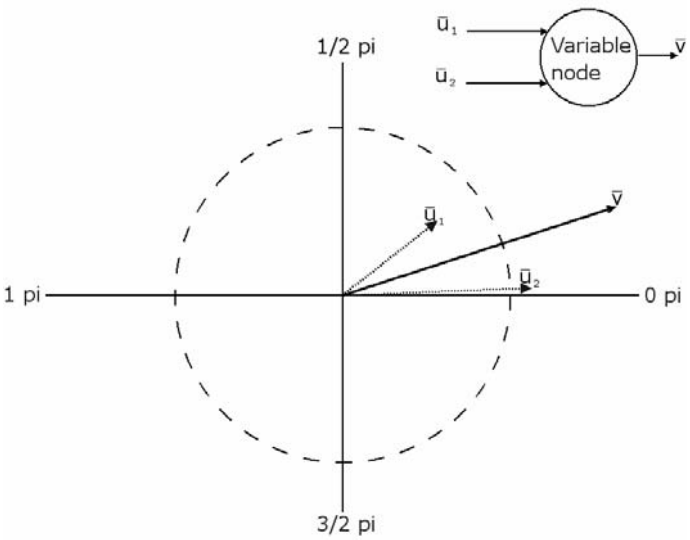


Figure 4.1 Length and angle calculations of the outgoing message vectors from the variable nodes. Here v is calculated out of three incoming messages (one from the channel and two from check nodes).

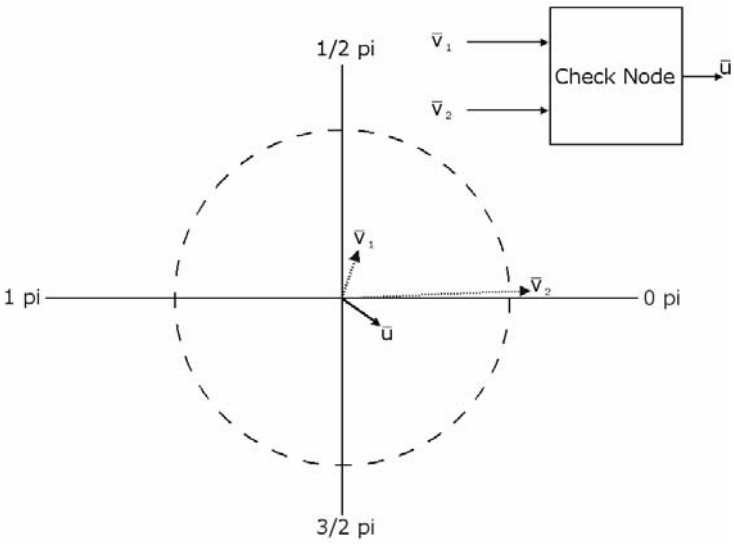


Figure 4.2 Length and angle calculations of the outbound message vectors from the check nodes.

4.1 Vector Summation in a Variable Node

As mentioned in Section 3.1.1, a variable node update should be something all edges should agree upon. When dealing with Belief Propagation, they determine the probabilities of the symbols. When two edges agree that a specific symbol has a high probability, the outbound probability for that symbol will be higher, while two edges agreeing on a low probability of a symbol, it will have a lower outbound probability. This behaviour should also be found in the approximate decoders and vector summation will do that.

As a matter of fact, given that two probability vectors are representing two vectors from the channel, a summation of these two vectors returns a vector that has a probability vector that is exactly the one that would be returned from a Belief Propagation variable node operation on the first two probability vectors. The proof of this can be found in Appendix A.3. That is, having two vectors taken from an AWGN channel using M-PSK signaling,

$$\overline{m}_A = \begin{pmatrix} I_A \\ Q_A \end{pmatrix} \quad \overline{m}_B = \begin{pmatrix} I_B \\ Q_B \end{pmatrix},$$

and adding them,

$$\overline{m}_C = \overline{m}_A + \overline{m}_B,$$

will be equivalent to

$$P\left(m_i \middle| \overline{m}_C\right) = \frac{P\left(m_i \middle| \overline{m}_A\right) P\left(m_i \middle| \overline{m}_B\right)}{\sum_{j=0}^{q-1} P\left(m_j \middle| \overline{m}_A\right) P\left(m_j \middle| \overline{m}_B\right)}.$$

Having this proof, it is safe to determine that a vector summation in the variable node is equivalent to a Belief Propagation message update. This operation will be used as a variable node operation in the Angular Sum Decoder as well as the Table Decoder described in Section 5.

5 Table Decoder

Simulations performed on the Angular Sum Decoder from Section 4 indicated that this type of decoder would not work (simulation results in Section 7.2.2). Due to this, an alternative solution for MP decoding with angle and length messages needs to be found. As discussed in Section 8.1 the node operations that need to be changed are the operations in the check node. However, since the aim is still to send geometrical vectors, it is possible to reuse the variable node functions described in Section 4.

The solution lies in finding an alternative check node operation which gives an error correcting Message Passing decoder when using angle and length messages for the M-PSK symbols. This problem can be approached with a Black Box Model for the operations in the check node. As described in Section 3.1.4, it is possible to serialize a check node with any number of edges into a concatenated system of degree three check nodes, so the Black Box algorithm only needs to be designed for check node with two inputs and one output.

5.1 Variable Node Operation

As discussed in Section 4.1, making a vector summation is equivalent to a Belief Propagation variable node operation, so the Table Decoder will also use vector summation in the variable nodes.

5.2 Black Box Model for Check Node Operations

The main idea with the black box approach is to use the knowledge about how the probability vector messages sent to and from a degree three check node in the Belief Propagation decoder are processed. This can be done by trying to create check node calculations that imitates check node calculations for Belief Propagation, but instead of messages based on probabilities, geometrical vectors are being used as messages. One way of creating the check node functions for geometrical vectors is to start with a Belief Propagation check node, but immediately inside the node, convert the messages from probabilities to geometrical vectors. On the outbound side, the opposite conversion is made. Now, all that needs to be made is to design some kind of function that makes the entire check node a good approximation of the original Belief Propagation check node, and then remove the conversions and pass the geometrical vectors as messages between the nodes (Figure 5.1).

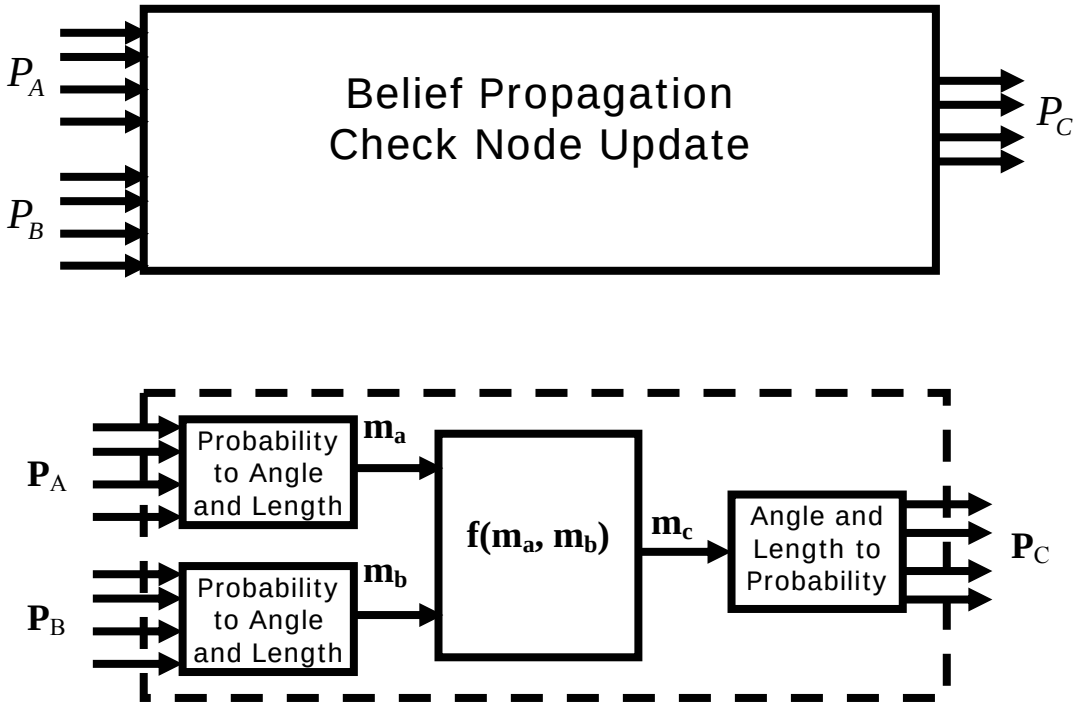


Figure 5.1 Approximation of a Belief Propagation check node update.

The internal messages, m_a, m_b, and m_c, consist of either angle and length (vector) information, or just the angle information. The angle/vector to probability conversion is described in Appendix A.1.

There is one problem left to deal with before designing the function $f(m_a,m_b)$ in Figure 5.1. It is the conversion from probabilities to geometrical vectors. There is no good analytical way to convert from arbitrary probabilities, but an approximate method can be used, which only allows a limited number of probabilities in a quantified probability space. The allowed probabilities can be found by converting a number of geometrical vectors to probability vectors. The number of vectors converted gives the number of reconstruction points used to convert from probability vectors to geometrical vectors, which also gives how fine the quantization will be.

Our proposed decoder will be designed as a four-dimensional table. The dimensions represent the angles and lengths of the two incoming signals. A simpler decoder does not keep the length information, so it can be represented as a two-dimensional table. The elements in the tables are the reconstruction points. An illustrative example of a two-dimensional table is presented in Figure 8.9.

5.2.1 Table Vector Decoder

When implementing the Table Vector decoder with the algorithm presented in the beginning of Section 5.2, we need to decide which $M \in \{2,4,8\}$ it will operate on. We also need to decide a SNR that it will assume to receive from the channel.

Next, the vector space need to be quantified into a set of allowed points. The suggested strategy for doing this is to first decide a set of allowed angles, preferably $A = a \cdot M$ angles (where a is a positive integer) evenly spread. This will allow each ‘correct’ angle (angle where a signal point is located) to be represented in the

set. Deciding upon a set of L allowed lengths is also suggested, which are $l \cdot k$ where $l = \{1,2,\ldots L\}$, and k is a fixed length. If only one length is allowed (for example length 1), then the decoder is called the Table Angle decoder. The points in the vector set are the reconstruction points r_j. More strategies, and a concise discussion of the placement of reconstruction points, can be found in Section 8.5.

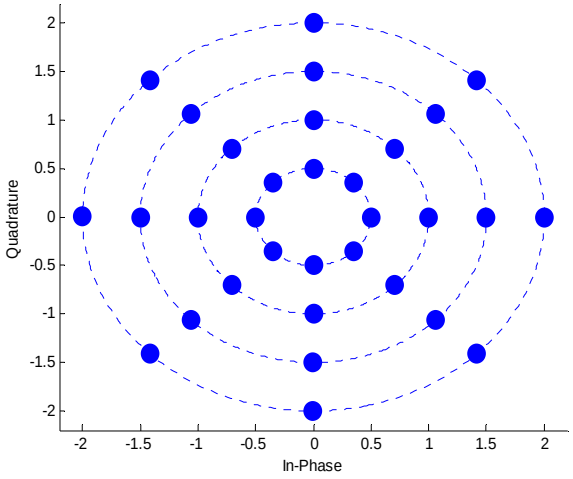


Figure 5.2 Reconstruction points for 4 lengths and 8 angles.

Now, a quantified vector space containing $N=AL$ reconstruction points has been developed. Next thing to do is to find the corresponding probability distributions P_j for these points (see Appendix A.1). Each vector space point r_j will now have one corresponding probability space point P_j (Figure 5.3). It is possible to have one reconstruction point in the origin, which would give an additional reconstruction point ($N=AL+1$).

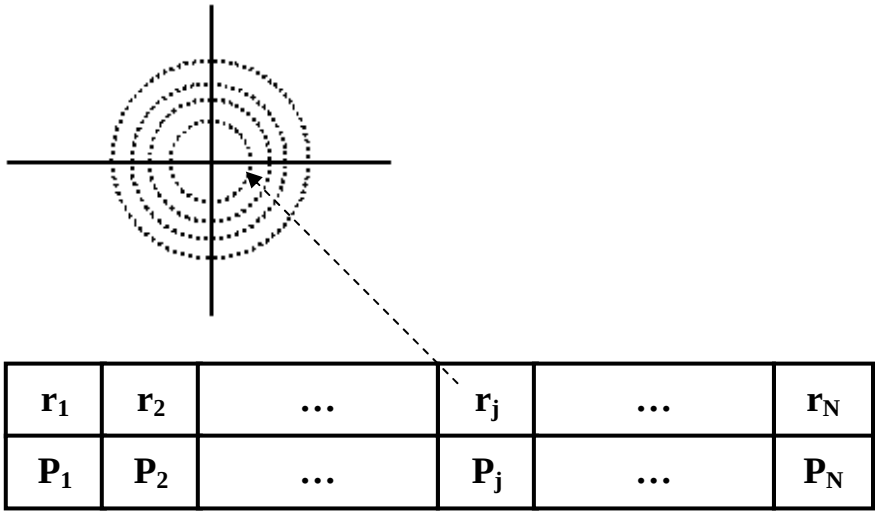


Figure 5.3 Each vector space point corresponds to one probability space point.

	P_1	P_2	…	P_i	…	P_N
P_1	P_{11}	P_{21}	…	P_{i1}	…	P_{N1}
P_2	P_{21}	P_{22}	…	P_{i2}	…	P_{N2}
⋮	…	…	…	…	…	…
P_j	P_{1j}	P_{2j}	…	P_{ij}	…	P_{Nj}
⋮	…	…	…	…	…	…
P_N	P_{1N}	P_{2N}	…	P_{iN}	…	P_{NN}

Figure 5.4 Probability Table, check node operation on probability space points. The return values need to be adjusted so they too belong to the probability space points.

The probability space is a known space (calculated in Appendix A.1). The parity check made for two incoming probability distributions is known, (Section 3.2.1). Now, a table of all possible outbound probabilities using all possible combinations of two inbound probabilities from the allowed set can be built (Figure 5.4). At this point, the probability table is not a proper algebraic operation table, since the outbound probabilities are not necessary from the set of allowed probabilities. This can be solved by, instead of returning the real outbound probability, return the probability from the allowed set that is closest to the real one. In other words quantize the real outbound probability distributions to the allowed probability distributions (reconstruction points r_j). There are several ways of deciding the closest point; the algorithm used in this thesis decides the closest reconstruction points, r_j, by calculating the closest minimum squared Euclidian distance:

$$P_{out} = \min_{P_k}\{\sum_{a=0}^{M-1}(P_{out}^a - P_k^a)^2\}$$

$$P_{out} = \begin{pmatrix} P_{out}^0 \\ \vdots \\ P_{out}^{M-1} \end{pmatrix},$$

$$P_k = \begin{pmatrix} P_k^0 \\ \vdots \\ P_k^{M-1} \end{pmatrix}.$$

The probability table is now only containing probabilities from the allowed set, P_j, which makes it possible to perform algebraic operation with its elements. Not only that, but since all probabilities are from the allowed set, each one of

them corresponds to a point, r_j, in the vector space which allows us to create a table that is a copy of the probability table, but contains the corresponding geometrical vectors instead. This table is all we need when making an approximation of the parity check summation described in Section 3.2.1. The Vector Table used in Vector Decoding is built according to the following steps.

1. Decide M and an appropriate SNR.
2. Decide allowed lengths and angles and create a table of allowed vectors r_i with them.
3. Calculate the probability distributions P_j from the allowed vectors r_i (Appendix A.1), having M possible symbols and the decided SNR.
4. Build a probability table (Figure 5.4). Quantify the outbound probability vectors to the closest allowed probability vectors P_j in the probability table.
5. Replace the probability vectors P_j in the probability table with the corresponding vector r_j , gives the corresponding Vector Table (Figure 5.5).

	$\mathbf{r_1}$	$\mathbf{r_2}$	…	$\mathbf{r_i}$	…	$\mathbf{r_N}$
$\mathbf{r_1}$	$\mathbf{r_{11}}$	$\mathbf{r_{21}}$	…	$\mathbf{r_{j1}}$	…	$\mathbf{r_{N1}}$
$\mathbf{r_2}$	$\mathbf{r_{21}}$	$\mathbf{r_{22}}$	…	$\mathbf{r_{j2}}$	…	$\mathbf{r_{N2}}$
:	…					
:		…				
:			…			
						…
$\mathbf{r_j}$	$\mathbf{r_{1j}}$	$\mathbf{r_{2j}}$	…	$\mathbf{r_{ij}}$	…	
:	…					
:		…				
:			…			
						…
$\mathbf{r_N}$	$\mathbf{r_{1N}}$	$\mathbf{r_{2N}}$	…	$\mathbf{r_{jN}}$	…	$\mathbf{r_{NN}}$

Figure 5.5 Vector Table of check node operation on vector space points.

The incoming messages to the parity check node need to be quantified into the allowed vector points r_j. This can also be done by finding the minimum squared Euclidian distance between the incoming messages v_a and v_b to the closest reconstruction points r_j (Figure 5.6).

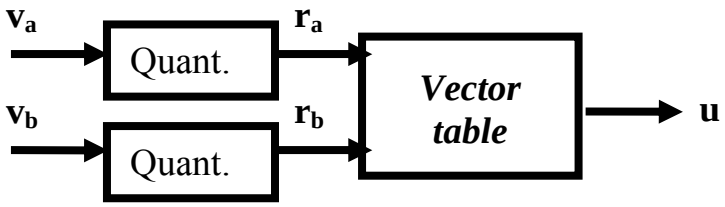


Figure 5.6 Check node operation on two arbitrary geometrical vectors as inbound messages.

One unfortunate effect is that the Vector Table will probably be less correct if using a different SNR. That is because the SNR level of the incoming messages from the channel will effect the conversion from vector representation into probability vector representation. Also, the amount of different possible signals sent will affect the conversion, so a different *M* requires a different Vector Table. However, it will be shown in Section 8.3.1 that once the Vector Table is calculated for a specified *M* and a sufficiently high SNR, it will be possible to change the channel SNR with a little or no degradation of the performance. From now on, a decoder with a table having only angular information will be called *Angle Table Decoder*, and one with length information as well will be called *Vector Table Decoder*. The family of these two decoders will be called *Table Decoder*.

5.2.2 Visualization of Output Values for the Angle Table

If only angles are used for creating a table, it is possible to visualize it with a 3-dimensional surface plot. The result is a very peculiar shaped surface with very sharp edges, see Figures 5.7 and 5.8. The figures show the inbound angles to the check node on the x- and y axis, and the outbound angle from the check node on the z-axis. These tables are made for *4*- and *8-PSK* over an AWGN channel with SNR at *6 dB* and *13 dB* respectively. The inbound angles to the check node are on the horizontal axes, ranging from 0 to 2π. The resulting outbound angle is the angle on the vertical axis. Some of the edges in the figure are 2π jumps, since the output angle is shown in the interval $-\pi$ to $+\pi$.

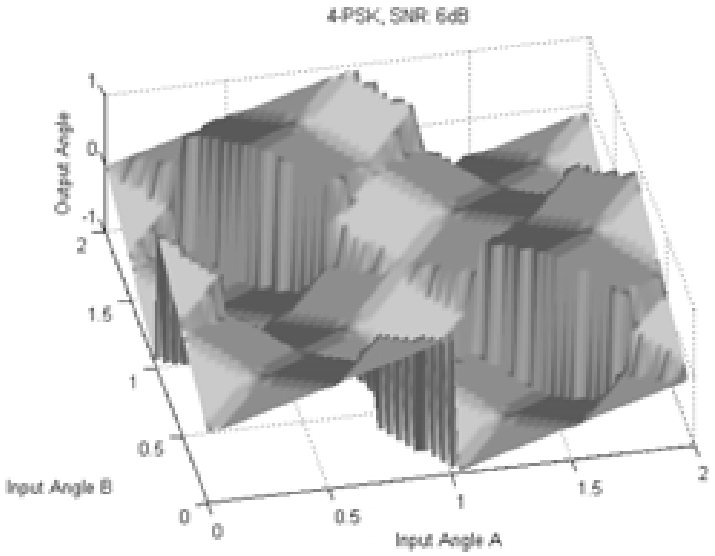


Figure 5.7 3-D surface plot of the angle tables used in the Table Angle Decoder for 4-PSK

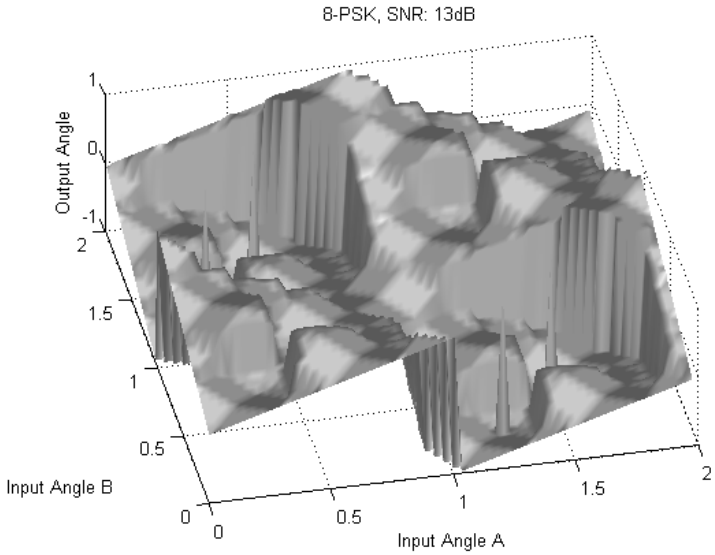


Figure 5.8 3-D surface plot of the angle tables used in the Table Angle Decoder for 8-PSK.

6 EXIT Chart Calculations for M-PSK

This section contains the results from EXIT chart calculation for an ensemble of regular *(3, 6)* LDPC codes when using M-PSK symbols. The theory and algorithms from Section 3.4.3 and 3.4.4 are used in the following EXIT chart calculations. The purpose of this section is to calculate the $SNR_{Threshold}$ of the ensemble of non-binary regular *(3, 6)* LDPC Codes when using M-PSK symbols.

6.1 Threshold Calculation with EXIT Chart

The results that will be presented here are calculated with the pdf for the zero symbols '0' (of M possible symbols) as in Section 3.4.4, not its binary representation 000. When using the EXIT chart algorithm from Section 3.4.3 to calculate the $SNR_{Threshold}$, the first SNR to be tested was *-4dB*, then the SNR value was increased by *0.1 dB* every time the algorithm was iterated until SNR_{max} was reached according to this algorithm. If the $SNR_{Threshold}$ is- or is not found the procedure in the algorithm is followed. The number of symbols transmitted over an AWGN channel in each analysis to compute the initial pdf from the channel is *100 000*. The number of iterations n in each EXIT chart calculation is *1000*. No more is needed since after only a couple of hundred iterations the error probability P_e^n has already converged. The $SNR_{Threshold}$ of an ensemble of regular *(3, 6)* LDPC codes with M-PSK modulation for $M=\{2, 4, 8\}$ is presented in Table 6.1. Only 7 SNR values near the $SNR_{Threshold}$ for each M are presented, even though many more values were tested.

Figure 6.1 shows an illustrative example of how the initial error, P_e^0 , is calculated out of all received zero symbols from the channel, using the initial pdf, f^0. The EXIT chart for the ensemble with the modulation- and AWGN channel parameters $M=2$ and $SNR = -2dB$ is presented in Figure 6.2. Notice that the decoding tunnel is open and that 12 iterations will make the P_e^k very small, so P_{out}^n will converge towards zero. P_e^0 is the Likelihood Ratio between the black area and the total area (gray and black). The black area is the amount of sent '0' symbols through the AWGN channel that will be detected as another of the *M-1* possible symbols with a Maximum Likelihood detector.

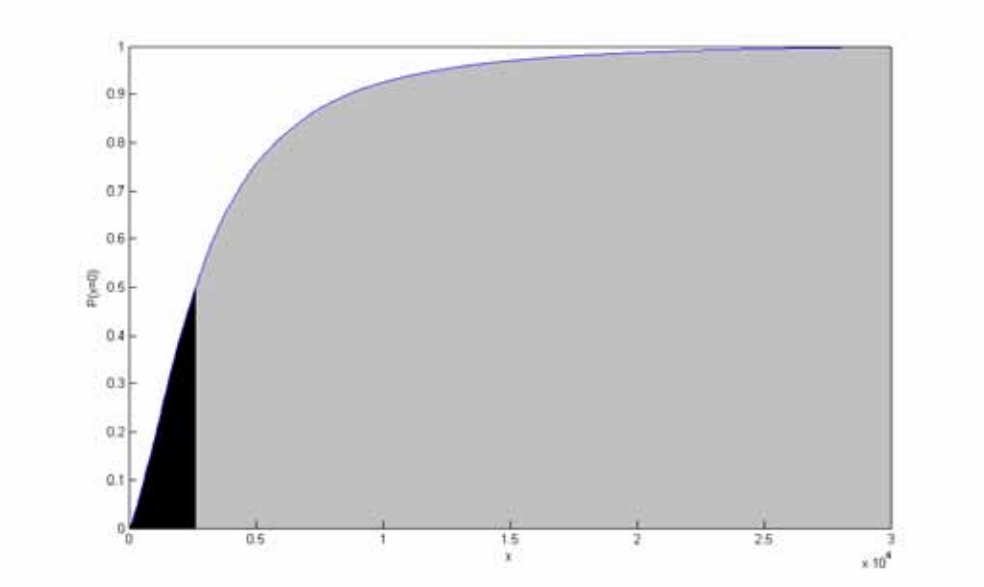


Figure 6.1 100 000 symbols with AWGN noise added (horizontal axis) and the probability of them being the zero symbol (vertical axis). The symbols are sorted in ascending order with respect to their probability to be the zero symbol. M=2, SNR=-2 dB, n=1000, P_e^0 =0.0367.

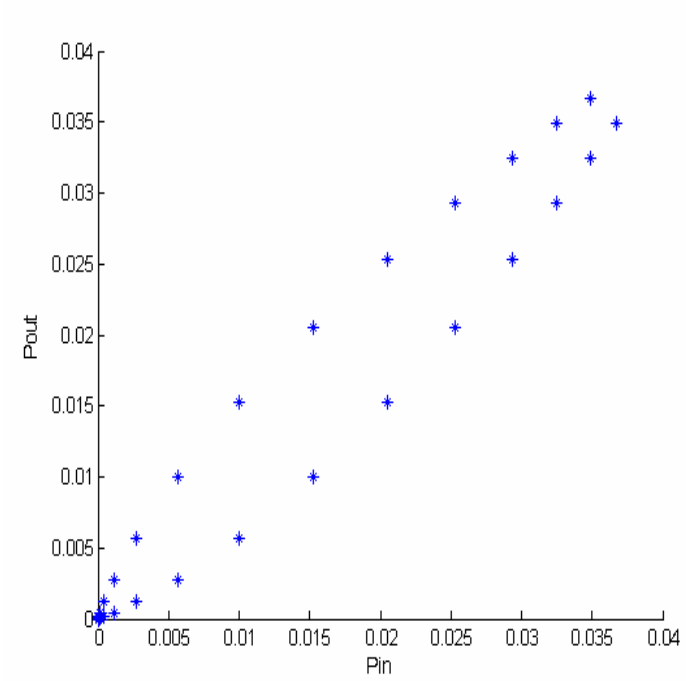


Figure 6.2 EXIT chart for the same parameters as in Figure 6.1.

M	SNR	P_0^n
2	-2.5	0.2577
2	-2.4	0.2570
2	-2.3	0.2562
2	-2.2	0
2	-2.1	0
2	-2.0	0
2	-1.9	0
2	-1.8	0
2	-1.7	0
2	-1.6	0
4	2.7	0.2612
4	2.8	0.2599
4	2.9	0.2591
4	3.0	0.2583
4	3.1	0.2575
4	3.2	0
4	3.3	0
4	3.4	0
4	3.5	0
4	3.6	0
8	7.8	0.2633
8	7.9	0.2625
8	8.0	0.2611
8	8.1	0.2601
8	8.2	0.2590
8	8.3	0.2583
8	8.4	0.2573
8	8.5	0.2564
8	8.6	0
8	8.7	0

Table 6.1 Threshold calculations whit EXIT chart algorithm for an ensemble of regular (3,6) LDPC codes with M-PSK symbols for different M. Number of EXIT chart iterations is n = 1000, and the amount of distorted symbols used to calculate the pdf f^0 for the symbols is 100 000. The $SNR_{Threshold}$ for the three different modulation levels are -2.2, 3.2 and 8.6 dB.

7 Simulations

In this section the results from simulations with the Angular Sum Decoder, Table Vector Decoder, Table Angle Decoder and the Belief Propagation Decoder are presented. The use of the term Table Decoder will refer to both the Table Angle- and Table Vector Decoder. The simulation results presented in this Section will be analyzed and discussed in Section 8.3**.** 80 Message Passing decoding iterations per decoding session were always used in the simulations. All simulations have been made using the same LDPC code [18], which is a regular, (*3, 6*) LDPC code of length 504.

The simulations were made using Monte Carlo simulations in Matlab where data was transmitted over an AWGN channel with various SNR values. The result consists of trace of *BER* and *FER* for the different setups. Up to 500000 codewords were transmitted through the AWGN channel and decoded with the tested decoder every decoder simulation. If less "bumpiness" for some of the curves in this section is needed, more codewords have to be sent during the decoder simulation.

When a Table Decoder uses a "floating table" it calculates and uses a new angle- or vector table for each channel SNR. These tables have to be precalculated and stored. This also results in additional bookkeeping during simulations/decoding. When a Table Decoder uses a "fixed table" only one table is calculated and used for each order of modulation, M, for all shifting channel SNR. This table is calculated with an SNR giving a very low *BER* when using the corresponding floating table for the order of modulation used. A fixed table gives much less data to store and less calculations to be made for the Table Decoder.

7.1 Simulation Algorithm

The same basic simulation algorithm was used for all the different decoders tested. It consists of a preparation step and an iterated simulation step.

Preparations for the decoder simulations

1. Choose the PSK modulation alphabet, M={2, 4, 8} (order of modulation).
2. Choose the LDPC code to be used in the simulation, which gives the generator matrix G and the parity check matrix H.
3. Set the SNR for the AWGN channel.
4. Set the number of codewords to be sent through the channel. A large amount of codewords sent and received in the simulation gives better protection against quick random changes in decoding performance.
5. Set a maximum number of MP iterations for the decoder.

Steps performed each iteration for the decoder simulation

1. Generate a codeword with a generator matrix G, using random input data. Input data is evenly distributed in $\{0, \ldots, M-1\}$.
2. Modulate the codeword symbols to the M-PSK constellation, with symbol '0' at *angle = 0*, going counter-clockwise (Section 3.3).

3. Transmit the modulated codeword through the AWGN channel (Section 2.2.2.3)
4. Receive the modulated codeword.
    a. If using Belief Propagation, transform the received symbols to probability vectors (Appendix A.1).
    b. If using the Angular Summation- or the Table Decoder, the vector representation of the messages received from the channel is used as messages (Section 4 and 5).
5. Try to decode the received codeword using the Message Passing algorithm with the used LDPC code and decoder. The general Message Passing decoding algorithm is described in more detail in Section 3.1. Stop the decoding if the maximum number of iterations is reached or if a codeword is found.
6. Compare the generated codeword from step 1 with the decoded in step 5. Determine if the codeword is properly decoded. If not, count the number of bits that differ and adjust *FER* and *BER*.

7.2 Simulation Results

In this section the simulation results will be presented for the different decoders together with some modifications on the construction of the tables used in the Table Decoders. The main focus in the simulations was on Table Decoders using 4-PSK signalling. This is because 2-PSK signalling with Table Decoders is not that interesting because of bad error correcting- and complexity performance compared to the Belief Propagation Decoder. Even though simulations for 8-PSK signalling with Table Decoders are very interesting, they are also very time-consuming (further discussion in Section 8.3).

7.2.1 Belief propagation Decoder

The simulation results for the Belief Propagation (Section 3.2) are presented in this Section. M-PSK signaling was used (Section 3.3), and different order of M-PSK modulation on the symbols are *2*, *4* and *8*. The results are presented in Figure 7.1.

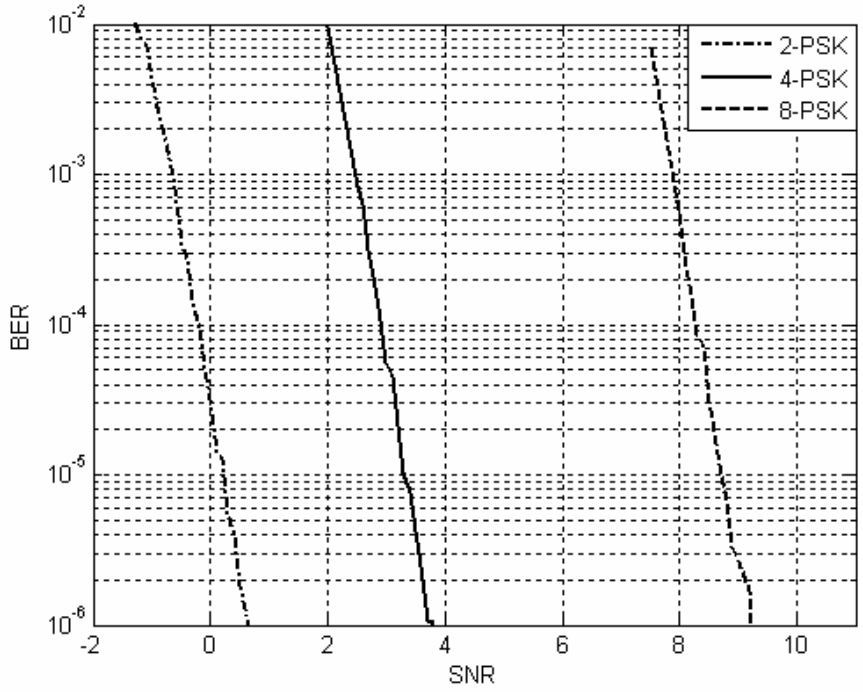


Figure 7.1 Simulations with Belief Propagation Decoder for the regular (3,6) LDPC code with codeword length 504 and M-PSK modulation. The maximum number of iterations is 80.

7.2.2 Angular Sum Decoder

The simulations with the Angular Sum Decoder (Section 4) was performed for {2, 4, 8}-PSK. The results were very poor. The Angular Sum Decoder failed to decode a correct codeword, even for very large SNR. Even when the SNR was set to a level so only one of all the received symbols in the codeword was incorrectly detected, the decoder still failed to decode the codeword. Figure 7.2 visualizes the performance of the decoder for iterations 0-9. 2-PSK signalling was used and only one symbol was initially on the left hand side of the Q-axis.

This led us to the conclusion that the approach of Angular Sum Decoding presented in Section 4 with M-PSK symbols do not work.

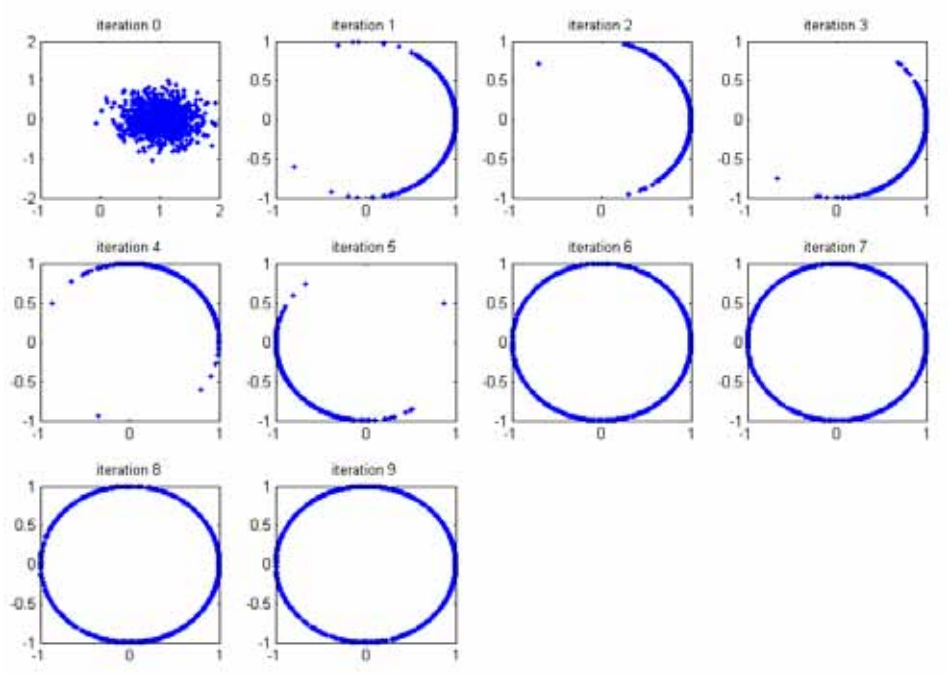


Figure 7.2 Decoding BPSK symbols with Angular Sum Decoder and the regular (3,6) LDPC code with codeword length 504, from iteration 0 to 9. Only one of all the symbols in the codeword is wrong in iteration 0, i.e. a very large SNR is used. The horizontal axes are the In-Phase signal and the vertical axes are the Quadrature signals.

7.2.3 Angular Sum Decoder using Length Information

The simulations in this section were carried out in the same way as with the Angular Sum Decoder, but with node operations that also updated the length (Section 4).

The result from decoding with added length information was as bad as decoding with only angle information. This lead us to question whether there is something fundamentally wrong with Message Passing decoding using the algorithms described in Section 4. This will be further discussed in Section 8.1.

7.2.4 Table Angle Decoder Using Floating Table

Simulations with the Table Angle Decoder from Section 5.2 were made using a floating table. The number of possible angles was set to *512*. This relatively small number of reconstruction points was chosen because of the large amount of time it took to create the different tables for shifting channel SNR. *512* was the highest number of reconstruction points that still kept table creation time fast enough. The 512 reconstruction points are evenly distributed on the unit circle. The results for different M-PSK signalling are presented in Figure 7.3.

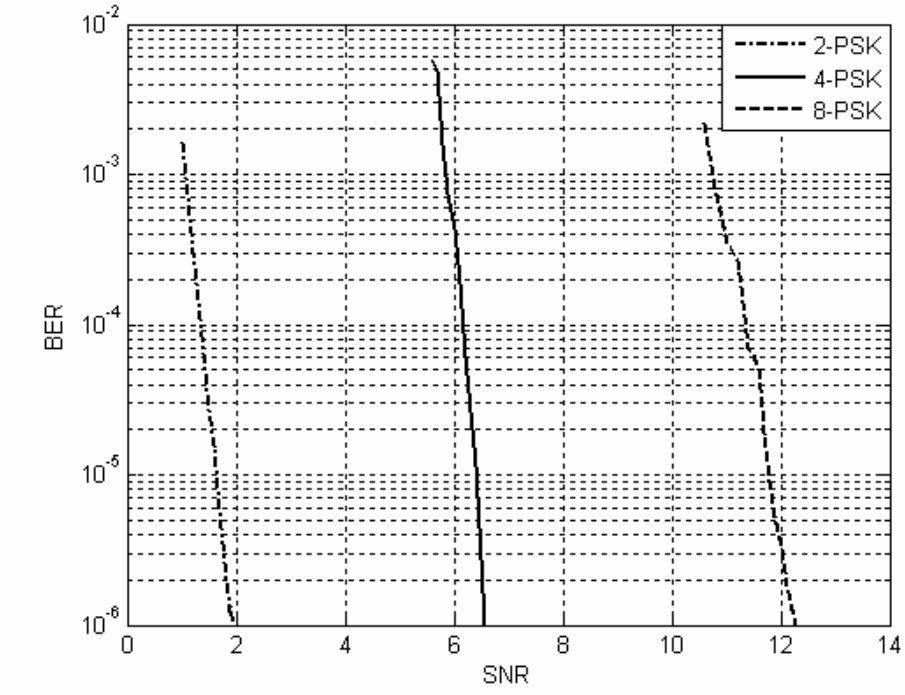


Figure 7.3 Simulations with Table Angle Decoder for the regular (3,6) LDPC code with codeword length 504 using floating table and M ={2, 4, 8}. The number of angles (reconstruction points) is 512. The maximum number of iterations is 80.

7.2.5 Table Vector Decoder Using Floating Table

The simulations with the Table Vector Decoder described in Section 5.2 were made using *512* reconstruction points. Now, in the Table Vector Decoder, the angular resolution is lower, but there are different amplitudes (lengths) which represents different reliabilities for the message. The Table Angle Decoder has higher angular resolution but all messages have the same reliability. The reconstruction points in the vector tables for 4-PSK used in this simulation were built with *16* different lengths and *32* angles, evenly distributed in the vector plane (see Figure 5.2 for an example). The created tables had a reconstruction point in the origin in the vector plane**.** The simulation results are presented in Figure 7.4.

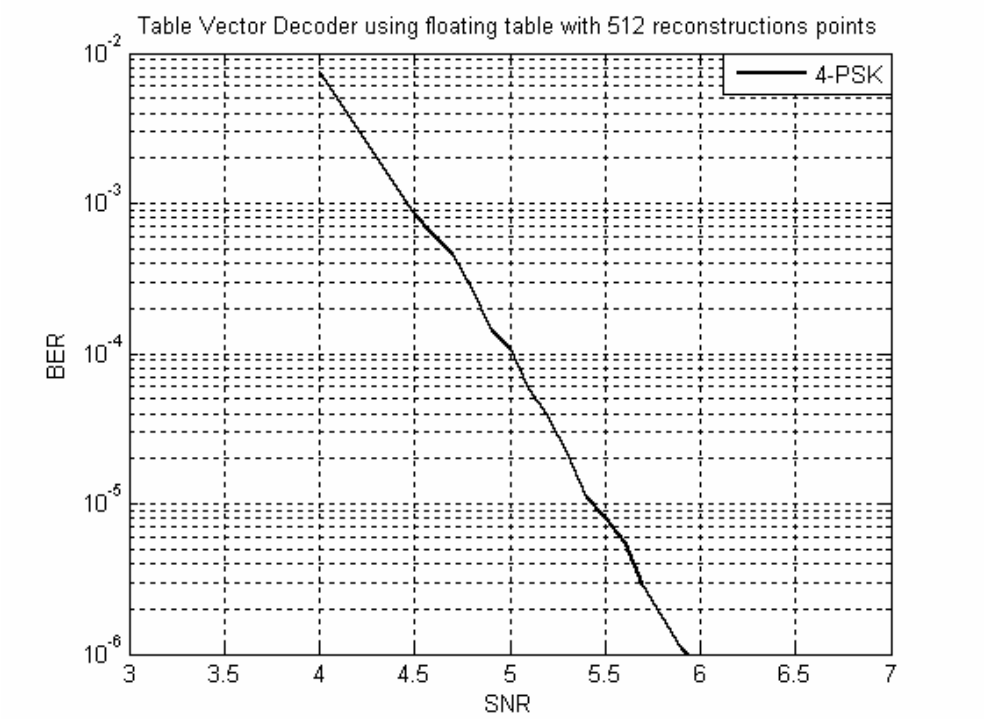


Figure 7.4 Simulations with Table Vector decoder for the regular (3,6) LDPC code with codeword length 504 and a floating table. The number of reconstruction points is 512, with 16 different angles and 32 different lengths.

7.2.6 Table Angle Decoder Using Fixed Table

In a real implementation of a Table Angle Decoder, it will not be preferable to calculate and keep multiple tables in storage (one for each SNR). Because then a different vector table for the decoder have to be calculated for each SNR value the channel is expected to have (Section 7.2.4 and Section 7.2.5). These simulations will instead create one angle table (fixed table) based on an SNR value which gives very low *BER* in Section 7.2.4 and 7.2.5 for the different M-PSK used. The reconstruction points (possibly angles) used in the tables are evenly distributed on the unit circle. These fixed tables (one for every $M = (2,4,8)$) are the used in the Table Angle Decoders for all shifting channel SNR. The results from the simulations are presented in Figure 7.5.

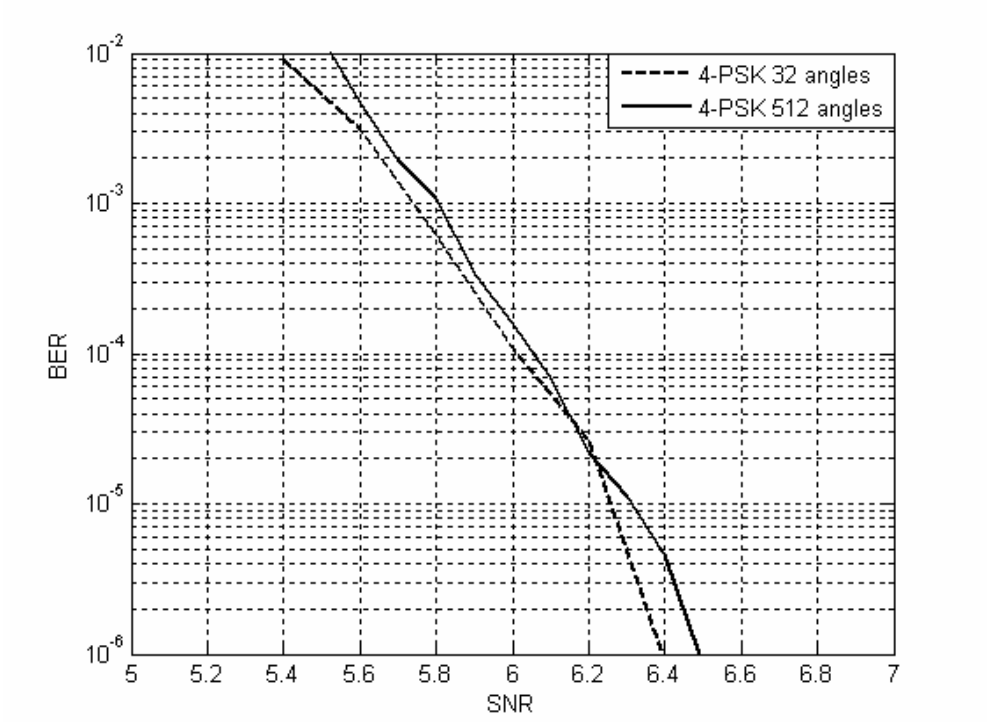


Figure 7.5 Simulations with Table Angle decoder for the regular (3,6) LDPC code with codeword length 504 and a fixed table. The angle resolutions for the tables are noted in the figure.

7.2.7 Table Vector Decoder Using Fixed Table

For the same reason as in Section 7.2.6, a fixed table is created and used for every order of modulation *M* in the Table Vector Decoder. The simulations were made for 4-PSK when having evenly distributed angles and lengths, and a reconstruction point in the origin in the vector plane. The simulation results are presented in Figure 7.6. The simulation results for 4-PSK when having evenly distributed angles and lengths but with no reconstruction point in the origin is presented in Figure 7.7. Finally the results for 8-PSK simulation with evenly distributed angles and lengths but with no reconstruction point in the origin in the vector plane is presented in Figure 7.8.

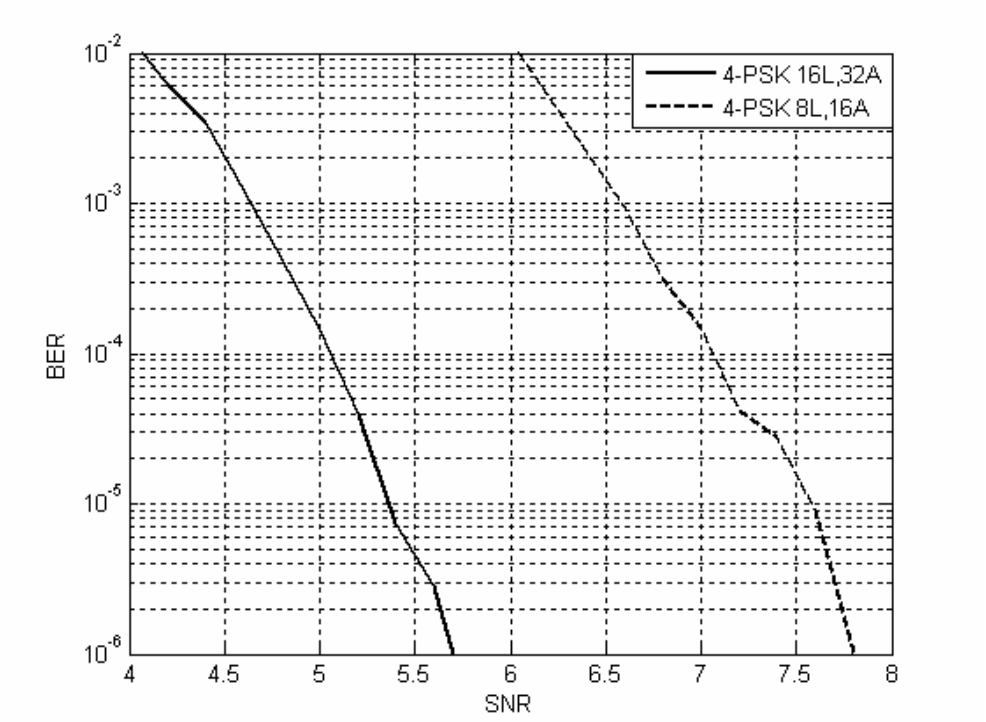


Figure 7.6 Simulations with Table Vector decoder for the regular (3,6) LDPC code with codeword length 504 and a fixed table and 4-PSK. The number of possible angles and lengths in the tables is denoted L and A in the figure.

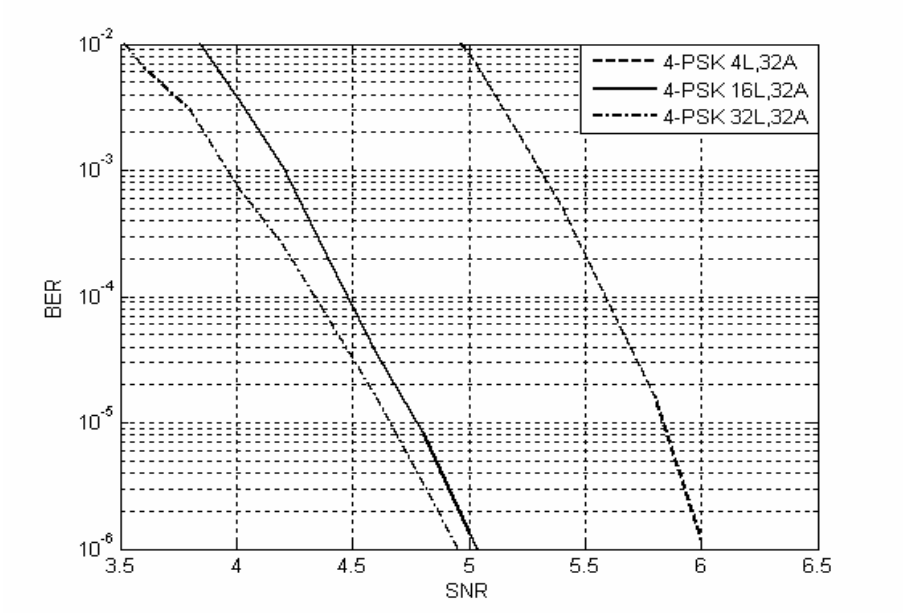


Figure 7.7 Simulations with Table Vector decoder for the regular (3,6) LDPC code with codeword length 504 and a fixed table and 4-PSK. The number of possible angles and lengths in the tables is denoted L and A in the figure.

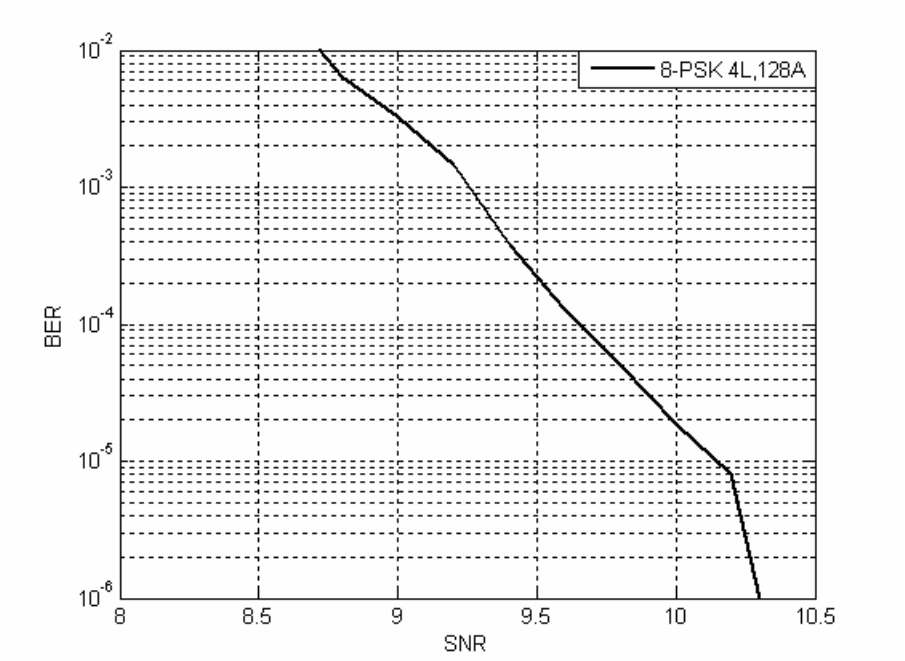


Figure 7.8 Simulations with Table Vector decoder for the regular (3,6) LDPC code with codeword length 504 and a fixed table and 8-PSK. The number of possible angles and lengths in the tables is denoted L and A in the figure.

8 Results and Analysis

This section will analyze the results from the simulations in Section 7. Suggestions on future work will also be given.

8.1 Why Does Not Angular Summation Work?

Here, we will try to give an explanation why Angular Sum Decoding does not work. It is not a proof in any sense, only a short discussion of our way of thinking when we stopped developing it.

To begin with, LDPC codes are using finite fields, while the Angular Sum Decoder is not. It is possible to decode with the Angular Sum Decoder using an LDPC code as long as only the all-zero codeword is sent, since this codeword also exist in this decoder. The all-zero codeword corresponds to sending the zero angles for all the codewords symbols.

The parity check in this decoder is a *modulo* 2π summation of the incoming nodes. As long as this summation is made using a finite set of possible angles (N angles), it will work in a ring. A finite set can be assumed when using computers, since only a fixed number of decimals are stored. It may not be clear that it is a ring, but if all elements are multiplied by $N/2\pi$, the elements would be integers ranging from 0 to N. The summation can be made as a *modulo* N summation. As mentioned in Example 3, a modulo summation of integers is a ring. N can be made very large and thus make the angle precision be close to infinite.

Out of the N possible angles, only M angles are corresponding to a symbol, and problems occur when another angle is received. It is possible to quantify the angle to the closest symbol angle, but in that case all channel information (soft information) is lost and only the symbol itself (hard information) can be passed as messages in the decoder. Another approach is to make all angles correspond to a symbol, thus getting an N-sized alphabet. This is what is done in the Angular Sum Decoder.

The problem with increasing the alphabet is that an uncoded signal would have an error probability that is corresponding to the distance between the angles. Having a very large set of allowed angles would require a very large SNR in order to get a low BER. In essence, an infinitely small angular difference would require an infinite SNR to get an arbitrary small error, and no coding would be able to fix that.

8.2 EXIT Chart Analysis

Because of the lack of working equations in the check node for the Table Angle- and Table Vector Decoder it was not possible to perform EXIT chart Analysis on these decoders. It is probably possible to perform EXIT chart analysis on the table based Table Angle- and Table Vector Decoder, but this will most certainly be a complex and time demanding solution and is beyond this thesis work. The main idea behind EXIT chart analysis of Table Vector- and Table Angle Decoder is briefly discussed together with possible approximations in Section 8.6.2. In this Master Thesis work, the EXIT chart analysis performed is on the ensemble when

using Belief Propagation (Section 3.2). The result is presented in Table 8.1. The analysis was made on the vector symbol representation instead of the binary representation of the M-PSK symbols (Section 3.3). The EXIT chart algorithm and calculation results are presented in Section 3.4.4 and Section 6. As expected, increasing the *M* possible M-PSK symbols on the unit circle for the modulation will increase the $SNR_{Threshold}$ for the ensemble used. When the symbols lie close to each other when modulated, a large SNR is needed to separate them at detection.

It seems like the increase for the $SNR_{Threshold}$ is linear when going from *2-PSK* to *4-PSK*, and from *4-PSK* to *8-PSK*. In both cases the $SNR_{Threshold}$ is increased by *5.4 dB* (Table 8.1).

M	$SNR_{Threshold}$
2	-2.2
4	3.2
8	8.6

Table 8.1 $SNR_{Threshold}$ for 2-, 4 – and 8-PSK using a regular (3,6) LDPC ensemble with Belief Propagation.

8.3 Simulations Results

The simulations in Section 7 were performed with the purpose of producing comparable results for the Belief Propagation-, Table Angle- and Table Vector Decoder, using different order of M-PSK modulated symbols. Some modifications on the Table Angle- and Table Vector Decoder were tested to see how they affected the simulation results (different tables). The three decoders with their modifications could then be compared using the simulation results. First, a comparison will be made on the SNR needed for low decoding error using the different decoders. Next, a complexity comparison between the decoders will be made. The regular (*3, 6*) LDPC code [18] used in these simulations is a code optimized for BPSK, but it does not matter since only the relative results between the decoders are of interest here. The main focus on the curves in the figures should be at $BER > 10^{-5}$, because when having *BER* lower than 10^{-5}, small random errors will have a large impact on the local appearance of the curve. The local "jumps" in the curves depends on an insufficient number of simulations. If, say, another million simulations (codewords sent and decoded) were executed, the resulting curves can then be expected to be smoother. This is of minor importance here, the general shape of the curves is still clear for simulations with 500 000 codewords. The use of the term Table Decoders refer to both the Table Angle Decoders and Table Vector Decoders.

8.3.1 Table Decoding with a Fixed Table

Simulations with Table Decoding using a fixed table independent of the SNR changes of the channel were performed in Section 7.2.6 and Section 7.2.7. This was done to see if the SNR used to design the table needs to be equal to the actual SNR of the channel in order for the decoder to perform well. If it is necessary to recalculate the table, or store several versions of the table (floating table, one per

SNR), then it could be argued that this type of decoder is too complex to justify its decoding performance. The SNR used to calculate the angle- and vector table were *6.4 dB*. The results are presented in Figure 8.1 for *4-PSK*.

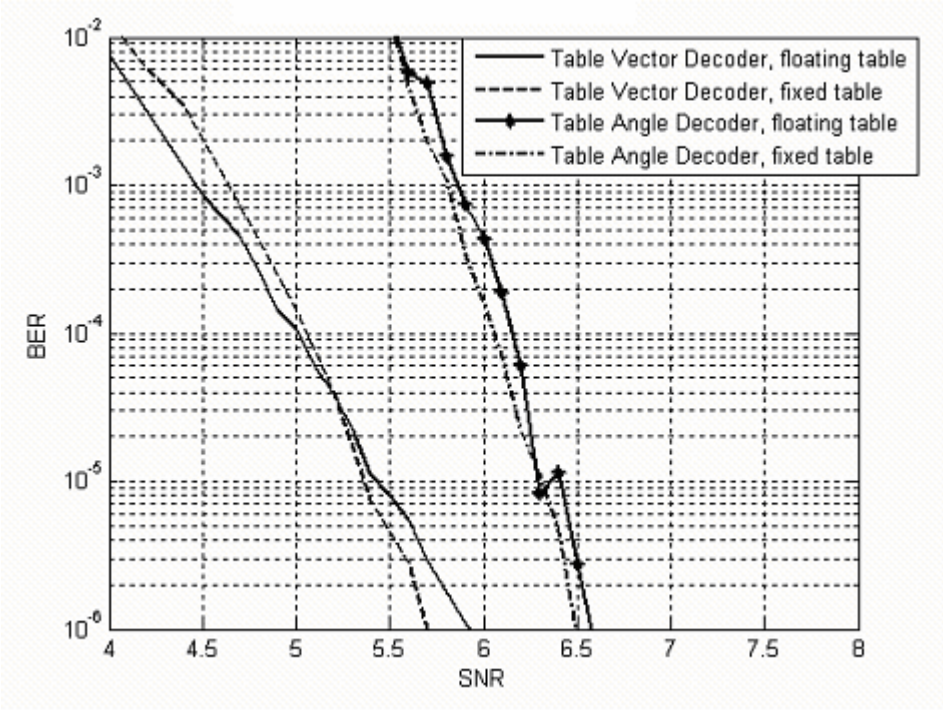


Figure 8.1 Table Angle Decoder and Table Vector Decoder performance for fixed and floating tables. Order of modulation is 4-PSK.

From Figure 8.1 one can draw the concussion that the performance of the decoder is approximately the same for both a fixed and floating table vector- and angle tables for *4-PSK*. Because of this, the Table Decoders using fixed tables will from now on be used when comparing decoding performances. This is because the lesser complexity in these decoders compared with the ones using a multiple set of tables (that is, a floating table) for each possible SNR. From now on, the terms Table Vector- and Table Angle Decoder will refer to Table Decoders with fixed tables. The possibility to use fixed instead of floating tables in the Table Decoders will give a large improvement of the decoders due to the reduced complexity.

8.3.2 2-PSK Simulations

In this section two different decoders are compared for *2-PSK* symbols in Figure 8.2. It is possible to see that the Table Angle Decoder needs approximately *2 dB* more than the Belief Propagation Decoder to achieve the same *BER*. *2-PSK* is not a good case for using a Table Decoder, since a Belief Propagation Decoder can be made very simple, and it is the best known decoder. This is why Table Decoding on *2-PSK* is only presented very briefly, just to show that it works, although not very good compared to the Belief Propagation Decoder.

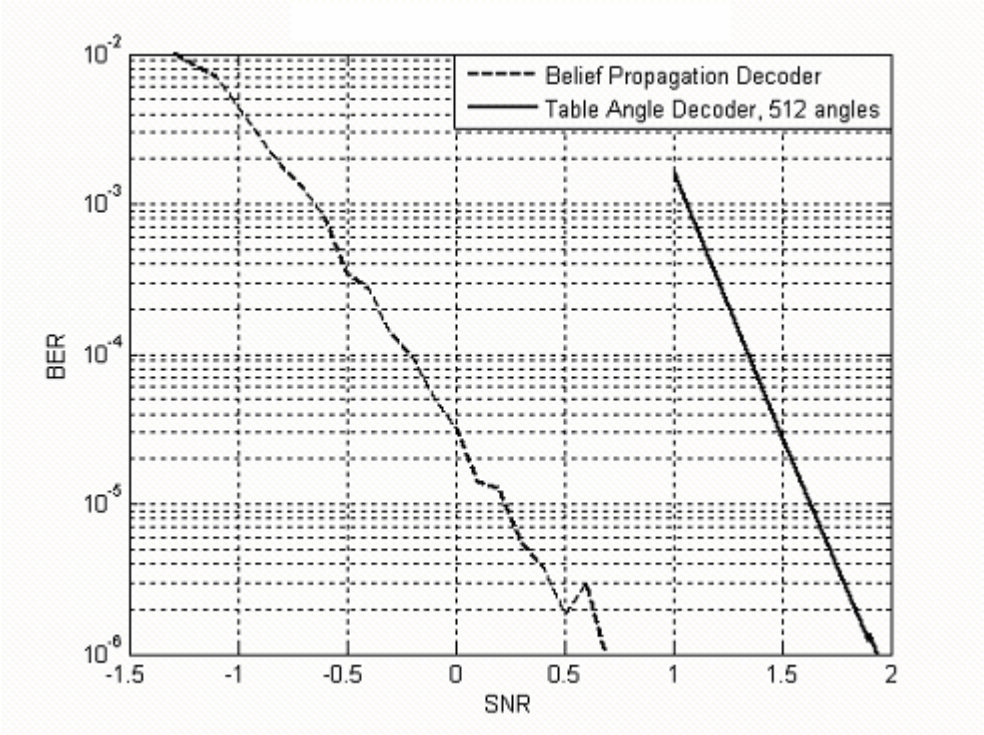


Figure 8.2 2-PSK simulations with the Belief Propagation- and Table Angle Decoder.

8.3.3 4-PSK Simulations

The first result that was noted when running 4-PSK was that it did not matter if a fixed table, designed for a sufficiently high SNR, was used instead of a floating table designed for every shifting SNR (Figure 8.1). This lead to the conclusion that it is possible to use a single fixed table designed for a specific M-PSK signalling with approximately the same error correction performance. Another thing that was noted was that there seemed to be a limit on how many angles that was needed. When the Table Angle Decoder was used, this limit was measured to 32 angles. In Figure 8.3, it can be seen that all three table designs have almost the same performance. The differences can be explained by the relatively low number of simulations (codewords sent) that were made. In Figure 8.4, the simulation using 32A, 16L (Table Vector Decoder) can be considered as a "refined" Table Angle Decoder, now with 16 lengths as well. This increases performance by 0.5-0.7 dB

When simulating using the Table Vector Decoder, the reconstruction points could not be removed as easily. Figure 8.4 shows the effects of cutting the number of lengths and angle to half, thus reducing the number of reconstruction points to ¼. A large drop in performance was noted immediately. Since 8L, 16A is worse than 1L, 32A, it indeed seems like 32A is the minimum angular resolution needed for 4-PSK.

A very important thing was discovered when trying to improve the Table Vector Decoder. Removing reconstruction points from the origin gave an increase in performance with up to 1dB when using 4-PSK (Figure 8.5). These reconstruction points were placed evenly in the range of $\{0.00625,\ldots,2\}$. Tests with other placements (with and without points in the origin) did not give as good results.

Finally, it was noted that an increase in the number of reconstruction points could give an increase in performance. The results of having 2048 instead of 512 reconstruction points in the Table Vector Decoder can be seen in Figure 8.6. However, it was not easy to find a good placement of the reconstruction points, and it is probably possible to find points that give better results. The best Table Decoder found was the Table Vector Decoder using 2048 reconstruction points. It was performing about 1dB worse than the Belief Propagation Decoder (Figure 8.7).

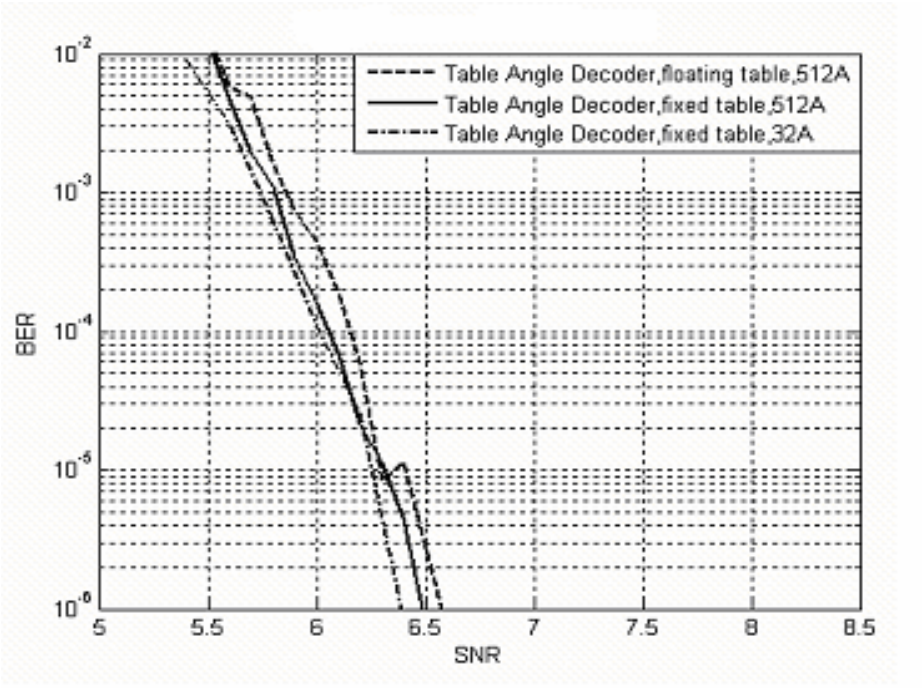


Figure 8.3 Comparison of fixed table, floating table, and reduced number of reconstruction points for 4-PSK.

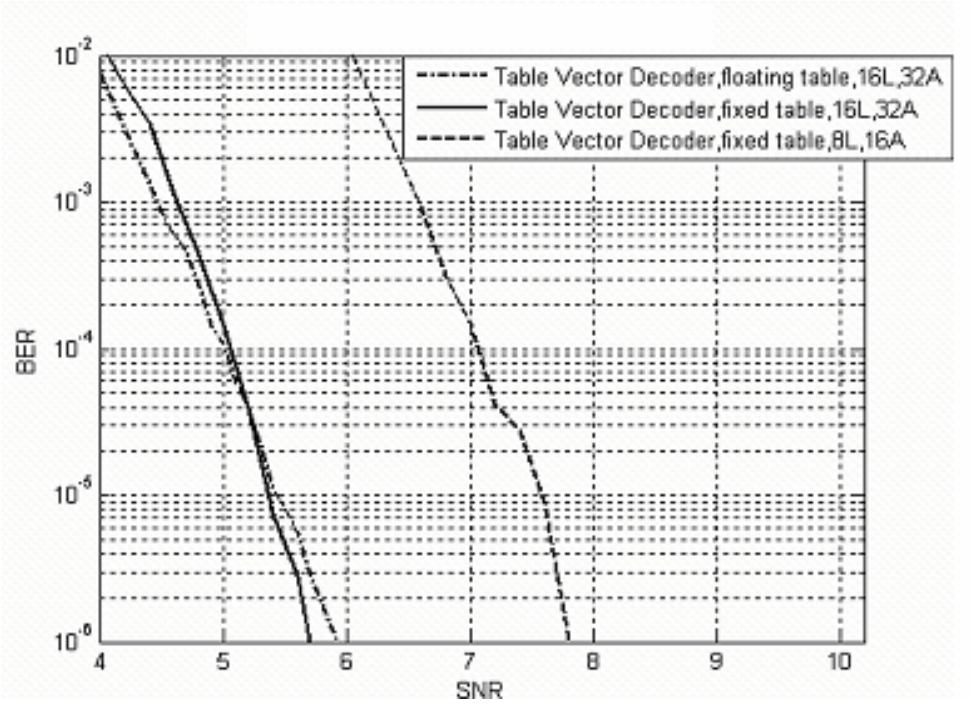


Figure 8.4 4-PSK simulations with fixed and floating tables, and reduced numbers of reconstruction points. All tables have reconstruction points in the origin.

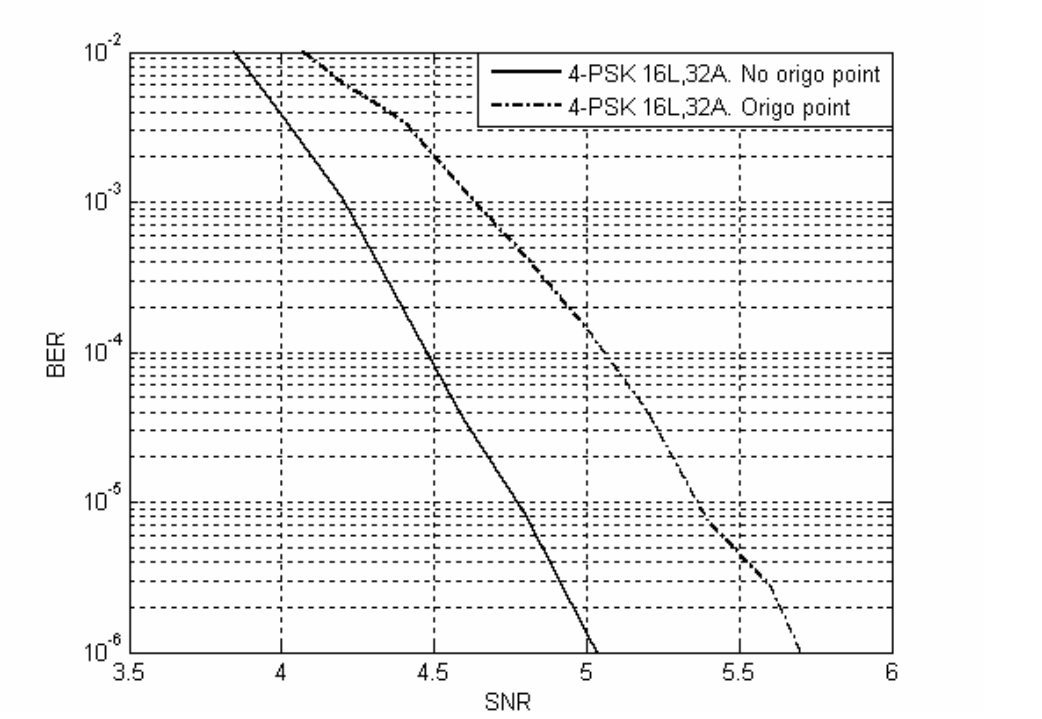


Figure 8.5 4-PSK simulation with and without reconstruction points in the origin.

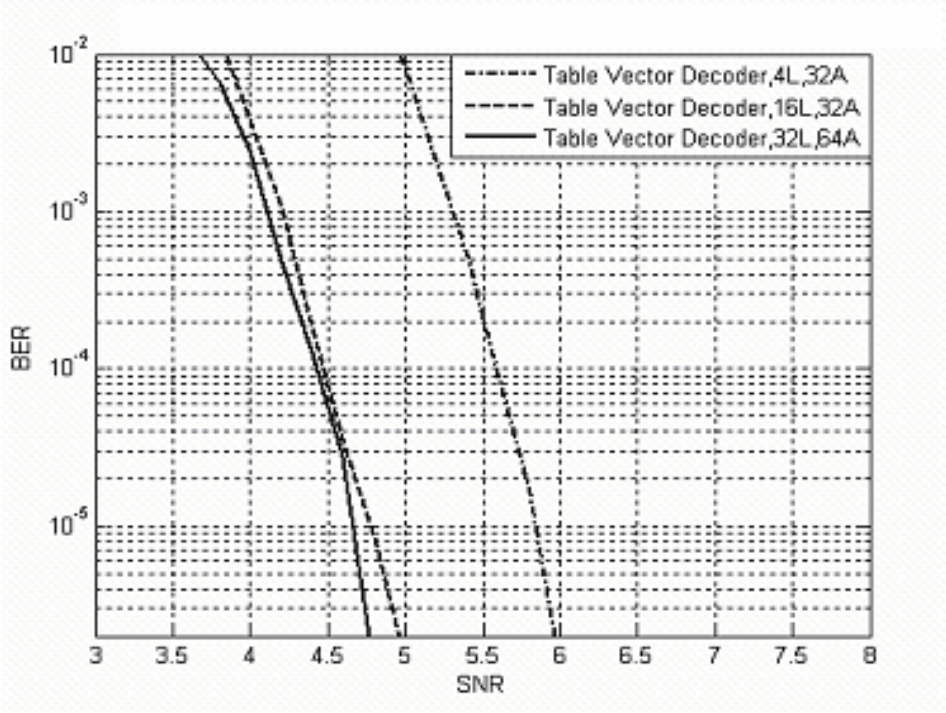


Figure 8.6 4-PSK simulations without reconstruction points in the origin.

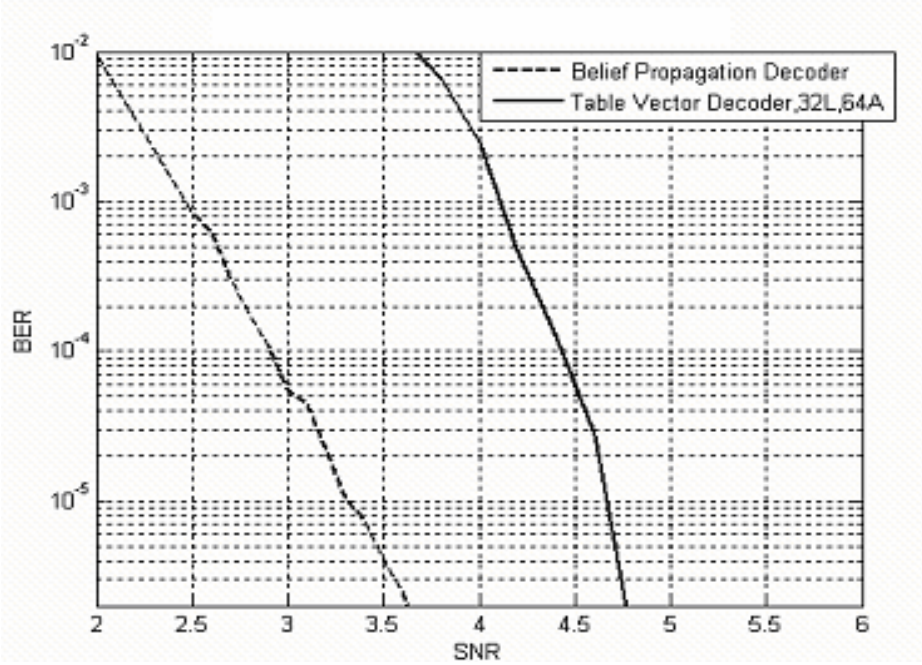


Figure 8.7 Comparison between the Belief Propagation Decoder and the best Table Decoder found.

8.3.4 8-PSK Simulations

In Figure 8.8, the *BER* curves as function of SNR for the three different decoders depending on SNR when using *8-PSK* signalling are presented. The Table Vector Decoder needs approximately 1 dB more than the Belief Propagation Decoder to achieve the same *BER*. The Table Vector Decoder performs approximately 2 dB better than the Table Angle Decoder. The Table Angle- and Table Vector Decoder both have *512* reconstruction points but with different placement in the plane as in the *4-PSK* case. There is an increase in the time that it takes to make the table when using higher order modulations, which made it difficult to make many simulations using 8-PSK. However, a Table Vector Decoder that performed about 1 dB worse than the Belief Propagation was found. The lengths were placed evenly in the range $\{0.333,\ldots,1.333\}$. The decoding itself is, however, much faster than Belief Propagation, which makes this Table Vector Decoder a very interesting approximation of the Belief Propagation Decoder.

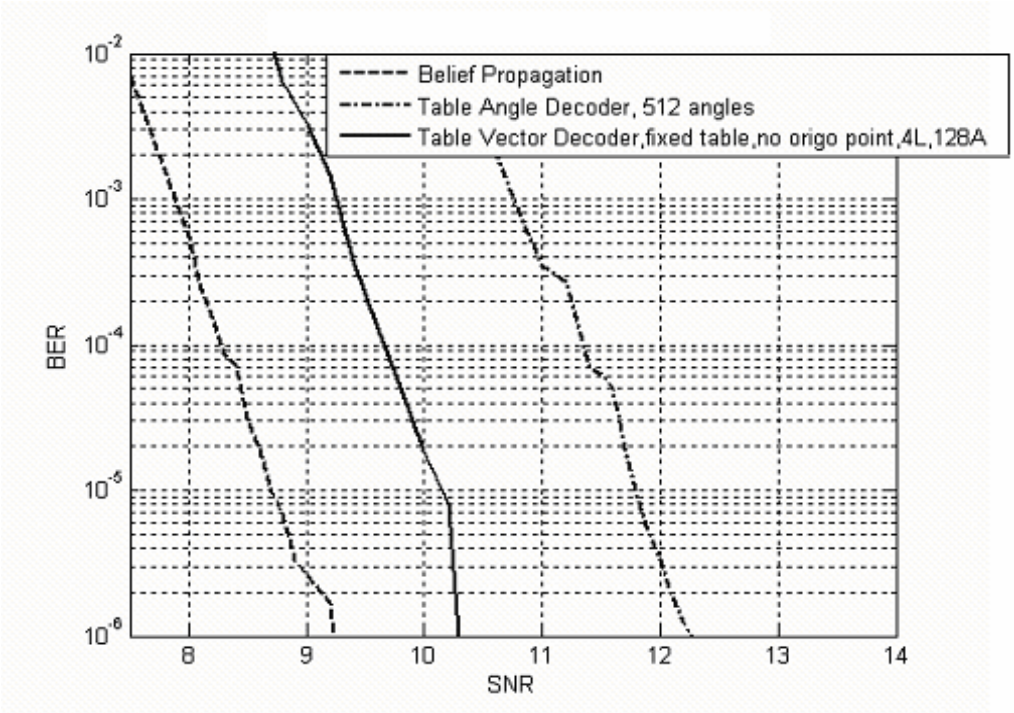


Figure 8.8 8-PSK simulations with Belief Propagation, Table Angle Decoder, and Table Vector Decoder.

8.4 Analysis of Simulation Results

With the results from Section 8.3, we will now analyze the decoding performance for the three decoders for $M=\{2, 4, 8\}$, with *BER* as the quality parameter, depending on the SNR. From the diagrams in Section 8.3, we can see that, not very surprisingly, the Belief Propagation Decoder performs better than the Table Decoders. It is to be expected, since the Table Decoder is an approximation of the Belief Propagation Decoder, with information loss in the quantization made on the messages in The Table Decoder (Section 5.2.1). With a larger amount of reconstruction points, and/or with good reconstruction point placement, the performance of the Table Decoder gets closer to the Belief Propagation Decoder. An en-

couraging result from the simulations in Section 8.3.1 is that the Table Decoder only needs to store one pre-calculated table. The performance gap is slim between Table Decoders with a fixed table and Table Decoders with floating tables.

An illustrative figure describing the similar appearance between the angle tables created for different SNR but for the same *M* is presented in Figure 8.9. The surfaces are approximately the same, but the surface for the 3 dB plot is more rugged.

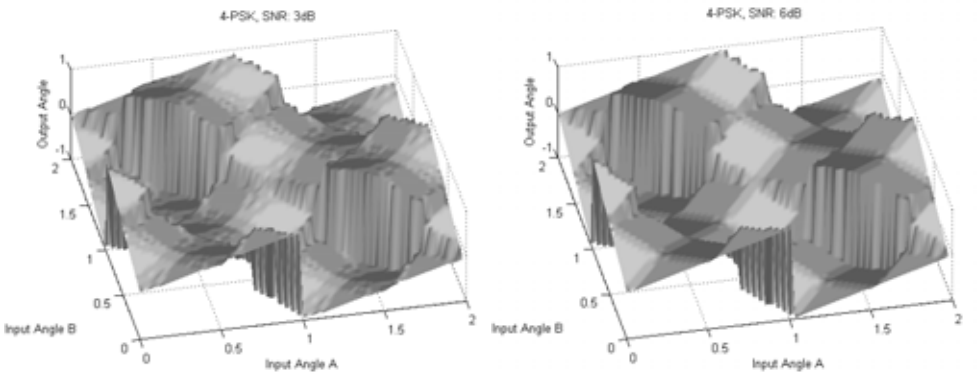


Figure 8.9 The angle tables for 3 dB and 6 dB over 4-PSK signaling.

A very interesting result is also that the Table Vector Decoder performs significantly better than the Table Angle Decoder for all SNR, even though they both have 512 reconstruction points, just with different placement in the plane. The Table Vector decoder performs better when there is no reconstruction points in the zero point in the plane. This could have something to do with that it is better for the decoder to quantize messages to a vector other than the null vector, because the point in the origin gives no information about the probability of a symbol. In other words; it is better that the decoder guesses on the possible symbol, even with very little information, than to "say" that the message could be any of the possible symbol with equal possibility (null vector). Much could probably be gained by careful design of the Table Decoders fixed table, regarding the placement and amount of reconstruction points. The above implies that it would be possible to optimize a Table Vector Decoder (the best Table Decoder) for an *M* and a channel by placing the limited reconstruction points in a certain pattern on plane. This Table Vector Decoder would probably perform close to the Belief Propagation Decoder with respect to the *BER* if the number of reconstruction points is sufficiently large.

The next quality parameter to take into consideration when comparing the best Table Decoder, the Table Vector Decoder, with the Belief Propagation Decoder is the decoding complexity for the decoders. Decoding complexity is an important quality parameter because it gives a measurement of the time and space (hardware implementation) consumption for the decoding algorithm. When considering this quality parameter, the advantage of the Table Vector Decoder starts to show. The increasing decoding complexity of the Belief Propagation algorithm when going from *2-PSK* to *4-* and *8-PSK* is far from linear [15]. Belief Propagation with *4-PSK* symbols is four times more complex than for *2-PSK*, and Belief

Propagation for *8-PSK* is *16* times as complex as for *2-PSK*. The Table Decoders have constant decoding complexity for *2*-, *4*- and *8-PSK* symbols, so if one only looks at the decoding complexity it is obvious that the largest complexity gain when changing from a Belief Propagation- to a Table Decoder is achieved for *8-PSK*. In the *2-PSK* case the Belief Propagation Decoder is less complex than the Table Decoder. That is because BPSK is a special case since it has only *2* probabilities which can be expressed as a single Likelihood Ratio [13].

There is log-linear loss in decoding performance (*BER*) when using Table Vector decoding instead of Belief Propagation decoding for *2*-, *4*- and *8-PSK*. The loss for the best Table Vector Decoders tested in Section 7.2 in SNR about 1 dB for *4-PSK* and *8-PSK*. One could therefore argue that The Table Vector Decoder for *8-PSK* is the best choice of the three decoders, since this decoder gives the best complexity gain comparing to the SNR loss when going from Belief Propagation to Table Vector Decoding. The Table Vector Decoder is probably a better choice than the Belief Propagation Decoder even in the *4-PSK* case. It is possible to improve the results for 4-PSK and 8-PSK even further by running more simulations and finding a better reconstruction point placement. However, using *2-PSK* gives the Belief Propagation a clear advantage since it has a smaller decoding error and a less complex decoder than the Table Vector Decoder. In this thesis, we have not simulated much at all using 2-PSK signalling, since there is no gain to be made in this case by changing from Belief Propagation to Table Decoding.

8.5 Analysis of the Implementation

There are many aspects to consider when designing a Table Decoder. In this chapter, we will try to bring focus to some important discoveries we made when we made our Table Decoder. It is not a complete description of our system, but a discussion of some problems and solutions we found on the way.

The calculation complexity of the Table Decoder only depends on the LDPC code it is designed to decode, not on the number of reconstruction points (size of the table). However, the memory requirements are based on the number of reconstruction points. More precisely, it needs $O(N^2)$ memory cells where N is the number of reconstruction points in the table.

Although our implementation actually sends real value angles and lengths, it is possible to minimize the messages sent between the nodes by only sending the indices of the reconstruction points. Doing this will limit the information that needs to be passed between the nodes in the decoder to $log_2 N$ bits.

The design of the reconstruction points can have a great impact on the decoding performance. A good example can be found in Figure 8.8, which shows a gap between using only angles and using angles and lengths in *8-PSK* signalling.

Some notes for good reconstruction point design:

1. Use even length differences. Having two inputs to the variable node pointing in the same direction should always increase the length. The length is a reliability measure, and having two pointing in the same direction should increase the reliability.

2. Make sure there are lengths on both sides of *1*, this greatly improves the decoder. If all values are less than *1*, most inputs will start at the maximum possible length, and cannot improve.
3. Use a multiple of *M* angles. Split the angles evenly into *M* groups and distribute each group around a constellation point. Distribute all groups in the same way. Doing so will make the decoder act the same way regardless of the symbol received. Our implementation spread all angles evenly around the unit circle.
4. Avoid putting reconstruction points in the origin. According to our simulations, they will make the error correcting capabilities worse.

8.6 Future Work

Simulations were done using a large number of reconstruction point constellations. However, it is very likely that there exist constellations that have better error correcting performances. Using methods for designing Vector Quantizers (VQs) one could probably find the optimal set of reconstruction points.

8.6.1 Blackbox Modelling for Check Nodes Using Equations

An alternative solution to the table lookup presented in Section 5.2.1 is to find an equation depending on v_a, v_b, SNR and M, which returns u. This should give approximately the same result as if the Table Vector Decoder method was used. The advantage with an equation approach instead of a table lookup approach for the decoder is that it is then not necessary to store a Table for each different M and SNR, just an equation depending on these variables. If an equation is found that approximates the Belief Propagation Decoder very well, it may lead to a decoder with less complexity than the Table Decoder and the Belief Propagation Decoder.

The surfaces in Figures 5.7 and 5.8 have numerous flat surfaces. It is difficult to describe them in a single function, but it is possible that using neural networks or linear discriminant functions [6] can be a good approach for calculating them fast.

8.6.2 Density Evolution on the Table Decoder

It would be of great interest to calculate the $SNR_{Threshold}$ for different ensembles using the Table Angle- and Table Vector Decoder. These analysis results could then be compared with the results for the Belief Propagation Decoder (Section 8.2) for the different ensembles. But to perform a more exact Density Evolution analysis one would need the Variable- and Check node functions, since the Density Evolution algorithm is developed out of these functions [19]. We have developed the variable node function but not the check node function. Density Evolution will probably be possible if Black box Modelling Using Equations (Section 8.6.1) is first developed for the Table Angle- and Table Vector Decoder. This would give the check node function needed to perform the Density Evolution.

An alternative solution that could work would be to use tables for the angles like in the Table Angle Decoder, and from all the resulting Angle combinations out of the check node Develop the discrete message pdf for every Density Evolu-

tion iteration. This approach will probably result in rather complex and time demanding computer calculations for *4-* and *8-PSK*. It is annoying to have to wait for the computer during time demanding calculations, but since DE is an analysis tool, not a decoding method, it only has to be performed once to establish the threshold.

Density Evolution could probably be performed using table representation of the check node function. However, it would probably have to be a fairly large table (high resolution) to give useful results.

9 Conclusions

The main focus in this Master's Thesis has been to create a Message Passing decoder that uses angles, and possibly lengths, as the messages. Two decoder types were constructed and one of them, the Table Decoder, was decoding correctly when angles only, and angles with lengths, were used.

9.1 Angular Sum Decoder

The first approach, using a decoder with simple angular summations (Section 4), did not produce any successful results at all (Section 7.2.2). Although not mathematically proven, it is not likely that a decoder using angular summation will work (Section 8.1). One important result from this decoder was made. In Section 4.1, it was proven that vector summation is equivalent to the variable node operation made in Belief Propagation, when the probability distributions are taken from the channel.

9.2 Table Decoder

The approach to solve the problem encountered in Section 9.1 was to imitate a Belief Propagation Decoder (Section 3.2) by using a pre-calculated lookup table (Section 5) with angle and length inputs. This approach made decoding possible, but demanded a higher SNR than Belief Propagation. The performance gap is however reducible, and improvements to the decoder have been discussed Section 8.5 and Section 8.6.1. It was also showed in Section 8.3.1 that only one fixed table per modulation type (M-PSK) has to be pre-calculated as long as the SNR used to calculate the fixed table is sufficiently high. The fixed table can then be used for all SNR values on the channel for a specified modulation. This gives a large complexity gain for the decoder. An extra advantage is that the complexity of this decoder is constant, regardless of modulation technique (e.g. {2, 4, 8}-PSK), which gives the decoder an advantage over the Belief Propagation Decoder, since its complexity has a quadratic increase for higher order modulation. These two decoders are called Table Angle- and Table Vector Decoder.

9.3 Performance of the Table Angle- and Table Vector Decoder

The Table Vector Decoder performs significantly better than the Table Angle Decoder (Section 8.3), while both can be designed to have the same complexity. However, it takes a great deal of reconstruction point design to make a good Table Vector Decoder. In some cases, the performance is decreased even if extra reconstruction points are added. However, the table only needs to be designed once. With a careful design and a proper amount of reconstruction points in the vector table, the Table Vector Decoder could probably perform very close to the Belief Propagation Decoder, with far less decoding complexity in the *8-PSK* case (Section 8.4).

9.4 EXIT Chart Calculations

The method of using a table lookup made it unfeasible to analyze the Table Decoders with Density Evolution or EXIT chart. But analysis was made on the regular *(3, 6)* ensemble [18] for Belief Propagation (Section 8.2), and showed that the specific code used throughout the thesis was a good one, even for higher fields.

Appendix A

A.1 Conversion from Vector (Angle and Length) Representation to Probability Representation

Let m_i be the vectors on the unit circle representing the M possible M-PSK symbols for $i \in \{0 \quad 1 \quad ... \quad M-1\}$. Let $\overline{m}$ be an arbitrary vector on the vector plane and $P_{\overline{m}}$ be the M-length probability vector, where all of its elements $P_i(m_i \mid \overline{m})$ are the probabilities that m_i is sent given that $\overline{m}_i$ is received.

When sending $m_i \in \{m_0 \quad m_1 \quad ... \quad m_{M-1}\}$ through a channel with noise, the angle and length of the symbol will be distorted in a certain way depending on the SNR and the type of channel noise Θ. The received signal vector $m_{received}$ representing the distorted M-PSK message will then with great probability point at another point in the vector plane then the sent m_i. This is illustrated by an example in Figure A.1. Depending on the noise distortion on m_{sent}, the probability changes that $m_{received}$ is one of the m_i possible M-PSK symbols. There is one joint probability $P_i(m_i \mid m_{recieved})$ for each possible symbol m_i, i.e. the probability that that one of each M possible symbols m_i is sent, when $m_{received}$ is the received vector.

$$P_{Sent} = [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

$$m_{\mathrm{Re}ceived} = m_1 + \Theta$$

$$P_{\mathrm{Re}cieved} = \begin{matrix}[0.1125 & 0.7156 & 0.1665 & 0.0033.. \\ ..0.0001 & 0.0000 & 0.0000 & 0.0019]\end{matrix}$$

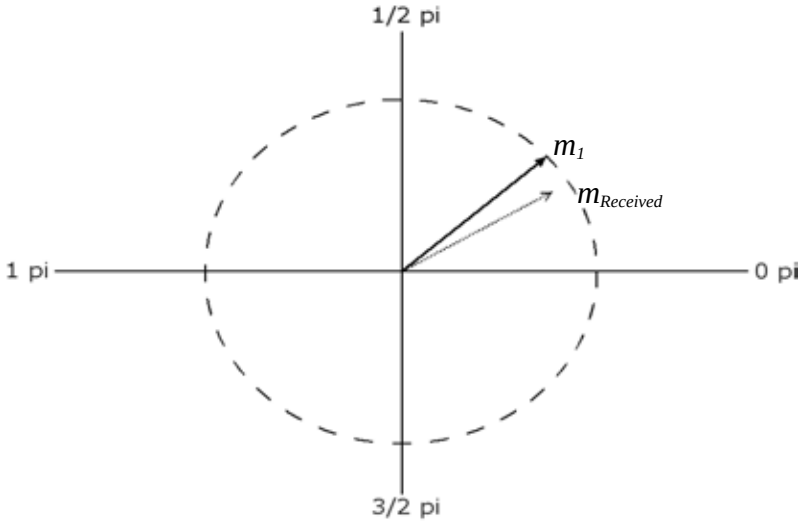


Figure A.1 Sending the 8-PSK '1' symbol through a channel with Gaussian noise.

When having an arbitrary vector $\overline{m}$ in the vector plane, $\overline{m}$ can be represented by a probability vector $P_{\overline{m}}$ (Equation A.1.1). Here $\overline{m}$ is the general case of $m_{received}$.

$$P_{\overline{m}} = \left[P_0\left(m_0|\overline{m}\right) \quad P_1\left(m_1|\overline{m}\right) \ldots P_{M-1}\left(m_{M-1}|\overline{m}\right)\right] \tag{A.1.1}$$

With length M, and each element in the vector giving the probability for $\overline{m}$ being one of the M possible M-PSK vectors m_i.

$P_{\overline{m}}$ can be calculated by calculating every element $P_i(m_i|\overline{m})$ in Equation A.1.1 for every $i \in \{0 \quad 1 \quad \ldots \quad M-1\}$ with the following equations:

First the geometrical distances d_i from the vector point of $\overline{m}$ to all M possible symbol points for m_i on the unit circle is calculated with Equation A.1.2. (Figure A.2)

$$d_i^2 = \left|\overline{m} - m_i\right|^2 \tag{A.1.2}$$

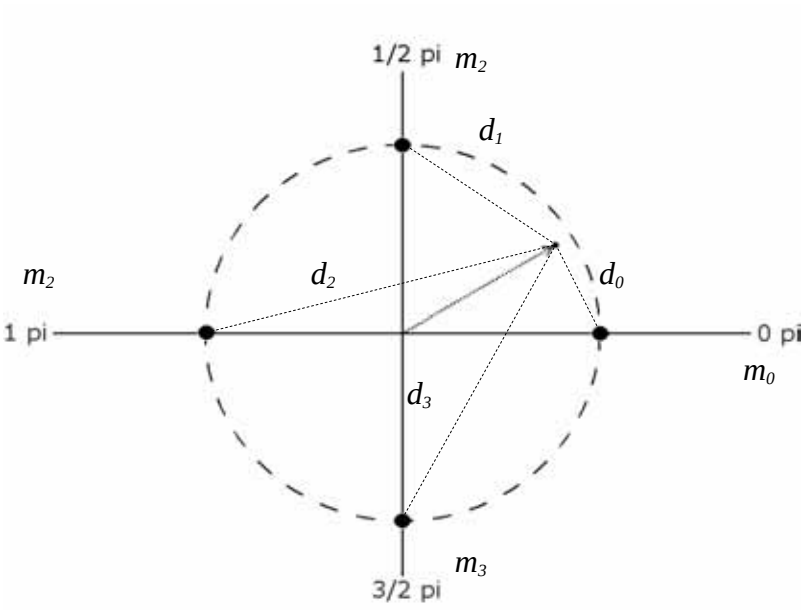


Figure A.2 The geometrical distances to the four different signal points m_i where $i \in \{0,1,2,3\}$.

The following holds if the noise is AWGN with known noise variance σ.

$$P\left(\overline{m}|m_i\right) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-d_i^2}{2\sigma^2}} \tag{A.1.3}$$

Bayes' rule gives:

$$P\left(m_i|\overline{m}\right) = \frac{P\left(\overline{m}|m_i\right)P\left(m_i\right)}{\sum_{j=0}^{M-1} P\left(\overline{m}|m_j\right)P\left(m_j\right)} \tag{A.1.4}$$

where $P(m_i)$ and $P(m_j)$ is the probability that m_i and m_j is sent. Inserting Equation A.1.3 into Equation A.1.4 and assuming that all symbols have equal probability to be sent gives

$$P\left(m_i|\overline{m}\right)=\frac{\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-d_i^2}{2\sigma^2}}}{\sum_{j=0}^{M-1}\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-d_j^2}{2\sigma^2}}}$$

$$\Leftrightarrow$$

$$P\left(m_i|\overline{m}\right)=\frac{e^{\frac{-d_i^2}{2\sigma^2}}}{\sum_{j=0}^{M-1}e^{\frac{-d_j^2}{2\sigma^2}}} \tag{A.1.5}$$

Equation A.1.5 and Equation A.1.2 are used to calculate every element $P(m_i|\overline{m})$ in $P_{\overline{m}}$ (Equation A.1.1) for $i \in \{0 \quad 1 \quad ... \quad M-1\}$.

A.2 Conversion from Probability Representation to Vector (Angle and Length) Representation

The conversion from probability representation $P_{\overline{m}}$ of M-PSK symbols into vector representation $\overline{m}$ on the vector plane is achieved by first rewriting Equation A.1.5 to (11).

$$\sum_{j=0}^{M-1} e^{\frac{-d_j^2}{2\sigma^2}} = \frac{e^{\frac{-d_i^2}{2\sigma^2}}}{P\left(m_i|\overline{m}\right)}$$

$$\Leftrightarrow$$

$$\sum_{j\in U} e^{\frac{-d_j^2}{2\sigma^2}} + e^{\frac{-d_i^2}{2\sigma^2}} - \frac{e^{\frac{-d_i^2}{2\sigma^2}}}{P\left(m_i|\overline{m}\right)} = 0 \qquad U = \{0\ldots(N-1)\neg i\}$$

$$\Leftrightarrow$$

$$\sum_{j\in U} e^{\frac{-d_j^2}{2\sigma^2}} + \frac{P\left(m_i|\overline{m}\right)-1}{P\left(m_i|\overline{m}\right)} e^{\frac{-d_i^2}{2\sigma^2}} = 0 \qquad \text{(A.2.1)}$$

Setting up a system of equations with Equation A.2.1 for $i \in \{0 \quad 1 \quad ... \quad M-1\}$, gives a nonlinear system of equations Equation A.2.2.

For simplicity the following notations are used:

$$p_i = P\left(m_i|\overline{m}\right)$$

$$e_i = e^{\frac{-d_i^2}{2\sigma^2}}$$

By setting up all equations for every *i*, the following equations are received.

$$\begin{cases} \frac{p_0-1}{p_0} e_0 + e_1 + \ldots + e_{M-1} = 0 \\ e_0 + \frac{p_1-1}{p_1} e_1 + \ldots + e_{M-1} = 0 \\ \qquad\qquad \ldots\ldots\ldots \\ e_0 + e_1 + \ldots + \frac{p_{M-1}-1}{p_{M-1}} e_{M-1} = 0 \end{cases} \qquad \text{(A.2.2)}$$

Given the joint probabilities p_i, it is not possibly to develop Equation A.2.2 into an explicit solution for d_i. A numerical solution with the Newton Raphson method is also very difficult to achieve because of the extreme nonlinearity of the system of equations with respect to d_i.

A.3 Proof That Vector Addition is Equal to Variable Node Operation in Belief Propagation

Assume M-PSK signalling over an AWGN channel with constant variance σ^2 and equal probability for all signals to be sent.

Let *I* denote the In-phase signal value in the vector model, and *Q* denote the Quadrature signal value.

Bayes' Rule returns the probability that symbol m_i was sent, given that the received signal is $\overline{m}_A$.

$$P(m_i|\overline{m}_A) = \frac{P(\overline{m}_A|m_i)P(m_i)}{\sum_{j=0}^{M-1} P(\overline{m}_A|m_j)P(m_j)}. \tag{A.3.1}$$

Equal probability to send all signals gives $P(m_i) = P(m) \forall i$, so

$$P(m_i|\overline{m}_A) = \frac{P(\overline{m}_A|m_i)}{\sum_{j=0}^{M-1} P(\overline{m}_A|m_j)}.$$

AWGN channel gives

$$P(\overline{m}_A|m_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-d_{Ai}^2}{2\sigma^2}}$$

where d_{Ai}^2 is the distance between symbol *i*'s constellation point and the received signal vector $\overline{m}_A$.

$$d_{Ai}^2 = \left(I_A - \cos\left(\frac{2\pi i}{M}\right)\right)^2 + \left(Q_A - \sin\left(\frac{2\pi i}{M}\right)\right)^2 =$$
$$I_A^2 + Q_A^2 + 1 - 2I_A \cos\left(\frac{2\pi i}{M}\right) - 2Q_A \sin\left(\frac{2\pi i}{M}\right)$$

$P(m_i|\overline{m}_A)$ can be simplified further:

$$P(m_i|\overline{m}_A) = \frac{P(\overline{m}_A|m_i)}{\sum_{j=0}^{M-1} P(\overline{m}_A|m_j)} = \frac{\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-d_{Ai}^2}{2\sigma^2}}}{\sum_{j=0}^{M-1} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-d_{Aj}^2}{2\sigma^2}}} = \frac{e^{\frac{-d_{Ai}^2}{2\sigma^2}}}{\sum_{j=0}^{M-1} e^{\frac{-d_{Aj}^2}{2\sigma^2}}} \tag{A.3.2}$$

$e^{\frac{-d_{Ai}^2}{2\sigma^2}}$ can be rewritten as

$$e^{\frac{-d_{Ai}^2}{2\sigma^2}} = e^{\left[\frac{-\left(I_A^2+Q_A^2+1-2I_A\cos\left(\frac{2\pi i}{M}\right)-2Q_A\sin\left(\frac{2\pi i}{M}\right)\right)}{2\sigma^2}\right]} = e^{\left[\frac{-\left(I_A^2+Q_A^2+1\right)}{2\sigma^2}\right]} e^{\left[\frac{I_A\cos\left(\frac{2\pi i}{M}\right)+Q_A\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}$$

inserted in Equation A.3.2 gives:

$$P\left(m_i\middle|\overline{m}_A\right) = \frac{e^{\left[\frac{-\left(I_A^2+Q_A^2+1\right)}{2\sigma^2}\right]} e^{\left[\frac{I_A\cos\left(\frac{2\pi i}{M}\right)+Q_A\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{\sum_{j=0}^{M-1} e^{\left[\frac{-\left(I_A^2+Q_A^2+1\right)}{2\sigma^2}\right]} e^{\left[\frac{I_A\cos\left(\frac{2\pi j}{M}\right)+Q_A\sin\left(\frac{2\pi j}{M}\right)}{\sigma^2}\right]}} = \frac{e^{\left[\frac{I_A\cos\left(\frac{2\pi i}{M}\right)+Q_A\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{\sum_{j=0}^{M-1} e^{\left[\frac{I_A\cos\left(\frac{2\pi j}{M}\right)+Q_A\sin\left(\frac{2\pi j}{M}\right)}{\sigma^2}\right]}}$$

Variable node operations in a Belief Propagation Decoder are performed as a symbol by symbol probability multiplications, whit normalized results. Assuming the probabilities are taken from the channel, this would look like:

$$\frac{1}{\alpha} P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right) \quad i \in \{0\ldots M-1\} \quad \overline{m}_A = \begin{pmatrix} I_A \\ Q_A \end{pmatrix} \quad \overline{m}_B = \begin{pmatrix} I_B \\ Q_B \end{pmatrix}$$

and α is set so

$$\sum_{i=0}^{M-1} \frac{1}{\alpha} P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right) = 1, \text{ that is}$$

$$\alpha = \sum_{i=0}^{M-1} P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right).$$

So, the symbol probability function would look like:

$$\frac{1}{\alpha} P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right) = \frac{P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right)}{\sum_{j=0}^{M-1} P\left(m_j\middle|\overline{m}_A\right)P\left(m_j\middle|\overline{m}_B\right)}$$

where

$$P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right)=\frac{e^{\left[\frac{I_A\cos\left(\frac{2\pi i}{M}\right)+Q_A\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{\sum_{j=0}^{M-1}e^{\left[\frac{I_A\cos\left(\frac{2\pi j}{M}\right)+Q_A\sin\left(\frac{2\pi j}{M}\right)}{\sigma^2}\right]}}\frac{e^{\left[\frac{I_B\cos\left(\frac{2\pi i}{M}\right)+Q_B\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{\sum_{k=0}^{M-1}e^{\left[\frac{I_B\cos\left(\frac{2\pi k}{M}\right)+Q_B\sin\left(\frac{2\pi k}{M}\right)}{\sigma^2}\right]}}$$

$$=\frac{e^{\left[\frac{I_A\cos\left(\frac{2\pi i}{M}\right)+Q_A\sin\left(\frac{2\pi i}{M}\right)+I_B\cos\left(\frac{2\pi i}{M}\right)+Q_B\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{\sum_{j=0}^{M-1}\sum_{k=0}^{M-1}e^{\left[\frac{I_A\cos\left(\frac{2\pi j}{M}\right)+Q_A\sin\left(\frac{2\pi j}{M}\right)+I_B\cos\left(\frac{2\pi k}{M}\right)+Q_B\sin\left(\frac{2\pi k}{M}\right)}{\sigma^2}\right]}}=\frac{e^{\left[\frac{(I_A+I_B)\cos\left(\frac{2\pi i}{M}\right)+(Q_A+Q_B)\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{F\left(M,\sigma,I_A,I_B,Q_A,Q_B\right)}.$$

The function F is introduced to simplify the notation.

Adding this to Equation A.3.1 leads to

$$\frac{P\left(m_i\middle|\overline{m}_A\right)P\left(m_i\middle|\overline{m}_B\right)}{\sum_{j=0}^{M-1}P\left(m_j\middle|\overline{m}_A\right)P\left(m_j\middle|\overline{m}_B\right)}=\frac{e^{\left[\frac{(I_A+I_B)\cos\left(\frac{2\pi i}{M}\right)+(Q_A+Q_B)\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{F\left(M,\sigma,I_A,I_B,Q_A,Q_B\right)\sum_{j=0}^{M-1}\frac{e^{\left[\frac{(I_A+I_B)\cos\left(\frac{2\pi j}{M}\right)+(Q_A+Q_B)\sin\left(\frac{2\pi j}{M}\right)}{\sigma^2}\right]}}{F\left(M,\sigma,I_A,I_B,Q_A,Q_B\right)}}=$$

$$\frac{e^{\left[\frac{(I_A+I_B)\cos\left(\frac{2\pi i}{M}\right)+(Q_A+Q_B)\sin\left(\frac{2\pi i}{M}\right)}{\sigma^2}\right]}}{\sum_{j=0}^{M-1}e^{\left[\frac{(I_A+I_B)\cos\left(\frac{2\pi j}{M}\right)+(Q_A+Q_B)\sin\left(\frac{2\pi j}{M}\right)}{\sigma^2}\right]}}=P\left(m_i\middle|\overline{m}_C\right)$$

when

$$\overline{m}_C=\begin{pmatrix}I_C\\Q_C\end{pmatrix}=\begin{pmatrix}I_A+I_B\\Q_A+Q_B\end{pmatrix}.$$

That is, adding together two vectors that are delivered from *M-PSK* signalling from an AWGN channel is equal to the Variable Node Operation in a Belief Propagation Decoder.

List of References

[1] Stephen B. Wicker. Error Control Systems (1995). Prentice Hall, Inc

[2] Simon Haykin (1988). Digital Communications. John Wiley & Sons.

[3] John B. Anderson. Digital Transmission Engineering (1998). IEEE Press.

[4] Raymond W. Young (2002). A First Course in Information Theory. Kluwer Academic / Plenum Publishers.

[5] Mikael Olofsson, Thomas Ericson, Robert Forchheimer and Ulf Henriksson (2003). Basic Telecommunication. University of Linköping.

[6] Richard O. Duda, Peter E. Hart and David G (2001). Stork. Pattern Classification. John Wiley & Sons, Inc.

[7] Lars Ahlin and Jens Zander (1998). Principles of Wireless Communications. Studentlitteratur Lund.

[8] Thomas J. Richardson and Rüdiger L. Urbanke. The Capacity of Low-Density Parity-Check Codes under Message Passing Decoding. IEEE Trans. Inform. Theory, vol. 47, NO. 2 Febr. 2001.

[9] Sae- Young Chung, Thomas J. Richardson and Rüdiger L. Urbanke. Analysis Of Sum- Product Decoding Of Low- Density Parity- Check Codes Using A Gaussian Approximation. IEEE Trans. Inform. Theory. Sept 19. 2000.

[10] Masoud Ardakani and Frank R. Kschischang. A More Accurate One-Dimensional Analysis and Design Of Irregular LDPC Codes. IEEE Trans. Inform. Theory, vol 52, NO. Dec. 2004.

[11] Masoud Ardakani . Efficient Analysis, Design and Decoding of Low-Density Parity- Check Codes. Doctor of Philosophy Thesis, The Edward S. Rogers Sr. Department, University of Toronto. Pp. 10-26, 56-92, 136-142. 2004.

[12] Ravi Narayanaswami. Coded Modulation with Low Density Parity Check Codes. Master of Science Thesis, Texas A&M University. pp. 25-38. June. 2001.

[13] John R. Barry. Low- Density Parity- Check Codes. Georgia Institute of Technology. Oct. 5. 2001.

[14] Thomas J. Richardson and Rüdiger L. Urbanke. The Renaissance Of Gallager's Low- Density Parity- Check Codes. IEEE Communications Magazine. Aug. 2003.

[15] Matthew C. Davey and David MacKay. Low- Density Parity Check Codes over GF(q). IEEE Communications Letters. Vol 2, NO 6, June 1998.

[16] Igor V. Kozintsev. Signal Processing, Software http://www.kozintsev.net/. Oct 31, 2005.

[17] David J. C. MacKay. Error Correcting Codes http://www.inference.phy.cam.ac.uk/ Oct 31, 2005.

[18] David J.C. MacKay. http://www.inference.phy.cam.ac.uk/mackay/codes/252.252.3.252. Oct 31, 2005

[19] Wang Lin, Xiao Juan and Guanro Chen. Density Evolution method and threshold decision for irregular LDPC codes. IEEE Communications, Circuits and Systems. Volume 1. 2004.