KUNGL TEKNISKA HÖGSKOLAN
Royal Institute of Technology

Iterative Decoding of Product Codes

OMAR AL-ASKARY

RADIO COMMUNICATION SYSTEMS LABORATORY

KUNGL TEKNISKA HÖGSKOLAN
Royal Institute of Technology

Iterative Decoding of Product Codes

OMAR AL-ASKARY

A dissertation submitted to
the Royal Institute of Technology
in partial fulfillment of the requirements
for the degree of Licentiate of Technology

April 2003

TRITA—S3—RST—0305
ISSN 1400—9137
ISRN KTH/RST/R--03/05--SE

RADIO COMMUNICATION SYSTEMS LABORATORY
DEPARTMENT OF SIGNALS, SENSORS AND SYSTEMS

Abstract

Iterative decoding of block codes is a rather old subject that regained much interest recently. The main idea behind iterative decoding is to break up the decoding problem into a sequence of stages, iterations, such that each stage utilizes the output from the previous stages to formulate its own result. In order for the iterative decoding algorithms to be practically feasible, the complexity in each stage, in terms of number of operations and hardware complexity, should be much less than that for the original non-iterative decoding problem. At the same time, the performance should approach the optimum, maximum likelihood decoding performance in terms of bit error rate.

In this thesis, we study the problem of iterative decoding of product codes. We propose an iterative decoding algorithm that best suits product codes but can be applied to other block codes of similar construction. The algorithm approaches maximum likelihood performance. We also present another algorithm which is suboptimal and can be viewed as a practical implementation of the first algorithm on product codes. The performance of the suboptimal algorithm is investigated both analytically and by computer simulations. The complexity is also investigated and compared to the complexity of GMD and Viterbi decoding of product codes.

This page intentionally contains only this sentence.

Acknowledgements

The work on my Licentiate thesis was long and exciting even though it had many setbacks. I have learned a lot during these years. Maybe the most important thing I learned is that there is so much more to learn. I am mostly grateful to my colleagues who taught me most of the fundamentals of what I have written in this thesis.

Special thanks to Professor Slimane Ben Slimane, my adviser, for his valuable comments and enlightening discussions and mostly for encouraging me and giving me the chance to make this work possible. Many thanks to Professor Jens Zander for his encouragement and guiding. Professor Zander's guidelines of the procedure with which a Ph.D. student can smoothly go through the various stages of his research was of great help to me through my work. Also, many thanks to Magnus Lindström for our discussions and for his extensive help with my computer problems. Special thanks go to Lise-Lotte Wahlberg for the extensive help in clearing the practical and administrative details regarding printing and presenting the thesis. Thanks also to Johan Malmgren for proofreading the thesis. I shouldn't forget all my other colleagues in Radio Systems group for their feedback. I am also grateful to Jan Nilsson from FOI for his valuable comments.

I would also like to thank my former colleagues in Datatransmission group in Linköping University for their help. Special thanks go to Professor Thomas Ericson who taught me the art of approaching problems by careful defining, and thus understanding them. Also many thanks to Jonas Olsson and Danyo Danev for our fruitful discussions.

Special thanks to Professor Youzhi Xu from Jönköping University for his valuable comments and feedback.

I would also like to thank Wafaa, my wife, for tolerating my childish tantrums caused by stress during my research. Finally, I would like to thank Muhammad-Ali, my son. The quality time we spent together changing diapers or playing was a much needed distraction that helped me relax after work.

This page intentionally contains only this sentence.

To my parents who endure my absence.
To my wife who tolerates my presence.

This page intentionally contains only this sentence.

Contents

List of Figures

List of Abbreviations

AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
B-M	Berlekamp-Massey decoding
BMD	Bounded Minimum Distance
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetrical Channel
GMD	Generalized Minimum Distance
i.i.d.	independent identically distributed
LDPC	Low Density Parity Check
MAP	Maximum Aposteriori Probability
ML	Maximum Likelihood
MPSK	M-ary Phase Shift Keying
MDS	Maximum Distance Separable
OP	Number of Operations
RM	Reed-Muller code
RS	Reed-Solomon code

This page intentionally contains only this sentence.

Chapter 1

Introduction

1.1 Background

The task of data communication on a noisy channel involves many different problems which can be dealt with more or less separately. One of the main concerns is how to deal with the errors introduced by the communication channel to the received message. Claude Shannon showed in his famous work, see [1], that this problem can be remedied by channel coding in the communication system. This lead to an explosive search for constructions of powerful channel codes, where we mean by powerful, that they have good error correction capability.

The concept of product codes is a good way to obtain long and powerful codes by using simple constituent codes. Product codes were first presented by Elias in [2]. In their simplest form, product codes can be represented as a set of matrices such that each row in these matrices is a codeword in one constituent code and each column is a codeword in another constituent code. These codes had a very significant role in providing many theoretical results in coding theory. For instance, in [2], Elias constructed multidimensional product codes that, asymptotically, have a non-vanishing rate and non-vanishing fractional minimum distance[1]. The product codes constructed by Elias were the first example of codes with such asymptotic property. The idea of product codes was later developed into the concept of concatenated codes by Forney, [3] [4], Blokh and Zyablov, [5] and Zyablov and Zinoviev, [6] [7].

Product codes are also very efficient in wireless communication channels. Wireless communication channels suffer from noise and fading due to multi-path prop-

[1]Fractional minimum distance is the ratio between the minimum distance and the length of the code.

agation. Fading causes burst errors in the transmitted data. Interleaving is, in general, used to transform burst errors into random errors which then can be corrected by forward error control codes. However, the effectiveness of interleaving is limited by the maximum delay that can be supported by the communication system. Product codes, on the other hand, have the proper structure for burst error correction without the need for extra interleaving.

A well known decoding procedure is to decode the received message up to half the minimum distance of the code. Such a decoder is called a Bounded Minimum Distance decoder (BMD). However, this decoding procedure is not very efficient in decoding powerful codes such as product codes. This is caused by the following explanation: Powerful codes have, in general, large minimum distances and thus the risk for the occurrence of undecodable error patterns is higher. I.e., when the number of errors occurring in the received message is slightly greater than half the minimum distance of the code, there is a high risk that there are no codewords at all at a distance less than half the minimum distance from the received message. The result would be a decoding failure of the BMD decoder even though the sent codeword is the closest codeword to the received message [2]. This is true for all classes of long codes with large minimum distance.

This will have a direct effect on the system performance and the coding scheme will operate properly only at high Signal-to-Noise Ratio (SNR). In fact, with BMD decoding alone it is impossible to approach the channel capacity. This is illustrated in Figure 1.1 where asymptotic bounds (upper and lower) on the rates of codes as a function of the transition probability of a memoryless Binary Symmetric Channel (BSC), with BMD decoding are given [8, pp. 557-566]. It is observed that for transition probabilities greater than 0.06, the gap between the rates of optimum codes and that predicted by the channel capacity is very large. This clearly shows the short coming of bounded minimum distance decoding and thus more powerful decoding algorithms (beyond half the minimum distance) are needed.

It is worth noting that in Figure 1.1, there is no constructive proof that codes satisfying the Gilbert-Varshamov lower bound exist, see [8, pp. 306-315]. However, Blokh and Zyablov showed in [9] that concatenated codes that reach this bound exist. Since their proof is not constructive, it is reasonable to say that, in practice, the code used should have a rate *much less* than the rates predicted by the Gilbert-Varshamov lower bound when the decoding is limited to half the minimum distance.

In general, the more powerful a code is, the more difficult it is to decode. The decoding complexity of long block codes with large minimum distance increases very fast. For instance Lin showed that the complexity of decoding Bose-Chaudhuri-

[2]An example of this is that product codes can correct burst errors of Hamming weight much greater than half the minimum distance of the code. Decoding only up to half the minimum distance means that burst errors will not be corrected

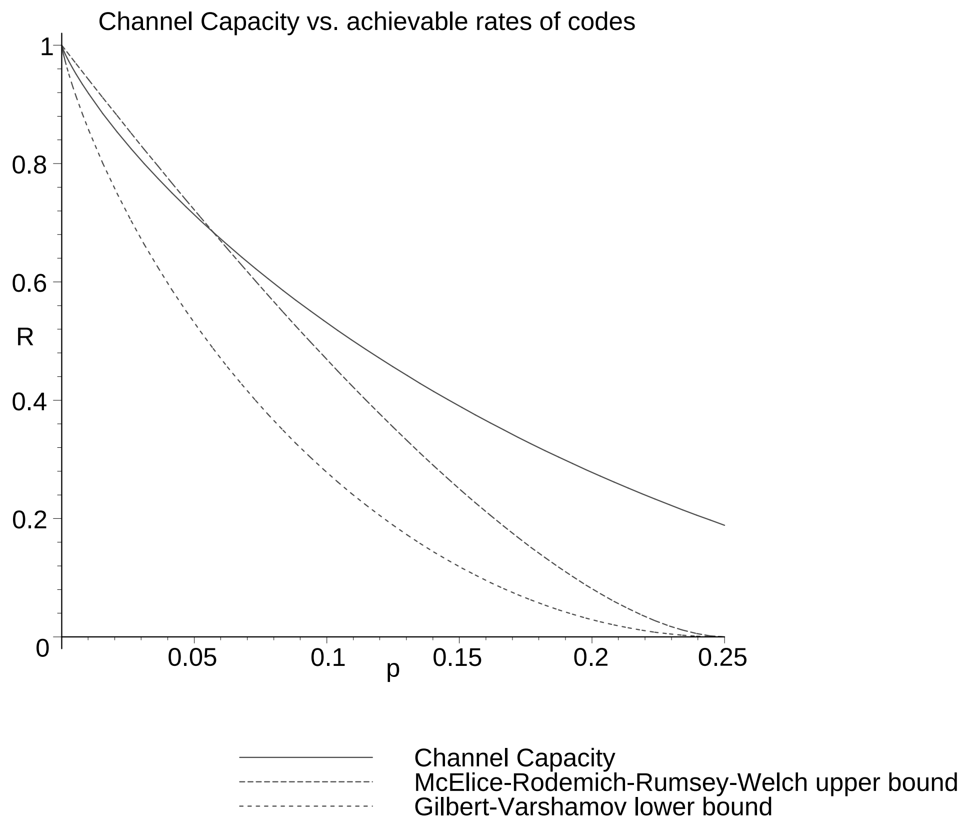


Figure 1.1: Channel capacity compared to achievable code rates using BMD decoding on BSC.

Hocquenghem (BCH), codes increases with, at least, the square of the minimum distance [10, pp. 129-131]. This rule, however, is not totally applicable to product codes and codes related to them. Usually, decoding a product codes is performed by successive decoding operations of the constituent codes of the product code used. Therefore, the complexity of decoding product codes is more dependent on the complexity of decoding their, much smaller, constituent codes.

1.2 Product Codes and their Advantages

Even though the minimum distance of product codes is much smaller than the minimum distance of optimal codes of comparable length, the error correcting potential of product codes is quite large. In order to illustrate this capability, we observe some of the characteristics of product codes. One important property of product codes is burst error correction. It can be easily seen that all error patterns that are restricted to a number of rows less than half the minimum distance of the column code or a number of columns less than half the minimum distance of the row code are correctable.

Also, for random errors, if the number of errors in each row does not exceed half the minimum distance of the row code then these errors are correctable. This is true, in a similar fashion, for the case of errors not exceeding half the minimum distance of the column code in each column. Needless to say, a received message with such error patterns is still closest to the original sent codeword, since every other codeword is even further from the received message. Therefore, a Maximum Likelihood (ML) decoder is also capable of correcting these error patterns.

We also observe that the covering radius [3] of product codes is, usually, much greater than half the minimum distance of the code, see Cohen et al [11, page 17] and [12]. This means that even when the error exceeds half the minimum distance of the code, there is still a possibility to correct all the errors when using an ML decoder. This definitely doesn't mean that it is possible to correct all such errors, rather, it means that not all such errors are uncorrectable. Thus, random error patterns such that the number of errors in some rows and some columns exceed half the minimum distance of the row code or the column code, respectively, might still be correctable using a maximum likelihood or near maximum likelihood decoder. A bounded minimum distance decoder, on the other hand can never correct random errors of this type. It is this improvement in error correction that the algorithms introduced in this thesis posses and which makes them superior to other algorithms like Generalized Minimum Distance (GMD) decoding with a slight increase in complexity.

The main reasons why we decided to investigate the decoding of product codes can be summarized as follows:

1. Low complexity decoding algorithms will allow the use of more powerful product codes. The results obtained by implementing Turbo decoding on product codes prove that these codes have very good error correcting potential. The only obstacle is the high complexity required for decoding them with Turbo decoders.
2. Product codes include interleaving as an inherent feature in their design. They, therefore, have very good burst error correction capability which in turn makes them good candidates for radio communication.
3. Product codes are very closely related to multilevel codes and generalized concatenated codes. We hope that an efficient algorithm devised for product

[3]The covering radius of a linear code can be defined as the maximum Hamming weight of a correctable error pattern from the all zero codeword

codes can easily be modified for decoding concatenated codes and multilevel codes.

It should also be mentioned that the simple structure of product codes makes them even more attractive from the analytical point of view when analyzing the qualities and the decoding algorithms of these codes.

1.3 Advantages of Iterative Decoding

Iterative algorithms for decoding block codes are in general a good compromise between complexity and performance. Even though in most cases the results obtained by iterative decoding only approach the performance of optimal algorithms such as ML decoding, the decrease in decoding complexity makes iterative algorithms an attractive alternative to optimal algorithms. Gallager's Low Density Parity Check (LDPC) codes, [13], and their iterative decoding algorithms and Berrou and Glavieux's turbo codes and turbo decoding are clear proofs of the claim that iterative decoding is an efficient replacement to optimal decoding such as ML decoding or Maximum Aposteriori Probability (MAP) decoding. This claim becomes quite clear especially when the size of the code used is very large which makes optimal decoding practically impossible.

Using a long and powerful code is a basic requirement for utilizing the full capacity of the channel. Utilizing the full capacity of the channel is especially important in the case of limited resources in the channel where many users compete to use the same bandwidth. There are many codes that fulfill the requirements of being long and powerful. However, the problem of decoding these codes is, in many cases, the decisive factor of using or not using them in applications. Therefore, the iterative decoding algorithms presented by Gallager, Berrou and Glavieux and the improvements made on these basic algorithms by later researchers are very significant. This is because they open the door for using certain codes that were previously considered impractical from the point of view of decoding.

Utilizing turbo decoding and related decoding algorithms with product codes show a very clear improvement in the performance of these codes in comparison to previous, suboptimal, decoding algorithms. However, the complexity of turbo decoding product codes is quite high and is actually exponentially increasing with the code length if the fractional minimum distances of the constituent codes were kept constant. The problem, we believe, is inherent and is caused by the structure of product codes. The main reason for the high complexity of turbo decoding of product codes is that, usually, the constituent codes are chosen to be optimal [4] or

[4]As previously mentioned, optimal codes are the codes that have the largest possible minimum distance for a given length and cardinality

near optimal block codes, e.g., BCH codes and Reed-Muller codes. These codes are very hard to decode with known MAP algorithms except in the cases when the codes have very high or very low rates. However, MAP algorithms are essential in the case of turbo decoding. We believe, therefore, that in order to increase the performance of product codes without drastic increase in complexity calls for developing new algorithms that are categorically different from turbo decoding tailored to fit the qualities of product codes.

In this thesis we present two algorithms for decoding product codes. These algorithms are iterative in nature and are based on successive decoding of the rows and columns of the incoming message. This iterative technique makes these proposed algorithms similar to turbo decoding algorithms. The similarity, however, stops there and the proposed algorithms are fundamentally different from turbo decoding. The performance of the first algorithm proposed in the thesis approaches ML decoding while the performance of the other proposed algorithm, which we will refer to as the suboptimal algorithm, only approach the performance of ML algorithms with increasing complexity. It will be shown both analytically and with the help of computer simulations that the second algorithm gives rather good results at a fraction of the complexity needed for ML decoding. The main objective of designing the new algorithms is to keep the complexity of the decoding to a minimum, comparable to BMD decoding or GMD decoding of the product code in the cases of hard decoding and soft decoding respectively. We mean by comparable that the difference in complexities between the proposed algorithm and BMD decoding does not exceed or is a fraction of the total complexity of decoding.

Both proposed algorithms in the thesis are based on representing product codes as an intersection of two codes. These two codes can easily be list decoded by list decoding of the rows and columns of the matrix that is undergoing decoding. When using the optimal algorithm, the rows and columns of the received message are list decoded and those lists are used without further alteration throughout all the decoding iterations. The suboptimal algorithm, on the other hand, list decodes the rows or the columns from the previous iteration instead of the original message and forgets this list after using it in each iteration. This is done so as to keep the size of the list as small as possible and, as will be shown in Section 4.2, to decrease the total number of iterations needed for decoding.

1.4 Related Work

Recently, many researchers have looked into the problem of decoding beyond half the minimum distance of the code. A possible approach is to choose a simple code construction, usually a concatenation of two or more codes, and try to decode the received message much beyond half the minimum distance of the code. Even if the minimum distance of the code used is small in comparison with optimal codes, the result will, in average, be better than that of a long code with large minimum distance and BMD decoding. A very good example of such approach was given by Glavieux and Berrou [14] with parallel concatenation of two convolutional codes and an iterative decoding algorithm, a combination which they called Turbo Codes. It was later discovered that Gallager, in a much earlier work [13], proposed a similar idea which he called Low Density Parity Check Codes. The work continued in the same track as Berrou and Glavieux to implement the same decoding algorithm, namely, Turbo decoding, on other types of concatenated codes. Many researchers implemented Turbo decoding with product codes.

Decoding product codes up to half their minimum distance of the code is quite simple. It is just an instance of the GMD decoding introduced by Forney [3]. However, since the minimum distance of product codes is small compared to optimal codes, BMD decoding of product codes is not very interesting from a practical point of view. Because of that product codes did not gain a lot of attention during the past years. Interest in product codes increased with the introduction of Turbo decoding. One of the reasons is that product codes are closely related to concatenated codes and multilevel codes, [15] [16]. A solution that works for product codes can easily be extended to concatenated codes and multilevel codes. The other reason is that product codes have a very simple structure and that makes them easy to analyze and to implement.

Hagenauer, Offer and Papke were the first to investigate the idea of Turbo decoding of product codes [17]. It was, however, found that direct application of turbo decoding on product codes is too complex to implement and not possible to use for codes of interest. Turbo decoding requires MAP decoding on the Trellis of the constituent codes [18]. Since the constituent codes of product codes are usually chosen to be linear block codes, their trellis complexity is quite high even for very simple codes [19].

To overcome this complexity problem, Pyndiah, [20], proposed a new iterative decoding algorithm for product codes. The proposed algorithm is an approximation to Turbo decoding where the MAP decoding of the constituent codes was replaced by a modification of Chase's second decoding algorithm [21]. However, the approximations proposed by Pyndiah are not always explained or motivated by a theoretical background. This makes Pyndiah's algorithm very hard to analyze and, therefore, becomes even harder to improve or generalize to other codes.

What is common in the results of both Hagenauer and Pyndiah is that the error correcting performance of product codes was shown to be much greater than that predicted by BMD decoding. In fact, the obtained results showed performance comparable to that of Turbo codes when the number of iterations is kept small and with a comparable decoding complexity. There is nowadays a great interest in using product codes in combination with Turbo decoding both from the universities and the industry, [22]. The decoding complexity is, however, still very high and only very short product codes can be used.

Many researchers tried afterward to analyze or improve the efficiency of iterative decoding of product codes. For example, Fang *et al.*, [23] introduce a special family of product codes that are easily decodable by trubo decoding. Martin *et al.* [24] tried to decrease the complexity of turbo decoding of product codes by lowering the complexity of MAP decoding of the constituent codes and Be'ery *et al.*, [25] [26], investigated the convergence of turbo-decoding of product codes.

After the first reports about the effectiveness of iterative decoding of product codes were published, many researchers investigated the possibility of using product codes in communication systems. The following is but a sample of a huge number of work published in the area. Hagenauer, [27] investigated the possibility of using product codes in Forward error correcting for Code Division Multiple Access (CDMA) systems. Picart and Pyndiah, [28], investigated the possibility of using product codes in combination with turbo-decoding in multilevel constructions. Sanzi *et al.*, [29], investigated the possibility of iterative channel estimation and decoding in multi-carrier systems using product codes. Buch and Burkert, [30] investigated the use of Unequal error protection with product codes with turbo decoding and Souvignier *et al.*, [31] tried to implement product codes with turbo decoding in partial response channels.

1.5 Scope of the Thesis

The thesis can be regarded as an extensive discussion and motivation of the results on this subject presented by the author in [32], [33] and [34]. The thesis begins in Chapter 2, by defining product codes and discuss their features. We also give a rather detailed discussion about the background of these codes and the background of the decoding problem that we address in this thesis. The aim of Chapter 2 is to point out the missing parts in the previous research regarding decoding of product codes. This way we give a motivation to our research and the solutions we present in the thesis.

In Chapter 3 we introduce a new representation for product codes defined as an intersection of two simpler codes. From this representation of product codes, a decoding algorithm, referred to as the "basic decoding algorithm", is developed. We prove in Chapter 3 that, under certain conditions, the basic decoding algorithm has ML performance. We also prove in Chapter 5 that for good channels, i.e., sufficiently low transition probability for binary symmetrical channels or high signal to noise ratio for Euclidean channels, the complexity of the basic decoding algorithm will be less than that for maximum likelihood Viterbi decoding on the trellis of product codes. The basic decoding algorithm is very useful from a theoretical point of view and can be used to derive bounds on the decoding complexity of product codes and their performance in Additive White Gaussian Noise (AWGN) channels as done in Chapter 3 and in the Appendix.

The complexity of the basic decoding algorithm can be limited to a preset upper limit. By varying this limit one can trade decoding complexity for performance and vice versa. We try in this thesis to express the performance in terms of the chosen complexity.

As mentioned above, the basic decoding algorithm only has theoretical value. We, therefore, developed an iterative decoding algorithm, referred to as "suboptimal iterative decoder", based on the basic decoding algorithm. This is done in Chapter 4. The proposed iterative decoder shares many features with Turbo decoding and especially that proposed by Pyndiah [20]. It is, however, fundamentally different from Turbo decoding. Turbo decoding, including Pyndiah's algorithm, rely on MAP decoding of the rows and columns in each iteration. The result from MAP decoding is used to generate a vector of *extrinsic reliability information* to be used in the following iteration. The iterative algorithm proposed in this thesis is based on *list decoding* the rows and the columns at each iteration so that each row or column will have several candidates and choosing only one of these candidates for each row or each column to be a part of the result for the current iteration. I.e., no MAP decoding or generation of extrinsic information is needed. The complexity of the iterative decoding algorithm can be controlled by limiting the complexity of the list decoder for the rows and the list decoder for the columns. Decreasing the complexity is, however, done at the expense of performance. Since one of the main issues of the thesis is to limit the complexity, we concentrate mostly on the implementations where the complexity is as low as possible, comparable to GMD decoding of the product code under investigation.

The simulation results in Chapter 6 show that the performance of the proposed decoding algorithm is better than GMD decoding for comparable complexities. For example, by using GMD decoders of the constituent codes as list decoders for the rows and for the columns and by keeping the total number of iterations sufficiently small, then we will keep the complexity of the iterative decoder comparable to that of GMD decoding of the product code. This is due to the fact that GMD decoding of the product code incorporates GMD decoders of the constituent codes.

Also, in Chapter 6, the complexity of the suboptimal iterative decoder is studied and it is shown that for the codes used in the simulation, the complexity of decoding truly is comparable to the complexity of GMD decoding of the same product code.

This page intentionally contains only this sentence.

Chapter 2

Product Codes

In this chapter we try to present the basic concepts regarding the definition of product codes, their characteristics and the decoding algorithms that can be used in combination with product codes. We also touch on the subject of complexity of using product codes in communication systems. We hope that by presenting and partly analyzing the alternative methods of decoding, we will be able to give and explain the motivation for devising a new decoding algorithm that can be used with product codes.

Most of the information in this chapter is compiled from articles and results of other researchers, e.g., Elias [2], MacWilliams and Sloane [8], Forney [3] [35], Viterbi [36], Berrou and Glavieux [14], Pyndiah Pynd98 and Vardy [19].

2.1 Definition of Product Codes

Product codes are serially concatenated codes [8, pp. 568-571]. They were first presented by Elias in 1954 [2]. The concept of product codes is very simple and powerful at the same time where very long block codes can be constructed by using two or more much shorter constituent codes. Consider two block codes $\mathcal{A}'$ and $\mathcal{B}'$ with parameters $[n, k_A, d_A]$ an $[m, k_B, d_B]$, respectively. It should be noted that we follow MacWilliams and Sloane's notations, [8], where n, k_A and d_A are,r respectively, the length, dimension and minimum Hamming distance of the code $\mathcal{A}'$ and m, k_B and d_B are,r respectively, the length, dimension and minimum Hamming distance of the code $\mathcal{B}'$. The rates of the codes $\mathcal{A}'$ and $\mathcal{B}'$ are denoted by R_A and R_B, respectively, and are equal to:

$$R_A \triangleq \frac{k_A}{n}, R_B \triangleq \frac{k_B}{m}.$$

The product code $\mathcal{C}$ is obtained from the codes $\mathcal{A}'$ and $\mathcal{B}'$ in the following manner:

1. place $k_A \times k_B$ information bits in an array of k_B rows and k_A columns.
2. coding the k_B rows using the code $\mathcal{A}'$. Note that the result will be an array of k_B rows and n columns.
3. coding the n columns using the code $\mathcal{B}'$.

The construction of the product code $\mathcal{C} = \mathcal{A}' \times \mathcal{B}'$ is illustrated in Figure 2.1. The

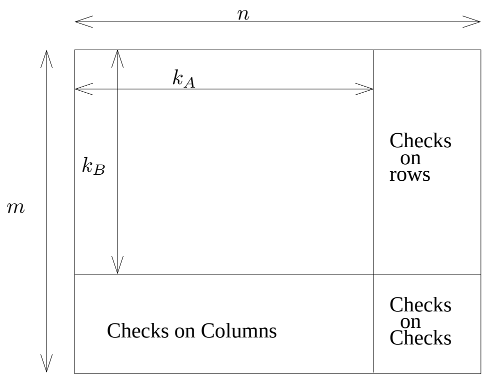


Figure 2.1: Construction of product codes

parameters of the resulting product code will be $[mn, k_A k_B, d_A d_B]$ and its rate will be equal to $R_A R_B$. Therefore, we can construct long block codes starting by combining two short codes.

Another, and more general, definition of product codes is as follows. For the same codes $\mathcal{A}'$ and $\mathcal{B}'$ defined above, the product code $\mathcal{C}$ is an $[mn, k_A k_B, d_A d_B]$ code whose codewords can be represented by all $m \times n$ matrices such that each row and each column of these matrices are members of the codes $\mathcal{A}'$ and $\mathcal{B}'$ respectively. Note that this definition is valid for all constituent codes over any alphabet, linear or non-linear.

Let G_A and G_B be the generator matrices of the codes $\mathcal{A}'$ and $\mathcal{B}'$ respectively. The generator matrix for the code $\mathcal{C}$ can be obtained from the generator matrices, G_A and G_B, of the constituent codes by taking the *Kronecker product*, denoted by $\otimes$, of the two matrices, see MacWilliams and Sloane [8, pages 421 and 568], as will be shown. Let G_A be the generator matrix of the code $\mathcal{A}'$ and let G_B be the generator matrix of the code $\mathcal{B}'$. The generator matrix of the product code $\mathcal{C}$, denoted by G_C is equal to:

$$G_C = G_B \otimes G_A,$$

where the Kronecker product between two matrices, L and M of dimensions $a \times b$ and $c \times d$ respectively, is defined as follows:

$$L \otimes M \triangleq \begin{array}{|c|c|c|c|} \hline L_{11}M & L_{12}M & \ldots & L_{1b}M \\ \hline L_{21}M & L_{22}M & \ldots & L_{1b}M \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline L_{a1}M & L_{a2}M & \ldots & L_{ba}M \\ \hline \end{array},$$

where the resulting matrix will have dimensions $ac \times bd$. We, therefore, denote the product code $\mathcal{C}$ by:

$$\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}'.$$

It is worth noting that we change the order of codes in the product code operation notation above as compared with the definition of the Kronecker product of two matrices. We do this for the sake of clarity when describing the decoding algorithm in this thesis.

When there is no possibility of misunderstanding, we will simply denote the parameters of a product code as $[m, k_B, d_B] \times [n, k_A, d_A]$, meaning that for the product code in question, the constituent codes for the columns and the rows have parameters $[m, k_B, d_B]$ and $[n, k_A, d_A]$ respectively.

A codeword $\boldsymbol{c}$ in the product code can either be generated by multiplying a $k_A k_B$ long binary vector with the generator matrix for $\mathcal{C}$ or by using the following equation:

$$\boldsymbol{c} = G_B^T \boldsymbol{u} G_A,$$

where $\boldsymbol{u}$ is a $k_B \times k_A$ binary matrix and G_B^T is the transpose of the matrix G_B. The codeword $\boldsymbol{c}$ will then be an $m \times n$ binary matrix.

The minimum distance of the resulting product code will also be much larger than the constituent codes $\mathcal{A}'$ and $\mathcal{B}'$. However, the fractional minimum distance of the product code will be much smaller than the fractional minimum distance of both the constituent codes as will be shown. Let δ_A and δ_B be the fractional minimum distances of the codes $\mathcal{A}'$ and $\mathcal{B}'$, respectively, defined as follows:

$$\begin{aligned} \delta_A &\triangleq \frac{d_A}{n}, \\ \delta_B &\triangleq \frac{d_B}{m}, \\ \delta_C &\triangleq \frac{d_A d_B}{mn}. \end{aligned}$$

Clearly, the following is correct:

$$\delta_C = \delta_A \delta_B < \delta_A, \delta_B.$$

This decrease in the fractional minimum distance makes these codes less interesting in classical coding theory. In classical coding theory, great interest and effort is put into finding long codes with large fractional minimum distance. There are many other constructions that combine two or more simple codes that result with codes of lengths comparable to product codes but with much larger fractional minimum distance. An example of such codes is Justesen codes, see [37] or [8, pp.306-315]

2.2 Qualities of Product Codes

As shown in Section 2.1, the minimum distance of product codes is small in comparison with the minimum distance of optimal codes of similar lengths and rates. However, the minimum distance is a good measure of the error-correcting capability of a code when the number of errors is less than half the minimum distance of the code. If the number of errors exceeds half the minimum distance of the code, then, the error-correcting potential of the code is related, in the case of linear codes, to the weight distribution of the code, i.e., the number of codewords with a certain Hamming weight for all possible weights. If the number of errors was slightly greater than half the minimum distance of the code then, the error probability will be small if the number of codewords with small Hamming weights is small and vice versa if the number of codewords with small Hamming weights was large. The following example compares the weight distributions of a product code with another code:

Example 2.1 Let the constituent code of both the rows and the columns of the product code $\mathcal{C}$ be the $[8,4,4]$ Reed-Muller code. The parameters of $\mathcal{C}$ are $[64,16,16]$ and its weight distribution is:

$$\{(0,1),(16,196),(24,4704),(28,10752),(32,34230),(36,10752),(40,4704),\\ (48,196),(64,1)\},$$

where the first entry in each member of the set is the Hamming weight and the second entry is the number of codewords in the code that have this Hamming weight. The number of codewords with Hamming weight equal to or less than 16 is 197 which is $3.01 \cdot 10^{-3}$ of the total number of codewords. On the other hand, the number of codewords with Hamming weight equal to or less than 24 is 4901 which is $7.48 \cdot 10^{-2}$ of the total number of codewords. We compare this code with the $[64,16,24]$ extended BCH code $\mathcal{A}'$. The weight distribution of $\mathcal{A}'$ is:

$$\{(0,1),(24,5040),(28,12544),(32,30366),(36,12544),(40,5040),(64,1)\}.$$

The number of codewords with Hamming weight equal to or less than 24 is 5041 which is $7.69 \cdot 10^{-2}$ of the total number of codewords. This means that when the number of errors is equal to 12, the error-correcting capability of the code $\mathcal{C}$ might be slightly better than that for the code $\mathcal{A}'$.

Many other examples can be given showing the same characteristics of the weight distribution of product codes in comparison to other binary codes. However, a general statement about the weight distribution of product codes is very hard and requires extensive studies, [38].
The covering radius, ρ, for any code, is defined as the smallest integer such that all vectors in the containing space are within Hamming distance, ρ, of some codeword. Estimating the covering radius of codes is very hard when the lengths of the codes are large. There exists, however, a very good lower bound on the covering radius of product codes introduced by Cohen *et al.*, [12]. Let the codes $\mathcal{A}'$ and $\mathcal{B}'$ be the constituent codes of the product code $\mathcal{C}$ with lengths n and m respectively, then:

$$\rho(\mathcal{C}) \geq \max(m\rho(\mathcal{A}'), n\rho(\mathcal{B}')). \tag{2.1}$$

The error-correcting potential of product codes can only be achieved if the employed decoder can decode up to the covering radius of the code or at least close to the covering radius of the code. It is easily seen that the covering radius of a product code is much greater than its minimum distance which supports the argument for trying to develop a decoder that decodes beyond half the minimum distance of the code.

In order to illustrate the point regarding the high error correcting capabilities of product codes, we give some examples of error patterns that are correctable using product codes even when the Hamming weights of these error patterns exceed half the minimum distance of the product code under study. The first example we give is the ability of product codes to correct burst errors. Imagine the case where the received message has errors located in a number of rows not exceeding $\lfloor (d_B - 1)/2 \rfloor$ and no errors in all the other rows in the message. Obviously, for every column in the received message, the closest codeword in the code $\mathcal{B}'$ to this column is the corresponding column in the codeword sent by the transmitter. Therefore, without consideration to how many errors are there in these $\lfloor (d_B - 1)/2 \rfloor$ rows, the received message is still correctable. The same argument is true for the case when there is a burst error in the received message that is located in a number of columns not exceeding $\lfloor (d_A - 1)/2 \rfloor$ and no errors in all the other rows in the message. For every row in the received message, the closest codeword in the code $\mathcal{A}'$ to this row is the corresponding row in the codeword sent by the transmitter.

In Chapter 3 the error correction capability of product codes is discussed even further and more examples of correctable error patterns are presented and discussed.

2.3 Decoding of Product Codes

Many decoding algorithms for decoding product codes were presented since their introduction by Elias in 1954. The most obvious method of decoding is the one suggested by Elias himself in his original work [2]. In Elias's algorithm, the rows in the received message are decoded using a decoder for the code $\mathcal{A}'$ that decodes up to half the minimum distance of $\mathcal{A}'$. The columns of the resultant matrix are then decoded using a decoder for the code $\mathcal{B}'$ that decodes up to half the minimum distance of $\mathcal{B}'$. It can easily be shown that such a decoder can correct only up $\lfloor (d_A d_B)/4 \rfloor$, see Elias [2] and Ericson [39].

We start by presenting the system model used in the thesis and then follow by presenting what we consider the most important and famous decoding algorithms that were suggested for decoding product codes.

2.3.1 System Model

We first describe and define the system that we are investigating in the thesis. This system will be the platform for comparing different decoding algorithms both in performance and complexity. In the thesis we will only consider linear binary product codes. The algorithms and the analytical results, however, are easily extended to non-binary codes, linear or non-linear.

Consider the system shown in Figure 2.2 We assume the channel to be an

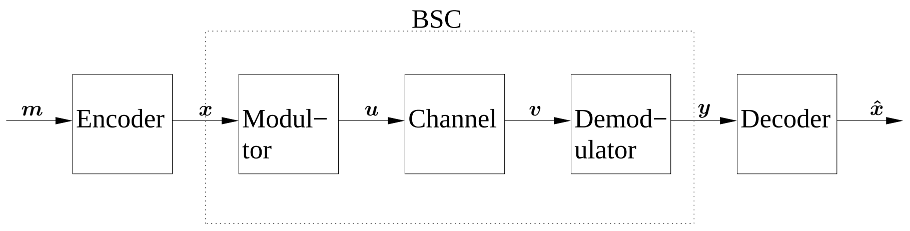


Figure 2.2: Model of the system used in the thesis

AWGN Channel with double sided power spectral density of the noise equal to $N_0/2$. In our analysis, we consider both soft decision decoding and hard decision decoding. In hard decision decoding, the channel will be equivalent to a Binary Symmetrical Channel (BSC) with transition probability, p, which is related to the used modulation as shown in Figure 2.2. Choosing the channel to be additive and memoryless is a way to simplify the model and make it easier for analysis. We use a very simplified model because our aim is to verify the correctness and investigate the potential of the new decoding algorithms proposed in the thesis. As seen in the figure, the encoder receives a message $\boldsymbol{m}$ from the source or the sender. In the case of binary product codes, $\boldsymbol{m}$ can be considered to be a binary array of dimensions $k_B \times k_A$. However, any other message space can be used and the only limitation is that there is a one to one mapping, bijection, between the messages in the message space and the codewords used in the code space. Since there is a one to one mapping between the codewords and the messages, it is always possible to find an estimation of the sent message as long as the decoder can produce some estimation $\hat{\boldsymbol{x}}$ of the codeword.

The encoder encodes $\boldsymbol{m}$ to a codeword $\boldsymbol{x}$ where in the case of binary product codes, this codeword can be considered to be a binary array of dimensions $m \times n$. The modulator modulates each binary symbol in the codeword to the Euclidean space using a certain mapping $\mathcal{M}$, related to the used modulation. For coherent BPSK modulation the mapping $\mathcal{M}$ is as follows:

$$\begin{array}{rrcl} \mathcal{M}: & \{0,1\} & \mapsto & \{+1,-1\} \\ & 0 & \to & +1 \\ & 1 & \to & -1 \end{array} . \tag{2.2}$$

We write:

$$\boldsymbol{u} = \mathcal{M}(\boldsymbol{x}), \tag{2.3}$$

to denote that the symbols of the codeword $\boldsymbol{x}$ are modulated one by one using the mapping shown in 2.4. The output from the modulator is an $m \times n$ real matrix $\boldsymbol{u}$ of $+\sqrt{E_c}$'s and $-\sqrt{E_c}$'s with E_c representing the average energy of the coded bit

$$E_c = R_C E_b,$$

where E_b is the average energy per uncoded information bit and R is the rate of the code. The channel adds an error matrix, $\boldsymbol{e}$ to the codeword $\boldsymbol{x}$ as follows:

$$\boldsymbol{v} = \boldsymbol{e} + \boldsymbol{u},$$

with the elements of $\boldsymbol{e}$ are i.i.d. Gaussian variables with zero mean and variance $N_0/2$. In the case of BSC, the error matrix $\boldsymbol{e}$ is a binary matrix. In this case, the demodulator demodulates each symbol $\boldsymbol{v}_{ij}$ in the received matrix, using the following rule:

$$\boldsymbol{y}_{ij} = \begin{cases} 0, & \text{if} \qquad \boldsymbol{v}_{ij} \geq 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.4}$$

The matrix $\boldsymbol{y}$ is then decoded to the binary matrix $\hat{\boldsymbol{x}}$ using some decoder for product codes.

For soft decision decoding, the demodulator and the channel decoder cooperate. In this case, the soft received vector $\boldsymbol{v}$ is used directly by the channel decoder. Each member in the matrix $\boldsymbol{v}$ can be written as:

$$\boldsymbol{v}_{i,j} = \mathcal{M}(\boldsymbol{c}_{i,j}) + \boldsymbol{e}_{i,j}, \quad \forall i \in \{1, \ldots, m\},\ j \in \{1, \ldots, n\}. \tag{2.5}$$

where $\mathcal{M}$ is the modulation function given in (2.2). In matrix form, it can be written as:

$$\boldsymbol{v} = \mathcal{M}(\boldsymbol{c}) + \boldsymbol{e} \tag{2.6}$$

If the energy of each coded bit was equal to E_c, each element in $\boldsymbol{v}$ can be written as follows, see [40]:

$$\boldsymbol{v}_{i,j} = \pm\sqrt{E_c} + \boldsymbol{e}_{i,j}, \tag{2.7}$$

where the $\pm$ signs are chosen according to the value of $\boldsymbol{c}_{i,j}$. When a hard decoder is used in a AWGN channel and if coherent BPSK is used, the transition probability of the BSC is given by, [41, p. 500] [42, p. 161]:

$$p = \mathrm{Q}(\sqrt{\frac{2R_C E_b}{N_0}}), \tag{2.8}$$

where R_C is the rate of the code used and Q is defined as, [41, pp. 150-151]:

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} \exp^{-\frac{t^2}{2}dt}. \tag{2.9}$$

The squared Euclidean distance between two sequences, $\boldsymbol{v}$ and $\boldsymbol{w}$ of length n, in the $\mathbb{R}^n$ Euclidean space, is given as follows:

$$d_E^2(\boldsymbol{v}, \boldsymbol{w}) \triangleq \frac{\sum_{i=1}^{n}(v_i - w_i)^2}{E_c}. \tag{2.10}$$

In some publications, the definition in (2.10) is called the *normalized* Euclidean distance. Since we never use any *non-normalized* form of the Euclidean distance in this thesis we will, if there is no possibility for confusion, refer to it simply as the Euclidean distance. A soft decoder is capable of utilizing the information about the reliability of the symbols in the received sequence in order to return an estimation of the sent codeword that is closer to the received message than that returned by the hard decoder.

A maximum likelihood decoder returns the codeword that has the greatest probability of being sent given the received message. Formally, for a received message $\boldsymbol{v}$, the ML estimation, $\hat{\boldsymbol{x}}_{ML}$ of this received message is a codeword in the code $\mathcal{C}$ such that for any other codeword $\boldsymbol{x'} \in \mathcal{C}$ the following is true:

$$P(\boldsymbol{x'}|\boldsymbol{v}) < P(\hat{\boldsymbol{x}}_{ML}|\boldsymbol{v}),$$

where $P(\cdot|\cdot)$ is the conditional probability. In memoryless Euclidean channels, the ML solution coincides with the codeword that has the least Euclidean distance between its modulated image and the received sequence, i.e.,:

$$d_E^2(\mathcal{M}(\boldsymbol{x'}), \boldsymbol{v}) \;\geq\; d_E^2(\mathcal{M}(\bar{\boldsymbol{x}}), \boldsymbol{v}).$$

In soft decoding a certain received sequence, we say that one received symbol is more reliable than another symbol in the same sequence if the squared Euclidean distance between the received symbol and its estimate is smaller than the squared Euclidean distance of the second symbol and its corresponding estimate. This definition of reliability of the received symbols in the same sequence is important for soft decision decoding of the constituent codes of the product code using Generalized Minimum Distance decoding [3] or Chase decoding [21].

In order to evaluate the performance of the codes and decoders used in the system, the channel capacity, see Cover and Thomas [43, pp. 183-223] and Johannesson [44, p. 50], can be used for comparison. where the channel capacity for BSC is:

$$\boldsymbol{C} \triangleq 1 - h(p), \tag{2.11}$$

where p is the transition probability of the channel and h is the binary entropy function defined as:

$$h(x) \triangleq -p \log_2 p - (1-p)\log_2(1-p), \tag{2.12}$$

In certain cases it is good to compare the performance of codes in terms of signal to noise ratio instead of the transition probability. If we assume that the channel used was AWGN channel, the modulation is BPSK and that hard decoding was used for each bit.

The probability of error for each bit will be, as discussed in Section 3.3 and shown in (3.13) which we state here one more time for the sake of clarity:

$$p = \mathrm{Q}\left(\sqrt{\frac{2R_cE_b}{N_0}}\right), \tag{2.13}$$

where R_c is rate of the code and Q is as defined in(2.9),

In the case of band-limited AWGN channels, the rate, R, of the code used is limited from above as follows, [43, p. 250], [44, pp. 208-211]:

$$R \le \boldsymbol{C} \triangleq \frac{1}{2}\log_2\left(1 + \frac{P}{N_0W}\right) \text{ bits per sample,} \tag{2.14}$$

where P is the power of the signal, $N_0/2$ is the power spectral density and W is the bandwidth of the channel. The definition of the channel capacity in (2.14) is sometimes called If we assume that a code of length n and rate R is used and that sending one codeword over the channel requires T seconds, then, the signal power can be written in terms of information bit energy, E_b, as:

$$P = \frac{E_b n R}{T}.$$

Since the receiver needs at least n samples to decode the message and there are at most $2WT$ samples of the signal received in time T, each of which has a noise of variance $N_0/2$. The ratio P/N_0W can be written in terms of the information bit energy to noise ratio E_b/N_0 as follows:

$$\begin{aligned}\frac{P}{N_0W} &= \frac{E_bnR}{N_0TW} \\ &= \frac{2E_bnR}{2N_0TW} \\ &= 2R\frac{E_b}{N_0},\end{aligned} \tag{2.15}$$

where R should be equal to the capacity of the channel in order to obtain equality in (2.14). A more detailed discussion on the channel capacity can be found in [43, p. 250], [44, pp. 208-211] and [40, pp. 380-387,399].

2.3.2 Generalized Minimum Distance Decoding

Decoding product codes up to half the minimum distance is somewhat simple and can be achieved by using a variant of the GMD decoder introduced by Forney, see [3]. A GMD decoder was first suggested by Forney as a method of decoding binary block codes in a way that makes use of the soft information coming from the channel while still using an algebraic decoder that can only use the hard interpretation, i.e., zero or one for each symbol, of the symbols from coming from the channel. The simplest definition of the term *Generalized Distance*, d_{GD}, between two sequences, is the sum of the distances between the symbols in the two sequences without consideration to what distance metric is used between these symbols. For example, if the distance between the symbols was taken to be the Hamming distance, then, the generalized distance is the Hamming distance between the two sequences. Similarly, if the distance between the symbols is Euclidean distance, then, the generalized distance between the two sequences will be the summation of the absolute Euclidean distances of the corresponding symbols in the two vectors, and so on.

The term *Generalized Minimum Distance* refers to the minimum correctable generalized distance between a vector in the Euclidean space and a codeword in the code used in transmission using the algorithm presented by Forney. For a code with minimum Hamming distance equal to d, the Generalized Minimum Distance is proportional to d. It is also possible to use the square Euclidean distance instead of the Generalized Distance as a metric when performing GMD decoding algorithm with exactly the same results.

It was later shown by Forney [4], Blokh and Zyablov [9] and Zyablov and Zinoviev [7], that the GMD decoding algorithm can be used for a whole class of codes called concatenated codes including product codes.

GMD decoding of product codes assumes that there exists separate decoders for both the row code and the column code that can correct all errors up to half the minimum distance of the respective code. As a first step the GMD decoder decodes each row in the received matrix up to half the minimum distance of the row code and stores the result. Then, each column of the resultant matrix is decoded up to half the minimum distance for the column code. The GMD decoder, then, starts to successively erase the least reliable rows two by two as long as the number of erased rows is less than the minimum distance of the column code. The columns are re-decoded each time two rows are erased and the result is stored. In the end the GMD decoder chooses from the different results, the codeword that is closest to the received matrix. It can be shown that GMD decoding can correct all error patterns of Hamming weight less than half the minimum distance of the code, see Forney [3], Blokh and Zyablov [5] and Ericson [39]. However, there the GMD decoding algorithm can decode many other patterns and some burst errors with Hamming weight that is greater than half the minimum distance of the product code.

The GMD decoder of product codes can be made to take into consideration the soft information of the symbols coming from the channel. This is simply done by decoding the rows using a GMD decoder for the rows instead of a decoder that corrects up to half the minimum distance of the row code.

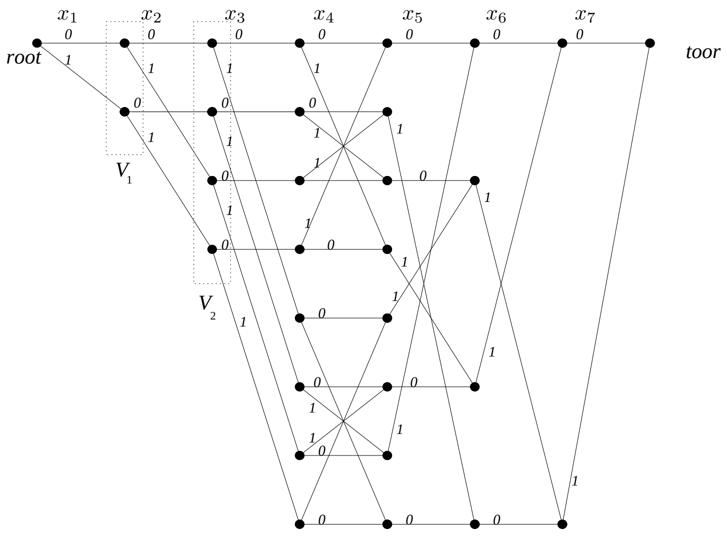
2.3.3 Maximum Likelihood Decoding

As shown in 2.3.1, the ML solution in memoryless Euclidean channels is the modulated image of the codeword that is closest to the received message. One simple, and obvious, method to obtain the ML solution would be to compare all the distances between the codewords in the code and the received message and pick the codeword that is closest to the received message. Needless to say, such a method is very time consuming and is impractical except in certain cases of very short codes. Viterbi, [36], introduced a decoding algorithm for decoding convolutional codes that makes ML Decoding practically feasible. Later, Forney, [35], showed that the Viterbi algorithm is actually a dynamic algorithm for finding the shortest path between the first node and the last node in a certain type of graphs called the *trellis* of the code.

A trellis T representing a code $\mathcal{U}$ of length n is a graph composed of a finite set of vertices, V, a finite set of labeled edges, E and a set of labels L where the label set is the alphabet of the code. The vertices can be partitioned into disjoint sets, $V_0, V_1, \ldots, V_n$, where we call i the *time*. The trellis is such that for each subset V_i there are edges connecting the vertices in V_i with the vertices in V_{i-1} and connecting the vertices in V_i with the vertices in V_{i+1}, and no other edges exist. I.e., we can find paths of labeled edges connected by vertices starting from the first set of vertices V_0 and ending in the last set of vertices V_n. For such a trellis, each path, sequence of edges, of length n going through the vertices is a codeword in the code $\mathcal{U}$, see Vardy [19].

In 1974, Bahl, Cocke, Jelinek and Raviv [18], showed that linear block codes can also be represented by a trellis and presented a method for constructing it. The construction given by Bahl *et al.* was later shown by McEliece, [45] to be *minimal*, where we mean by minimal that when comparing the minimal trellis T with any other trellis representations, T', of the same code, the number of vertices at each time i is less in T than that in T'. The definition of minimal trellis is important when discussing the subject of decoding complexity.

In order to further illustrate what is meant, we show, as an example, the trellis representation of the $[7, 4, 3]$ Hamming code in Figure 2.3. The method used for



$[7, 4, 3]$ Hamming code

Figure 2.3: Trellis of the $[7, 4, 3]$ Hamming code.

constructing the minimal trellis above, is the same method introduced by Bahl *et al.* mentioned above. By observing the trellis of the $[7, 4, 3]$ Hamming code, it is clear that most of the attractive features in Viterbi decoding on the trellises of convolutional codes are missing in the case of block codes. For example, it can be seen that the number of vertices in the trellis of the Hamming code are different for each time. Also, the edge connections between the vertices are very complicated and different for each time. This is true for almost all classes of non-trivial and famous block codes of interest, e.g., BCH codes, alternant codes, Reed-Muller codes and many more. On the other hand, the trellises of convolutional codes are very simple meaning that the number of states in a subset of vertices at a certain time is equal to the number of vertices in a subset at *almost* any other time in the trellis. Also the connection of edges between the vertices of a subset at a certain time to the vertices of the subset in the previous time, is identical to *almost* any other connection of edges at any other time in the trellis. However, in the trellises of convolutional codes, the first few subsets of vertices in the trellis and the edges between them are different in number and form from those in the rest of the trellis. Also, The last few subsets of vertices in the trellis and the edges between them are different in number and form from those in the rest of the trellis. A much more detailed explanation of trellises of convolutional codes can be found in [46].

It can be shown that the number of operations needed for performing Viterbi decoding on a trellis T with vertices V and edges E is equal to:

$$2|E| - |V| + 1,$$

see Vardy [19]. It can also be shown that the number of edges in a trellis is closely related to the number of vertices. Therefore, taking the number of vertices in the trellis as the complexity measure between trellises is appropriate. An upper bound on the logarithm of the maximum number of vertices at any time in a trellis of an $[n, k]$ binary linear code is the famous Wolf bound [47], which states:

$$\log_2 V_i \leq \min\{k, n-k\}, \quad i \in \{1, 2, \ldots, n\}.$$

In the case of Maximum Distance Separable (MDS) codes, equality is achieved. It was shown by Vardy, [19], that this bound is actually very good, meaning that in most cases, the logarithm of the maximum number of vertices at any time, is very close to the Wolf bound.

It can easily be shown that for a product code with parameters $[m, k_B, d_B] \times [n, k_A, d_A]$, the Wolf bound looks like:

$$\begin{aligned} \log_2 V_i \quad &\leq \quad \min\{k_A k_B, (n-k_A)k_B, k_A(m-k_B), (n-k_A)(m-k_B)\}, \\ & \qquad i \in \{1, 2, \ldots, mn\}. \end{aligned} \tag{2.16}$$

The proof of this claim is done by observing the generator matrix of the product code that is generated by taking the Kronecker product of the generator matrices of the constituent codes that both have a *minimal span* form. For the definition and construction of minimal span form generator matrices we refer to Kschischang and Sorokine [48] and Vardy [19].

There is some work done in the area of investigating trellis constructions of product codes and Viterbi decoding on them. However, the discussion above shows that the complexity of Viterbi decoding on the trellis of product codes is exponentially increasing with the size of the code. Therefore, we believe that Maximum Likelihood Viterbi decoding on the trellis of product code is not practical except in cases of very short codes or product codes with very high, alternatively, very low rate. This belief is shared by many prominent researchers in this area which supports our conviction.

2.3.4 Turbo Decoding

We mentioned in Chapter 1 that Glavieux and Berrou were the ones who introduced turbo decoding in 1993, [49]. The decoding algorithm was designed to iteratively decode a parallel concatenation of two convolutional codes using a Maximum Aposteriori Probability (MAP) soft decoder of the constituent convolutional codes. The decoder introduced by Bahl *et al.* [18], is a modification of the MAP algorithm in such a way that it becomes directly implementable in decoding on a trellis representation of the code. MAP decoding on the trellis utilizes a Viterbi-like stage that perform decoding from the start of the trellis and forward or from the end of the trellis and backward. It was later shown by Wiberg, [50], that this algorithm is a subclass of algorithms that were later called the Forward-Backward algorithm, see Forney [51]. MAP decoding has a complexity comparable to that of Viterbi decoding. It can be shown that it requires a total number of operations that is almost four times that required by Viterbi decoding.

The original form of the MAP decoder dates back to Hartmann and Rudolph [52] and Battail [53]. We will try to give a quick presentation of the straight forward implementation of this algorithm. I.e., our explanation does not include the modification that makes it implementable on trellises. Let a binary code $\mathcal{U}$ with parameters $[n, k, d]$ be used for decoding. Let the received sequence be $\boldsymbol{y}$ and the result from the decoder be $\hat{\boldsymbol{x}}$. The MAP decoder returns real value for each symbol $\hat{\boldsymbol{x}}_i$ that can be evaluated as follows:

$$\mathcal{L}(\hat{\boldsymbol{x}}_i) = \ln \frac{\displaystyle\sum_{\boldsymbol{x}\in\mathcal{U},\boldsymbol{x}_i=0} \prod_{l=1}^{n} P(\boldsymbol{y}_l|\boldsymbol{x}_l)}{\displaystyle\sum_{\boldsymbol{x}\in\mathcal{U},\boldsymbol{x}_i=1} \prod_{l=1}^{n} P(\boldsymbol{y}_l|\boldsymbol{x}_l)}, \tag{2.17}$$

where $\mathcal{L}$ is the log-likelihood function of the symbols in $\hat{\boldsymbol{x}}$. The binary values of the symbols in $\hat{\boldsymbol{x}}$ are found by setting each symbol $\hat{\boldsymbol{x}}_i$ to zero if its log-likelihood function was greater than zero. Alternatively, the symbol is set to one if its corresponding log-likelihood function was less than zero.

In turbo decoding, the received sequence is MAP decoded on the first constituent code and the real values obtained from MAP decoding are used, without mapping the results to binary symbols, as input for MAP decoding in the next stage to MAP decode on the second constituent code. The procedure is repeated in the following iterations using the real values returned by the MAP decoder in the previous iteration.

The same result for the MAP decoder can be obtained by decoding on the *dual* of the code, [8, p. 26]. Decoding on the dual code is more efficient for decoding codes with very high rate. This is because the number of codewords in the dual code of a high rate code is much less than that for the original code. The form of MAP decoding on the dual code can be obtained by performing the discrete Fourier transform or the Hadamard transform on (2.17). This was first shown by Rudolph and Hartmann [52] and later by Battail *et al.* [53]. We prefer, however, to present it in the form shown be Hagenauer [17]. For the AWGN channel, the soft value for each symbol returned by MAP decoding on the dual code can be given as follows:

$$\mathcal{L}(\hat{\boldsymbol{x}}_i) = \ln \frac{P(\boldsymbol{y}_i|\boldsymbol{x}_i = +1)}{P(\boldsymbol{y}_i|\boldsymbol{x}_i = -1)} + \ln \frac{\displaystyle\sum_{\boldsymbol{x}\in\mathcal{U}^\perp} \prod_{l=1,l\neq i}^{n} \tanh(\frac{\mathcal{L}_{ch}\boldsymbol{y}_l}{2})^{\boldsymbol{x}_l}}{\displaystyle\sum_{\boldsymbol{x}\in\mathcal{U}^\perp} (-1)^{\boldsymbol{x}_i} \prod_{l=1,l\neq i}^{n} \tanh(\frac{\mathcal{L}_{ch}\boldsymbol{y}_l}{2})^{\boldsymbol{x}_l}}, \tag{2.18}$$

where $\mathcal{L}_{ch}$ is a constant that depends on the signal to noise ratio of the channel. Hagenauer *et al.* [17] described and showed that it is possible to use turbo decoding on product codes using MAP decoders on the constituent codes. They also gave quite an extensive explanation and comparison between using MAP decoding on the trellis of the constituent codes, using (2.17) or using (2.18).

It should be noted that, indeed, in the case of convolutional codes, the MAP decoder on the trellis of the code as introduced by Bahl *et al.* [18], would mean great decrease in complexity as compared with (2.17). This is due to the fact that in convolutional codes, the maximum number of vertices at any time in the trellis is much less than the total number of codewords in the convolutional code. However, in block codes, the maximum number of vertices at any time in the trellis of the code will be, using Wolf's bound, of the same order of the total number of codewords of either the original code or the dual code. Therefore, MAP decoding on the trellis of block codes will not necessarily result in a decrease in decoding complexity. This is a major obstacle toward using good block codes instead of convolutional codes in systems that incorporate turbo decoding except in very limited cases when the size of the code is very small or when the constituent codes are very simple.

In order to solve the problem with complexity of MAP decoding the constituent codes, many suggestions were made. Hagenauer presented a soft output algorithm called the Soft Output Viterbi Algorithm (SOVA), that approximates MAP decoding, Lucas [54] presented an iterative algorithm that approximates MAP decoding on the dual code were only the minimum weight codewords of the dual code are used. Pyndiah [20], suggested another approximation for MAP decoding of the constituent codes where a Chase II decoder is used to obtain a list of codewords that are closest to the received sequence and then (2.17) is implemented using only this subset of codewords instead of the whole code. In a Chase II decoder, the f least reliable bits in the received sequence are identified and then, all 2^f error patterns that have one's in these f bits and zeros elsewhere, are added to the received sequence and decoded using a BMD decoder. In the original work of Chase [21], f is chosen to be equal to $\lfloor (d-1)/2 \rfloor$, where d is the minimum distance of the code.

Pyndiah, however, uses another variant were f is chosen to be much larger than that suggested by Chase. Choosing a larger f is done in order to obtain a much larger set of codewords to be used in the approximated MAP decoder.

2.4 Discussion

From the previous section we see that the decoding algorithms of product codes can be split into two categories. The first category is algorithms with low complexity but with low performance, e.g., GMD decoding of product codes. The other category is algorithms with very high complexity and very high performance, e.g., Viterbi decoding on the trellis of product codes. Even though turbo decoding of product codes is much less complex than Viterbi decoding, we still consider turbo decoding of product codes to be, in general, a high complexity algorithm. Our reasoning about the complexity of turbo decoding of product codes goes as follows: Let the product code used have dimensions equal to $[m, k_B, d_B] \times [n, k_A, d_A]$. We present our idea when the product code has low rate. The same argument can be applied to high rate product codes after slight modifications. Each row requires that (2.17) be performed for each symbol in the row. Similarly, each column requires that (2.17) be performed for each symbol in column. This means that the MAP equation will be performed $2mn$ times for the whole matrix. Since (2.17) requires 2^{k_A} additions for the rows and 2^{k_B} for the columns, which means that a total of $m2^{k_A} + n2^{k_B}$ operations are needed for one iteration stage in turbo decoding. This is a great decrease in complexity compared with ML decoding, since, ML decoding requires a number of operations of order $2^{k_A k_B}$. This means that the complexity of ML decoding is exponentially increasing with the size of the product code while the complexity of turbo decoding is exponentially increasing with the size of the constituent codes. However, using very large codes with turbo decoding is a requirement for gaining some advantages in terms of low bit error probability. Therefore, although the complexity of turbo decoding of product codes is much less than that of ML decoding, we still consider it a high complexity algorithm.

The contributions into decreasing the complexity of MAP decoding of block codes truly decrease the total complexity of turbo decoding. However, how much does the decrease in total complexity affects the performance is not investigated and uncertain. It is quite possible that the number of iterations in turbo decoding is increased to compensate for using suboptimal MAP decoder for the constituent codes, which is undesired. In other cases, the degradation of performance becomes so great that makes the idea of using a turbo decoder unnecessary.

We believe that the problems associated with turbo decoding of product codes are inherent and cannot be solved by trimming the different processes in turbo decoding or by introducing some ad hoc modifications of certain parts of the turbo decoder. In this thesis, we state the question, is it possible to design a decoding algorithm for product codes that has as low complexity as possible for a given level of performance? It is quite clear that this question is a bit vague since it is a question of values and qualities such as the terms, "performance" and "complexity" which have to be defined. In this thesis we present decoding algorithms for product codes and analyze their performance and complexity using the definitions given in the thesis. We try to show that it is possible to perform decoding at a complexity much lower than that for other decoding algorithms with almost the same performance. It is true that the qualitative analysis of these algorithms and the comparison with other decoding algorithms might be different if we use other definitions of performance and complexity. However, we believe that the comparative results will be similar even when using other definitions of "performance" and "complexity". This is due to the fact that we chose very basic definitions of these two terms as will be shown. Most of the alternative definitions of these terms are dependent or related to the definitions we give in the thesis.

The basic idea about the new algorithms proposed in this thesis is to avoid MAP decoding of the constituent codes altogether. The algorithms presented in the thesis rely on methods for list decoding the received message and intelligent sorting of the different candidates in a way that the number of unnecessary operations is kept as small as possible. The suboptimal, iterative algorithm presented in Chapter 4 goes even further in decreasing the complexity by fixing the complexity at each iteration to maximum acceptable value. The iterative algorithm, however, incorporates a method that combines the results of decoding the rows and the columns in a way that they help each other toward the correct solution as will be shown in Chapter 4.

The thesis also give some important theoretical results regarding decoding product codes in the form of upper bounds on the complexity of decoding given the performance. Alternatively, we also present in the thesis upper bounds on the performance of decoding given the maximum complexity allowed. These bounds will be shown to be practical bounds that can be used when designing a communication system that incorporates product codes.

This page intentionally contains only this sentence.

Chapter 3

The Basic Decoding Algorithm

Let us assume that a certain code was used for data transmission on a certain channel. It is quite clear that Maximum Likelihood decoding can, by definition, be achieved by comparing the distances between the received vector and all the codewords in the code and choosing the codeword closest to the received message. This, however, is not practical unless the cardinality of the code is very small. Therefore, what is needed is a method for excluding from the comparison set, those codewords that are far from the received message or, equivalently, excluding error patterns with low probability, thus keeping the average number of comparisons to a minimum. In this chapter, we show that the problem of excluding error patterns from the check set is equivalent to the problem of designing efficient algorithms for sorting the weights of these error patterns, a problem that one is usually confronted with in the design of computer algorithms. To solve this problem, the algorithm should take advantage of certain features in the structure of product codes. A new definition of product codes that emphasizes these features will first be made and the decoding algorithm will be tailored to it accordingly. In this chapter, we restrict ourselves to binary linear product codes and analyze their performance when used with Binary Symmetrical Channel. Further generalizations of the algorithm are discussed very briefly.

3.1 Product codes and their decoding

3.1.1 Alternative representation of product codes

We give an alternative description of product codes as an intersection of two simpler codes. This description allows us to devise a ML decoding algorithm for product codes later on. Let $\mathcal{A}'$ be an (n, N, d_A) binary code and let $\mathcal{A}$, be the code that can be represented by the set of all $m \times n$ matrices such that each *row* in these matrices is an element of the code $\mathcal{A}'$. In a similar manner, let $\mathcal{B}'$ be an (m, M, d_B) binary code and let $\mathcal{B}$, be the code that can be represented by the set of all $m \times n$ matrices such that each *column* in these matrices is an element of the code $\mathcal{B}'$. The code $\mathcal{A}$ can also be defined as the m-fold Cartesian product of the code $\mathcal{A}'$ with itself, and the code $\mathcal{B}$ can be defined as the n-fold Cartesian product of the code $\mathcal{B}'$ with itself. Let the code $\mathcal{C}$ be the product code obtained from the codes $\mathcal{A}'$ and $\mathcal{B}'$. It is quite clear that $\mathcal{C}$ can also be described as the intersection of the two codes $\mathcal{A}$ and $\mathcal{B}$ as given below:

$$\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}' = \mathcal{A} \cap \mathcal{B}. \tag{3.1}$$

If the two codes $\mathcal{A}'$ and $\mathcal{B}'$ were chosen to be simple codes in terms of decoding complexity, the codes $\mathcal{A}$ and $\mathcal{B}$ will also have low decoding complexity. We will use this fact in designing the algorithm.

3.1.2 A maximum likelihood decoder for product codes

As shown in Chapter 2, Viterbi decoding on the trellis of product codes is very complex. We present a maximum likelihood decoder which is the main idea behind this thesis. We first give a definition for the list decoder, ξ_e for a code $\mathcal{V} \subset \mathbb{F}_2^n$. Let $\boldsymbol{u} \in \mathbb{F}_2^n$ be the received vector, then, we say that $\xi_e(\boldsymbol{u}, \mathcal{V})$ is the list of all codewords in $\mathcal{V}$ with Hamming distance from $\boldsymbol{u}$ equal to or less than e, ordered according to their distance from $\boldsymbol{u}$. If e was equal to n, then the result will be all the codewords in the code ordered according to their distance from $\boldsymbol{u}$. We call e the *decoding radius* of the list decoder. Let the product code $\mathcal{C}$ given in (3.1) be used for data transmission on a binary channel and let $\boldsymbol{y}$ and $\hat{\boldsymbol{x}}$ be, respectively, the received matrix and the codeword in $\mathcal{C}$ that is closest to $\boldsymbol{y}$. In the case of BSC, the covering radius $\rho(\mathcal{C})$ is equal to the maximum Hamming weight of all error patterns that are uniquely corrected using a ML decoder. Let $\boldsymbol{A}$ be a list of all the codewords in $\mathcal{A}$ with Hamming distances from $\boldsymbol{y}$ less than or equal to $\rho(\mathcal{C})$ listed in an ascending order using their Hamming distances d_H from $\boldsymbol{y}$. When list decoding beyond the covering radius of a code, the problem of ties occur. In order to avoid this problem, certain rules should be followed to uniquely decide which one of two or more vectors is closer to $\boldsymbol{y}$, even though they have the same Hamming distance from $\boldsymbol{y}$. It is easy to see that $\hat{\boldsymbol{x}}$ will be a member of $\boldsymbol{A}$ since $\hat{\boldsymbol{x}}$ is an element in $\mathcal{A}$. In a similar manner, let $\boldsymbol{B}$ be a list of all the codewords in $\mathcal{B}$ with Hamming distance from $\boldsymbol{y}$ less than the covering radius of $\mathcal{C}$ listed in an ascending order using their Hamming distances from $\boldsymbol{y}$ and the same set of rules for solving ties used before. The ML estimation, the codeword $\hat{\boldsymbol{x}}$, will be a member of this list also. It can also be proved that the codeword $\hat{\boldsymbol{x}}$ will be the first member of the list $\boldsymbol{A}$ that is also a member of the code $\mathcal{B}$. This is true because, otherwise, there has to exist a codeword in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than $\hat{\boldsymbol{x}}$ which contradicts the assumption that $\hat{\boldsymbol{x}}$ is the codeword in $\mathcal{C}$ that is closest to $\boldsymbol{y}$. Therefore, the decoding can commence by beginning from the top of the list $\boldsymbol{A}$, picking one word at a time, and checking to see if it is also a codeword in $\mathcal{B}$. If it is, the algorithm stops and returns it as the correct answer, otherwise, it picks the next word in $\boldsymbol{A}$, which is even further from $\boldsymbol{y}$ and so on. An alternative variant would be to jump between the two lists, looking for a valid codeword in both the lists. This is illustrated by Figure 3.1, where ML decoding can be performed by checking each member of $\boldsymbol{A}$ beginning from the first, to see if it also was a member of the code $\mathcal{B}$. Alternatively, one can jump between the two lists, checking the members at increasing distance. The algorithm above can be

$$\boldsymbol{A} \triangleq \begin{array}{|c|}\hline \boldsymbol{a}^1 \\ \hline \boldsymbol{a}^2 \\ \hline \vdots \\ \hline \hat{\boldsymbol{x}} \\ \hline \vdots \\ \hline \boldsymbol{a}^f \\ \hline \end{array}, \qquad \boldsymbol{B} \triangleq \begin{array}{|c|}\hline \boldsymbol{b}^1 \\ \hline \boldsymbol{b}^2 \\ \hline \vdots \\ \hline \hat{\boldsymbol{x}} \\ \hline \vdots \\ \hline \boldsymbol{b}^g \\ \hline \end{array}$$

$$d_H(\boldsymbol{a}^i, \boldsymbol{y}) \le d_H(\boldsymbol{a}^j, \boldsymbol{y}), \forall i < j, \qquad d_H(\boldsymbol{b}^i, \boldsymbol{y}) \le d_H(\boldsymbol{b}^j, \boldsymbol{y}), \forall i < j$$

Figure 3.1: List decoding of the codes $\mathcal{A}$ and $\mathcal{B}$.

applied to any block code, since any block code can be described as an intersection of two other codes. It is obvious, however, that unless one of the two codes or both have very simple list decoding algorithms, the algorithm will not be feasible. In the case of product codes, however, list decoding on $\mathcal{A}$ or $\mathcal{B}$ can be done by list decoding the rows and columns respectively as follows: For the received message $\boldsymbol{y}$, we use a complete list decoder $\xi_n(\boldsymbol{y}_{i,\cdot}, \mathcal{A}')$ for each of the rows i of $\boldsymbol{y}$. Thus, we can generate a list of all the codewords in $\mathcal{A}$ sorted according to their Hamming distance from $\boldsymbol{y}$. Even though complete list decoding of the rows can result in a complete list decoder for $\mathcal{A}$ as discussed above, this is neither practical nor is it simple to analyze. A more practical method would be to use a list decoder for the rows with a limited decoding radius to generate a short list using the candidate codewords for each row. If after checking this short list, a valid codeword in the product code is not found, the decoding radius of the list decoder for the rows is increased, thus resulting in an even longer list of matrices that can be checked.

3.2 Sorting and decoding

From the previous section we understand that the problem of decoding product codes is transformed into a problem of list decoding one of the constituent codes, or both, in addition to a task of sorting the resultant list of codewords according to their distance from the received vector. Therefore, in order to decrease the complexity of the decoder, efficient sorting algorithms that are adapted to this kind of problem and low complexity list decoders for the constituent codes should be implemented. We concentrate on the problem of generating only the l first elements of the ordered list $\boldsymbol{A}$, where l is an integer, and try to analyze the performance and complexity of the decoder when it is limited to using only those elements. Almost all the ideas in this section can be found in elementary books on computer algorithms. However, we present it in a manner that best suits the problem of decoding product codes in terms of complexity.

In Figure 3.2 below, we show an example of how the algorithm should find a list of only two matrices that are closest to the received matrix $\boldsymbol{y}$. The search for a list of matrices closest to the received matrix is similar to looking through a search tree. The sorting algorithm should look through all the l^m different combinations of candidates for each row for the closest l matrices to $\boldsymbol{y}$ that can be generated from these candidates. The value of l is equal to 2 in this example. The list decoder for

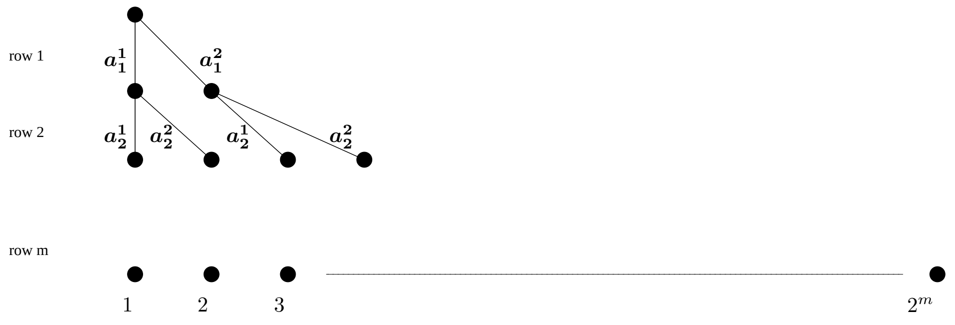


Figure 3.2: Search tree for finding a list of matrices.

the rows, in this example, returns for row i, two candidates, $\boldsymbol{a_i^1}$ and $\boldsymbol{a_i^2}$, where $\boldsymbol{a_i^1}$ is closer to the corresponding row in $\boldsymbol{y}$ than $\boldsymbol{a_i^2}$. By using one of the two candidates for each row, we can generate 2^m different matrices. The distance of each one of these matrices to the received matrix $\boldsymbol{y}$ is equal to the sum of distances of each row to the corresponding row in $\boldsymbol{y}$. Since the candidates for each row are ordered according to their distance from the corresponding row in $\boldsymbol{y}$, it would be easier to define a *weight*[1], w, with respect to another vector and associated with each candidate and thus we can say that:

$$w(\boldsymbol{a_i^j}) \stackrel{\triangle}{=} D(\boldsymbol{a_i^j}, \boldsymbol{y}_{i,\cdot}),$$

and that:

$$w(\boldsymbol{a_i^j}) \leq w(\boldsymbol{a_i^k}), \quad j < k.$$

The function D is the metric used in the channel. E.g., it can be the Hamming distance, d_H, in BSC or the squared Euclidean distance, d_E^2, for Euclidean channels. If we consider a matrix $\boldsymbol{a}$ that is constructed in the following way:

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a_1^{j_1}} \\ \boldsymbol{a_2^{j_2}} \\ \vdots \\ \boldsymbol{a_m^{j_m}} \end{bmatrix}$$

I.e., $\boldsymbol{a_i^{j_i}}$ is the i:th row in $\boldsymbol{a}$. The weight will be:

$$w(\boldsymbol{a}) = w(\boldsymbol{a_i^{j_1}}) + w(\boldsymbol{a_2^{j_2}}) + \ldots + w(\boldsymbol{a_m^{j_m}}).$$

As mentioned above, the algorithm is required to return a list of length l of codewords in the code $\mathcal{A}$ that are closest to the received matrix $\boldsymbol{y}$. The problem can be solved by a *Depth First Search*, see Aho et al [55, Chapter 5]. The required result, however, is a list of members instead of a single, closest member. What we show below is such a method. The task of generating the first l elements of $\boldsymbol{A}$ can be separated into two different tasks. The first is to list decode the rows generating m lists of cardinality l each. We call the list of members for each row the list of *candidates* for that row. We assume, for now, that such a list decoder for the rows exists. However, the practical aspect of the list decoder is further discussed in Chapter 6. We also assume that each of the lists returned from the list decoder for the rows is an ordered list according to the Hamming distance of its members to the corresponding row in $\boldsymbol{y}$. Each one of the l^m combinations of candidates for each row will be a member of the list $\boldsymbol{A}$. Therefore, the second problem is to find an efficient sorting algorithm that chooses only l combinations from all l^m different combinations of candidates for each row. The search for the l closest members can be performed in the following way: Beginning from the first and second row, find the l closest combinations of the candidates for the two rows and exclude the rest. The closest combinations of the candidates for the first two rows are the ones that have the least weight, where the weight of each combination is the sum of

[1]In coding theory, the weight usually refers to the Hamming weight, i.e., the number of non-zero positions in the vector. What we mean by the *weight* here is different from that used in coding theory and is only essential to the sorting algorithm we describe later on in this chapter. The procedure of the sorting algorithm depends only on the real values, weights, of the members of the lists to be sorted.

the weights of the candidate for the first row and the weight of the candidate of the second row. The same is done using these combinations and the list of candidates for the third row to find the l closest combinations of the three rows. This is continued for each row using only l closest combinations of the previous rows until reaching the last row. We assume here, that the basic sorting function is to compare and sort two members at each time. Other sorting alternatives where the sorting function compares and sorts more than two members at a time are also possible but will not be considered here. As an example, let us assume that the two candidates for the first row are $\boldsymbol{a_1^1}$ and $\boldsymbol{a_1^2}$ with weights 0 and 5 respectively. The candidates for the second row are $\boldsymbol{a_2^1}$ and $\boldsymbol{a_2^2}$ with weights 3 and 4 respectively. There are 4 combinations of these candidates, namely, $(\boldsymbol{a_1^1}, \boldsymbol{a_2^1})$, $(\boldsymbol{a_1^1}, \boldsymbol{a_2^2})$, $(\boldsymbol{a_1^2}, \boldsymbol{a_2^1})$, and $(\boldsymbol{a_1^2}, \boldsymbol{a_2^2})$, with weights 3, 4, 8 and 9 respectively. The sorting algorithm should pick only the first two and combines them with the candidates of the third row and so on. Thus, an important part of the sorting algorithm is an efficient procedure that operates on two ordered lists of l real numbers, $\boldsymbol{u}$ and $\boldsymbol{v}$, i.e., the weights of the candidates for a row or combination of candidates of previous rows. This procedure should return a list $\boldsymbol{f}$ of l smallest combinations of members of the lists $\boldsymbol{u}$ and $\boldsymbol{v}$. Without loss of generality, we assume that the first elements u_1 and v_1 of the two lists $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively, are equal to zero. It is possible, of course, to generate all l^2 combinations of elements from the two lists, sort them and then choose the l smallest combinations from them. This however is not very efficient since sorting such a list with l^2 members requires at least $O(l^2 \log l)$ comparisons, see for example Aho et al [55, Chapter 3]. We present in Figure 3.3 a function λ that finds the l least combinations between members of two ordered lists $\boldsymbol{a}$ and $\boldsymbol{b}$. This algorithm completes the task with $O(l \log l)$ comparisons and returns an array of three columns and l rows, where the last column contains the l smallest combinations of numbers and the other two columns contain the orders of the elements in $\boldsymbol{u}$ and $\boldsymbol{v}$ that produced this number.

Simply explained, the algorithm adds two components of the two lists at a time, beginning from the smallest members of the two lists, sorting the result in a new list and stops when the size of the new list becomes equal to l. The algorithm makes use of the *stack* $\boldsymbol{g} = \{\boldsymbol{g_1}, \boldsymbol{g_2}, \ldots, \boldsymbol{g_l}\}$, which is simply an array of l cells each of which has three elements. The first two members of each cell in the stack are natural numbers, each of which pointing to an element in $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively. The result of adding these two elements is stored in the third element of the same cell. The cells are then ordered by their last members. In the algorithm below, we make use of a function called, **PushSort**, which is an algorithm for pushing a cell into an already ordered stack in its ordered position. Efficient algorithms that use binary search trees and perform this task are well known in literature and it can be proven that they require at most $\log l$ comparisons. See for example Aho *et al.* [55, Chapters 2 and 3]. The sorting algorithm above is best explained by an example.

Algorithm 3.1 *Function λ*

```
1  Input: Two ordered lists of real numbers $\boldsymbol{u}=(u_1,\dots,u_l)$ and $\boldsymbol{v}=(v_1,\dots,v_l)$
2  Output: $l\times 3$ array $\boldsymbol{f}$. Variables: $i\in\mathbb{N}$, $z\in\mathbb{N}\times\mathbb{N}\times\mathbb{R}$ and stack $\boldsymbol{g}$ of $l$ elements
   such that $\boldsymbol{g}^i\in\mathbb{N}\times\mathbb{N}\times\mathbb{R},\forall i\in\{1,\dots,l\}$
3  Initialize:
4  for $i\leftarrow 1$ to $l$ do
5      $\boldsymbol{g}_{i,1}=i,\quad \boldsymbol{g}_{i,2}=0,\quad \boldsymbol{g}_{i,3}=u_i$
6  end for
7  $i\leftarrow 1$.
8  while $i\leq l$ do
9      $z\leftarrow \boldsymbol{g}_1$                        ▷ Copy the contents of the first cell of
10     $\boldsymbol{g}\leftarrow \boldsymbol{g}\backslash\boldsymbol{g}_1$                       ▷ the stack to z and remove it.
11     if $z_2=0$ then
12         $z_2\leftarrow 1$                      ▷ If the first member was not a valid
13         $z_3\leftarrow \boldsymbol{u}_{z_1}+\boldsymbol{v}_{z_2}$               ▷ combination of two members of u and v,
14         $\boldsymbol{g}\leftarrow$ PushSort$(z,\boldsymbol{g})$           ▷ process it to find a new combination.
15     else
16         $\boldsymbol{f}_i\leftarrow z$                    ▷ If it was a valid combination, then copy
17         $i\leftarrow i+1$                  ▷ it's contents to f, process it to find the
18         $z_2\leftarrow z_2+1$                ▷ next cell.
19         if $z_2\leq l$
20             $z_3\leftarrow \boldsymbol{u}_{z_1}+\boldsymbol{v}_{z_2}$
21             $\boldsymbol{g}\leftarrow$ PushSort$(z,\boldsymbol{g})$
22         end if
23     end if
24 end while
```

Figure 3.3: Algorithm that finds a list of combinations of two lists.

Example 3.1 Consider the two lists of numbers $(0, 2, 3, 5)$ and $(0, 1, 1, 3)$. There will be 2^4 different combinations of members of the two lists, namely, $0 + 0$, $0 + 1$, $0 + 1$, $0 + 3$, $2 + 0$, $2 + 1$, $2 + 1$, $2 + 3$, $3 + 0$, $3 + 1$, $3 + 1$, $3 + 3$, $5 + 0$, $5 + 1$, $5 + 1$ and $5 + 3$. The algorithm should return the 4 smallest combinations without actually creating the whole list of 16 members. Figure 3.4 shows how the components of the stack change at each step. In order to find the resultant list, f, we look at the first member of the stack at each step and check if it was a valid component, i.e., it is made up of combining two members from the input lists. If it was valid, it is added to f otherwise we continue to the next step. The stack f will in the end be $\{(1, 1, 0), (1, 2, 1), (1, 3, 1), (2, 1, 2)\}$. And we also see that the algorithm stops at step number 5 even though in the figure we continue beyond this step for illustrative purposes. It should be noted, however, that it is also possible for the algorithm to return the ordered list of all l^2 combinations simply by letting it continue and not stop it after the first l members are found.

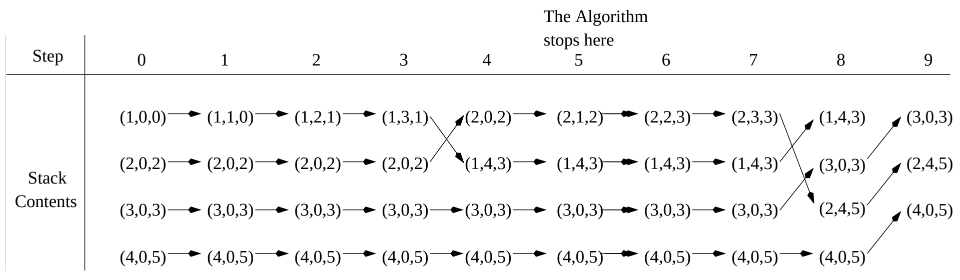


Figure 3.4: The progress of Algorithm 3.1 to solve Example 3.1

It is very important to note that algorithm 3.1 is only an example of many algorithms that can perform the same task in $O(l)$ steps or, equivalently, that it requires $O(l \log l)$ comparisons and at most $2l$ additions. We present the algorithm here in order to develop the ideas for analyzing the decoding algorithm afterward.

We are now ready to present the decoding algorithm, which we will denote by μ, in a more formal manner. In the description below, we assume that the decoding is on a code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$ and that the decoding radius e for the list decoder for the rows, ξ_e is chosen in such a way that the list decoder always returns a list of at least l codewords in $\mathcal{A}$. This will be further discussed in Section 3.3 below. The list $\boldsymbol{L} = \{\boldsymbol{L}^{(1)}, \boldsymbol{L}^{(2)}, \ldots, \boldsymbol{L}^{(m-1)}\}$ is a list of $m - 1$ stacks, each of which is similar to the stack $\boldsymbol{g}$ used in Algorithm 3.1 above and the sorting function λ is used for sorting these stacks. The matrix Y in the algorithm below is $m \times l$ of elements such that $Y_{i,j} \in \mathcal{A}'$. In step 6, we mean that only the l closest members of $\xi_e(\boldsymbol{y}_{i,\cdot}, \mathcal{A})$ are assigned to Y_i and that they are ordered according to their distances from the corresponding row in $\boldsymbol{y}$. If the number of candidate codewords for a row, say, i was *less* than l, then, only this limited number of solutions is used and copied to Y_i.

Algorithm 3.2 *Decoding Function μ*

```
1   Input: $m \times n$ matrix $\boldsymbol{y}$ of real numbers and a real number $e$.
2   Output: $m \times n$ matrix $\hat{\boldsymbol{x}}$ of binary numbers and a binary flag $S$
3   Variables: $h, i, j \in \mathbb{N}$, $m \times l$ matrix $Y$, a list $\boldsymbol{L} = (\boldsymbol{L}^{(1)}, \ldots, \boldsymbol{L}^{(m-1)})$ such that
    $\boldsymbol{L}^{(k)} \in \mathbb{N}^l \times \mathbb{N}^l \times \mathbb{R}^l$.
4   Initialize: $i \leftarrow 1$, $j \leftarrow 1$, $h \leftarrow 1$, $S \leftarrow 0$.
5   for $i = 1$ to $m$ do
6       $Y_i \leftarrow \xi_e(\boldsymbol{y}_{i,\cdot}, \mathcal{A}')$
7   end for
8   $\boldsymbol{L}^{(1)} \leftarrow \lambda(D(\boldsymbol{y}_{1,\cdot}, Y_1), D(\boldsymbol{y}_{2,\cdot}, Y_2))$
9   for $i = 2$ to $m$-1 do
10      $\boldsymbol{L}^{(i)} \leftarrow \lambda(\boldsymbol{L}^{(i-1)}_{3,\cdot}, D(\boldsymbol{y}_{i+1,\cdot}, Y_{i+1}))$   ▷ Generate the lists of stacks.
11  end for
12  while $j \leq l$ AND $S = 0$ do
13      $h \leftarrow \boldsymbol{L}^{(m-1)}_{2,j}$                  ▷ Construct the j:th element of the list $\boldsymbol{A}$
14      $\hat{\boldsymbol{x}}_{i,\cdot} \leftarrow Y_{i,h}$                  ▷ by combining its rows from the matrix $Y$.
15      for $i = m$-1 downto 1 do
16          $h \leftarrow \boldsymbol{L}^{(i)}_{1,j}$
17          $\hat{\boldsymbol{x}}_{m,\cdot} \leftarrow Y_{m,h}$
18      end for
19      if $\hat{\boldsymbol{x}} \in \mathcal{B}$ then                ▷ Check to see if the j:th element is
20          $S \leftarrow 1$                         ▷ a valid codeword. If it was, stop
21      end if                               ▷ the search. Otherwise, continue
22      $j \leftarrow j + 1$                         ▷ to check the j+1:th element
23  end while
```

Figure 3.5: Decoding algorithm for product codes.

This is a slight abuse of the mathematic notation but we chose this form instead of a more correct but cumbersome notation. If the flag S is equal to 1, then we know that the decoding was successful. Step 19 can be implemented, for example, by a simple syndrome check or information set check. If the decoder doesn't find a valid codeword, then it stops and declares that decoding was unsuccessful or we can choose one of the words in the list to acquire the information bits. It is also possible to add new codewords to the list afterward. This option, however, doesn't help in the analysis of the algorithm. We also observe that the algorithm doesn't generate the list $\boldsymbol{A}$ of ordered codewords in $\mathcal{A}$. Rather, it generates lists of indices $\boldsymbol{L}$ of candidate codewords for each row and then using these lists, in steps 12-23, to generate each element in $\boldsymbol{A}$. This is done in order to decrease the amount of memory needed for storage in the algorithm.

3.3 Analysis of performance

We saw in Section 3.1 that the performance of the decoding algorithm improves by increasing the list size l so as to ensure that the ML codeword is contained in the list. In order to increase l, however, we should use a list decoder $\xi_e(\cdot, \mathcal{A})$ of greater decoding radius e which means an increase in complexity. On the other hand, if there exists l candidate solutions for each row in the received matrix $\boldsymbol{y}$, it is possible to generate a list of l^m matrices that can be checked until a valid codeword, i.e., a codeword that is an element in both $\mathcal{A}$ and $\mathcal{B}$ is found. Also, the condition that the list decoder for the rows returns a list of at least l elements can be eased a bit, i.e., the lists of candidate codewords for each row can be shorter than l. Still, we can generate a very long list, let us call this list $\boldsymbol{A}^*$, which is exactly $\prod_{i=1}^{m} |\xi_e(\boldsymbol{y}_{i,\cdot}, \mathcal{A})|$ long. We will not, however, rewrite Algorithm 3.2 with the new modification since this won't add much to understanding the procedure. It is clear that $\boldsymbol{A}^* \subset \boldsymbol{A}$ and it is also quite clear that $\boldsymbol{A}^*$ is a function of the decoding radius of the list decoder for the rows and the received message $\boldsymbol{y}$. We will refrain however from writing other symbols showing this dependence for the sake of simplicity of notation. However, it should be noted that it is not certain that a codeword found by successively checking the members of $\boldsymbol{A}^*$ instead of $\boldsymbol{A}$ will be the ML codeword.
We discuss here the probability of decoding error, i.e., the probability that the decoded message is different from the sent message and not whether it was the ML codeword or not. We thus have to analyze the performance of the decoder for both random errors and burst errors when the list decoder for the rows has a fixed decoding radius. Even though we leave the discussion of complexity to a later chapter, we can state that the complexity of the list decoder for the rows increases greatly when the decoding radius exceeds d_A. Therefore, we will give special interest to the case where the decoding radius of the list decoder for the rows is less than d_A. We also assume that a system of an encoder using a binary linear product code, $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$, with parameters $(mn, MN, d_A d_B)$, and the slight modification mentioned above of Algorithm 3.2 is used for decoding on a binary channel. The ideas can be easily modified to accommodated other types of channels or other types of product codes.

We define a *burst error* as a very long pattern of errors that occupy less than $\lfloor d_A/2 \rfloor$ columns without consideration to the total Hamming weight of the error or, alternatively, an error pattern covering less than $\lfloor d_B/2 \rfloor$ rows without consideration to the total Hamming weight of the error. We assume that a burst error will either cover several columns or several rows but not both at the same time since it would then mean that the Hamming weight of the error pattern is less than $\lfloor d_A d_B/4 \rfloor$ located in a rectangle of dimensions less than $d_B/2 \times d_A/2$. We present the following proposition:

Proposition 3.1 *Let $\mathcal{A}'$, $\mathcal{B}'$ be, respectively, an $[n, k_A, d_A]$ code and an $[m, k_B, d_B]$ code and let $\mathcal{A}$ be the code represented by all $m \times n$ matrices with their rows codewords in $\mathcal{A}'$. Also let $\mathcal{B}$ be the code represented by all $m \times n$ matrices with their columns codewords in $\mathcal{B}'$. Let the product code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$ be used in combination with Algorithm 3.2 above for data transmission on a channel with burst errors. If e in Algorithm 3.2 is chosen between $\lfloor d_A/2 \rfloor$ and $\rho(\mathcal{A}')$, then, the decoder can correct all burst errors covering less than $d_A/2$ columns and some burst errors covering less than $\rho(\mathcal{A}')$ columns.*

Proof: Assume that the all zero codeword was sent and the received message $\boldsymbol{y}$ has a burst error covering less than $\rho(\mathcal{A}')$ columns. If the decoding radius of the row decoder is equal to $\rho(\mathcal{A}')$, the zero codeword will be one of the members of the list of candidate codewords for each row. This means that the all zero codeword will be a member of $\boldsymbol{A^*}$. If the all zero codeword was the closest codeword to the received message, the decoder will confirm that it is a valid codeword and return it as the correct answer. Otherwise, if there was another valid codeword in the list that is closer to $\boldsymbol{y}$, then it will be chosen instead. If the number of columns containing burst errors is less than $\lfloor d_A/2 \rfloor$, then there cannot be any other codeword $\boldsymbol{x} \in \mathcal{C}$ in $\boldsymbol{A^*}$ that is closer to $\boldsymbol{y}$ than the all zero codeword. If the burst error covers more than $d_A/2$ columns, then, the decoder will pick another codeword if and only if it was closer to $\boldsymbol{y}$ than the all-zero codeword. □

Next is to consider random errors. There are two aspects for the analysis of such errors. The first is to consider the probability that the sent codeword is a member of the list $\boldsymbol{A^*}$, and the other is the probability that there is no other codeword within $\rho(\mathcal{C})$ that is a member of $\boldsymbol{A^*}$ and, at the same time, closer to the received message than the sent codeword. Let us denote the probability that the sent codeword is not a member of the list $\boldsymbol{A^*}$ by P_{list}. We begin by noticing that for a Binary Symmetrical Channel (BSC), with transition probability p, the probability that one row of the received matrix contains e or less errors is equal to:

$$\sum_{i=0}^{e} \binom{n}{i} p^i (1-p)^{n-i},$$

where e is the decoding radius of the list decoder for the rows. If at least one of the rows of the received matrix contains more that e errors, then, the sent codeword will not be a member of the list $\boldsymbol{A^*}$. Therefore, the probability that the sent codeword is not a member of the list $\boldsymbol{A}$ is simply:

$$P_{\text{list}} = 1 - \left(\sum_{i=0}^{e} \binom{n}{i} p^i (1-p)^{n-i} \right)^m . \tag{3.2}$$

A similar expression is obtained for the probability that the sent codeword does not exist in the list $\boldsymbol{B^*}$ by replacing n by m and vice versa. The decoding radius for list decoder of the columns in the expression above is also e. It can be shown, using elementary probability theory, that the event that the sent codeword is a member of the list $\boldsymbol{A}^*$ is independent from the event that it is a member of the list $\boldsymbol{B}^*$. This means that the probability that the sent codeword does not exist in $\boldsymbol{A}^* \cap \boldsymbol{B}^*$ can be written as:

$$P_{\boldsymbol{A} \cap \boldsymbol{B}} = 1 - \left(\sum_{i=0}^{e} \binom{n}{i} p^i (1-p)^{n-i} \right)^m \left(\sum_{j=0}^{e} \binom{m}{j} p^i (1-p)^{m-j} \right)^n, \quad (3.3)$$

where it is assumed that the decoding radius for the list decoder of the rows is equal to the decoding radius for the list decoder of the columns and is equal to e. The probability $P_{\boldsymbol{A} \cap \boldsymbol{B}}$ is the probability that at least one row or one column contains more than e errors. The probability that there is another valid codeword in $\boldsymbol{A}^*$ that is closer to $\boldsymbol{y}$ than the sent message, is much more complicated. It is possible, however, to give some information about the structure of such a codeword and derive an upper bound on the probability of such event. We start by noticing that in order for a block error leading to one of the minimum weight codewords in $\mathcal{C}$, to occur, there requires at least $\lceil d_A d_B/2 \rceil$ ones located in the support, i.e., the set of indices of non-zero positions, of such a codeword. But all such codewords constitute, up to permutation of the rows and columns, a rectangle of dimensions at most $d_B \times d_A$. Thus, the supports of those $\lceil d_A d_B/2 \rceil$ ones must be contained inside such a rectangle. Therefore, it is possible to say that for any error pattern that can be decoded to one of the minimum weight codewords, we can find at least $\lceil d_A d_B/2 \rceil$ ones that are located inside a rectangle of dimensions $d_B \times d_A$. We will try to make a similar constriction on the structure of the error in the received matrix for a more general case and not only for the case of error patterns leading to a minimum weight codeword. We present the following lemma and leave the task of proving it to Appendix A. If the number of errors in each row is less than or equal to the decoding radius of the list decoder for the rows, e, then, the sent codeword will always be a member of the list $\boldsymbol{A}^*$. However, a decoding error might still occur if there existed in $\boldsymbol{A}^*$ another valid codeword that is closer to the received message. The following lemma explains which error patterns can lead to a decoding error under such conditions. Before we continue with the analysis of performance we present the following definitions first. The *generalized Hamming weights* of the code $\mathcal{D}$ with dimension k, see [56], are defined as:

$$d_i(\mathcal{D}) \overset{\triangle}{=} \min_{E} |\mathrm{Supp}(E)|, \quad i = 1, 2, \ldots, k, \quad (3.4)$$

where the minimum is taken over all linear sub-code $E \subseteq \mathcal{D}$ that have dimension i. For convenience, we assume:

$$d_0(\mathcal{D}) \overset{\triangle}{=} 0,$$

by definition. It is clear that $d_1(\mathcal{D}) = d(\mathcal{D})$, i.e., the minimum distance of the code. Let $\mathcal{A}'^{\perp}$ be the dual code of the code $\mathcal{A}'$. We define the sequence $d_1^{\perp}, d_2^{\perp}, \ldots, d_{k_A}^{\perp}$

to be the generalized Hamming weights of the dual code. Let $\mathcal{A}^*$ be an $[n^*, k_A^*, d_A^*]$ code obtained by shortening some of the coordinates of $\mathcal{A}'$, see MacWilliams and Sloane [8, page 29].

We also define the *constructing rectangles* of a product code $\mathcal{C}$ as all codewords that have the shape, up to permutations of the rows and columns, of a rectangle.

Lemma 3.2 *Let the product code $\mathcal{C} \triangleq \mathcal{A} \cap \mathcal{B}$ be used with the decoder μ presented in Algorithm 3.2 introduced above for decoding. Let the decoding radius e of the list decoder for the rows be less than d_A and let the received matrix be $\boldsymbol{y}$. If all the following:*

1. *The Hamming weight of the error in each row in $\boldsymbol{y}$ is less than e.*
2. *The Hamming weight of the total error is less than $\omega_A/2$, where:*
$$\omega_A \triangleq \frac{d_A d_B}{d_A - 2}(n - k_A - r'), \tag{3.5}$$
where r' is an integer satisfying:
$$d^{\perp}_{r'+1} \geq n - \frac{\omega_A}{d_B}, \quad d^{\perp}_{r'} < n - \frac{\omega_A}{d_B}. \tag{3.6}$$
3. *The support of every constructing rectangle in $\boldsymbol{y}$ with dimensions $f \times g$ where $g \leq 2e$ contains less than $fg/4$ errors.*

is correct, then, the decoding will be error-free.

Proof: See Appendix A □

It should be noted that Inequality (3.6) can always be satisfied for some integer r'. This is easily proven by noticing that ω_A is monotonically decreasing with r' while $d^{\perp}_{r'}$ is monotonically increasing with r'. Furthermore, there exists a certain point, namely $r' = 0$, such that:
$$d^{\perp}_0(\mathcal{A}') = 0 < n - \frac{\omega_A}{d_B}.$$
We can also find another point, namely $r' = n - k_A$ such that:
$$d^{\perp}_{n-k_A}(\mathcal{A}') = n.$$
Therefore, the two functions intersect at some point within the interval $0 < r' \leq n - k_A$.

The lemma above shows that under the conditions given in the lemma, an error pattern leading to a decoding error will be such that at least $fg/4$ errors are contained in the support of a constructing rectangle of dimension less than $f \times g$. Even though this is very useful information, there is an even stronger condition for certain error weights. The following lemma gives such a condition. The content of this lemma is based on the properties of product codes and their weight distribution. A more detailed discussion about the weight distribution of product codes can be found in [57] and [38].

Lemma 3.3 *Let the product code $\mathcal{C} \overset{\triangle}{=} \mathcal{A} \cap \mathcal{B}$ be used with Algorithm 3.2 with hard decoding. Let the decoding radius of the list decoder for the rows be e, where:*

$$e \geq \lfloor \frac{d_A - 1}{2} \rfloor.$$

Let the sent codeword be $\boldsymbol{x}$ and the received matrix be $\boldsymbol{y}$. Let:

$$d_H(\boldsymbol{x}, \boldsymbol{y}) < d_A d_B - \lfloor \frac{d_A}{2} \rfloor \lfloor \frac{d_B}{2} \rfloor. \tag{3.7}$$

If there exists a codeword $\boldsymbol{c}$ in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than the sent codeword, then, there exists at least $\lceil w_H(\boldsymbol{c})/2 \rceil$ errors contained in the support of a constructing rectangle of dimensions less than $f \times g$, where $g \leq 2e$ and $f = w_H(\boldsymbol{c}/g)$.

Proof: Assume that the all zero codeword was sent. The minimum weight codewords in the product code have the shape of a rectangle, up to permutation of the rows and columns, with which sides are either minimum weight codewords from $\mathcal{A}'$ and $\mathcal{B}'$ or the all zero vector. If the codeword is made up of adding two constructing rectangles, each of which is a minimum weight codeword in the product code, then, the weight of such a codeword will be at least:

$$w(d_A, d_B) \overset{\triangle}{=} 2 d_A d_B - 2 \lfloor \frac{d_A}{2} \rfloor \lfloor \frac{d_B}{2} \rfloor, \tag{3.8}$$

since two constructing rectangles cannot overlap by more than:

$$\lfloor \frac{d_A}{2} \rfloor \lfloor \frac{d_B}{2} \rfloor,$$

ones. All codewords with weights ranging between $d_A d_B$ and $w(d_A, d_B)$ given in Expression (3.8) will have the shape of a rectangle. Therefore, an error pattern with weight less than that given in (3.7), will lead to a different codeword if and only if at least $fg/2$ errors are located in the support of a constructing rectangle with size less than $f \times g$, where f and g are the sides of a constructing rectangle. □

We are now ready to give an upper bound on the probability of block error for product codes when the decoder described above is used:

Theorem 3.4 *Let the product code $\mathcal{C} \overset{\triangle}{=} \mathcal{A} \cap \mathcal{B}$ and the Algorithm 3.2 be used for data transmission on a BSC. Let the decoding radius for the list decoder of the rows be e, where e is less than $\min(d_A, d_B)$ and let the transition probability for the channel be p. The probability for block error, $P_{\mathcal{E}}$, in decoding is upper bounded as follows:*

$$\begin{aligned} P_{\mathcal{E}} \leq\; & 1 - \left(\sum_{i=0}^{e} \binom{n}{i} p^i (1-p)^{n-i} \right)^m \\ & + \sum_{i=d_A}^{2e} \beta_i(\mathcal{A}') \sum_{j=d_B}^{m} \beta_j(\mathcal{B}') \sum_{h=\lceil ij/2 \rceil}^{\lfloor w(d_A,d_B)/2 \rfloor} P(p,h,i,j,2) \\ & + \sum_{i=d_A}^{2e} \beta_i(\mathcal{A}') \sum_{j=d_B}^{m} \beta_j(\mathcal{B}') \sum_{h=\lfloor w(d_A,d_B)/2 \rfloor + 1}^{\omega_A/2} P(p,h,i,j,4) \\ & + \sum_{i=\omega_A/2}^{mn} \binom{mn}{i} p^i (1-p)^{mn-i}, \end{aligned} \tag{3.9}$$

where β_i is the number of codewords that have weight equal to i, [8, pp. 40]. The function $P(p,h,i,j,l)$ is the probability that the received matrix $\boldsymbol{y}$ has h errors and such that at least $\lceil ij/l \rceil$ errors are contained in a rectangle of dimensions $j \times i$ as shown below:

$$P(p,h,i,j,l) = \binom{mn}{h} p^h (1-p)^{mn-h} \sum_{g=\lceil ij/l \rceil}^{ij} \frac{\binom{h}{g}\binom{mn-h}{ij-g}}{\binom{mn}{ij}}. \tag{3.10}$$

Comments about Theorem 3.4

Before presenting the proof of the theorem, we give some explanation about the meaning of this theorem. The theorem says that an error event can occur in two cases: The first is when the sent codeword is not a member of the list of matrices that will be checked. The probability of this event is given in the first row of Inequality (3.9). The second event that may lead to an error is that even though the sent codeword is a member of the list there exists another codeword in the list that is closer to the sent codeword. A bound on the probability of the event that there exists another codeword in the list can be given by dividing this event into three sub-events. The first sub-event is when the number of errors added by the channel is greater than $\lceil d_A d_B/2 \rceil$ and less than:

$$d_A d_B - \lfloor \frac{d_A}{2} \rfloor \lfloor \frac{d_B}{2} \rfloor. \tag{3.11}$$

For error patterns of this error weight, the only way for an error to occur is an error leading to codeword which has the shape of a rectangle. The second sub-event is when the weight of the error pattern exceeds that given in (3.11) and smaller than $\omega_A/2$. Under the conditions imposed by the theorem, an error pattern with such weight that may lead to an error in decoding should have at least $\lceil ij/4 \rceil$ errors contained in the support of a constructing rectangle of dimensions less than $i \times j$. For error patterns of weight exceeding $\omega_A/2$, the theorem does not give any prediction whether an error pattern of weight greater than $\omega_A/2$ will cause a decoding error or not, rather, it assumes that all error patterns of such weight will cause an error.

Proof: Let the sent codeword be the all-zero codeword and the received matrix be $\boldsymbol{y}$. Using Lemma 3.2, The Event of error *might* occur if one of the following conditions is satisfied:

1. The all-zero codeword is not a member of the list $\boldsymbol{A}^*$.
2. The all zero codeword *is* a member of the list $\boldsymbol{A}^*$ but there exists in $\boldsymbol{A}^*$ another codeword in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than the all-zero codeword.

The second event above can be further partitioned and the total event of error can be written in the following way:

1. The all-zero codeword is not a member of the list $\boldsymbol{A}^*$.
2. The all zero codeword is a member of the list $\boldsymbol{A}^*$. There exists in $\boldsymbol{A}^*$ another codeword in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than the all-zero codeword. The weight of the error pattern is greater than $\lceil d_A d_B/2 \rceil$ but less than $d_A d_B - \lfloor d_A/2 \rfloor \lfloor d_B/2 \rfloor$.
3. The all zero codeword is a member of the list $\boldsymbol{A}^*$. There exists in $\boldsymbol{A}^*$ another codeword in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than the all-zero codeword. The weight of the error pattern is greater than $d_A d_B - \lfloor d_A/2 \rfloor \lfloor d_B/2 \rfloor$ but less than $\omega_A/2$.
4. The all zero codeword is a member of the list $\boldsymbol{A}^*$. There exists in $\boldsymbol{A}^*$ another codeword in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than the all-zero codeword. The weight of the error pattern is greater than $\omega_A/2$.

The probability that the all-zero codeword is not a member of the list $\boldsymbol{A}^*$ P_{list} given in (3.2).
The probability of the fourth event can be bounded by the probability that the error pattern is greater than $\omega_A/2$ which is

$$\sum_{i=\omega_A/2}^{mn} \binom{mn}{i} p^i (1-p)^{mn-i}.$$

The second event can be bounded in the following manner: If there exists in $\boldsymbol{A}^*$ a codeword in $\mathcal{C}$ other than the all-zero codeword, then, the number of ones in each row of this codeword cannot exceed $2e$. Therefore, using Lemma 3.3, there has to exist in $\boldsymbol{y}$ at least $\lceil ij/2 \rceil$ ones located in the support of at least one constructing rectangle of dimensions less than $j \times i$.

The third event can be bounded in the following manner: If there exists in $\boldsymbol{A}^*$ a codeword in $\mathcal{C}$ other than the all-zero codeword, then, the number of ones in each row of this codeword cannot exceed $2e$. Therefore, using Lemma 3.2, there has to exist in $\boldsymbol{y}$ at least $\lceil ij/4 \rceil$ ones located in the support of at least one constructing rectangle of dimensions less than $j \times i$.

The probability that a specific rectangle of dimensions $j \times i$ in $\boldsymbol{y}$ contains g ones given that the Hamming weight of $\boldsymbol{y}$ is h, is similar to the probability of picking ij balls from an urn containing h black balls and $mn - h$ white balls and such that g balls of the chosen ij are black. See, for example, Hines et al [58, page 30] and Blom [59, page 30]. The probability of this occurring will then be:

$$\frac{\binom{h}{g}\binom{mn-h}{ij-g}}{\binom{mn}{ij}}. \tag{3.12}$$

The probability that the Hamming weight of $\boldsymbol{y}$ is equal to h given that the transition probability is p, is:

$$\binom{mn}{h} p^h (1-p)^{mn-h}.$$

The number of constructing rectangles with dimensions $j \times i$ is equal to $\beta_i(\mathcal{A}')\beta_j(\mathcal{B}')$. By multiplying with this number and by summing over all the probabilities of different Hamming weights of $\boldsymbol{y}$, we prove the second and the third term in (3.9). $\square$

The bound given in the theorem above is an upper bound on probability of block error for product codes since the decoding algorithm used for proving the theorem is suboptimal in comparison to a ML decoder.

It should be noted that the second term of the bound sum up the probabilities that more than half the symbols in some rectangle in the received matrix are in error. It can be checked, however, that the dominant probability is that a rectangle of dimensions exactly equal to $d_B \times d_A$ has $\lceil d_A d_B/2 \rceil$ errors. The probability that a rectangle of greater dimensions with more than half of its symbols in error is much lower. The same is true for the third term of the bound.

Since the bound on the block error probability makes use of the union bound, some peculiarities might be noticed in the value of this bound. For example, for very high transition probability, the value of the bound might exceed 1 which makes the bound useless. Also, when the decoding radius approaches the minimum distance of the row code, d_A, the bound on probability that at least one rectangle contains $d_A d_B/4$ ones becomes higher than it should, since the probability of more than one rectangle containing $d_A d_B/4$ ones at the same time becomes very high.
It is clear that the bound requires that we have some information about the weight distribution of the constituent codes, but it is always possible to use some bounds on the weight distribution.

In order to investigate the practicality of using the basic decoding algorithm shown above, several examples of systems are given to illustrate the possibilities and limitations of implementing the algorithm. As will be shown in Chapter 5, the complexity of the list decoder of the rows will increase exponentially when the decoding radius becomes greater than the minimum distance of the code. Therefore, we will assume in the following examples a decoding radius of the list decoder less than or equal to $d_A - 1$.

Example 3.2 Consider a simple case where the constituent codes of the product code are the same and are the $[8, 4, 4]$ extended binary Hamming code. The rate of this code is 0.25 and, using the bound on the probability of block error given in Theorem 3.4 and, taking for example a transition probability equal to 0.05, the upper bound on the probability for block error when the decoding radius of the list decoder for the rows is equal to 3 will be equal to 0.03. The half the minimum distance bound predicts a block error probability of about 0.044 for the same transition probability. For transition probabilities less than 0.04, the half the minimum distance bound is better than the new bound.

Example 3.3 If we use an even larger code, for example, if we use the extended binary Golay code as a constituent code, the resultant product code has a rate equal to 0.25. The transition probability used is the same as in the example above and a decoding radius for the rows equal to 5. The upper bound on the probability of block error will be 0.023. The half the minimum distance bound predicts a block error probability less than 0.23.

Comparing the two examples above, we see that increasing the size of the code and the minimum distance can result with better performance, as expected, even though the decoding radius of the list decoder for the rows is slightly greater than $\lfloor (d_A - 1)/2 \rfloor$ for the two codes. In the following example we investigate the different terms in the bound given in Theorem 3.4 and how the bound on the error varies for different decoding radii.

Example 3.4 The constituent codes for the product code considered in this example are the same, the $[32, 21, 6]$ extended BCH code. In Figure 3.6, the graph shows the bounds on the probability of block error for the [32,21,6]X[32,21,6] product code. The different terms in the bound are shown separately to illustrate their effect on the total sum. We see that for high transition probabilities, the probability that at least one rectangle in the received matrix contains more than $\lfloor d_A d_B/4 \rfloor$ errors, i.e., the third term in Inequality (3.9) has most effect on the bound. For lower transition probabilities, the probability that at least one row in the received matrix has more errors than the list decoder can handle, will be more eminent. The probability that a rectangle in the received matrix has $d_A d_B/2$ errors will always be very small in comparison to the other terms in the bound. In Figure 3.7, we compare the bound on the same code but with different decoding radii for the list decoder of the rows. The bounds are given as a function of the signal-to-noise ratio E_b/N_0. This is done in order to appreciate the improvement of the bound measured in dB in comparison to half the minimum distance bound. Assuming coherent BPSK BPSK modulation in AWGN channel, the transition probability for such a system can be written as given in Equation 2.8:

$$p = \mathrm{Q}(\sqrt{\frac{2R_C E_b}{N_0}}), \tag{3.13}$$

where E_b/N_0 is the signal to noise ration per symbol, R_C the rate of the code and Q is the Q-function defined in (2.9). Also, in the same graph, simulation results for turbo decoding of the same product code are included, [14] [17]. Soft decoding of rows and columns was performed using MAP decoding, see Bahl et al [18] on the dual codes of the constituent codes, see Battail et al [53] and [60], Berkmann [61], Hagenauer [17] and Riedel [62]. The number of iterations chosen is 10 in order to be as close to optimum decoding as possible. It can be seen that the bound on block error is much better than the half the minimum distance bound for almost all the span of signal to noise ratio and closer to the results of turbo decoding of the product code. Only at very high signal to noise ratio is the half the minimum distance bound better than the new bound. We should keep in mind, however, that in practice, the interesting region of block error probability is between 10^{-3} and 10^{-2}, see Furuskär [63, p. 18] and the references therein.

From the previous examples we see that increasing both the size and the minimum distance of the constituent codes might result in better performance. This leads to the question of what might happen in the asymptotic case if we continue to increase the size and minimum distance of the code. The first observation is that if the number of errors in each row were less than half the minimum distance for the row code, then it would be sufficient to use a bounded minimum distance decoder instead of a list decoder for the rows and the maximum likelihood codeword will be the first element in the list $\boldsymbol{A^*}$. In practice, a bounded minimum distance decoder for the rows will not be sufficient. Instead, a $d_A/2+1$ list decoder can be used. For very large product codes, i.e., when the length of the constituent codes approach

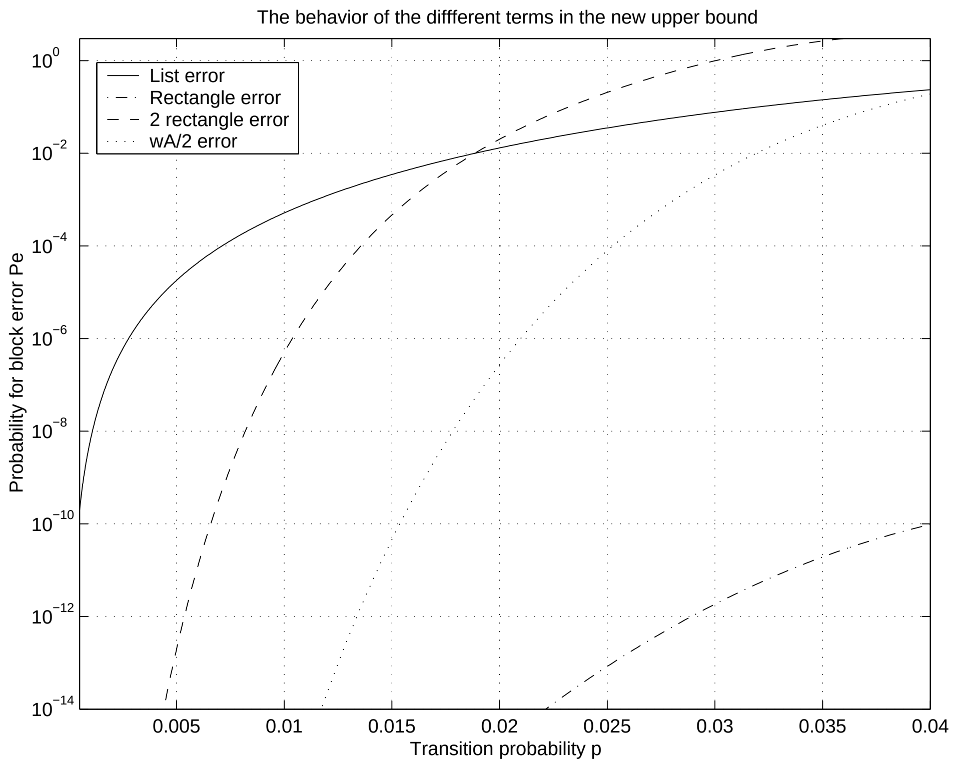


Figure 3.6: Different terms of bound (3.9)

infinity, the transition probability of the channel should not exceed:

$$p < \frac{d_A}{2n}.$$

This is much larger than the transition probability predicted by using the half minimum distance bound which is less than $d_A d_B/2mn$.

It should be noted that other bounds on the probability of block error can also be used. For example, if the weight distribution of the product code was known, then it is possible to use a bound similar to Viterbi's bound, [36] or Meeberg's bound, [64], as shown in [65]. Such bounds, however, require full or very good knowledge of the weight distribution of product codes. The bound presented in this thesis, on the other hand, only requires some knowledge on the weight distribution of the constituent codes. The new bound also requires some knowledge on the weight hierarchy of the row code in order to evaluate the value of ω_A. It is, however, possible to bound the value of ω_A without knowing the weight hierarchy of the row code as shown in Appendix A.

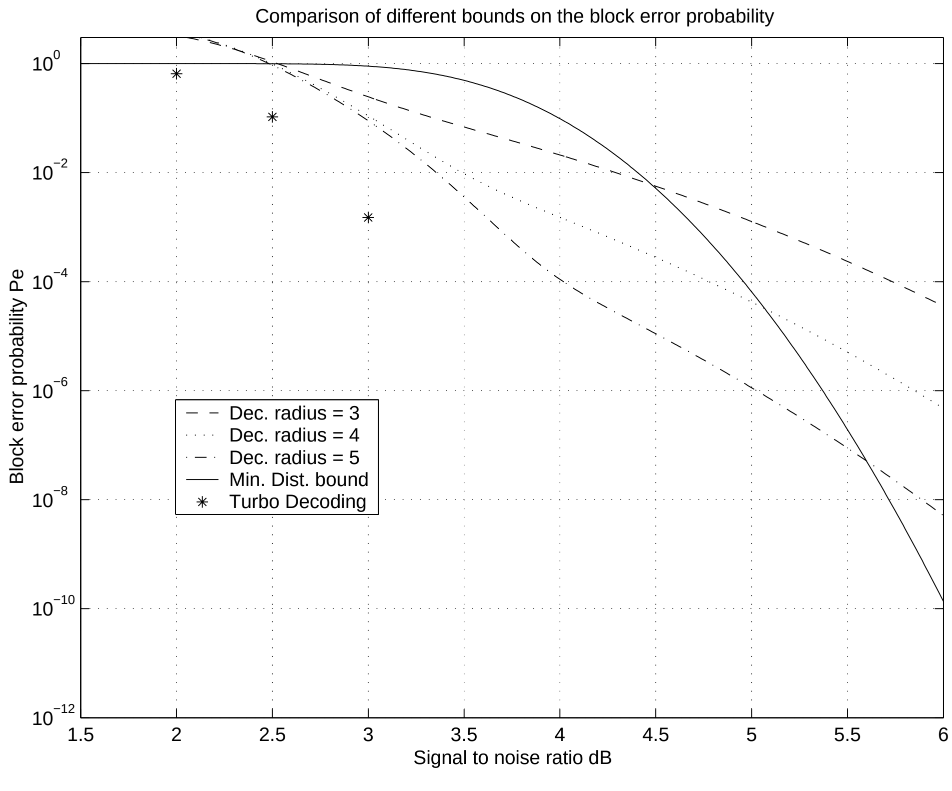


Figure 3.7: Comparison between the new upper bound and half the minimum distance bound.

This page intentionally contains only this sentence.

Chapter 4

Suboptimal Low Complexity Decoding

In the previous chapter, a maximum likelihood decoder was presented and some of it's properties were discussed. Also, another variant was mentioned where the decoder jumps between the two lists $\boldsymbol{A}$ and $\boldsymbol{B}$, shown in Figure 3.1, while looking for a valid codeword. Even though the two lists procedure converges faster than the single list variant towards the maximum likelihood codeword, the complexity is twice as great. Another important disadvantage is that the decoding radius of the list decoder for the rows should be sufficiently large in order to guarantee that the maximum likelihood codeword will be a member of the list $\boldsymbol{A}$. Furthermore, at each stage the decoder does not make use of the possibility that many of the errors may already be corrected by a previous stage. In this chapter, we present an iterative, suboptimal variant of the two lists method that fixes the complexity of the decoder to a predetermined value and instead of bookkeeping a long list of candidate codewords for the rows and columns, the result of each stage is re-decoded by the following stage creating a new list and forgetting the old ones from the previous stages. It will be shown that the performance of this algorithm improves by increasing the decoding radius of the list decoders for the rows and for the columns until it becomes maximum likelihood when the limitations are totally removed. This flexibility of adjusting the complexity in order to improve the performance, and keeping the complexity of the decoder constant for each stage are the main advantages for using this suboptimal algorithm.

4.1 Description of the iterative algorithm

We assume, as we did in the previous chapter, that the product code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$ is used, where $\mathcal{A}$ is the code represented by all $m \times n$ matrices with rows codewords in the $[n, k_A, d_A]$ code $\mathcal{A}'$ and $\mathcal{B}$ is the code represented by all $m \times n$ matrices with columns codewords in the $[m, k_B, d_B,]$ code $\mathcal{B}'$. Also, let the received matrix be $\boldsymbol{y}$. The iterative decoder we present, needs only two different list decoders, one for the rows and the other for the columns implemented in hardware or software. The incoming message goes through the decoder and the output is fed back to the same decoder, which is quite common for all iterative methods. It is, however, easier to analyze the decoder by imagining that there exists a series of similar decoders cascaded one after the other, processing the data from the previous stage. This is illustrated in Figure 4.1. Decoding of the received matrix $\boldsymbol{y}$ is performed by using a decoder for the rows, ϕ, and a decoder for the columns, ψ. The result is re-decoded at each stage.

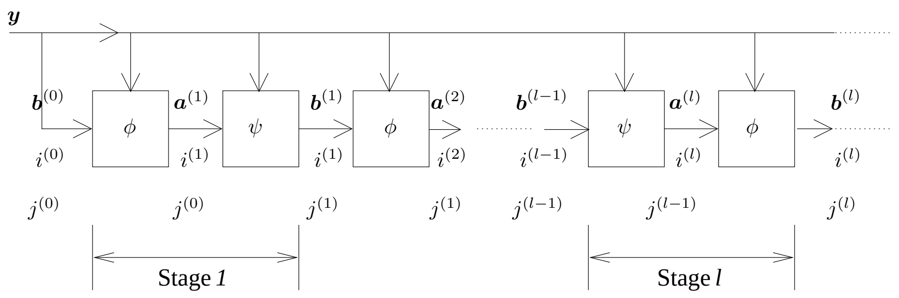


Figure 4.1: Decoding stages of the iterative decoder

Each stage is comprised of two functions, ϕ which is mainly a decoder for the code $\mathcal{A}$ and ψ which is a decoder for the code $\mathcal{B}$. We begin by explaining the different variables shown in the figure above. At stage l, for example, the suggested solution to the decoding problem from function ϕ is the matrix $\boldsymbol{a^l}$ which is a codeword in the code $\mathcal{A}$. This matrix, is processed by function ψ which gives its suggestion for the decoding problem in matrix $\boldsymbol{b^l}$ which is a member of the code $\mathcal{B}$. The matrix $\boldsymbol{b^l}$ is, in turn, processed by the function ϕ in stage $l+1$ and so on. At stage l, for example, the variables i^l and j^l are simply as follows:

$$\begin{aligned}
i^0 &\triangleq 0,\\
i^l &\triangleq D(\boldsymbol{a^l}, \boldsymbol{y}), \quad l = 1, 2, \ldots\\
j^0 &\triangleq 0,\\
j^l &\triangleq D(\boldsymbol{b^l}, \boldsymbol{y}), \quad l = 1, 2, \ldots . \qquad (4.1)
\end{aligned}$$

The function ϕ checks the incoming matrix $\boldsymbol{b^{l-1}}$ to see if it also is a member of the code $\mathcal{A}$. If it is, ϕ returns this matrix as the solution for the decoding. In a similar manner, the function ψ checks the matrix $\boldsymbol{a^l}$ to see if it also is a member of the code $\mathcal{B}$. If it is, ψ returns this matrix as the solution for the decoding. Otherwise, Those two functions process the incoming matrices in the following way: Included in each of the functions ϕ and ψ is a decoder similar to that shown in Algorithm 3.2, one for the rows and the other for the columns, respectively. Therefore, two lists are associated to each stage, namely, $\boldsymbol{A^l}$ and $\boldsymbol{B^l}$ the first generated at ϕ and the second at ψ. Each row of each member of the list $\boldsymbol{A^l}$ is one of the candidates of the list decoder of the rows for the corresponding row in $\boldsymbol{y}$. This can be written as a Cartesian product as follows:

$$\boldsymbol{A^l} = \prod_{h=1}^{m} \xi_{e_A}(\boldsymbol{b^{l-1}}_{h,\cdot}, \mathcal{A}'), \qquad (4.2)$$

where e_A is the decoding radius of the list decoder of the rows implemented in ϕ. In a similar manner, each column of each member of the list $\boldsymbol{B^l}$ is one of the candidates of the list decoder of the columns for the corresponding column in $\boldsymbol{y}$ and the list $\boldsymbol{B^l}$ can be written as follows:

$$\boldsymbol{B^l} = \prod_{h=1}^{n} \xi_{e_B}(\boldsymbol{a^l}_{\cdot,h}, \mathcal{B}'), \qquad (4.3)$$

where e_B is the decoding radius of the list decoder of the columns implemented in ψ. The matrices $\boldsymbol{a^l}$ and $\boldsymbol{b^l}$ are chosen from those lists in the following manner:

$$\begin{aligned}
\boldsymbol{a^l} &= \arg \min_{\substack{\boldsymbol{a'} \in \boldsymbol{A^l}\\ D(\boldsymbol{a'}, \boldsymbol{y}) > \min(i^{l-1}, j^{l-1})}} D(\boldsymbol{a'}, \boldsymbol{y})\\
\boldsymbol{b^l} &= \arg \min_{\substack{\boldsymbol{b'} \in \boldsymbol{B^l}\\ D(\boldsymbol{b'}, \boldsymbol{y}) > \min(i^{l}, j^{l-1})}} D(\boldsymbol{a'}, \boldsymbol{y}) \qquad (4.4)
\end{aligned}$$

In other words, the function ϕ at stage l, chooses the member of the list $\boldsymbol{A^l}$ that is closest to $\boldsymbol{y}$, but at a distance greater than the previous suggested solutions in the previous stages, namely, $\min(i^{l-1}, j^{l-1})$. Similarly, the function ψ at stage l, chooses the member of the list $\boldsymbol{B^l}$ that is closest to $\boldsymbol{y}$, but at a distance greater than the previous suggested solutions in the previous stages, namely, $\min(i^l, j^{l-1})$.

Algorithm 4.1 *Iterative decoding*

```
1  Input: $m \times n$ matrix $\boldsymbol{a}$ of real numbers and an integer number maximum- number-of-iterations.
2  Output: $m \times n$ matrix $\hat{\boldsymbol{x}}$ of binary numbers.
3  Initialize: $i \leftarrow 0$, $j \leftarrow 0$.
4  if $\boldsymbol{y} \in \mathcal{C}$ then
5      $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{y}$
6      Stop
7  end if
8  $\boldsymbol{b} \leftarrow \boldsymbol{y}$
9  while $l \leq$ maximum-number-of-iterations do
10     if $\boldsymbol{b} \in \mathcal{A}$ then
11         $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{b}$
12         Stop
13     end if
14     $(\boldsymbol{a}, i) \leftarrow \phi(\boldsymbol{b}, i, j)$
15     if $\boldsymbol{a} \in \mathcal{B}$ then
16         $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{b}$
17         Stop
18     end if
19     $(\boldsymbol{b}, j) \leftarrow \psi(\boldsymbol{a}, i, j)$
20     $l \leftarrow l + 1$
21 end while
```

Figure 4.2: The iterative, suboptimal algorithm for decoding product codes.

In Figure 4.2 we show the complete decoding algorithm.

From the definition of the two functions ϕ and ψ above, it is easy to see that at each stage, l, the two lists $\boldsymbol{A^l}$ and $\boldsymbol{B^l}$ associated with this stage are, respectively, subsets of the two lists $\boldsymbol{A}$ and $\boldsymbol{B}$ shown in Figure 3.1 for the maximum likelihood algorithm. It is not certain, however, that the maximum likelihood codeword will be chosen as the result in the end. The decoder given in Figure 4.2 might miss the maximum likelihood codeword, $\hat{\boldsymbol{x}}$ because of the following reasons:

- The maximum likelihood codeword is not a member of neither $\boldsymbol{A^l}$ nor $\boldsymbol{B^l}$ for all the stages.
- For any stage l that the maximum likelihood codeword $\hat{\boldsymbol{x}}$ is a member of $\boldsymbol{A^l}$, the distance $D(\hat{\boldsymbol{x}}, \boldsymbol{y})$ is less than $\min(i^{l-1}, j^{l-1})$.
- Similarly, for any stage l that the maximum likelihood codeword $\hat{\boldsymbol{x}}$ is a member of $\boldsymbol{B^l}$, the distance $D(\hat{\boldsymbol{x}}, \boldsymbol{y})$ is less than $\min(i^{l}, j^{l-1})$.

The fact that the decoder might miss the maximum likelihood codeword affects not only the performance of the decoder, it also means that the algorithm does not always converge to an answer. This is quite unfortunate and therefore the choice of the maximum number of iterations is of crucial importance. Furthermore, the decoder must be able to manage the cases where no valid codeword is found after the maximum number of iterations is reached. Ad hoc solutions may be implemented to solve the last problem, for example, the decoder may return a *failure* message or the decoder may choose any codeword of the list generated at this stage and uses it to return the information symbols. Even though it is not certain that the algorithm converges, it is possible, however, to say something about the probability of convergence. We begin by presenting the following proposition:

Proposition 4.1 *Let $\mathcal{A}$ be the code represented by all $m \times n$ matrices with rows codewords in the $[n, k_A, d_A]$ code $\mathcal{A}'$ and let $\mathcal{B}$ be the code represented by all $m \times n$ matrices with columns codewords in the $[m, k_B, d_B]$ code $\mathcal{B}'$. Let the product code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$, be used in combination with Algorithm 4.1 shown in Figure 4.2 for decoding. Also, let the decoding radius of the list decoder of the rows, e_A, and the decoding radius of the list decoder of the columns, e_B, be equal to or greater than $\rho(\mathcal{A}')$ and $\rho(\mathcal{B}')$, the covering radii of $\mathcal{A}'$ and $\mathcal{B}'$, respectively. Let the sent codeword be $\boldsymbol{x}$ and the received matrix be $\boldsymbol{y}$. Define:*

$$\begin{aligned} \mathcal{I} &\triangleq \{i | i \in \{1, 2, \ldots, m\}, D(\boldsymbol{x_{i,\cdot}}, \boldsymbol{y_{i,\cdot}}) > \frac{d_A}{2}\} \\ \mathcal{J} &\triangleq \{j | j \in \{1, 2, \ldots, n\}, D(\boldsymbol{x_{\cdot,j}}, \boldsymbol{y_{\cdot,j}}) > \frac{d_B}{2}\}. \end{aligned} \tag{4.5}$$

If $|\mathcal{I}| < d_B/2$ and $|\mathcal{J}| < d_A/2$, then, the decoder will converge after a sufficient number of iterations.

Proof: Let the sent codeword be the all zero matrix. Since e_A and e_B are equal to or greater than $\rho(\mathcal{A}')$ and $\rho(\mathcal{B}')$, respectively, then, the event that the list decoders for the rows and columns cannot produce any solution is not possible for all stages of decoding. In accordance with Figure 4.1, let $\boldsymbol{a^l}$ and $\boldsymbol{b^l}$ be the outputs from functions ϕ and ψ respectively for stage l. Due to the conditions imposed by the proposition, the matrix $\boldsymbol{a^1}$ will have less than $d_B/2$ rows that are not zero and the rest of the rows in the matrix are zero. Therefore, the all zero matrix will be one of the members of the list $\boldsymbol{B^1}$ associated with the first stage. The matrix $\boldsymbol{b^1}$, in its turn will have less than $d_A/2$ columns that are not zero. This means that the all zero matrix will be a member of the list $\boldsymbol{A^2}$ associated with the second stage. The function ϕ in stage 2 either chooses the all zero matrix as the result or there exists some other codeword in $\mathcal{A}$ that is closer to $\boldsymbol{y}$ than the all zero matrix. The same argument applies for all stages in decoding. This means that unless there exists some other codeword in $\mathcal{C}$ that is closer to $\boldsymbol{y}$ than the all zero matrix, the decoder will either choose the zero matrix or some other valid codeword, i.e., it converges. □

It is easy to see that when a BSC, with transition probability p, is used, the probability that, at least $m - d_B/2$ rows have a number of errors less than $d_A/2$, is greater than the following expression:

$$\left(\sum_{i=0}^{d_B/2} \binom{n}{i} p^i (1-p)^{n-i} \right)^{m-d_B/2}, \tag{4.6}$$

Similarly, the probability that at least $n - d_A/2$ columns have a number of errors less than $d_B/2$, is greater than the following expression:

$$\left(\sum_{i=0}^{d_A/2} \binom{m}{i} p^i (1-p)^{m-i} \right)^{n-d_A/2}, \tag{4.7}$$

We can thus say that the probability of convergence is of the same order as the expressions given in Equations (4.6) and (4.7). An exact expression, however, is quite cumbersome and would not add much to understanding the process.

4.2 Error correction capability of the suboptimal algorithm

As explained in the previous section, the iterative algorithm only approaches maximum likelihood performance when the decoding radii for the list decoders of the rows and the columns are very large and for unlimited number of operations. It is possible, however, to analyze its performance concerning certain types of error patterns. We begin by considering burst errors. As in Chapter 3, we define burst errors as a very long pattern of errors that occupy less than $\lfloor d_A/2 \rfloor$ columns or, alternatively, less than $\lfloor d_B/2 \rfloor$. We present the following proposition:

Proposition 4.2 *Let $\mathcal{A}$ be the code represented by all $m \times n$ matrices with rows codewords in the $[n, k_A, d_A]$ code $\mathcal{A}'$ and let $\mathcal{B}$ be the code represented by all $m \times n$ matrices with columns codewords in the $[m, k_B, d_B]$ code $\mathcal{B}'$. Let the product code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$, be used in combination with Algorithm 4.1 shown in Figure 4.2 for decoding. Also, let the decoding radius of the list decoder of the rows, e_A, and the decoding radius of the list decoder of the columns, e_B, be equal to or greater than $d_A/2$ and $d_B/2$ respectively. Then, the decoder can correct all burst errors covering less than $d_A/2$ columns or all burst errors covering less than $d_B/2$ rows.*

Proof: Let us assume that the all-zero codeword was sent and that the received matrix is $\boldsymbol{y}$ and start by proving the proposition for the case where the burst covers less than $d_A/2$ columns. Sine for each row, the all-zero solution will be the closest candidate in $\mathcal{A}'$ to the corresponding row in $\boldsymbol{y}$, then, the all-zero codeword will be the first member of the list boldmath A^1 associated with the first stage. Thus the all-zero codeword will be chosen as the correct answer. Now, let us assume that the burst covers less than $d_B/2$ rows. Then, the solution from the function ϕ in the first stage, i.e., $\boldsymbol{a^1}$, will have errors in less than $d_B/2$ rows. Sine for each column, the all-zero solution will be the closest candidate in $\mathcal{B}'$ to the corresponding column in $\boldsymbol{y}$, then, the all-zero codeword will be the first member of the list $\boldsymbol{B^1}$ associated with the first stage. Thus the all-zero codeword will be chosen as the correct answer in the following stage. This is illustrated in Figure 4.3. The figure shows the correction of burst errors that are contained in less than $d_A/2$ columns (left picture) or burst errors contained in less $d_B/2$ rows (right picture). The parameters e_A and e_B are the decoding radii of the list decoders of the rows and the columns respectively. □

Besides the ability to correct burst errors, the previous proposition points out the fact that correcting burst errors requires at most one iteration. It can also be noted that, if the decoding radii of the list decoders of the rows and the columns are equal to or greater than $\rho(\mathcal{A}')$ and $\rho(\mathcal{A}')$ respectively, then the decoder can correct some, but not all, burst errors that cover more than $d_A/2$ columns or $d_B/2$ rows.

The next type of errors to consider is when the number of errors is less than half the minimum distance of the product code. The theorem below states that, given some conditions, the decoder can correct all errors less than half the minimum distance:

Theorem 4.3 *Let $\mathcal{A}$ be the code represented by all $m \times n$ matrices with rows codewords in the $[n, k_A, d_A]$ code $\mathcal{A}'$ and let $\mathcal{B}$ be the code represented by all $m \times n$*

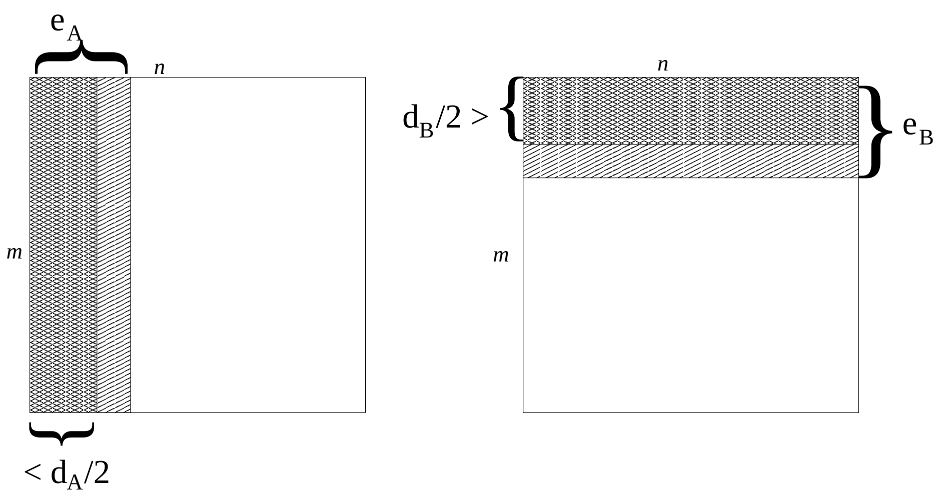


Figure 4.3: Correction of burst errors.

matrices with columns codewords in the $[m, k_B, d_B]$ code $\mathcal{B}'$, such that the covering radii for the codes $\mathcal{A}'$ and $\mathcal{B}'$ are less than d_A and d_B respectively. Let the product code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$, be used in combination with Algorithm 4.1 shown in Figure 4.2 for data transmission. Also, let the decoding radius of the list decoder of the rows, e_A, and the decoding radius of the list decoder of the columns, e_B, be equal to or greater than $d_A - 1$ and $d_B - 1$ respectively. Then, the decoder can correct all error patterns of Hamming weight less than half the minimum distance, $d_A d_B$, of the product code.

Proof: The proof is similar to the proof of Proposition 4.2. Since the decoding radii of the list decoders for the rows and columns are equal to or greater than the covering radii of the codes $\mathcal{A}'$ and $\mathcal{B}'$ respectively, the lists of candidates for the rows and columns will never be empty for all the stages of decoding. Let the sent codeword be the all-zero matrix and let the received message be the matrix $\boldsymbol{y}$. There will be four cases to consider:

1. Let the number of errors in each row be less than or equal to $d_A - 1$ and let the number of errors in each column be less than or equal to $d_B - 1$. Since the decoding radius of the list decoder for the rows is equal to $d_A - 1$, The list $\boldsymbol{A^i}$ associated with stage i will contain the all-zero codeword. Similarly, the list $\boldsymbol{B^i}$ associated with stage i will contain the all-zero codeword. Therefore, the decoder will, eventually, return the all-zero codeword as the correct solution since there are no other codeword in the code $\mathcal{C}$ closer to $\boldsymbol{y}$ than the all-zero codeword.

2. Let us assume that the number of errors in some rows in $\boldsymbol{y}$ are greater than $d_A - 1$. Even though the lists $\boldsymbol{A^i}$ might not contain the all-zero codeword for all stages i, the lists $\boldsymbol{B^i}$ will contain the all-zero codeword for all stages i of decoding and the decoder will eventually return it as the correct solution.

3. In a similar manner, let us assume that some columns in $\boldsymbol{y}$ contain more than $d_B - 1$ errors and that the number of errors in all the rows are less than or equal to $d_A - 1$. In this case all the lists $\boldsymbol{A^i}$ for each stage i will contain the all-zero codeword.

4. The last possibility is that some rows contain more than $d_A - 1$ errors and some columns contain more than $d_B - 1$ errors. The output from the function ϕ in the first stage, i.e., the matrix $\boldsymbol{a^1}$, will have errors contained in at most $d_B - 1$ rows. This means that the all-zero codeword will be a member of the list $\boldsymbol{B^1}$ generated at ψ. The matrix $\boldsymbol{b^1}$, in its turn, will have errors contained in at most $d_A - 1$ columns which means that the all-zero codeword will be contained in the list $\boldsymbol{A^2}$. Therefore, the matrices $\boldsymbol{a^i}$ and $\boldsymbol{b^i}$ decoded at each stage i will be similar to either the second case or the third case and the the all-zero codeword will be contained in either list $\boldsymbol{A^i}$ or in list $\boldsymbol{B^i}$ for each stage i of decoding and the decoder will eventually return it as the correct answer.

Figure 4.4 below illustrates the different cases of the proof. The four different cases of the theorem are when the number of errors in each row and in each column is less than or equal to $d_A - 1$ and $d_B - 1$ respectively, (upper left), the number of errors in some columns is greater than $d_B - 1$, (upper right), the number of errors in some rows is greater than $d_A - 1$ (lower left) and when the number of errors in some rows and in some columns exceed $d_A - 1$ and $d_B - 1$ respectively, (lower right). $\square$

The analysis above deals with hard decision decoding of the received matrix $\boldsymbol{y}$. It can also be proved that, instead of using a list decoder for the rows and a list decoder for the columns with decoding radii equal to or greater than $d_A - 1$ and $d_B - 1$ respectively, a bounded half the minimum distance decoder that can return the erasure symbol can also be used. We do not prove this for the hard decision case, however, and content ourselves by proving it for the case of soft decision decoding, since the proof contains the case of hard decision decoding. When soft decision decoding is used, the problem of list decoding the rows and columns can be simplified a bit by using a suboptimal list decoder, for example a GMD decoder, see Forney [3] or a Chase III decoder, [21]. By suboptimal, we mean that it is not necessary that the list returned by these decoders is actually the closest set of candidates to the received message and, even more, in some cases the list of candidates will be empty and therefore an erasure symbol, Δ is returned. These algorithms, however, can correct all errors provided that the distance of the received

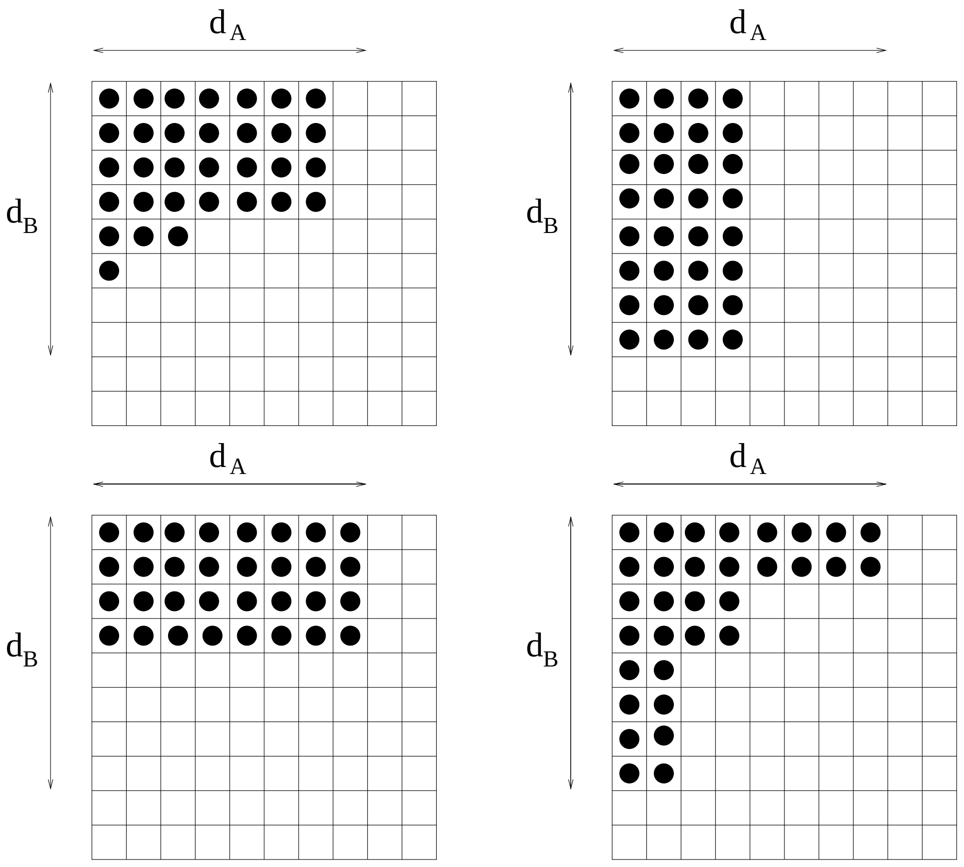


Figure 4.4: Proof of Theorem 4.3.

message does not exceed the square of half the minimum Euclidean distance of the code. We denote the Euclidean distance between two vectors by d_E and it can easily be seen that if the mapping $\{0,1\} \rightarrow \{1,-1\}$ is used to map the symbols from from $\mathbb{F}_2$ to $\mathbb{R}$, then for any code with minimum Hamming distance d, the normalized minimum squared Euclidean distance is $4d$ and the square of half the minimum Euclidean distance of this code will be d. The low complexity of such decoders, however, makes them attractive from the practical point of view. We can thus define a GMD list decoder as follows:

$$\xi_{\text{gmd}} \overset{\triangle}{=} \{\text{The set of all solutions returned by GMD algorithm}\} \cup \{\Delta\}. \tag{4.8}$$

The erasure symbol should also have a distance from the received message and, assuming that we are decoding the received message $\boldsymbol{v} \in \mathbb{R}^n$ on the $[n,k,d]$ code $\mathcal{U}$. Let $\boldsymbol{V}$ be the list of codewords returned by the GMD decoder except the erasure. I.e.,:

$$\boldsymbol{V} = \xi_{\text{gmd}}(\boldsymbol{v},\mathcal{U})\backslash\{\Delta\}. \tag{4.9}$$

We choose the following distance for the erasure symbol:

$$d_E^2(\Delta,\boldsymbol{v}) = \begin{cases} d & \text{if} \qquad \boldsymbol{V} = \{\} \\ \max\{d_E^2(\boldsymbol{V},\boldsymbol{v}) \cup \{d\}\} + \delta & \text{otherwise} \end{cases} \tag{4.10}$$

where $d_E^2(\boldsymbol{V},\boldsymbol{v})$ is the set of distances of the members of $\boldsymbol{V}$ from $\boldsymbol{v}$ and δ is a constant very small in comparison to d. The importance of the constant, δ is only shown when the real values from the channel are quantized and it can be considered to be zero when the precision is infinite.

The meaning of 4.10 in words is: if the GMD decoder fails to return any codeword, the distance of the erasure symbol will be set equal to the square of half the minimum Euclidean distance of the code. Otherwise, the distance of the erasure symbol to the received message is set to be slightly larger than the distance of the candidate that is farthest from the sent message. This is done in order to ensure that at each stage of the iterative decoder shown in Algorithm 4.1, the list of candidates for each row and each column are exhausted before giving up and trying the erasure symbol. The GMD decoder needs to have some information about the reliability of the binary symbols and we choose the following method: at each stage, l, in decoding, the reliability of each binary symbol in the matrices $\boldsymbol{a^l}$ and $\boldsymbol{b^l}$ is inversely proportional to the distance of this bit to the corresponding entry in the received matrix $\boldsymbol{y}$. I.e., if the squared Euclidean distance $d_E^2(\boldsymbol{a^l}_{i,j},\boldsymbol{y}_{i,j})$ is greater than $d_E^2(\boldsymbol{a^l}_{i,h},\boldsymbol{y}_{i,h})$, where j and h are not the same, then, the bit $\boldsymbol{a^l}_{i,j}$ is more *unreliable* than the bit $\boldsymbol{a^l}_{i,h}$. We show that this arrangement is also good and that the iterative decoder that uses GMD decoders for the rows and for the columns, can correct up to the square of half the minimum Euclidean distance of the product code. In the following discussion we assume, as shown in the discussion in Subsection 2.3.1, that the following mapping for each coordinate is used:

$$0 \quad \mapsto \quad 1$$

$$1 \quad \mapsto \quad -1 \tag{4.11}$$

If the sent codeword is $\boldsymbol{c}$, each element in the received matrix $\boldsymbol{y}$ will be a real variable equal to ± 1 and some noise of real value added to it as shown in Subsection 2.3.1. We present the following theorem:

Theorem 4.4 *Let $\mathcal{A}$ be the code represented by all $m \times n$ matrices with rows codewords in the $[n, k_A, d_A]$ code $\mathcal{A}'$ and let $\mathcal{B}$ be the code represented by all $m \times n$ matrices with columns codewords in the $[m, k_B, d_B]$ code $\mathcal{B}'$. Let the product code $\mathcal{C} = \mathcal{A} \cap \mathcal{B}$, be used in combination with Algorithm 4.1 shown in Figure 4.2 for data transmission. Also, let the list decoders for the rows and for the columns be ξ_{gmd} defined in (4.8). Then, the decoder can correct all error patterns provided that the squared Euclidean distance between the received vector and the sent message is less than the square of half the minimum Euclidean distance of the product code, i.e., when the square of the Euclidean distance between the received message and the sent message is less than $d_A d_B$.*

Proof: Without loss of generality, assume that the the sent codeword is the all-zero codeword and the received, real valued, matrix is $\boldsymbol{y}$. Assume that the first d_B rows are farthest from the corresponding rows in the sent codeword, i.e., for all $i \in \{1, \ldots, d_B\}$ and $j \in \{d_B + 1, \ldots, m\}$, the following is correct:

$$d_E^2(\boldsymbol{y}_{i,\cdot}, \boldsymbol{0}) \geq d_E^2(\boldsymbol{y}_{j,\cdot}, \boldsymbol{0}). \tag{4.12}$$

Assume that the distances of the first d_B rows in $\boldsymbol{y}$ to the all-zero n vector, are slightly less than d_A, then the row decoder will find the correct candidate for all the rows and the all-zero matrix will be obtained as the correct solution. We should keep in mind that the all-zero matrix will be missed by the column decoder only if more than $d_B/2$ rows are at a squared Euclidean distance greater than d_A, since, otherwise, the GMD decoder for the columns will always include the all-zero vector as a candidate for each of the columns. Let us begin by assuming that the list of candidates for each row consist of, at most, one candidate in addition to the erasure symbol. Now, suppose that the distance of the first row to the all-zero vector is equal to $d_A + \delta_1$, then, in the worst case, one of the first d_B rows, let it be the d_B:th row, has a distance to the all-zero vector less than $d_A - \delta_1$. Let the rest of the distances of the first d_B rows to the all-zero vector be d_A. But this means that even if the all-zero vector was not included in the list returned by GMD decoding the first row, the distance of the nearest candidate for the first row to its corresponding row in $\boldsymbol{y}$ will be greater than $d_A - \delta_1$, which is greater than the distance of the d_B:th row to the all-zero vector. Therefore, if the column decoder doesn't find the all-zero matrix first and return it as the correct answer, the iterative decoder will eventually choose the erasure symbol for the first $d_B - 1$ rows and the all-zero vector for the last row. The result of GMD decoding the columns of the previous arrangement will be the all-zero matrix. Now, let the first $\lfloor d_B/2 \rfloor$ rows have distances from the all-zero vector equal to $d_A + \delta_i$. Then, the sum of the distances of these rows to their closest candidate is greater than $d_B d_A/2 - \sum_{i=1}^{d_B/2} \delta_i$. But this means that the sum of the distances of the rest of the d_B rows, i.e., rows $d_B/2+1$ to row d_B, to the all-zero vector is less than $d_B d_A/2 - \sum_{i=1}^{d_B/2} \delta_i$. This means that the iterative decoder will eventually replace the first $d_B/2$ rows by erasures, since they are less reliable than the remaining rows, and thus, the GMD decoder for the columns will return the all-zero matrix as the result. If the list of candidates for each row can include more than one solution instead of at most one candidate in addition to the erasure symbol, then, the result will be similar, since the iterative decoder searches through all different combinations of the different candidates. $\square$

The previous theorem indicates that the performance of the iterative algorithm, in the worst case, degenerates to that of GMD decoding of the product code. The fact that the decoders for the rows and for the columns contain a list instead of a single candidate in addition to the erasure symbol, increases the probability that the sent codeword will be found among the members of the lists $\boldsymbol{A^l}$ and $\boldsymbol{B^l}$ associated with each stage l of decoding. Furthermore, the iterative algorithm is more inclined to return an answer for each row or column instead of giving up and returning the erasure symbol, which decreases the probability of there being so many erasures that the GMD decoder for each row or each column cannot handle. It is also clear that Algorithm 4.1 can correct many other patterns with a square Euclidean distance greater than the square of half the minimum Euclidean distance of the code. The number of correctable error patterns increases when the decoding radii of the list decoders for the rows and the columns are increased. For example, if the number of errors in each row and in each column is less than e_A and e_B respectively, such that, $e_A > \lfloor d_A/2 \rfloor$ and $e_B > \lfloor d_B/2 \rfloor$. Then the decoder can correct the *correctable* error patterns of this type. By correctable we mean that there does not exist any other codeword in the product code that is closer to the received matrix than the sent codeword.
If we consider the case of hard decoding of the incoming message, then, with a slight modification of Theorem 4.4, we prove that all error patterns of weight less than half the minimum distance of the product code. This is a much stronger result than that presented in Theorem 4.3 which demands that the list decoders for the rows and for the columns have a decoding radius equal to or greater than $d_A - 1$ and $d_B - 1$ respectively. The difference between the two theorems is that in Theorem 4.3, unlike Theorem 4.4, the list decoders of the rows and for the columns are not allowed to return the all erasure symbol as a member of the list of candidates for each row or each column.
It was mentioned above that another suboptimal decoder can be used, based on the Chase III decoder. Let us define this decoder as follows:

$$\xi_{\mathrm{Ch}} \overset{\triangle}{=} \{\text{The set of all solutions returned by Chase III algorithm}\} \cup \{\Delta\}, \quad (4.13)$$

where Δ is the erasure symbol explained above. We can also set the distance of the erasure symbol to the received vector in the same manner as given in (4.10). It was shown by Nilsson, see [66], that the performance of the Chase III decoder is at least as good as the GMD decoder for binary codes. Therefore, this decoder can be used as a suboptimal list decoder for the rows and columns instead and the performance will be at least as good as that shown in Theorem 4.4.

We conclude this chapter by giving an example of the the error correction capability of this algorithm:

Example 4.1 Let the $[7, 4, 3]$ Hamming code be used as the constituent code for both the rows and the columns and let the all-zero codeword be sent using coherent BPSK modulation with the mapping of (4.11). Let the received message $\boldsymbol{y}$ be as shown in Figure 4.5. The matrix to the left in the figure is the received message $\boldsymbol{y}$ and to the right in the figure is the hard decision version of the received matrix which is also the first matrix to be decoded. For the sake of simplicity, we assume that the precision of calculations is two decimal points only. We also set the constant δ in 4.1 to zero since in this example it does not affect the procedure of the algorithm. The squared Euclidean distance of the matrix $\boldsymbol{y}$ to the sent message, namely, the BPSK modulated all-zero matrix, is 8.2. This is less than the minimum distance of the product code, which means that this error pattern is correctable using a decoder that corrects up to the generalized minimum distance of the product code as explained in Subsection 2.3.2. The hard decision matrix, $\tilde{\boldsymbol{y}}$ is fed to the decoder as shown in Figure 4.1. The decoding continues as shown in

−0.02	−0.01	−0.01	1	1	1	1
−0.01	−0.01	−0.01	1	1	1	1
−0.01	−0.02	0.95	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

$\boldsymbol{y}$

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

$\tilde{\boldsymbol{y}} = \boldsymbol{b}^{\mathbf{0}}$

Figure 4.5: Using GMD decoders instead of list decoders in the algorithm

Figure 4.6 where the output of each function at each stage is shown. At iteration stage number one, the row decoder will find one candidate for each row in the matrix. The matrix $\boldsymbol{a}^{\mathbf{1}}$ is constructed from the candidates returned from the row decoder as shown in Figure 4.6. The value of $i^{(1)}$ is updated using 4.1 from 0 to 7.92. The column decoder ψ in the first iteration stage will decode each column in

$\boldsymbol{a^1}$ and will find one candidate for each column in this matrix. The matrix $\boldsymbol{b^1}$ is constructed from the candidates returned from the column decoder ψ. The value of $j^{(1)}$ is updated using 4.1 from 0 to 7.92 also. In the second iteration stage, the row decoder tries to find candidates for each row and will find the same candidates as in $\boldsymbol{a^1}$. However, since the squared Euclidean distance of the BPSK modulated form of this matrix from $\boldsymbol{y}$ is the same as $i^{(1)}$, then, the second stage is not allowed to return this matrix due to the conditions in 4.4. Therefore, the row decoder will try to replace the candidate of at least one row by another candidate for this specific row and such that it has greater Euclidean distance from the corresponding row in $\boldsymbol{y}$. Since the row decoder returns only one candidate per row, the choice will be the erasure vector for this row. Using 4.10, if the second row is replaced by the erasure vector, then, the squared Euclidean distance between the new matrix and $\boldsymbol{y}$ will be greater than that for $\boldsymbol{a^1}$. However, replacing any row other than the second row by the erasure vector, will increase the squared Euclidean distance to the received message even more. Therefor, the row decoder in the second iteration stage will choose the matrix $\boldsymbol{a^2}$. The value of $i^{(2)}$ is updated using 4.1 from 7.92 to 7.98. The column decoder ψ in the second iteration stage decodes the columns of $\boldsymbol{a^2}$ and returns with candidates for each column similar to those in $\boldsymbol{b^1}$. Since the column decoder ψ cannot choose the same matrix due to the conditions in 4.4, the decoder will choose to replace one of the columns with the erasure vector and it will choose the second column for the same reason as for the row decoder. The value of $j^{(2)}$ is updated using 4.1 from 7.92 to 7.96. The row decoder in the third iteration stage will decode the rows and try to find a matrix that has greater Euclidean distance to $\boldsymbol{y}$ than $\min(i^{(2)}, j^{(2)})$. However, choosing to replace the first column with the erasure vector will have a squared Euclidean distance from $\boldsymbol{y}$ similar to that for $\boldsymbol{b^2}$. The other alternative would be to replace the third column with the erasure vector. However, the squared Euclidean distance to $\boldsymbol{y}$, namely 8.84, will be greater than that if it chooses to erase both the first and second columns which has a Euclidean distance from $\boldsymbol{y}$ equal to 8.04. Therefore, the column decoder in the third iteration stage will choose to return the matrix $\boldsymbol{b^3}$ which has erasures in its first two columns. The value of $j^{(3)}$ is updated using 4.1 from 7.96 to 8.04. The matrix $\boldsymbol{b^3}$ is easily decoded to the all zero matrix by function ϕ at stage 4 and is returned as the correct solution.

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

$\boldsymbol{a}^1$

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

$\boldsymbol{b}^1$

1	1	0	0	0	0	0
Δ	Δ	Δ	Δ	Δ	Δ	Δ
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$\boldsymbol{a}^2$

1	Δ	0	0	0	0	0
1	Δ	0	0	0	0	0
1	Δ	0	0	0	0	0
0	Δ	0	0	0	0	0
0	Δ	0	0	0	0	0
0	Δ	0	0	0	0	0
0	Δ	0	0	0	0	0

$\boldsymbol{b}^2$

Δ	Δ	Δ	Δ	Δ	Δ	Δ
1	1	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$\boldsymbol{a}^3$

Δ	Δ	0	0	0	0	0
Δ	Δ	0	0	0	0	0
Δ	Δ	0	0	0	0	0
Δ	Δ	0	0	0	0	0
Δ	Δ	0	0	0	0	0
Δ	Δ	0	0	0	0	0
Δ	Δ	0	0	0	0	0

$\boldsymbol{b}^3$

Figure 4.6: Decoding of the the received message in Example4.1.

Chapter 5

Complexity

As mentioned earlier, the most obvious method, and the easiest to understand, to reach the maximum likelihood solution would be to search the code, one codeword at a time, in search for the codeword that is closest to the received message. It is, however, quite obvious that this is a very computationally demanding method. In practical applications, there exist certain algorithms that have very low complexity compared to maximum likelihood decoding at the price of a very poor performance. For a proposed algorithm to be practically feasible, it should have a performance superior to such algorithms with a comparable complexity. One important feature that a decoding algorithm should have is that the number of operations needed in the algorithm for decoding should not increase as an exponential function of the length of the code or one of its parameters. Therefore, we shall try to prove that the average number of operations, used by the algorithms presented in this thesis, needed to perform the decoding is a polynomial function of the length of the product code. In some cases we try to investigate the worst case and make some conclusions about the complexity needed to obtain a certain decoding performance. In investigating the complexity of Algorithm 4.1, we concentrate on the case where the list decoders used for the rows and the columns are GMD decoders for the row code and the column code respectively. This is in order to keep the complexity of the algorithm to a minimum.

5.1 Complexity of Algorithm 3.2

We start by investigating the average complexity of the basic algorithm presented in Section 3.1 when the channel is a BSC. In all the discussion below, we assume that the $[n, k_A, d_A]$ $\mathcal{A}'$ code and the $[m, k_B, d_B]$ $\mathcal{B}'$ code are used to construct the product code $\mathcal{C}$. We also define the code $\mathcal{A}$ as the code represented by all $m \times n$ matrices with rows codewords in $\mathcal{A}'$ and define the code $\mathcal{B}$ as the code represented by all $m \times n$ matrices with columns codewords in $\mathcal{B}'$. We also assume that the product code $\mathcal{C}$ is used for data transmission on. We denote the sent codeword by $\boldsymbol{x}$ and the received message by $\boldsymbol{y}$. It was explained in Chapter 3 that $\boldsymbol{y}$ is list decoded over the code $\mathcal{A}$ in a list $\boldsymbol{A}$, by list decoding the rows of $\boldsymbol{y}$ as shown in Figure 3.1. The probability that the sent codeword will be a member of the list $\boldsymbol{A}$ increases by increasing the decoding radius of the list decoder for the rows. However, increasing the decoding radius of the list decoder for the rows means increasing the complexity of the decoder. We call the event that the sent codeword is not a member of the list $\boldsymbol{A}$ by a *list error*. We shall investigate the probability of list error given the transition probability of the channel, p. We should find the decoding radius of the list decoder for the rows that guarantees a probability of list error less than a predetermined value. We restate (3.2) which gives the probability that the sent codeword is not a member of the list $\boldsymbol{A}$.

$$P_{\text{list}} = 1 - \left[\sum_{i=0}^{e_A} \binom{n}{i} p^i (1-p)^{n-i} \right]^m , \tag{5.1}$$

where e_A is the decoding radius of the list decoder for the rows and p is the transition probability of the channel. We define $\tilde{e_A}$ as the least decoding radius for the list decoder of the rows which ensures that the list error is less than a given value given the transition probability of the channel.

Example 5.1 Consider the BSC with transition probability equal to $4 \cdot 10^{-4}$. The product code that has the Hamming $[31, 26, 3]$ code as the constituent code for both the rows and the columns is used for data transmission. The value of $\tilde{e_A}$ needed so as the list error is equal to or less than 10^{-5} is equal to 2. This means that a list decoder with a decoding radius greater than half the minimum distance by one is used. If, on the other hand, the BCH $[31, 21, 5]$ code was used, a bounded half the minimum distance decoder is enough to ensure that the list error does not exceed 10^{-5}. I.e., $\tilde{e_A} = 2$ is enough to guarantee the required maximum list error.

Since the decoder looks at the elements of the list $\boldsymbol{A}$ one at a time beginning with the member that is closest to the received message, it becomes apparent that the cardinality of this list is an important factor in determining the complexity of decoding. Therefore, we study both the maximum size of the list, and its average size. The first parameter affects the maximum size of the memory needed to store the list, while the second parameter affects the average number of operations needed for decoding. The average size of the list also gives some information about how to decrease the size of storage memory without excessive degradation of the performance. We should keep in mind that the members of the list $\boldsymbol{A}$ are not actual $m \times n$ binary matrices. Rather, each member of $\boldsymbol{A}$ is a list of pointers to certain candidates for each row. In [67] Justesen and Høholdt introduced a bound on the number of codewords returned by a list decoder for MDS codes using design theory. We choose instead to restrict the case of study to binary codes and use a much simpler argument to acquire a similar bound.

Proposition 5.1 *Let $\boldsymbol{w}$ be a binary vector of length n and let ξ_e be a list decoder on the binary $[n, k, d]$ code $\mathcal{U}$. The cardinality of the list of codewords returned by decoding $\boldsymbol{w}$ is:*

$$|\xi_e(\boldsymbol{w}, \mathcal{U})| \leq \begin{cases} 1 & \text{if} \quad e \leq \lfloor \frac{d-1}{2} \rfloor \\ 1 + \binom{n}{e - \lfloor \frac{d-1}{2} \rfloor} & \text{if} \quad \lfloor \frac{d-1}{2} \rfloor < e \leq d-1 \\ 1 + \binom{n}{\lfloor \frac{d-1}{2} \rfloor} + \sum_{i=d}^{e} \binom{n}{i} & \text{otherwise} \end{cases} \tag{5.2}$$

Proof: The first part is obvious. The second part can be proved by counting the number of error vectors that can be added to the received message to obtain different codewords in $\mathcal{U}$. There can be, at most, one codeword at a distance equal to or less than $\lfloor (d-1)/2 \rfloor$ from $\boldsymbol{w}$. This accounts for the '1' in the second part of the inequality. For a codeword $\boldsymbol{c} \in \mathcal{C}$ such that,

$$\left\lfloor \frac{d-1}{2} \right\rfloor < d_H(\boldsymbol{w}, \boldsymbol{c}) \leq e,$$

there exists a vector $\boldsymbol{v}$ of weight:

$$w_H(\boldsymbol{v}) = e - \left\lfloor \frac{d-1}{2} \right\rfloor,$$

and such that:

$$w_H(\boldsymbol{w} + \boldsymbol{v}, \boldsymbol{c}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor.$$

The total number of vectors of Hamming weight $e - \lfloor (d-1)/2 \rfloor$ is:

$$\binom{n}{e - \lfloor \frac{d-1}{2} \rfloor}.$$

This proves the second part of the inequality. The third part of the inequality can be proved by first calculating the number of vectors of weight $\lfloor (d-1)/2 \rfloor$ and adding to it the total number of all vectors of weight equal to or less than d up to e. □

The proposition indicates that when the list decoder for the rows has a decoding radius greater than d_A, the minimum distance of the row code, the size of the list generated by the list decoder might grow to be quite unmanageable. Therefore, even though increasing the decoding radius of the list decoder for the rows improves

the performance of the decoder, the complexity will grow exponentially when the decoding radius is increased beyond d_A. If the decoding radius, on the other hand is kept below d_A, the complexity will be limited.
Even when the decoding radius is kept below d_A, the bound on the size of the list given in (5.2) is rather pessimistic. We show, in the following discussion, that the average size of the list is lower than this bound and quite acceptable. We start by defining the *density*, γ of a binary code $\mathcal{U}$ of dimension k and size n in the Hamming space as being:

$$\gamma(\mathcal{U}) \triangleq 2^{k-n}. \tag{5.3}$$

We define the *volume* of a sphere S_e in the Hamming space, where e is the radius, as the number of all the points contained in the sphere:

$$V(S_e) \triangleq \sum_{i=0}^{e} \binom{n}{i}. \tag{5.4}$$

We can thus say that, given an $[n, k]$ code $\mathcal{U}$ and a random vector $\boldsymbol{w}$ of size n, the average size of the list returned by a list decoder with decoding radius e is:

$$E[|\xi_e(\boldsymbol{w}, \mathcal{U})|] = \gamma(\mathcal{U})V(S_e) = 2^{k-n} \sum_{i=0}^{e} \binom{n}{i}. \tag{5.5}$$

Example 5.2 Consider the product code where the $[31, 26, 3]$ Hamming code is the constituent code for both the rows and the columns. If a list decoder for the rows with decoding radius equal to 2 was used, then the maximum number of candidates for each row will be 32. The average number of candidates for each row, on the other hand, will be 15.5. If the $[31, 21, 5]$ BCH code was used instead and the decoding radius of the list decoder for the rows is equal to 3, then the maximum number of candidates for each row will be 32 but the average number of the candidates for each row will be 4.875. If the decoding radius was equal to 4, the maximum number of candidates will be 466 while the average number of candidates for each row will be 35.6.

It should be noted that the average list length in the equation above is true only in the case that a random vector is received. The actual case is that the received vector is a codeword added to it a noise vector that has a certain probability distribution. This is a different, and much harder, problem. To estimate the average number of codewords returned by the list decoder for the actual case, we need to know the full weight distribution of the coset leaders of the code in question. We, therefore, consider (5.5) a good measure from the point of view of practical design and for comparison with other decoding algorithms.

As mentioned in Chapter 3, the received message $\boldsymbol{y}$ is list decoded by list decoding its rows and the list $\boldsymbol{A}$ is generated by the decoder. Each member of $\boldsymbol{A}$ beginning from the member that is closest to $\boldsymbol{y}$ is checked to see if it was a valid codeword until such a codeword is found. The probability that the sent codeword will be a member of the list $\boldsymbol{A}$ increases by increasing the decoding radius of the list decoder for the rows. If the sent codeword was a member of the list $\boldsymbol{A}$, then, the probability that its position in the list is near the top increases when the transition probability of the channel decreases. This means that when the transition probability of the channel is small, i.e., the channel is good, there is a better chance of finding the sent codeword by checking the first few members of the list $\boldsymbol{A}$. Otherwise, if the transition probability was high, i.e., a bad channel, then, the probability that the sent codeword will be one of the first members of the list $\boldsymbol{A}$ will be small. This also means that if the decoding of the received message demands more operations than average, then the probability that the error vector is of large Hamming weight than the average case will also be large. This gives some indication on when to stop checking the members of the list and return a decoding failure flag instead.

We have to estimate the *average* position of the sent codeword in the list $\boldsymbol{A}$, which we denote by L, as a function of p. We know that when a BSC is used with transition probability p, then, the distance of any row in the sent codeword to the corresponding row in the received matrix will have a binomial distribution. Without loss of generality, we assume that the situation that the sent codeword is the all zero codeword. Consider first the case when the number of errors in a certain row is equal to i where $i \leq e$. The all zero codeword will be contained in the sphere of radius i surrounding the received vector for this specific row. The number of codewords in the sphere of radius i surrounding the vector will be less than $\alpha_{2i}(\mathcal{A}')$ where α_j is, as defined in Chapter 3, the number of codewords of weight j or less in the code $\mathcal{A}'$. If all the codewords in the sphere of radius $2i$ were ordered according to their distance from the received vector, then, we see that the order of the all zero row vector will be less than the total number of codewords in the code $\mathcal{A}'$of weight equal to or less than $2i$. If, on the other hand, the number of errors in the specific row i was greater than the decoding radius of the list decoder, e, then, the all zero codeword will not be a member of the list of candidates for this row. The total number of candidates will be, on the average for all codewords, less than $\alpha_{2e}(\mathcal{A}')$. If we take the average over all error weights up to n we get:

$$L(p) \leq \left[\sum_{i=0}^{e} \alpha_{2i}(\mathcal{A}') P(n,i,p) + \sum_{i=e+1}^{n} \alpha_{2e}(\mathcal{A}') P(n,i,p) \right]^m , \tag{5.6}$$

where p is the transition probability of the channel and $P(n,i,p)$ is:

$$P(n,i,p) = \binom{n}{i} p^i (1-p)^{n-i}. \tag{5.7}$$

We explained in Chapter 3 that the basic decoding algorithm is comprised of three different stages, list decoding the rows of the received message, sorting the resultant codewords in a list and, last, checking each member of the list to see if it was a valid

codeword. The complexity of list decoding the rows was explained in the discussion above. As for sorting the list of matrices, then, it was shown in Chapter 3 that if we were satisfied with a list of size l, then, the sorting procedure would require $O(ml \log l)$ comparisons. Checking the members of the list of matrices in search of a valid codeword is simply done by multiplying each matrix by the parity check matrix of the column code $\mathcal{B}$. If the result was equal to $\mathbf{0}$, the matrix in question is a member of both the row code and the column code, i.e., a member of the product code $\mathcal{C}$. It can then be noticed that the number of operations needed for sorting is very small in comparison with the number of operations needed for list decoding the rows or the number of operations needed for checking the different members of the list $\boldsymbol{A}$ in search for a valid codeword. Therefore, we concentrate ourselves on the complexities introduced by list decoding the rows and the parity check operations. It was shown by Sudan in [68] that list decoding of Reed-Solomon codes can be done in polynomial time as long as the decoding radius was less than a specific value. This result was developed even more in the work of Guruswami and Sudan in [69]. Justesen and Hohøldt showed in [67] that the decoding complexity is actually associated with the number of codewords contained within a sphere of radius equal to the decoding radius of the decoder. It is quite obvious that these results can be slightly modified to apply to related codes, e.g., BCH codes. We choose, however, to follow the following simple explanation that is only correct in the case of binary codes. The explanation we use is similar to the idea behind Chase I decoding algorithm, [21].
If the decoding radius of the list decoder for the rows is equal to or less than $\lfloor (d_A - 1)/2 \rfloor$, then, we need to decode each row using a bounded half the minimum distance decoder, for example a Berlekamp-Massey decoder, see [70], for decoding cyclic codes. If, on the other hand:

$$\left\lfloor \frac{d_A - 1}{2} \right\rfloor < e_A \leq d_A - 1, \tag{5.8}$$

then, it is possible to acquire the list of codewords at a distance e_A or less from the message by deliberately adding error vectors of weight $e_A - \lfloor (d_A - 1)/2 \rfloor$ to the received message and decoding the resulting vectors using a bounded half the minimum distance decoder. Since there exists

$$\binom{n}{e_A - \lfloor \frac{d_A - 1}{2} \rfloor}$$

such vectors, list decoding of the rows requires, at most, $O(n^{e_A - \lfloor (d_A - 1)/2 \rfloor})$ decoding operations of the bounded half the minimum distance type. If a Berlekamp-Massey decoder was implemented as a part of the list decoder, then, the order of the number of operations needed for list decoding will be
$O(n^{e_A - \lfloor (d_A - 1)/2 \rfloor + 2})$ binary operations, since the Berlekamp-Massey decoder requires a number of binary operations of order $O(nd)$, see Nilsson [66] and Youzhi [71]. Other list decoders on the constituent codes of the product code can also be

implemented. One possible alternative was suggested by Forney, see [35], where the Viterbi decoding on the trellis is modified to store a list of best paths instead of storing only one, thus generating a list of codewords. The Viterbi algorithm has better performance than the bounded minimum distance decoding due to the fact that it is maximum likelihood. However, the constituent codes of a product code are usually chosen to be block codes, for example, BCH codes or Reed-Muller codes and Viterbi decoding on the trellis of such codes is much more complicated than bounded minimum distance decoding. Another variant of the list decoder can be implemented if the constituent codes were binary BCH codes. Using a modified version of Sudans algorithm for list decoding Reed-Solomon codes, see [68]. This is due to the fact that BCH codes are strongly related to Reed-Solomon codes, see MacWilliams and Sloane [8, page 294]. We, however, satisfy ourselves with the fact that for any list decoding algorithm used, for any linear code, the order of the complexity of the list decoder cannot exceed $O(n^{e_A - \lfloor (d_A-1)/2 \rfloor})$ decoding operations of the bounded minimum distance type.
Checking the members of $\boldsymbol{A}$ in search for a valid codeword can be done, as mentioned earlier, by multiplying the member of the list under investigation by the parity check matrix of the code $\mathcal{B}'$. If the result was the all-zero matrix, then, this member is a codeword in both the codes $\mathcal{A}$ and $\mathcal{B}$, i.e., a codeword in the product code $\mathcal{C}$. multiplying two binary matrices with dimensions $(m - k_B) \times m$ and $m \times n$, requires $mn(m - k_B)$ binary multiplications and $(m-1)n(m-k_B)$ binary additions. In the binary case, addition and multiplication is of comparable complexity, since, in the first case it is an *XOR* function and in the second case it is an *AND* function. We, therefore, say that the parity check operation requires at most $2mn(m - k_B)$ binary additions and binary multiplications for each element of the list $\boldsymbol{A}$.

We are now ready to summarize what we know about the complexity of decoding a received message using the basic algorithm when used in a BSC with transition probability p. Given the maximum allowed list error, we find the minimum decoding radius for the list decoder of the rows that satisfy (5.1) as follows:

$$P_{\text{list}} \leq 1 - \left[\sum_{i=0}^{\tilde{e_A}} \binom{n}{i} p^i (1-p)^{n-i} \right]^m . \tag{5.9}$$

We can therefore use Equation (5.2) to bound the maximum size of storage memory, Mem_{max}, measured in the number of binary cells, bits, needed to store the candidates for all the rows as:

$$\text{Mem}_{\text{max}} \leq \begin{cases} mn & \text{if} \quad \tilde{e_A} \leq \lfloor \frac{d_A}{2} \rfloor \\ mn \left[1 + \binom{n}{e - \lfloor \frac{d_A - 1}{2} \rfloor} \right] & \text{if} \quad \lfloor \frac{d_A}{2} \rfloor < \tilde{e_A} \leq d_A - 1 \\ mn \left[1 + \binom{n}{\lfloor \frac{d_A}{2} \rfloor} + \sum_{i=d_A}^{\tilde{e_A}} \binom{n}{i} \right] & \text{otherwise.} \end{cases} \tag{5.10}$$

The average number of bits needed for storing the candidates for all the rows, $\mathrm{Mem}_{\mathrm{avg}}$, measured in bits, is estimated to be:

$$\mathrm{Mem}_{\mathrm{avg}} = mn2^{k_A - n} \sum_{i=0}^{\tilde{e_A}} \binom{n}{i}. \tag{5.11}$$

If $\tilde{e_A}$ satisfies the following inequality:

$$\left\lfloor \frac{d_A - 1}{2} \right\rfloor < \tilde{e_A} \leq d_A - 1,$$

then, we can write the number of operations needed for list decoding all the rows, OP(List), written in terms of the number of list decodings, $\xi_{\tilde{e_A}}$, needed. The complexity $\xi_{\tilde{e_A}}$, in its turn is written in terms of the number of operations required by a bounded half the minimum distance decoder of the constituent code for the rows, OP(BMD), and will be bounded by:

$$\mathrm{OP(List)} = m\mathrm{OP}(\xi_{\tilde{e_A}}) \leq m \binom{n}{\tilde{e_A} - \lfloor \frac{d_A - 1}{2} \rfloor} \mathrm{OP(BMD)}. \tag{5.12}$$

The number of operations needed for parity check, OP(check), will, on average, be the number of operations needed for parity check on one matrix times the average order of the sent codeword in the list $\boldsymbol{A}$, L, given in (5.6). This means:

$$\begin{aligned} \mathrm{OP(check)} &= 2mn(m - k_B)L(p) \\ &\leq 2mn(m - k_B)\left[\sum_{i=0}^{e} \alpha_{2i}(\mathcal{A}')P(n,i,p) + \sum_{i=0}^{e} \alpha_{2i}(\mathcal{A}')P(n,i,p)\right]^m. \end{aligned} \tag{5.13}$$

Therefore, the average number of operations needed for decoding,
OP(Algorithm 3.2), will be bounded by:

$$\begin{aligned} \mathrm{OP(Alg.\ 3.2)} &= \mathrm{OP(check)} + \mathrm{OP(List)} \\ &\leq m \binom{n}{\tilde{e_A} - \lfloor \frac{d_A - 1}{2} \rfloor} \mathrm{OP(BMD)} + 2mn(m - k_B)L(p). \end{aligned} \tag{5.14}$$

Example 5.3 As in the previous examples, the product code $\mathcal{C}$ whose constituent codes are the $[31, 21, 5]$ BCH code is investigated. The channel is BSC with transition probability 10^{-3}. The maximum list error allowed is 10^{-6}. Using (5.9) we find that the minimum decoding radius, $\tilde{e_A}$, that is required to give a list error less than 10^{-6}, is equal to 3. By using (5.10), we find that $\mathrm{Mem}_{\mathrm{max}}$ is less than $31 \times 31 \times 32 = 30752$ bits. Using (5.11), the average size of the storage memory needed, $\mathrm{Mem}_{\mathrm{avg}}$ will be $31 \times 31 \times 4.875 = 4685$ bits. I.e., the average size of the memory needed is, approximately, 15% of the maximum size of the memory needed for storage, $\text{Mem}_{\max}$. The number of operations needed for list decoding, written as a function of the number of operations needed for bounded minimum distance decoding of the $[31, 21, 5]$ BCH code, OP(BMD), will be bounded by:

$$\text{OP(List)} \leq 31 \binom{31}{3 - \lfloor \frac{5-1}{2} \rfloor} \text{OP(BMD)} = 961 \cdot \text{OP(BMD)}.$$

The average position of the sent codeword in the list $\boldsymbol{A}$, using (5.6), will be the second member of the list and therefore the average number of operations needed for parity check will be equal to or less than or equal to $2 \times 31 \times 31 \times (31 - 21) \times 2 = 38440$ binary operations.

5.2 Complexity of Algorithm 4.1

The main advantage of the iterative algorithm, Algorithm 4.1, over the basic decoding algorithm, Algorithm 3.2, is that in the iterative algorithm, the complexity of the list decoders for the rows and the columns are set to, predetermined, fixed values. Thus, the complexity of each stage in the iteration does not exceed some value that is considered acceptable from the point of view of implementation. The maximum number of iterations is also fixed to a value such that the delay in decoding is as small as possible. There exists, however, minimum requirements on the performance of the list decoders, which means that the complexities of the list decoders for the rows and for the columns cannot be less than certain values. We proved in Theorem 4.4, that in order for the iterative decoder to be able to correct all errors up to half the minimum distance of the product code, the decoding radii of the list decoder for the rows and the list decoder for the columns, both should be greater than half the minimum distance of the row code and half the minimum distance of the column code, respectively. Furthermore, these list decoders should be able to correct both errors and erasures due to the fact that the previous stage may not be able to find any solution for some rows/columns and, thus, returns erasure symbols for the whole row/column to the next stage. We assume, therefore that the decoding radius for the list decoder for the rows is set to a fixed value, $\tilde{e_A}$, and the decoding radius for the list decoder for the columns is set to a fixed value, $\tilde{e_B}$, greater than $\lfloor (d_A - 1)/2 \rfloor$ and $\lfloor (d_B - 1)/2 \rfloor$ respectively. We then try to estimate the complexity of decoding in terms of the number of operations required by the list decoder for the rows, $\text{OP}(\xi_{\tilde{e_A}})$, and the number of operations needed by the list decoder for the columns, $\text{OP}(\xi_{\tilde{e_B}})$. We give special interest to the case when the list decoders for the rows and for the columns are GMD decoders. The analysis in this section is similar to the one in the previous section and, therefore, we will use the same notations and concepts. We assume, as we did before, that the $[n, k_A, d_A]$ code $\mathcal{A}'$ and $[m, k_B, d_B]$ code $\mathcal{B}'$ are the constituent codes of the product code $\mathcal{C}$. Let $\mathcal{A}$ be the set of all $m \times n$ matrices with rows codewords in

$\mathcal{A}'$. Also let $\mathcal{B}$ be the set of all $m \times n$ matrices with columns codewords in $\mathcal{B}'$. And, evidently, the product code $\mathcal{C}$ can be written as an intersection of $\mathcal{A}$ and $\mathcal{B}$, as was shown in Chapter 3. Let the channel used for transmission be a BSC with transition probability p and let the received matrix be $\boldsymbol{y}$. Let us imagine that two different decoders of the basic algorithm type, Algorithm 3.2, were used to decode $\boldsymbol{y}$. The first decoder decodes $\boldsymbol{y}$ on the code $\mathcal{A}$. The second decoder decodes $\boldsymbol{y}$ on the code $\mathcal{B}$. Let the lists $\boldsymbol{A}$ and $\boldsymbol{B}$ be the lists associated with the first decoder and the second decoder respectively, as explained in Chapter 3. Let $L_A(p)$ and $L_B(p)$ be the average position of maximum likelihood codeword in $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively as functions of p. We will use these notations frequently in our discussion. Using the same arguments in the discussion prior to (5.6), we can bound $L_A(p)$ and $L_B(p)$ as follows:

$$\begin{aligned} L_A(p) &\leq \left[\sum_{i=0}^{e_A} \alpha_{2i}(\mathcal{A}')P(n,i,p) + \sum_{i=e_A+1}^{n} \alpha_{2e}(\mathcal{A}')P(n,i,p)\right]^m , \\ L_B(p) &\leq \left[\sum_{i=0}^{e_B} \alpha_{2i}(\mathcal{B}')P(m,i,p) + \sum_{i=e_B+1}^{m} \alpha_{2e}(\mathcal{B}')P(m,i,p)\right]^n . \end{aligned} \tag{5.15}$$

As we did in the previous section, we investigate both the average and maximum number of operations required for decoding the received message. We also investigate the average and maximum size of storage memory needed. Recall Figure 4.1 and consider stage l in decoding $\boldsymbol{y}$ using Algorithm 4.1. Let $\boldsymbol{A}^l$ and $\boldsymbol{B}^l$ be the lists associated with functions ϕ and ψ respectively for stage l. As explained in Chapter 4, $\boldsymbol{A}^l$ and $\boldsymbol{B}^l$ are subsets of the lists $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. We start by noticing that, since for the first stage, the decoder checks the first member of the list only, then the cardinality of both $\boldsymbol{A}^1$ and $\boldsymbol{B}^1$ will not exceed 1. Similarly for stage two, the decoder only checks the members of the lists which are the second nearest codewords to $\boldsymbol{y}$ in $\boldsymbol{A}^2$ and $\boldsymbol{B}^2$, respectively. In general, we can write the following:

$$|\boldsymbol{A}^{\boldsymbol{l}}| \leq l \ \ ; \ \ |\boldsymbol{B}^{\boldsymbol{l}}| \leq l \ \ , \ \ l = 1, 2, \ldots . \tag{5.16}$$

Let the average number of iterations needed to find the maximum likelihood codeword be $I_{\text{avg}}(p)$. We conclude from (5.16) that the average number of iterations need not exceed the average order of the maximum likelihood codeword in lists $\boldsymbol{A}$ and $\boldsymbol{B}$. I.e.,

$$I_{\text{avg}}(p) \leq \min\left[L_A(p), L_B(p)\right]. \tag{5.17}$$

An important reduction in complexity is obtained if the decoding radii of the list decoder of the rows and the list decoder of the columns are less than d_A and d_B, respectively. We present the following two propositions that show this fact both for the BSC and the Euclidean channel:

Proposition 5.2 *Let the code $\mathcal{U}$ with minimum distance d be used for transmission on a BSC and let the received vector be $\boldsymbol{y}$. If $\boldsymbol{y} \in \mathcal{U}$, then the list of codewords returned by the list decoder $\xi_e(\boldsymbol{y},\mathcal{U})$ will have only one member, that is $\boldsymbol{y}$, iff $e < d$.*

The proposition above is self explanatory and its proof is a simple matter of observing that there cannot exist any codewords in a sphere of radius less than d around any codeword in the code. The following proposition applies for the Euclidean channel and is less obvious:

Proposition 5.3 *Let the code $\mathcal{U}$ with minimum distance d be used for transmission on a Euclidean channel with noise. Let the received vector after demodulation be $\boldsymbol{y}$. If $\boldsymbol{y} \in \mathcal{U}$, then the list of codewords returned by the the GMD decoder $\xi_{gmd}(\boldsymbol{y},\mathcal{U})$ will have only one member, that is $\boldsymbol{y}$.*

Proof: The GMD decoder successively erases the least reliable symbols in $\boldsymbol{y}$ up to $d-1$ symbols and decodes the resulting vector afterwards. Since $\boldsymbol{y}$ is a codeword in $\mathcal{U}$, then, erasing any combination of $d-1$ or less symbols and decoding using a Bounded Minimum Distance decoder will result with the same input vector, $\boldsymbol{y}$. □

Despite the simplicity of the propositions above, their impact on lowering the decoding complexity is great. To explain this matter, assume that the decoding radii of the list decoders of the rows and the columns are less than d_A and d_B, respectively. Let us observe a certain row in the received matrix and, without loss of generality, let this row be the first row. Imagine a situation where the function ϕ in stage l decodes the first row to a certain codeword $\boldsymbol{v}$ in $\mathcal{A}'$. If the function ψ in the same stage does not affect the first row, i.e., the first row in the matrix entering stage $l+1$ is still $\boldsymbol{v}$, then the function ϕ in stage $l+1$ cannot change the first row because any other solution will have a distance of at least d_A which is greater than the decoding radius of the list decoder of the rows. The same is true if ϕ does not affect the result for some columns from the previous stage, then, the function ψ cannot alter the solution for these columns. This means that, except for the first stage, only the rows/columns that were altered in the previous iteration need to be redecoded. Therefore, the decoder can be designed so that ϕ returns a binary vector of length n showing which columns that were affected in the process. The function ψ will use this vector to decide which columns to redecode. The function ψ in its turn will return a binary vector of length m that shows which rows that were altered. The function ϕ in the next stage will in its turn use the vector returned by ψ to decide which rows to redecode. This will lower the number of redecoded rows/columns dramatically. The second effect that Propositions 5.2 and 5.3 will have on the complexity is as follows. Let the matrix $\boldsymbol{a}$ be undergoing processing by function ϕ. If one of the rows of $\boldsymbol{a}$ was a codeword of the row code $\mathcal{A}'$, then, the GMD decoder for the rows will perform only one BMD decoding operation, e.g., Berlekamp-Massey decoding, on this row instead of $\lfloor (d_A+1)/2 \rfloor$ BMD decoding operations. Thus, the complexity of decoding is lowered if some of the errors are corrected in some rows by the previous stage. The exact impact on the complexity of decoding, however, is very hard to estimate and, therefore, we discuss it further in Chapter 6 as part of the simulations study.

We will try here to bound the complexity of Algorithm 4.1. We begin by noting that the first limitation on complexity is the maximum number of iterations allowed by the decoder. We can therefore say that if the total number of iterations was limited to I_{max}, then, the maximum number of decoded rows will be bounded by:

$$mI_{max}, \tag{5.18}$$

and the maximum number of decoded columns will be bounded by:

$$nI_{max}. \tag{5.19}$$

The average number of decoded rows or columns is much harder to compute. It is possible, however, to bound these entities for the case when the total number of errors is less than $d_A d_B/2$, (the minimum distance of the product code). The worst case condition in terms of complexity of decoding is when all the errors are located in a rectangle, up to permutation of the rows and the columns, of dimensions less than or equal to $d_B \times d_A$ as shown in Figure 5.1 below. Worst case scenario from the point of view of complexity when the total number of errors is less than half the minimum distance of the product code. All the errors are contained in a rectangle of dimensions less than or equal to $d_B \times d_A$. This is because an error in decoding the rows contributes to an error in decoding the columns and vice versa. When

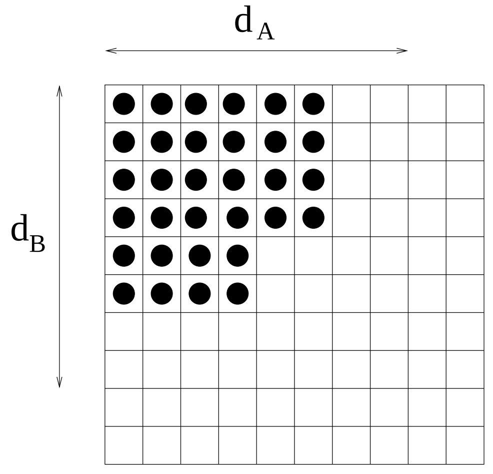


Figure 5.1: Worst case of an error pattern of weight $< \frac{d_A d_B}{2}$

decoding such an error pattern, all the rows and all the columns are decoded in the first iteration. However, starting from the second iterations, the total number of decoded rows and decoded columns at each iteration will not exceed d_A and d_B respectively. We can therefore say that, if the number of errors is less than half the minimum distance of the product code, then, the average number of decoded rows and columns will be less than:

$$m + n + (I_{max} - 1)(d_A + d_B). \tag{5.20}$$

We can therefore say that the maximum number of operations needed to decode the rows and the columns for all iterations will be less than:

$$[m\text{OP}(\xi_{\tilde{e_A}}) + n\text{OP}(\xi_{\tilde{e_B}})]I_{max}. \tag{5.21}$$

where $\text{OP}(\xi_{\tilde{e_A}})$ and $\text{OP}(\xi_{\tilde{e_B}})$ are, respectively, the complexities of list decoding a row and a column up to $\tilde{e_A}$ and $\tilde{e_B}$. When the number of errors is less than $\lfloor((d_A d_B - 1)/2)\rfloor$, then, the total number of operations will be less than:

$$[m + (I_{max} - 1)d_B]\text{OP}(\xi_{\tilde{e_A}}) + [n + (I_{max} - 1)d_A]\text{OP}(\xi_{\tilde{e_B}}). \tag{5.22}$$

If GMD decoders were used as list decoders for the rows and for the columns, then, a similar bound on the complexity can be written by replacing the complexity of the list decoders for the rows and for the columns by the complexity of the GMD decoder for the respective case.

We now turn to the problem of the storage memory needed for decoding. In a way similar to the discussion leading to the result for the average memory needed for Algorithm 4.1 given in (5.11), the average memory needed to store the candidating results for each row will be:

$$\text{Mem}^{\text{A}}_{\text{avg}} \leq mn2^{k_A - n} \sum_{i=0}^{\tilde{e_A}} \binom{n}{i}. \tag{5.23}$$

For the rows and:

$$\text{Mem}^{\text{B}}_{\text{avg}} \leq mn2^{k_B - m} \sum_{i=0}^{\tilde{e_B}} \binom{m}{i}. \tag{5.24}$$

for the columns. Since decoding the rows and the columns occur consecutively, then, it is possible to reuse the same storage space to store the result for the rows and then for the columns. Therefore, the average storage memory needed will be :

$$\text{Mem}_{\text{avg}} \leq \max(\text{Mem}^{\text{A}}_{\text{avg}}, \text{Mem}^{\text{B}}_{\text{avg}}), \tag{5.25}$$

where $\tilde{e_A}$ and $\tilde{e_B}$ are, respectively, the decoding radii of the list decoder for the rows and the list decoder for the columns. We now consider the case when GMD decoder for the row code $\mathcal{A}'$ and the column code $\mathcal{B}'$ are used instead of a list decoder. The GMD decoder for the rows returns a list of length less than or equal to $\lceil(d_A + 1)/2\rceil$

of candidating codewords from $\mathcal{A}'$ for each row. Similarly, the GMD decoder for the columns returns a list of length less than or equal to $\lceil (d_B+1)/2 \rceil$ of candidating codewords from $\mathcal{B}'$ for each column. The average memory needed for storing the intermediate results will, in this case, be:

$$\text{Mem}_{\text{avg}} \leq \max\left(m \left\lceil \frac{d_A+1}{2} \right\rceil, n \left\lceil \frac{d_B+1}{2} \right\rceil\right), \tag{5.26}$$

Even though the number of operations needed for sorting the lists at each stage of the iteration is very small in comparison to the number of operations needed for decoding the rows and the columns at each stage as shown in the previous section, we discuss below the number of operations needed for sorting. Since the size of the lists $\boldsymbol{A}^l$ and $\boldsymbol{B}^l$ is less than or equal to l as shown in (5.16), then, for stage l, the number of operations needed for sorting one of the two lists will be of the order $l \log l$. Therefore, the number of operations needed for sorting the lists will be:

$$\text{OP(list)} \leq 2 \sum_{l=1}^{I_{\max}} l \log l, \tag{5.27}$$

which is obviously much less than the number of operations needed for list decoding the rows and the columns for all the iterations.

We can now give an example that summarizes the complexity of decoding a product code using Algorithm 4.1.

Example 5.4 We investigate the product code $\mathcal{C}$ whose constituent codes are the $[31, 21, 5]$ BCH code. For a BSC channel, a BMD decoder is used for both the rows and the columns. The BMD decoder returns an erasure for each row or column that is at a distance 3 or more from all the codewords in the constituent code. Such an arrangement will ensure that all error patterns of Hamming weight equal to or less than 12 are corrected and that all burst errors occupying less than 3 rows or less than 3 columns. This was shown in Theorem 4.4 and Proposition 4.2. If the maximum number of iterations was limited to $I_{\max}$, then, the total number of operations needed to decode the rows and the columns for all the iterations will be, according to (5.21), less than:

$$2 \times 31 \times \text{OP(BMD)} I_{\max}.$$

If the number of errors was less than 12, then, the total number of operations will be less than:

$$2[31 + (I_{\max} - 1)5]\text{OP(BMD)}.$$

5.3 Outline and comparison

The previous sections gave an idea or some bounds on the complexity of decoding when using Algorithm 3.2 or Algorithm 4.1. In order to appreciate these results, we make two different comparisons.
The first comparison is between Algorithm 3.2 and the maximum likelihood Viterbi decoding on the trellis of product codes. This seems to be a reasonable comparison since Algorithm 3.2 has maximum likelihood performance when allowing the list of candidating codewords to be sufficiently large as shown in Chapter 3. In order to compare the complexity of Algorithm 3.2 with the complexity of Viterbi decoding on the trellis of product codes, we compare the size of the list $\boldsymbol{A}$ with the complexity of the trellis of the product code in question defined as the maximum number of states in the trellis, see [19]. We motivate our choice as follows: Since Viterbi decoding requires saving the data for all the potential survivor paths in the trellis. Each of these paths represents a codeword, or a part of a codeword. Therefore, the amount of data that must be stored at the widest parts of the trellis will be comparable in size to a list of $m \times n$ matrices with a length equal to the maximum number of states in the trellis. The other motivation for comparing Algorithm 3.2 with Viterbi decoding on the trellis is of practical nature. The performance of Algorithm 3.2 can be made arbitrarily close to that of maximum likelihood decoding and in fact will be maximum likelihood when removing the restriction on the decoding radius of the list decoder for the rows. It is therefore reasonable to compare the complexity of the two algorithms.

We begin by stating an upper bound and a lower bound on the maximum number of states in the minimal trellis for block codes. A minimal trellis of a code is defined as the trellis representing the code in question and such that the maximum number of states in this trellis are less than or equal to any other trellis representing the same code.
One of the well known upper bounds on the maximum number of states in the trellis of codes is the Wolf bound [47], which states the following:

Theorem 5.4 *The maximum number of states in the trellis of an $[n, k, d]$ q:ary linear code $\mathcal{C}$ cannot exceed $q^{\min\{k,n-k\}}$.*

The proof of this proposition is given in [19]. This bound is especially interesting because of its simplicity. What makes this bound even more interesting, is that even though it is an upper bound, it is tight in the case of MDS codes and very close to the actual value in many interesting codes, e.g., BCH codes. Following the discussion in Section 2.3 and the form of the Wolf bound given in (2.16). It is reasonable to say that in the case of binary product codes, the maximum number of states in the trellis will be of order:

$$2^{\min\{k_A k_B, k_A(m-k_B), k_B(n-k_A), (n-k_A)(m-k_B)\}} \tag{5.28}$$

Let $s(\mathcal{C})$ be the trellis complexity of the code $\mathcal{C}$, where we mean by trellis complexity as the base two logarithm of the maximum number of states in the minimal trellis of the code. In order to establish that Algorithm 3.2 has a lower complexity than Viterbi decoding, a lower bound on the complexity of the trellis of product codes is needed. Since the feasibility of Algorithm 3.2 is only noticed when decoding very large codes, we choose to compare the complexity of this algorithm with the asymptotic lower bound for codes on the trellis complexity given by Vardy [19]. In the case of product codes, this lower bound looks as follows:

$$\zeta(\mathcal{C}) \stackrel{\triangle}{=} \frac{s(\mathcal{C})}{mn} \geq \frac{d_A d_B k_A k_B}{m^2 n^2} = \delta_A \delta_B R_A R_B, \tag{5.29}$$

where:

$$\begin{aligned} \delta_A &\stackrel{\triangle}{=} \frac{d_A}{n}, & \delta_B &\stackrel{\triangle}{=} \frac{k_A}{m}, \\ R_A &\stackrel{\triangle}{=} \frac{k_A}{n}, & R_B &\stackrel{\triangle}{=} \frac{k_B}{m}, \end{aligned} \tag{5.30}$$

Even though there are better asymptotic bounds on the complexity of the trellis of codes, we are content with this simple bound. This bound is sufficient to prove the point we are trying to state, namely, if we can prove that under some conditions the complexity of Algorithm 3.2 is less than the lower bound given by (5.29), then, it means that Algorithm 3.2 has lower complexity than Viterbi decoding under these specific conditions.

We define the *binary entropy function* as:

$$h(p) \stackrel{\triangle}{=} -p \log_2 p - (1-p) \log_2 1 - p. \tag{5.31}$$

We need the following simple lemma:

Lemma 5.5 *Let the $[n, k_A, d_A]$ code $\mathcal{A}'$ and the $[m, k_B, d_B]$ code $\mathcal{B}'$ be the constituent codes for the product code $\mathcal{C}$. Let Algorithm 3.2 be used for decoding and let the decoding radius of the list decoder for the rows be:*

$$e_A \leq d_A - 1. \tag{5.32}$$

Then, the length of the list $\boldsymbol{A}$ of matrices generated by the decoder will be less than the maximum number of states of the minimal trellis of the code $\mathcal{C}$ provided that:

$$h\left(\frac{e_A - \lfloor \frac{d_A-1}{2} \rfloor + 1}{n}\right) \leq \frac{s(\mathcal{C})}{mn}, \tag{5.33}$$

where h is the binary entropy function and s is the trellis complexity of a code.

Proof: Using (5.2), we can say that the total number of candidates for each row can not exceed:

$$1+\binom{n}{e_A-\lfloor\frac{d_A-1}{2}\rfloor}.$$

Therefore, the length of the list $\boldsymbol{A}$ will be:

$$|\boldsymbol{A}|\leq\left[1+\binom{n}{e_A-\lfloor\frac{d_A-1}{2}\rfloor}\right]^m. \tag{5.34}$$

Taking the logarithm of the expression above and continuing as follows:

$$\begin{aligned}\log_2|\boldsymbol{A}| &\overset{a}{\leq} m\log_2\left[1+\binom{n}{e_A-\lfloor\frac{d_A-1}{2}\rfloor}\right]\\ &\overset{b}{\leq} m\log_2\binom{n}{e_A-\lfloor\frac{d_A-1}{2}\rfloor+1}\\ &\overset{c}{\leq} m\log_2\left[2^{nh\left(\frac{e-\lfloor\frac{d_A-1}{2}\rfloor+1}{n}\right)}\right]\\ &= mnh\left[\frac{e-\lfloor\frac{d_A-1}{2}\rfloor+1}{n}\right],\end{aligned} \tag{5.35}$$

where inequality sign (a) follows directly from (5.34) and inequality sign (b) follows by noticing that:

$$e_A-\lfloor\frac{d_A-1}{2}\rfloor<\frac{n}{2},$$

for all values of d_A and $e_A<d_A$. Inequality sign (c) in (5.35) follows by applying Stirling's formula, [8, p. 309]. □

The following theorem follows directly:

Theorem 5.6 *Let the codes $\mathcal{A}'$ and $\mathcal{B}'$ with parameters $[m,k_A,d_A]$ and $[m,k_B,d_B]$ respectively, be the constituent codes for the product code $\mathcal{C}$. Let Algorithm 3.2 be used for decoding $\mathcal{C}$ with a list decoder for the rows of decoding radius e_A. The length of the list of words, $\boldsymbol{A}$, to be checked in the algorithm will be less than the maximum number of states of the trellis of the product code provided that:*

$$h\left(\frac{e_A-\lfloor\frac{d_A-1}{2}\rfloor+1}{n}\right)\leq\delta_A\delta_BR_AR_B. \tag{5.36}$$

Proof: The proof follows directly from (5.29) and Lemma 5.5 □

Example 5.5 Consider the product code $\mathcal{C}$ which has the $[65536, 33551, 4000]$ extended BCH code as the constituent code for both the rows and the columns. The channel used is BSC with transition probability equal to 0.025. The decoding radius of the list decoder for the rows is chosen to be:

$$e_A = \lfloor \frac{d_A - 1}{2} \rfloor + 2 = 2001.$$

With these parameters, the cardinality of the list $\boldsymbol{A}$ will be less than the maximum number of states of the trellis of the product code $\mathcal{C}$ according to Theorem 5.6. Using (5.1) we find that the probability of one or more of the rows in the received matrix has more errors than the decoding radius of the decoder is insignificantly small at the given transition probability. If the decoding radius of the list decoder of the rows was increased beyond e_A, the cardinality of the list $\boldsymbol{A}$ will exceed the maximum number of states of the trellis of the product code.
If on the other hand, the parameters of the constituent code were $[32, 21, 6]$, then the cardinality of the list $\boldsymbol{A}$ will always be greater than the maximum number of states of the trellis of the product code except for the trivial case when the decoding radius of the list decoder is the BMD decoder.

The previous example signifies two properties of Algorithm 3.2. The first is that this algorithm will have a complexity less than Viterbi decoding only when the length of the code is sufficiently large. When the length of the code is small, the complexity of Algorithm 3.2 will be greater than that for Viterbi decoding on the trellis for the same code.

The second important feature is that the decoding radius of the list decoder for the rows should slightly exceed half the minimum distance of the row code. If the decoding radius was increased much beyond half the minimum distance of the row code, then, the length of the list $\boldsymbol{A}$ will exceed the maximum number of states of the trellis of the product code $\mathcal{C}$. The same insight is achieved by considering (5.6) where we can see that if the transition probability of the channel is sufficiently small, then, the position of the sent codeword in the list $\boldsymbol{A}$ will be less than the maximum number of states of the minimal trellis of the product code.

We now turn to Algorithm 4.1 and compare its complexity with GMD decoding of the product code. The reason behind the choice of comparison with GMD decoding of product codes is that the error correction capability of Algorithm 4.1 is at least as good as GMD decoding as was shown in Proposition 4.2 and Theorem 4.4. The error correction capability of Algorithm 4.1 increases with increasing the decoding radii of the list decoders for the rows and for the columns. We concentrate here on the case where the list decoders for the rows and columns are GMD decoders of the constituent codes.

Decoding the product code with a GMD decoder requires a GMD decoder of the constituent code for the rows and a BMD error-erasure decoder the constituent code

for the columns. A GMD decoder for the product code starts by GMD decoding the rows of the received matrix. Afterwards, the GMD decoder erases the least reliable rows, i.e., the rows furthest from the corresponding rows in the received matrix, two at a time. Each time more of the rows are erased, the columns are decoded using a BMD error-erasure decoder. This continues until the total number of erased rows is equal to $d_B - 1$. Also, if the GMD decoder of the product code finds a codeword that is at a distance less than half the squared Euclidean minimum distance of the product code, then, it stops and returns the current codeword as the correct answer.
The description above of the GMD decoder of the product code shows that when the distance between the received message and the sent codeword exceeds half the minimum distance of the product code, then, the total number of operations required will be approximately equal to:

$$m\mathrm{OP}(\xi_{GMD,A}) + n\mathrm{OP}(\xi_{GMD,A}), \tag{5.37}$$

where $\mathrm{OP}(\xi_{GMD,A})$ is the number of operations required by a GMD decoder for the rows and $\mathrm{OP}(\xi_{GMD,B})$ is the number of operations required by a GMD decoder for the columns. This is due to the fact that the decoder used for the rows is a GMD decoder and that the columns are redecoded $\lfloor (d_B - 1)/2 \rfloor + 1$ times by a BMD decoder which is exactly what is required by a GMD decoder for the columns.

We first look at the case where the number of errors is less than half the minimum distance of the product code. From (5.20) we see that if:

$$(I_{max} - 1)(d_A + d_B) \ll m + n,$$

then, the average number of decoded rows and columns will be almost that for GMD decoding of the product code.

On the other hand, if the number of errors exceed half the minimum distance of the product code, then, the total number of operations required by Algorithm 4.1 will be less I_{max} times the operations required by a GMD decoder for the product code as is obvious from (5.37) and (5.21). This, however, is a very pessimistic bound on the number of operations required by Algorithm 4.1 since it possible to find many examples of error patterns such that the total number of operations required by Algorithm 4.1 is actually *less* than that required for GMD decoding of the product code. An example of the case when Algorithm 4.1 requires a number of operations less than that required for GMD decoding of the product code is when only few of the rows and the columns require successive erasing of the least reliable bits and redecoding. The GMD decoder for the product code requires that *whole* rows are erased and *all* the columns are redecoded accordingly. Algorithm 4.1 on the other hand requires the erasure of certain bits in some of the columns. Figure 5.2 shows such an error pattern. The error pattern shown in the figure is correctable by both Algorithm 4.1 and GMD decoding. The constituent codes of the product code have minimum distance equal to 5 in this example. The black circles are errors

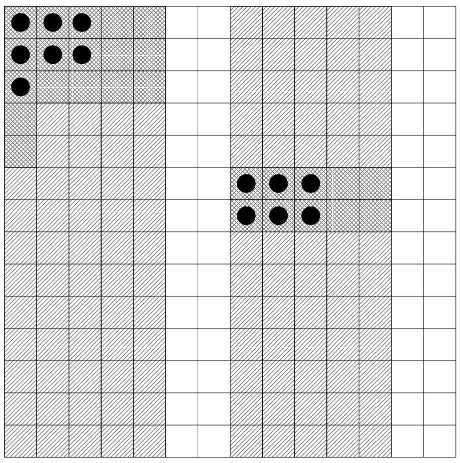


Figure 5.2: Example of a correctable error pattern

added by the transmission channel. In the first iteration, the decoder for the rows adds further errors to the rows containing errors in the message. These errors are marked by the doubly shaded region. When the decoder for the columns operates in the first iteration, only the shaded columns might need several BMD decodings while decoded by a GMD decoder and the rest of the ccolumns, the non-shaded columns, undergo only one BMD decoding stage without erasing any of the bits in the columns and redecoding. GMD decoding, on the other hand, requires that the rows containing errors are erased two at a time. After each erasure all the columns, including the non-shaded columns, are redecoded using a BMD decoder. This clearly shows that for this error pattern both algorithms correct all the errors and that the total number of operations required by Algorithm 4.1 is less than that required by GMD decoding of the product code. The difference, however, is very small and we prefer to state that the discussion above shows that the total number of operations required by the two algorithms are comparable to each other.

The memory required for storing the intermediate results in Algorithm 4.1 , however, is greater than that required for GMD decoding of product codes. The GMD decoding requires a storage memory not more than one $m \times n$ matrix to store the intermediate decoding result following each erasure of some of the rows. Another $m \times n$ matrix is needed for storing the survivor of the all the results of decoding. The average memory needed by Algorithm 4.1 for storing the intermediate results, on the other hand, requires storing the different candidates for each row and for each column in each iteration as shown in (5.26).

Chapter 6

Performance

It was shown in the previous chapter that the complexity of Algorithm 3.2 makes it uninteresting from the practical point of view. The fixed complexity of Algorithm 4.1 at each iteration, on the other hand, makes it possible for use in practical situations. We, however, were not able to analytically estimate the full error correction capability of Algorithm 4.1 and the bounds on performance of Algorithm 4.1 given in Chapter 4 are not enough argument for using the new algorithm instead of other algorithms or for using other codes than product codes. We, therefore, try to estimate the performance of this algorithm by simulation and, in certain cases, comparing it with GMD decoding of the product code. We do this comparison because it was proven in Chapter 5 that the two algorithms have comparable complexities. In all the cases below, we chose square product codes with a BCH code as a constituent code for both the rows and the columns. The decision for using the same code as the constituent code for both the rows and the columns in the product code is because it is more practical to use the same BMD decoder in order to decrease the hardware complexity. The decision to use BCH codes as constituent codes is due to the fact that the parameters of BCH codes are very close to optimal codes in terms of cardinality and minimum distance. Also, BCH codes are very well studied and have very efficient decoding algorithms, e.g., Berlekamp-Massey decoder.

In the end of this chapter, we try to obtain more detailed information about the complexity of Algorithm 4.1 for some of the cases.

6.1 Bit error probability

We performed two different simulation strategies. The first is to let Algorithm 4.1 run with as little restriction on its complexity as possible. This is done in order to see the full error correction capability of the algorithm. The drawback, however, is that for practical reasons we can only do this kind of simulation on rather simple, small sized product codes on AWGN channel with hard decoding. The reason is that when removing the restrictions on the complexity of the list decoders, this would cause the list of matrices that have to be checked at each iteration to be very long and, therefore, this method of decoding can only be used for relatively simple codes of small size.

In the other simulation strategy the complexity of Algorithm 4.1 was kept to a minimum. The most important reasons for trying to minimize the complexity of Algorithm 4.1 is to have a fair level of comparison with GMD decoding of product codes and to show the practical value of the new algorithm. We, however, tried to keep the rest of system as simple as possible. The modulation method used is coherent Binary Phase Shift Keying (BPSK) and the channel was chosen to be Additive White Gaussian Noise (AWGN) channel. Since the product codes under investigation are linear codes and since the function of the decoder is only dependent on the error pattern and not the sent codeword, the sent codeword used was always the all zero codeword. The number of samples taken at each point of simulation was chosen in a way such that a 95% confidence interval is assumed for the estimated bit error rate.

The list decoders for the rows and for the columns were chosen to be Chase III decoders. These decoders have almost the same complexity and performance as GMD decoders. This choice was made after noticing that using GMD decoders for the rows and for the columns did not change the overall results for the decoder of the product code in any way. The number of candidates stored for each row and each column was limited to two candidates at most in order to keep the overall complexity of Algorithm 4.1 to a minimum.

6.1.1 Small codes with hard decoding and high complexity list decoding of the rows and columns

As mentioned above, we first investigate the situation when the product codes used are simple and small in size but with high complexity in the decoder. The maximum number of iterations was preset to never exceed 14 iterations. The list decoder for the rows and for the columns is a variant of Chase II decoder, [21]. In each iteration, for each row or column, the bits that were flipped in the previous iteration are considered to be the bits with lowest reliability. All possible error vectors with a support contained by the positions of the lowest reliability bits are added to the corresponding row or column and then the resultant vectors are decoded. The different results of decoding a row or column are taken to be a list of candidates for this specific row or column.

Example 6.1 The first example we look at is the $[225, 121, 9]$ product code whose constituent code for both the rows and the columns is the $[15, 11, 3]$ Hamming code. The channel is chosen to be AWGN channel and hard decoding of the received symbols is assumed. The simulation results are shown in Figure 6.1. In order

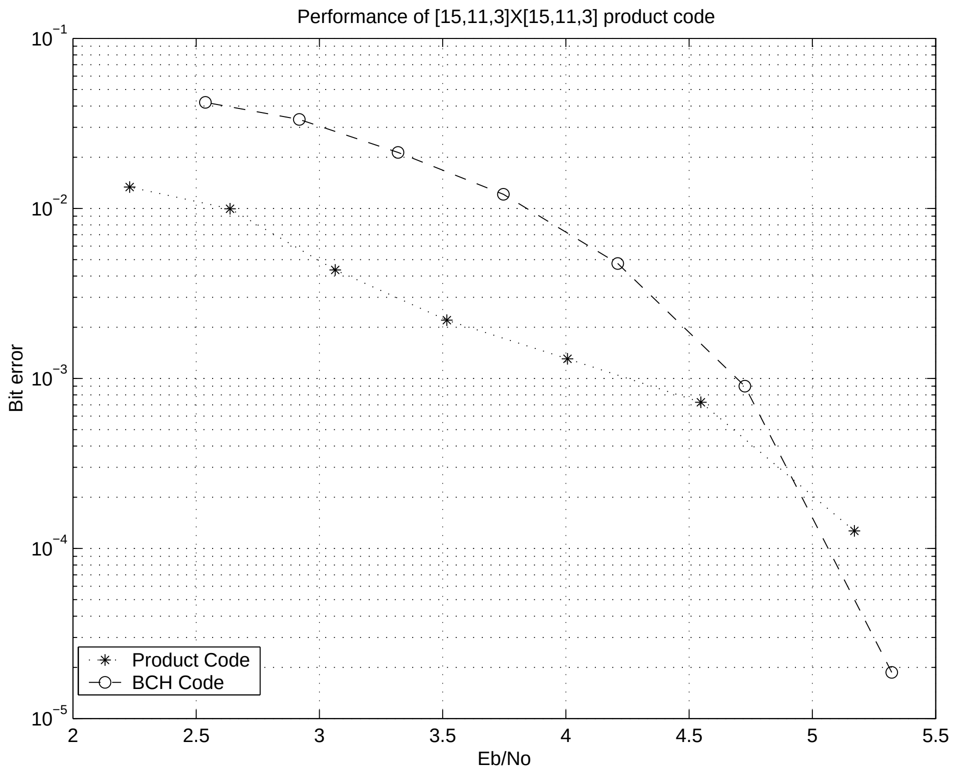


Figure 6.1: Average bit error rate of $[15, 11, 3] \times [15, 11, 3]$ product code.

to appreciate the results, we also show in the same figure the simulation results for using another code of similar rate and comparable length, namely, the binary $[255, 139, 31]$ BCH code. The chosen BCH code has a rate equal to 0.545 which is almost the same as the rate of the product code of 0.537 and thus, the two codes have the same bandwidth efficiency making the comparison fair. The BCH code is used to decode the same sequences as the product code using a Berlekamp-Massey decoder which decodes up to half the minimum distance of the code, namely, up to 15 errors. We see however that, in spite of the fact that the minimum distance of the BCH code, namely 31, is much greater than that of the product code, namely 9, the performance of the product code is better than that of the BCH code for low signal to noise ratios. When the signal to noise ratio in the channel is greater than 5 dB, the BCH code begins to outperform the product code.

There are many objections to comparing the performance of the product code to that of the BCH code above. The main objection is that it is very hard to compare the complexity of the BCH decoder to the complexity of Algorithm 4.1. It would seem like comparing apples with oranges. We, therefore, give some comments about these complexities without giving a direct proof that explicitly states the complexity for decoding the product code used in this example to be less than that for decoding the BCH code. The decoder for the $[255, 139, 31]$ BCH code is a Berlekamp-Massey decoder which incorporates polynomial operations in the Galois Field $GF(2^8)$, see Clark and Cain [72, pp. 205-214] and Blahut [73, pp. 176-204]. The iterative decoder for the $[15, 11, 3] \times [15, 11, 3]$ product code incorporates list decoders for the columns and the rows which are simply the same Hamming decoder. There are many different realizations of Hamming decoders but for the sake of comparison we mention one where it uses polynomial operations in the Galois Field $GF(2^4)$. Even though polynomial operations in $GF(2^8)$ are much more computationally demanding than $GF(2^4)$, list decoding the rows and the columns of the product code demands repeating the same decoding operation many times for the same row or column after deliberately adding a certain error pattern each time, as done in Chase II decoding. However, when the number of errors in each row or column is small, the number of flipped bits will also be small which means that the error patterns added to each row or column before decoding will be small. For example, in the first iteration when decoding the rows, each row will have at most one flipped bit. This means that, in average, when decoding the columns, there will be, at most, one flipped bit in each column. Thus, only one more error pattern is added to each column before decoding.

Another way to evaluate the performance of the product code is to compare it with the theoretical channel capacity of the equivalent binary symmetrical channel given in Equation (2.11). We see that a channel with signal to noise ratio equal to about 1.84 dB with hard decoding is equivalent to a BSC with transition probability equal to about 0.1. This channel would have a capacity equal to 0.537. Compared to the performance of the product code in the example, a signal to noise ratio of about 5 dB is required in order for the bit error ratio not to exceed 10^{-4}.

When a bounded minimum distance decoder for the $[15, 11, 3] \times [15, 11, 3]$ product code is used with hard decoding to decode the same sequences instead of the Algorithm 4.1, the results would be much worse. It is estimated that a signal to noise ratio equal to 8 dB is required in order to have bit error rate equal to or less than 10^{-4} after decoding.

Example 6.2 We now investigate the performance of a somewhat larger code than the previous one. The product code chosen has the $[31, 26, 3]$ Hamming code as the constituent code for both the rows and the columns. The rate of this code is approximately 0.7 and its length is 961 bits. The results of the simulation are shown in Figure 6.2. The constituent code of the product code for both the rows and the columns is the [31,26,3] Hamming code. As a method of comparison, the

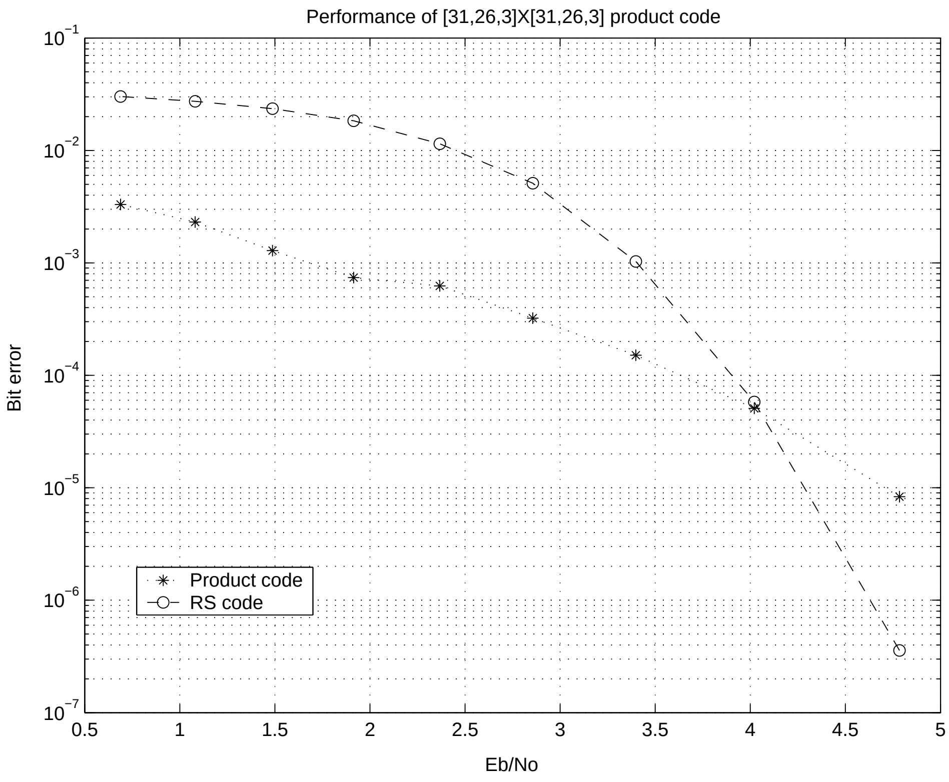


Figure 6.2: Average bit error rate for $[31, 26, 3] \times [31, 26, 3]$ product code.

same error sequences used in the simulation for the product code, were decoded using a $[127, 89, 39]$ Reed Solomon decoder over $GF(2^7)$. Each symbol in the Reed Solomon code is mapped to a binary sequence using natural mapping. Therefore, the resulting code would be a binary $[127 \times 7, 89 \times 7, 39]$ code. which has almost the same rate as the product code under investigation and comparable length. It is quite obvious from the simulation results that the product code performs better than the Reed Solomon code when the signal to noise ratio is less than about 4 dB. At signal to noise ratio equal to 4 dB, the bit error rate after decoding the product code is as low as 5×10^{-5}

As we did in the previous example we use (2.11) to find that the lowest possible signal to noise ratio required for reliable transmission with rate 0.7 is equal to 2.75 dB if AWGN channel with coherent BPSK and hard decoding is used. This can be compared to a signal to noise ratio of about 3.7 dB required to achieve a bit error rate of 10^{-4} using the $[31, 26, 3] \times [31, 26, 3]$ product code in the example in combination with the proposed iterative decoder. This means that using Algorithm 4.1 in combination with a large code performs very well, close to one dB away from the theoretical limit on the channel capacity, provided that the complexity of the list decoders for the rows and columns are not restricted.

The performance of a BMD for the product code is not included in Figure 6.2. The reason for not doing that is similar to that in the previous example which is that the performance of a BMD decoder for the product code is much worse than that for the iterative decoder. To give an example, a signal to noise ratio of at least 7.9 dB is required in order to achieve a bit error rate after decoding of about 10^{-4}. This can be compared with signal to noise ratio of about 3.7 dB that is required to achieve the same bit error rate when the iterative decoder is incorporated.

6.1.2 Large codes with soft decoding and low complexity list decoding of the rows and columns

We now turn to the other simulation strategy where we use large codes while keeping the complexity of the list decoders of the rows and columns to a minimum. The constituent codes of the product codes under investigation were chosen to be BCH codes of different rates. The modulation used on the channel is coherent BPSK and the channel is AWGN. As mentioned earlier, the complexity of the list decoders for the rows and the columns were kept to a minimum in order to limit the total complexity of the iterative decoder to a level comparable to that of GMD decoding of the same product code. Therefore, the list decoders for the rows and for the columns were chosen to be Chase III decoders which have almost the same complexity and performance as that of GMD decoders of the BCH codes. Even more decrease in the complexity of the iterative algorithm is achieved by making a further restriction on the algorithm where the number of candidates for each row or column is always limited to at most two instead of $\lfloor (d_A - 1)/2 + 1 \rfloor$ for the rows and $\lfloor (d_B - 1)/2 + 1 \rfloor$ for the columns. This way, the length of the list of matrices that should be checked at each iteration will be much lower than in the non-limited case. The total number of iterations is also limited to six iterations at most which was considered comparable to the total number of iterations needed for the GMD algorithm where it requires decoding the rows and then, at most, $\lfloor (d_B - 1)/2 + 1 \rfloor$ successive erasures of the least reliable rows and re-decoding the columns.

The maximum number of iterations was set to be six iterations at most. If the decoder cannot find any valid codeword after completing the iterations, the decoder chooses from the list of matrices in the last stage, the matrix that is closest to the received matrix and uses the information symbols in this matrix as the final result of the decoder.

Example 6.3 The first code considered is the $[16129, 14400, 9]$ product code which has the $[127, 120, 3]$ Hamming code as the constituent code for both the rows and the columns. This code has a rate equal to 0.893 and was chosen because of the simplicity of decoding Hamming codes without being trivial. Product codes

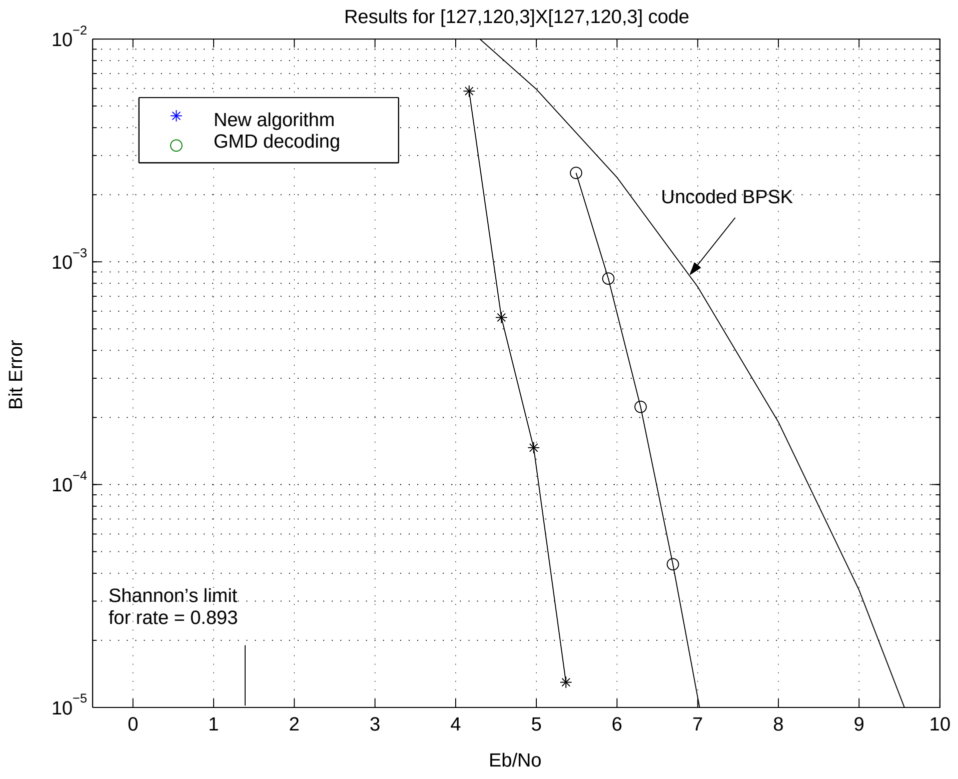


Figure 6.3: Bit error rate for $[127, 120, 3] \times [127, 120, 3]$ code on AWGN.

that have the Hamming code and their extended versions as their constituent codes are important in applications, see [74] and [22], since decoding Hamming codes is very simple compared to decoding other codes, e.g., BCH codes of minimum weight greater than 3. It should be noted that Hamming codes *are* BCH codes of minimum distance equal to 3.

The simulation results for the average bit error rate of the system when using the new algorithm is illustrated in Figure 6.3. Simulation results when using a GMD decoder for the same code is also included in the figure for comparison.

We have also indicated in the figure, using (2.14), the Shannon's limit on the lowest possible signal to noise ratio required for error free transmission for a code of rate 0.893 on band-limited channels. Using the new algorithm, a signal to noise ratio of about 5.4 dB is required in order to achieve an average bit error rate equal to 10^{-5}, which means that the performance of the product code with the new algorithm is about 4 dB away from Shannon's limit at the bit error rate mentioned above. It is also observed that the new algorithm outperforms the GMD decoder by about 1.5 dB.

Example 6.4 We now turn to investigate the performance of another product code, namely the $[16129, 12769, 25]$ which has the $[127, 113, 5]$ BCH code as the constituent code for both the rows and the columns. This code has a rate equal to 0.79 and the results for the simulation are shown in Figure 6.4. As in the previous example, the iterative algorithm has a coding gain of about 1.5 dB as compared to the case of GMD decoding the same product code. The theoretical, least possible signal to noise ratio that is required for reliable communication with a 0.79 rate code, by using (2.14), is equal to 1 dB. This means that the performance of the iterative decoder is about 3.5 dB away from the theoretical limit for a given bit error rate equal to 10^{-5}.

The previous two codes were chosen to prove the feasibility of the algorithm for large codes. In data transmission systems, the size of the packets transmitted each time is much smaller than 16 kbits. For example, the Internet protocol, see [75], has a recommended packet size ranging between 512 Bytes and 1500 Bytes. In the case of Internet protocol over the wireless channel where the resources are limited, see [76], the restriction is more severe so as not to cause high latencies in transmission. The rates of the codes used in applications are also less than that of the two codes above. The length of the error correcting code should be comparable to the size of the packets in order to avoid high latencies in transmission caused by the extra time needed for decoding. We therefore chose two more product codes with size 63×63 which has a more implementable size of about 4 kbits.

Example 6.5 Let us consider the $[3969, 2025, 49]$ product code which has the $[63, 45, 7]$ BCH code as the constituent code for both the rows and the columns. The rate of this code is equal to 0.51 and the simulation results are shown in Figure 6.5. Simulation results in the case of GMD decoding of the same code are also included in the figure for comparison. Even though the size of the product code used is much less than that in Figures 6.3 and 6.4 above, the iterative decoder is still able to outperform the GMD decoder. We notice here that a coding gain of about 1 dB, as compared to GMD decoding, is obtained. By using (2.14) we find that the theoretical, least possible signal to noise ratio required for reliable communication for a code with rate 0.51 is, a little above 0 dB. This means that the performance of this product code is about 4 dB away from the theoretical limit at a required bit error rate equal to 10^{-5}. In comparison with the results in Figure 6.4, this is a degradation of about 0.5 dB which we believe is caused by decreasing the size of

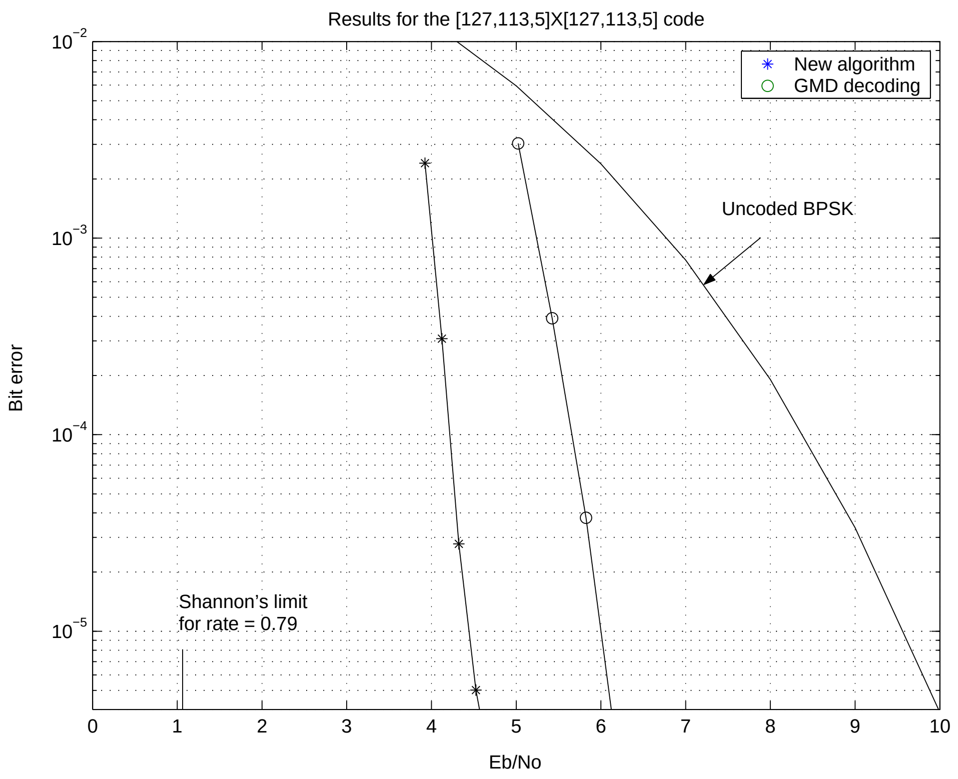


Figure 6.4: Bit error rate for $[127, 113, 5] \times [127, 113, 5]$ code on AWGN.

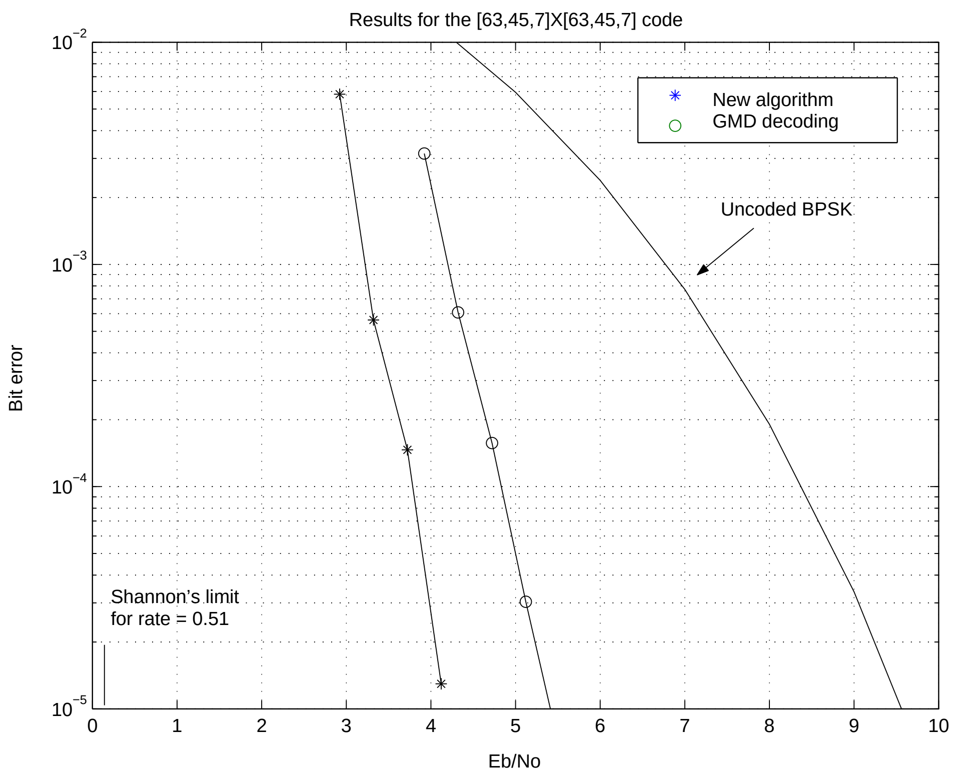


Figure 6.5: Average bit error rate for the $[63, 45, 7] \times [63, 45, 7]$ product code.

the code.

Example 6.6 As a fourth code we consider the $[3969, 1521, 81]$ product code which has the $[63, 39, 9]$ BCH code as the constituent code for both the rows and the columns. The code has the same block length as in the previous example, however, with a lower rate of 0.38. The error correction capability for this code is also higher than that of the code considered in the previous example.

Simulation results for the average bit error rate when using the iterative or GMD decoding is illustrated in Figure 6.6. As expected, with decoding algorithms, the coded system outperforms the uncoded system. We also notice that the iterative decoder outperforms the GMD decoder by about 1.5 dB at a bit error rate of 10^{-5}. By using (2.14) we find that the theoretical, least possible signal to noise

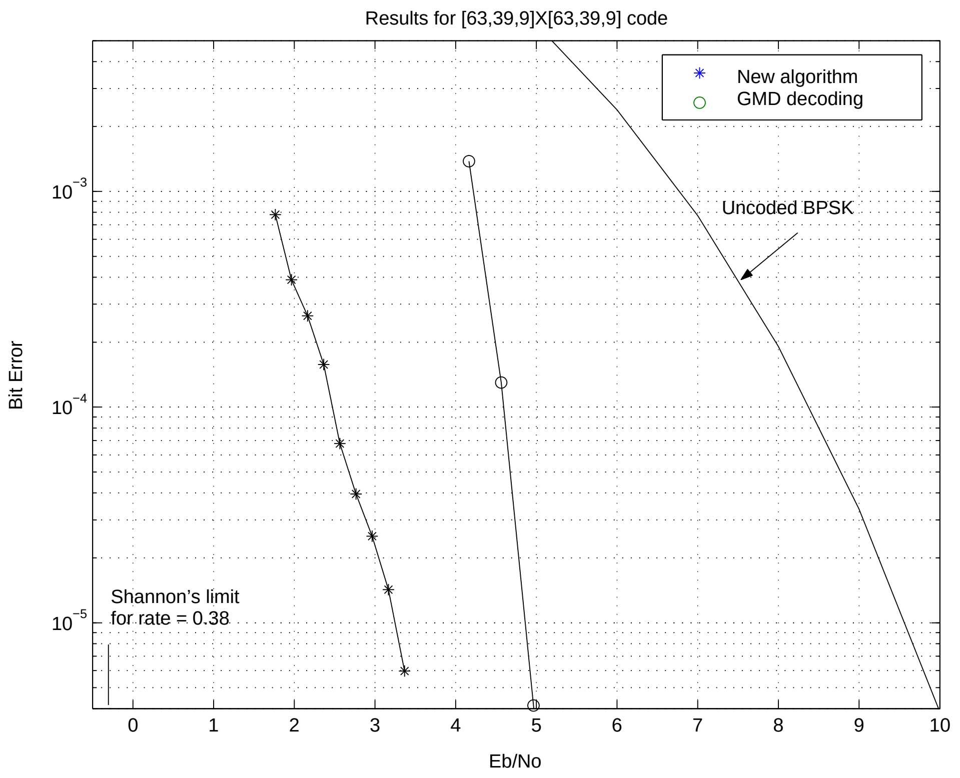


Figure 6.6: Average bit error rate for the $[63, 39, 9] \times [63, 39, 9]$ product code.

ratio required for reliable communication with rate 0.38 is, about -0.39 dB. This means that the performance of this product code is about 3.5 dB away from the theoretical limit at bit error rate equal to 10^{-5} after decoding.

We can make some general comments regarding the results of the four codes treated in the previous examples. The first conclusion is that the new algorithm always performs better than GMD decoding of the same product code for all rates and block lengths. The difference in coding gain between the new iterative algorithm and GMD decoding is about 1.5 dB. We should remember that both algorithms incorporate a Berlekamp-Massey decoder which means that the hardware used for the two algorithms is almost the same. The other conclusion we get by comparing the performance of the larger codes in Examples 6.3 and 6.4 with that of the smaller codes in Examples 6.5 and 6.6. We find that the larger codes have a performance closer to Shannon's limit by about 0.5 dB compared to that of the smaller codes. This is expected, of course, since large codes perform better than small codes in terms of error correction capability. It should be kept in mind, however, that the only modulation technique we use is BPSK. Binary modulation techniques are especially disadvantageous for high rate transmission. In high rate transmission, a multilevel modulation technique, e.g., M-ary Phase Shift Keying (MPSK) is preferred instead of BPSK.

6.2 Measured complexity

We try here to investigate more about the complexity of Algorithm 4.1 when the list decoders for the rows and for the columns are taken to be Chase III decoders. No similar investigation of the complexity of the algorithm is done for the case when the restriction on the complexity of the list decoders is removed.

We investigate mainly two aspects of complexity. The first aspect of complexity we investigate is the number of iterations needed to perform the decoding of a received sequence as a function of the signal to noise ratio of the channel. The second aspect of complexity we look into is the average number of rows or columns that have to be re-decoded at each iteration as a function of the iteration number when decoding a received sequence.

6.2.1 Number of iterations needed for decoding

In Section 5.2, we showed that when the quality of the channel improves, in terms of error probability, the number of iterations required to decode a received sequence using Algorithm 4.1 decreases. We try here to investigate some practical examples to support this claim. Two codes were chosen for investigation. One of the codes has a high rate while the other has a low rate. This was done in order to see if the rate of the code has any effect on the behavior of the decoder.

Example 6.7 As a first example for the complexity investigation we consider the product code of Example 6.4 which has the $[127, 113, 5]$ BCH code as the constituent code for both the rows and the columns. At each value of the signal to noise ratio of the channel used under the simulation, the percentage of the total number of decoded messages that required a certain number of iterations is determined and plotted in Figure 6.7. In order to be on the safe side, a message that is decoded at a certain iteration, i but only enters the following iteration, $i+1$, to be checked to see if it is a valid solution, is considered to have required $i+1$ iterations. This explains why the percentage of messages that required only one iteration is set to zero in the figure. This statistic can in fact be seen as the

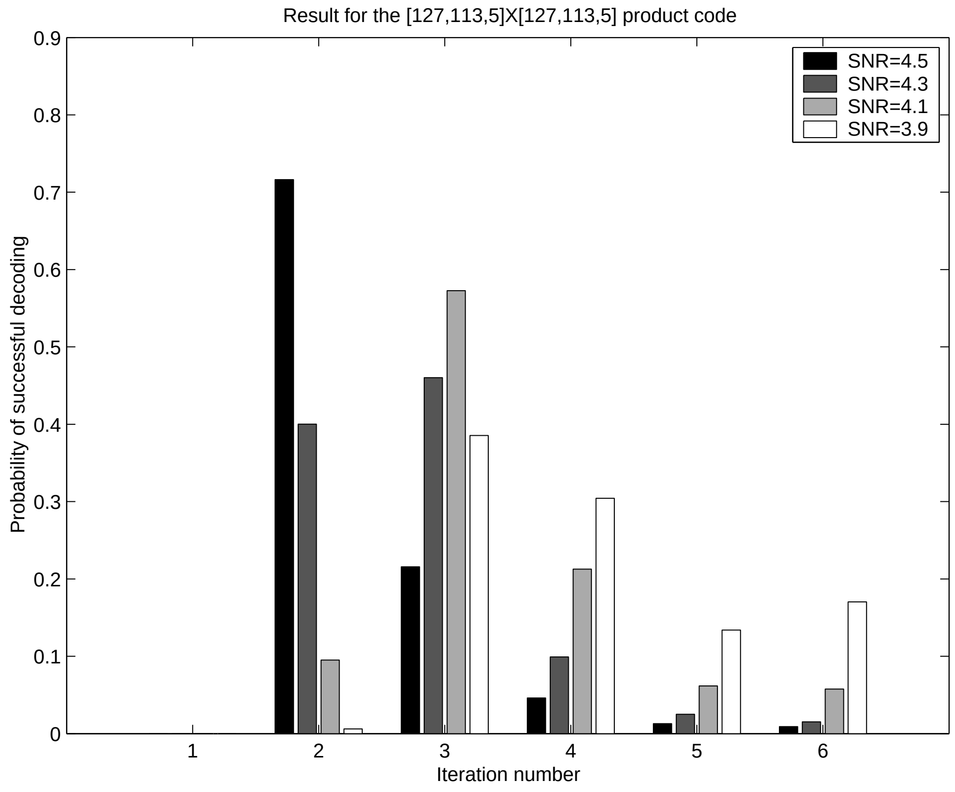


Figure 6.7: Probability of decoding in i iterations for $[127, 113, 5] \times [127, 113, 5]$ code.

probability that a given coded block is decoded in a given number of iterations. This probability, denoted $P_{sc}(l)$ is defined as:

$$P_{sc}(l) \overset{\triangle}{=} Pr\{\text{A coded block is decoded in exactly } l \text{ iterations}\}. \tag{6.1}$$

In Figure 6.7 we see very clearly that when the signal to noise ratio increases, the probability that the received message require fewer iterations increases. For example, at a signal to noise ratio equal to 4.5 dB the probability that the received message requires two iterations or less is equal to 0.72. The probability that the received block require 3 or less iterations at the same signal to noise ratio is equal to 0.94. However, for the case when the signal to noise ratio is equal to 3.9 dB, the probability that the received message require 3 or less iterations for decoding is equal to about 0.40. This means that the the algorithm has very good potential for use in practical situations since it requires persistent and continuing iterations only when the channel quality degrades. Thus the decoding delay will, in average, be rather small for good channels. Another important conclusion is that the average number of iterations seems to be a very good measure of the quality of the channel. Even more, we can conclude that if a certain message requires much more iterations than the average at the measured signal to noise ratio, then, we can say that the received message is unreliable. These indicators are very important in implementations that require some sort of channel parameter estimation, since they give an indication on how good the channel estimation is at the time.

It should be noted that in Figure 6.7, it can be seen that, for some signal to noise ratios, the percentage of number of messages that require 6 iterations is more than the percentage of messages that require 5 iterations. This is caused by the fact that we restricted the number of iterations to, at most, six iterations. Therefore, for many messages the decoding is stopped after six iterations and the best sequence at the last iteration is returned as the solution even though it may not be a valid solution.

The estimate on the probability of decoding, P_{sc}, can be used to estimate the average number of iterations, denoted $\bar{l}$, as follows:

$$\bar{l} \triangleq \sum_{l} P_{sc}(l) \cdot l \tag{6.2}$$

Figure 6.8 shows the average number of iterations needed as a function of the signal to noise ratio of the channel. The standard deviation of the number of iterations required was also estimated and was found to vary between 0.834 when the signal to noise ratio is equal to 4.5 dB to about 1.07 when the signal to noise ratio is equal to 3.9 dB.

Example 6.8 The second code we investigate is the product code which has the $[63, 39, 9]$ BCH code as the constituent code for both the rows and the columns. The probability of decoding in a given number of iterations is determined and plotted in Figure 6.9. In a manner similar to that of the $[127, 113, 5] \times [127, 113, 5]$ code in Figure 6.7, we see that for the $[63, 39, 9] \times [63, 39, 9]$ code, when the signal to noise ratio is high, the probability that a message requires fewer iterations is also high and vice versa. Comparing the results for the two codes show that the behavior of the decoder in terms of the number of iterations required is similar

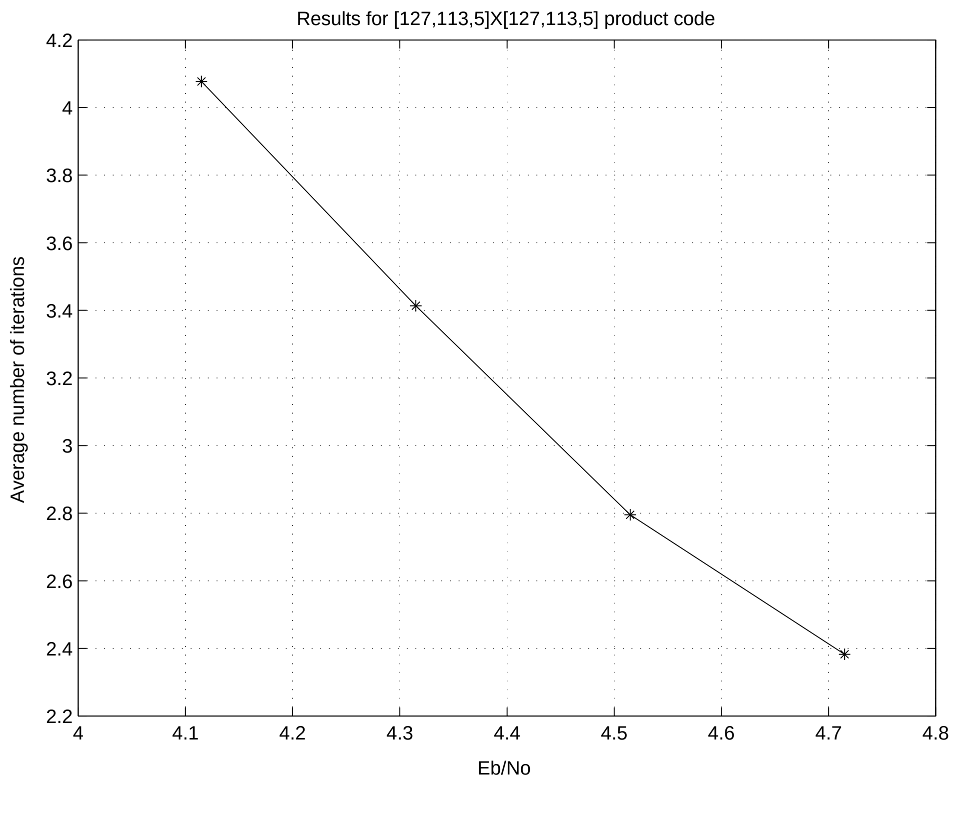


Figure 6.8: Average number of iterations for the $[127, 113, 5] \times [127, 113, 5]$ product code.

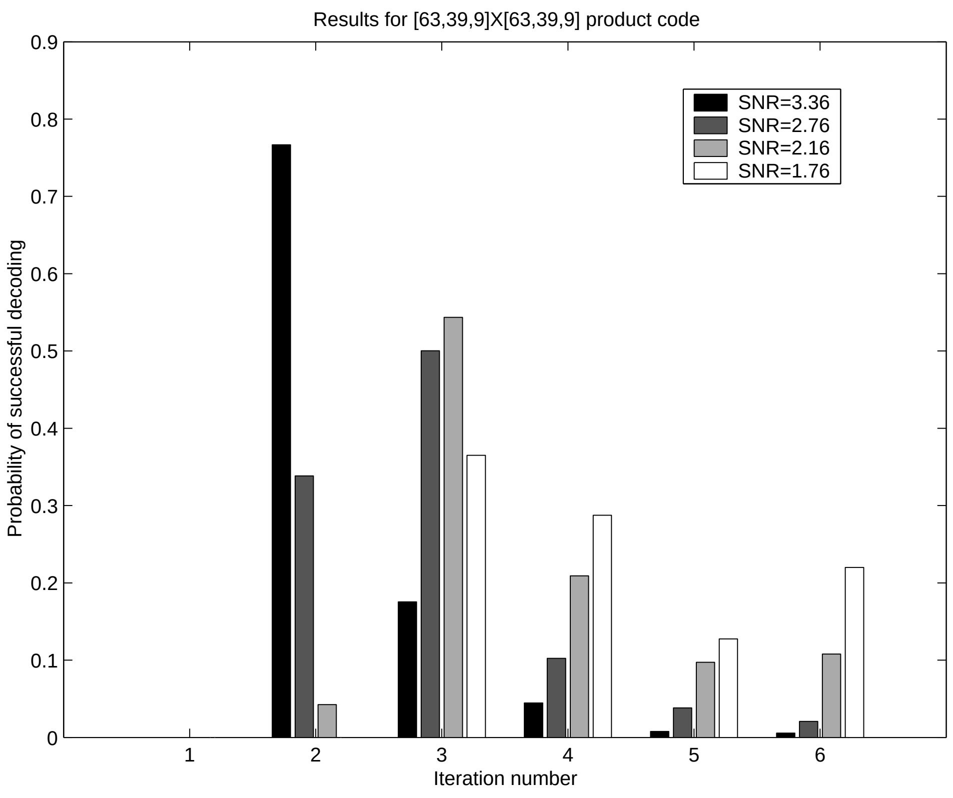


Figure 6.9: Required number of iterations for the $[63, 39, 9] \times [63, 39, 9]$ code.

for the two codes if we disregard the difference in the values of the signal to noise ratios of the channels for the two cases.

Figure 6.10 shows the average number of iterations needed as a function of the signal to noise ratio of the channel. The standard deviation of the number of

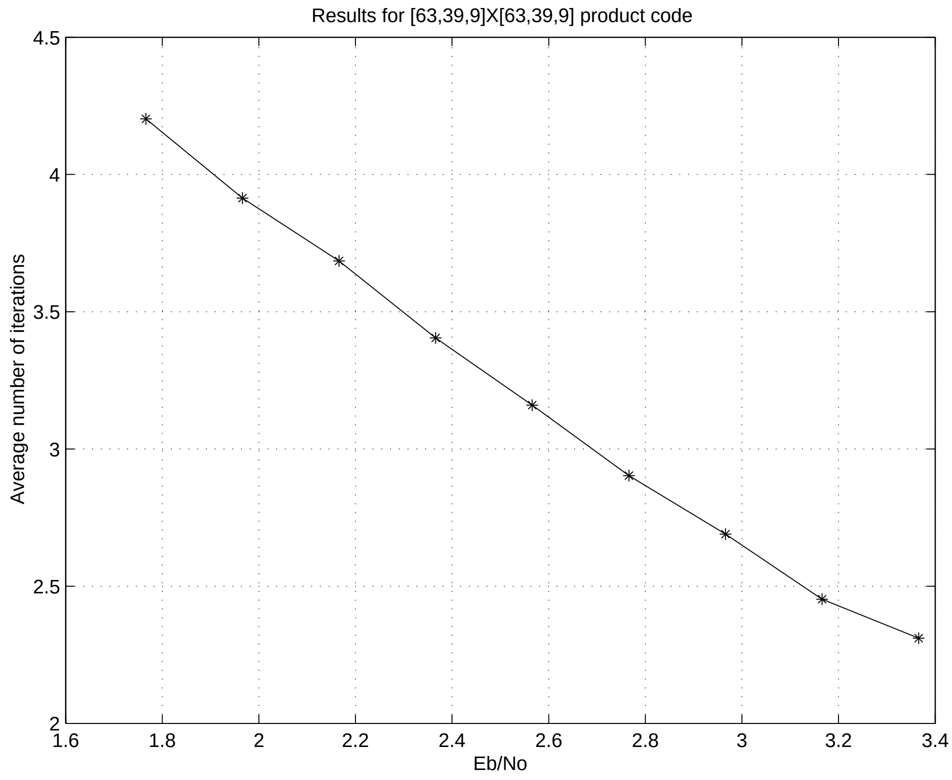


Figure 6.10: Average number of iterations for the $[63, 39, 9] \times [63, 39, 9]$ product code.

iterations required was also estimated and was found to vary between 0.646 when the signal to noise ratio is equal to 3.36 dB to about 1.15 when the signal to noise ratio is equal to 1.77 dB.

A very important remark should be made about the two previous examples. The codes studied in the examples are codes that have the same BCH code as the constituent code for the rows and for the columns. Therefore, it is plausible to say that if the decoder decodes the columns first and then the rows, the average number of iterations required will be the same. However, if the product code under study uses different constituent codes, then, the average number of required iterations might be different if the decoder decodes the columns before the rows.

However, we believe the trend of decreasing average number of required iterations with increasing signal to noise ratio will be the same.

6.2.2 Average number of decoded rows and columns at each iteration

We now consider another claim concerning the average number of decoded rows and columns at each iteration. In Section 5.2 it was argued that the number of rows and columns that require re-decoding at each iteration decreases with the number of iterations. The reason given was that, if the received message was at all decode-able, then, the row decoder and column decoder will, in average, decrease the number of errors existing in each row or column. Therefore, at each iteration there will be certain rows and columns that do not require re-decoding. We try here to verify this claim by investigating this subject for the same codes studied in 6.2.1.

We start with the product code which has the $[127, 113, 3]$ BCH code as the constituent code for both the rows and the columns used in Example 6.7. The results for this code are shown in Figure 6.11 where it shows the average number of re-decoded rows and columns for the messages that *actually* reached this iteration, i.e., the messages that require less iterations than the actual number are not included in the total number of messages. Instead of labeling the x-axis with the iteration number as an integer, the numbers show at which stage of decoding the measurement is made. So, for example, 1 in the x-axis means that this is the number *rows* decoded at the first iteration, stage ϕ in Figure 4.1, and 1.5 in the x-axis means that this is the number of *columns* decoded at the first iteration. Likewise, number two in the x-axis means that this is the number *rows* decoded at the *second* iteration and so on. I.e., the labeling in the x-axis of the graph is in half-iterations. It is assumed that in the first iteration, all rows and all columns are decoded, therefore, the graph was not drawn for stages one and two. It can be seen in Figure 6.11 that the average number of decoded rows and columns at each iteration decreases greatly with the number of iterations for all signal to noise ratios. When the signal to noise ratio is high, though, the decrease is even more drastic than for the case when the signal to noise ratio is small. This means that the demand for computational power is concentrated in the first iteration. For example, at a signal to noise ratio of 4.5 dB, the decoder decodes all 127 rows and 127 columns of the received message in the first iteration. In the second iteration the decoder decodes only about 46 rows and 5 columns. For the following iterations the number of decoded rows and columns is even less. The same argument is true when the signal to noise ratio is low. However, the number of decoded rows and columns for low signal to noise ratios tends to be higher than that for high signal to noise ratio.

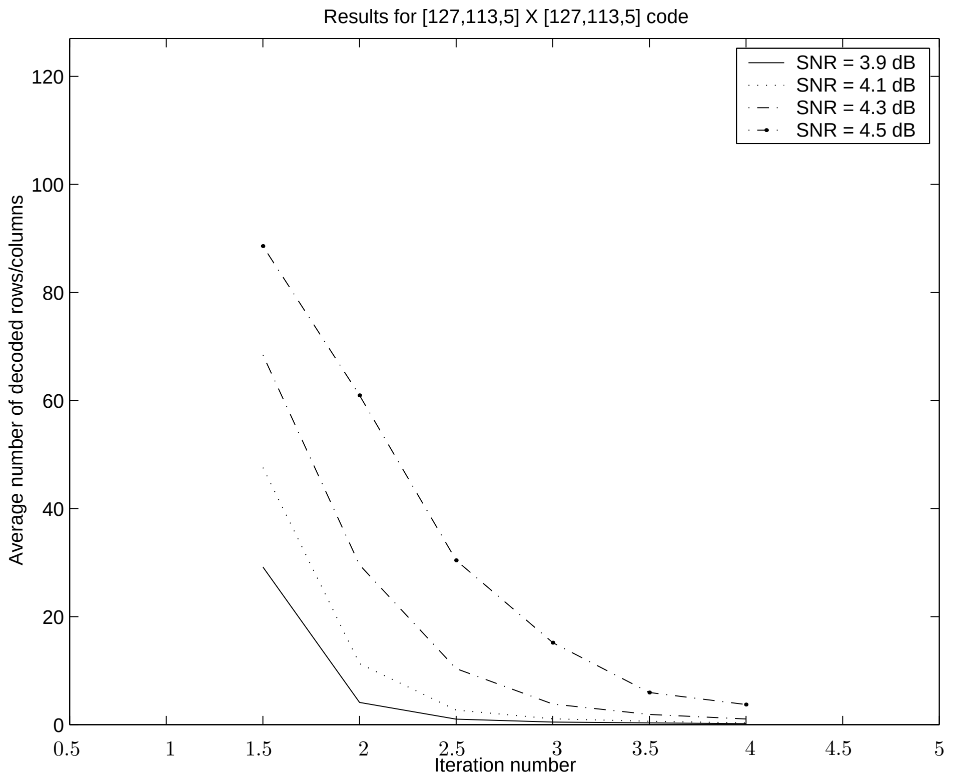


Figure 6.11: Number of re-decoded rows and columns for the $[127, 113, 5] \times [127, 113, 5]$ code.

We arrive at the same conclusions when studying product code whose constituent code for the rows and the columns is the $[63, 39, 9]$ BCH code used in Example 6.8. The results are shown in Figure 6.12. The number of rows and columns that are decoded at each iteration drop steeply after the first iteration. For example, at a signal to noise ratio equal to 3.36 dB, the decoder decodes all 63 rows and 63 columns of the received message in the first iteration. In the second iteration the decoder decodes only about 47 rows and 4 columns and so on. Looking at the results for lower signal to noise ratio, the trend is repeated.

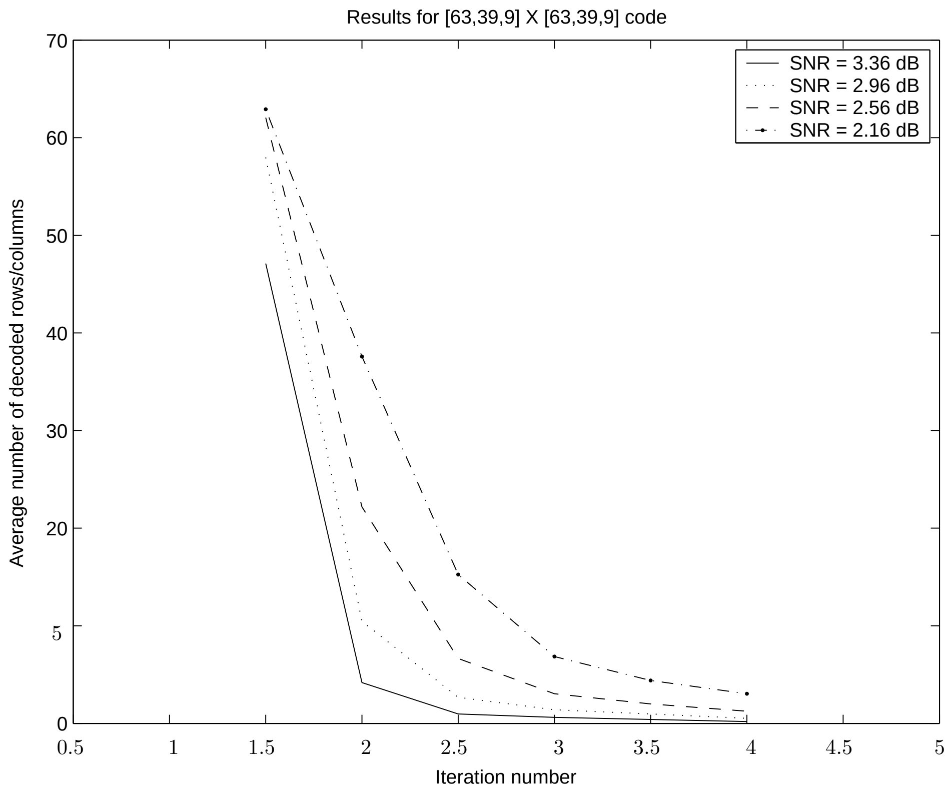


Figure 6.12: Number of re-decoded rows and columns for the $[63, 39, 9] \times [63, 39, 9]$ code.

6.3 Comments regarding the complexity of Algorithm 4.1 compared to GMD decoding of product codes

Let us consider the results for the $[63, 39, 9] \times [63, 39, 9]$ shown in Figure 6.12 for two different values of the signal to noise ratio of the channel. Consider the case when the signal to noise ratio is equal to 3.36 dB. In order for the GMD decoder to decode a received message, it starts by GMD decoding the rows of the message which requires 315 decoding operations by a BMD decoder. This is followed by 5 Berlekamp-Massey decoding operations for each of the columns using a BMD decoder after erasing some of the rows. The total number of BMD decoding operations required is, thus, equal to 630 decoding operations of the BMD type.

The iterative decoder, on the other hand, starts with GMD decoding of all the rows and the columns in the first iteration, which was shown in Section 5.3 that it requires, *at most*, the same number of BMD decoding operations as that for GMD decoding of the product code. For the second iteration, the number of rows and columns that require decoding is equal to 47 and 4 respectively. For the third and the following iteration the total number of decoded rows and columns is negligible. Therefore, the iterative decoder requires, at most:

$$(47 + 5)5 = 260,$$

BMD decoding operations more than that required for the GMD decoder of the same product code.

If we consider a lower signal to noise ration, e.g., 2.16 dB, we find, in Figure 6.12 that the total number of rows and columns that are decoded in the second iteration and above is equal to:

$$63 + 37 + 16 + 8 + 5 + 4 = 133,$$

which means that, at most, a total of $133 \times 5 = 665$ decoding operations of the BMD type, are required more than that required by the GMD decoder. This is a doubling in complexity. It is, however, up to the designer of the communication system to decide if this, possible, increase in complexity is acceptable.

We should, however, keep in mind that the estimations of the increase in complexity are only *upper bounds* on the possible increase in complexity that the iterative algorithm requires. This is because we make a pessimistic assumption that the first iteration *always* requires as many BMD decoding operations as GMD decoding of the product code, which is not necessarily true. There are many cases where the iterative decoder requires less operations than GMD decoding as shown in Section 5.3.

We tried to make another comparison between the complexity of the iterative algorithm and that of GMD decoding for certain codes in the simulations. The complexity measure used was the total number of operations required for decoding a single message. The results were that the number of operations required for decoding was quite similar for the two decoders and high signal to noise ratio the iterative decoder required less operations than the GMD decoder. However, it should be noted that this metric does not reflect the true nature of the complexities of the two algorithms. This is due to the fact that the total number of operations depends to a large extent on the effectiveness of the program used in simulations.

Chapter 7

Concluding Remarks

7.1 Conclusions

One of the main contributions of this thesis is that shows that product codes are very good candidates for use in practical communication systems. That product codes have very high error correction capability was already known before we started our study. However, this thesis shows that this high error correcting potential can be tapped with very modest complexity compared to that of GMD decoding of product codes.

In communication systems design, and especially in the case of wireless systems, the complexity of the decoder is the main obstacle in the way for using powerful codes. This is because in most cases, the designed communication system should be cheap and does not require extensive power consumption. On the other hand, using a powerful code in transmission will decrease the probability of resending data when the message is too noisy. Decreasing the number of retransmissions is an important factor for decreasing the latency and for decreasing the time of reserving the available bandwidth for each transmission which is, in its turn, very important when the bandwidth resources are limited.

Thus, for practical problems, a well chosen product code in combination with Algorithm 4.1 will be a very attractive alternative to other coding solutions. Product codes will then present the possibility of decreasing the probability of uncorrectable errors presented by the channel with a complexity comparable to that of existing alternative solutions. Furthermore, it is up to the designer to choose the list decoders incorporated in Algorithm 4.1, which means that for implementations that require very low complexity, Chase III or GMD decoders of the constituent codes can be incorporated as shown in Chapters 4 and 6. These decoding algorithms use a Berlekamp-Massey stage which is very familiar in communication system design. For implementations that require higher performance, some variant of Chase II decoder or the *ordered statistics* decoder presented by Fossorier and Lin [77] [78]. When, and if, more efficient decoding algorithms of the constituent codes are presented, they can be incorporated instead of those mentioned above.

As shown in Chapter 5, we see that the main decrease in complexity in Algorithm 4.1 is gained by using a very low complexity list decoder for the rows and the columns that is much simpler than a maximum likelihood decoder or a MAP decoder of the same codes. However, further decrease in complexity is gained due to the construction of the algorithm and the structure of product codes themselves. First, the algorithm stops its iterations of the received message as soon as it finds that the intermediate result is a codeword in the product code. This means that for good channels, only two iterations are required in average to reach an adequate solution. This is much simpler than the stop criterions that are suggested for turbo decoding and other iterative algorithms, [17]. The convergence time will be much shorter for good channels. The convergence time will also work as warning flag to the channel estimation. When the convergence time becomes longer than expected, then, this means a degradation in the quality of the channel.

The other degrease in complexity is gained by the fact that not all the rows and the columns need to be re-decoded at each iteration. This feature is due to both the algorithm and the structure of product codes. As shown in Chapter 5, the decrease in complexity compared to not using this feature is huge. For good channels, the total number of re-decoded rows and columns is only a slight fraction of the total number of rows and columns in the received message. This feature has no equivalent in other previous decoding algorithms.

From the theoretical point of view, the basic decoding algorithm for product codes proposed in Chapter 3 provide us with many results. The bound on the block error probability in Inequality (3.9) relates the block error probability to, both the characteristics of the product code under study and the complexity of the decoder. To explain that further, we point out that one of parameters used in the bound is the decoding radius of the list decoder of the rows used in the decoder. The decoding radius of the list decoder was shown in Chapter 5 to have the greatest effect on the total complexity of the algorithm. Thus, the different graphs for different decoding radii of the list decoder shown in Figure 3.7 can be considered to give an indication of the decoding gain when increasing the complexity of the decoder. Bound (3.9) also depends on the characteristics of the product code as mentioned above. However, unlike other bounds, this bound does not require detailed information about the weight distribution of the product code. Rather, a small part of the weight distribution of the constituent codes is needed and some information regarding their weight hierarchy. This is a great simplification in comparison to acquiring the weight distribution of the product code itself especially for very large product codes.

The bounds on complexity given in Chapter 5 have a different purpose than that for the bound on the block error probability in (3.9). The bounds on complexity give a more substantial measure on the complexity in terms of total number of operations required to achieve a certain performance. It also gives an indication on the size of the storage memory needed for storing the intermediate results. Some of the bounds given in Chapter 5 are bounds on the maximum value and others are bounds on the average value. These two types of bounds have different practical values. Bounds on the maximum value help the system designer to have a picture of the worst case scenario that may occur in decoding while bounds on the average value help the designer not to exaggerate in dimensioning the decoder.

Finally, the basic decoding algorithm itself has a very practical value. It can be used to develop other algorithms that are more interesting from a practical point of view. The iterative suboptimal decoder Algorithm 4.1 is an example.

7.2 Future research

Many parts of the thesis can be extended or improved in the future. We present here some of these possible future extensions and improvements. For example, the bound on the error probability of product codes in (3.9) only applies to hard decision decoding on Euclidean memoryless channels. The bound would be even more useful in practical applications if it is improved to include the case of soft decision decoding. Also, very important extensions of the bound are applying it to fading channels and modulation methods other than BPSK.

The other track of research is to study and develop efficient list decoders for the constituent codes since this part is the bottle-neck of the algorithm. Important candidates are Chase II decoding and Fossorier's decoding by ordered statistics. However, it is possible that other algorithms can be found that are more suited to the decoding algorithms presented in the thesis or to develop more efficient, i.e., less complex, decoding algorithms, with the price of slight decrease in performance.

A very important part of the future research would be to study the efficiency of product codes in combination with the new decoding algorithms in wireless channels. In order to reduce the effect of fading on wireless channels, interleaving is frequently used. The codewords in product codes have a matrix shape, therefore, product codes have an advantage over other codes since interleaving is an inherent property in their structure.

Finally, an obvious next step in the research is to generalize the algorithm to apply it on other codes especially generalized concatenated codes and multilevel codes. This is very important in order to pursue a more efficient exploitation of the resources in Euclidean channels. It was shown, for example, in [79] that multilevel codes approach the channel capacity for bandwidth-limited and power-limited Euclidean channels. Low-complexity near-optimal decoding of these codes is, however, still an open question. We believe that the iterative decoding algorithm proposed in this thesis can easily be adapted to multilevel codes and that it will perform in a manner competitive to that of existing decoding algorithms for multilevel codes. The idea is to propose specific guidelines of constructing multilevel codes such that they would be decodable by the iterative, suboptimal Algorithm 4.1 and at the same time, have a satisfactory performance in fading channels.

Appendix A

Proof of Lemma 3.2

We give below the proofs of Lemma 3.2 needed for the proof of the bound given in Theorem 3.4 on the probability of block error for product codes.

A.1 The concept of constructing rectangles

Let $\mathcal{C}$ be a product code as follows:

$$\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}' \tag{A.1}$$

Where $\mathcal{A}'$ and $\mathcal{B}'$ are an $[n, k_A, d_A]$ and an $[m, k_B, d_B]$ binary linear codes respectively. We define $\mathcal{A}$ to be the binary code represented by all $m \times n$ matrices with rows as codewords in $\mathcal{A}'$. Similarly, we define $\mathcal{B}$ to be the binary code represented by all $m \times n$ matrices with columns as codewords in $\mathcal{B}'$. Let $\boldsymbol{t} \in \mathcal{C}$ such that each row $\boldsymbol{t}_{i,\cdot}$ in this codeword is either equal to the all-zero vector or equal to $\boldsymbol{a} \in \mathcal{A}'$. I.e., all non-zero rows are similar. Also, each column $\boldsymbol{t}_{\cdot,j}$ in this codeword is either equal to the all-zero vector or equal to $\boldsymbol{b} \in \mathcal{B}'$. It is clear that the shape of the non-zero positions of this codeword will be that of a rectangle up to permutation of the rows and the columns. We call codewords with these properties *constructing rectangles* and denote them by:

$$\boldsymbol{t} = (\boldsymbol{a}, \boldsymbol{b}), \quad \boldsymbol{a} \in \mathcal{A}', \quad \boldsymbol{b} \in \mathcal{B}'.$$

The following example explains this concept:

Example A.1 Let $\mathcal{A}' = \mathcal{B}'$ be the $[7, 4, 3]$ binary Hamming code and let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be defined as above. Figure A.1 below, illustrates two different codewords in this product code. Black dots illustrate non-zero positions. In image (a) to the left of the figure, the (0111100, 0111110) constructing rectangle is shown, while image (b) to the right of the figure shows a codeword that results from adding the two rectangles, (0111100, 0111110) and (1110000, 1110000).

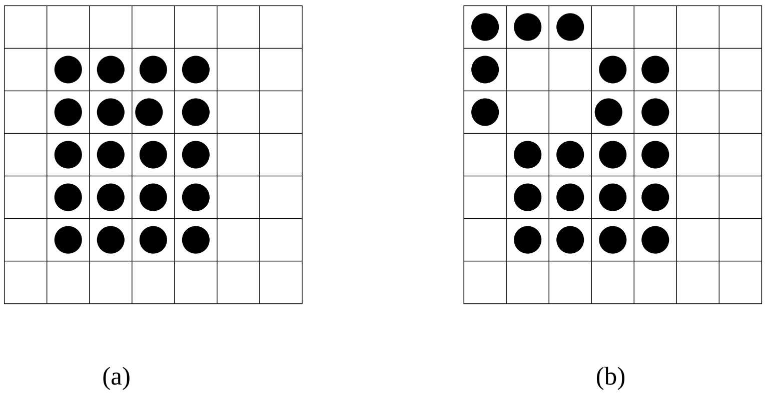


Figure A.1: Figure illustrating Example A.1.

We refer to the codewords $\boldsymbol{a}$ and $\boldsymbol{b}$ above, simply by the sides of the constructing rectangle $\boldsymbol{t}$. The Hamming weight of the sides will be referred to as the lengths of the sides. Let w_H denote the Hamming Hamming weight of a word and let d be the Hamming distance between any two words of similar length. The support of a word is the set of non-zero locations in the word and we denote this set by Supp. In the case of a word matrix, the support, of a word is the set of pairs of indices for the rows and columns of non-zero locations in the matrix. The support of a set of words is simply the union of the supports of it's members.

Let G_A and G_B be the generator matrices for the codes $\mathcal{A}'$ and $\mathcal{B}'$ respectively. Then, any codeword $\boldsymbol{c} \in \mathcal{C}$ can be written as:

$$\boldsymbol{c} = G_B^T \boldsymbol{u} G_A. \tag{A.2}$$

Where G_B^T is the transposition of the matrix G_B and $\boldsymbol{u}$ is a $k_B \times k_A$ matrix.

We are now ready for the first lemma.

Lemma A.1 *Let the codes $\mathcal{A}'$ and $\mathcal{B}'$ have the generator matrices G_A and G_B respectively. Let $\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}'$. Any codeword $\boldsymbol{c} \in \mathcal{C}$, can be described as a sum of rectangles each of which is a codeword in $\mathcal{C}$.*

Proof: Let $\omega = w_H(\boldsymbol{c})$. The codeword $\boldsymbol{c}$ can be described as in (A.2) and we continue as follows:

$$\boldsymbol{c} \quad = \quad G_B^T \boldsymbol{u} G_A,$$

$$\begin{aligned} &= G_B^T(\boldsymbol{u}_1 + \boldsymbol{u}_2 + \ldots + \boldsymbol{u}_\omega)G_A, \quad w_H(\boldsymbol{u}_i) = 1, \\ &= G_B^T\boldsymbol{u}_1 G_A + G_B^T\boldsymbol{u}_2 G_A + \ldots + G_B^T\boldsymbol{u}_\omega G_A. \end{aligned} \tag{A.3}$$

It can easily be seen that if each of the matrices $\boldsymbol{u}_i$ has only one non-zero entry, then multiplying them by any two matrices from the left and the right will result with a rectangle, which completes the proof. □

We call the set of constructing rectangles used to describe a codeword $\boldsymbol{c}$ as in (A.2) as a *constructing set*. From (A.2) we see that we can obtain $k_A k_B$ different rectangles by replacing the matrix $\boldsymbol{u}$ by matrices that have only one non-zero element. This set of rectangles forms a basis for the code $\mathcal{C}$. Since the generator matrices G_A and G_B for the codes $\mathcal{A}'$ and $\mathcal{B}'$ are not unique, then the constructing set of rectangles used in the sum describing a certain codeword is also not unique. We call the set T a *minimal* constructing set if the horizontal sides of the rectangles in T are chosen from the rows of the codeword matrix $\boldsymbol{c}$ with least weight. I.e., the horizontal sides of the rectangles in set T are chosen from the *non-zero* rows in the codeword $\boldsymbol{c}$ one by one starting with the row of least weight and then choosing the second least weight row and adding it to the set as long as long as it is independent from the chosen set. The rows in the codeword $\boldsymbol{c}$ are checked one by one and adding them to the set as long as they are independent from the existing members in the set. This procedure continues until the horizontal sides of the rectangles in set T will be a basis for the rows in the codeword $\boldsymbol{c}$.

It is clear that the cardinality of the set of all constructing rectangles is very large but in the discussion below we will only be interested in constructing rectangles that have certain dimensions, thus, the cardinality of this subset will be much lower than the total set.

Lemma A.2 *Let $\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}'$ where $\mathcal{A}'$ and $\mathcal{B}'$ are, respectively, an $[n, k_A, d_A]$ and an $[m, k_B, d_B]$ codes. Let $\boldsymbol{c} \in \mathcal{C}$ and such that:*

$$\boldsymbol{c} = \sum_{\boldsymbol{t} \in T} \boldsymbol{t}, \quad \boldsymbol{t} = (\boldsymbol{a}, \boldsymbol{b}), \quad \boldsymbol{a} \in \mathcal{A}', \quad \boldsymbol{b} \in \mathcal{B}',$$

where T is a minimal constructing set of the codeword $\boldsymbol{c}$. If the codeword $\boldsymbol{c}$ has the following property:

$$\frac{1}{m}\sum_{i=1}^{m} \sum_{(\boldsymbol{a},\,\boldsymbol{b}) \in T} |\{i\} \cap Supp(\boldsymbol{b})| \leq 2, \tag{A.4}$$

then, the support of the codeword can be divided into $|T|$ disjoint subsets such that the average of their cardinalities is greater than:

$$\frac{1}{|\boldsymbol{T}|} \sum_{(\boldsymbol{a},\boldsymbol{b}) \in \boldsymbol{T}} \frac{|\boldsymbol{a}||\boldsymbol{b}|}{2}.$$

Proof: What the lemma says in words is that if the ones in each row are shared, in average, by at most two rectangles of the constructing rectangles for the codeword, then in average, at least one half of the supports of the constructing rectangles in the codeword matrix will be non-zero. Figure A.2 below, shows a codeword that can be described as a sum of four constructing rectangles and each non-zero row in the codeword matrix is covered by at most two rectangles. Shown in (a) on the left of the figure is a codeword that is described by four constructing rectangles. Each non-zero row in the codeword is covered by at most two constructing rectangles. In (b) to the right of the figure, another codeword is shown. Some of the rows are covered by more than two constructing rectangles. However, in average the ones in each row are covered by at most two rectangles. The doubly shaded regions are the overlap between the two rectangles.

Therefore, to prove the lemma, it is sufficient to first prove it for the case when the ones in each row belong to exactly two rectangles each of which has dimensions exactly equal to $d_B \times d_A$ and then consider the average case which follows directly. Since:

$$w_H(\boldsymbol{a}) \geq d_A, \quad \forall (\boldsymbol{a}, \boldsymbol{b}) \in T,$$

then, there will be, in average, $d_A/2$ ones in every non-zero row in $\boldsymbol{c}$ contained in the support of some constructing rectangle $\boldsymbol{t} \in T$. Since there are at least d_B rows covered by each constructing rectangle, then, the average of ones existing in each rectangle is greater than $d_A d_B/2$.

Assume now that each rectangle $\boldsymbol{t}_i$ has horizontal, row, side length equal to $|\boldsymbol{a}_i|$ instead of d_A. Since these sides are the minimum weight rows in $\boldsymbol{c}$, then, each rectangle will contain at least $|\boldsymbol{a}_i| d_B/2$ ones. If the vertical sides, columns, of the rectangles were $|\boldsymbol{b}_i|$ instead of d_B then, each rectangle will contain at least $|\boldsymbol{a}_i||\boldsymbol{b}_i|/2$

If $\boldsymbol{c}$ fulfills (A.4) instead of the condition that each row is shared by at most two rectangles, then, the result will be the same because for each row shared by more than three rectangles, there exists a row that is contained by only one rectangle and each row shared by four rectangles, there exist two rows contained by only one rectangle each, and so on.

□

Lemma A.3 *Let $\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}'$ where $\mathcal{A}'$ and $\mathcal{B}'$ are, respectively, an $[n, k_A, d_A]$ and an $[m, k_B, d_B]$ codes. Let $\boldsymbol{c} \in \mathcal{C}$. Furthermore, let:*

$$k_A \leq \frac{2n}{d_A}, \tag{A.5}$$

then, the support of any codeword in $\mathcal{C}$ can be divided into disjoint subsets each of which is contained in the support of a constructing rectangle of this codeword. The average of the cardinalities of these subsets is greater than $d_A d_B/2$.

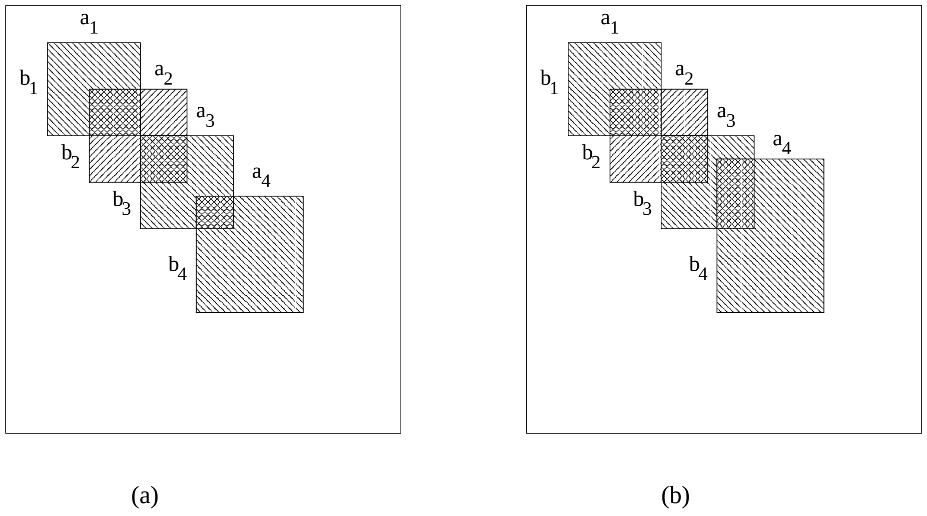


Figure A.2: Figure illustrating the proof of Lemma A.2.

Furthermore, if T is the minimal constructing set of the codeword $\boldsymbol{c}$, then the support of $\boldsymbol{c}$ can be divided into disjoint subsets. The average of the cardinalities of these subsets is greater than:

$$\frac{1}{|\boldsymbol{T}|}\sum_{(\boldsymbol{a},\boldsymbol{b})\in\boldsymbol{T}}\frac{|\boldsymbol{a}||\boldsymbol{b}|}{2}.$$

Proof: We begin by noticing that for any two constructing rectangles to overlap, it is required that the supports of the two respective sides should overlap. More formally, let $(\boldsymbol{a}_1, \boldsymbol{b}_1)$ and $(\boldsymbol{a}_2, \boldsymbol{b}_2)$, be two different constructing rectangles for the code $\mathcal{C}$ then:

$$\begin{aligned}\text{Supp}((\boldsymbol{a}_1, \boldsymbol{b}_1)\cap(\boldsymbol{a}_2, \boldsymbol{b}_2)) \neq \{\} \quad \Leftrightarrow \quad & \text{Supp}(\boldsymbol{a}_1)\cap\text{Supp}(\boldsymbol{a}_2) \neq \{\} \\ & \wedge\text{Supp}(\boldsymbol{b}_1)\cap\text{Supp}(\boldsymbol{b}_2) \neq \{\}.\end{aligned}$$

Where $\wedge$ is the *AND* symbol between two events. Therefore, we shall, in the beginning, try to prove that, under some conditions, there will, in average, be at most two constructing rectangles covering each non-zero row of the codeword matrix. Let G_A and G_B be the generator matrices of the codes $\mathcal{A}'$ and $\mathcal{B}'$ respectively and supposing that these generator matrices are used to define the constructing rectangles for the code $\mathcal{C}$ as in (A.3). Assuming that each row of G_A and G_B contains exactly d_A and d_B ones respectively. Due to Property (A.5), it is easy to see that each column in G_A will, in average, have at most two ones. If we assume that each column contains exactly two ones, then, this means that for any codeword in

$\mathcal{C}$, every non-zero column will be covered by exactly two constructing rectangles. Let $(\boldsymbol{a}_1, \boldsymbol{b}_1)$ and $(\boldsymbol{a}_2, \boldsymbol{b}_2)$, be two different constructing rectangles and let them cover a certain column. Without loss of generality, let it be the first column in the codeword matrix. I.e.,

$$1 \in \mathrm{Supp}(\boldsymbol{b}_1) \cap \mathrm{Supp}(\boldsymbol{b}_2).$$

If $\boldsymbol{a}_1 \neq \boldsymbol{a}_2$, then, the ones in the first column will be shared by these two rectangles. Otherwise, if $\boldsymbol{a}_1 = \boldsymbol{a}_2$, then, the first column will be void of ones, which contradicts the assumption that the first row is not the all-zero vector. The same argument applies for all the non-zero columns in any codeword and, thus, any codeword in $\mathcal{C}$ will have Property (A.4) which proves the lemma by using Lemma A.2. If the number of ones in each column of G_A is not restricted to two but the average number of ones in each column is less or equal to two, then, if there were three rectangles covering the same column there has to exist a column that is covered by only one rectangle and so forth. If the weights of the rows in G_A and G_B were greater than d_A and d_B respectively, then, even though (A.4) might not be fulfilled anymore, the result will still be correct except that the constructing rectangles will have dimensions greater than $d_B \times d_A$. $\square$

We now need to present the following definition. For any code $\mathcal{D}$ with parameters $[n, k, d]$, the *generalized Hamming weights* of the code $\mathcal{D}$, see [56], are defined as:

$$d_i(\mathcal{D}) \overset{\triangle}{=} \min_{E} |\mathrm{Supp}(E)|, \quad i = 1, 2, \ldots, k, \tag{A.6}$$

where the minimum is taken over all linear sub-code $E \subseteq \mathcal{D}$ that have dimension i. It is clear that $d_1(\mathcal{D}) = d(\mathcal{D})$, i.e., the minimum distance of the code. Let $\mathcal{A}'^{\perp}$ be the dual code of the code $\mathcal{A}'$. We define the sequence $d_1^{\perp}, d_2^{\perp}, \ldots, d_{k_A}^{\perp}$ to be the generalized Hamming weights of the dual code. Let $\mathcal{A}^*$ be a $[n^*, k_A^*, d_A^*]$ code obtained by shortening some of the coordinates of $\mathcal{A}'$, see MacWilliams and Sloane [8][page 29]. We define $r(\mathcal{A}^*)$ as an integer such that:

$$d_{r+1}^{\perp} \geq n - |\mathrm{Supp}(\mathcal{A}^*)|, \quad d_r^{\perp} < n - |\mathrm{Supp}(\mathcal{A}^*)|, \tag{A.7}$$

The lemma below follows directly from the previous discussion:

Lemma A.4 *Let $\mathcal{A}'$ be a $[n, k_A, d_A]$ linear code and let $\mathcal{A}^*$ be the $[n^*, k_A^*, d_A^*]$ code obtained by shortening some of the coordinates in the code $\mathcal{A}$. Let $d_1^{\perp}, d_2^{\perp}, \ldots, d_{k_A}^{\perp}$ be, as defined above, the generalized Hamming weights of the dual code. Then $d_A^* \geq d_A$ and:*

$$n - n^* \leq k_A - k_A^* - r(\mathcal{A}^*), \tag{A.8}$$

Proof: Let H_A be the parity check matrix of the code $\mathcal{A}'$. Shortening the I coordinates in $\mathcal{A}'$ is equivalent to deleting the columns in H_A corresponding to those coordinates. Let us denote the new parity check matrix by H_A^*. If the number of deleted columns, $|I|$ is less than $d_1^{\perp}$ i.e., the minimum distance of the dual code, then, H_A^* will have rank $n - k_A$. However, if the number of deleted columns exceed $d_1^\perp$, then it is possible that one of the rows in H_A^* will be zero which means that the rank of H_A^* will be $n - k_A - 1$. This means that the shortened code will have parameters $[n - |I|, k_A - |I| + 1, d_A^*]$, where $d_A^* \geq d_A$. In a similar manner, if the number of deleted columns is greater than $d_{r-1}^\perp$ but less than $d_r^\perp$, then the dimension of the shortened code can be:

$$k_A - |I| \leq k_A^* \leq k_A - |I| + r(\mathcal{A}^*).$$

The lemma is proved by noticing that $|I| = n - n^*$ and taking the right hand side of the inequality above. □

Consider a codeword $\boldsymbol{c} \in \mathcal{A}^* \otimes \mathcal{B}'$ such that the number of independent rows in $\boldsymbol{c}$ is equal to k_A^*. It is clear that:

$$|\mathrm{Supp}(\mathcal{A}^*)| \leq \frac{w_H(\boldsymbol{c})}{d_B}. \tag{A.9}$$

We are now ready to present the following theorem which, roughly explained, states that, if the Hamming weight of a codeword is less than a certain value, then, we can find many rectangles in the codeword with limited length sides such that the average of their Hamming weights is greater than $d_A d_B/2$. Those rectangles cover all the non-zero locations in the codeword.

Theorem A.5 *Let $\mathcal{C} = \mathcal{A}' \otimes \mathcal{B}'$ where $\mathcal{A}'$ and $\mathcal{B}'$ are, respectively, an $[n, k_A, d_A]$ and an $[m, k_B, d_B]$ codes. Let $\boldsymbol{c} \in \mathcal{C}$ such that:*

$$\boldsymbol{c} = \sum_{\boldsymbol{t} \in T} \boldsymbol{t}, \quad \boldsymbol{t} = (\boldsymbol{a}, \boldsymbol{b}), \quad \boldsymbol{a} \in \mathcal{A}', \quad \boldsymbol{b} \in \mathcal{B}',$$

where T is a minimal constructing set of the codeword $\boldsymbol{c}$. Let $\mathcal{A}^*$ be the code obtained by shortening $\mathcal{A}'$ in the coordinates where $\boldsymbol{c}$ has all-zero columns. If

$$w_H(\boldsymbol{c}) \leq \omega_A \overset{\triangle}{=} \frac{d_A d_B}{d_A - 2}(n - k_A - r'), \tag{A.10}$$

where r' is an integer such that:

$$d_{r'+1}^\perp \geq n - \frac{\omega_A}{d_B}, \quad d_{r'}^\perp < n - \frac{\omega_A}{d_B}, \tag{A.11}$$

then, the support of the codeword $\boldsymbol{c}$ can be divided into disjoint subsets each of which is contained in the support of a constructing rectangle of this codeword and such that the average of the cardinalities of these subsets is greater than:

$$\frac{1}{|\boldsymbol{T}|} \sum_{(\boldsymbol{a}, \boldsymbol{b}) \in \boldsymbol{T}} \frac{|\boldsymbol{a}||\boldsymbol{b}|}{2}.$$

Furthermore, if:

$$w_H(\boldsymbol{c}_{i,.}) \leq 2e < 2d_A, \quad \forall i \in \{1, \ldots, m\} \tag{A.12}$$

where e is a real number, then, each rectangle $\boldsymbol{t}_i$ of these rectangles will have dimensions less than $|\boldsymbol{b}_i| \times 2e$.

Before we present the proof, we should point out that the value of ω_A given in (A.10) is found by iteratively increasing the value of r' beginning from 0. We check to see if Inequality (A.11) is satisfied, then we have found ω_A, otherwise, we increase r' by one and try again.

Proof: We start first by permuting the rows and columns in the codeword $\boldsymbol{c}$ such that the non-zero rows and columns are gathered in one place as shown in Figure A.1 below. All non-zero rows and columns are gathered in one region. The

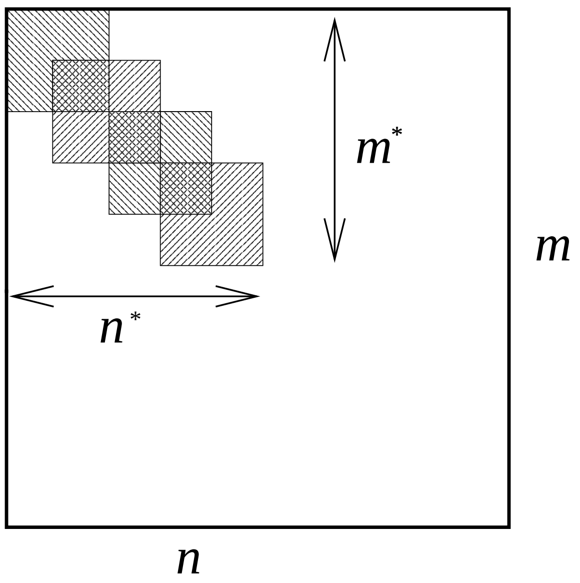


Figure A.3: Figure used in the proof of Theorem A.5.

non-zero region in the codeword matrix can be regarded as a shortened code, $\mathcal{C}^* = \mathcal{A}^* \otimes \mathcal{B}^*$, where $\mathcal{A}^*$ is obtained from $\mathcal{A}'$ by shortening all $\{n^*+1, \ldots, n\}$ coordinates and $\mathcal{B}^*$ is obtained from $\mathcal{B}'$ by shortening all $\{m^*+1, \ldots, m\}$ coordinates. Let k_A^* and k_B^* be the number of independent rows and columns respectively. Also, let the integer $r(\mathcal{A}^*)$ be as defined in (A.7). We first consider the case were the Hamming weight of the non-zero columns in $\boldsymbol{c}$ is exactly equal to d_B. The length of the shortened code is:

$$n^* = \frac{w_H(\boldsymbol{c})}{d_B}.$$

This means that the value of $d_r^\perp$ will, therefore, be exactly equal to $d_{r'}^\perp$. which leads to:

$$r(\mathcal{A}^*) = r'. \tag{A.13}$$

Let us choose the k_A^* least weight independent rows as a basis for the shortened code $\mathcal{A}^*$ and use these rows as rows in the generator matrix G_A^* for the code $\mathcal{A}^*$. In a similar manner, Let us choose the k_B^* least weight independent columns as a basis for the shortened code $\mathcal{B}^*$ and use these columns as rows in the generator matrix G_B^* for the code $\mathcal{B}^*$. Therefore, if we use these two generator matrices to generate the constructing rectangles for the code $\mathcal{C}^*$, each constructing rectangle will have horizontal, row, side length less than $2e$. Due to Property (A.10), we have:

$$\begin{aligned}
& n^* \le \frac{w_H(\boldsymbol{c})}{d_B} \le \frac{d_A d_B}{d_A - 2}\frac{n - k_A - r'}{d_B} \\
\Rightarrow\quad & n^* - \frac{2}{d_A}n^* \le n - k_A - r' \\
\Rightarrow\quad & k_A - \frac{2}{d_A}n^* \le n - n^* - r' \le k_A - k_A^* + r(\mathcal{A}^*) - r' \\
\Rightarrow\quad & k_A^* \le \frac{2n^*}{d_A},
\end{aligned} \tag{A.14}$$

where the last inequality is due to (A.13). This means, due to Lemma A.3, that the support of the codeword $\boldsymbol{c}$ can be divided into a finite number of subsets each of which is contained in a rectangle with row side length less than $2e$ and such that the average of their cardinalities is greater than:

$$\frac{1}{|\boldsymbol{T}|}\sum_{(\boldsymbol{a},\boldsymbol{b})\in\boldsymbol{T}} \frac{|\boldsymbol{a}||\boldsymbol{b}|}{2}. \tag{A.15}$$

For any other codeword $\boldsymbol{c'}$ that has the same Hamming weight as $\boldsymbol{c}$ except that the weight of the columns in $\boldsymbol{c'}$ is greater than d_B, then, Lemma A.3 is not satisfied. However, the number of constructing rectangles of $\boldsymbol{c'}$ will be, at most, the same as for the codeword $\boldsymbol{c}$. Therefore, the average number of ones in each constructing rectangle will still be greater than that given in (A.15). □

A.2 The suboptimal decoder

We now consider a suboptimal decoder with respect to a maximum likelihood decoder and try to estimate its performance.
Let the code used in the transmitter be the product code:

$$\mathcal{C} \overset{\triangle}{=} \mathcal{A}' \otimes \mathcal{B}',$$

and let the two codes $\mathcal{A}$ and $\mathcal{B}$ be defined as above. Let $\boldsymbol{y}$ be the received matrix and let $\hat{\boldsymbol{c}}$ be the maximum likelihood codeword. Let $\xi_e(\cdot,\cdot)$ be a list decoder defined as follows:

$$\xi_e(\mathcal{A},\boldsymbol{y}) \triangleq \{\boldsymbol{a} \in \mathcal{A} | d_H(\boldsymbol{y},\boldsymbol{a}) \leq e\}$$

Maximum likelihood decoding can be performed by list decoding the received matrix $\boldsymbol{y}$ over the code $\mathcal{A}$ up to $\rho(\mathrm{C})$, the covering radius of $\mathcal{C}$. We then sort the set in a list according to their distance from $\boldsymbol{y}$ and checking each member of the list, beginning from the top, to see if it is a member of the code $\mathcal{B}$. If it was, this codeword will be returned as the maximum likelihood solution. List decoding of the code $\mathcal{A}$ can be done by list decoding the rows of the matrix $\boldsymbol{y}$ over the code $\mathcal{A}'$ and taking the direct sum of all the sets. Let us call the maximum likelihood decoding algorithm by Algorithm 1.
Now, let us restrict the decoder in such a way that the list decoder for the rows can only decode up to e errors and let the list of such codewords be called $\boldsymbol{A}^*$.This way, any error pattern that has d_A or more errors in one or more rows cannot be corrected as shown below:

$$\begin{aligned} \boldsymbol{A}^* \quad &= \begin{array}{|c|} \hline \boldsymbol{a}_1 \\ \hline \boldsymbol{a}_2 \\ \hline \vdots \\ \hline \hat{\boldsymbol{c}} \\ \hline \vdots \\ \hline \boldsymbol{a}_{(last)} \\ \hline \end{array} \\ d(\boldsymbol{a}_i,\boldsymbol{y}) \quad &\leq \quad d(\boldsymbol{a}_j,\boldsymbol{y}), \quad \forall i < j. \end{aligned}$$

Let us call this decoding algorithm Algorithm 2.

Lemma A.6 (Lemma 3.2) *Let the product code $\mathcal{C} \triangleq \mathcal{A} \cap \mathcal{B}$ and the suboptimal decoder Algorithm 2 above be used for data transmission. Let the decoding radius for the list decoder of the rows be e, where e is less than d_A and let the received matrix be $\boldsymbol{y}$. If all the following:*

1. *The Hamming weight of the error in each row in $\boldsymbol{y}$ is less than e.*
2. *The Hamming weight of the total error is less than $\omega_A/2$, where:*

$$\omega_A \triangleq \frac{d_A d_B}{d_A - 2}(n - k_A - r'), \tag{A.16}$$

where r' is an integer satisfying:

$$d^{\perp}_{r'+1} \geq n - \frac{\omega_A}{d_B}, \quad d^{\perp}_{r'} < n - \frac{\omega_A}{d_B}. \tag{A.17}$$

3. *The support of every constructing rectangle in* $\boldsymbol{y}$ *with dimensions* $f \times g$ *where* $g \leq 2e$ *contains less than* $fg/4$ *errors.*

then, the decoding will be error-free.

Proof: Let the sent codeword be the all-zero codeword and assume that every constructing rectangle, $(\boldsymbol{a}, \boldsymbol{b})$ in $\boldsymbol{y}$ contains less than $|\boldsymbol{a}||\boldsymbol{b}|/4$ errors. Assume that their exists in $\boldsymbol{A^*}$, the list obtained from the Algorithm 2, a codeword $\boldsymbol{c}$ such that:

$$d(\boldsymbol{c}, \boldsymbol{y}) \leq d(\boldsymbol{0}, \boldsymbol{y}),$$

then, we can replace the received matrix $\boldsymbol{y}$ by a matrix $\boldsymbol{y'}$ that only has 1's in the places where both $\boldsymbol{y}$ and $\boldsymbol{c}$ have 1's and zeroes elsewhere. This matrix will be closer to both $\boldsymbol{c}$ and $\boldsymbol{0}$ than $\boldsymbol{y}$ is. Therefore, if this message was decoded instead of $\boldsymbol{y}$, the list of codewords that result from decoding $\boldsymbol{y'}$ will have both $\boldsymbol{c}$ and $\boldsymbol{0}$ as members with $\boldsymbol{c}$ coming before $\boldsymbol{0}$ in the list $\boldsymbol{A^*}$. Let T be the minimal constructing set for generating $\boldsymbol{c}$ by using a subset of the rows as a generator matrix for the row code and a subset of the columns as a generator matrix for the column code.

Let us first prove the lemma for the case that the dimensions of all the constructing rectangles for the codeword $\boldsymbol{c}$ are exactly equal to $d_B \times d_A$.

Due to the conditions imposed by the theorem, the Hamming weight of each row and each column in $\boldsymbol{c}$ will be less or equal to $2e$. Theorem A.5 implies that the support of $\boldsymbol{c}$ can be divided into finitely many subsets located inside rectangles in $\boldsymbol{c}$ with horizontal side length not exceeding $2e$ and the average of the cardinalities of these subsets is greater than $d_A d_B/2$. Let us refer to these subsets by $\square_i, i \in \{1, 2, \ldots, |T|\}$. The members of a subset $\square_i$ are contained in the support of the corresponding constructing rectangle $\boldsymbol{t}_i \in T$. Let:

$$\square'_i = \text{Supp}(\boldsymbol{y'}) \cap \square_i.$$

Also, let:

$$\mathcal{I} = \text{Supp}(\boldsymbol{c}) = \bigcup_{i=1}^{|T|} \square_i,$$

and:

$$\mathcal{I}' = \text{Supp}(\boldsymbol{y'}) = \bigcup_{i=1}^{|T|} \square'_i.$$

Since $\boldsymbol{y}$ is closer to $\boldsymbol{y'}$ than $\boldsymbol{0}$ is, then:

$$|\mathcal{I}'| \geq \frac{1}{2}|\mathcal{I}| \geq \frac{1}{2}|T|\frac{d_A d_B}{2},$$

$$\sum_{i=1}^{|T|} |\square'_i| \geq |T|\frac{d_A d_B}{4}.$$

From the last inequality we see that the average of the weights of those rectangles in $\boldsymbol{y'}$ that occupy the same supports of the constructing rectangles of the codeword $\boldsymbol{c}$, is greater or equal to $d_A d_B/4$, which means that the weight of at least one of those rectangles is greater than $d_A d_B/4$.

We now turn to the case where the constructing rectangles of the codeword $\boldsymbol{c}$ have dimensions greater than $d_B \times d_A$. In this case, and in a similar manner to the above, the average number of errors contained in the supports of all constructing rectangles should be greater than:

$$\frac{1}{|\boldsymbol{T}|} \sum_{(\boldsymbol{a},\boldsymbol{b}) \in \boldsymbol{T}} \frac{|\boldsymbol{a}||\boldsymbol{b}|}{4}.$$

Therefore, there has to be at least one constructing rectangle that contains errors in more than one fourth of its support. This completes the proof. □

It should be noted that even though the method for obtaining ω_A as given in (A.10) is rather tedious and requires the knowledge of the weight hierarchy of the row code, it is possible to bound the value of ω_A. We present here two methods. The first method is done by bounding the generalized Hamming weights of the dual code $\mathcal{A}^{\perp}$ of the code $\mathcal{A}'$. This is done by using the very well known Griesmer Bound, [8, p. 547]:

$$d_{r'}^{\perp} \geq d_1^{\perp} + \lceil \frac{d_1^{\perp}}{2^1} \rceil + \lceil \frac{d_1^{\perp}}{2^2} \rceil + \ldots + \lceil \frac{d_1^{\perp}}{2^{r'-1}} \rceil,$$

where, $d_1^{\perp}$ is the minimum distance of the dual code as shown in (A.6) and the discussion that follows.

Using this bound on the generalized Hamming weights of the dual code-to obtain ω_A, instead of the actual values, it is possible to obtain a value, ω'_A that is a lower bound on ω_A.

It is also possible to bound ω_A in a different manner using Lemma A.3 and specifically (A.5). The way to do that is by noticing that a codeword $\boldsymbol{c} \in \mathcal{C}$ that has a minimal constructing set $\boldsymbol{T}$ and occupies n^* non-zero columns, has to be an element in some linear code with parameters $[n^*, k_A^*, d_A^*]$, where:

$$\begin{array}{rcl} k_A^* & = & |\boldsymbol{T}| \\ d_A^* & \geq & d_A. \end{array}$$

However, the value of k_A^* cannot exceed:

$$k_A^* \leq \kappa \overset{\triangle}{=} \log_2 (\text{Best cardinality upper bound}(n^*, d_A)), \tag{A.18}$$

where we mean by the above that we take the best known bound on the cardinality of a binary code with length n^* and minimum distance d_A. Thus, ω_A can be bounded by looking for the largest n^* such that $\kappa \leq 2n^*/d_A$ and thus ω_A can be bounded as follows:

$$\omega_A \geq \omega_A'' \stackrel{\triangle}{=} n^* d_B. \tag{A.19}$$

This page intentionally contains only this sentence.

References

[1] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27, 1948.

[2] P. Elias. Error-free coding. *IEEE Transactions on Information Theory*, 4:29–37, 1954.

[3] David G. Forney Jr. Generalized Minimum Distance Decoding. *IEEE Transactions on Information Theory*, IT-12, 1966.

[4] David G. Forney Jr. *Concatenated Codes.* The M.I.T. press, Cambridge Massachusetts, 1966.

[5] B.B. Zyablov and E.L. Blokh. *Linear Concatenated Codes.* Moscow, 1982. in Russian, unofficial translation to english by Suhail Fawakhiri.

[6] V. A. Zinov'ev. Generalized cascade codes. *Problemy Peredachi Informatsii*, 12(1), January 1976.

[7] B.B. Zyablov and V. A. Zinoviev. Decoding of non-linear generalized cascade codes. *Problemy Peredachi Informatsii*, 14(2), 1978.

[8] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-Correcting Codes.* North-Holland, 1977.

[9] B.B. Zyablov and E. L. Blokh. Existence of linear concatenated binary codes with optimal error correcting capability. *Problemy Peredachi Informatsii*, 9(4), 1973.

[10] Shu Lin. *An introduction to error-correcting codes.* Prentice Hall, 1970.

[11] G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes.* North Holland Math. Library, 1997.

[12] G. Cohen, M. Karpovsky, H. Mattson Jr., and J. Schatz. Covering radius - survey and recent results. *IEEE Transactions on Information Theory*, vol. 31(nr. 3), May 1985.

[13] R. G. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, IT-8:21–28, January 1962.

[14] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: Turbo codes. *IEEE Transactions on Information Theory*, 44(10), Oct. 1996.

[15] H. Imai and S. Hirakawa. A new multilevel coding method using error correcting codes. *IEEE Transactions on Information Theory*, IT-23:371–377, 1977.

[16] A. R. Calderbank. Multilevel codes and multistage decoding. *IEEE Transactions on Information Theory*, 37(3):222–229, Mar 1989.

[17] J. Hagenauer, E. Offer, and Lutz Papke. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on Information Theory*, vol. 42(nr. 2), March 1996.

[18] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT. 20, March 1996.

[19] Alexander Vardy. Trellis Structure of Codes. In Vera S. Pless, W. Cary Huffman, and Richard Brualdi, editors, *Handbook of Coding Theory*. Elsevier Science Publishers, Amsterdam, 2000.

[20] R. M. Pyndiah. Near-optimum decoding of product codes: Block turbo codes. *IEEE Transactions on Communications*, 46(8):1003–10, Aug. 1998.

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

[22] A. Risley and K. Pickavance. Turbo product codes in wireless communications. In *Proceedings of the Int'l Symp. on Communication Theory and Applications*, pages 322–322. HW Communications, HW Communications ltd, july 2001.

[23] J. Fang and F. Buda. A special family of product codes ”turboly” decodable with application to atm cell transmission. In *1998 IEEE International Symposium on Information Theory*, page 289. IEEE, IEEE, aug 1998.

[24] P.A. Martin and D.P. Taylor. Distance based adaptive scaling in suboptimal iterative decoding. *IEEE Transactions on Communications*, 50(6):869–871, jun 2002.

[25] A. Krause, A. Sella, and Y. Be'ery. Convergence analysis of turbo-decoding of serially concatenated product codes. In *2001 IEEE International Symposium on Information Theory*, page 318. IEEE, IEEE, jun 2001.

[26] A. Sella and Y. Be'ery. Convergence analysis of turbo-decoding of product codes. In *2000 IEEE International Symposium on Information Theory*, page 484. IEEE, IEEE, jun 2000.

[27] J. Hagenauer. Forward error correcting for cdma systems. In *ISSSTA'95 International Symposium on Spread Spectrum Techniques and Applications*, pages 566–569. IEEE, IEEE, sep 1996.

[28] A. Picart and R. Pyndiah. Performance of turbo-decoded product codes used in multilevel coding. In *ICC/SUPERCOMM '96 - International Conference on Communications*, pages 107–111. IEEE, IEEE, jun 1996.

[29] F. Sanzi and S. ten Brink. Iterative channel estimation and decoding with product codes in multicarrier systems. In *Vehicular Technology Conference Fall 2000*, pages 1338–1344. IEEE, IEEE, sep 2000.

[30] G. Buch and F Burkert. Unequal error protection with product-like turbo codes. In *1998 IEEE International Symposium on Information Theory*, page 60. IEEE, IEEE, aug 1998.

[31] T. Souvignier, C. Argon, S.W. McLaughlin, and K. Thamvichai. Turbo product codes for partial response channels. In *International Conference on Communications*, pages 2184–2188. IEEE, IEEE, jun 2001.

[32] O. Al-Askary. Low complexity maximum-likelihood decoding of product codes. In *Proceedings of the Int. Symp. on Information Theory*, page 87. IEEE, jun 2000.

[33] O. Al-Askary. Near maximum-likelihood decoding of product codes. In *Proceedings of the Winter School on Coding Theory, Reisensburg, Germany*. University of Ulm, dec. 2000.

[34] O. Al-Askary. An upper bound on the probability of block error for product codes. In *Proceedings of the Int'l Symp. on Communication Theory and Applications*, pages 323–328. HW Communications, HW Communications ltd, july 2001.

[35] David G. Forney Jr. The viterbi algorithm. *IEEE Transactions on Information Theory*, 61(3), Mar 1973.

[36] Andrew J. Viterbi. Convolutional codes and their performance in communication systems. *IEEE Transactions on Communication Technology*, COM-19:751–772, 1971.

[37] J. Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, IT-18(5):662–666, sep 1972.

[38] Ludo M. G. M. Tolhuizen. More results on the weight enumerator of product codes. *IEEE Transactions on Information Theory*, 48(9):2573–2576, September 2002.

[39] T. Ericson. A simple ananlysis of the blokh-zyablov decoding algorithm. *Proceedings of the AAECC-4, Karlsruhe*, September 1986. Printed in Lecture Notes in Computer Science, nr 307.

[40] John Proakis. *Digital Communications*. McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill, second edition edition, 1995.

[41] John Proakis and Masoud Salehi. *Communication Systems Engineering*. Prentice Hall, second edition edition, 2002.

[42] Lars Ahlin and Jens Zander. *Principles of Radio Communications*. Studentlitteratur, second edition edition, 1998.

[43] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. New York : Wiley, 1991.

[44] Rolf Johannesson. *Informationsteori - grundvalen för (tele)kommunikation*. Studentlitteratur, 1988.

[45] R. J. McEliece. On the bcjr trellis for linear block codes. *IEEE Transactions on Information Theory*, 42(4):1072–1092, jul 1996.

[46] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of convolutional coding*. IEEE Press series on mobile & digital communication. New York : IEEE Press, 1999.

[47] J. K. Wolf. Efficient maximum likelihood decoding of linear block codes using a trellis. *IEEE Transactions on Information Theory*, IT-24(1):76–80, June 1978.

[48] F. R. Kschischang and V. Sorokine. On the trellis structure of block codes. *IEEE Transactions on Information Theory*, 41(6):1924–1937, nov 1995.

[49] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Proceedings of ICC '93 - IEEE International Conference on Communications*, pages 1064–1070. IEEE, IEEE, May 1993.

[50] Niclas Wiberg. *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, 1996.

[51] Jr Forney, G.D. The forward-backward algorithm. In *Thirty-Fourth Annual Allerton Conference on Communication, Control, and Computing*, pages 432–446. Univ. Illinois, Univ. Illinois, Oct 1996.

[52] C.R.P. Hartmann and L.D. Rudolph. An optimum symbol-by-symbol decoding rule for linear codes. *IEEE Transactions on Information Theory*, IT-22(5):514–517, Sept. 1976.

[53] G. Battail, M.C. Decouvelaere, and P Godlewski. Replication decoding. *IEEE Transactions on Information Theory*, IT-25:332–335, 1979.

[54] Rainer Lucas. *On Iterative Symbol-by-Symbol Decision Decoding of Linear Binay Block Codes.* PhD thesis, University of Ulm, 1997.

[55] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[56] V. K. Wei. Generalized hamming weights for linear codes. *IEEE Transactions on Information Theory*, 37(5):1412–1418, September 1991.

[57] Richard Andrew. The weight distribution. In *Proceedings of the Int. Symp. on Information Theory*, page 226. IEEE, jun 2000.

[58] William W. Hines and Douglas C. Montgomery. *Probability and Statistics in Engineering and Management.* John Wiley and Sons Inc., 2 edition, 1980.

[59] Gunnar Blom. *Sannolikhetsteori med tillämpningar.* Studentlitteratur, 2 edition, 1998.

[60] G. Battail. A conceptual framework for understanding turbo codes. *IEEE Journal on Selected Areas in Communications*, 16:245–254, 1998.

[61] Jens Berkmann. On turbo decoding of nonbinary codes. *ICL*, 2(4):94–96, 1998.

[62] Sven Riedel. New symbol by symbol map decoding algorithms for high rate convolutional codes which use reciprocal dual codes. *IEEE Journal on Selected Areas in Communications*, 16:175–185, February 1998.

[63] Anders Furuskär. *Can 3G Services Be offered in Existing Spectrum?* Licentiate thesis, Royal Institute of Technology, 2001.

[64] Van de Meeberg. A tightened uper bound on the error probability of binary convolutional codes with viterbi decoding. *IEEE Transactions on Information Theory*, IT-20:389–391, 1974.

[65] Ludo Tolhuizen, Stan Baggen, and Ewa Hekstra-Nowacka. Union bounds on the performance of product codes. In *Proceedings of the Int. Symp. on Information Theory*, page 267. IEEE, jun 2000.

[66] Jan Nilsson. *On Hard and Soft Decoding of Block Codes.* PhD thesis, Universitetet i Linköping, 1994.

[67] J. Justesen and T. Hoholdt. Bounds on list decoding of mds codes. In *International Symposium on Information Theory,*, page 480. IEEE, 2000.

[68] M. Sudan. Maximum likelihood decoding of reed solomon codes. In *37th Annual Symposium on Foundations of Computer Science, 1996*, pages 164–172, 1996.

[69] V. Guruswami and M. Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, sept 1999.

[70] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[71] Youzhi Xu. *Contributions to the Decoding of Reed-Solomon and Related Codes*. PhD thesis, Universitetet i Linköping, 1991.

[72] George C. Clark and J. Bib Cain. *Error-Correction Coding for Digital Communications*. Applications of Communication Theory. Plenum Press, r. w. lucky edition, 1981.

[73] Richard E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.

[74] D.G Williams. Turbo product codes and their bandwidth efficiency. In *IEE Colloquium on Turbo Codes in Digital Broadcasting - Could It Double Capacity?*, pages 6/1 –6/29. IEEE, 1999.

[75] University of Southern California Information Sciences Institute. RFC 791: INTERNET PROTOCOL, DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION. Technical report, Defense Advanced Research Projects Agency, September 1981.

[76] ANSI/IEEE std 802.11, 1999 edition. Technical report, ANSI/IEEE, 1999.

[77] M.P.C. Fossorier and Shu Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, 41(5):1379–1396, sep 1995.

[78] M.P.C. Fossorier and Shu Lin. Computationally efficient soft-decision decoding of linear block codes based on ordered statistics. *IEEE Transactions on Information Theory*, 42(3):738–750, may 1996.

[79] U. Wachsmann, R.F.H. Fischer, and J.B. Huber. Multilevel codes: theoretical concepts and practical design rules. *IEEE Transactions on Information Theory*, 45(5):1361 –1391, Jul. 1999.

TRITA—S3—RST—0305
ISSN 1400—9137
ISRN KTH/RST/R--03/05--SE

Radio Communication Systems Lab.
Dept. of Signals, Sensors and Systems
Royal Institute of Technology
S–100 44 STOCKHOLM
SWEDEN