EXTENDING THE DORSCH DECODER FOR EFFICIENT SOFT DECISION DECODING OF LINEAR BLOCK CODES

By

SEAN MICHAEL COLLISON

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

May 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of Sean Michael Collison find it satisfactory and recommend that it be accepted.

Thomas R. Fischer, Ph.D., Chair

Martin Tomlinson, Ph.D.

Benjamin J. Belzer, Ph.D.

Jabulani Nyathi, Ph.D.

ACKNOWLEDGMENTS

I would like to express my appreciation to my advisor, Dr. Thomas Fischer, for his guidance and support during my master's degree. He has been a great advisor whom I am honored to have studied under. I thank Dr. Pat Owsley and Dr. Brian Banister for their support as well. Pat and Brian's support during my time at Comtech AHA Corporation was paramount to the completion of research for my thesis. Additionally, I would like to thank Dr. Martin Tomlinson for his support and guidance during my research, as well as for being one of my committee members.

I would also like to thank the School of Electrical Engineering and Computer Science for providing me with many opportunities and excellent classroom instruction. The Staff Assistanceship position received from Student Computing Services during this last year has also been much appreciated.

This work was supported, in part, by *Comtech AHA Corporation*.

No acknowledgment section would be complete without expressing my appreciation for my friends and fellow students. The numerous conversations that I have had with Peter Osheroff and Chris Keeser have challenged my assumptions and truly helped me progress as a student.

EXTENDING THE DORSCH DECODER FOR EFFICIENT SOFT DECISION DECODING OF LINEAR BLOCK CODES

Abstract

by Sean Michael Collison, M.S.
Washington State University
May 2009

Chair: Thomas R. Fischer

This thesis builds upon a method of decoding linear block codes presented by Dorsch in 1974. An algorithm is presented that details how to extend the Dorsch Decoding method to check error patterns in the most likely order that they occur. Furthermore, another method is described that increases decoding speed by using various stopping criteria to limit the depth of the error pattern search. Multiple codes of various rates are investigated to find the best performing code subject to a decoding complexity constraint.

Contents

List of Figures

List of Tables

Dedication

This thesis is dedicated to my parents; without their support and guidance I wouldn’t be where I am today.

Chapter 1

Introduction

1.1 Overview

The fundamentals of forward error correction and information theory were first presented by Claude Shannon in his landmark paper, 'A Mathematical Theory of Communication' [1], in 1948. In this paper, Shannon presented a theory that errors introduced by a noisy channel could effectively be reduced to any desirable level if the information rate of the channel was kept strictly less than the capacity of the channel. This paper effectively proved how well error correcting codes could perform on a given channel, however it gave no indication of how to design codes that achieve the performance. Much research has been done in the following years to find codes that perform as close as possible to the performance guaranteed by Shannon, commonly referred to as the Shannon limit.

A class of error correcting codes, some with performance close to the Shannon limit, is linear block codes [2]. Linear block codes are commonly referred to as (n,k,d_{min}) codes, where n represents the total block length (in bits) of a codeword in the code, k represents the total number of information bits in a codeword, and d_{min} is a parameter, specified only if known, representing the minimum Hamming

distance (the number of different bits) between two codewords.

A simple method of decoding a linear block code is to use only the hard decision value of a bit in a codeword, either a 0, or a 1, discarding all extra information about the confidence of the received bit that can be obtained from an analog sample of a matched filter receiver. Discarding the confidence of a received bit allows for simple and fast decoding of a particular code, however it results in suboptimal performance compared to decoding the same code using a soft-decision decoder, ie., one that uses the confidence of a received bit in a codeword to help in the decoding process. According to [3], a soft-decision decoder can have approximately 3 dB of coding gain when compared to a hard decision decoder for the same code. A method of soft-decision decoding of linear block codes is presented in this thesis which can be used to decode any linear block code.

1.2 Previous Work

Much work has been done on the soft-decision decoding of linear block codes. The method this thesis builds upon is a reliability-based decoding scheme which processes the most reliable positions (MRPs) of a received codeword.

Codewords in a (n, k) binary linear block code are represented as $\underline{c}$ in equation (1.1) with each c_i taking on the value of either a 0 or 1.

$$\underline{c} = (c_1, c_2, \ldots, c_n) \tag{1.1}$$

When $\underline{c}$ is transmitted over an AWGN (Additive White Gaussian Noise) channel using binary antipodal signaling, each 0 is mapped to a -1 and each 1 is mapped

to a $+1$, yielding a modified codeword vector, $\underline{x}$, shown in equation (1.2).

$$\underline{x} = (x_1, x_2, \ldots, x_n) \tag{1.2}$$

The AWGN channel contributes noise to the codeword, represented by the vector $\underline{z}$. The discrete channel noise samples are modeled as Gaussian, zero-mean, independent, and identically distributed with variance ${\sigma_z}^2$. A received vector transmitted over the channel is matched filtered and represented as $\underline{r}$ in equation (1.3).

$$\underline{r} = \underline{x} + \underline{z} \tag{1.3}$$

A reliability-based decoding scheme uses the soft-decision received sequence, $\underline{r}$, from equation (1.3), in addition to a sequence, $\underline{c}'$, decoded by the hard decision rule presented in (1.4).

$$c_i' = \begin{cases} 0, & \text{when } r_i < 0 \\ 1, & \text{when } r_i \geq 0 \end{cases}, i = 1, \ldots, n. \tag{1.4}$$

A soft-decision decoding algorithm called a most reliable independent position (MRIP) decoding or MRIP-reprocessing decoding algorithm is presented in [3]. The MRIP-reprocessing algorithm requires a set of k independent most reliable bit positions (MRP) in $\underline{r}$. This number of bits is the minimum needed to uniquely determine a codeword in a (n, k) linear code. The most reliable bit positions are chosen on the basis that they have the least probability of being incorrect when using hard decision decoding. The received vector of equation (1.3) has components $x_i \in \{1, -1\}$. With the hard decision threshold set at zero, the

probability of a bit error in the hard decision vector component c_i conditioned on $\alpha_2 \geq |r_i| \geq \alpha_1$ is shown in equation (1.5) and the expression for $p(r|x_i)$ is given in equation (1.6).

$$P(\text{error in } x_i \mid \alpha_2 \geq |r_i| \geq \alpha_1) = \frac{\int_{\alpha_1}^{\alpha_2} p(r|x_i = -1)\, dr}{\int_{\alpha_1}^{\alpha_2} p(r|x_i = -1)\, dr + \int_{\alpha_1}^{\alpha_2} p(r|x_i = 1)\, dr} \tag{1.5}$$

$$p(r|x_i) = \frac{1}{\sqrt{2\pi}\sigma_z} exp\left(\frac{-(r - x_i)^2}{2{\sigma_z}^2}\right) \tag{1.6}$$

Examining the limiting case where $\alpha_2 - \alpha_1 \to 0$, the upper and lower bounds of the integral approach r_i and the expression in equation (1.5) simplifies to equation (1.7).

$$P(\text{error in } x_i \mid |r_i|) = \frac{exp\left(\frac{-(|r_i|+1)^2}{{\sigma_z}^2}\right)}{exp\left(\frac{-(r_i+1)^2}{{\sigma_z}^2}\right) + exp\left(\frac{-(r_i-1)^2}{{\sigma_z}^2}\right)} \tag{1.7}$$

Taking the derivative of equation (1.7) it is observed that it is monotonically decreasing for increasing $|r_i|$. Thus the k most reliable hard-decision decoded bits are those with the largest magnitude $|r_i|$.

In the MRIP-reprocessing algorithm summarized as follows, E will be a set of low-weight error patterns. The algorithm is:

MRIP-reprocessing Algorithm

1. Determine the error pattern set E based on the k MRIPs of $\underline{r}$.

2. For each error pattern $\underline{e}$ in E, encode $\underline{c}' + \underline{e}$ into a codeword in C, forming a list of candidate codewords to be evaluated.

3. For each codeword in the list from step 2, compute the soft-decision de-

coding metrics of these candidate codewords. Choose the one with the best metric (lowest Euclidean distance or highest correlation) as the decoded codeword.

1.2.1 Distance and Correlation Decoding Metrics

As presented in [3], two decoding metrics, a squared Euclidean distance calculation or correlation, can be used to find the closest candidate codeword $\underline{c}$ to a received vector $\underline{r}$. The squared Euclidean distance calculation chooses a candidate codeword $\underline{c} \in C$ such that it minimizes the squared distance between the received vector and all candidate codewords as seen in equation (1.8). Note that for each $c_i \in \underline{c}$, the value of $c_i \in \{-1, 1\}$.

$$\underline{c} = \underset{\underline{c} \in C}{\operatorname{argmin}} \sum_{i=1}^{n} (r_i - c_i)^2 \tag{1.8}$$

The squared distance computation in equation (1.8) can be expanded to the form observed in equation (1.9). From this equation, the ${r_i}^2$ component is constant across all $\underline{c} \in C$, and ${c_i}^2$ is also a constant.

$$\underline{c} = \underset{\underline{c} \in C}{\operatorname{argmin}} \sum_{i=1}^{n} {r_i}^2 - 2 r_i c_i + {c_i}^2 \tag{1.9}$$

When minimizing the squared distance calculation, it is observed that the $-2 r_i c_i$ term is the only term which affects the minimization. The -2 scaling factor can be removed from the computation, which switches the minimization problem to a maximization and an equivalent relation to minimizing the squared distance

emerges. This new relation realized in equation (1.10) is a correlation.

$$\underline{c} = \underset{\underline{c} \in C}{\operatorname{argmax}} \sum_{i=1}^{n} r_i c_i \tag{1.10}$$

Both the squared distance calculation along with the correlation are equivalent measures for determining if a prospective codeword is closer than another to a received vector. The correlation is computationally simpler, only requiring n multiplications and $n-1$ additions for each codeword tested, whereas the squared distance calculation requires n multiplications and $2n-1$ additions.

1.2.2 Rearranging the Parity Check Matrix for MRIP Processing

As presented previously, the MRIP-reprocessing algorithm generates prospective error patterns based upon the k most reliable independent positions. The remaining $n-k$ parity bits of a prospective codeword or error pattern can be generated using a modified parity check matrix, $\mathbf{H}'$. The process of manipulating the parity check matrix to generate $\mathbf{H}'$ presented in [4], is described below. A slightly different method is presented in [3].

Assume a parity check matrix, **H**, is given for an (n,k) code, in the standard form of equation (1.11).

$$\mathbf{H} = [\mathbf{P}\mathbf{I}_{n-k}] = \begin{bmatrix} p_{1,1} & \cdots & p_{1,k} & 1 & 0 & \cdots & 0 \\ p_{2,1} & \cdots & p_{2,k} & 0 & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \ddots & \vdots \\ p_{n-k,1} & \cdots & p_{n-k,k} & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{1.11}$$

Each noisy received vector, $\underline{r}$, is sorted by the magnitude of its positions to cre-

ate a new sorted vector $\underline{r}'$. The new vector $\underline{r}'$ is of the form presented in equation (1.12), where the sorting operation is defined as a reordering permutation π.

$$\underline{r}' = \{\pi[\underline{r}] : |r'_1| \geq |r'_2| \geq ... \geq |r'_n|\} \tag{1.12}$$

The reordering permutation π is then subsequently applied to the columns of the parity-check matrix, $\mathbf{H}$, to create an intermediary parity check matrix $\mathbf{H}_{int}$, as in equation (1.13).

$$\mathbf{H}_{int} = \pi[\mathbf{H}] \tag{1.13}$$

The intermediary parity-check matrix is then modified to a standard or row-reduced echelon form of $\mathbf{H}' = [\mathbf{P}'\mathbf{I}_{n-k}]$. Row reducing the intermediary matrix yields an updated parity-check matrix for the sorted codeword presented in equation (1.14), where rref denotes row-reduced echelon form.

$$\mathbf{H}' = [\mathbf{P}'\mathbf{I}_{n-k}] = \text{rref}(\mathbf{H}_{int}) = \text{rref}(\pi[\mathbf{H}]) \tag{1.14}$$

With the updated parity-check matrix sharing the form of the parity-check matrix in equation (1.11), a new parity matrix $\mathbf{P}'$ can be extracted. The transpose of $\mathbf{P}'$, when multiplied by the k user settable data bits of a prospective codeword, will generate the remaining parity bits of codeword, as seen in equation (1.15).

$$[p_1, p_2, ..., p_{n-k}] = [c'_1 + e_1, c'_2 + e_2, ..., c'_k + e_k] \cdot \mathbf{P}'^T \tag{1.15}$$

1.3 Outline of Thesis

Chapter 2 covers various algorithms that are used to generate sets of error patterns to test. This chapter covers the order-i reprocessing algorithm as well as a new algorithm that introduces error patterns in their most probable order. The performance of the order-i reprocessing algorithm is compared to the most likely error pattern generation algorithm for decoding a (136,68,24) linear block code.

Chapter 3 covers a variety of algorithms that can be used to improve decoder efficiency by reducing the average number of error patterns that need to be tested per codeword, while achieving similar decoding performance as if the total number of error patterns were tested. One early termination metric based on the correlation of parity bits is presented, along with a refined method allowing significantly better termination with no loss in decoding performance. A second method is presented that allows termination of error pattern generation based upon a squared distance calculation between the received vector and candidate codeword being less than a set threshold. Lastly, the squared distance termination metric is improved through a method of bounding overly confident bit positions in a received vector with the same hard decision value of the prospective codeword when the squared distance calculation is made. Both squared distance termination metrics are lossy in terms of decoding performance. However, by using a fixed squared radius dependant upon the code, the performance of the decoder can be maintained quite close to the case when all error patterns are tested.

Chapter 4 considers the performance limits of the decoder, subject to a complexity constraint on the maximum number of error patterns tested per codeword. Three code rates, 0.5, 0.75, and 0.9, are explored with codes of multiple block

sizes, and a maximum of 300,000 error patterns tested per codeword. The best performing code in terms of bit error rate and block error rate for a given code rate is tested again with a higher limit on the number of error patterns per codeword (3,000,000) and also compared to the performance of LDPC codes of a larger block size generated at Comtech AHA Corporation.

1.4 Summary of Contribution

- Design of an algorithm that generates candidate error patterns in the most likely order that they would occur.
- Design of a stopping criterion based upon a Euclidean squared distance threshold between a prospective codeword and received vector.
- Design of an enhanced stopping criterion based upon Euclidean squared distance and bounding of overly confident bits in the codeword.
- Study of codes of various rates and block sizes, to find the best performing code at a given block size given a decoding complexity constraint.

Chapter 2

Candidate Error Pattern Generation

In this chapter various error pattern set generation algorithms are explored for most reliable independent position (MRIP) decoding. These algorithms include a family of ordered statistic algorithms, in addition to a new algorithm that generates error patterns in the probabilistic order in which they are likely to occur. The performance of these algorithms is then evaluated for decoding a (136,68,24) linear block code.

2.1 Quantization

Prior to introducing the error pattern generation algorithms, a quantization rule Q needs to be defined. Using quantized values instead of real-numbered values simplifies error pattern generation, in addition to reducing the complexity required for correlation and distance calculations. The quantization rule Q is used to map the received vector $\underline{r}$ in equation (1.3), where each r_i is real-valued, to a new vector $\underline{rq}$ where each rq_i is an integer value.

The quantization rule Q describes a uniform mid-tread quantizer that allows possible reconstruction levels to range from $-2^{L-1}+1$ to $2^{L-1}-1$, where L is the total number of bits used to represent the quantized value. Equation (2.1) de-

scribes the mid-tread quantizer. In this equation, r_i is a real-valued received vector component of $\underline{r}$, rq_i is the output of the quantizer indicating the quantization index that r_i is mapped to, and one_index is a scalar used to scale $\underline{r}$ so that the index corresponding to $r_i = 1$ is in the positive output range of the quantizer. The value of one_index can range from 1 to $2^{L-1} - 1$.

$$rq_i = Q(r_i) = \begin{cases} \text{sign}(r_i)(2^{L-1} - 1), & \text{when } |\frac{r_i \cdot 2 \cdot one\_index}{2^L - 1}| \geq 1 \\ \lfloor \frac{(2^L)(one\_index)r_i}{2^L - 1} + 0.5 \rfloor, & \text{when } |\frac{r_i \cdot 2 \cdot one\_index}{2^L - 1}| < 1 \end{cases} \tag{2.1}$$

In the following sections, examples of error pattern generation are given with respect to a received vector for a (7,4) Hamming code. The real-valued received vector, $\underline{r}$, for these examples is given in equation (2.2).

$$\underline{r} = (-0.2, -1.5, 0.2, 0.6, 0.3, 0.7, -0.7) \tag{2.2}$$

When the reordering permutation presented in equation (1.12) is applied to this vector, the vector $\underline{r}'$ is produced as seen in equation (2.3).

$$\underline{r}' = \pi[\underline{r}] = (-1.5, 0.7, -0.7, 0.6, 0.3, 0.2, -0.2) \tag{2.3}$$

The quantization rule Q is applied to $\underline{r}'$ to produce a quantized, sorted received vector $\underline{rq}$ shown in equation (2.4). In this example, the quantization process uses the scalar value $one\_index = 10$ and $L = 5$.

$$\underline{rq} = Q(\underline{r}') = (-15, 7, -7, 6, 3, 2, -2) \tag{2.4}$$

2.2 Ordered Statistic Error Pattern Generation

Ordered statistic error pattern generation is quite common and presented in numerous texts and papers, e.g., [3], [5]. This method of introducing error patterns consists of a multitude of reprocessing steps, with the total number denoted as i. Generating error patterns with i reprocessing steps is commonly referred to as order-i error pattern generation.

Order-i reprocessing operates on the k most reliable independent positions of a received codeword for a (n, k) linear block code. An outline of the algorithm is presented, along with an example in the following section.

Order-i Reprocessing Algorithm

1. **Initialization:**
   - Set error pattern $e_j = 0$ for $j = 1, \ldots, n$.
   - Set counter $l = 0$.
   - Obtain sorted, quantized received vector $\underline{rq}$ from $\underline{r}$ as detailed in section (2.1).
   - Obtain updated H-matrix, $\mathbf{H}'$, from $\mathbf{H}$ as detailed in section (1.2.2).
2. **Error Pattern Generation:**
   - while $l \leq i$
     - Generate all possible error patterns with a Hamming weight of l. For a (n, k) linear block code, this will correspond to generating $\binom{k}{l}$ error patterns. Error patterns at this stage are of the form

$\underline{e} = (e_1, e_2, \ldots, e_k)$ and do not include parity bits. These error patterns are assigned to a set E.

- Generate a candidate codeword set, C, from the $\binom{k}{l}$ error patterns in E, as $\underline{c}_{1,\ldots,\binom{k}{l}} = \underline{e}_{1,\ldots,\binom{k}{l}} + \pi(\underline{c}')$. Where $\underline{c}'$ is the hard decision received vector from equation (1.4) and $\pi()$ is the reordering permutation defined in equation (1.12). Note that only the first k bits are used after the reordering permutation operates on the hard-decision received vector.
- Generate the remaining $n - k$ parity bits of the candidate codewords. For each $\underline{c} \in C$, $(c_{k+1}, \ldots, c_n) = (c_1, \ldots, c_k) \cdot \mathbf{P}^T$, where $\mathbf{P}^T$ is the transpose of the parity section of the permuted H-matrix described in equation (1.14).
- Test each candidate codeword $\underline{c} \in C$ to determine if it is closer to the quantized received vector $\underline{rq}$. An updated squared distance calculation is presented in equation (2.5). This updated equation scales the candidate codeword by one_index to preserve the scaled distance computation. The original correlation method of equation (1.10) is also valid as a decoding method, only needing to be modified to use the quantized received vector $\underline{rq}$, as shown in equation (2.6).
- Record the closest candidate codeword $\underline{c} \in C$ to the received vector $\underline{rq}$ and compare its decoding metric with the best codewords chosen in previous reprocessing steps. If the codeword $\underline{c}$ is the best, record it as the best codeword thus far.
- Increment l, $l = l + 1$.

$$\underline{c} = \underset{\underline{c} \in C}{\operatorname{argmin}} \sum_{i=1}^{n} (r_i - one\_index \cdot c_i)^2 \tag{2.5}$$

$$\underline{c} = \underset{\underline{c} \in C}{\operatorname{argmin}} \sum_{i=1}^{n} rq_i c_i \tag{2.6}$$

From examining the algorithm, it is evident that if order-i processing is used to decode a received codeword, the total number of error patterns generated will be:

$$total\_error\_patterns = \sum_{j=0}^{i} \binom{k}{j}.$$

2.2.1 Order-i Error Pattern Generation Example

Generating error patterns for order-i reprocessing is quite trivial. Error pattern generation equates to "walking" a number of '1's through all possible combinations in the first k positions, with the number of '1's dependent upon the current reprocessing level. The received codeword, $\underline{r}$, used in the following example is the one presented in equation (2.2). After quantization and being passed through the reordering permutation, the quantized received vector becomes that of equation (2.4), which is presented again in equation (2.7).

$$\underline{rq} = (-15, 7, -7, 6, 3, 2, -2) \tag{2.7}$$

The following example details how to generate error patterns for this received vector using order-i reprocessing. In this particular example, all 2^4 possible error patterns are generated, which equates to performing order-4 reprocessing. Once an error pattern $\underline{e}$ is created, the first 4 bits are added to the first 4 bits of the

reordered hard decision received vector, $\underline{c}'$ (detailed in equation (2.8), forming a candidate codeword $\underline{c\_codeword}'$.

$$\underline{c}' = (0, 1, 0, 1, 1, 1, 0) \tag{2.8}$$

The remaining $n - k$ parity bits of the candidate codeword $\underline{c\_codeword}$ are generated through a matrix multiplication of the transpose of the parity section of the rearranged H-matrix detailed in section 1.2.2. Distance or correlation decoding metrics can be computed as detailed in equations (2.5) and (2.6).

- **Order-0 Reprocessing** Order-0 reprocessing introduces the all 0's error pattern and computes the decoding metric describing the distance between the quantized received vector and the corresponding hard decision codeword generated by the first k positions of the hard decision received vector.

$$\underline{e} = (0, 0, 0, 0, 0, 0, 0)$$

- **Order-1 Reprocessing** Order-1 reprocessing introduces error patterns of weight 1 to test against the quantized received vector. These error patterns are:

$$\underline{e} = (1, 0, 0, 0, p_1, p_2, p_3)$$

$$\underline{e} = (0, 1, 0, 0, p_1, p_2, p_3)$$

$$\underline{e} = (0, 0, 1, 0, p_1, p_2, p_3)$$

$$\underline{e} = (0, 0, 0, 1, p_1, p_2, p_3)$$

- **Order-2 Reprocessing** Order-2 reprocessing introduces error patterns of weight 2 to test against the quantized received vector. The error patterns that are introduced are:

$$\underline{e} = (1, 1, 0, 0, p_1, p_2, p_3)$$

$$\underline{e} = (1, 0, 1, 0, p_1, p_2, p_3)$$

$$\underline{e} = (1, 0, 0, 1, p_1, p_2, p_3)$$

$$\underline{e} = (0, 1, 1, 0, p_1, p_2, p_3)$$

$$\underline{e} = (0, 1, 0, 1, p_1, p_2, p_3)$$

$$\underline{e} = (0, 0, 1, 1, p_1, p_2, p_3)$$

- **Order-3 Reprocessing** Order-3 reprocessing introduces error patterns of weight 3 to test against the quantized received vector. The error patterns that are introduced are:

$$\underline{e} = (1, 1, 1, 0, p_1, p_2, p_3)$$

$$\underline{e} = (1, 1, 0, 1, p_1, p_2, p_3)$$

$$\underline{e} = (1, 0, 1, 1, p_1, p_2, p_3)$$

$$\underline{e} = (0, 1, 1, 1, p_1, p_2, p_3)$$

- **Order-4 Reprocessing** Order-4 reprocessing is the last possible order of error patterns that can be introduced, as it employs all of the first k most

likely independent bit positions. The last error pattern is of weight 4:

$$\underline{e} = (1, 1, 1, 1, p_1, p_2, p_3)$$

2.3 Most Likely Error Pattern Generation

The most likely error pattern generation algorithm proposed in this section generates error patterns in the most probable order in which they are estimated to occur based upon the magnitude of the first k most reliable, independent quantized bits. This method is different from order-i reprocessing algorithms, such as the one presented in section 2.2, in that with the most likely error pattern generation algorithm, an error pattern with a higher order (a larger number of 1's in the most reliable independent k positions of the error pattern) could be tried before all error patterns of a lower order have been exhausted. With this algorithm, trying only the most likely error patterns improves the probability of correctly finding the sent codeword when a fixed number of error patterns are generated and tested.

The most likely error pattern generation algorithm can be generalized to the following steps which can apply to any linear block code size and rate. Equation (2.9) represents a received vector of a (n, k) linear block code, while equation (2.10) represents the same vector that has been sorted and quantized as detailed in section 2.1.

$$\underline{r} = (r_1, r_2, \ldots, r_n) \tag{2.9}$$

$$\underline{rq} = (rq_1, rq_2, \ldots, rq_n) \tag{2.10}$$

An error pattern is introduced in equation (2.11). Similar to the error patterns presented in the previous section on order-i reprocessing, the error pattern only contains k user defined bits, with the remaining $n-k$ parity bits generated after the error pattern has been added to the first k bits of the permuted hard decision received vector.

$$\underline{e} = (e_1, e_2, ..., e_k, p_1, p_2, ..., p_{n-k}) \tag{2.11}$$

Introducing error patterns in the most likely order in which they are expected to occur requires computing likelihood ratios for the first $i = 1, 2, \ldots, k$ bits of the quantized received vector $\underline{rq}$. The likelihood ratio compares the probability of a hard decision error at the quantization level, rq_i with reproduction level $\frac{rq_i}{one\_index}$, to the probability that the hard decision at that quantization level was correct. The likelihood ratio is presented in equation (2.12), with $p(\frac{|rq_i|}{one\_index}|x_i)$ defined in equation (2.13).

$$lr_i = \frac{p(\frac{|rq_i|}{one\_index}|x=-1)}{p(\frac{|rq_i|}{one\_index}|x=1)} \tag{2.12}$$

$$p\left(\frac{|rq_i|}{one\_index}|x_i\right) = \frac{1}{\sqrt{2\pi}\sigma_z} exp\left(\frac{-(\frac{|rq_i|}{one\_index} - x_i)^2}{2{\sigma_z}^2}\right) \tag{2.13}$$

Taking the natural logarithm of lr_i in equation (2.12), creates an updated ratio called the log likelihood ratio (llr). The log likelihood ratio is presented in equation (2.14).

$$llr_i = \ln(lr_i) = \ln\left(\frac{p(\frac{|rq_i|}{one\_index}|x=-1)}{p(\frac{|rq_i|}{one\_index}|x=1)}\right) = \frac{-2\,|rq_i|}{{\sigma_z}^2 \cdot one\_index} \tag{2.14}$$

From equation (2.14), it is observed that the log likelihood ratio is directly proportional to the magnitude of the quantized received value $|rq_i|$, with a scaling factor

of $\frac{-2}{{\sigma_z}^2 \cdot one\_index}$.

Introducing error patterns with a decreasing probability of occurrence corresponds to introducing error patterns with a decreasing llr, since both the probability of error and llr are monotonically decreasing for increasing values of $|rq_i|$. When an error pattern is chosen, the probability of its occurrence is the product of the probabilities of each individual bit in the error pattern being in error as seen in equation (2.15).

$$P(\underline{e}) = \prod_{i=1}^{k} \begin{cases} P(\text{error in } rq_i), \text{ when } e_i = 1 \\ 1, \text{ when } e_1 = 0 \end{cases} \tag{2.15}$$

In terms of the llr, the corresponding likelihood of an error pattern occurring is the sum of the llr's of each quantized received value involved in the error pattern, seen in equation (2.16). The likelihood value is inversely proportional to the probability of that error pattern occurring. Thus if one error pattern has a likelihood value less than another, the probability of that error pattern occurring is greater than that of the other.

$$P(\underline{e}) = \sum_{i=1}^{k} e_i \cdot llr_i \tag{2.16}$$

An observation can be made that each error pattern likelihood generated in equation (2.16) is a scaled version of the sum of the magnitudes, $|rq_i|$, of bits in the error pattern. Since this scaling factor is constant across all error patterns tested at a fixed noise level, the magnitude of the received value $|rq_i|$ can be substituted for llr_i in the likelihood computation, yielding an updated version of equation (2.16),

in equation (2.17) where $P(\underline{e})$ is replaced with a new measure $cost$.

$$cost = \sum_{i=1}^{k} e_i \cdot |rq_i| \tag{2.17}$$

In this equation, $0 \leq cost \leq \sum_{i=1}^{k} |rq_i|$, and the lower the cost, the greater the probability that the error pattern occurred. Since $|rq_i|$ is an integer, ranging from 0 to $2^{L-1} - 1$, the value of cost will always be an integer. Most likely error pattern generation operates by generating error patterns of the lowest possible cost, 0, then incrementing the cost by 1 and generating all possible error patterns that satisfy that cost before the cost is incremented further to generate less probable error patterns. In terms of probability, two fundamental equations (2.18) and (2.19) arise, laying the groundwork for this method of decoding. Equation (2.18) states that the probabilities of two error patterns are the same if and only if their costs computed in equation (2.17) are the same. Additionally, if the cost of one error pattern is greater than another, then the probability of that error pattern occurring is less than that of the other error pattern as detailed in equation (2.19).

$$p(\underline{e}_1) = p(\underline{e}_2) \iff cost(\underline{e}_1) = cost(\underline{e}_2) \tag{2.18}$$

$$p(\underline{e}_1) > p(\underline{e}_2) \iff cost(\underline{e}_1) < cost(\underline{e}_2) \tag{2.19}$$

Most Likely Error Pattern Generation Algorithm

1. **Initialization:**

   - Set $cost = 0$.
   - Set $e_i = 0$ for $i = 1, \ldots, n$.

- Obtain sorted, quantized received vector $\underline{rq}$ from $\underline{r}$ as detailed in section (2.1).
- Obtain updated H-matrix $\mathbf{H}'$ from $\mathbf{H}$ as detailed in section (1.2.2).
- Set $ep\_count = 0$.

2. **Error Pattern Generation:**

- while $e_i \neq 1$ for $i = 1, \ldots, k$
  - while a new $\underline{e}$ can be generated at $cost$ and $ep\_count < max\_ep\_count$
    * Generate error pattern $\underline{e}$ that satisfies equation 2.17, ($cost = \sum_{i=1}^{k} e_i \cdot rq_i$).
    * Increment ep_count, $ep\_count = ep\_count + 1$.
    * Generate first k bits of candidate codeword $\underline{c}$. This can be done in the following equation $\underline{c} = \underline{e} + \pi(\underline{c}')$, where $\underline{c}'$ is the hard decision received vector from equation (1.4) and $\pi()$ is the reordering permutation defined in equation (1.12). Note that only the first k bits are kept after the reordering permutation operates on the hard decision received vector.
    * Generate remaining parity bits of candidate codeword. $(c_{k+1}, ..., c_n) = (c_1, ..., c_k) \cdot \mathbf{P}^T$, where $\mathbf{P}^T$ is the transpose of the parity section of the permuted H-matrix described in equation (1.14).
    * Test the candidate codeword $\underline{c}$ to determine if it is the closer to the quantized received vector $\underline{rq}$. An updated squared distance calculation is presented in equation (2.5). This updated equation scales the candidate codeword by one_index to pre-

serve the scaled distance computation. The original correlation method of equation (1.10) is also valid as a decoding method, only needing to be modified to use the quantized received vector $\underline{rq}$ as seen in equation (2.6).

- Increment cost level, $cost = cost + 1$.

2.3.1 Most Likely Error Pattern Generation Example

This section, similar to section 2.2.1, provides an example of error pattern generation using the most likely error pattern generation algorithm. The following example will use the same received vector as that of section 2.2.1, $\underline{r}$, of equation (2.2). Prior to introducing error patterns, the received vector $\underline{r}$ is required to be sorted and quantized as detailed in section 2.1 to obtain a transformed vector, $\underline{rq}$, presented in equation (2.4), and again in equation (2.20).

$$\underline{rq} = (-15, 7, -7, 6, 3, 2, -2) \tag{2.20}$$

From the quantized received vector, $\underline{rq}$, of equation (2.20), only the first k, (4) most reliable positions are used in error pattern generation. With the first k positions defined, the next step is to create a mapping as detailed in Table 2.1. This table contains a count of how many of the first k elements are present at various quantization levels. This table will aid later when checking if error patterns can be formed from the quantized values present.

As presented in 2.3, error patterns will be introduced according to equation (2.17) with $cost$ starting at 0. Only after all possible error patters of the current cost have been generated, then can the cost be incremented to generate less prob-

Table 2.1: Quantization Levels and Items at that Level

Quantization Level	**Level Count**
15	1
14	0
13	0
12	0
10	0
9	0
8	0
7	2
6	1
5	0
4	0
3	0
2	0
1	0

able error patterns. In the following example, all possible 2^4 error patterns are generated. Once an error pattern $\underline{e}$ is created, the first 4 bits are added to the first 4 bits of the reordered hard decision received vector, $\underline{c}'$ (detailed in equation (2.21), forming a candidate codeword $\underline{c\_codeword}'$.

$$\underline{c}' = (0, 1, 0, 1, 1, 1, 0) \tag{2.21}$$

The remaining $n - k$ parity bits of the candidate codeword $\underline{c\_codeword}$ are generated through a matrix multiplication of the transpose of the parity section of the rearranged H-matrix detailed in section 1.2.2. Distance or correlation decoding metrics can be computed as detailed in equations (2.5) and (2.6).

- The error pattern with a cost of 0 is generated and tested. (This is the initial correlation / distance computation).
- Since no quantized received positions can have a cost of 1 to 5, the cost is incremented to 6. At this level, positions that have a quantization level of

6 are employed in creating an error pattern. This would yield a single error pattern shown below. No other error patterns can be formed at this cost, since no other received positions in the first k bits exist at lower quantization levels.

$$[0, 0, 0, 1, p_1, p_2, p_3]$$

In this error pattern, p_1, p_2, and, p_3 are left blank and computed as described previously from the transpose of the parity section of the transformed parity check matrix.

- Incrementing the cost to 7, there are 2 received positions at that quantization level which can be employed in creating error patterns. These positions yield the following 2 error patterns. Note that if one received position at a quantization level of 1 were present, it could be combined with the one position at a quantization level of 6 to create an error pattern equivalent to the other two in terms of its probability of occurring.

$$[0, 0, 1, 0, p_1, p_2, p_3]$$

$$[0, 1, 0, 0, p_1, p_2, p_3]$$

- No possible error patterns exist between a cost of 7 and 12. However at a level of 13, 2 error patterns can be formed by using 1 of the two positions at a quantization level 7 combined with the position at a quantization level of 6. The error patterns that are produced can be observed below.

$$[0, 1, 0, 1, p_1, p_2, p_3]$$

$$[0, 0, 1, 1, p_1, p_2, p_3]$$

- Incrementing the cost by 1 to 14, a single error pattern can be formed at this level by employing the two received positions at a quantization level of 7. The following error pattern is produced.

$$[0, 1, 1, 0, p_1, p_2, p_3]]$$

- At a cost of 15, the last received position with a quantization limit of 15 can be used in creating an error pattern. This creates the following error pattern.

$$[1, 0, 0, 0, p_1, p_2, p_3]$$

- The next cost which contains valid error pattern is 20. At this level, 2 positions at a quantization level of 7, and 1 at a level of 6 are used in forming the error pattern.

$$[0, 1, 1, 1, p_1, p_2, p_3]$$

- A cost of 21 permits the generation of a single error pattern, which uses positions at a quantization level of 15 and 6 producing the following error pattern.

$$[1, 0, 0, 1, p_1, p_2, p_3]$$

- Incrementing the cost to 22, allows for the creation of 2 error patterns employing the one received position at a quantization level of 15, and individually each of the received positions at a quantization level of 7. This yields

the following error patterns.

$$[1, 1, 0, 0, p_1, p_2, p_3]$$

$$[1, 0, 1, 0, p_1, p_2, p_3]$$

- The next 2 possible error patterns can be formed at a cost of 28. At this level, the received position with a quantization level of 15 is utilized along with one of the possible 2 positions at a quantization level of 7, and the last position with a quantization level of 6.

$$[1, 1, 0, 1, p_1, p_2, p_3]$$

$$[1, 0, 1, 1, p_1, p_2, p_3]$$

- The last possible error pattern can be formed by employing all possible positions with a cost of 35.

$$[1, 1, 1, 1, p_1, p_2, p_3]$$

2.4 Performance of Various Algorithms

Performance of the order-i and most likely error pattern generation algorithms can now be compared. Performance of both algorithms were evaluated using a (136,68,24) double circulant linear block code obtained from Martin Tomlinson, Professor in Fixed and Mobile Communications at the University of Plymouth, UK [10]. This is the same code used in [7]. In all cases, the number of error pat-

terns tried were the same for each generation method, and the number of error patterns tried remained on an even boundary for order-i processing of the (136,68,24) code. Points on each curve are given after observing an average of 50-block error events at the particular noise level on an AWGN channel. All plots include Shannon's sphere packing bound [6], which is the best block error rate performance a code of a fixed block size can have. Software to calculate this bound was obtained from Martin Tomlinson [10].

In the first simulation, order-2 decoding was done with the order-i decoder and the same number of error patterns,

$$2,346 = \binom{68}{1} + \binom{68}{2}$$

was the maximum number of error patterns allowed to be generated by the most likely error pattern generation algorithm. Simulation results can be observed in Figure 2.1. From the results it is observed that there is a slight performance gain (approximately 0.1 dB using the most likely error pattern generation algorithm at low SNR and high block error rates ($> 10^{-3}$). After this point, the set of error patterns that are introduced by the order-i reprocessing algorithm are the same as the ones introduced by the most likely error pattern generation algorithm.

The second simulation was done using order-3 decoding. Paralleling the first simulation, only

$$52,462 = \binom{64}{1} + \binom{64}{2} + \binom{64}{3}$$

error patterns were allowed to be generated by the most likely error pattern generation algorithm. Observing the results of the simulation in Figure 2.2, there is a also a slight performance gain from the most likely error pattern generation al-

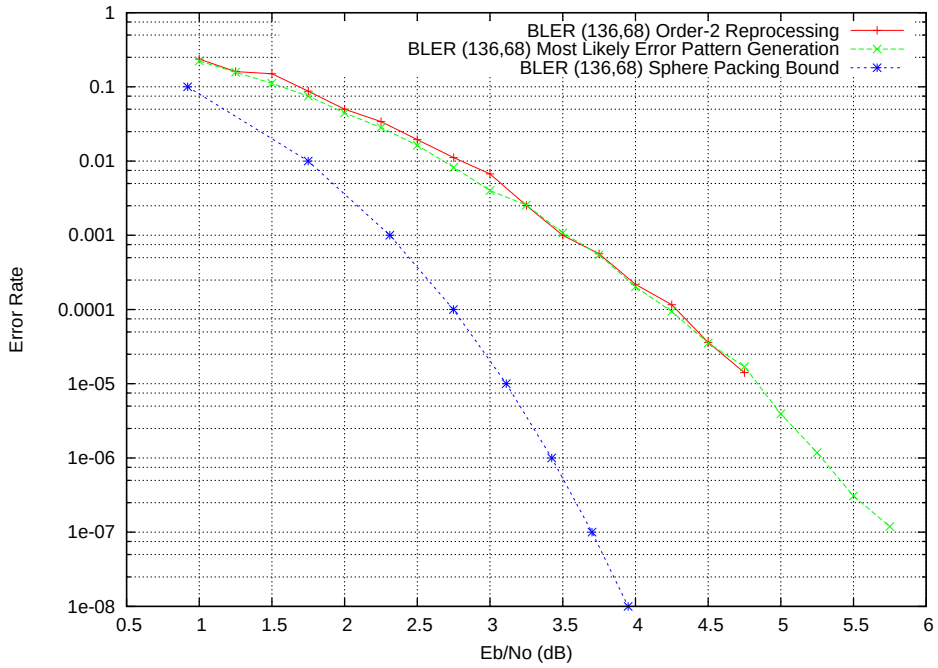

Figure 2.1: Comparison of Order-2 reprocessing and Most Likely Error Pattern Generation for (136,68,24) Code

gorithm of approximately 0.07 dB. This gain is observed to a block error rate of 10^{-5} and above.

The third and final simulation was done using order-4 decoding. This simulation, similar to the first two, only allowed

$$866,847 = \binom{64}{1} + \binom{64}{2} + \binom{64}{3} + \binom{64}{4}$$

error patterns to be introduced by the most likely error pattern generation algorithm. In Figure 2.3, it is observed that there is no appreciable gain using the most likely error pattern generation algorithm compared to the order-i reprocessing algorithm. This is due to the fact that many of the error patterns generated to be tested are the same.

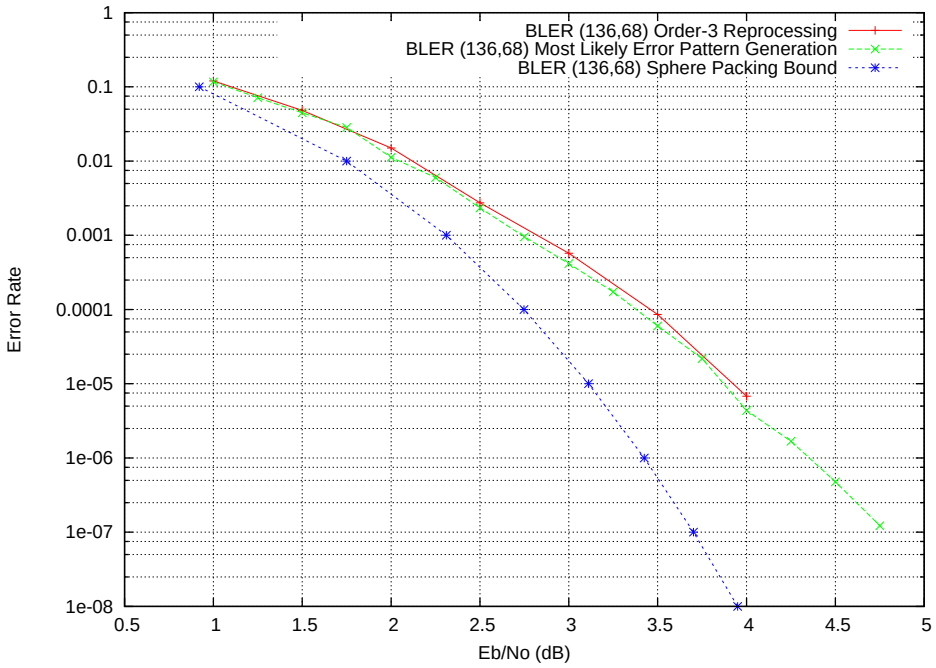

Figure 2.2: Comparison of Order-3 reprocessing and Most Likely Error Pattern Generation for (136,68,24) Code

Comparing the performance of the multiple simulations to the performance of the decoder presented in [7], the performance is quite similar.

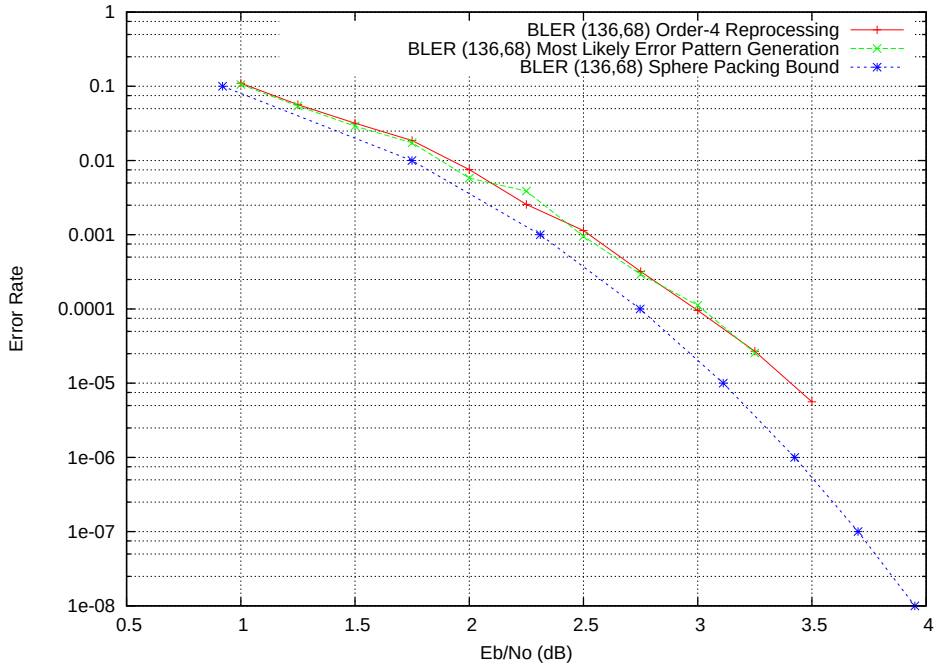

Figure 2.3: Comparison of Order-4 reprocessing and Most Likely Error Pattern Generation for (136,68,24) Code

Chapter 3

Improving Decoder Efficiency

Numerous steps can be taken to improve decoder efficiency. As presented in the previous chapter, the algorithm stopping criterion is an upper limit on the number of error patterns that are tested prior to choosing the closest candidate codeword. To guarantee true maximum-likelihood decoding, a total of 2^k error patterns need to be tested for a (n, k) block code. However, with codes in which $k > 20$, the testing of all 2^k error patterns becomes impractical, and an upper limit is required on the number of error patterns generated and tested. In this chapter, three methods are presented which improve decoding efficiency by requiring significantly less error patterns to be tested on average per codeword, while still being able to maintain the same decoding performance as if a fixed number of error patterns are tried.

The first method presented has been employed and studied in many papers. This method involves analyzing the parity section of a received codeword and not generating error patterns that negate the best correlation possible of the parity portion of the received codeword. The second method introduced permits the termination of error pattern generation if the squared distance between the received vector and candidate codeword is less than a set threshold, i.e., if the received

vector is determined to be within a hyper-sphere of a fixed radius about a codeword. Finally, the termination of error pattern generation using a fixed radius is improved by bounding overly confident positions in a received vector if they correlate positively with the same positions in a prospect codeword. This allows for more volume to be included in the corresponding decoding hyper-sphere, permitting early termination. By employing these methods, the number of error patterns that actually need to be tested can be significantly reduced from a fixed maximum, yet the same performance (bit error rate and block error rate) can be obtained as if the full number of error patterns are generated and tested.

3.1 Early Termination based upon Parity Correlation

Terminating error pattern generation based upon the magnitude of the parity bits was presented in [5]. No decoding performance in terms of bit error rate or block error rate is lost when this method is employed.

This method of terminating error pattern generation operates using the sorted quantized received vector $\underline{rq}$, whose derivation is detailed in section (2.1). The first k positions of the vector $\underline{rq}$ are the most reliable independent positions of the received vector $\underline{r}$ and are treated as the data bits in which a candidate error pattern is introduced. The remaining $n-k$ positions of $\underline{rq}$ correspond to the least reliable positions of $\underline{r}$ and are treated as parity bits.

As illustrated in section (1.2.1), maximizing the correlation of a prospective codeword, $\underline{c} \in C$, with a received vector is equivalent to finding the closest codeword to a received vector. The correlation equation is presented again in (3.1)

with the $c_i \in \{0, 1\}$ term modified to map 0 to -1 and 1 to 1.

$$\max_{\underline{c} \in C} \sum_{i=1}^{n} rq_i(2 \cdot c_i - 1) \tag{3.1}$$

The candidate codeword $\underline{c}$ in equation (3.1) can be broken into its individual components $\underline{c} = \underline{c}' + \underline{e}$. If $\underline{e}$ is set to $\underline{0}$, a new correlation emerges in equation (3.2). In this equation, the first k bits of $\underline{c}'$ are generated by the hard decision rule in equation (1.4), whereas the remaining $n - k$ parity bits are generated according to equation (1.15), with $e_i = 0$ for $i = 1, \ldots, k$.

$$\max_{\underline{c} \in C} \sum_{i=1}^{n} rq_i(2 \cdot c_i' - 1) \tag{3.2}$$

From equation (3.2), it is observed that the $\sum_{i=1}^{n} rq_i(2 \cdot c_i' - 1)$ term is constant with regards to any $\underline{c} \in C$, thus maximizing the correlation becomes dependent upon the error pattern. The error pattern, $\underline{e}$, has components $e_i \in \{0, 1\}$, with a 0 representing that no error should be introduced in a specific bit position, and 1 representing that an error should be introduced in the specific position. Errors in specific bit positions represent the flipping of a bit in the hard decision codeword $\underline{c}'$. The new incremental correlation based solely on the error pattern is observed in equation (3.3).

$$\max_{\underline{e} \in E} \left[\sum_{i=1}^{k} -2rq_i \cdot e_i + \sum_{i=k+1}^{n} -2rq_i(2 * c_i' - 1) * e_i \right] \tag{3.3}$$

The first correlation over the initial k bits, $\sum_{i=1}^{k} -2rq_i e_i \leq 0$. This is because all k initial data bits are assumed to be correct, along with their hard decision representations, and when an error is introduced in a position i, $1 \leq i \leq k$,

the cross product of $rq_i \cdot (2 \cdot c_i - 1)$ will be negative. The second correlation, $\sum_{i=k+1}^{n} -2rq_i(2 \cdot c'_i - 1) \cdot e_i$, over the $n - k$ parity bits can be both positive and negative depending upon the value of the parity bits of the candidate error pattern, $\underline{c}'$. The maximum possible value of the second correlation would be obtained if the parity bits of the candidate error pattern $e_i = 1$ for $i = 1, \ldots, n$, and the hard decision decoding of $\underline{rq}$ is opposite that of $\underline{c}'$ for all bit positions. This maximization is observed in equation (3.4).

$$max \sum_{i=k+1}^{n} -2rq_i(2 \cdot c'_i - 1) \cdot e_i = \sum_{i=k+1}^{n} 2|rq_i| \tag{3.4}$$

If equation (3.5) is satisfied for a particular error pattern, $\underline{e}$, the error pattern should be immediately discarded, as the incremental correlation is less than that if the all zero's error pattern is chosen.

$$\sum_{i=1}^{k} -rq_i e_i > \sum_{i=k+1}^{n} |rq_i| \tag{3.5}$$

If the most likely error pattern generation algorithm, described in section (2.3), is employed along with the relation in equation (3.5), a set of stopping criterion can be developed. The first summation of the inequality in equation (3.5), can be equated to the negation of the cost function in equation (2.17). Thus once the cost reaches $\sum_{i=k+1}^{n} |rq_i|$, no new error patterns should be searched as they are proven to push the candidate codeword, $\underline{c}$, further from the sorted, quantized received vector, $\underline{rq}$.

3.1.1 Extending Parity Based Termination

Parity-based termination can be further extended to provide more opportunity for early termination, while also preventing any loss in decoding performance. This method of termination is similar to that presented in the previous section, but extended to look at the maximum possible gain in correlation from the current closest codeword. This method is a direct modification of the method presented in [5], such that the termination metric can be checked with each increasing cost of equation (2.17) for most likely error pattern generation.

Given in equation (3.6) is a hard decision received vector $\underline{rq_{hd}}$ which has been derived according to the hard decision decoding rule in equation (1.4) from a sorted, quantized received vector, $\underline{rq}$. The vector, $\underline{rq}$, is derived from the original received vector $\underline{r}$ according to section 2.1.

$$\underline{rq}_{hd} = (rq_{hd_1}, rq_{hd_2}, \ldots, rq_{hd_n}) \tag{3.6}$$

In equation (3.7), the first k bits are the same as that of the hard decision received vector $\underline{rq}$, and the last $n-k$ bits are generated through a matrix multiplication of the transpose of the parity section of the rearranged H-matrix detailed in section 1.2.2.

$$\underline{c} = (c_1, c_2, \ldots, c_n) \tag{3.7}$$

Comparing $\underline{rq}$ to $\underline{c}$, the first k bits, considered data bits, are the same, whereas the remaining $n-k$ parity bits may be different. The bit positions where $rq_i \neq c_i$ can be used to form a expression detailing the maximum increase in correlation possible if an error pattern $\underline{e}$ is introduced and has parity bits $e_i = 1$ in only these

positions. This maximum increase in correlation is presented in equation (3.8).

$$\text{max\_corr\_increase} = \sum_{i=k+1}^{n} d_H(rq_{hd_i}, c_i) \cdot e_i \cdot |rq_i| \tag{3.8}$$

When an error pattern is introduced, positions $(e_1, e_2, \ldots, e_k)$ must contain at least a single 1. The correlation from these bits is always negative as shown in the previous section, and is the same as the negation of the cost function, $\sum_{i=1}^{k} rq_i e_i$ described in equation (2.17). When generating error patterns to test, if the cost is ever greater than max_corr_increase, it is not possible to have a closer codeword to the received vector because the total incremental correlation shown in equation (3.9) will be less than that of the current correlation.

$$\text{total\_correlation} = -\sum_{i=1}^{k} |rq_i| e_i + \sum_{i=k+1}^{n} d_H(rq_{hd_i}, c_i) \cdot e_i \cdot |rq_i| \tag{3.9}$$

If multiple error patterns are tried, and a new best codeword, $\underline{c}_{\text{best\_cw}}$, is found, the max_corr_increase of equation (3.8) requires updating. The update process for the new codeword, $\underline{c}_{best\_cw}$ is exactly the same as the method presented for the hard decision codeword $\underline{c}$, with $\underline{c}$ replaced by $\underline{c}_{best\_cw}$.

Performance gains using this method are presented in Table (3.1) for decoding the same (136,68,24) block code as presented in section 2.4 at an $E_b/N_0 = 3$ dB. The maximum number of error patterns allowed to be generated is 300,000 and the most likely error pattern generation algorithm of section 2.3 is used. The average number of correlations with termination represents the number of correlations tried per codeword averaged over a large number of codewords while simultaneously using the parity based termination metric.

Table 3.1: Performance Gain for (136,68,24) Code at an $E_b/N_0 = 3$ dB with Parity Based Termination

Parameter	Value
Max Correlations	300,000
Average Correlations with termination	14,321

3.2 Early Termination based upon Squared Distance

Early termination of error pattern generation can be based upon a squared distance calculation between a quantized received vector, $\underline{rq}$, and candidate codeword, $\underline{c}$, being less than or equal to a set threshold, as seen in equation (3.10). If the received vector is within the threshold squared distance of the quantized received vector, it will be considered the most optimal codeword with a high probability.

$$\sum_{i=1}^{k}(r_i - one\_index \cdot c_i)^2 \leq threshold \tag{3.10}$$

As presented in section 1.2, independent and identically distributed additive white Gaussian noise samples, $\underline{z}$, with zero mean and variance ${\sigma_z}^2$ are added to the encoded codeword, $\underline{x}$, to generate a received vector, $\underline{r} = \underline{x} + \underline{z}$. To observe how the noise samples form about the received codeword we refer to [8]. From this text, it is observed that the presence of noise will cause the received vector to fall within a fixed radius and variance, both determined by the current noise level, of the sent codeword.

Since the codeword sent, $\underline{x}$, is independent of the noise on the channel, the noise vector $\underline{z}$ can be examined itself. Presented in [8], we find an equation examining the mean square length l of the N-dimensional noise vector $\underline{z}$, normalized

by the dimensionality of the vector. This equation is presented again in (3.11).

$$l = \frac{1}{N}\sum_{i=1}^{N} {z_i}^2 \tag{3.11}$$

Examining the expected value of l in equation (3.12), the expected squared radius for the noise about a sent codeword normalized by the dimensionality of the noise vector, is found to be the variance of the noise, σ_z^2.

$$E[l] = E\left[\frac{1}{N}\sum_{i=1}^{N} {z_i}^2\right] = \frac{1}{N}\sum_{i=1}^{N} E\left[{z_i}^2\right] = {\sigma_z}^2 \tag{3.12}$$

Examining the variance of l in equation (3.11), it is found that the variance is inversely proportional to the dimensionality of the noise vector as well as directly proportional to the square of the noise variance, ${\sigma_z}^2$.

$$\begin{aligned} var(l) =& E\left[l^2\right] - E\left[l\right]^2 \\ =& E\left[\left(\frac{1}{N}\sum_{i=1}^{N} {z_i}^2\right)\left(\frac{1}{N}\sum_{i=1}^{N} {z_i}^2\right)\right] - {\sigma_z}^4 \\ =& E\left[\frac{1}{N^2}\left(\sum_{i=1}^{N} {z_i}^4 + \sum_{i=1}^{N}\sum_{\substack{j=1 \\ i\neq j}}^{N} {z_i}^2 {z_j}^2\right)\right] - {\sigma_z}^4 \\ =& \frac{1}{N^2}\left[N \cdot 3{\sigma_z}^4 + N(N-1){\sigma_z}^4\right] - {\sigma_z}^4 \\ =& \frac{2}{N}{\sigma_z}^4 \end{aligned} \tag{3.13}$$

Thus when the noise variance is fixed and the dimensionality is increased, the expected radius of the normalized mean square length of the noise vector remains constant, while the variance of the normalized mean square length about that expected radius decreases inversely proportional to the dimension of the noise vec-

tor. This phenomenon is known as sphere hardening.

If the expected mean square length radius, and variance are not normalized to the dimensionality of the noise vector, as presented in equations (3.14) and (3.15) respectively, then relations emerge describing the expected squared distance between the received vector and sent codeword and variance of the expected squared distance.

$$E[Nl] = N{\sigma_z}^2 \tag{3.14}$$

$$var(Nl) = 2\,N\,\left(\sigma_z^2\right)^2 \tag{3.15}$$

With the expected squared distance between the received vector and codeword determined, along with the expected variance of that distance, a threshold can be chosen relatively close to this value with a decoding rule that if the received vector is within this squared distance of the codeword, it is selected. Since expected squared distance depends on the noise power, if the operating SNR is low, it may be larger than half the minimum Euclidean squared distance between two codewords of a linear code. Half the Euclidean squared distance between two codewords can be determined from the minimum distance parameter for linear codes, d_{min}, representing the minimum number of bits in which two codewords differ. Based on the mapping from codeword bits to code vector amplitudes $(1 \rightarrow 1, 0 \rightarrow -1)$, the minimum squared Euclidean distance between codewords in a code with a minimum distance of d_{min} is presented in equation (3.16).

$$\text{dist}^2 = 4\,d_{min} \tag{3.16}$$

Thus half of the minimum Euclidean distance between two codewords is observed

Table 3.2: Expected Operating Parameters for (136,68,24) code at an $E_b/N_0 = 3$ dB

Parameter	Value
E_b/N_0 operating point	3.0 dB
σ_z^2	0.5011
Expected noise radius $E[Nl]$	68.1526
Expected noise variance $var(Nl)$	68.3056
d_{min}	24

in equation (3.17).

$$\left(\frac{\text{dist}}{2}\right) = \left(\frac{2\sqrt{d_{min}}}{2}\right)^2 = d_{min} \tag{3.17}$$

If the expected squared distance is greater than that in equation (3.17) for a particular code and noise level, care must be taken to choose a squared cutoff distance, d^2, such that the overall bit error rate and block error rate do not increase more than some allowable factor, say q, for a constrained error pattern count, ep_count, as seen in equation (3.18).

$$P(\text{error in codeword}|\text{cutoff dist} = d^2, \text{error pattern count} = \text{ep\_count}) < q \cdot P(\text{error in codeword}|\text{error pattern count} = \text{ep\_count}) \tag{3.18}$$

If the squared distance calculation of equation (3.19) is less than half the minimum Euclidean squared distance between two codewords, d_{min}, the prospective codeword is guaranteed to be the maximum likelihood decoded codeword with probability 1.

$$\sum_{i=1}^{k} (rq_i - one\_index \cdot c_i)^2 \leq d_{min} \tag{3.19}$$

Analyzing the same (136,68,24) code simulated in previous sections at an $E_b/N_0 = 3$ dB, yields the expected parameters presented in table 3.2. Simulating

the (136,68,24) code for varied squared radii thresholds produces two updated decoding statistics; an updated block error rate, and average number of error patterns required per codeword. As the squared radius threshold increases, the average number of error patterns tested is expected to decrease as more codewords are selected and the algorithm terminated. Correspondingly, the block error rate is expected to increase as more codewords are selected prematurely before the optimal codeword can be tested. Table 3.3 summarizes simulation results as the squared radius cutoff is varied, for a maximum error pattern count of 300,000 and an E_b/N_0 operating point of 3 dB. This simulation also employs the parity-based termination scheme of section 3.1.1. The results of Table 3.3 are presented in Figure 3.1. There is a clear trade-off between complexity and probability of block decoding error. It is observed that increasing the radius linearly, results in an exponential increase in the block error rate, while also an exponential decrease in the number of correlations required. For example, if the tolerable increase in the block error rate, q, of equation (3.18) is less than 2, then the average number of error patterns tested per codeword can drop by a factor of 3.2. The parity-based termination scheme provides an independent reduction in complexity, by reducing the number of error patterns tested on average by a factor of 67.1.

3.3 Early Termination based upon Bounding Overly Confident Positions

As presented in section 3.2, a prospective codeword can be selected as the maximum likelihood codeword if it is within a squared radius slightly larger than expected squared noise radius of the quantized received vector. In this section, the squared radius termination metric is considered along with a new form of the squared distance calculation allowing for early termination on a larger percentage

Table 3.3: Simulated Operating Parameters for (136,68,24) Code at an $E_b/N_0 = 3$ dB with Varied Termination Radius

Threshold Squared Radius	Block Error Rate	Average Number of Error Patterns Tested Per Codeword
0 - no cutoff	1.04E-4	14,321
77.5	1.26E-4	6,224
80	1.83E-4	4,470
82.5	3.22E-4	3,155
85	7.10E-4	2,125
87.5	1.56E-3	1,231
90	3.34E-3	683
92.5	6.25E-3	398
95	1.24E-2	230

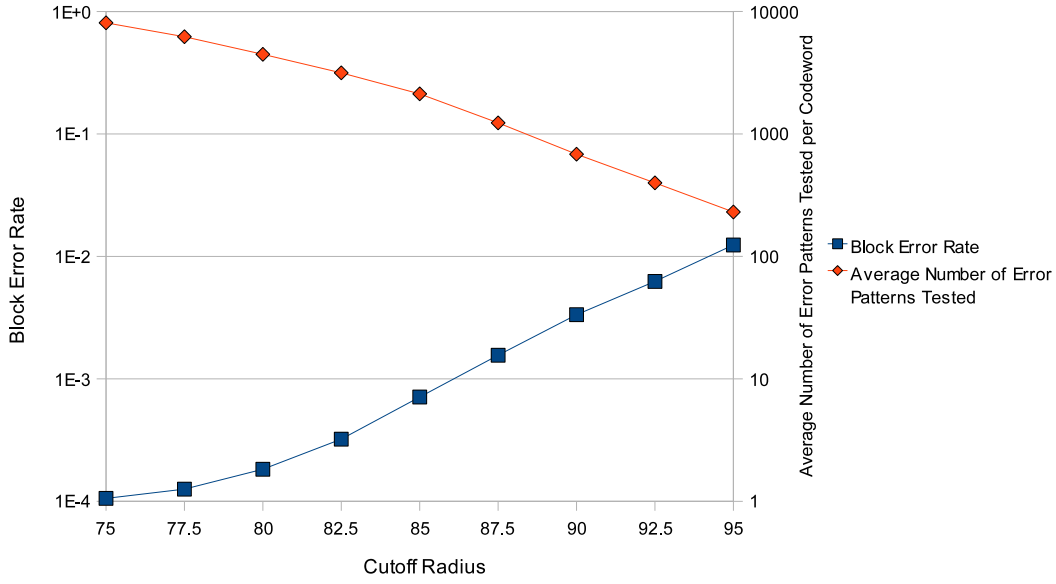

Figure 3.1: Block Error Rate and Average Number of Error Patterns Tested vs. Cutoff Radius

of candidate codewords.

The original distance calculation, presented again in equation (3.20), is mod-

ified such that if a bit in the sorted, quantized received vector, $\underline{rq}_i$, and candidate codeword, $\underline{c}_i$ have the same hard-decision value, and the same value of $|rq_i| >$ one_index, then the value of r_i can be truncated to $sign(r_i) \cdot$ one_index for the current distance calculation.

$$\sum_{i=1}^{k}(rq_i - one\_index \cdot c_i)^2 \leq \text{threshold} \tag{3.20}$$

This truncation diminishes the squared distance contribution from any dimension in the received vector satisfying the two criterion. Equation (3.21) presents the truncation process to create a new truncated vector, rq_trunc, that can be used in equation (3.22) to perform the updated squared distance calculation.

$$\text{rq\_trunc}_i = \begin{cases} \text{sign}(rq_i) \cdot one\_index, & \text{when } |rq_i| \geq one\_index \text{ and} \\ & \text{sign}(c_i) = \text{sign}(r_i) \\ rq_i, & \text{otherwise} \end{cases} \tag{3.21}$$

$$\sum_{i=1}^{k}(\text{rq\_trunc}_i - one\_index \cdot c_i)^2 \leq \text{threshold} \tag{3.22}$$

Truncating overly confident bits that have the same hard decision value as the prospective codeword changes the expected squared radius and variance of the noise vector from the received codeword. In a single dimension, the probability density function describing the truncation of AWGN noise is

$$p_Z(z) = \frac{1}{2}\delta(z) + (1 - u(z))\frac{1}{\sqrt{2\pi}\sigma_z}e^{-z^2/2\sigma_z^2}, \tag{3.23}$$

where $u(z)$ is the unit-step function

$$u(z) = \begin{cases} 1, z \geq 1 \\ 0, z < 1, \end{cases} \tag{3.24}$$

and $\delta(z)$ is the Dirac impulse function. The mean and variance of the truncated random variable are respectively

$$E[z] = \int_{-\infty}^{\infty} z \cdot p_z(z)\, dz = \frac{-\sigma_z}{\sqrt{2\pi}} \tag{3.25}$$

$$\sigma_z'^2 = E[z^2] - E[z]^2 = \frac{{\sigma_z}^2}{2} - \left(\frac{-\sigma_z}{\sqrt{2\pi}}\right)^2 = \frac{{\sigma_z}^2}{2}\left(1 - \frac{1}{\pi}\right). \tag{3.26}$$

Similar to the derivation in section 3.2, the mean and variance of the expected normalized mean square length, l, of the truncated N-dimensional noise vector, $\underline{z}$, are now determined using the truncated noise probability density function, $p_Z(z)$. The truncated vector's squared radius per dimension is

$$l = \frac{1}{N}\sum_{i=1}^{N} {z_i}^2 \tag{3.27}$$

The mean is calculated as

$$E[l] = E\left[\frac{1}{N}\sum_{i=1}^{N} {z_i}^2\right] = \frac{1}{N}\sum_{i=1}^{N} E[{z_i}^2] = \frac{{\sigma_z}^2}{2}, \tag{3.28}$$

where ${\sigma_z}^2$ is the variance of the untruncated AWGN noise.

Equation (3.29) presents the variance of l, based upon the truncated noise

distribution.

$$\begin{aligned} var(l) =& var\left(\frac{1}{N}\sum_{i=1}^{N} {z_i}^2\right) \\ =& \frac{1}{N^2}\sum_{i=1}^{N} var\left({z_i}^2\right) \\ =& \frac{1}{N^2}\sum_{i=1}^{N}\left(E\left[{z_i}^4\right] - E\left[{z_i}^2\right]^2\right) \\ =& \frac{1}{N^2}\sum_{i=1}^{N}\left(\frac{3{\sigma_z}^4}{2} - \left(\frac{{\sigma_z}^2}{2}\right)^2\right) \\ =& \frac{1}{N^2}\sum_{i=1}^{N}\left(\frac{5{\sigma_z}^4}{4}\right) \\ =& \frac{1}{N}\left(\frac{5{\sigma_z}^4}{4}\right) \end{aligned} \tag{3.29}$$

If the truncated squared radius is not normalized by the dimension, then the mean and variance are presented in equations (3.30) and (3.31) respectively, describing the expected squared distance between the received vector and sent codeword, and variance of the expected squared distance.

$$E[Nl] = \frac{N{\sigma_z}^2}{2} \tag{3.30}$$

$$var(Nl) = \frac{N\,5{\sigma_z}^4}{4} \tag{3.31}$$

A comparison of the expected squared radii and variance of the radii of the noise vector for both the pure and truncated distance calculations is presented in Table 3.4. From this table, it is observed that, by using the truncated confidence distance calculation, the expected noise radius decreases by a factor of 2 and the variance of the expected radius decreases by a factor of 1.6. The truncation effectively clusters the received points within a tighter hyper-shell about the sent

Table 3.4: Noise Radius and Variance Comparison for Pure and Truncated Distance Calculations

	Pure Distance	Truncated Distance
Radius, $(E[Nl])$	$N{\sigma_z}^2$	$\frac{N{\sigma_z}^2}{2}$
Variance, $(\sigma^2[Nl])$	$2\,N\ ({\sigma_z}^4)$	$\frac{N\,5{\sigma_z}^4}{4}$

Table 3.5: Expected Operating Parameters for (136,68,24) code at $E_b/N_0 = 3$ dB Using Truncation of Confident Bits

Parameter	Value
E_b/N_0 operating point	3.0 dB
$\sigma_z'^2$	0.1708
Expected noise radius $E[Nl]$	34.0807
Expected noise variance $var(Nl)$	42.7021
d_{min}	24

codeword.

The expected squared radius of the received vector for this decoding method is tabulated in Table 3.5 for the $(136, 68, 24)$ code. All tabulated parameters are targeted at an operating point of $E_b/N_0 = 3$ dB.

The same simulation as was done in section 3.2, with the exception of sweeping over smaller radii, is done with the newly introduced truncated distance calculation for the (136,68,24) code at an SNR of $E_b/N_0 = 3$ dB, with the complexity constrained to a maximum error pattern count of 300,000. The results, detailing the updated block error rate and average number of correlations per codeword, are presented in Table 3.6. The results of the simulation are also presented in Figure 3.2.

As observed previously with the pure distance calculation, there is a clear trade-off between complexity and probability of block decoding error. It is observed that increasing the radius linearly, results in an exponential increase in the

Table 3.6: Simulated Operating Parameters for (136,68,24) code at an $E_b/N_0 = 3$ dB with varied termination radius using truncated distance calculation

Threshold Squared Radius	Block Error Rate	Average Number of Error Patterns Tested Per Codeword
0 - no cutoff	1.03E-4	14,246
45	1.04E-4	6463
47.5	1.02E-4	4,096
50	1.09E-4	2,286
52.5	1.11E-4	1,183
55	1.49E-4	603
57.5	2.65E-4	314
60	6.04E-4	174
62.5	1.75E-3	130
65	5.05E-6	105
67.5	1.31E-2	70

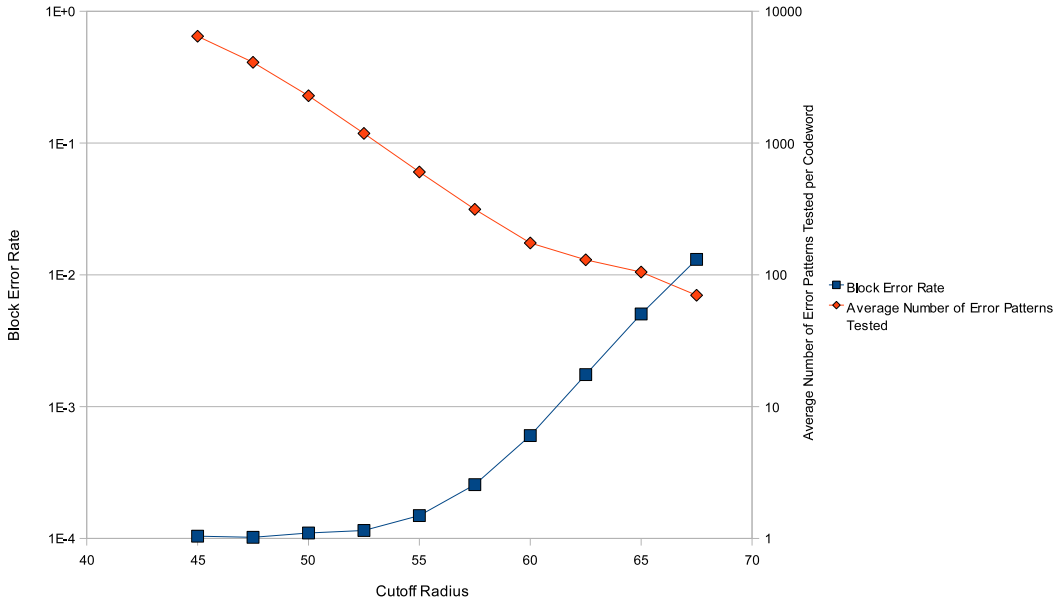

Figure 3.2: Block Error Rate and Average Number of Error Patterns Tested vs. Cutoff Radius Using Truncated distance calculation

block error rate, while also an exponential decrease in the number of correlations required. There is a sizeable performance gain, even more so than with the pure distance calculation, if the tolerable factor of increase in the block error rate, q, of equation (3.18) remains less than 2. The average number of error patterns tested per codeword can drop by a factor of 23.62, independent of the reductions from the parity-based termination scheme, and compared to testing all 300,000 error patterns, the reduction is by a factor of 497. If q is required to be smaller, a sizeable performance gain can still be achieved with this method.

When comparing the truncated distance calculation method to the pure distance calculation method of section 3.2, the truncated distance method performs superior to the full distance method. Truncated distance calculations permit earlier algorithm termination, while simultaneously preserving decoding performance. A performance comparison between the two methods can be observed in Figure 3.3. In this figure, the termination radius for the pure distance calculation has been normalized, such that the block error probabilities of each method coincide in the 10^{-4} region.

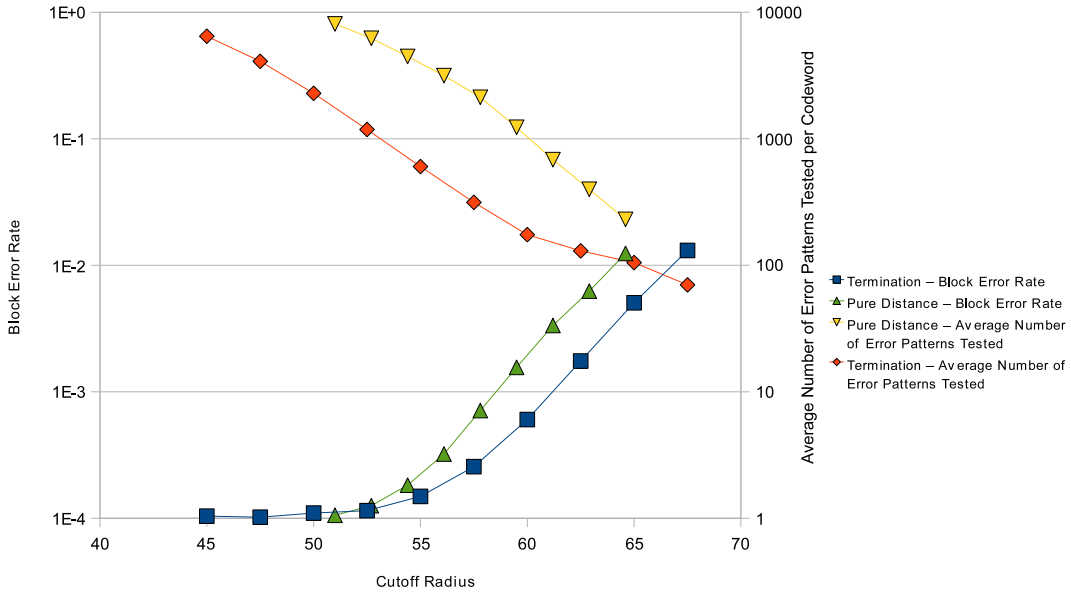

Figure 3.3: Block Error Rate and Average Number of Error Patterns Tested vs. Cutoff Radius Using Truncated Distance Calculation and Pure Distance Calculation

Chapter 4

Bounding Code Performance

In this chapter the decoding algorithms discussed in the previous chapters are investigated to determine the best achievable performance of the decoder as a function of the code rate and block size, and subject to a complexity constraint on the maximum number of error patterns tested per codeword. The decoding methods used in this section include the most likely error pattern generation algorithm of section 2.3, the parity-based termination of section 3.1.1, and the distance based calculations described in section 3.3 with no restriction on the cutoff radius. The best performing code at each rate is selected and simulated again with a higher limit on the maximum number of tested error patterns. These results are compared with the performance of Low Density Parity Check (LDPC) codes generated at Comtech AHA Corporation, that can be implemented directly in their family of hardware-based decoders [11].

4.1 Code Performance at a Fixed Rate with Varied Block Size

In this section three code rates, 0.5, 0.75, and 0.9, are investigated for various block sizes. For each code and rate, the code construction method is presented, along with simulated decoding performance results, subject to the complexity con-

Table 4.1: Collection of Rate 0.5 Codes Generated to Test Decoder

Code	Actual Rate	Construction Method
(127,64)	0.504	BCH - t=9
(136,68)	0.5	Obtained From Martin Tomlinson [10]
(160,79)	0.494	Shortened BCH - t=9 (511,430) shortened by 351 bits
(180,90)	0.5	Shortened BCH - t=10 (511,421) shortened by 331 bits
(200,101)	0.505	Shortened BCH - t=11 (511,412) shortened by 311 bits
(255,131)	0.513	BCH - t=19

straint of a maximum of 300,000 error patterns tested per codeword.

4.1.1 Rate 0.5 Code Performance

A collection of 6 codes with rate close to 0.5 have been generated and are tabulated in Table 4.1. The method of construction for each code is detailed in the 'Construction Method' column. All BCH and shortened BCH codes were created using the MAGMA Computational Algebra System [9].

The decoding performance of each code in Table 4.1 is simulated subject to the complexity constraint of a maximum of 300,000 error patterns tested per codeword. The block error rate performance of each code is presented in Figure 4.1. In this figure, it is observed that the decoding performance increases with block size, as expected, until the (180,90) code is reached. At this block size, the performance of the larger codes start to decrease. Although one expects codes of a larger block size to perform better than codes with smaller block size, the performance decrease in the figure is due to the complexity constraint on the maximum number of error patterns tested per codeword. Codes that are half-rate and have a block size $n \geq 180$ require more than 300,000 error patterns to be tested to compete with the best performing (160,79) code. The (160,79) code is selected as the best

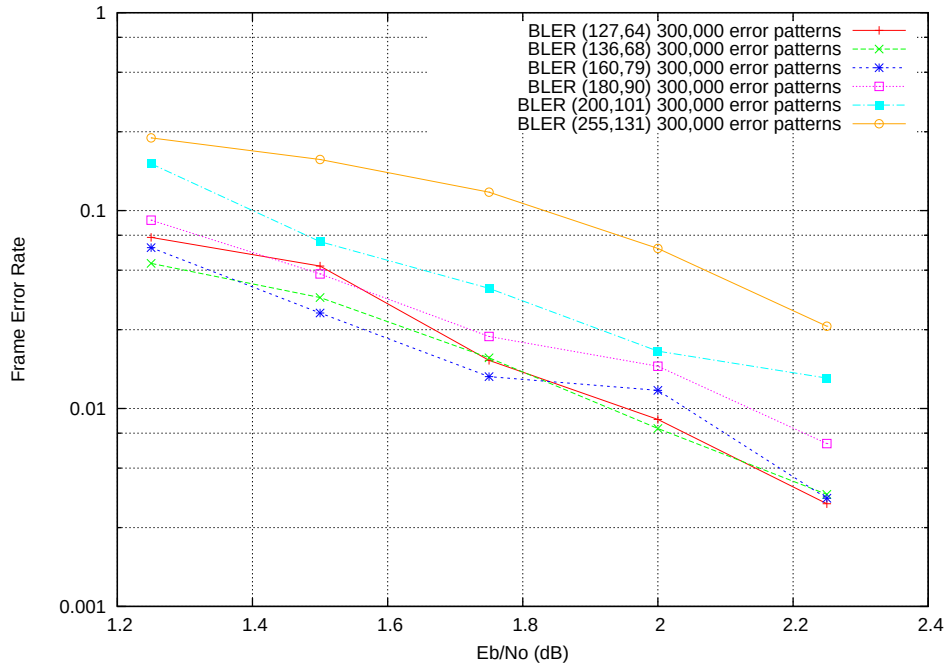

Figure 4.1: Rate 0.5 Block Error Rate Comparison - 300,000 Error Patterns

performing rate 0.5 code and is used in section 4.2.1 to compare to LDPC codes.

4.1.2 Rate 0.75 Code Performance

A collection of 6 codes with rate close to 0.75 have been generated and are tabulated in Table 4.2. The method of construction for each code is detailed in the 'Construction Method' column. All BCH and shortened BCH codes were created using the MAGMA Computational Algebra System [9].

Each code in Table 4.2 is simulated subject to the complexity constraint of a maximum of 300,000 error patterns tested per codeword. The block error rate performance of each code is presented in Figure 4.2. In this figure, it is observed that the decoding performance increases with block size, as expected, until the (280,208) code is reached. At this block size, the performance of the larger codes

Table 4.2: Collection of Rate 0.75 Codes Generated to Test Decoder

Code	Actual Rate	Construction Method
(127,92)	0.724	BCH - t=5
(255,191)	0.749	BCH - t=8
(280,208)	0.743	Shortened BCH - t=8 (511,439) shortened by 231 bits
(320,239)	0.746	Shortened BCH - t=9 (511,430) shortened by 191 bits
(383,284)	0.741	Shortened BCH - t=11 (511,412) shortened by 128 bits
(511,385)	0.753	BCH - t=14

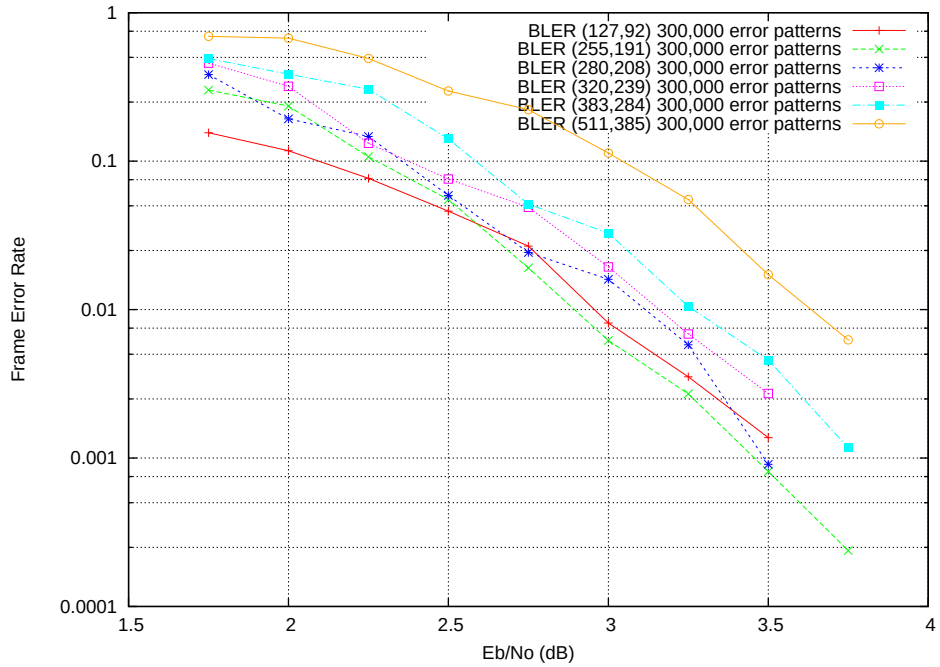

Figure 4.2: Rate 0.75 Block Error Rate Comparison - 300,000 Error Patterns

start to decrease. Again, the reason for the performance decrease is due to the complexity constraint on the maximum number of error patterns tested per codeword. Codes that are rate 0.75 and have a block size $n \geq 280$ require more than 300,000 error patterns to be tested to compete with the best performing (255,191) code. The (255,191) code is selected as the best performing rate 0.5 code and is used in section 4.2.2 to compare to LDPC codes.

Table 4.3: Collection of Rate 0.9 Codes Generated to Test Decoder

Code	Actual Rate	Construction Method
(400,360)	0.9	Shortened BCH - t=4 (1023,983) shortened by 623 bits
(500,450)	0.9	Shortened BCH - t=5 (1023,973) shortened by 523 bits
(600,540)	0.9	Shortened BCH - t=6 (1023,963) shortened by 423 bits
(700,630)	0.9	Shortened BCH - t=7 (1023,953) shortened by 323 bits

4.1.3 Rate 0.9 Code Performance

A collection of 4 codes with rate 0.9 have been generated and are tabulated in Table 4.3. The method of construction for each code is detailed in the 'Construction Method' column. All BCH and shortened BCH codes were created using the MAGMA Computational Algebra System [9].

Each code in Table 4.3 is simulated subject to the complexity constraint of a maximum of 300,000 error patterns tested per codeword. The block error rate performance of each code is presented in Figure 4.3. In this figure, it is observed that the decoding performance increases with block size, as expected, until the (600,540) code is reached. At this block size, the performance of the larger codes start to decrease. Codes that are rate 0.9 and have a block size $n \geq 600$ require more than 300,000 error patterns to be tested to compete with the best performing (500,450) code. The (500,450) code is selected as the best performing rate 0.9 code and is used in section 4.2.3 to compare to LDPC codes.

4.2 Best Code at a Fixed Rate vs. LDPC Code Performance

The best performing codes of rates 0.5, 0.75, and 0.9 found in section 4.1 are selected and the respective decoding performance tested, allowing a maximum of

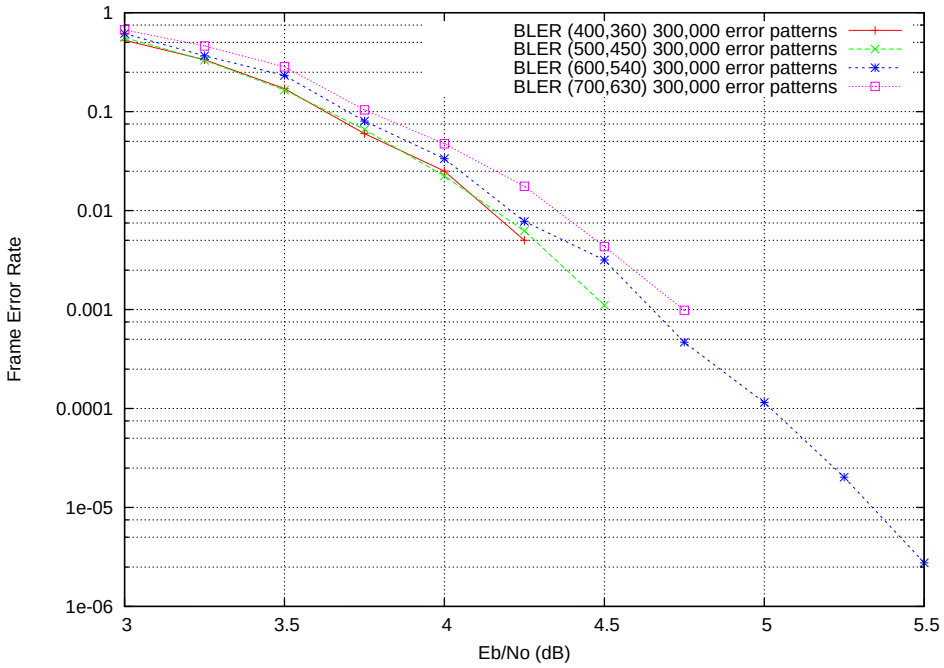

Figure 4.3: Rate 0.9 Block Error Rate Comparison - 300,000 Error Patterns

3,000,000 error patterns tested per codeword. For each code rate, LDPC codes of a larger block size are generated and decoding performance simulated using Comtech AHA's proprietary software. This software provides bit level parallelism with their hardware-based LDPC decoders. All the LDPC codes used have a fixed row weight and variable column weights in the construction of the parity check matrix. Decoding of all LDPC codes is done using 30 iterations of a min-sum algorithm which utilizes the 3 largest terms when performing the iterative updates.

4.2.1 Rate 0.5 Code Performance

The best performing (160,79) code of section 4.1.1 is simulated again with a maximum of 3,000,000 error patterns tested per codeword. Additionally, the 3 LDPC codes tabulated in Table 4.4 are generated and simulated using Comtech AHA's

Table 4.4: Collection of Rate 0.5 LDPC Codes

Code	Actual Rate
(480,240)	0.5
(720,360)	0.5
(960,480)	0.5

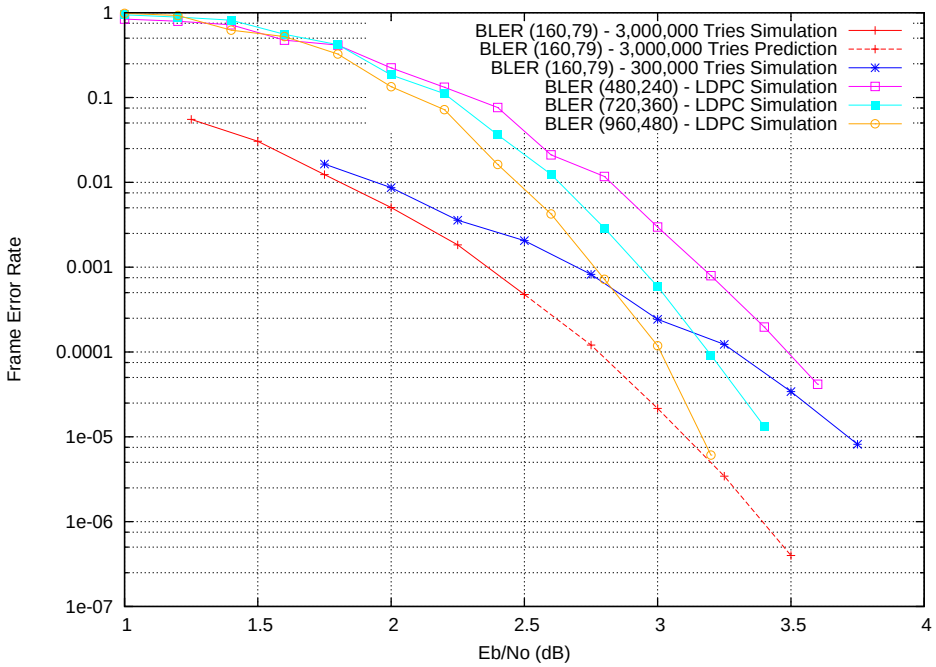

Figure 4.4: Rate 0.5 Block Error Rate Comparison

software-based LDPC decoder. Simulation results of the block error rate for the (160,79) code and LDPC codes are presented in Figure 4.4. The simulation results for the bit error rate are presented in Figure 4.5. In both figures, simulations for the (160,79) code are provided for 300,000 and 3,000,000 error patterns. The 3,000,000 error pattern simulation includes a performance projection if the code were to be further simulated at lower bit and block error rates. This projection is based upon a linear extrapolation of the growing separation between the sphere packing bound for the (160,79) code and the current code performance.

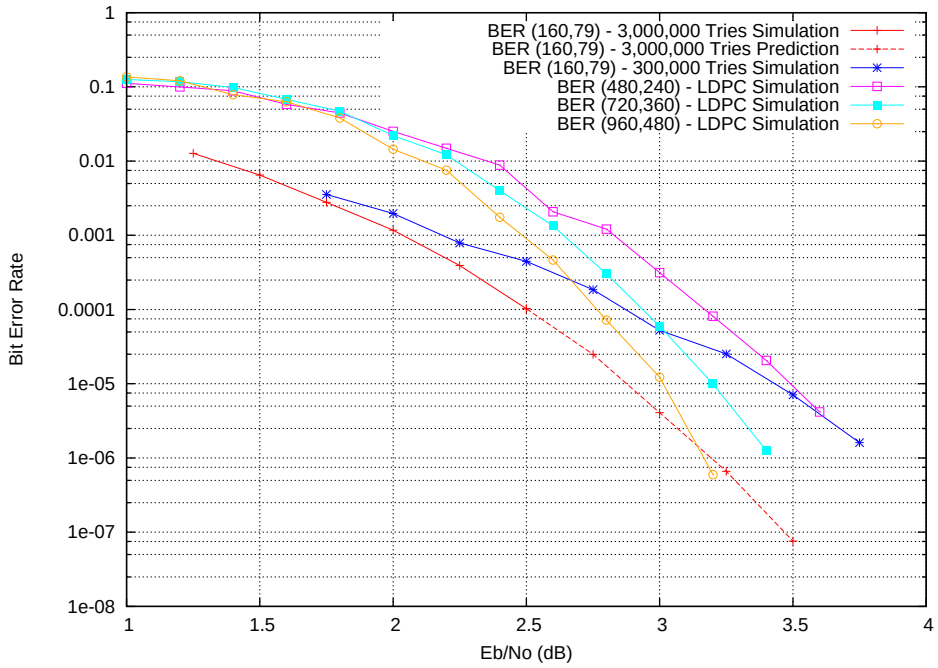

Figure 4.5: Rate 0.5 Bit Error Rate Comparison

From the simulation results, it is observed that the (160,79) code decoded using a maximum of 3,000,000 error patterns per codeword can beat the performance of the (960,480) LDPC code (with a block size 6-times larger) above a block error rate of 10^{-5}. The (160,79) code decoded using a maximum of 300,000 error patterns is able to beat the performance of the (480,240) LDPC code above a block error rate of $2 \cdot 10^{-5}$. In terms of bit error rate, the (160,79) code decoded using a maximum of 3,000,000 error patterns is able to beat the (960,480) LDPC code down to 10^{-6}.

4.2.2 Rate 0.75 Code Performance

The best performing (255,191) code of section 4.1.2 is simulated again with a maximum of 3,000,000 error patterns tested per codeword. Additionally, the 2

Table 4.5: Collection of Rate 0.75 LDPC Codes

Code	Actual Rate
(1000,750)	0.75
(2064,1548)	0.75

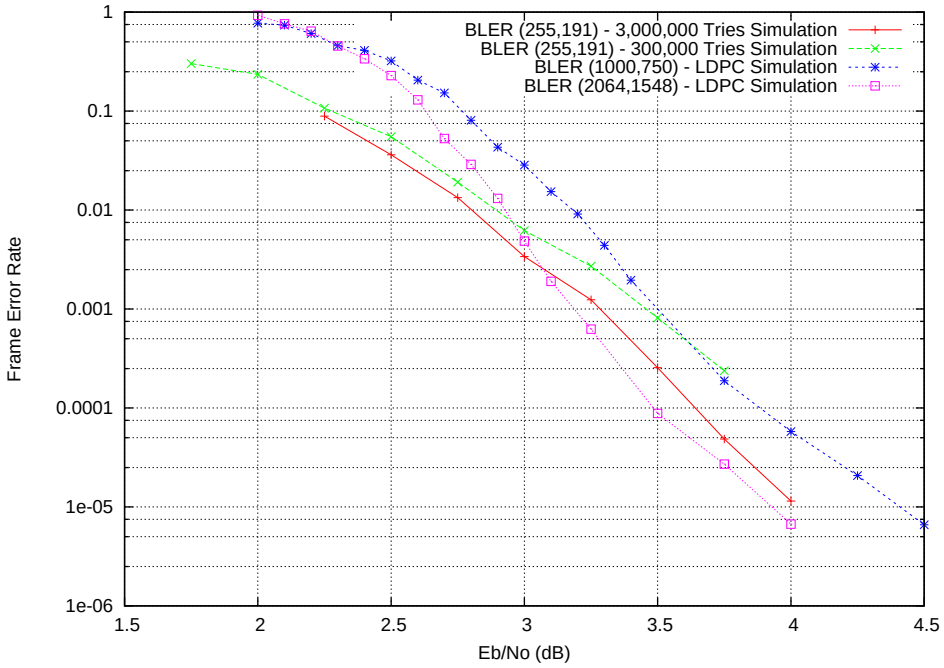

Figure 4.6: Rate 0.75 Block Error Rate Comparison

LDPC codes tabulated in Table 4.5 are generated and simulated using Comtech AHA's software-based LDPC decoder. Simulation results of the block error rate for the (255,191) code and LDPC codes are presented in Figure 4.6. The simulation results for the bit error rate are presented in Figure 4.7. In both figures, simulations for the (255,191) code are provided for 300,000 and 3,000,000 error patterns.

From the simulation results, it is observed that the (255,191) code decoded using a maximum of 3,000,000 error patterns per codeword can beat the performance of the (2064,1548) LDPC code (with a block size 8-times larger) above a

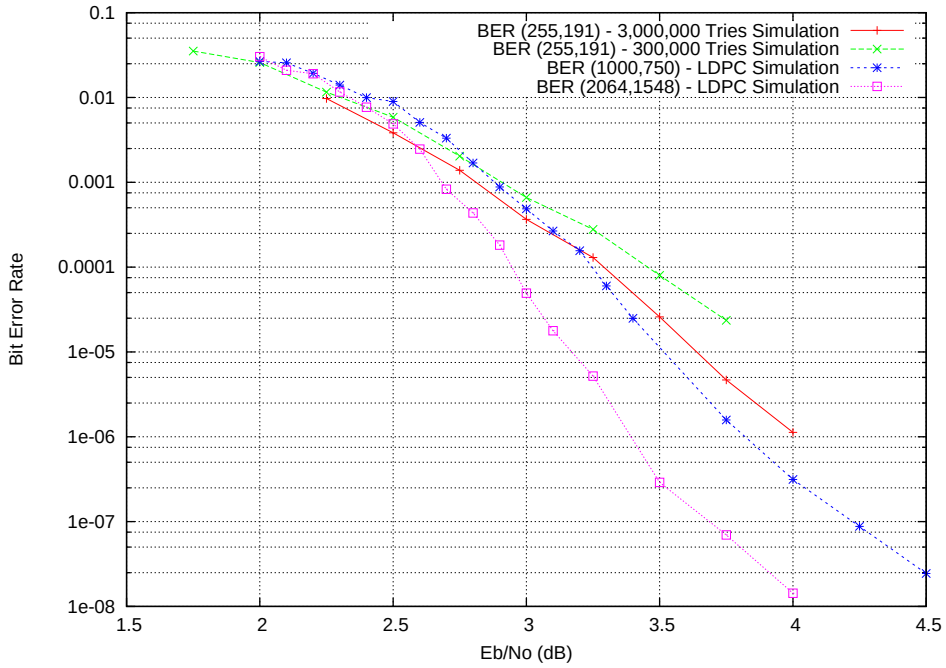

Figure 4.7: Rate 0.75 Bit Error Rate Comparison

block error rate of $2 \cdot 10^{-3}$ and below a block error rate of $8 \cdot 10^{-6}$. The (255,191) code decoded using a maximum of 300,000 error patterns is able to beat the performance of the (1000,750) LDPC code above a block error rate of $5 \cdot 10^{-4}$ and below $3 \cdot 10^{-5}$. In terms of bit error rate, the (255,191) code decoded using a maximum of 3,000,000 error patterns is able to beat the (1000,750) LDPC code above 10^{-4}, and below $2 \cdot 10^{-7}$ due to a floor in the LDPC code.

4.2.3 Rate 0.9 Code Performance

The best performing (500,450) code of section 4.1.3 is simulated again with a maximum of 3,000,000 error patterns tested per codeword. Additionally, the 2 LDPC codes tabulated in Table 4.6 are generated and simulated using Comtech AHA's software-based LDPC decoder. Simulation results of the block error rate

Table 4.6: Collection of Rate 0.9 LDPC Codes

Code	Actual Rate
(2520,2268)	0.9
(5040,4536)	0.9

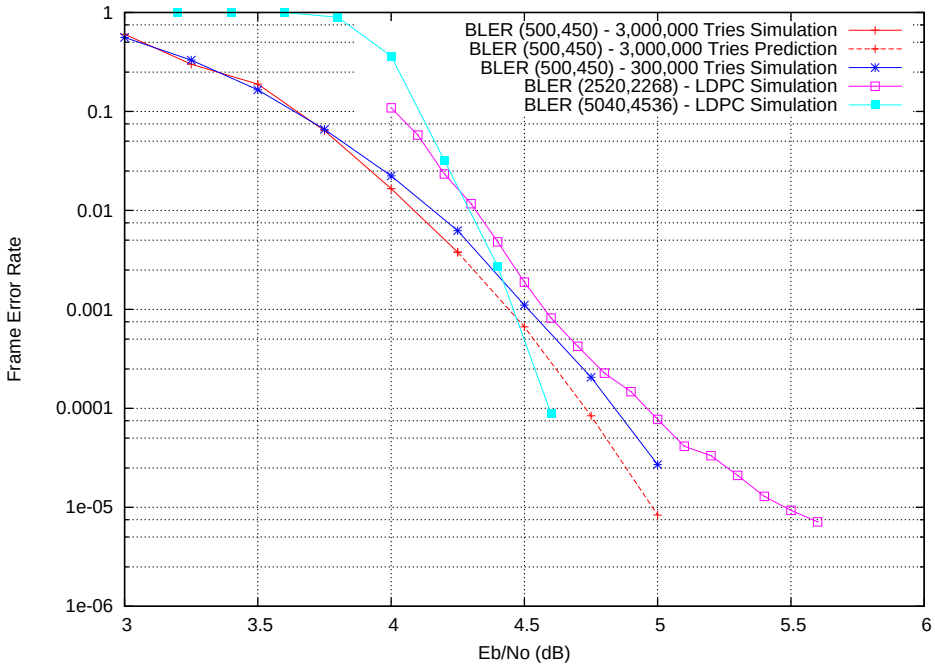

Figure 4.8: Rate 0.9 Block Error Rate Comparison

for the (500,450) code and LDPC codes are presented in Figure 4.8. The simulation results for the bit error rate are presented in Figure 4.9. In both figures, simulations for the (500,450) code are provided for 300,000 and 3,000,000 error patterns. The 3,000,000 error pattern simulation includes a performance projection if the code were to be further simulated. This projection is based upon a linear extrapolation of the growing separation between the sphere packing bound for the (500,450) code and the current code performance.

From the simulation results, it is observed that the (500,450) code decoded using a maximum of 300,000 error patterns can beat the performance of the

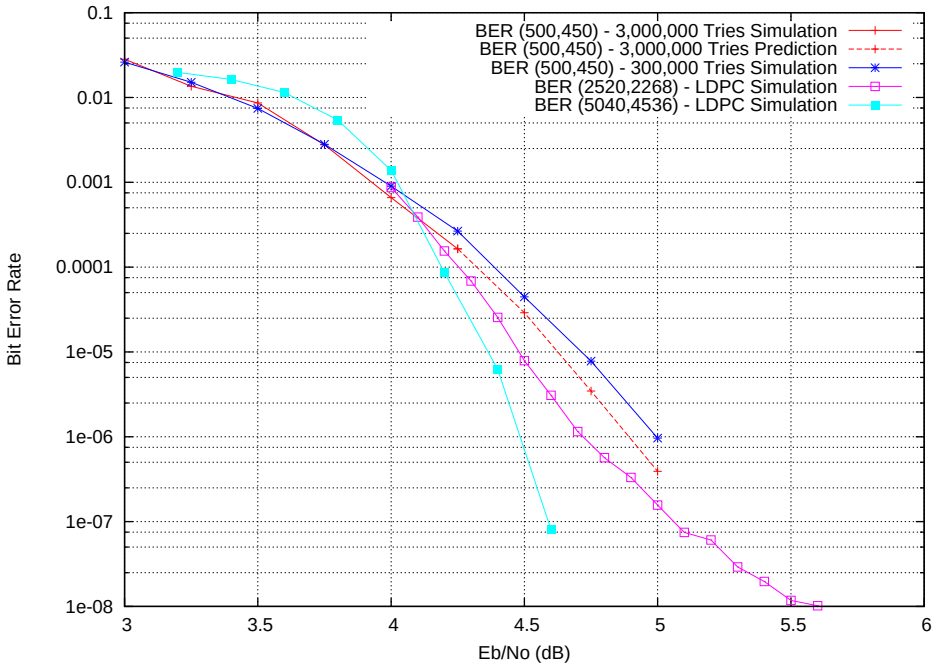

Figure 4.9: Rate 0.9 Bit Error Rate Comparison

(2520,2269) LDPC code (with a block size 5-times larger) at any block error rate. Comparing the bit error rate performance, both LDPC codes beat the (500,450) code below a bit error rate of $2.5 \cdot 10^{-4}$ due to their steeper waterfall regions.

Chapter 5

Conclusion

5.1 Summary of Thesis Contribution

In this work, a soft-decision decoding method for linear block codes has been developed and thoroughly explored. An algorithm that introduces error patterns in the most likely order in which they are expected to occur was compared to a common order-i reprocessing algorithm. Two supplemental algorithms were presented which allow for an increase in the speed of the decoder by reducing the number of error patterns that need to be tested per codeword, one reliant upon the correlation of the parity portion of the received vector, and the other based upon a truncated squared distance calculation between the received vector and prospective candidate codeword. The performance of the decoder subject to a complexity constraint on the maximum number of error patterns tested per codeword has also been explored for multiple code rates to find the largest code with the best bit error rate and block error rate performance. For each code rate, the best performing code was selected and compared with LDPC codes of larger block sizes.

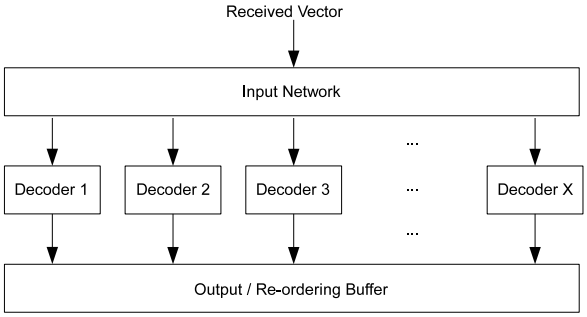

Figure 5.1: Decoder Network

5.2 Areas of Further Research

While this work fully explored the maximum performance of a decoder, it did nothing to address how to implement an overall efficient decoder. Further research should be done on the possibility of creating a network of decoders, similar to that of Figure 5.1. A decoder network could be constructed in such a way that a received vector could be assigned to any empty decoder, and when the vector is decoded into the best guess codeword, the best guess codeword would be released to an output buffer. The output buffer would then release best guess codewords in the same order that they were received as received vectors. This setup would effectively allow multiple codewords to be processed simultaneously, and allow one codeword to be processed longer than others if it requires more error patterns. Constraints such as the maximum allowed latency through the system, size of the output buffer, and number of decoders would need to be considered.

Furthermore, a single decoder should be studied in detail to find its maximum achievable throughput. This upper bound on throughput would be subject to a implementation constraint, such as the maximum number of error patterns that could be tested per second.

Bibliography

[1] C.E. Shannon, A Mathematical Theory of Communication," Bell Syst. Tech. J., pp. 379-423 (Part 1); pp. 623-56 (Part 2), July 1948.

[2] V.S. Pless and W.C. Huffman, Handbook of Coding Theory Volume I, Amsterdam: Elsevier, 1998.

[3] S. Lin and D. Costello Jr., Error Control Coding, 2nd ed. Upper Saddle River: Prentice Hall, 2004.

[4] B.G. Dorsch, "A decoding algorithm for binary block codes and J-ary output channels", IEEE Trans Inf. Theory, 1974, 20, pp. 391 - 394.

[5] M. Fossorier and S. Lin, "Soft-Decision Decoding of Linear Block Codes based on Ordered Statistics," IEEE Trans Inf. Theory, 1995, 41, pp. 1379-1396.

[6] C.E. Shannon, "Probability of Error for Optimal Codes in a Gaussian Channel," Bell Syst. Tech. J., Vol 38, No. 3, pp 611-656, 1959.

[7] M. Tomlinson, C. Tjhai, and M. Amrboze, "Extending the Dorsch decoder towards achieving maximum-likelihood decoding," IET Communications, 2007, Vol. 1, No. 3, pp 479-488.

[8] Wozencraft, J. and Jacobs, I. , ‘Principles of Communication Engineering’, New York: John Wiley & Sons, Inc., 1965.

[9] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. J. Symbolic Comput., 24(3-4):235-265, 1997.

[10] M. Tomlinson and S. Collison, "Private Communication," 2007.

[11] Comtech AHA Corporation, ‘Low Density Parity Check Codes’, Avaliable: http://www.aha.com/show_prod_type.php?id=8. [Accessed April 14, 2009].