Two-Dimensional Tail-Biting Convolutional Codes

By
Liam Alfandary

Supervised by
Dr. Dani Rephaely

THESIS SUBMITTED FOR THE DEGREE OF “MASTER OF SCIENCE”
2008
Acknowledgements

I wish to thank the following people for their help in the conduction of this research:
First and foremost, I wish to thank my supervisor, Dr. Dani Rephaely for his guidance
and patience, and for his useful pointing of research directions.
To Dr. Yaron Shani, to whom I owe many hours of lectures in algebra, and who helped
me formulate and prove two important propositions in this dissertation.
To Talya Meltzer of Weizmann institute, for her support in understanding the GBP
algorithm.
To Dr. Simon Litsyn and Dr. David Burshtein of Tel-Aviv University for their
consultation.
To my work colleagues, Dr. Boaz Kol, Dr. Shraga Bross and Dr. Ilan Reuven, for helping
me clarify my ideas, and taking time to review drafts of this dissertation.
Finally I wish to thank my parents, Azriel and Pnina, and my grandparents, Dov and
Miriam, for their ongoing support throughout my studies.
Abstract
Multidimensional convolutional codes (m-D CC, where m is the dimension) generalize the well known notion of a 1-D convolutional code, generated as a convolution of an information vector with a set of generating vectors. Likewise, a m-D CC is generated by convolving a m-D information sequence with m-D generating sequences. Whereas in 1-D convolutional codes are isomorphically represented over the univariate polynomial ring, in m-D they are isomorphically represented over a multivariate polynomial ring, which makes the generalization non-trivial. While 1-D CCs have been thoroughly understood and analyzed, only a handful of works exist that deal with m-D CCs, and most of them lay the algebraic foundation for m-D CCs. The coding power of m-D CCs is virtually unknown, since no effort has been done to find good codes in this family. Additionally, no efficient algorithms exist to decode m-D CCs.

In this dissertation we look at the specific case of two-dimensional tail-biting convolutional codes (2-D TBCCs). "Tail-biting" is an efficient way to reduce coding overhead in convolutional codes. However, this technique transforms the CC problem from the infinite polynomial ring to finite quotient polynomial rings, which requires a modified algebraic treatment. Therefore, we extend the algebraic foundation of m-D CCs to tail-biting codes. We also demonstrate the application of Groebner bases to the analysis of the properties of 2-D TBCCs.

Next, we tackle the problem of the coding strength of 2-D TBCCs, and attempt to find powerful codes in this family. To this end we take an existing algorithm for calculating the weight distribution of a 1-D CC, modify it to fit the 2-D case. We use the union bound to obtain an upper-bound on optimal decoding. We compare the performance of the best codes found with other comparable known codes. We find that in many cases the performance of 2-D TBCCs is similar to, and sometimes better than comparable codes.

Finally, we tackle the problem of soft decoding of 2-D TBCCs. First we apply the optimal ML decoder (Viterbi algorithm) after reducing the problem to a 1-D problem. Since this algorithm has complexity which is exponential with the dimension of the input, we devise a sub-optimal belief propagation decoding algorithm based on an extension of the trellis concept to 2-D. We find conditions for the convergence of this algorithm. We also explore the application of loopy belief propagation (LBP) and generalized belief propagation (GBP) techniques for decoding 2-D TBCCs, using their parity check matrices. Since generally the codes are not systematic, an inverse encoding operation is required to recover the information sequence. We demonstrate how to construct an inverse encoding belief propagation network, and compare the performance of the algorithms.
Contents
Acknowledgements............................................................................................................. ii
Abstract .............................................................................................................................. iii
Contents ............................................................................................................................. iv
List of Figures .................................................................................................................... vi
List of Tables ................................................................................................................... viii
1 Introduction .................................................................................................................. 1
  1.1 Existing Literature ................................................................................................. 1
  1.2 2-D Tail-Biting Convolutional codes ................................................................. 2
  1.3 Outline of Dissertation ......................................................................................... 4
  1.4 Contributions ....................................................................................................... 5
2 Algebra of Multivariate Polynomials ........................................................................ 6
  2.1 Polynomials, Rings and Ideals .............................................................................. 6
  2.2 Multivariate Polynomials and Groebner Bases .................................................. 8
    2.2.1 Monomial Orderings ...................................................................................... 8
    2.2.2 A Division Algorithm in $k[x_1, ..., x_n]$ ..................................................... 10
    2.2.3 Groebner bases ............................................................................................ 11
    2.2.4 Quotients of Polynomial Rings ................................................................. 15
3 Algebraic Aspects of 2-D Tail-Biting Convolutional Codes .................................... 16
  3.1 Representation of 2-D Information ...................................................................... 16
  3.2 Definition of 2-D Tail-Biting Convolutional Codes ............................................. 20
    3.2.1 Codes, Generator Matrices and Encoders .................................................. 20
    3.2.2 Non-Degenerate Codes ................................................................................ 22
    3.2.3 Relationship with Quasi-Cyclic Codes ...................................................... 24
  3.3 Inverse Encoders and Parity Check Matrices ..................................................... 27
    3.3.1 Parity Check Matrices .................................................................................. 27
    3.3.2 Inverse Encoders ......................................................................................... 29
    3.3.3 Relationship to Belief Propagation Algorithms ....................................... 32
4 Distance Properties .................................................................................................... 34
  4.1 Basic concepts ....................................................................................................... 34
  4.2 BEAST algorithm ................................................................................................. 36
    4.2.1 1-D BEAST .................................................................................................. 36
    4.2.2 2-D BEAST ................................................................................................. 39
  4.3 Results for Sample Codes ..................................................................................... 41
    4.3.1 Kernel Support 2×2 ; Information Support 4×4 ......................................... 41
    4.3.2 Kernel Support 2×2 ; Information Support 6×6 ......................................... 43
    4.3.3 Kernel Support 3×3 ; Information Support 6×6 ......................................... 44
  4.4 Comparison with other codes ............................................................................... 46
    4.4.1 Compared to 1-D CCs .................................................................................. 46
    4.4.2 Compared to short LDPCCs ........................................................................ 49
    4.4.3 Compared to BCH codes .......................................................................... 52
  4.5 BER vs. WER ....................................................................................................... 54
5 Survey of Belief Propagation Algorithms ................................................................. 58
  5.1 Loopy Belief Propagation – LBP ......................................................................... 58
  5.2 Generalized Belief Propagation – GBP .............................................................. 61
5.2.1 Free Energies ................................................................. 61
5.2.2 Bethe Approximation ..................................................... 64
5.2.3 Kikuchi Approximation: .................................................. 66

6 Decoding in the Information Domain ........................................ 69
6.1 Optimal Decoding .................................................................. 69
6.1.1 Reduction to a 1-D Problem ............................................... 69
6.1.2 Relationship with Belief Propagation ..................................... 70
6.2 2D-Trellis Local Belief Propagation ......................................... 73
6.2.1 Creating a Factor Graph ..................................................... 73
6.2.2 Formulation of the 2D-Trellis algorithm ............................... 75
6.2.3 Convergence of the 2D-Trellis Algorithm .............................. 78
6.3 Results for Sample Codes ..................................................... 82
6.3.1 Kernel Support 2x2; Information Support 4x4 ....................... 83
6.3.2 Kernel Support 2x2; Information Support 6x6 ....................... 85
6.3.3 Kernel Support 3x3; Information Support 6x6 ....................... 86

7 Decoding in the Parity Check Domain ......................................... 92
7.1 Classical Loopy Belief Propagation: PC-LBP ............................ 92
7.1.1 Description of the algorithm .............................................. 92
7.1.2 Constructing an Inverse BP Network .................................... 94
7.1.3 Other BP Networks ......................................................... 97
7.2 Generalized Belief Propagation – PC-GBP ................................. 98
7.2.1 Constructing a Region Graph ............................................ 98
7.2.2 Message Initialization and Message Passing .......................... 101
7.3 Results for Sample Codes ..................................................... 104
7.3.1 Kernel Support 2x2; Information Support 4x4 ....................... 104
7.3.2 Kernel Support 2x2; Information Support 6x6 ....................... 106
7.3.3 Kernel Support 3x3; Information Support 6x6 ....................... 107

8 Conclusion ............................................................................. 111
8.1 Research Summary ............................................................. 111
8.2 Topics for Further Research .................................................. 112
Appendix A. Proofs for GBP algorithms ........................................ 113
A.1 Bethe Approximation .......................................................... 113
A.2 Kikuchi Approximation ........................................................ 115
Bibliography ............................................................................. 119
List of Figures
Figure 1-1 Convolutional Encoding Process: left – 1-D CC ; right – 2-D CC ................. 2
Figure 1-2 Convolutional Encoding Process: 2-D Tail-Biting CC .................................. 3
Figure 4-1 Example: 1-D BEAST: Left: Forward Tree, Right: Backward Tree .......... 37
Figure 4-2 BEAST algorithm ; left - forward recursion ; right - backward recursion.... 38
Figure 4-3 Example: 2-D BEAST algorithm ................................................................ 40
Figure 4-4 Histogram of codes by $d_{\text{min}}$; $K=2\times2$ ; $N=4\times4$ .......................... 42
Figure 4-5 Performance asymptotes for codes ; $K=2\times2$ ; $N=4\times4$ ........................ 42
Figure 4-6 Histogram of codes by $d_{\text{min}}$; $K=2\times2$ ; $N=6\times6$ ........................... 43
Figure 4-7 Performance asymptotes for codes ; $K=2\times2$ ; $N=6\times6$ ........................ 44
Figure 4-8 Histogram of codes by $d_{\text{min}}$; $K=3\times3$ ; $N=6\times6$ ........................... 45
Figure 4-9 Performance asymptotes for codes ; $K=3\times3$ ; $N=6\times6$ ........................ 46
Figure 4-10 Comparison with 1-D CC(13,17) ....................................................... 47
Figure 4-11 Comparison with 1-D CC(13,17) ....................................................... 48
Figure 4-12 Comparison with 1-D CC(561,753) .................................................... 48
Figure 4-13 Comparison with LDPCC(2,4,16) ....................................................... 51
Figure 4-14 Comparison with LDPCC(2,4,36) and LDPCC(3,6,36) .......................... 51
Figure 4-15 Comparison LDPCC(2,4,36) and LDPCC(3,6,36) ............................... 52
Figure 4-16 Comparison with BCH(31,16) ............................................................ 53
Figure 4-17 Comparison with BCH(63,30) and BCH(63,36) ................................. 54
Figure 4-18 BER vs. $E_b/N_0$ of a K=2x2 code for various N ............................... 56
Figure 4-19 WER vs. $E_b/N_0$ of a K=2x2 code for various N ............................... 56
Figure 4-20 BER vs. WER of a K=2x2 code for various N ............................... 57
Figure 4-21 BER Comparison of 2-D TBCC with 1-D TBCC .............................. 57
Figure 5-1 Simple Factor Graph Example ............................................................ 58
Figure 5-2 Factor graph example with $x_i$ at the root .......................................... 59
Figure 6-1 Division of 2-D plane to conditionally independent regions .................. 70
Figure 6-2 Classical Graph Representation of a 1-D Trellis .................................... 71
Figure 6-3 Region Graph Representation of a 1-D Trellis ....................................... 72
Figure 6-4 Constraint regions of an information sequence ...................................... 74
Figure 6-5 Constraint region factor graph .............................................................. 75
Figure 6-6 Division of constraint region to sub-regions ......................................... 79
Figure 6-7 Code 1, 2-D Trellis Performance: $K=2\times2$, $N=4\times4$; $d_{\text{min}}=4$ ................. 84
Figure 6-8 Code 2, 2-D Trellis Performance: $K=2\times2$, $N=4\times4$; $d_{\text{min}}=6$ .................. 84
Figure 6-9 Code 3, 2-D Trellis Performance: $K=2\times2$, $N=6\times6$; $d_{\text{min}}=6$ .................. 85
Figure 6-10 Code 4, 2-D Trellis Performance: $K=2\times2$, $N=6\times6$; $d_{\text{min}}=6$ ................. 86
Figure 6-11 Codes satisfying the 2D-Trellis condition. $K=3\times3$; $N=6\times6$ ......................... 87
Figure 6-12 Code 5, 2-D Trellis Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=12$ .................. 89
Figure 6-13 Code 6, 2-D Trellis Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=12$ .................. 89
Figure 6-14 Code 7, 2-D Trellis Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=11$ .................. 90
Figure 6-15 Code 8, 2-D Trellis Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=11$ .................. 90
Figure 6-16 Code 9, 2-D Trellis Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=8$ .................... 91
Figure 7-1 Two Stage Decoding: Decoder and Inverse Encoder .................................................. 92
Figure 7-2 Tanner Graph of a 2D TBCC ...................................................................................... 93
Figure 7-3 Code Inverse: Info. Bit error prob. vs. Code bit error prob. ........................................... 96
Figure 7-4 Code Inverse: Ratio of Info. Bit error prob. to Code bit error prob. ......................... 96
Figure 7-5 GBP Region Graph Construction for Hamming(7,4) code ........................................... 99
Figure 7-6 Example: Coverage of the large regions of a 2-D TBCC ............................................. 100
Figure 7-7 Example: Part of the region graph of a 2-D TBCC ...................................................... 101
Figure 7-8 Code 1, PC-BP Performance: $K=2\times2$; $N=4\times4$; $d_{\text{min}}=4$ ......................... 105
Figure 7-9 Code 2, PC-BP Performance: $K=2\times2$; $N=4\times4$; $d_{\text{min}}=6$ ......................... 105
Figure 7-10 Code 3, PC-BP Performance: $K=2\times2$; $N=6\times6$; $d_{\text{min}}=6$ ......................... 106
Figure 7-11 Code 4, PC-BP Performance: $K=2\times2$; $N=6\times6$; $d_{\text{min}}=6$ ......................... 107
Figure 7-12 Code 5, PC-BP Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=12$ ....................... 108
Figure 7-13 Code 6, PC-BP Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=12$ ....................... 109
Figure 7-14 Code 7, PC-BP Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=11$ ....................... 109
Figure 7-15 Code 8, PC-BP Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=11$ ....................... 110
Figure 7-16 Code 9, PC-BP Performance: $K=3\times3$; $N=6\times6$; $d_{\text{min}}=8$ ......................... 110
List of Tables

Table 4-1 Best codes with kernel support 2×2 over information support 4×4 ............... 41  
Table 4-2 Best codes with kernel support 2×2 over information support 6×6 ............... 43  
Table 4-3 Best codes with kernel support 3×3 over information support 6×6 ............... 45  
Table 4-4 Reference 1-D CCs for comparison ................................................................. 47  
Table 4-5 Reference BCH Codes for comparison .......................................................... 53  
Table 4-6 Weight Distribution of a K=2x2 code for various N ........................................ 55  
Table 6-1 Variable-to-Function messages in the 2D-Trellis algorithm ............................... 76  
Table 6-2 Function-to-Variable messages in the 2D-Trellis algorithm ............................... 76  
Table 6-3 Example I: Constraint regions by associated code fragments ......................... 82  
Table 6-4 2D-Trellis performance summary for codes with K=2×2, N=4×4................. 83  
Table 6-5 2D-Trellis performance summary for codes with K=2×2, N=6×6..................... 85  
Table 6-6 Best codes potentially suitable for the 2D-Trellis Alg.: K=3×3 ; N=6×6 ....... 88  
Table 6-7 2D-Trellis performance summary for codes with K=3×3, N=6×6.................... 88  
Table 7-1 PC-BP Performance summary for codes with K=2×2, N=4×4 ......................... 104  
Table 7-2 PC-BP Performance summary for codes with K=2×2, N=6×6 ......................... 106
1 Introduction

In this chapter we introduce the reader to the concept of $m$-D convolutional codes ($m$-D CCs) and $m$-D tail biting convolutional codes ($m$-D TBCCs). We give a short outline of the dissertation, and summarize the salient innovations in it.

1.1 Existing Literature

Convolutional codes in one dimension are a well known and extensively researched subject in coding theory [4]. Their structure and algebraic properties were first investigated by Forney [5], and optimum decoding techniques have been developed for them by Viterbi [6] and Bahl et. al. [7].

Multi-dimensional convolutional codes ($m$-D CCs, where $m$ stands for the dimension) are an extension of the convolutional code notion to multi-dimensional information sequences and generator polynomials. However, in contrast to the one dimensional case, little research has been done in the field of multi-dimensional convolutional codes, and there are only a handful of papers that discuss them. Significantly, the algebraic theory for multi-dimensional convolutional codes has been laid out by Foransini & Valcher [23], [24], [25] and Weiner [26]. Lobo et. al. [27] have also investigated the subject, concentrating on a sub-family of these codes dubbed “Locally Invertible $m$-D CCs”. Some progress in the field was also made by Charoenlarpnopparut et al., [28], who suggest a method to realize a 2-D convolutional encoder, and a construction for its parity check matrix.

Two problems especially stand out among the open problems in the field of $m$-D CCs: The first problem is the encoding properties of $m$-D CCs: Are there any powerful codes in this family, and how are they expected to perform in comparison to other known codes?

The second problem is to efficiently decode $m$-D CCs. As Marrow and Wolf [20] have shown, the existing Viterbi algorithm can be extended to decode $m$-D CCs by transforming the problem to a one-dimensional code. However, the complexity of the algorithm then becomes exponential with the size of the encoded word, which is very much undesired. Wiener [26] proposes a simple algebraic decoding algorithm for sub-family of simple $m$-D CCs dubbed “Unit-Memory” codes, but no performance results are given. Lobo [27] asserts that locally-invertible $m$-D CCs can be decode by a table-driven decoder [16], but does not specify a full algorithm or give any performance results. Moreover both the algorithms mentioned above are hard decision algorithms, and therefore one can improve upon them if soft decision metrics can be brought into account. None of the above authors considered tail biting version of the $m$-D CC. We define such codes and derive some properties of these codes.

In this dissertation we focus mostly on codes over short information sequences. This was primarily motivated by the fact that LDPCs and Turbo Codes already provide good solutions for the coding problem over long sequences. Additionally, short blocks were preferred in order to simplify the discussion, and due to practical limitations (simulation processing power). For further simplicity, most of the examples in this dissertation are of 2-D TBCCs of rate $\frac{1}{2}$. 

1.2 2-D Tail-Biting Convolutional codes

The concepts in this section shall be discussed at length in the body of the dissertation, but for now let us survey briefly the operation of 2-D TBCCs. Recall the operation of a 1-D CC: A 1-D CC of rate $1/n$ operates by convolving an information sequence $u$ with a set of short sequences $\{g_1, g_2, \ldots, g_n\}$, known as generator polynomials, to create a set of $n$ sequences:

$$v_j[k] = \sum_{l=0}^{K-1} g_j[l]u[k-l] = \{g_j * u\}[k]$$  \hspace{1cm} (1.2.1)

Where $i=1,2,\ldots,n$, and $k=0,1,\ldots,N$. $K$ is the maximal length of $\{g_i\}$, and is termed the constraint length. For finite information sequences of length $N$ the convolution operation has to be terminated. One option is to pad the sequence with $K-1$ zeros, which results in $n(K-1)$ additional coded bits, known as tail bits. This means that although the nominal code rate is $1/n$, the actual code rate is smaller:

$$R_c = \frac{1}{n} \cdot \frac{N}{N + K - 1} < \frac{1}{n}$$  \hspace{1cm} (1.2.2)

The operation of a 1-D CC with zero termination is shown on the left hand of Figure 1-1. One way to avoid the generation of tail bits is tail biting, where the linear convolution operation is replaced with a cyclic convolution:

$$v_j[k] = \sum_{l=0}^{K-1} g_j[l]u[(k-l) \mod N] = \{g_j \otimes u\}[k]$$  \hspace{1cm} (1.2.3)

This avoids the necessity of zero termination, and keeps the code rate at $1/n$.

![Convolutional Encoding Process: left – 1-D CC; right – 2-D CC](image-url)
In two dimensions, we simply replace the 1-D convolution operation with a 2-D convolution. Thus, we are dealing with a 2-D information sequence \( u[k,l] \), where \( k=1,2,\ldots,N_1 \), \( l=1,2,\ldots,N_2 \), and 2-D generator sequences \( \{g_i[k,l]\}, \ i=1,2,\ldots,n, \ k=1,2,\ldots,K_1, \ l=1,2,\ldots,K_2 \). In the context of \( m \)-D convolution we shall sometimes refer to the sequences \( \{g_i\} \) as the convolution kernels.

For non-tail biting codes we use the linear convolution operation:

\[
v_i[k_1,k_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} g_i[k_1,k_2] u[k_1-l_1,k_2-l_2]
\]  

(1.2.4)

Once again, zero termination is required. However, the number of tail bits generated in 2-D is much higher in proportion to the information sequence: \( (N_1+K_1-1)(N_2+K_2-1) - N_1N_2 \) tail bits are generated. After some manipulation, this means that the actual code rate is:

\[
R_c = \frac{1}{n} \frac{N_1N_2}{(N_1+K_1-1)(K_2-1)+N_2(K_1-1)} < \frac{1}{n}
\]  

(1.2.5)

This is shown on the right hand of Figure 1-1. Therefore it is highly desirable when dealing with \( m \)-D codes to use the cyclic convolution operation, in order to keep the code rate at \( 1/n \):

\[
v_i[k_1,k_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} g_i[k_1,k_2] u[(k_1-l_1) \mod N_1, (k_2-l_2) \mod N_2]
\]  

(1.2.6)

When using the cyclic convolution in 2-D, the edges of the information sequence are “glued” together. This is similar to taking a square piece of paper, and first rolling it lengthwise to a cylinder. Then the ends of the cylinder are brought together to form a torus. Thus, whereas in the linear convolution the convolution kernels slide over a Euclidean plane, in cyclic convolution the kernels slide over a torus formed by “stitching” the edges of the information word together. This geometrical interpretation is shown in Figure 1-2.

2-D CCs in general, and 2-D TBCCs in particular are a little-researched subject in information theory. Their coding strength is relatively unknown, and no efficient decoding algorithms have been devised for them. While strict maximum likelihood decoding is possible, it turns out to
have complexity of $O(2^{N_1(K_2-1)})$, where we take $N_1$ to be the minimal dimension of the information sequence (see chapter 6.1), and $K_2$ the complementary dimension of the convolution kernel. Thus, there is much academic interest in investigating the coding power of 2-D TBCCs, and find efficient methods for their decoding.

At this point we wish to make a distinction between 2-D CCs and product convolutional codes. In product codes, encoding is done over 2-D information sequences, but each dimension is encoded separately. E.g., first each row of the input is encoded, then each column of the result is encoded. Product convolutional codes can be regarded as a subset of 2-D CCs, where the convolution kernel is *separable*, i.e., the convolution operation can be written as:

$$v_i[k_1,k_2] = g^{(x)}[k_1] \otimes g^{(y)}[k_2] \otimes u[k_1,k_2] \quad (1.2.7)$$

So the convolution operation can be broken into two individual convolution operations along each dimension. In this dissertation we do not explicitly investigate product convolutional codes, as they are already well researched (see for example [19]).

### 1.3 Outline of Dissertation

The dissertation has two distinct parts. Chapters 2 to 4 discuss the encoding properties of 2-D CCs, mostly from an algebraic point of view. Chapters 5 to 7 discuss various soft decoding algorithms, based on belief propagation techniques.

We start the dissertation by giving some necessary algebraic background in Chapter 2. This is basically a summary of relevant chapters from [1] that are relevant to the subject. The two main notions in this chapter that are essential to the understanding of m-D CCs are: (a) long divisions of multi-variate polynomials, and (b) Groebner bases of multi-variate polynomial rings. We will use these notions to find valid codes, and to test their properties.

In Chapter 3 we lay down the foundation for working with 2-D sequences. We define basic operations over 2-D sequences and discuss their equivalence to bivariate polynomials.

Next, we discuss the algebraic properties of 2D CCs. We show how valid codes can be found, and also discuss basic concepts such as generator matrices, parity check matrices, and code inverses. Works by Wiener [26] and Lobo [27] are the main sources for this chapter.

Chapter 4 deals with distance properties of convolutional codes. We extend the BEAST [17] algorithm to 2-D CCs, and use it to search for codes with large minimum distance. We also compare the properties of these codes with some other known codes.

Chapter 5 surveys belief propagation algorithms, and lays the theoretical background for the soft decoding algorithms explored later. The main concepts in this chapter are factor graphs, loopy belief propagation (LBP) and generalized belief propagation (GBP). We give an introduction to factor graphs and belief propagation based on [33], and an introduction to generalized belief propagation based on [34][35], and [36].

In chapters 6 and 7 we explore the performance of several soft decoding algorithms that can be applied to 2-D CCs. We start by discussing the optimal Viterbi algorithm, and then explore several sub-optimal algorithms. We present performance results and compare them with known bounds.

Chapter 8 concludes the work and points directions for further research.
1.4 Contributions

There are several contributions made in this dissertation. In chapter 3 we define m-D tail-biting codes and extend some results from [26], [27] and [28] to tail-biting codes. In chapter 4 we extend some results from [26], [27] and [28] to tail-biting codes. We also show how to test for valid codes, how to construct parity check matrices for codes of rate $1/n$, how to find invertible codes, and how to construct their code inverses. Tail biting CCs are shown to be quasi-cyclic codes.

In chapter 5 we extend the BEAST algorithm from 1-D convolutional codes to 2-D convolutional codes, and give the distance properties of codes found as a result of this search.

In chapter 7 we devise a heuristic extension of the 1-D Viterbi / BCJR algorithms to 2-D codes and investigate its converge and performance for various codes. This algorithm has the advantage of having a complexity exponential with the size of the generator polynomials (which is analogous to the complexity of Viterbi in the 1-D case), however it only performs reasonably well for low-complexity codes or codes with very low rates.

Finally, in chapter 8 we explore using generalized belief propagation techniques proposed by [27] to decode 2-D CCs. A modification to the graph proposed by [35][34] is used and found to yield better performance. We also compare them with conventional loopy belief propagation and present performance results.
2 Algebra of Multivariate Polynomials

Algebra is an important tool in the analysis of codes in general and convolutional codes in particular. When dealing with one-dimensional codes, the algebra of polynomials of a single variable (univariate polynomials) is often used. When dealing with multi-dimensional codes, we will often require the use of algebra of multi-variate polynomials. This algebra generalizes many of the results of univariate polynomials, but many generalizations are not trivial (for example, long division). In this chapter we give an introduction to the algebra of multivariate polynomials, which we later use for the analysis of codes.

We use [1], chapters 1, 2 and 5, as a primary source.

2.1 Polynomials, Rings and Ideals

Let us start by repeating the definitions of some basic concepts. We let \( k \) be some arbitrary field of scalars.

**Definition:** A monomial in \( x_1, \ldots, x_n \) is a product of the form

\[
x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}
\]

(2.1.1)

where all of the exponents \( \alpha_1, \ldots, \alpha_n \) are non-negative integers. The total degree of this monomial is the sum \( \alpha_1 + \cdots + \alpha_n \).

We can simplify the notation for monomials as follows: let \( \alpha = (\alpha_1, \ldots, \alpha_n) \) be an \( n \)-tuple of nonnegative integers. Then we set \( x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} \).

When \( \alpha = (0, \ldots, 0) \), note that \( x^\alpha = 1 \). We also let \( \|\alpha\| = \alpha_1 + \cdots + \alpha_n \) denote the total degree of the monomial \( x^\alpha \).

**Definition:** A polynomial \( f \) in \( x_1, \ldots, x_n \) with coefficients in \( k \) is a finite linear combination (with coefficients in \( k \)) of monomials. We will write a polynomial \( f \) in the form,

\[
f = \sum_{\alpha} a_{\alpha} x^\alpha, \quad a_{\alpha} \in k,
\]

(2.1.2)

where the sum is over a finite number of \( n \)-tuples \( \alpha = (\alpha_1, \ldots, \alpha_n) \). The set of all polynomials in \( x_1, \ldots, x_n \) with coefficients in \( k \) is denoted \( k[x_1, \ldots, x_n] \).

**Definition:** Let \( f = \sum_{\alpha} a_{\alpha} x^\alpha \) be a polynomial in \( k[x_1, \ldots, x_n] \).

(i) We call \( a_{\alpha} \) the **coefficient** of the monomial \( x^\alpha \).

(ii) If \( a_{\alpha} \neq 0 \), then we call \( a_{\alpha} x^\alpha \) a **term** of \( f \).

(iii) The **total degree** of \( f \), denoted \( \deg(f) \), is the maximum \( \|\alpha\| \) such that the coefficient \( a_{\alpha} \) is nonzero.
**Definition:** Given a field \( k \) and a positive integer \( n \), we define the \( n \)-dimensional **affine space** over \( k \) to be the set
\[
k^n = \{(a_1, ..., a_n) : a_1, ..., a_n \in k\}
\] (2.1.3)

**Definition:** A subset \( I \subset k[x_1, ..., x_n] \) is an **ideal** if it satisfies:

(i) \( 0 \in I \).

(ii) If \( f, g \in I \), then \( f + g \in I \).

(iii) If \( f \in I \) and \( h \in k[x_1, ..., x_n] \), then \( hf \in I \).

The first natural example of an ideal is the ideal generated by a finite number of polynomials.

**Definition:** Let \( f_1, ..., f_s \) be polynomials in \( k[x_1, ..., x_n] \). Then we set,
\[
\langle f_1, ..., f_s \rangle = \left\{ \sum_{i=1}^s h_i f_i : h_i \in k[x_1, ..., x_n] \right\}
\]
(2.1.4)

This set is the **ideal generated by** \( f_1, ..., f_s \).

(Proof that the above set is an ideal is straightforward).

Next, we discuss polynomials of one variable and the division algorithm. This simple algorithm has some surprisingly deep consequences—for example, we will use it to determine the structure of ideals of \( k[x] \) and to explore the idea of a greatest common divisor.

**Definition** Given a nonzero polynomial \( f \in k[x] \), let
\[
f = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0
\]
where \( a_i \in k \) and \( a_m \neq 0 \) [thus, \( m = \deg(f) \)]. Then we say that \( a_m x^m \) is the leading term of \( f \), written \( \text{LT}(f) = a_m x^m \).

**Proposition:** (The Division Algorithm). Let \( k \) be a field and let \( g \) be a nonzero polynomial in \( k[x] \). Then every \( f \in k[x] \) can be written as,
\[
f = q g + r,
\]
where \( q, r \in k[x] \), and either \( r = 0 \) or \( \deg(r) < \deg(g) \). Furthermore, \( q \) and \( r \) are unique, and there is an algorithm for finding \( q \) and \( r \).

**Proof.** Here is the algorithm for finding \( q \) and \( r \), presented in pseudo-code:

1. \( q := 0; \)
2. \( r := f \)
3. \( g, f \)
4. \( q, r \)

(2.1.5)
WHILE \( r \neq 0 \) AND \( \text{LT}(g) \) divides \( \text{LT}(r) \) DO
\[
q := q + \frac{\text{LT}(r)}{\text{LT}(g)} \\
r := r - \left( \frac{\text{LT}(r)}{\text{LT}(g)} \right) \cdot g
\]

**Proposition.** If \( k \) is a field, then every ideal of \( k[x] \) can be written in the form \( \langle f \rangle \) for some \( f \in k[x] \). Furthermore, \( f \) is unique up to multiplication by a nonzero constant in \( k \).

**Proof:** See [1] pp 41.

In general, an ideal generated by one element is called a principal ideal. In view of the last proposition, we say that \( k[x] \) is a principal ideal domain, abbreviated PID.

### 2.2 Multivariate Polynomials and Groebner Bases

In this section, we present the method of Groebner bases, which will allow us to formulate algorithms for solving problems about polynomial ideals. The method of Groebner bases is also used in several powerful computer algebra systems, such as CoCoA [3] to study specific polynomial ideals that arise in applications. We will focus on the following problems, which have a direct bearing on the properties of codes:

a. The Ideal Description Problem: Does every ideal \( I \subset k[x_1, \ldots, x_n] \) have a finite generating set? In other words, can we write \( I = \langle f_1, \ldots, f_s \rangle \) for some \( f_i \in k[x_1, \ldots, x_n] \)?

b. The Ideal Membership Problem: Given \( f \in k[x_1, \ldots, x_n] \) and an ideal \( I = \langle f_1, \ldots, f_s \rangle \), determine if \( f \in I \).

#### 2.2.1 Monomial Orderings

If we examine in detail the division algorithm in \( k[x] \) and the row-reduction (Gaussian elimination) algorithm for systems of linear equations (or matrices), we see that a notion of ordering of terms in polynomials is a key ingredient of both.

For the division algorithm on polynomials in one variable, then we are dealing with the degree ordering on the one-variable monomials:
\[
\cdot > x^{m+1} > x^m > \cdots > x^2 > x > 1.
\]

The success of the algorithm dividing \( f \) by \( g \) depends on working systematically with the leading terms in \( f \) and \( g \).

A major component of any extension of division and row-reduction to arbitrary polynomials in several variables is an ordering on the terms in polynomials in \( k[x_1, \ldots, x_n] \).
We can reconstruct the monomial \( x^a = x_1^{a_1}x_2^{a_2} \cdots x_n^{a_n} \) from the \( n \)-tuple of exponents \( a = (a_1, \ldots, a_n) \in \mathbb{Z}_{\geq 0}^n \). This observation establishes a one-to-one correspondence between the monomials in \( k[x_1, \ldots, x_n] \) and \( \mathbb{Z}_{\geq 0}^n \).

Furthermore, any ordering \( > \) we establish on the space \( \mathbb{Z}_{\geq 0}^n \) will give us an ordering on monomials: if \( a > \beta \) according to this ordering, we will also say that \( x^a > x^\beta \).

We will require that our orderings be linear or total orderings. This means that for every pair of monomials \( x^a \) and \( x^\beta \), exactly one of the three statements

\[
x^a > x^\beta, \quad x^a = x^\beta, \quad \text{or} \quad x^a < x^\beta
\]

should be true.

**Definition:** A monomial ordering on \( k[x_1, \ldots, x_n] \) is any relation \( > \) on \( \mathbb{Z}_{\geq 0}^n \), or equivalently, any relation on the set of monomials \( x^a, \ a \in \mathbb{Z}_{\geq 0}^n \), satisfying:

1. \( > \) is a total (or linear) ordering on \( n \geq 0 \).
2. If \( a > \beta \) and \( \gamma \in \mathbb{Z}_{\geq 0}^n \), then \( a + \gamma > \beta + \gamma \).
3. \( > \) is a well-ordering on \( n \geq 0 \). This means that every nonempty subset of \( n \geq 0 \) has a smallest element under \( > \).

**Lemma:** An order relation \( > \) on \( n \geq 0 \) is a well-ordering if and only if every strictly decreasing sequence in \( \mathbb{Z}_{\geq 0}^n \),

\[
a(1) > a(2) > a(3) > \cdots
\]

eventually terminates.

**Proof:** See [1] pp 55.

**Example:** (Lexicographic Order). Let \( a = (a_1, \ldots, a_n) \) and \( \beta = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}_{\geq 0}^n \). We say that \( a >_{\text{lex}} \beta \) if, in the vector difference \( a - \beta \in \mathbb{Z}^n \), the leftmost nonzero entry is positive. We will write \( x^a >_{\text{lex}} x^\beta \) if \( a >_{\text{lex}} \beta \).

From now on, we will assume that some well-ordering is defined on the ring \( k[x_1, \ldots, x_n] \).

Now that an order of monomials has been fixed, we can define the following properties for polynomials:

**Definition:** Let \( f = \sum a_a x^a \) be a nonzero polynomial in \( k[x_1, \ldots, x_n] \) and let \( > \) be a monomial order.

1. The *multidegree* of \( f \) is: \( \text{multideg}(f) = \max(\alpha \in \mathbb{Z}^n : a_\alpha \neq 0) \)

   (the maximum is taken with respect to \( > \)).

2. The *leading coefficient* of \( f \) is: \( \text{LC}(f) = a_{\text{multideg}(f)} \in k \).
The leading monomial of \( f \) is: 
\[ \text{LM}(f) = x_{\text{multideg}(f)} \]
(with coefficient 1).

The leading term of \( f \) is:  
\[ \text{LT}(f) = \text{LC}(f) \cdot \text{LM}(f). \]

**Lemma:** Let \( f, g \in k[x_1, \ldots, x_n] \) be nonzero polynomials. Then:

(i) \( \text{multideg}(fg) = \text{multideg}(f) + \text{multideg}(g) \).

(ii) If \( f + g \neq 0 \), then \( \text{multideg}(f + g) \leq \max(\text{multideg}(f), \text{multideg}(g)) \). If, in addition, \( \text{multideg}(f) = \text{multideg}(g) \), then equality occurs.

From now on, we will assume that one particular monomial order has been selected, and that leading terms, etc., will be computed relative to that order only.

### 2.2.2 A Division Algorithm in \( k[x_1, \ldots, x_n] \)

The division algorithm for polynomials in \( k[x_1, \ldots, x_n] \) extends the algorithm for \( k[x] \). The goal is to divide \( f \in k[x_1, \ldots, x_n] \) by \( f_i \), \( \ldots, f_s \in k[x_1, \ldots, x_n] \). As we will see, this means expressing \( f \) in the form:

\[ f = a_1 f_1 + \cdots + a_s f_s + r \]

where the “quotients” \( a_1, \ldots, a_s \) and remainder \( r \) lie in \( k[x_1, \ldots, x_n] \). We will then see how the division algorithm applies to the ideal membership problem.

**Division Algorithm in \( k[x_1, \ldots, x_n] \):**

Fix a monomial order \( > \) on \( \mathbb{Z}_{\geq 0}^n \) and let \( F = (f_1, \ldots, f_s) \) be an ordered \( s \)-tuple of polynomials in \( k[x_1, \ldots, x_n] \). Then every \( f \in k[x_1, \ldots, x_n] \) can be written as 

\[ f = a_1 f_1 + \cdots + a_s f_s + r \]

where \( a_i, r \in k[x_1, \ldots, x_n] \), and either \( r = 0 \) or \( r \) is a linear combination, with coefficients in \( k \), of monomials, none of which is divisible by any of \( \text{LT}(f_1), \ldots, \text{LT}(f_s) \). We will call \( r \) a **remainder** of \( f \) on division by \( F \). Furthermore, if \( a f_i \neq 0 \) then we have,

\[ \text{multideg}(f) \geq \text{multideg}(a f_i). \]

**Proof.** We prove the existence of \( a_1, \ldots, a_s \) and \( r \) by giving an algorithm for their construction and showing that it operates correctly on any given input.

**Input:** \( f_1, \ldots, f_s, f \)

**Output:** \( a_1, \ldots, a_s, r \)

\[ a_1 := 0; \ldots; a_s := 0; \; r := 0 \]

\[ p := f \]

\[ \text{WHILE} \ p \neq 0 \text{ DO} \{ \]

\[ i := 1 \]

\( \text{(2.2.1)} \)
divisionoccurred := false

WHILE $i \leq s$ AND divisionoccurred = false DO {
    IF LT($f_i$) divides (p) THEN {
        $a_i := a_i + \text{LT}(p)/\text{LT}(f_i)$
        $p := p - (\text{LT}(p)/\text{LT}(f_i)) f_i$
        divisionoccurred:= true
    } ELSE {
        $i := i + 1$
    }
}

IF divisionoccurred = false THEN {
    $r := r + \text{LT}(p)$
    $p := p - \text{LT}(p)$
}

Unfortunately, this division algorithm does not share the same nice properties as in the one-variable case. The division algorithm in $k[x]$ has the following important properties:

(i) The quotient and remainder are uniquely determined. This is not the case in the multi-variate version. The results of the above algorithm depend on the order of the $s$-tuple of divisors, $(f_1,..., f_s)$, and both the quotients {$a_i$} and the remainder {$r$} can change if the {$f_i$} are re-arranged.

(ii) The division algorithm in $k[x]$ solves the ideal membership problem, meaning that a polynomial $f(x) \in I = \langle g(x) \rangle$ iff the remainder $r$ of $f : g$ is zero. Again, this is not the case in the multi-variate version. The condition $r=0$ for $f : (f_i,..., f_s)$ is a sufficient condition for $f \in \langle f_1,..., f_s \rangle$, but not a necessary one (i.e. there may exist $f \in \langle f_1,..., f_s \rangle$ such that $r \neq 0$).

Thus, we must conclude that the division algorithm given in (2.2.1) is an imperfect generalization of its one-variable counterpart. To remedy this situation, may ask if there is a different, “good” set of polynomials that generate $\langle f_1,..., f_s \rangle$ for which the two properties listed above hold for the multi-variate division algorithm. Namely, we require that for such a set the remainder $r$ on division by the “good” generators is uniquely determined and the condition $r = 0$ should be equivalent to membership in the ideal.

### 2.2.3 Groebner bases

The solution for the problems posed by the division algorithm in $k[x_1,...,x_n]$ is found in the theory of Groebner bases, which provide a “good” generating set for any ideal in $k[x_1,...,x_n]$. In [1] sections 4 to 6 of chapter 2 are devoted to laying the foundations of the theory of Groebner
bases, and the interested reader is referred there for the full derivation and proofs of the propositions made in this section. We shall only specify the main results which are relevant to our subject.

We start our discussion of Groebner bases by examining the leading terms of the members of an ideal \( I \subset k[x_1, \ldots, x_n] \).

**Definition:** Let \( I \subset k[x_1, \ldots, x_n] \) be an ideal other than \( \{0\} \).

(i) We denote by \( \text{LT}(I) \) the set of leading terms of elements of \( I \). Thus,
\[
\text{LT}(I) = \{cx^\alpha : \text{there exists } f \in I \text{ with } \text{LT}(f) = cx^\alpha \}.
\]

(ii) We denote by \( \langle \text{LT}(I) \rangle \) the ideal generated by the elements of \( \text{LT}(I) \).

Note that the set \( \text{LT}(I) \) may be an infinite set, but as we shall see immediately, this does not mean that \( \langle \text{LT}(I) \rangle \) does not have a finite set of generators:

**Proposition:** Let \( I \subset k[x_1, \ldots, x_n] \) be an ideal. Then there are \( g_1, \ldots, g_t \in I \) such that
\[
\langle \text{LT}(I) \rangle = \langle \text{LT}(g_1), \ldots, \text{LT}(g_t) \rangle.
\]

**Proof:** See [1] pp. 76.

The following theorem is central to the theory of Groebner bases:

**Theorem (Hilbert Basis Theorem).** Let \( I \subset k[x_1, \ldots, x_n] \) be an ideal other than \( \{0\} \). Then:

(i) \( I \) has a finite generating set. That is \( I = \langle g_1, \ldots, g_t \rangle \) for some \( g_1, \ldots, g_t \in I \).

(ii) The generating set is given by the generators of \( \langle \text{LT}(I) \rangle = \langle \text{LT}(g_1), \ldots, \text{LT}(g_t) \rangle \).

**Proof:** We give a sketch of the full proof found in [1] pp. 76-77. Start with the group \( \{g_1, \ldots, g_t\} \) with satisfies \( \langle \text{LT}(I) \rangle = \langle \text{LT}(g_1), \ldots, \text{LT}(g_t) \rangle \). The fact that \( \langle g_1, \ldots, g_t \rangle \subset I \) is immediate since every \( g_t \) is in \( I \). Next, it is shown that \( I \subset \langle g_1, \ldots, g_t \rangle \), and thus \( I = \langle g_1, \ldots, g_t \rangle \). This is done by taking some \( f \in I \), dividing it by \( \langle g_1, \ldots, g_t \rangle \) and examining the remainder \( r \). By virtue of the division algorithm, no term of \( r \) divides by any leading term \( \text{LT}(g_i) \). On the other hand, it is shown that \( \langle \text{LT}(r) \rangle \subset \langle \text{LT}(g_1), \ldots, \text{LT}(g_t) \rangle \). The only \( r \) that satisfies both demands is 0, and thus \( f \in \langle g_1, \ldots, g_t \rangle \), which completes the proof.

We are now ready to define Groebner bases:

**Definition:** Fix a monomial order. A finite subset \( G = \{g_1, \ldots, g_t\} \) of an ideal \( I \) is said to be a Groebner basis (or standard basis) if,
\[ \{LT(I)\} = \{LT(g_1),...,LT(g_t)\} \]

Equivalently, but more informally, a set \( \{g_1,\ldots,g_t\} \subseteq I \) is a Groebner basis of \( I \) if and only if the leading term of any element of \( I \) is divisible by one of the \( \text{LT}(g_i) \).

**Proposition:** Every ideal \( I \subset k[x_1,\ldots,x_n] \) other than \( \{0\} \) has a Groebner basis. Furthermore, any Groebner basis for an ideal \( I \) is a basis of \( I \).

**Proof:** This follows almost immediately from the two claims of the Hilbert basis theorem.

Groebner bases have the following important properties:

**Proposition:** Let \( G = \{g_1,\ldots,g_t\} \) be a Groebner basis for an ideal \( I \subset k[x_1,\ldots,x_n] \) and let \( f \in k[x_1,\ldots,x_n] \).

1) The remainder \( r \) of the division of \( f \) by \( G \) is unique, no matter how the elements of \( G \) are listed when using the division algorithm. Formally, there is a unique \( r \in k[x_1,\ldots,x_n] \) with the following two properties:
   (i) No term of \( r \) is divisible by any of \( \text{LT}(g_1),\ldots,\text{LT}(g_t) \).
   (ii) There is \( g \in I \) such that \( f = g + r \).

2) \( f \in I \) if and only if the remainder on division of \( f \) by \( G \) is zero.

**Proof:** See [1] pp 82.

We now discuss how to test whether a given generator set of \( I \) is a Groebner basis, and how to construct a Groebner basis for an ideal \( I \) given some generating \( F = \{f_1,\ldots,f_s\} \).

**Definition:** Let \( f, g \in k[x_1,\ldots,x_n] \) be nonzero polynomials.

(i) If \( \text{multideg}(f) = \alpha \) and \( \text{multideg}(g) = \beta \), then let \( \gamma = (\gamma_1,\ldots,\gamma_n) \), where \( \gamma_i = \max(\alpha,\beta_i) \) for each \( i \). We call \( x^{\gamma} \) the **least common multiple** of \( \text{LM}(f) \) and \( \text{LM}(g) \), written \( x^{\gamma} = \text{LCM}(\text{LM}(f),\text{LM}(g)) \).

(ii) The **S-polynomial** of \( f \) and \( g \) is the combination

\[
S(f,g) = \frac{x^{\gamma}}{\text{LT}(f)} f - \frac{x^{\gamma}}{\text{LT}(g)} g
\]

**Theorem (Buchberger’s Criterion):** Let \( I \) be a polynomial ideal. Then a basis \( G = \{g_1,\ldots,g_t\} \) for \( I \) is a Groebner basis for \( I \) if and only if for all pairs \( i \neq j \), the remainder on division of \( S(g_i, g_j) \) by \( G \) (listed in some order) is zero.

The Buchberger Criterion tests whether a generating set is a Groebner basis. The next algorithm constructs a Groebner basis from a given generating set:

**Theorem (Buchberger’s Algorithm):** Let \( I = \langle f_{i_1}, \ldots, f_{i_s} \rangle \neq \{0\} \) be a polynomial ideal. Then a Groebner basis for \( I \) can be constructed in a finite number of steps by the following algorithm:

Input: \( F = (f_{i_1}, \ldots, f_{i_s}) \)

Output: a Groebner basis \( G = (g_{i_1}, \ldots, g_{t}) \) for \( I \), with \( F \subseteq G \)

\[
G := F
\]

REPEAT

\[
G' := G
\]

FOR each pair \( \{p, q\}, p \neq q \) in \( G' \) DO

\[
S := S(p, q) \mod G'
\]

IF \( S \neq 0 \) THEN \( G := G \cup \{S\} \)

UNTIL \( G = G' \)

The Buchberger Algorithm given in (2.2.2) is one of the key results about Groebner bases. We have seen that Groebner bases have many nice properties, but, so far, it has been difficult to determine a Groebner basis for an ideal. The Buchberger algorithm gives us a method to construct such a basis for any ideal, given a generating set. As a side outcome, the algorithm may also output \( (g_{i_1}, \ldots, g_{t}) \) as a linear combination of \( (f_{i_1}, \ldots, f_{i_s}) \). We will see that the coefficients of this linear combination turn out to be useful in determining inverse encoders.

We finish this section with the concept of the **reduced Groebner basis**, which is the most compact representation of any ideal \( I \subseteq k[x_1, \ldots, x_n] \):

**Definition:** A reduced Groebner basis for a polynomial ideal \( I \) is a Groebner basis \( G \) for \( I \) such that:

(i) \( \text{LC}(p) = 1 \) for all \( p \in G \).

(ii) For all \( p \in G \), no monomial of \( p \) lies in \( \langle \text{LT}(G - \{p\}) \rangle \).

**Proposition:** Let \( I \neq \{0\} \) be a polynomial ideal. Then, for a given monomial ordering, \( I \) has a unique reduced Groebner basis.

**Proof:** See [1] pp. 92.
2.2.4 Quotients of Polynomial Rings

In this sub-section, we introduce the concept of a quotient ring, which is important for dealing with polynomials of a finite degree. This, in turn, will correspond to dealing with sequences of finite support and cyclic convolution, as is shown in the next chapter. The quotient ring extends the concept of a modulo operation known from the algebra of univariate polynomials. The proofs for the propositions given below are found in [1], Chapter 5.

**Definition:** Let \( I \subseteq k[x_1,\ldots,x_n] \) be an ideal, and let \( f, g \in k[x_1,\ldots,x_n] \). We say \( f \) and \( g \) are congruent modulo \( I \), written \( f \equiv g \mod I \), if \( f - g \in I \).

**Definition:** The quotient of \( k[x_1,\ldots,x_n] \) modulo \( I \), written \( k[x_1,\ldots,x_n]/I \), is the set of equivalence classes for congruence modulo \( I \):

\[
\{ f \in k[x_1,\ldots,x_n] : f \equiv 0 \mod I \}
\]

**Proposition:** The sum and product operations,

\[
[f] + [g] = [f + g] \\
[f] \cdot [g] = [f \cdot g]
\]

yield the same classes in \( k[x_1,\ldots,x_n]/I \) on the right-hand sides no matter which \( f' \in [f] \) and \( g' \in [g] \) we use. (We say that the operations on classes given above are well-defined on classes.)

**Theorem:** Let \( I \) be an ideal in \( k[x_1,\ldots,x_n] \). The quotient \( k[x_1,\ldots,x_n]/I \) is a commutative ring under the sum and product operations given above.

**Definition:** Let \( R, S \) be commutative rings.

(i) A mapping \( \varphi : R \to S \) is said to be a ring isomorphism if:

a. \( \varphi \) preserves sums: \( \varphi(r + r') = \varphi(r) + \varphi(r') \) for all \( r, r' \in R \).

b. \( \varphi \) preserves products: \( \varphi(r \cdot r') = \varphi(r) \cdot \varphi(r') \) for all \( r, r' \in R \).

c. \( \varphi \) is one-to-one and onto.

(ii) Two rings \( R, S \) are isomorphic if there exists an isomorphism \( \varphi : R \to S \). We write \( R \cong S \) to denote that \( R \) is isomorphic to \( S \).

(iii) A mapping \( \varphi : R \to S \) is a ring homomorphism if \( \varphi \) satisfies properties (a) and (b) of (i), but not necessarily property (c), and if, in addition, \( \varphi \) maps the multiplicative identity \( 1 \in R \) to \( 1 \in S \).

**Definition:** A subset \( I \) of a commutative ring \( R \) is said to be an ideal in \( R \) if it satisfies:

(i) \( 0 \in I \) (where \( 0 \) is the zero element of \( R \)).

(ii) If \( a, b \in I \), then \( a + b \in I \).

(iii) If \( a \in I \) and \( r \in R \), then \( r \cdot a \in I \).

**Corollary:** Every ideal in the quotient ring \( k[x_1,\ldots,x_n]/I \) is finitely generated.
3 Algebraic Aspects of 2-D Tail-Biting Convolutional Codes

This chapter and the next follow the formulations found in [26], [27], [23]. These works deal with m-D convolutional codes which are not tail biting. We will follow these formulations only for the 2-D case (though most results can be easily extended to the 2-D case), and make the appropriate modifications the formulation to fit tail-biting convolutional codes.

In the first section of this chapter we discuss the representation of 2-D sequences and their relationship with the ring of bivariate polynomials \( k[x, y] \). We also analyze operations on finite 2-D sequences with support \( N_1 \times N_2 \) and their relationship with the quotient ring \( R/\langle x^{N_1} + 1, y^{N_2} + 1 \rangle \). This section touches on the algebraic theory of modules, delve too deeply into it. For a background on modules we refer the reader to [2]

The second section deals with the basic definitions of 2-D tail-biting convolutional codes. We define the concept of a 2-D tail-biting convolutional code, and show how to encode it using a generator matrix. In order for the codes to be useful, they have to be bijective, or non-degenerate. A degenerate code maps more than one information sequence to a single code sequence, and is therefore useless. We give a condition that easily tests whether 2-D TBCCs are non-degenerate. We finish the section by demonstrating that 2-D TBCCs are a subset of the wider family of Quasi-Cyclic Codes.

The third section introduces the concept of parity check matrices (or syndrome formers) and inverse encoders. These concepts are very important to the decoding process of 2-D TBCCs. We discuss the condition a 2-D TBCC has to satisfy in order to have a syndrome former or an inverse encoder, and give algorithms for their construction in the case they exist. We finish the section by discussing the relationship of syndrome formers and inverse encoders to decoding algorithms.

3.1 Representation of 2-D Information

Let \( F = F_q \) be a finite field with \( q \) elements. Let \( N \) be the set of nonnegative integers, and let \( N^2 \) represent a 2-D lattice with axes \( i, j \in N \).

Definition: A 2-D sequence over \( F \) is a map: \( u : N^2 \rightarrow F \), that associates the coordinates \((i,j)\) of \( N^2 \) with the elements \( u(i,j) \in F \). The sequence \( u \) is said to have finite support if \( u(i,j) = 0 \) for all but finitely many \( (i,j) \in N^2 \).

Next, we consider the case when there is more than one element of the field attached to each coordinate of the lattice.

Definition: If there are \( k \) elements of \( F \) attached to each coordinate of \( N^2 \), then a 2-D composite sequence over \( F \) is a map \( u : N^2 \rightarrow F^k \), that associates the coordinates \((i,j)\) of \( N^m \) with the vectors \( [u^{(1)}(i,j), ..., u^{(k)}(i,j)] \) in the k-tuple vector space \( F^k \). The sequence \( u = [u^{(1)}, ..., u^{(k)}] \) is called a composite sequence, because it can be viewed as being made up of \( k \) interleaved 2-D sequences \( u^{(j)}, j = 1, ..., k \) with elements \( u^{(j)}(i,j) \in F \).
**Definition:** A finite 2-D sequence space with dimensions $N_1 \times N_2$ of $F$ is the set of all mappings: $S_{N_1 \times N_2} = \{ u : N^2 \rightarrow F \}$, where every 2-D sequence $u$ has finite support with dimensions $N_1 \times N_2$, i.e.,

$$S_{N_1 \times N_2} = \{ u(i, j) | u(i, j) = 0 \text{ for } i < 0, i \geq N_1', j < 0, j \geq N_2 \}$$

(3.1.1)

In particular, the finite 2-D sequence space $S_{N_1 \times N_2}$ has the following distinguished sequences:

a. The additive identity in $S$ is the zero sequence, where $u(i, j) = 0$ for all $(i, j) \in N^2$.

b. The multiplicative identity in $S$ is the sequence with $u(i, j) = 0$ for all but the origin $(0, ..., 0)$ of $N^2$, and $u(0, ..., 0) = 1$.

For a given $m$, we define an addition and a multiplication of sequences as follows.

a. The sum $w = u + v$ of two sequences $u, v$ is given by:

$$w(i, j) = u(i, j) + v(i, j)$$

(3.1.2)

where $u(i, j) + v(i, j)$ denotes addition of $u(i, j)$ and $v(i, j)$ in $F$.  

b. The product (discrete 2-D cyclic convolution) $w = u \otimes v$ of two sequences $u, v$ is given by:

$$w(i, j) = \sum_{l=0}^{N_1-1} \sum_{k=0}^{N_2-1} u((i - k) \mod N_1, j - l) \mod N_2 \cdot v(k, l)$$

(3.1.3)

where $u(i, j) \cdot v(k, l)$ denotes multiplication of $u(i, j)$ and $v(k, l)$ in $F$.

It is easy to see that the sequence space $S_{N_1 \times N_2}$ is closed under both these operations, that is, $w_1, w_2 \in S_{N_1 \times N_2}$.

**Definition:** A finite 2-D composite sequence space with dimensions $N_1 \times N_2$ of $F^k$ is the set of all mappings: $S^k_{N_1 \times N_2} = \{ u : N^m \rightarrow F^k \}$, where every 2-D composite sequence $u = (u^{(1)}, ..., u^{(k)})$ consists of $u^{(j)} \in S^1_m$ with finite support.

For given $m$ and $k$, the following operations on composite sequences are well defined:

a. The sum $w = u + v$ of two composite sequences $u = (u^{(1)}, ..., u^{(k)})$, $v = (v^{(1)}, ..., v^{(k)})$ in $S^k_m$ given by $w^{(j)} = u^{(j)} + v^{(j)}$ in $S^1_m$.

b. Scalar multiplication $w = \alpha \otimes u$ of a composite sequence $u = (u^{(1)}, ..., u^{(k)})$ in $S^k_m$ and a sequence $\alpha \in S^1_m$ is given by $w^{(j)} = \alpha^{(j)} \otimes u^{(j)}$ in $S^1_m$.

We can now easily verify that $S$ satisfies all the axioms of a commutative ring with a multiplicative identity. In fact, a finite 2-D sequence space of $F$ is isomorphic to a bivariate polynomial ring over $F$. Specifically the isomorphic ring is:
Let us first define the following notations:

- Let $R = F[x, y]$ be the polynomial ring of bivariate polynomials in $x$ and $y$.
- Let $I = \langle x^{N_1} - 1, y^{N_2} - 1 \rangle$ be the ideal generated by $x^{N_1} - 1$ and $y^{N_2} - 1$, i.e.:
  \[ I = \{ h \mid h = f \cdot (x^{N_1} - 1) + g \cdot (y^{N_2} - 1), f, g \in k[x, y] \} \]
- Let $\overline{R} := R / I$ be the ring of polynomials with $\deg x < N_1$ and $\deg y < N_2$.

A 2-D sequence in $S$ can be uniquely mapped to a bivariate polynomial $u(x, y)$ in $\overline{R}$ with the transformation,

\[ \psi : S_{N_1 \times N_2} \rightarrow \overline{R} ; u \rightarrow \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} u(i, j)x^i y^j \quad (3.1.5) \]

by associating the coordinates of $N^2$ with monomials of $R$ via the correspondence,

\[ (i, j) \leftrightarrow x^i y^j \]

An element $u(i, j) \in F$ of the sequence $u \in S$ at the coordinate $(i, j)$ becomes the coefficient of the term $x^i y^j$ of the polynomial $u(x, y) \in R$.

Since the product of two sequences is carried out using discrete cyclic convolution in the domain $S_{N_1 \times N_2}$ and polynomial multiplication in the range $R$, the bijective map $\psi : S_{N_1 \times N_2} \rightarrow \overline{R}$ is an $F$-isomorphism with the law of composition,

\[ \psi(u \otimes v) = \psi(u) \psi(v) \]

consider the addition and multiplication operation described above, under this transformation:

a. **Addition**: $\psi(u) + \psi(v) = \psi(u + v)$

Proof:

\[ \psi(u) + \psi(v) = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} u(i, j)x^i y^j + \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} v(i, j)x^i y^j = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} [v(i, j) + u(i, j)]x^i y^j = \psi(u + v) \]

b. **Multiplication**: $\psi(u) \psi(v) \mod I = \psi(u \otimes v)$

Proof:

\[
\psi(u) \psi(v) \mod I = \left( \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} u(i, j)x^i y^j \right) \left( \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} v(i, j)x^i y^j \right) \mod I = \ldots
\]

\[
... = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} \sum_{k=0}^{N_1-1} \sum_{l=0}^{N_2-1} u(i, j)v(k, l)x^{i+k} y^{j+l} \mod I = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} \sum_{k=0}^{N_1-1} \sum_{l=0}^{N_2-1} u(i, j)v(k, l)x^{(i+k)(l+j)\mod N} y^{l+j\mod N} = \ldots
\]

\[
... = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} \sum_{k=0}^{N_1-1} \sum_{l=0}^{N_2-1} u(i, j)v(k-i\mod M, l-j\mod N)x^{i} y^{j} = \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} (u \otimes v)x^{i} y^{j} = \psi(u \otimes v)
\]
Thus, the finite sequence space $S_{N_1 \times N_2}$ is isomorphic to the finite ring $\overline{\mathbb{R}}$.

A 2-D composite sequence $u \in S^k$ is transformed into a vector of $k$ bivariate polynomials in the $k$-tuple $\mathbb{R}$-module $R^k$ using the $F$-isomorphism,

$$\psi : S_{N_1 \times N_2} \rightarrow \overline{\mathbb{R}} : \left[ u^{(1)}, ..., u^{(k)} \right] \rightarrow \left[ \psi(u^{(1)}), ..., \psi(u^{(k)}) \right]$$

(3.1.6)

The sequence space to polynomial ring transformation is useful for analyzing the structural properties of sequence spaces. For $m = 1$, the univariate polynomial ring $\mathbb{R} = F[x]$ is a principal ideal domain (PID), whereas for values of $m > 1$ the multivariate polynomial ring $\mathbb{R} = F[z_1, ..., z_m]$ is a noetherian ring but not a PID. Modules are natural generalizations of vector spaces to rings, where the scalars are taken from the ring over which they are defined instead of a field. A submodule of $R^k$ is a nonempty subset that is closed under addition and scalar multiplication. Since $S^k$ is isomorphic to $R^k$, it is natural to define subsets of $S^k$ that are analogous to submodules of $R^k$. In order to do this we need to first define the following operation on $S$.

**Definition:** A cyclic right-shift operation on a sequence $u \in S_{N_1 \times N_2}$, denoted by $x^i y^j(u) \mod I$ is defined as,

$$x^i y^j : S_{N_1 \times N_2} \rightarrow S_{N_1 \times N_2} ; u \rightarrow \psi^{-1}\left( x^i y^j \cdot \psi(u) \mod I \right)$$

(3.1.7)

The sequence space $S_{M \times N}$ is closed under the cyclic right-shift operation, i.e.,

$$x^i y^j(u) \mod I \in S_{N_1 \times N_2} \text{ for } \forall i, j$$

(3.1.8)

The exponents $i,j$ of the variables $x,y$ represent the delay or the amount by which the sequence $u$ is shifted to the right along the each dimension of sequence space $S_{N_1 \times N_2}$.

**Definition:** A 2-D composite sequence subspace of $S^k$ is a nonempty subset $C$ that satisfies the following closure properties:

a. $F$-linearity: If $u, v \in C$, then the $F$-linear combination $f_1 u + f_2 v \in C$ for all $f_1, f_2 \in F$.

b. Right-shift invariance: If $u \in C$, then $x^i y^j(u^{(1)}, ..., u^{(k)}) \in C$ for all $i,j \in \mathbb{N}^2$. That is, $C$ is invariant with respect to shifts in $\mathbb{N}^2$ along the coordinate axes.

From the $F$-linearity and right-shift invariance of $C$ it follows that: If $u \in C$, then $\alpha \ast u \in C$ for all $\alpha \in S$.

By virtue of the $F$-linearity and right-shift invariance of $C \subseteq S^k$, the $F$-isomorphism defined in (3.1.7) transforms $C$ into a $\mathbb{R}$-submodule $\psi(C)$ of $R^k$. If $C$ is any composite sequence subspace of $S^k$, we can speak of linear combinations, linear dependence and linear independence just as we do in modules. A set of elements $\{ u_1, ..., u_r \}$ of $C$ is said to generate (or span) $C$ if every $u \in C$ is a linear combination:
\[ u = s_1u_1 + \cdots + s_ru_r, \text{ with } s_i \in S, i=1,\ldots,r. \]  

(3.1.9)

We call the set of elements \{u_1,\ldots,u_r\} of \( C \) independent if no nontrivial linear combination is zero. That is,

\[ s_1u_1 + \cdots + s_ru_r = 0, \text{ with } s_i \in S, \text{ then } s_i = 0, i = 1,\ldots,r. \]  

(3.1.10)

A generating set is a basis if it is independent. We must be careful not to apply to \( C \) any vector space results that depend on division by nonzero scalars. In particular, if \( u_1,\ldots,u_r \) are linearly dependent, we cannot conclude that some \( u_i \) is a linear combination of the others. Here we note that for \( m = 1 \), since \( R \) is a PID any submodule of \( R^k \) is free (i.e. it has a basis). The well known 1-D convolutional codes can be viewed as submodules of \( R = \mathbb{F}[x] \). For \( m > 1 \), since \( R \) is a noetherian ring any submodule of \( R^k \) is finitely generated but is not necessarily free. That is, for \( m > 1 \) every submodule of \( R^k \) has a finite number of generators, but some submodules of \( R^k \) may not have a basis.

In the next section, we will see that this is a crucial reason which makes the generalization of convolutional codes to multidimensional spaces nontrivial.

### 3.2 Definition of 2-D Tail-Biting Convolutional Codes

In the following we will consider the finite 2-D composite sequence space \( S_{N_1 \times N_2}^k \) to be the information space or the input space. Our goal is to protect input sequences in \( S_{N_1 \times N_2}^k \), by adding redundancy such that the resulting encoding scheme is cyclic-shift-invariant with respect to the coordinate axes of the 2-D space.

To achieve this we have to inject the input sequences into a larger space \( S_{N_1 \times N_2}^n \), where \( k < n \). We will refer to the finite 2-D composite sequence space \( S_{N_1 \times N_2}^n \) as the output space. Let the \( R \)-modules \( \overline{R}^k \) and \( \overline{R}^n \) be the equivalent polynomial representations of \( S_{N_1 \times N_2}^k \) and \( S_{N_1 \times N_2}^n \) transformed using the F-isomorphism defined in (3.1.6).

**Note:** To avoid cumbersome notation we will henceforth use the symbol “\( z \)” to represent the variables \( x,y \) in the 2-D case, \( z_1,\ldots,z_m \) in the m-D case. Whenever necessary, the context (dimension) will be made clear by explicitly defining the value of \( m \).

#### 3.2.1 Codes, Generator Matrices and Encoders

**Definition:** A multidimensional code of rate \( k/n \) is a map \( C : S_m^k \rightarrow S_m^n \), that associates composite sequences of order \( k \), \( u \in S_m^k \) with composite sequences of order \( n \), \( v \in S_m^n \).

**Note:** In this dissertation we will usually limit ourselves to rate \( 1/n \) 2-D codes, i.e. to codes that map \( C : S_2^k \rightarrow S_2^n \). An 2-D convolutional code
Let \( G(z) \in \overline{R}^{k \times n} \) be a rectangular \((k<n)\) bivariate polynomial matrix with elements \( g_{i,j}(z) \in \overline{R} \),

\[
G(z) = \begin{pmatrix}
g_{1,1}(z) & \cdots & g_{1,n}(z) \\
\vdots & \ddots & \vdots \\
g_{k,1}(z) & \cdots & g_{k,n}(z)
\end{pmatrix}
\tag{3.2.1}
\]

Consider the \( R \)-module homomorphism induced by \( G(z) \) between the \( R \)-modules \( \overline{R}^k \) and \( \overline{R}^n \),

\[
\overline{R}^k \xrightarrow{G(z)} \overline{R}^n \iff u(z) \rightarrow v(z)
\tag{3.2.2}
\]

A input polynomial vector \( u(z) \in \overline{R}^k \) is encoded by left multiplication (modulo-I) with \( G(z) \) to obtain the output polynomial vector \( v(z) \in \overline{R}^n \). The image \( C = \text{im}[G(z)] \subset \overline{R}^n \) generated by the row space of \( G(z) \) is an \( R \)-submodule of \( \overline{R}^n \).

\[
C = \{ v(z) \in \overline{R}^n \mid v(z) = u(z)G(z) \mod I, \text{ for } u(z) \in \overline{R}^k \}
\tag{3.2.3}
\]

Since the \( R \)-matrix \( G(z) \) generates the \( R \)-module \( C \), we refer to it as the generator matrix of \( C \).

The rank of the generator matrix \( G(z) \) is equal to largest integer \( l \leq k \) for which there is nonzero (modulo-I) \( l \times l \) minor of \( G(z) \).

In order to be able to reconstruct the input \( u(z) \) from the output \( v(z) \) we require the generator map to be injective; that is, the generator matrix should be of full row rank.

**Definition:** A generator matrix \( G(z) \in \overline{R}^{k \times n} \) is called a 2-D tail-biting convolutional encoder if it has rank \( k \).

A \( R \)-module is said to be free if it has a basis. The number of elements in the basis is called the rank of the free \( R \)-module. Clearly, the input space \( \overline{R}^k \) and the output space \( \overline{R}^n \) are free \( R \)-modules with rank \( k \) and \( n \) respectively because they contain the \( k \)-tuple and \( n \)-tuple standard basis vectors. Since a convolutional encoder is a generator matrix that has full row rank, its rows form a basis for the \( R \)-module \( C \) that it generates.

**Definition:** A \( m \)-D convolutional code \( C \) is a free \( R \)-submodule of \( \overline{R}^n \). The elements of \( C \) are called codewords. If the rank of \( C \) is \( k \), then the rate of the code is \( k/n \).

A convolutional code and its encoder constitute different entities. But the two are related by the fact that one cannot exist without the other. This is especially important when considering codes over modules. For \( m = 1 \) we have \( R = F[x] \), and so the definition reduces to that of a convolutional code defined over a univariate polynomial ring. Here, we refer to such a code as a 1-D convolutional code. Since \( R = F[x] \) is a PID, every \( R \)-submodule of \( R^n \) is free. Therefore, for \( m = 1 \) the term “free” can be dropped from the definition. This is no longer true for \( m > 1 \). The multivariate polynomial ring \( R = F[z_1, \ldots, z_m] \) is a noetherian ring but not a PID, and so every \( R \)-submodule of \( R^n \) is finitely generated but is not necessarily free. The spanning set of a finitely
generated $R$-submodule can be used to construct the rows of its generator matrix. But if an $R$-submodule of $R^n$ is not free, it will not admit an m-D convolutional encoder. This implies that it is not possible to inject elements from any input space $R^k$; $1 < k < n$, into the $R$-submodule, and for this reason we cannot use it as a convolutional code.

In this section we discuss algebraic aspects of rate $1/n$ 2-D TBCC that are relevant to the encoding and decoding process. Unless stated otherwise, the encoder is defined by a generator matrix:

$$G(x, y) = [g_1(x, y), g_2(x, y), \ldots, g_n(x, y)],$$  (3.2.4)

And operates over the sequences with finite support $N_1 \times N_2$,

$$u(x, y) \in \overline{R} = R/I ,$$  (3.2.5)

Where,

$R=F[x,y]$ is the bivariate polynomial ring, and

$I = \langle x^{N_1} - 1, y^{N_2} - 1 \rangle$ is the ideal generated by $x^{N_1} - 1, y^{N_2} - 1$.

### 3.2.2 Non-Degenerate Codes

Consider the rate $1/n$ 2-D TBCC, described above. The encoder induces a map,

$$G(z) : \overline{R} \rightarrow \overline{R}^n \Leftrightarrow v = u \cdot G(z),$$  (3.2.6)

such that: $u \in R \rightarrow v = (v_1, \ldots, v_n)$, $v_j \in \langle g_j \rangle/I$

For the encoder to be useful in a communications system, it has to represent faithfully any information sequence $u$. In other words, the map has to be invertible or one-to-one. That is, for any $u_1, u_2 \in \overline{R}$ the statement $v_1 = u_1 \cdot G(x, y) \neq u_2 \cdot G(x, y) = v_2$ holds.

Since the code is linear, the above statement is equivalent to requiring,

$$\forall u \in \overline{R}, u \neq 0 \Leftrightarrow v = u \cdot G \neq 0 ,$$  (3.2.7)

**Definition:** A rate $1/n$ encoder $G$ is *non-degenerate* iff there is no input other than $u=0$ such that $v=0$. Alternatively $G$ is *degenerate* if there exists an input $u \neq 0$ such that $v=0$.

We now give a simple condition to test whether an encoder $G$ is degenerate over a given input sequence space. The following proposition was formulated and proven by Yaron Shani:

**Proposition:** Let $\langle g_1, g_2, \ldots, g_n \rangle$ be the ideal generated by the kernel polynomials $g_1, \ldots, g_n$. A rate $1/n$ tail-biting convolutional encoder $G$ is non-degenerate iff:
\[ \langle x^{N_1} - 1, y^{N_2} - 1 \rangle : \langle g_1, \ldots, g_n \rangle = \langle x^{N_1} - 1, y^{N_2} - 1 \rangle, \]

(3.2.8)

**Proof:** The proposition \( I \subseteq \langle I : J \rangle \) holds for any two ideals \( I, J \), therefore,

\[ \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \subseteq \langle \langle x^{N_1} - 1, y^{N_2} - 1 \rangle : \langle g_1, \ldots, g_n \rangle \rangle \]

On the other hand, if \( G \) is degenerate,

\[ \exists f \notin \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \text{ such that } f \cdot g_j \notin \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \text{ for all } j=1,2,\ldots,n \]

\[ \Rightarrow \exists f \notin \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \text{ such that } f \cdot \langle g_1, \ldots, g_n \rangle \subseteq \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \]

\[ \Rightarrow \exists f \notin \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \text{ such that } f \in \langle \langle x^{N_1} - 1, y^{N_2} - 1 \rangle : \langle g_1, \ldots, g_n \rangle \rangle \]

From the last statement it follows that iff \( G \) is degenerate, then

\[ \langle \langle x^{N_1} - 1, y^{N_2} - 1 \rangle : \langle g_1, \ldots, g_n \rangle \rangle \neq \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \]

Inverting the statement we get that iff \( G \) is non-degenerate,

\[ \langle \langle x^{N_1} - 1, y^{N_2} - 1 \rangle : \langle g_1, \ldots, g_n \rangle \rangle = \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \]

Then \( G \) is non-degenerate. ■

This condition is easily tested for any given encoder in an algebraic software such as CoCoA [3]. From now on we will assume that we are dealing with non-degenerate codes.

An important observation that can be made from the above proposition, is that an encoder may be non-degenerate for a finite ring \( R/I_1 \), but degenerate over another finite ring \( R/I_2 \), as is illustrated by the following example:

**Example:** Consider the following generator matrix:

\[ G(x, y) = (x + y + 1, \ xy + x + y + xy) \Leftrightarrow g_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \ g_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \]

We let \( J \) be the ideal generated by the members of \( G(x, y) \), \( J = \langle x + y + 1, \ xy + x + y \rangle \)

This code is non-degenerate over sequences with support 5×5, represented by the ring \( R/I_1 \). This follows from a computation in CoCoA, which shows:

\[ I_1 : J = \langle x^5 + 1, \ y^5 + 1 \rangle : \langle x + y + 1, \ xy + x + y \rangle = \langle x^5 + 1, \ y^5 + 1 \rangle = I_1 \]

However, the code is degenerate over 6×6 sequences, represented by the ring \( R/I_2 \). Testing in CoCoA, we get:

\[ I_2 : J = \langle x^6 + 1, \ y^6 + 1 \rangle : \langle x + y + 1, \ xy + x + y \rangle = \langle x^6 + 1, \ y^6 + 1, \ u_0 \rangle \neq I_2 \]

Where,

\[ u_0 = x^5 + x^4 y^5 + x^5 y^3 + x^3 y^5 + x^4 y^3 + x^3 y^4 + x^5 y + x^4 y^2 + x^4 y^2 + x^5 y + x^3 + \ldots \]

\[ + x^3 y^2 + x^2 y^3 + y^5 + x^4 + x^3 y + y^4 + x^2 y + x y^2 + x^2 + y^2 + x + y \]

\[ u_0 = x^5 + x^4 y^5 + x^5 y^3 + x^3 y^5 + x^4 y^3 + x^3 y^4 + x^5 y + x^4 y^2 + x^4 y^2 + x^5 y + x^3 + \ldots \]

\[ + x^3 y^2 + x^2 y^3 + y^5 + x^4 + x^3 y + y^4 + x^2 y + x y^2 + x^2 + y^2 + x + y \]

\[ u_0 = x^5 + x^4 y^5 + x^5 y^3 + x^3 y^5 + x^4 y^3 + x^3 y^4 + x^5 y + x^4 y^2 + x^4 y^2 + x^5 y + x^3 + \ldots \]

\[ + x^3 y^2 + x^2 y^3 + y^5 + x^4 + x^3 y + y^4 + x^2 y + x y^2 + x^2 + y^2 + x + y \]
The polynomial \( u_0 \) is the information word which is mapped to the zero codeword, which can be easily verified:

\[
\begin{align*}
\text{either} & : \quad u_0 \cdot (x + y + 1) = (y^4 + y^3 + y + 1)(x^6 + 1) + (x^4 + x^3 + x + 1)(y^6 + 1) = 0 \mod I_2 \\
\text{either} & : \quad u_0 \cdot (xy + x + y) = (y^5 + y^3 + y^2 + 1)(x^6 + 1) + (x^5 + x^3 + x^2 + 1)(y^6 + 1) = 0 \mod I_2 \\
\end{align*}
\]

### 3.2.3 Relationship with Quasi-Cyclic Codes

In this section we demonstrate that 2-D TBCCs are Quasi-cyclic code. For the sake of clarity, we bring the following definition from [8]:

**Definition:** A \((n,k)\) binary linear code of dimensions \(mn_0, mk_0\) is called Quasi-cyclic iff every cyclic shift of a codeword by \(n_0\) bits is also a codeword.

It is well known that the one-dimensional cyclic convolution operation can be represented as a multiplication of a vector by a circulant matrix.

Let \(u[k], g[k]\) are sequences with finite support \(N\). We take \(N\) to be the maximum of the supports of the two sequences. If the sequence \(g\) has support \(K<N\), than we simply pad it with zeros, \(g[K]=g[K+1]=...=g[N-1]=0\). Then their cyclic convolution is given by:

\[
v[i] = g[i] \otimes u[i] = \sum_{k=0}^{N-1} g[i-k \mod N] u[k], \quad (3.2.9)
\]

Which can be written in the vector form:

\[
v = u \cdot \text{circ}(g), \quad (3.2.10)
\]

Where \(u=[u(0), u(1),..., u(N-1)]\) is the sequence \(u\) in row vector form, and \(\text{circ}(g)\) is the circulant matrix, whose rows (and columns) are cyclic shifts of the vector \(g=[g(0), g(1),..., g(N-1)]\),

\[
\text{circ}(g) = \begin{pmatrix}
  g(0) & g(1) & g(2) & \cdots & g(N-1) \\
  g(N-1) & g(0) & g(1) & \cdots & g(N-2) \\
  g(N-2) & g(N-1) & g(0) & & \vdots \\
  \vdots & \vdots & \vdots & \ddots & g(1) \\
  g(1) & g(2) & g(3) & \cdots & g(0)
\end{pmatrix}, \quad (3.2.11)
\]

In two dimensions we have:

\[
v[i, j] = g[i, j] \otimes u[i, j] = \sum_{k=0}^{N_1-1} \sum_{l=0}^{N_2-1} g[i-k \mod N_1, j-l \mod N_2] u[k, l], \quad (3.2.12)
\]

Converting to vector form we have:
\[ v[j] = \sum_{l=0}^{N-1} u[l] \text{circ}(g[l - j \mod M]) = \sum_{l=0}^{N-1} u[l] \cdot G_{l-j}, \quad (3.2.13) \]

Where we define,
\[ u[l] = [u(0,l), u(1,l), \ldots, u(N_2 - 1, l)] \]
\[ g[l] = [g(0,l), g(1,l), \ldots, g(N_2 - 1, l)] \]
\[ G_j = \text{circ}(g[l \mod N_1]) \]

Now, if we define \( u \) as a row stack of \( u[l] \), \( u = [u[0], u[1], \ldots, u[N-1]] \), we can re-write the 2-D convolution as,

\[ v[j] = [u[0], u[1], \ldots, u[N-1]]. \begin{bmatrix} G_{N-1-j} \\ G_{N-1-j+1} \\ \vdots \\ G_{N-1-j} \end{bmatrix} = u \cdot \begin{bmatrix} G_{N-1-j} \\ G_{N-1-j+1} \\ \vdots \\ G_{N-1-j} \end{bmatrix}, \quad (3.2.14) \]

Or, taking \( v \) as a row stack of \( v[j] \), \( v = [v[0], v[1], \ldots, v[N-1]] \), we finally have,

\[ v = [v[0], v[1], \ldots, v[N-1]] = [u[0], u[1], \ldots, u[N-1]]. \begin{bmatrix} G_0 & G_{N-1} & \cdots & G_1 \\ G_{N-1} & G_0 & \cdots & G_1 \\ \vdots & \vdots & \ddots & \vdots \\ G_{N-1} & G_{N-2} & \cdots & G_0 \end{bmatrix} = u \cdot G, \quad (3.2.15) \]

Where we define,
\[ G = \begin{bmatrix} G_0 & G_1 & \cdots & G_{N-1} \\ G_{N-1} & G_0 & \cdots & G_1 \\ \vdots & \vdots & \ddots & \vdots \\ G_1 & G_2 & \cdots & G_0 \end{bmatrix} = \text{circ(circ}(g)), \quad (3.2.16) \]

We see therefore, that the 2-D cyclic convolution operation can be described in matrix form by taking the row stack of the input sequence \( u \) and multiplying it by the matrix \( G \) which is a\textit{ circulant of circulants}, formed by zero-padding the kernel sequence \( g \) and forming circulant matrices from its rows.

For a rate 1/n 2-D convolutional code with a polynomial space generator matrix,\n\[ G(x, y) = [g_1(x, y), g_2(x, y), \ldots, g_n(x, y)] \]
We can form the sequence space generator matrix by taking,
\[ G^{(i)} = \text{circ}(\text{circ}(g_i)) \]
And the generator matrix \( G \) is given by:
\[ G = [G^{(1)}, G^{(2)}, \ldots, G^{(n)}] \]

**Example:** Take the following rate ½ code over 4×4 input sequences:
\[ G(x, y) = (x + y + 1, xy + x + y) \leftrightarrow g_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, g_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \]

We wish to find the sequence-space generator matrix of the code. We start by padding the kernel sequences \( g_1 \) and \( g_2 \) to the support of the input sequence space,
\[ g_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}; \quad g_2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \]

We now form circulants from the columns of \( g_1, g_2 \), \( G^{(i)}_k = \text{circ}(g_i[k]) \), where \( i=1,2 \) and \( k=1,2,3,4 \) is the column index:
\[
G^{(1)}_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}; \quad G^{(2)}_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad G^{(3)}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}; \quad G^{(4)}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}
\]
\[
G^{(2)}_2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}; \quad G^{(2)}_3 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}; \quad G^{(2)}_4 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \]

Finally, we create the generator matrix, which is a block matrix of the matrices \( G^{(1)} \) and \( G^{(2)} \), which are circulants of circulants:
\[
G^{(i)} = \begin{bmatrix} G^{(i)}_1 & G^{(i)}_2 & G^{(i)}_3 & G^{(i)}_4 \\ G^{(i)}_1 & G^{(i)}_2 & G^{(i)}_3 & G^{(i)}_4 \\ G^{(i)}_2 & G^{(i)}_3 & G^{(i)}_4 & G^{(i)}_1 \end{bmatrix}, \quad i=1,2
\]
\[ G = [G^{(1)}, G^{(2)}] = \]
27

\[
\begin{pmatrix}
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\]

Where zeros in bold (\textbf{0}) denote 4×4 blocks of zeros.

We have shown that a rate 1/n 2-D TBCC over the sequence space with support \(N_1 \times N_2\) is equivalent to a code generated by a generator matrix \(N_1N_2 \times nN_1N_2\) composed of circulants matrices of size \(N_2 \times N_2\). But such a code is exactly a \((nN_1N_2, N_1N_2)\) quasi-cyclic code [8].

\textbf{Conclusion:} Rate \(k/n\) 2-D TBCC over the sequence space with support \(N_1 \times N_2\) are equivalent to a \((nN_1N_2, kn_1N_2)\) quasi-cyclic code with a generator matrix composed of \(kN_1 \times nN_1\) circulants of size \(N_2 \times N_2\). Thus, every shift by \(N_2\) bits of a codeword, is also a codeword.

We leave further investigation of the properties of 2-D TBCCs as quasi-cyclic codes for future research. For discussion of quasi-cyclic codes, the interested reader is referred to [8], [9] and [10].

3.3 Inverse Encoders and Parity Check Matrices

In this section we introduce the concepts of the syndrome former and the inverse encoder. These concepts are very important in the decoding process as we shall see later in this dissertation.

3.3.1 Parity Check Matrices

We first consider the concept of parity check matrices, and their application to the decoding of codes.

\textbf{Definition:} A parity check matrix \(H\) for a generator matrix \(G\) of a code \(C\) is any matrix that satisfies the following condition:

For any \(n\)-tuple of polynomials \(v(x,y)=[v_1(x,y),\ldots,v_n(x,y)]\), \(v \cdot H^T = 0\) iff \(v\) is a valid codeword of \(C\), i.e. there exists a \(k\)-tuple \(u(x,i)\) such that \(v = u \cdot G\).

A corollary of this definition is that since by definition any linear combination of the rows of \(G\) is valid codeword of \(C\), we have the following identity:
\( G \cdot H^T = 0 \)  \hspace{1cm} (3.3.1)

**Proposition:** If \( (g_1, g_2, \ldots, g_n) \) have no common divisors except for units, then \( G \) has a parity check matrix \( H \) of the form,

\[
H(x, y) = \begin{pmatrix}
g_2 & g_3 & \cdots & g_n \\
g_1 & 0 & 0 & \vdots \\
0 & g_1 & \ddots & 0 \\
0 & \cdots & 0 & g_1
\end{pmatrix}
\]  \hspace{1cm} (3.3.2)

**Proof:** Let \( u \in R/I \) and \( v = u \cdot G \). Then by construction of \( H \):

\[
v \cdot H = u \cdot G \cdot H = u \cdot [g_1g_2 + g_2g_1, \ldots, g_1g_n + g_ng_1] = [0, \ldots, 0]
\]

On the other hand, let \( v \not\in \text{Im}(G) \) and suppose that \( v \cdot H = 0 \).

Then, for \( g_1, g_j, j=2, \ldots, n \) we have,

\[
v_i g_j + v_j g_1 = 0
\]

We can write \( v_i \) as: \( v_i = g_j u + r_j \), where \( r_j \parallel g_j \) then,

\[
0 = (g_ju + r_j)g_j + (g_ju + r_j)g_1 = r_j g_j + r_j g_1
\]

\[
\Rightarrow g_1 = -\frac{r_j g_j}{r_j}
\]

Now since \( g_j \) and \( g_j \) are polynomials and \( r_j \parallel g_j \), then \( r_j/r_j \) must be polynomial, which means that \( g_1 | g_j \), which contradicts the assumption that \( g_1 \) and \( g_j \) have no common divisors except for units. Therefore \( v \in \text{Im}(G) \) \( \blacksquare \)

A more general procedure for constructing parity check matrices for codes of rate \( k/n \) is given in [28], which makes use of Groebner bases (Actually, the procedure uses Groebner bases for modules, which are an extension of the Groebner bases for ideals discussed in section 2.2.3).

The parity check matrix is related to the concept of a syndrome which can be used to detect errors in a codeword transmitted over a noisy channel. Suppose a codeword \( v \) is transmitted over a noisy channel, and a corrupted copy \( r = v + e \) is received. The vector \( e \) is the *error polynomial*. Then, by applying the parity check matrix \( H \) to the received vector \( r \), we get:

\[
s = r \cdot H^T = (v+e) \cdot H^T = v \cdot H^T + e \cdot H^T = e \cdot H^T
\]  \hspace{1cm} (3.3.3)

The vector \( s \) is the syndrome of the error polynomial \( e \). The immediate use of a syndrome is the detection of errors: Since for any valid codeword the syndrome is zero, a non-zero syndrome indicates that an error has occurred. The syndrome may also be used for decoding codes, by creating lists of the error patterns and associating them with their syndromes. To decode a word,
we first compute the syndrome, then select the most probable error pattern from the list of that syndrome, and add it to the received word. The result is the most probable valid codeword that has been transmitted. However, such a scheme is largely impractical due to the immense number of possible syndromes. For very specific linear codes efficient algebraic methods exist to extract the most probable error pattern from the syndrome. However, no such methods are known for 2-D convolutional codes.

### 3.3.2 Inverse Encoders

Consider the use of a 2-D convolutional encoder $G(x,y)$ in a communications system. Given an input $u$, the encoder generates a codeword $v = uG$, which is transmitted over a noisy channel. At the output of the channel the decoder operates on the received vector $r$ to produce an estimate $\hat{u}$ of the input sequence $u$.

The decoder can be split conceptually into two parts: (a) A codeword estimator which produces an estimate $\hat{v}$ of the codeword $v$, followed by (b) An encoder inverse that recovers the input estimate $\hat{u}$ from $\hat{v}$.

A non-degenerate rate $1/n$ code necessarily has an encoder inverse. For a rate $1/n$ code, which is a $n \times 1$ column vector $G^{-1}(x,y) = [q_1(x,y), q_2(x,y), ..., q_n(x,y)]$, such that,

$$G(x,y)G^{-1}(x,y) = 1$$

(3.3.4)

In general, the encoder inverse may be rational (i.e. the elements of $G^{-1}$ are taken from $Q = \mathbb{F}(x,y)$, the field of fractions of $R$ and not necessarily a polynomial vector.

[does it mean that every encoder is invertible?]

Here we wish to make an important distinction which relates to the existence of an inverse encoder to a given encoder $G(x,y)$. An encoder $G(x,y)$ can be defined for any finite polynomial ring $R/I$, or for the infinite ring, $R$. While the encoder exists independently of the ring over which it operates, the existence of a polynomial inverse encoder depends on that ring, as we shall see below.

**Definition:** A polynomial $g(x,y)$ is a unit:

a. In $R$ iff there is a polynomial $p(x, y) \in R$ such that,

$$g(x, y)p(x, y) = 1$$

(3.3.5)

b. In $R/I$ iff there is a polynomial $p(x, y) \in R/I$ such that,

$$g(x, y)p(x, y) = 1 \mod I$$

(3.3.6)

We call $p(x,y)$ the inverse of $g(x,y)$ in $R$ or $R/I$, respectively.

From this definition we can already conclude, that if $G(x,y) = [g_1(x,y), ..., g_n(x,y)]$ contains a polynomial $g_j(x,y)$ which is a unit in $R$ (or $R/I$) then there exists an inverse encoder $G^{-1}(x,y)$, for which all members are zero excepts the $j$-th member, which is the inverse of $g_j(x,y)$ in $R$ (or $R/I$).
Proposition: A polynomial \( g(x,y) \) is a unit in \( R/I \), \( I = \langle q_1, \ldots, q_s \rangle \) iff the ideal generated by \( \langle g, q_1, \ldots, q_s \rangle = R \).

Proof: By definition.

Specifically for \( I = \langle x^{N_1} - 1, y^{N_2} - 1 \rangle \), \( g(x,y) \) is a unit in \( R/I \) iff \( \langle g, x^{N_1} - 1, y^{N_2} - 1 \rangle = R \).

The following proposition is well known:

Proposition: A rate \( 1/n \) convolutional encoder has a polynomial inverse encoder in \( R \) iff \( (g_1, g_2, \ldots, g_n) \) generate the ring \( R \) as an ideal

Proof: By definition, an inverse encoder \( G^{-1}(x,y) = [q_1(x,y), q_2(x,y), \ldots, q_n(x,y)] \) satisfies:

\[
\sum_{i=1}^{n} q_i(x,y)g_i(x,y) = 1
\]

Which means that \( 1 \in \langle g_1, \ldots, g_n \rangle \), which in turn means \( \langle g_1, \ldots, g_n \rangle = R \), which completes the proof. ■

We now generalize the last proposition to the case of 2-D TBCCs.

Proposition: A rate \( 1/n \) convolutional encoder has a polynomial inverse encoder in \( R/I \) iff \( (g_1, g_2, \ldots, g_n, x^{N_1} - 1, y^{N_2} - 1) \) generate the ring \( R \) as an ideal.

Proof: The requirement that \( (g_1, g_2, \ldots, g_n, x^{N_1} - 1, y^{N_2} - 1) \) generate \( R \) as an ideal, means that there exist polynomials \( a, b \in R/I \) such that, \( a(x^{N_1} - 1) + b(y^{N_2} - 1) + 1 \) divides by \( (g_1, g_2, \ldots, g_n) \). This in turn mean:

\[
\sum_{i=1}^{n} q_i(x,y)g_i(x,y) = a(x^{N_1} - 1) + b(y^{N_2} - 1) + 1 \mod I
\]

Where all \( q_i \) are polynomials. Therefore by definition the encoder inverse is \( G^{-1}(x,y) = [q_1(x,y), q_2(x,y), \ldots, q_n(x,y)] \), which is polynomial. ■

Note that more than one inverse encoder may exist for a given encoder. For example, if any of \( (g_1, g_2, \ldots, g_n) \) are units, then each inverse of a unit generator is an inverse encoder.

From the last two propositions we can see that if an encoder has a polynomial inverse in \( R \), denoted by \( G^{-1}(x,y) \), then it necessarily has an inverse encoder in \( R/I \), which is simply \( G^{-1}(x,y) \mod I \). However, the reverse is not true, as the following example demonstrates:

Example: Let’s consider again the encoder:
This encoder is not invertible or pseudo-invertible in $R$, since it’s Groebner basis is:

$$GB\left\{ x+y+1, \ xy+x+y \right\} = \left\{ x+y+1, \ y^2+y+1 \right\}$$

Which does not contain any monomials. However, over finite 4x4 sequences, we have:

$$GB\left\{ x+y+1, \ xy+x+y, \ x^4+1, \ y^4+1 \right\} = \{1\}$$

Which means the encoder is polynomially invertible in $R/I(x^4+1, \ y^4+1)$. In fact, at least two polynomial inverses exist:

$$G_1^{-1}(x,y) = (x^3+x^2y+xy^2+x+y^3+y+1, \ 0)$$
$$G_2^{-1}(x,y) = (0, \ x^3y^3+x^3y^2+x^2y^3+y^3)$$

From now on we shall refer to codes that are invertible in $R$ as universally invertible, or simply as invertible codes. On the other hand, we shall refer codes that are invertible for some finite ring $R/I$ as finitely invertible codes.

We may also define a pseudo-inverse encoder for a convolutional encoder $G(x,y)$. A pseudo inverse encoder is a matrix $G^\#(x,y)$ which reconstructs the information sequence $u$ from the codeword $v$, up to a two-dimensional delay. Formally, we make the following definition:

**Definition:** A matrix $G^\#(x,y)$ is a pseudo-inverse encoder to an encoder $G(x,y)$ iff the product of $G(x,y)$ and $G^\#(x,y)$ is a monomial:

$$G(x,y)G^\#(x,y) = x^\alpha y^\beta$$

We can interpret the result of applying the inverse encoder to a codeword $v$ as a cyclic right shift of the original information word $u$, and thus $u$ can be easily reconstructed.

$$v \cdot G^\#(x,y) = u \cdot G(x,y) \cdot G^\#(x,y) = u \cdot x^\alpha y^\beta$$

**Proposition:** A rate $1/n$ TBCC encoder has a polynomial pseudo-inverse encoder $P(x,y)$ iff the Groebner base of $\{g_1, g_2, ..., g_n, x^{N_1} - 1, y^{N_2} - 1\}$ contains the monomial $x^\alpha y^\beta$.

**Proof:** The procedure of constructing a Groebner base for an ideal described in equation (2.2.2) is by finding linear combinations of the original polynomials $(g_1, g_2, ..., g_n, x^{N_1} - 1, y^{N_2} - 1)$. If the Groebner base contains a monomial $x^\alpha y^\beta$, then there is some linear combination of the original polynomials that yields that monomial:

$$\sum_{i=1}^{n} q_i g_i = a(x^{N_1} - 1) + b(y^{N_2} - 1) + x^\alpha y^\beta = x^\alpha y^\beta \mod I$$

Therefore by definition the encoder has a polynomial pseudo-inverse, $G^\#(x,y)=[q_1(x,y), q_2(x,y), ..., q_n(x,y)]$.
We shall now see that in the quotient ring $\mathbb{R}/I$ a pseudo-invertible encoder is also an invertible encoder:

**Proposition:** A rate $1/n$ TBCC that has a polynomial pseudo-inverse encoder, also has a polynomial inverse encoder.

**Proof:** By multiplying both sides of equation (3.3.11) by $x^{N_1-a} y^{N_2-\beta}$ we get:

$$\sum_{i=1}^{n} x^{N_1-a} y^{N_2-\beta} q_i g_i = x^{N_1} y^{N_2} \mod I = 1 \mod I \quad (3.3.12)$$

Therefore the encoder has a polynomial inverse: $G^{-1}(x,y) = [q'_1(x,y), q'_2(x,y), ..., q'_n(x,y)]$, where $q'_i(x,y) = x^{N_1-a} y^{N_2-\beta} q_i(x,y)$. ■

At this point we conjecture (without proof) that every non-degenerate TBCC is also polynomially invertible. We were not able to prove this in the time frame of the research, but so far this conjecture is borne out by our simulation results.

Despite the last conjecture and the preceding proposition, using polynomial inverses over $\mathbb{R}/I$ has limited usefulness. First of all, if the inverse in $\mathbb{R}$ is not polynomial, then generally the polynomial inverse in $\mathbb{R}/I$ will have a degree similar to that of the input sequence, which makes the inverse operation unwieldy. In other cases, where a pseudo-inverse of low degree exists, it is usually more convenient to use it then convert it to a true inverse encoder, which has a higher degree.

From now on we shall refer to an encoder which is polynomially (pseudo-)invertible over $\mathbb{R}$ simply as a (pseudo-)invertible encoder, and disregard polynomial inverses over $\mathbb{R}/I$.

If an encoder is (pseudo-)invertible, the polynomials $\{q_i\}$ of the inverse encoder can be found as a by-product of the Buchberger algorithm from section 2.2.3.

### 3.3.3 Relationship to Belief Propagation Algorithms

In the following section we explore soft-decoding algorithms for 2DCCs. All these soft decisions algorithms will be shown to be Belief Propagation (BP) algorithms over graphs. The algebraic relationships described above are useful in constructing the graph representations of the soft decoding problem.

First, the generating equation:

$$v = u \cdot G \quad (3.3.13)$$

Gives rise to the well known Viterbi and BCJR algorithms which solve the problem for the Maximum-Likelihood (ML), and Maximum-A-posteriori-Probability (MAP) criteria, respectively. Unfortunately, for 2DCCs these algorithms have complexities that are exponential with the dimensions of the input $u$, and exponential with the dimensions of the components of $G$. We propose a sub-optimal algorithm similar to Viterbi and BCJR, with complexity that is linear...
with the input dimensions. However, this algorithm is not guaranteed to converge for all codes. These algorithms are explored in chapter 6.

Next we have the parity check equation:

\[ 0 = v \cdot H^T \]  \hspace{1cm} (3.3.14)

This equation is gives rise to the bi-partite Tanner graph commonly used in decoding Low-Density Parity-Check codes (LDPCs) [29][30]. It is known that if the bi-partite graph is a tree, then the belief propagation algorithm is optimal in the ML/MAP sense (depending on the variant used). For codes where \( H \) is sparse, the tree approximation holds. However, for most 2DCCs \( H \) is not sparse, and the resulting graph contains cycles, or loops. In this case, the tree approximation doesn’t hold, and the BP algorithm does not necessarily converge with the ML/MAP solution. The algorithm is then sometimes known as Loopy BP (LBP).

One of the approaches to solve the problem of loops, proposed in [34], [35], was to generalize the graph from a graph of single nodes to a graph of groups of nodes, or regions. This gives rise to the family of Generalized BP (GBP) algorithms, which are still based on the parity check equation above. Algorithms based on the PC equation are explored in section 7.

The problem with decoding based on the PC equation alone, is that once the algorithm has converged, we are left with an estimation of the codeword \( v \), rather than the information word \( u \). For systematic codes this is not a problem, as \( u \) forms a part of \( v \). For non-systematic codes we have to invert the encoding operation. One way of doing this is by using the (pseudo-)inverse encoding equation:

\[ v \cdot G^# = u \cdot x^\alpha y^\beta \]  \hspace{1cm} (3.3.15)

or in another form,

\[ v \cdot G^# + u \cdot x^\alpha y^\beta = \begin{bmatrix} v \\ u \end{bmatrix} \begin{bmatrix} G^# \\ x^\alpha y^\beta \cdot I \end{bmatrix} = 0 \]  \hspace{1cm} (3.3.16)

Where \( G^# \) is the pseudo inverse encoding matrix. We can construct this equation for every pseudo-inverse matrix the encoder has. Using these equations, we can then construct a bi-partite graph on which we can perform belief propagation to extract the unknown vector \( u \) from the decoded vector \( v \).

Again, if the code-inverse graph is a tree, we will have a correct approximation for the probability of \( u \). If the graph is not a tree, we can try to use GBP techniques to improve the results. Otherwise, we can resort to searching techniques, such as searching for the information word that fits the hard decisions of \( v \) (assuming this yields a valid codeword).
4 Distance Properties

Some of the basic properties of a linear code that allow to assess its usefulness in protecting data from errors, are its minimum distance and weight spectrum. In this section we discuss these properties, their definitions and methods to measure them. At the end of this section we present results for these properties for some 2-D TBCC.

4.1 Basic concepts

Let $C$ be a 2-D TBCC with a generator matrix $G$, and let $v$ be a codeword in $C$.

**Definition (Hamming weight):** The weight of $v$, denoted by $wt(v)$ is the number of non-zero elements in $v$.

Equivalently, if $v(x,y)$ is the polynomial representation of $v$, then $wt(v)$ is the sum of number of monomials in each of the components of the vector $v(x,y)$:

$$
wt(v(x,y)) = \sum_{i=1}^{n} wt(v_i(x,y))
$$

(4.1.1)

**Definition (Hamming Distance):** Let $v_1$, $v_2$ be codewords in $C$. The Hamming Distance between $v_1$ and $v_2$ is given by: $dist(v_1, v_2) = wt(v_1 - v_2)$.

The weight of a codeword $v$ is also its Hamming distance from the zero codeword, i.e., $wt(v) = dist(v, 0)$.

**Definition (minimum distance):** The minimum distance of a code $C$ is defined as:

$$
dist(C) = \min \{dist(v_i,v_j) | v_i, v_j \in C, v_i \neq v_j \}
$$

(4.1.2)

Since the code is linear, any linear combination of two codewords is also a codeword. Therefore, there exists a non-zero codeword $v_0$ such that,

$$
dist(C) = \min \{wt(v_0) | v_0 \in C, v_0 \neq 0 \}
$$

(4.1.3)

**Definition (spectral component):** The spectral component of a weight $\omega$ in a code $C$ is the number of codewords $\{v\}$ in $C$ such that $wt(v) = \omega$.

**Definition (weights spectrum):** The weights spectrum of a code $C$ is a vector $A(w)$, $w=0,1,2,\ldots$ where the $i$-th component of the vector is the spectral components of weight $w$ in $C$.

At this point we would like to make a few straightforward observations regarding the weights spectrum of a code $C$.

(i) If $C$ is non-degenerate (i.e., there is no information word $u \neq 0$ such that $v = C(u) = 0$), then the spectral component of $0$, $A(0) = 1$.

(ii) If $C$ is non-degenerate, then the first non-zero spectral component for $w > 0$ is for $w = dist(C)$.

(iii) If $C$ is of dimension $N$, than $A(w) = 0$ for $w > N$. 

Furthermore, for an m-D convolutional code with a generator matrix $G_{k \times n}$, the minimum distance of the code is upper-bounded by the minimum weight of all rows of $G$:

$$
dist(C) \leq \min_{i \in I} \{ \text{wt} (\text{row}(G(x, y))) \} = \min_{i \in I} \left\{ \sum_{j=1}^{n} \text{wt}(g_{i,j}(x, y)) \right\} \quad (4.1.4)
$$

The minimum distance and weights spectrum are useful in estimating the error-protection capability of a code. Over a binary-errors channel, the minimum distance determines the amount of errors the code can correct. If $d = \text{dist}(C)$, the code can correct up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors.

Over an AWGN channel, the weights spectrum allows us to calculate the Union Bound [Lin & Costello], which gives an upper bound to the probability of a decoding error, assuming a maximum-likelihood decoder.

The Union bound for a decoding word-error probability is:

$$
P_w \left( \frac{E_b}{N_0} \right) \leq \sum_{w=d_{\min}}^{N} A(w) \cdot Q \left( 2^{\frac{w}{2n}} \cdot \frac{E_b}{nN_0} \right) \quad (4.1.5)
$$

And for an information bit-error probability:

$$
P_b \left( \frac{E_b}{N_0} \right) \leq \frac{1}{k} \sum_{w=d_{\min}}^{N} A(w) \cdot B(w) \cdot Q \left( 2^{\frac{w}{2n}} \cdot \frac{E_b}{N_0} \right) \quad (4.1.6)
$$

Where $B(w)$ is the average number of bits of all information words that are mapped to codewords with weight $w$.

In practice, for large $E_b/N_0$, the first terms of the sum are dominant, as the function $Q(x)$ function decays very rapidly.

Unfortunately, calculating the minimum distance and weights spectrum of a linear block code is known to be an NP-hard problem [13][14]. On the other hand, convolutional codes in general have a structure which allows to compute the first few elements of their weight spectrum with a relatively small number of computations. In the following section we consider an algorithm for calculating the weight spectrum of 2-D TBCCs.

Another useful bound is the Sphere Packing Lower Bound (SPLB), derived by Shannon [11]. An excellent discussion of this bound can be found in [12].

The SPLB is a lower bound on the achievable word-error probability for codes with a finite word length.

$$
P_w(n, \frac{E_b}{N_0}) \geq \int_{0}^{\phi} \left( \frac{(n-1)(\sin \phi)^{n-2}}{\sqrt{\pi} \cdot 2^{n/2} \cdot \Gamma(\frac{n+1}{2})} \right) s^{n-1} e^{-\left( e^{-1/2} - e^{-1/8} \right)} ds d\phi \quad (4.1.7)
$$

Where $A = \sqrt{2(k/n)(E_b/N_0)}$ and $\phi$ solves the equation:
\[ \Omega_n(\theta) = \int_0^\theta \frac{(n-1)\Gamma(\frac{\theta}{2}+1)}{\sqrt{\pi} \cdot n \cdot \Gamma(\frac{\theta}{2})} (\sin \phi)^{n-2} d\phi = 2^{-k} \]  

(4.1.8)

### 4.2 BEAST algorithm

The BEAST algorithm (Bidirectional Efficient Algorithm for Searching Trees) was formulated in [17] for 1-D convolutional codes, both in their zero-terminated and tail biting forms. We give a short description of this algorithm, and demonstrate how it is extended and applied to 2-D convolutional tail-biting codes.

#### 4.2.1 1-D BEAST

The BEAST algorithm for 1-D tail biting codes presented in [17], searches for all codewords with weight \( w \). For this aim, a forward trellis and a backward trellis are constructed for each possible starting (and termination) state of the encoder. The trellises are kept small in size, so that only paths up to a certain weight and length are stored. Then the forward and backward trellises are searched for path pairs that are compatible: i.e., that the forward and backward trellises terminate in the same state, their combined length is equal to the length of the information sequence, and their combined code weight is equal to the desired weight \( w \). By searching for \( w=0,1,2,\ldots \) etc., the weight spectrum of the code is achieved.

Formally, the algorithm can be described as follows:

For a 1-D tail-biting code of rate \( k/n \), in order to find all codewords with weight \( w \), perform the following steps for each possible starting (termination) state, \( \sigma_0 \):

1) **Growing a forward tree.** A forward tree is described by the sets of paths \( F_j = \{ \pi_F \} \), \( j=0,1,\ldots,n-1 \), that start at state \( \sigma_0 \), and terminate at some set of arbitrary states \( \{ \sigma_F \} \). Each set is defined by constraints on the length and weight of the paths:

\[
F_j = \left\{ \pi_F \mid l_F(\pi_F) \leq l - \left\lfloor \frac{w/2}{n} \right\rfloor + j \text{ and } w_F(\pi_F) = \left\lfloor \frac{w}{2} \right\rfloor + j \right\} \tag{4.2.1}
\]

Where \( l_F \) is the length of the forward path and \( w_F \) is the weight of code bits accumulated along that path.

2) **Growing a backward tree.** A backward tree is described by the sets of paths \( B_j = \{ \pi_B \} \), \( j=0,1,\ldots,n-1 \), that terminate at state \( \sigma_0 \), and start at some set of arbitrary states \( \{ \sigma_B \} \). And complies with the following constraints:

\[
B_j = \left\{ \pi_B \mid l_B(\pi_B) \leq l - \left\lfloor \frac{w/2}{n} \right\rfloor + j \text{ and } w_B(\pi_B) = \left\lfloor \frac{w}{2} \right\rfloor - j \right\} \tag{4.2.2}
\]

Where \( l_B \) is the length of the backward path and \( w_B \) is the weight of code bits accumulated along that path.

3) **Compute the spectral component** \( A(w) \) of the code by counting all the pairs of paths in \( F_j \), \( B_j \), such that \( \sigma_F = \sigma_B \) and \( l_F + l_B = l \), for all starting states \( \sigma_0 \)

\[
A(w) = \sum_{\sigma_0} \sum_{j=0}^{n-1} \sum_{\{\pi_F, \pi_B\} \in F_j \times B_j} \chi(\pi_F, \pi_B) \tag{4.2.3}
\]

Where:
Example: Suppose we wish to find codewords with weight \( w \leq 5 \) of the 1-D CC with generator polynomials \((7,5)\) in octal notation. Then from equation (4.2.1) it follows that we have to grow a forward tree that includes all paths of \( w \leq 3 \). Similarly, from equation (4.2.2) it follows that the backward tree also includes all paths of \( w \leq 3 \). In Figure 4-1 we show the all resulting paths in the trellis, starting and terminating with state 00. Now we have to find compatible paths in the forward and backward trees that: a) terminate in the same state ; b) have total weight \( w \leq 5 \), and c) That their combined length equals the length of the information word. Let us assume the information is of length \( l=3 \). Then there is exactly one path in the diagram that meets the requirements: \( 00 \to 10 \to 01 \to 00 \) (denoted by the arc A). To find all the codewords with \( w \leq 5 \) we have to continue and construct forward and backward trees for the other possible starting-termination states, and count all the resulting codewords.

![Figure 4-1 Example: 1-D BEAST: Left: Forward Tree, Right: Backward Tree](image)

Growing the forward and backward trees is done by a track-back algorithm. At each node \( \sigma \) along a path \( \pi \), all the next branches are tested for the suitable length and weight conditions. If the conditions are not still met (i.e., the weight and length have not been exceeded), the algorithm continues to grow the path along that branch. If the conditions are exceeded, then the path is added to the appropriate set \( F_j \) or \( B_j \), or it is discarded.

The algorithms for growing \( F_j \) and \( B_j \) for tail-biting codes are shown in the form of flowcharts in Figure 4-2.
Figure 4-2 BEAST algorithm; left - forward recursion; right - backward recursion
4.2.2 2-D BEAST

The 1-D BEAST algorithm can be readily extended to two dimensions. The 1-D BEAST constructs a trellis of the code which consists of states and branches. In the 2-D case, we shall define a concept that binds together the concepts of state and branch.

Let \((g_1, g_2, \ldots, g_n)\) be the kernel polynomials of a 2DCC of rate \(1/n\). Let the kernels have maximum support \(K_1 \times K_2\). Then each code fragment at position \((i, j)\), \(v_{i,j}\) is uniquely determined by \(K_1 \times K_2\) information bits out of the information sequence \(u\).

**Definition:** A constraint region of a 2DCC code with kernels \(\{g_i\}, i=1,\ldots,n\) of support \(K_1 \times K_2\) is a sub-sequence of information bits \(R_{k,l} = \{u_{k,l}\} \subseteq u\), such that \(k = k_0, k_0 + 1, \ldots, k_0 + K_1 - 1\), and \(l = l_0, l_0 + 1, \ldots, l_0 + K_2 - 1\). This sub-sequence uniquely determines the code fragment \(v_{k,l} = \{v_{k,l}(1), \ldots, v_{k,l}(n)\}\).

Having defined the constraint region, we can see that the information sequence is composed of many overlapping constraint regions. Using this concept we can construct a “2-D trellis” composed of adjacent constraint regions. In the 1-D case the trellis has one axis, and the BEAST algorithm can advance either forward or backward along this axis. In the 2-D case, there are two axes, and consequently four directions: down, up, right and left. The up-down direction is determined by decrementing (up) or incrementing (down) the row index \(i\). The left-right direction is determined by decrementing (left) or incrementing (right) the column index \(j\).

Let us now define compatibility between two constraint regions in a certain direction if such that they contain the same bits in their overlap region.

**Definition:** Let \(R_1 = \{r_1(i,j)\}\) and \(R_2 = \{r_2(i,j)\}\) be two constraint regions, \(i = 0, \ldots, K_1 - 1, j = 0, \ldots, K_2 - 1\). Then \(R_2\) is,

\[ a. \text{ Up-compatible with } R_1 \text{ iff: } r_1(i,j) = r_2(i+1,j) \text{ for all } i = 0, \ldots, K_1 - 2, j = 0, \ldots, K_2 - 1 \]

\[ b. \text{ Down-compatible with } R_1 \text{ iff: } r_1(i,j) = r_2(i-1,j) \text{ for all } i = 1, \ldots, K_1 - 1, j = 0, \ldots, K_2 - 1 \]

\[ c. \text{ Right-compatible with } R_1 \text{ iff: } r_1(i,j) = r_2(i,j-1) \text{ for all } i = 0, \ldots, K_1 - 1, j = 1, \ldots, K_2 - 1 \]

\[ d. \text{ Left-compatible with } R_1 \text{ iff: } r_1(i,j) = r_2(i,j+1) \text{ for all } i = 0, \ldots, K_1 - 1, j = 0, \ldots, K_2 - 2 \]

A path in the 2-D trellis is defined by a set of adjacent compatible constraint regions.

To construct a forward path in a 2-D trellis is a path we start with the constraint region \(R_{0,0}\) and proceed in a down-then-right fashion. That is, we advance first in the down direction, by laying a sequence of down-compatible regions \(R_{0,0}, R_{2,0}, \ldots, R_{N_1,0}\). Once the last row has been reached, we go back to row 0 and advance in the right direction and lay region \(R_{1,0}\). We then continue in the down direction, laying regions that are down-compatible and right-compatible with previous regions.

To construct a backward path in the 2-D trellis, we start with the constraint region \(R_{N_1,1}, K_2 - 1\) and proceed in the same manner as with the forward path, except that we advance in an up-then-left fashion. That is, we lay up-compatible regions until the first row has been reached, upon which we wrap around, perform one step left, and continue in the up direction.

The length of the path is simply the number of constraint regions it contains.

The weight of the path is the sum of weights of all code fragments associated with the path’s constraint regions.
A pair of forward and backward paths are compatible with each other iff:

a. The sum of their length corresponds to the length of the information sequence
b. all the constraint regions contained along the boundary of the forward path are compatible with their corresponding regions on the boundary of the backward path.

The 2-D beast algorithm is now defined in the same way as the 1-D algorithm, except that states are replaced with constraint regions and the compatibility function $\chi(\xi_f, \xi_b)$ now denotes compatibility between a pair of forward and backward paths.

**Example:** Consider a 2-D TBCC with kernel support 2x2 over information support 4x4. In Figure 4-3 we show how its forward and backward trees are constructed. Diagrams (a)-(c) show the construction of the forward tree. We start by overlaying the kernel on the top-left corner of the information sequence. We then grow the paths by sliding the kernel down. There are 4 possible compatible regions, so our forward tree now contains 4 possible paths. We continue sliding the kernel down, growing the tree, until we reach the last row of the sequence. We then wrap around to the top (b), and slide the kernel right. If not all paths have exceeded the allowed weight, we can then continue sliding the kernel down. In this example, we show a forward path that exceeds the allowed weight and is terminated in (c). Diagrams (d)-(f) show the construction of the backward tree. This time we start at the bottom-right corner (d) and slide up, until the first row is reached, and then left (c) and up again. The backward path terminates in diagram (f). In Diagram (g) we show the overlapping between the forward and backward paths. The paths have to be compatible, so their overlapping bits are the same. They also have to satisfy the boundary conditions (not shown) imposed by the tail-biting: The leftmost and rightmost column must be the same, and the topmost and bottommost row must also be the same. If a pair of paths satisfies all these conditions, we can list its combined weight $w$ as a valid codeword of weight $w$.

![Figure 4-3 Example: 2-D BEAST algorithm](image-url)
is constructed is also binned according to their left or right boundary conditions (for forward and backward paths respectively).

### 4.3 Results for Sample Codes

In this section we give some results for small 2-D TBCC. We give the weights spectrums of the best codes found in this class, and plot the word-error and bit-error probability union bounds for some of these codes. Since calculating the entire weight distribution is prohibitive computationally, only the first few elements of the weight spectrum were calculated. However, since these are the dominant elements of the union bound, this has a small effect on the graphs.

We also note that for each of the codes in the table there may be more equivalent codes (i.e., codes with the same weight spectrum), that are achieved simply by rotating, transposing or flipping the kernel matrices. In the tables below we give only the “prototypes” of the best codes, to avoid prolonging the tables unnecessarily.

The search procedure was as follows:

1) First a CoCoA script was run to generate a list non-degenerate codes of given kernel support over a given information support.
2) A C++ program implementing the BEAST algorithm was used to calculate the first element of the weight distribution of each non-degenerate code.
3) A MATLAB script was run to eliminate equivalent codes and calculate the bounds.

#### 4.3.1 Kernel Support 2×2 ; Information Support 4×4

For K=2×2 and N=4×4 we have found 14 distinct non-degenerate codes, of which 12 codes have an inverse or a pseudo-inverse, and 7 are systematic. A histogram of the codes by their minimum distance is shown in Figure 4-4. The properties of the best codes found are given in Table 4-1. The best code found has $d_{\text{min}}=6$, which makes it a (32,16,6) code. At $P_w=10^{-3}$, this code is only ~0.9 dB from the SPLB for codes of length 32. The best invertible code and the best systematic code have $d_{\text{min}}=4$ and are at about ~1.8dB from the SPLB.

<table>
<thead>
<tr>
<th>Kernels</th>
<th>$d_{\text{min}}$</th>
<th>$A(d_{\text{min}}+i)$, $B(d_{\text{min}}+i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_1$, $g_2$</td>
<td>$i=0$</td>
<td>$i=1$</td>
</tr>
<tr>
<td>Best Code</td>
<td>6</td>
<td>32, 5.00</td>
</tr>
<tr>
<td>Best Invertible Code</td>
<td>4</td>
<td>4, 4.00</td>
</tr>
<tr>
<td>Best Systematic Code</td>
<td>4</td>
<td>8, 4.00</td>
</tr>
</tbody>
</table>
Figure 4-4 Histogram of codes by $d_{\text{min}}$; $K=2\times2$; $N=4\times4$

Figure 4-5 Performance asymptotes for codes; $K=2\times2$; $N=4\times4$

* Union bound asymptotes based on first 8 elements
4.3.2 Kernel Support 2×2 ; Information Support 6×6

For K=2×2 and N=6×6 we have found 12 distinct non-degenerate codes, all of which have an inverse or a pseudo-inverse, and 7 of them are systematic. A histogram of the codes by their minimum distance is shown in Figure 4-6. The properties of the best codes found are given in Table 4-2.

The best code found has $d_{\text{min}}=6$, which makes it a (72,36,6) code. Since the codeword is longer than for N=4×4, the SPLB is now ~1.4 dB better than this code. The best systematic code is ~2.5dB away from the SPLB.

<table>
<thead>
<tr>
<th>Code#</th>
<th>Kernels</th>
<th>$d_{\text{min}}$</th>
<th>$A(d_{\text{min}}+i)$</th>
<th>$B(d_{\text{min}}+i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$g_1$</td>
<td>$g_2$</td>
<td>$i=0$</td>
<td>$i=1$</td>
</tr>
<tr>
<td>Best Code (Invertible)</td>
<td>1, 1; 1, 1; 1, 1; 1, 1;</td>
<td>6</td>
<td>12, 6.00</td>
<td>36, 1.00</td>
</tr>
<tr>
<td>2nd Best Code</td>
<td>1, 1; 1, 0; 1, 1; 1, 1;</td>
<td>6</td>
<td>48, 3.00</td>
<td>0, 0.00</td>
</tr>
<tr>
<td>Best Systematic Code</td>
<td>1, 0; 0, 0; 1, 1; 1, 1;</td>
<td>5</td>
<td>36, 1.00</td>
<td>84, 2.57</td>
</tr>
</tbody>
</table>

Table 4-2 Best codes with kernel support 2×2 over information support 6×6

![Figure 4-6 Histogram of codes by $d_{\text{min}}$; K=2×2 ; N=6×6](chart.png)
4.3.3 Kernel Support 3×3; Information Support 6×6

To achieve a bigger minimum distance, bigger kernel sizes are required, with a larger weight. At $K=3\times3$, over $N=6\times6$ we have found 6147 distinct codes, of which 1617 are invertible or pseudo-invertible, and 92 are systematic. It is interesting to note that there are many codes with high $d_{\text{min}}$:

- For $d_{\text{min}}=12$ there are 235 codes, of which 15 are (pseudo-)invertible.
- For $d_{\text{min}}=11$ there are 346 codes, of which 11 are (pseudo-)invertible.
- For $d_{\text{min}}=10$ there are 1259 codes, of which 144 are (pseudo-)invertible.

When compared to the SPLB at $P_w=10^{-3}$, the best code found is only ~0.31dB away from the bound, while the two invertible codes shown in Table 4-1 are ~0.34dB and ~0.35dB away. The best systematic code has $d_{\text{min}}=8$, and is ~0.64 dB from the SPLB at $P_w=10^{-3}$, however this gap increases for lower $P_w$ because of its smaller minimum distance.
Table 4-3 Best codes with kernel support 3×3 over information support 6×6

<table>
<thead>
<tr>
<th>Kernels</th>
<th>(d_{\text{min}})</th>
<th>(A(d_{\text{min}}+1))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(i=0)</td>
</tr>
<tr>
<td>Best Code</td>
<td>1, 1, 0; 1, 0, 1; 0, 0, 1; 1, 1, 1;</td>
<td>12</td>
</tr>
<tr>
<td>Best Invertible Code</td>
<td>1, 1, 1; 1, 0, 0; 1, 1, 1; 0, 1, 0; 0, 1, 1;</td>
<td>12</td>
</tr>
<tr>
<td>Simple Invertible Code</td>
<td>1, 0, 1; 1, 1, 0; 0, 1, 0; 0, 1, 1;</td>
<td>12</td>
</tr>
<tr>
<td>Best Systematic Code</td>
<td>1, 0, 0; 1, 1, 0; 0, 0, 0; 1, 0, 1;</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 4-8 Histogram of codes by \(d_{\text{min}}\); \(K=3\times3\); \(N=6\times6\)

* We define a simple invertible code as one with a pseudo-inverse matrix \(G^\#\) with weight \(\text{wt}\{G^\#\} \leq 4\) (see chapter 7.1.2)
Figure 4-9 Performance asymptotes for codes; K=3×3; N=6×6

4.4 Comparison with other codes

Now that we have established the weight distributions (or at least the first few elements of those distributions), it is interesting to compare the performance of the codes we have found with known codes of other families. We have selected the following code families as comparison points:

1) One dimensional tail biting convolutional codes (1-D TBCCs)
2) Short Low Density Parity Check Codes (LDPCCs).
3) Primitive binary BCH codes

Where possible, we compare the union bounds on the word error probability for maximum likelihood decoding. If the union bound is not available, then we plot the performance of common decoding techniques. The 2-D TBCCs are decoded with a 1-D version of the Viterbi algorithm (see section 6.1).

4.4.1 Compared to 1-D CCs

The first natural comparison is with one-dimensional CCs. In order to have a fair comparison, the 1-D CCs and 2-D CCs need to have the same constraint size. Therefore, a 2-D CC with constraint size \( K_1 \times K_2 \) should be compared with a 1-D CC of constraint length \( K=K_1K_2 \). Similarly, the message size also has to be identical, and thus we have \( N=N_1N_2 \).

Lin & Costello give a list of optimal 1-D CCs with rate \( \frac{1}{2} \) as ([14], pp. 540). The weight distribution of these codes is found easily using the BEAST algorithm, and summed in Table 4-4.
Where the generator polynomials are given in conventional octal notation.

It can be seen that for rate ½, 1-D TBCCs are similar in their weight distributions to comparable 2-D TBCCs and therefore one can expect that their performance will also be similar. This is borne out by Figure 4-10 to Figure 4-12. In each of these figures the best 2-D CC found for a given constraint size and information size is compared with its closest 1-D CC counterpart. In each figure the Union bound asymptote is plotted for both codes, and also their performance under a 1-D Viterbi decoder [6]. The uncoded WER is also plotted for reference purposes. It can be seen that 1-D and 2-D TBCCs perform very similarly to each other.
Figure 4-11 Comparison with 1-D CC(13,17)

Figure 4-12 Comparison with 1-D CC(561,753)
4.4.2 Compared to short LDPCCs

Our next comparison is with short Low Density Parity Check Codes (LDPCCs). LDPCCs were devised by Gallager [29], and have proven to be very strong for large block sizes. LDPCCs can be decoded in a maximum likelihood fashion using belief propagation algorithms (see sections 5.1 and 7.1) provided that their Tanner graph [30] contains no cycles. It has been shown by Wei [32] that short LDPCCs can perform only at about 1.5dB away from the sphere packing bound. For simplicity, we have chosen to limit ourselves to regular LDPCCs only. A regular LDPCC is characterized by the triplet $(d_v, d_c, N)$, where $N$ is the information block size, $d_v$ is the variable node degree and $d_c$ is the check node degree. The parameters $d_v$ and $d_c$ characterize the parity check matrix $H$ of the code. Each row of $H$ describes a parity check equation or a check node, and contains exactly $d_c$ ‘1’s. Each column of $H$ corresponds to a coded bit, or variable node, and contains exactly $d_v$ ‘1’s. This means that every code bit participates in exactly $d_v$ parity check equations, and each parity check equation contains exactly $d_c$ code bits. It can be easily shown that the code rate is given by $R=1-d_v/d_c$. We denote the LDPCCs by $H(d_v, d_c, N)$.

We have randomly generated the following LDPCCs, which we describe in sparse matrix notation. I.e., we give the positions of ‘1’s in the matrix $H$ as a series of pairs $(i, j)$, where $i$ denotes the row index, and $j$ denotes the column index.

1) The code $H(2,4,16)$, which is comparable to a 2-D TBCC with $K=2\times2$ and $N=4\times4$. The parity check matrix of this code is given below, and has no short cycles of length 4 (achieved by repeated randomizations). Its performance in comparison to the 2-D TBCC is shown in Figure 4-13. It can be seen that the LDPCC performs ~1.5dB worse than the 2-D TBCC.

```
H(2, 4, 16)={
(1, 4), (1, 6), (1, 7), (1, 30), (2, 5), (2, 6), (2, 17), (2, 19),
(3, 1), (3, 2), (3, 17), (3, 23), (4, 13), (4, 15), (4, 18), (4, 20),
(5, 4), (5, 11), (5, 26), (5, 28), (6, 7), (6, 14), (6, 21), (6, 22),
(7, 5), (7, 21), (7, 23), (7, 24), (8, 2), (8, 8), (8, 13), (8, 24),
(9, 3), (9, 12), (9, 22), (9, 32), (10, 1), (10, 9), (10, 15), (10, 31),
(11, 3), (11, 16), (11, 18), (11, 27), (12, 9), (12, 14), (12, 20), (12, 28),
(13, 8), (13, 12), (13, 16), (13, 29), (14, 10), (14, 27), (14, 29), (14, 30),
(15, 19), (15, 25), (15, 26), (15, 31), (16, 10), (16, 11), (16, 25), (16, 32),
};
```

2) The code $H(2,4,36)$, which is given below. This code is not free of short cycles, but has the smallest number of short cycles among 10,000 randomizations. In Figure 4-14 we compare it with a 2-D TBCC with $K=2\times2$, $N=6\times6$, and find that the LDPCCs is only 0.3 dB worse than the 2-D TBCC. However, in Figure 4-15 we compare this code with the best 2-D TBCC with $K=3\times3$, $N=6\times6$. Here the gap is larger, at about ~1.2 dB.

```
H(2, 4, 36)=
(1, 6), (1, 37), (1, 38), (1, 60), (2, 16), (2, 30), (2, 38), (2, 46),
(3, 5), (3, 21), (3, 53), (3, 68), (4, 25), (4, 35), (4, 40), (4, 54),
(5, 6), (5, 22), (5, 25), (5, 41), (6, 8), (6, 30), (6, 62), (6, 65),
(7, 17), (7, 39), (7, 43), (7, 51), (8, 20), (8, 21), (8, 32), (8, 44),
(9, 24), (9, 37), (9, 41), (9, 55), (10, 11), (10, 19), (10, 28), (10, 46),
(11, 2), (11, 19), (11, 36), (11, 47), (12, 15), (12, 18), (12, 20), (12, 48),
```
3) The code $H(3, 6, 36)$, which is comparable in size to 2-D TBCCs with and $N=6\times6$. The parity check matrix of this code is given below, and has no short cycles. Its performance in comparison to a $K=2\times2$ 2-D TBCC is shown in Figure 4-14. It can be seen that the LDPCC performs ~2.0dB worse than this 2-D TBCC. In Figure 4-15 we also compare this code with the best 2-D TBCC with $K=3\times3$, $N=6\times6$. Here the gap is even larger, at about ~3.0 dB.

\[
H(3, 6, 36) = \{
(1, 12), (1, 25), (1, 34), (1, 58), (1, 68), (1, 70), (2, 20), (2, 30), (2, 49), (2, 52), (2, 53), (2, 57),
(3, 2), (3, 9), (3, 19), (3, 36), (3, 51), (3, 60), (4, 18), (4, 19), (4, 25), (4, 33), (4, 34), (4, 40),
(5, 18), (5, 23), (5, 24), (5, 25), (5, 46), (5, 62), (6, 1), (6, 6), (6, 15), (6, 17), (6, 20), (6, 42),
(7, 4), (7, 11), (7, 16), (7, 34), (7, 41), (7, 63), (8, 14), (8, 15), (8, 17), (8, 18), (8, 21), (8, 65),
(9, 3), (9, 13), (9, 14), (9, 22), (9, 59), (9, 68), (10, 10), (10, 14), (10, 20), (10, 29), (10, 43), (10, 49),
(11, 28), (11, 31), (11, 39), (11, 45), (11, 47), (11, 62), (12, 5), (12, 13), (12, 31), (12, 40), (12, 46), (12, 61),
(13, 9), (13, 38), (13, 46), (13, 56), (13, 57), (13, 71), (14, 10), (14, 13), (14, 23), (14, 37), (14, 47), (14, 53),
(15, 10), (15, 16), (15, 29), (15, 35), (15, 38), (15, 58), (16, 8), (16, 9), (16, 49), (16, 66), (16, 67), (16, 69),
(17, 12), (17, 27), (17, 40), (17, 44), (17, 53), (17, 67), (18, 2), (18, 6), (18, 11), (18, 47), (18, 50), (18, 72),
(19, 24), (19, 27), (19, 28), (19, 33), (19, 38), (19, 55), (20, 29), (20, 39), (20, 42), (20, 51), (20, 61), (20, 63),
(21, 11), (21, 18), (21, 19), (21, 32), (21, 57), (21, 59), (22, 23), (22, 29), (22, 42), (22, 51), (22, 56), (22, 58),
(23, 5), (23, 8), (23, 17), (23, 39), (23, 64), (23, 70), (24, 1), (24, 31), (24, 41), (24, 55), (24, 60),
(25, 3), (25, 41), (25, 48), (25, 61), (25, 64), (25, 66), (26, 4), (26, 5), (26, 35), (26, 45), (26, 52), (26, 65),
(27, 22), (27, 24), (27, 32), (27, 36), (27, 48), (27, 63), (28, 21), (28, 22), (28, 26), (28, 27), (28, 52), (28, 64),
(29, 3), (29, 7), (29, 16), (29, 36), (29, 62), (30, 33), (30, 43), (30, 56), (30, 66), (30, 69), (30, 72),
(31, 6), (31, 8), (31, 15), (31, 37), (31, 54), (31, 67), (32, 7), (32, 30), (32, 43), (32, 55), (32, 60), (32, 68),
(33, 1), (33, 35), (33, 50), (33, 54), (33, 69), (33, 71), (34, 2), (34, 4), (34, 37), (34, 44), (34, 50), (34, 70),
(35, 7), (35, 12), (35, 21), (35, 26), (35, 28), (35, 71), (36, 26), (36, 30), (36, 44), (36, 45), (36, 54), (36, 72),
\}

### Figure 4-13 Comparison with LDPC(2,4,16)

K=2x2; N=4x4; g₁=[0 1; 1 1]; g₂=[1 1; 1 0];

- 2D Code - Viterbi
- 2D Code - Union Bound
- LDPC (dv=2, dc=4, k=16)
- Uncoded

### Figure 4-14 Comparison with LDPC(2,4,36) and LDPC(3,6,36)

K=2x2; N=6x6; g₁=[1 1; 1 0]; g₂=[1 1; 1 1];

- 2D Code - Viterbi
- 2D Code - Union Bound
- LDPC (dv=2, dc=4, k=36)
- LDPC (dv=3, dc=6, k=36)
- Uncoded
4.4.3 Compared to BCH codes

Primitive BCH codes [39], [40] are among the best known binary codes for short block lengths, and are known to be capable of achieving large minimum distances for a given block size. A primitive BCH code with a code word length \( n=2^m-1 \), capable of correcting \( t \) binary errors, has \( d_{\text{min}} \geq 2t + 1 \), and encodes \( k \geq n - mt \) information bits.

BCH codes are commonly decoded using the hard decision Berlekamp-Massey algorithm [41],[42], and a maximum-likelihood soft decision decoder of BCH codes has been proposed by [43]. We have not simulated these decoders, and will use the Union bound to estimate optimum BCH performance.

Determining the weight distribution of BCH codes is also a hard problem, although bounds on it have been found [44], [45]. For convenience, we will use tables of weight distributions found in [46]. The first few elements of the weight distributions of the codes used for comparison are given in Table 4-5.

For comparison with rate \( \frac{1}{2} \) codes of information support 4×4, the closest BCH code is BCH(31,16), capable of correcting \( t=3 \) errors, and with \( d_{\text{min}}=7 \). Comparing union bounds asymptotes, this BCH code is 0.63dB better than the best \( K=2\times2 \), \( N=4\times4 \) 2-D TBCC found. This is shown in Figure 4-16.

For comparison with rate \( \frac{1}{2} \) codes of information support 6×6, we have chosen BCH codes of length 63. There are two codes which are of comparable rate:

a. BCH(63,30), with rate \( R=0.47 \), and \( d_{\text{min}}=13 \).
b. BCH(63,36), with rate R=0.57, and $d_{\text{min}}=11$. These codes are compared with the best 2-D TBCC found, with $K=3\times3$, and $d_{\text{min}}=12$. Here, the 2-D TBCC performs slightly better than the BCH codes, having a 0.15dB advantage over the BCH(63,30), and a 0.36dB advantage over the BCH (63,36).

Thus, we can conclude that our 2-D TBCC are comparable in performance to BCH codes of similar size.

### Table 4-5 Reference BCH Codes for comparison

<table>
<thead>
<tr>
<th>BCH</th>
<th>$d_{\text{min}}$</th>
<th>$A(d_{\text{min}}+i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$i=0$</td>
<td>$i=1$</td>
</tr>
<tr>
<td>(31,16)</td>
<td>7</td>
<td>155</td>
</tr>
<tr>
<td>(63,36)</td>
<td>11</td>
<td>5,670</td>
</tr>
<tr>
<td>(63,30)</td>
<td>13</td>
<td>1,764</td>
</tr>
</tbody>
</table>

Figure 4-16 Comparison with BCH(31,16)
As demonstrated above, we have chosen to present performance results of codes mainly in terms of WER rather than BER. This is because of the following reasons: Many decoding algorithms try to decode the received input to the most likely codeword, and therefore WER is the natural performance metric for them.

Due to (a) and because of the relatively short word length, BER might be high even for a relatively low WER, as each erroneous word contains a large proportion of erroneous bits.

Since the SPLB is formulated in terms of WER, it is easier to understand how far away the tested codes are from the SPLB in these terms.

In this section we give a sample of how BER behaves for different block sizes for the same generating kernels. Let us consider the code with the kernels:

$$g_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \quad g_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

In Error! Reference source not found. we summarize the weight distribution of this code for various information block sizes.

It can be seen that the code achieves its maximal minimum distance for information support $N=7\times7$. This is because for smaller block sizes the minimum distance is determined by tail-biting error events.
In Figure 4-18 and Figure 4-19 we plot the union-bound asymptotic BER and WER, respectively, for this code in these block sizes. It can be seen that the differences in BER between smallest and largest block sizes are larger than the corresponding difference in BER (2.2 [dB] at BER=10^{-5}, opposed to 1.0 [dB] at WER=10^{-3}).

Further, in Figure 4-20, we plot the asymptotic BER vs. the asymptotic WER of the code for each block size. It can be seen that as the block size increases, the BER decreases for a fixed WER. Therefore, for short codes, we prefer to use WER as a measurement of performance. However, we expect the BER to improve as we increase the block size.

In Figure 4-21 we compare the BER performance of a the 1-D TBCC with generators (13,17) which is listed in [4] as the best 1-D CC with K=4. This code has $d_{\text{min}}=6$ for all block sizes tested. This is smaller than the $d_{\text{min}}$ achieved by the 2-D TBCC for block sizes larger than 6×6. Furthermore, multiplicity of codewords of weight w, and the average number of information bits per codeword weight, $B(w)$ is smaller for the 2-D TBCC (compare with Table 4-4). These facts contribute to the fact that the 2-D TBCC out-performs the 1-D TBCC at block sizes 6×6 and up.

Thus, 2-D TBCCs have the potential to out-perform 1-D TBCCs with comparable constraint length over a given information word length, except for very small blocks.

<table>
<thead>
<tr>
<th>$N_1 \times N_2$</th>
<th>$d_{\text{min}}$</th>
<th>$A(d_{\text{min}}+i)$, $B(d_{\text{min}}+i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i=0</td>
<td>i=1</td>
</tr>
<tr>
<td>4×4</td>
<td>4 8 , 4.000</td>
<td>0 , 0.000</td>
</tr>
<tr>
<td>5×5</td>
<td>5 10 , 5.000</td>
<td>0 , 0.000</td>
</tr>
<tr>
<td>6×6</td>
<td>6 12 , 6.000</td>
<td>36 , 1.000</td>
</tr>
<tr>
<td>7×7</td>
<td>7 63 , 2.333</td>
<td>98 , 2.000</td>
</tr>
<tr>
<td>8×8</td>
<td>7 64 , 1.000</td>
<td>144 , 2.667</td>
</tr>
<tr>
<td>10×10</td>
<td>7 100 , 1.000</td>
<td>200 , 2.000</td>
</tr>
</tbody>
</table>

* Union bound asymptotes based on first 8 elements
Figure 4-18 BER vs. $E_b/N_0$ of a $K=2\times2$ code for various $N$

Figure 4-19 WER vs. $E_b/N_0$ of a $K=2\times2$ code for various $N$
Figure 4-20 BER vs. WER of a $K=2 \times 2$ code for various $N$

Figure 4-21 BER Comparison of 2-D TBCC with 1-D TBCC
5 Survey of Belief Propagation Algorithms

In this section we survey belief propagation algorithms that are used in soft decoding of block and convolutional codes. This section is based on [30], [31], [33], and [36].

5.1 Loopy Belief Propagation – LBP

The first belief propagation we present is the generic message passing algorithm as presented in [33]. This algorithm was shown to generalized many message passing algorithms, such as the Viterbi [6], BCJR [7], and LDPC [29] decoding algorithms.

The idea behind this algorithm is to use messages passed over a factor graph to compute the marginals of a global function $g(x_1, ..., x_n)$ that factors into a product of several local functions, each having some subset of $(x_1, ..., x_n)$ as arguments;

$$g(x_1, ..., x_n) = \prod_{j \in J} f_j(x_j),$$  \hspace{1cm} (5.1.1)

where $J$ is a discrete index set, is a subset of $(x_1, ..., x_n)$, and is a function having the elements of $x_j$ as arguments.

**Definition:** A factor graph is a bipartite graph that expresses the structure of the factorization (1). A factor graph has a variable node for each variable $x_i$, a factor node for each local function $f_j$, and an edge-connecting variable node to factor node if and only if is an argument of $f_j$. A factor graph is thus a standard bipartite graphical representation of a mathematical relation—in this case, the “is an argument of” relation between variables and local functions.

**Example** [33]: (A Simple Factor Graph): Let $g(x_1, x_2, x_3, x_4, x_5)$ be a function of five variables, and suppose that $g(x)$ can be expressed as a product,

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1)f_B(x_2)f_C(x_1, x_2, x_3)f_D(x_3, x_4)f_E(x_3, x_5),$$  \hspace{1cm} (5.1.2)

of five factors, $J={A,B,C,D,E}$ so that $x_A = \{x_1\}$, $x_B = \{x_2\}$, $x_C = \{x_1, x_2, x_3\}$, $x_D = \{x_3, x_4\}$, and $x_E = \{x_3, x_5\}$. The factor graph that corresponds to (5.1.2) is shown in Figure 5-1.

![Figure 5-1 Simple Factor Graph Example](image)

A marginal function of $g(x_1, ..., x_n)$ with respect to a variable $x_i$ is derived by summing $g(x_1, ..., x_n)$ over all variables except $x_i$, ...
\[ g_i(x_i) = \sum_{-\{x_i\}} g(x_1, \ldots, x_n), \]

(5.1.3)

To continue our example, the marginal function \( g_1(x_1) \) is given by:

\[
g_1(x_1) = \sum_{-\{x_1\}} f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3, x_5) = \ldots
\]

\[
\ldots = f_A(x_1) \cdot \sum_{-\{x_1\}} f_B(x_2) f_C(x_1, x_2, x_3) \cdot \sum_{-\{x_1\}} f_D(x_3, x_4) \cdot \sum_{-\{x_1\}} f_E(x_3, x_5),
\]

(5.1.4)

This can be visualized by taking the graph from Figure 5-1 and re-drawing it as a tree with the variable node \( x_1 \) at the root. This is shown in Figure 5-2.

If the factor graph is cycle-free, then it may be re-drawn as a tree factor graph, with any variable \( x_i \) at its root. Such a tree graph is termed an expression tree. Every expression tree represents an algorithm for computing the corresponding expression. One might describe the algorithm as a “bottom-up” procedure that begins at the leaves of the tree, with each operator node combining its operands and passing on the result as an operand for its parent. One can imagine that there is a processor associated with each node of the factor graph, and that the factor-graph edges represent channels for passing messages between processors.

Computation of \( g_i(x_i) \) for all simultaneously can be efficiently accomplished by essentially “overlaying” on a single factor graph all the expression trees for single marginal functions. No particular node is taken as a root node, so there is no fixed parent/child relationship among
neighboring vertices. Instead, each neighbor $w$ of any given node $v$ is at some point regarded as a parent of $v$.

This leads to the iterative Sum-Product Algorithm, which consists of the following messages:

(i) Variable to Function:

$$n^{(l)}_{i \rightarrow a}(x_i) = \alpha \prod_{a' \in N(i) \setminus a} m^{(l-1)}_{a' \rightarrow i}(x_i),$$  \hspace{1cm} (5.1.5)

(ii) Function to Variable:

$$m^{(l)}_{a \rightarrow i}(x_i) = \alpha \sum_{j \in N(a) \setminus i} f(x_i, u_{j \ldots}) \prod_{j} n^{(l)}_{j \rightarrow a}(u_j),$$  \hspace{1cm} (5.1.6)

Where $l$ is the iteration index.

If we take the logarithm of the above equations, and use the approximation,

$$\log \left( \sum_i x_i \right) \approx \max_i \log x_i,$$  \hspace{1cm} (5.1.7)

We arrive at the so-called Minimum-Sum algorithm:

(i) Variable to Function:

$$\nu^{(l)}_{i \rightarrow a}(x_i) = \sum_{a' \in N(i) \setminus a} \mu^{(l-1)}_{a' \rightarrow i}(x_i) + \text{const},$$  \hspace{1cm} (5.1.8)

(ii) Function to Variable:

$$\mu^{(l)}_{a \rightarrow i}(x_i) \approx \max_{j \in N(a) \setminus i} \left\{ \log f(x_i, u_{j \ldots}) + \nu^{(l)}_{j \rightarrow a}(u_j) + \text{const} \right\},$$  \hspace{1cm} (5.1.9)

Using belief propagation in the context of decoding error-correcting codes is based on the idea that we’re interested in computing the conditional probabilities for transmitted bits $p(u_i = j | r)$ where $u$ is the information word before coding, and $r$ is the received codeword corrupted by noise. Assuming bits are independent of each other, this translates to computing the marginal probability:

$$p(u_i = j | r) = \sum_{-\{u_i\}} p(u | r) = \sum_{-\{u_i\}} \prod_{j \in \{u_i\}} p(u_i \mid r),$$  \hspace{1cm} (5.1.10)

When the graph is cycle-free, or a tree, then the computation of the marginal functions exact and the algorithm solves the MAP/ML problem accurately. However, when the graph contains cycles, the algorithm does not generally converge to the MAP/ML solution. Nonetheless, this algorithm is commonly used over graphs with cycles as an approximation of the MAP/ML solution, usually taking one of the following assumptions:

a. The graph does not contain cycles that are “short”, and/or
b. The total number of cycles in the graph is small.

When the graph contains cycles this algorithm is known as Loopy Belief Propagation.

We will see how the algorithm described above generalizes the Viterbi and BCJR algorithm in the following chapter.

5.2 Generalized Belief Propagation – GBP

The problem posed by cycles in the graph to the LBP algorithm is that messages incoming into a node contain information that was previously sent from the same node in outgoing messages. In other words, it is not possible to separate the innovation contained in a message from information already known in the node.

An approach to tackle this problem is suggested by [36], with justifications from the field of statistical mechanics. This approach is known as Generalized Belief Propagation (GBP).

GBP algorithms start by taking a factor graph as described in the previous section and grouping individual nodes into regions. Messages are then passed between regions in a similar manner to the way they were passed between nodes in the LBP algorithm. However, in the GBP algorithms we make sure that only information not contained in the region itself is passed into the region, thus avoiding the problem incurred by loops.

GBP was previously used by Shantal & Weiss [21] to equalize 2-D ISI channels, and is also used in computer vision and artificial intelligence [22] to solve 2-D problems. This makes it a probable candidate for decoding 2-D codes.

We hereby give a short summary of [36] which covers the theory of the GBP algorithm.

5.2.1 Free Energies

The basic premise of the GBP algorithm is the equivalence between the calculation of marginal probabilities of nodes in the graph, and the minimization of an energy functional over the graph, called the Bethe Free Energy.

We start the parallelism by associating each variable node in the graph with a particle in a physical system. In a system with \( N \) particles (variable nodes), each node can be in one of a discrete number of states. We label the states of the \( i \)-th particle by \( x_i \). The overall state of the system is denoted by the vector \( x = \{ x_1, x_2, ..., x_N \} \). Each state of the system has a corresponding energy \( E(x) \). In statistical mechanics, the probability of the system to be in a state \( x \) is dictated by Boltzmann’s Law:

\[
p(x) = \frac{1}{Z} e^{-E(x)/T} \tag{5.2.1}
\]

Where \( T \) is the system temperature and \( Z \) is the partition function which serves as normalization constant:

\[
Z = \sum_{x \in S} e^{-E(x)/T} \tag{5.2.2}
\]

where the summation is over the space of all possible states \( S \).
From now on, we will assume that $T=1$, unless stated otherwise.

On the other hand, the particles (variable nodes) are tied to each other by $M$ factor nodes which impose constraints on the system. These constraints dictate that:

$$p(x) = \frac{1}{Z} \prod_{a=1}^{M} f_a(x_a) \quad (5.2.3)$$

Where $x_a$ is the set of variable nodes that are connected to the factor node $a$. To impose consistency with Boltzmann’s Law, we set the energy of state $x$ to be:

$$E(x) = -\sum_{a=1}^{M} \ln f_a(x_a) \quad (5.2.4)$$

Taking the logarithm of the probability $p(x)$, we get:

$$\ln p(x) = -\ln(Z) - E(x) \quad (5.2.5)$$

The quantity $-\ln(Z)$ is known as the *Helmholtz Free Energy*, $F_{\text{Helmholtz}} = -\ln(Z)$. The computation of this quantity is usually complex. One approach is to start with a trial probability distribution $b(x)$ and define a *variational free energy* by:

$$F(b) = F_{\text{Helmholtz}} + D(b \parallel p) \quad (5.2.6)$$

Where $D(b\|p)$ is the *Kullback-Leibler distance* between $b(x)$ and $p(x)$. Thus, the minimization of $F(b)$ yields $F_{\text{Helmholtz}}$. By definition,

$$D(b \parallel p) = \sum_{x \in S} b(x) \ln \frac{b(x)}{p(x)} \quad (5.2.7)$$

This allows us to represent $F(b)$ as,

$$F(b) = U(b) - H(b) \quad (5.2.8)$$

Where $H(b)$ is the *variational entropy*,

$$H(b) = -\sum_{x \in S} b(x) \ln b(x) \quad (5.2.9)$$

and $U(b)$ is the *variational average energy*:

$$U(b) = \sum_{x \in S} b(x)E(x) \quad (5.2.10)$$

This construction allows us to formulate the minimization of $F(b)$ in terms of belief propagation algorithms.
For example, a first-order approximation is the mean field approximation:

\[ b_{MF}(x) = \prod_{i=1}^{N} b_i(x_i) \]  

(5.2.11)

Substituting this in the above equations, we get:

\[ U_{MF}(b) = -\sum_{a=1}^{M} \sum_{x_a} \ln f_a(x_a) \prod_{i \in N(a)} b_i(x_i) \] 

\[ H_{MF}(b) = -\sum_{i=1}^{N} \sum_{x_i} b_i(x_i) \ln b_i(x_i) \]  

(5.2.12)

However, this approximation is usually too simplistic. A more advanced approach is to assume that some nodes are statistically dependent, but that the graph is composed of groups of nodes, or regions that are statistically independent.

In this approach, groups of nodes in the factor graph are lumped together to form regions. A region \( R \) of a factor graph is a set \( V_R \) of variable nodes and a \( F_R \) of factor nodes. The set \( F_R \) of factor may be empty. However, if \( F_R \) contains a factor node \( a \), then \( V_R \) must contain all the variable nodes that are connected to \( a \). The state \( x_R \) of the region is the collective set of its variable node states \( \{x_i | i \in V_R \} \). The marginal probability function of a region \( p_R(x_R) \) is obtained by marginalizing the global probability function \( p(x) \) over the set \( V_R \). We assign each region a belief \( b_R(x_R) \) which approximates the region’s probability.

The region energy is defined as:

\[ E_R(x_R) = -\sum_{a \in F_R} \ln f_a(x_a) \]  

(5.2.13)

Where the summation is over all the factor nodes in the region, and \( x_a \) denotes the state of all variable nodes connected to factor node \( a \).

Similarly, we define the region average energy \( U_R(b_R) \), the region entropy \( H_R(b_R) \) and the region free energy \( F_R(b_R) \) as,

\[ U_R(b_R) = \sum_{x_R} b_R(x_R) E_R(x_R) \]  

(5.2.14)

\[ H_R(b_R) = -\sum_{x_R} b_R(x_R) \ln b_R(x_R) \]  

(5.2.15)

\[ F_R(b_R) = U_R(b_R) - H_R(b_R) \]  

(5.2.16)

To calculate the total free energy, we would like to sum the free energy of all regions. However, since regions may overlap, we may end up summing the free energy contributed by some nodes
more than once. To avoid this, we multiply the free energy each region a counting number $c_R$ that ensures that each node is counted only once in the summation. Thus we have that the region based approximation for the total free energy is,

$$F = \sum_R c_R F_R(b_R)$$  \hfill (5.2.17)

And the approximations for the total average energy and entropy,

$$U = \sum_R c_R U_R(b_R) = \sum_R c_R \sum_{s_x} b_R(x_R) E_R(x_R) = -\sum_R c_R \sum_{s_x} b_R(x_R) \ln f_a(x_a)$$  \hfill (5.2.18)

$$H = \sum_R c_R H_R(b_R) = -\sum_R c_R \sum_{s_x} b_R(x_R) \ln b_R(x_R)$$  \hfill (5.2.19)

In the following sections we consider such region based approximations and the belief propagation algorithms that stem from them.

### 5.2.2 Bethe Approximation

The Bethe approximation uses two sets of regions, a set of “small regions” $R_S$, and a set of “large regions” $R_L$. The set $R_L$ contains $M$ regions. Each region in $R_L$ contains exactly one factor node and all the variable nodes neighboring that factor node. We note that no two regions in $R_L$ contain the same factor node.

Next, the set of small regions $R_S$, contains $N$ regions, each of which contains a single variable node. No two regions in $R_S$ contain the same variable node.

Lastly, we define that if $R_1$ and $R_2$ are two regions, we say that $R_1$ is a sub-region of $R_2$ and $R_2$ is a super-region of $R_1$ if the set of variable and factor nodes in $R_1$ are a subset of those in $R_2$.

The counting numbers $c_R$ for each region $R \in R$ are given by,

$$c_R = 1 - \sum_{S \in S(R)} c_S$$  \hfill (5.2.20)

where $S(R)$ is the set of regions that are super-regions of $R$.

The complete set of regions $R_{\text{Bethe}}$ included in the Bethe approximation is $R_{\text{Bethe}} = R_L \cup R_S$. Using our definition we see that for every region $R \in R_L$, $c_R = 1$, while for every region $R \in R_S$, $c_R = 1 - d_i$, where $d_i$ is the degree (number of neighboring factor nodes) of the variable node $i$. It is easy to confirm that the Bethe approximation will always be a valid approximation, as each factor and variable node will clearly be counted once. We can use our expressions for the counting numbers $c_R$ to obtain the Bethe approximation to the free energy, entropy, and average energy.
Substituting Eq. (5.2.20) in Eqs. (5.2.18) and (5.2.19) we get,

\[ U_{\text{Bethe}}(b) = -\sum_{R} c_{R} \sum_{x_{R}} b_{R}(x_{R}) \sum_{a \in F_{R}} \ln f_{a}(x_{a}) = -\sum_{a=1}^{M} \sum_{x_{a}} b_{a}(x_{a}) \ln f_{a}(x_{a}) \]  

(5.2.21)

Since only large regions contain factor nodes, and their counting number is 1

And for the entropy:

\[ H_{\text{Bethe}}(b) = -\sum_{R} c_{R} \sum_{x_{R}} b_{R}(x_{R}) \ln b_{R}(x_{R}) = \]

\[ = -\sum_{a=1}^{M} \sum_{x_{a}} b_{a}(x_{a}) \ln b_{a}(x_{a}) - \sum_{i=1}^{N} (1-d_{i}) \sum_{x_{i}} b_{i}(x_{i}) \ln b_{i}(x_{i}) \]

(5.2.22)

Where the first term is the contribution of the large regions, and the second is the contribution of the small regions.

The beliefs of each region must satisfy the \textit{normalization constraint}, i.e. they must sum up to 1, since they represent probabilities:

\[ \sum_{x_{a}} b_{a}(x_{a}) = 1 \quad ; \quad \sum_{x_{i}} b_{i}(x_{i}) = 1 \]

(5.2.23)

Another constraint which must be satisfied is the \textit{marginalization constraint}, which ties the beliefs of single variable nodes with the beliefs of the large regions which contain them. Specifically, the marginalization of the beliefs of a large region, \( b_{a}(x_{a}) \), with respect to a variable node \( x_{i} \) must yield the beliefs of this variable nodes \( b_{i}(x_{i}) \):

\[ \sum_{x_{a} \setminus x_{i}} b_{a}(x_{a}) = b_{i}(x_{i}) \]

(5.2.24)

It can be shown that minimizing the Bethe free energy approximation yields the standard BP algorithm from section 5.1, and a full proof of this is given in Appendix A.

The equations which describe this algorithm are:

Messages from small regions to large regions:

\[ n_{j \rightarrow a}(x_{i}) = \prod_{a \in N(i) \setminus a} m_{a \rightarrow j}(x_{i}) \]

(5.2.25)

Messages from large regions to small regions:

\[ m_{a \rightarrow j}(x_{i}) \propto \sum_{x_{a} \setminus x_{i}} f_{a}(x_{a}) \prod_{j \in N(a) \setminus j} n_{j \rightarrow a}(x_{i}) \]

(5.2.26)
Beliefs of small regions (single variable nodes):

\[ b_i(x_i) \propto \prod_{a \in N(i)} m_{a \rightarrow i}(x_i) \quad (5.2.27) \]

### 5.2.3 Kikuchi Approximation:

Another method for selecting a valid set of regions \( R \) and counting numbers \( c_R \) is the cluster variation method introduced by Kikuchi in 1951 and further developed in the physics literature since then [38].

In the cluster variation method, we begin with a set of distinct large regions \( R_0 \) such that every factor node \( a \) and every variable node \( i \) in our factor graph is included in at least one region \( R \subset R_0 \). We also require that no region \( R \subset R_0 \) be a sub-region of any other region in \( R_0 \). We then construct the set of regions \( R_1 \) by forming all possible intersections between regions in \( R_0 \), but discarding from \( R_1 \) any intersection regions that are sub-regions of other intersection regions. If possible, we then construct in the same way the set of regions \( R_2 \) from the intersections between regions in \( R_1 \). As long as there continue to be intersection regions, we construct sets of regions \( R_3, R_4, \ldots, R_K \) in the same way. Finally, the set of regions used in the cluster variation method will be \( R = R_0 \cup R_1 \cup \ldots \cup R_K \).

To form a region graph using the cluster variation method, we draw connections between the regions obtained in \( R \) in the following way. A region \( P \) is defined to be the parent of a region \( R \), iff:

1. \( R \) is contained in \( P \)
2. There is no region \( S \) such that \( S \) contains \( R \) and \( P \) contains \( S \)

Similarly, if these conditions are satisfied, we say that \( R \) is a child region of \( P \).

To create the graph we draw edges between each region and all its parent regions. We define the counting numbers in the cluster variation method to be:

\[ c_R = 1 - \sum_{S \in S(R)} c_S \quad (5.2.28) \]

Where \( S(R) \) is the set of all regions that contain the region \( R \) (not only direct parents).

Having defined the region graph, we are now ready to describe the BP algorithm that operates over it. This algorithm is named the “Two-Way Algorithm” in [36], since in this algorithm messages are passed both from parent regions to their children, and from children to their parents.

To motivate the two-way algorithm, we recall that in the standard BP algorithm, the belief equations can be written in the form,

\[ b_i(x_i) = \prod_{a \in N(i)} m_{a \rightarrow i}(x_i) \quad (5.2.29) \]
Given these equations, it is natural to aim for a generalization where the belief equations will have the form,

\[
b_{a}(x_{a}) = f_{a}(x_{a}) \prod_{i \in N(a)} n_{i \rightarrow a}(x_{i}) \tag{5.2.30}
\]

\[
n_{i \rightarrow a}(x_{i}) = \prod_{b \in N(i) \cap a} m_{b 
arrow i}(x_{i}) \tag{5.2.31}
\]

In other words, we aim to write the belief equations so that the belief in a region is a product of local factors and messages arriving from all the connected regions, whether they are parents or children. It will turn out that we can do this, but in order that the GBP algorithm correspond to the region graph free energy, we will need to use modified factors and a rather complicated relation between the \(n_{C \rightarrow R}(x_{C})\) messages and \(m_{P \rightarrow C}(x_{P})\) messages generalizing the relation for standard BP given in equation (5.2.31).

It will be convenient to denote the number of parents of region \(R\) by \(p_{R}\), and define the numbers \(q_{R} := (1 - c_{R})/p_{R}\) and \(\beta_{R} := 1/(2 - q_{R})\). When a region has no parent so that \(p_{R} = 0\) and \(c_{R} = 1\), we take \(q_{R} = \beta_{R} = 1\). Note that within the Bethe approximation, \(q_{R} = \beta_{R} = 1\) for all regions. We will assume that \(q_{R} \neq 2\) so that \(\beta_{R}\) is well-defined (normally, if one has a region graph with a region such that \(q_{R} = 2\), one should be able to change the connectivity of \(R\) to avoid this problem).

We first define the set of pseudo-messages for all regions \(R\) and their parents \(P\) and children \(C\):

\[
n_{R \rightarrow P}(x_{R}) = \tilde{f}_{R}(x_{R}) \prod_{P \in \mathcal{P}(R)} m_{P \rightarrow R}(x_{R}) \prod_{C \in \mathcal{C}(R)} n_{C \rightarrow R}(x_{C}) = \frac{b_{R}(x_{R})}{m_{P \rightarrow R}(x_{R})} \tag{5.2.33}
\]

\[
m_{R \rightarrow C}(x_{R}) = \sum_{x_{C} \in \mathcal{X}_{C}} \tilde{f}_{R}(x_{R}) \prod_{P \in \mathcal{P}(R)} m_{P \rightarrow R}(x_{R}) \prod_{C \in \mathcal{C}(R)} n_{C \rightarrow R}(x_{C}) = \frac{b_{R}(x_{R})}{n_{C \rightarrow R}(x_{C})} \tag{5.2.34}
\]

Where

\[
\tilde{f}_{R}(x_{R}) = \left( \prod_{a \in A_{R}} f_{a}(x_{a}) \right)^{c_{R}} \tag{5.2.35}
\]

Aside from the fact that we raised the product of the local factors to a power of \(c_{R}\), these pseudo-messages are what one would naively expect the message updates to look like. To obtain the true message updates, however, one needs to combine the pseudo-messages going in the two directions of a link as follows:

\[
n_{R \rightarrow P}(x_{R}) = \left( n_{R \rightarrow P}(x_{R}) \right)^{\phi_{a}} \left( m_{P \rightarrow R}(x_{R}) \right)^{\phi_{a}^{-1}} \tag{5.2.36}
\]
\[ m_{R \rightarrow C}(x_k) = \left( n^\beta_{C \rightarrow R}(x_C) \right)^{\beta - 1} \left( m^\beta_{R \rightarrow C}(x_C) \right)^{\beta_c} \]  \hspace{1cm} (5.2.37)

Note that when $\beta_R = 1$, the messages are precisely the same as the pseudo-messages. The two-way algorithm is completed by the belief equations, which have the form already given in equation (5.2.32).

Similarly to the case of the Bethe approximation, it can be shown that minimizing the Kikuchi free energy yields the above message passing algorithm. A complete proof is given by [34], and is also repeated in Appendix A.
6 Decoding in the Information Domain

In this chapter we describe BP algorithms for soft decoding of 2-D CC that try to directly estimate the information word, without first estimating the coded word. We describe the optimal MAP and ML decoding algorithms for 2-D CC, and show that their complexity is exponential with the dimensions of the input. We then formulate a heuristic BP algorithm, and discuss its performance and convergence.

6.1 Optimal Decoding

The problem of optimal decoding of a 2DCC using a Maximum-A-Posteriori Probability (MAP) or Maximum Likelihood (ML) criterion can be reduced to the problem of decoding a 1DCC over a non-binary field, GF(2^N), as shown by [20].

6.1.1 Reduction to a 1-D Problem

The ML criterion aims to find the information word \( u \) (or codeword \( v \)) transmitted over the noisy channel, that is most likely to produce the noisy observations \( r \) at the receiver:

\[
\hat{u}^{ML} = \arg \max_u \Pr(r \mid u)
\]  

(6.1.1)

The problem posed by this condition for 1DCC is solved by the well known Viterbi algorithm [6].

On the other hand, the MAP criterion seeks to find the most likely information bits \( u_i \), \( i=1,2,\ldots,K \) transmitted over the channel given the noisy observations \( r \):

\[
\hat{u}_i^{MAP} = \arg \max_{u_i} \Pr(u_i \mid r)
\]

(6.1.2)

This problem is solved for the 1-D case by the BCJR algorithm [7].

Optimal detection (MAP/ML) of a 1-D CC memory depth \( \nu \) can be achieved with a trellis with \( 2^\nu \) states, irrespective of the input data block length. Both the BCJR and Viterbi algorithms rely on the fact that 1-D CC is a causal Markov process, where the conditioning on a single trellis stage effectively separates past and future events.

In the 2-D case, there is no immediate notion of causality, and we must condition on a bounding strip to create two conditionally independent regions. The bounding region must completely isolate the two regions to avoid unaccounted-for interaction via some path. Two examples are depicted in Figure 6-1, where \( P(A,C \mid B) = P(A \mid B)P(C \mid B) \). On the left hand we depict an arbitrary division into conditionally independent regions. A more convenient division is shown on the right hand, which divides the 2-D grid along columns (of course, an equivalent division can be done along rows). This division is especially convenient when dealing with 2-D CCs.
Let us consider a 2DCC with a generator matrix \( G(x, y) \in R^{k \times n} \), that operates on composite sequences with finite support \( N_1 \times N_2, u \in S_{N_1 \times N_2}^k \). We say that \( G(x, y) \) has a constraint lengths \( K_1, K_2 \) along each dimension if:
\[
K_1 = \max_{i,j} \deg_{x,y} g_{i,j} (x, y), \quad K_2 = \max_{i,j} \deg_{x,y} g_{i,j} (x, y) \tag{6.1.3}
\]

Then, we can reduce the 2-D CC decoding problem to a 1-D CC decoding problem, by taking each column (row) of the input sequence to be a single input to an equivalent 1-D CC encoder, with memory consisting of the \( K_2-1 \) previous columns (\( K_1-1 \) previous rows). Thus, a single input to the encoder consists of \( 2^{kN_1} (2^{kN_2}) \) bits, and the memory of the encoder consists of \( 2^{k(K_2-1)N_1} (2^{k(K_1-1)N_2}) \) bits. The overall complexity of processing a trellis stage, given by number of states multiplied by the number of branches, using this division, is therefore \( 2^{kK_2N_1} (2^{kK_1N_2}) \). We can choose to use the division by columns or by rows that minimizes the trellis complexity. Thus, the complexity is \( O(2^{k(K_2-1)N_1}) \).

No matter which division to regions we use, in order to decode an entire 2-D grid of data, the bounding region must at some point span the entire width or height of the grid, making the complexity of the optimal detector exponentially dependent on the grid size. In the limit as \( N_1, N_2 \to \infty \) the grid grows to a plane and the decoding complexity per bit approaches infinity. The term 'finite-complexity' shall be used to describe algorithms whose per bit complexity remains finite as the data set becomes infinite. In this sense, the optimal MAP/ML algorithm described above does not qualify as finite complexity.

Nonetheless, for codes where one dimension is very small (the second may be infinite) this approach is feasible, and we shall use it to get lower bounds on bit-error probability and word-error probability.

### 6.1.2 Relationship with Belief Propagation

The BCJR and Viterbi algorithms have been shown in [31, 33] to be equivalent to the Sum-Product and Minimum-Sum variants of the Belief Propagation algorithm, respectively. The conventional factor graph representation of a trellis of a tail-biting 1-D CC is shown in Figure 6-2.

In this representation we have the following variable nodes:

a. Encoder state variable nodes, marked by \( s_t \), that correspond to the encoder state at time \( t \).
b. Information bit nodes, marked by $u_t$, that correspond to the $k$ input bits pushed into the encoder at time $t$.

c. Code bit nodes the nodes marked by $v_t$, that correspond to the $n$ output bits produced by the encoder at time $t$

d. Channel observation nodes marked by $y_t$, that correspond to copies of the coded bits $v_t$ that were corrupted by the channel.

And the following function nodes:

a. Channel conditional probability nodes, that compute $p(v_t|y_t)$

b. Trellis compatibility functions $T(u_t, v_t, s_{t-1}, s_t)$, that represents the compatibility of the values of the nodes $u_t$, $v_t$, $s_{t-1}$, $s_t$

$$T(u_t, v_t, s_{t-1}, s_t) = \begin{cases} 
1 & \text{if } u_t, v_t, s_{t-1}, s_t \text{ are compatible} \\
0 & \text{otherwise}
\end{cases}$$

This above description can be simplified if we lump together the state and information nodes to form region nodes which we denote by $r_t$. Each $n$-tuple of code bits is now uniquely defined by the region node with which it is associated, and adjacent nodes are connected together via a compatibility function. A graph constructed by this representation is shown in Figure 6-3.

In this representation we have the following variable nodes:

a. Region variable nodes, that correspond to a sequence of $k \cdot v$ information bits, with an overlap of $k \cdot (v - 1)$ between adjacent regions.

b. Code bit nodes, $v_t$, as before.

c. Channel observation nodes, $y_t$, as before.

And the following function nodes:

a. Channel conditional probability nodes, as before

b. Intersection compatibility functions $I(r_{t-1}, r_t)$, that indicate whether the intersection of the values of the nodes $r_{t-1}$, $r_t$, is valid.

$$I(r_{t-1}, r_t) = \begin{cases} 
1 & \text{if } r_{t-1}, r_t \text{ form a valid intersection} \\
0 & \text{otherwise}
\end{cases}$$

- Figure 6-2 Classical Graph Representation of a 1-D Trellis
c. Code compatibility functions $C(r_i, v_i)$, that indicate whether the coded bits $v_i$ are formed by the bits in the region $r_i$.

The two representations are completely equivalent. However, the second representation will be more helpful for us in constructing a graphical representation for 2-D CCs.

For a 2-D CC which is decoded column-by-column, we have: $2^{k_2 \cdot N_1}$ information bits and $2^{n_2 \cdot N_1}$ coded bits per region, so that for a well-defined code, knowing the coded bits constrains the solution for the region’s information bits.
6.2 2D-Trellis Local Belief Propagation

In this section we would like to extend the graph shown in Figure 6-3 to 2-D codes. We find that we can create such a graph, but the performance and indeed the convergence of the BP algorithm over this graph depend on the properties of the code. The resulting algorithm is similar to the BCJR algorithm. However, where the in the BCJR algorithm messages are passed forward and backward (hence it is also known as the forward-backward algorithm), in the 2D case messages are passed left, right, up and down. More crucially, in the 1D case the forward-backward algorithm gives the MAP solution for the decoder problem, which is not true for the 2D algorithm, for reasons which shall be explained below. Therefore, we expect a performance loss relative to true ML decoding.

6.2.1 Creating a Factor Graph

Let us first recall the concept of a constraint region introduced in section 4.2.2. Let \( G(x,y) \) be a generator matrix of a rate \( k/n \) 2-D TBCC code, with kernel support \( K_1 \times K_2 \), i.e.,

\[
K_1 = \max_{i,j} \text{deg}_i g_{i,j}(x,y), \quad K_2 = \max_{i,j} \text{deg}_j g_{i,j}(x,y)
\]

Then the code has constraint region of dimensions \( K_1 \times K_2 \). Each constraint region uniquely determines the code bits at position \((i,j)\).

Let \( u \) be an input sequence with finite support \( N_1 \times N_2 \), i.e.,

\[
u(x,y) \in \overline{R}(x,y)
\]

where: \( I = \{x^{N_1-1}, y^{N_2-1}\} \), and \( \overline{R}(x,y) = R(x,y) / I \).

Then the sequence \( u \) can be mapped into \( N_1 \times N_2 \) overlapping constraint regions such that,

\[(i,j) = \{u(k,l)\}, i=0,...,N_1-1, j=0,...,N_2-1, k=(i,...,i+K_1) \mod N_1, l=(i,...,i+K_2) \mod N_2.
\]

**Example:** We illustrate the division of the information sequence to regions for \( N_1=N_2=4 \) (i.e., information sequences of support \( 4 \times 4 \)), and the following rate \( 1/2 \) code:

\[
G(x,y) = (1 + xy, 1 + x + y) \iff g_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} g_2 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}
\]

The support of the kernels is \( 2 \times 2 \), therefore we divide the information sequence to 16 overlapping regions of size \( 2 \times 2 \). Figure 6-4 shows some regions of this division. Due to the modulo operation, regions at the edge of the sequence fold around to the opposite edge. This is illustrated by the corners of the sequence which form the region \( r(4,4) \).
Given this division, each region \( r(i,j) \) uniquely determines the coded bits associated with it \( v(i,j) \):

\[
v_q(i,j) = \sum_{p=1}^{k} \sum_{k=0}^{v_1-1} \sum_{l=0}^{v_2-1} g_{p,q}(v_1 - k - 1, v_2 - l - 1) \mu(i + k \mod M, j + l \mod N),
\]

where \( q=1,...,n \).

We define the code compatibility function between a region and the coded bits associated with it as:

\[
f_c(r,v) = \begin{cases} 
1 & \text{if } v \text{ is associated with } r \\
0 & \text{otherwise} 
\end{cases}, \quad v \in [0,1]^n, \quad r \in [0,1]^{v_1 \times v_2}
\]

Each region \( r(i,j) \) has four immediate neighbors: \( r(i+1,j), r(i-1,j), r(i,j+1), r(i,j-1) \). We define the horizontal and vertical intersection compatibility functions between two neighboring regions, as:

\[
f_V(r_{i,j}, r_{i+1,j}) = \begin{cases} 
1 & \text{if the intersection of } r(i,j) \text{ and } r(i+1,j) \text{ is valid} \\
0 & \text{otherwise} 
\end{cases}
\]

\[
f_H(r_{i,j}, r_{i,j+1}) = \begin{cases} 
1 & \text{if the intersection of } r(i,j) \text{ and } r(i,j+1) \text{ is valid} \\
0 & \text{otherwise} 
\end{cases}
\]

A piece of the resulting graph is shown in Figure 6-5.
We could extend the graph to neighbors which are not immediate (i.e., corner intersections, neighbors that are removed by a shift larger than 1), but we refrain from doing so, as will not improve the performance of a BP algorithm. Adding more intersections to the graph simply repeats messages (or parts of messages) passed between immediate neighbors, and will create more loops in the graph. Therefore, we keep only the immediate neighbors in the BP graph. In the next section we describe the BP algorithm messages over this graph.

### 6.2.2 Formulation of the 2D-Trellis algorithm

Using the region graph construction from the previous section, we can formulate a message passing algorithm based on the BP equations. The resulting algorithm is analogous to the 1-D BCJR algorithm but with the following notable differences:

1. The algorithm has messages propagating in both dimensions for each region. Where the 1-D algorithms have forward and backward messages, the 2-D algorithm has left-right and up-down messages.
2. The algorithm is heuristic and does not guarantee convergence to the MAP/ML solutions. This is because:
   a. The graph has many loops, which violates the tree assumption of the BP algorithm.
   b. The regions are not conditionally independent of each other, as required by a true Viterbi / BCJR.
   c. The code function between a region and its code bits may be such that it does not contain information about the intersection between the region and any one of its neighbors. This leads to no meaningful message passing from the region to its neighbors, which in turn leads to a complete failure of the algorithm (for a demonstration of this see example II in section 6.2.3).
We now turn to describing the sum-product 2D-Trellis algorithm in detail. The min-sum version and is lower complexity suboptimal version and is obtained from this algorithm by the log-max approximation. Let us recall the basic update equations of the BP algorithm:

Variable to Function:  \[ \mu^{(l)}_{v \rightarrow f}(x) = \alpha \prod_{(b) \in f} \mu^{(l-1)}_{h_{b} \rightarrow v}(x) \]

Function to Variable:  \[ \mu^{(l)}_{f \rightarrow v}(x) = \alpha \sum_{(a) \in v} f(x,u_{1},...) \prod_{u} \mu^{(l)}_{u_{1} \rightarrow f}(u_{i}) \]

Where \( l \) is the iteration index.

We adopt the following notations for messages in the 2D-Trellis algorithm. Define the following indices:

- \( i \): row index
- \( j \): column index
- \( l \): iteration index

The following tables summarize the notation of messages passed between nodes:

**Messages from variables to functions:**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Long notation</th>
<th>Short notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_{i,j} )</td>
<td>( f_{i,j}^{C} )</td>
<td>( \mu^{(l)}<em>{c</em>{i,j} \rightarrow f_{i,j}^{C}}(v) )</td>
<td>( \gamma^{(l)}_{i,j}(v) )</td>
</tr>
<tr>
<td>( r_{i,j} )</td>
<td>( f_{i,j}^{H} )</td>
<td>( \mu^{(l)}<em>{r</em>{i,j} \rightarrow f_{i,j}^{H}}(r) )</td>
<td>( \alpha^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( r_{i,j} )</td>
<td>( f_{i,j-1}^{H} )</td>
<td>( \mu^{(l)}<em>{r</em>{i,j} \rightarrow f_{i,j-1}^{H}}(r) )</td>
<td>( \beta^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( r_{i,j} )</td>
<td>( f_{i,j}^{V} )</td>
<td>( \mu^{(l)}<em>{r</em>{i,j} \rightarrow f_{i,j}^{V}}(r) )</td>
<td>( \eta^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( r_{i,j} )</td>
<td>( f_{i,j+1}^{V} )</td>
<td>( \mu^{(l)}<em>{r</em>{i,j} \rightarrow f_{i,j+1}^{V}}(r) )</td>
<td>( \zeta^{(l)}_{i,j}(r) )</td>
</tr>
</tbody>
</table>

**Messages from functions to variables:**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Long notation</th>
<th>Short notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_{i,j}^{C} )</td>
<td>( r_{i,j} )</td>
<td>( \mu^{(l)}<em>{f</em>{i,j}^{C} \rightarrow r_{i,j}}(r) )</td>
<td>( \hat{\lambda}^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( f_{i,j}^{H} )</td>
<td>( r_{i,j} )</td>
<td>( \mu^{(l)}<em>{f</em>{i,j}^{H} \rightarrow r_{i,j}}(r) )</td>
<td>( \lambda^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( f_{i,j}^{H} )</td>
<td>( r_{i,j+1} )</td>
<td>( \mu^{(l)}<em>{f</em>{i,j}^{H} \rightarrow r_{i,j+1}}(r) )</td>
<td>( \beta^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( f_{i,j}^{V} )</td>
<td>( r_{i,j} )</td>
<td>( \mu^{(l)}<em>{f</em>{i,j}^{V} \rightarrow r_{i,j}}(r) )</td>
<td>( \hat{\lambda}^{(l)}_{i,j}(r) )</td>
</tr>
<tr>
<td>( f_{i,j}^{V} )</td>
<td>( r_{i+1,j} )</td>
<td>( \mu^{(l)}<em>{f</em>{i,j}^{V} \rightarrow r_{i+1,j}}(r) )</td>
<td>( \hat{\lambda}^{(l)}_{i,j}(r) )</td>
</tr>
</tbody>
</table>

We now construct the update rules according to the factor graph message passing equations:
I. Code node → function node → region node:

\[ \gamma_{i,j}(c) = \Pr(c \mid y_{i,j}) \]  

(6.2.5)

\[ \lambda_{i,j}(r) = \alpha \sum_c I(c(r) = c) \cdot \Pr(c \mid y_{i,j}) = \Pr(r \mid y_{i,j}) \]  

(6.2.6)

These messages are constant for all iterations, therefore equations (6.2.5) and (6.2.6) only need to be calculated at the first iteration.

II. Region node → intersection functions

Right:

\[ \alpha^{(l)}_{i,j}(r) = p(r \mid y_{i,j}) \cdot A^{(l-1)}_{i,j-1}(r) \cdot H^{(l-1)}_{i-1,j}(r) \cdot Z^{(l-1)}_{i,j}(r) \]  

(6.2.7)

Left:

\[ \beta^{(l)}_{i,j}(r) = p(r \mid y_{i,j}) \cdot B^{(l-1)}_{i,j}(r) \cdot H^{(l-1)}_{i-1,j}(r) \cdot Z^{(l-1)}_{i,j}(r) \]  

(6.2.8)

Down:

\[ \eta^{(l)}_{i,j}(r) = p(r \mid y_{i,j}) \cdot A^{(l-1)}_{i,j-1}(r) \cdot B^{(l-1)}_{i,j}(r) \cdot H^{(l-1)}_{i-1,j}(r) \]  

(6.2.9)

Up:

\[ \zeta^{(l)}_{i,j}(r) = p(r \mid y_{i,j}) \cdot A^{(l-1)}_{i,j-1}(r) \cdot B^{(l-1)}_{i,j}(r) \cdot Z^{(l-1)}_{i,j}(r) \]  

(6.2.10)

III. Intersection functions → Region nodes

Right:

\[ A^{(l)}_{i,j}(r) = \sum_{r'} I_k(r, r') \cdot \alpha^{(l)}_{i,j}(r') = \]  

(6.2.11)

\[ = \sum_{r'} I_k(r, r') \cdot p(r' \mid y_{i,j-1}) \cdot A^{(l-1)}_{i,j-1}(r') \cdot H^{(l-1)}_{i-1,j}(r') \cdot Z^{(l-1)}_{i,j}(r') \]

Left:

\[ B^{(l)}_{i,j}(r) = \sum_{r'} I_k(r, r') \cdot \beta^{(l)}_{i,j}(r') = \]  

(6.2.12)

\[ = \sum_{r'} I_k(r, r') \cdot p(r \mid y_{i,j+1}) \cdot B^{(l-1)}_{i,j+1}(r') \cdot H^{(l-1)}_{i-1,j+1}(r') \cdot Z^{(l-1)}_{i,j+1}(r') \]

Down:
\[ H_{i,j}^{(r)} = \sum_{r'} I_{i,j}^{(r)} \cdot \eta_{i,j}^{(r)} = \sum_{r'} I_{i,j}^{(r)} \cdot p(r \mid y_{i+1,j}) \cdot A_{i,j-1}^{(r')} \cdot B_{i,j}^{(r')} \cdot H_{i-1,j}^{(r')} \] (6.2.13)

Up:
\[ Z_{i,j}^{(r)} = \sum_{r'} I_{i,j}^{(r)} \cdot \zeta_{i,j}^{(r')} = \sum_{r'} I_{i,j}^{(r)} \cdot p(r \mid y_{i+1,j}) \cdot A_{i,j+1}^{(r')} \cdot B_{i,j}^{(r')} \cdot Z_{i+1,j}^{(r')} \] (6.2.14)

We can see from the above formulation, that the 2-D trellis algorithm has lower complexity in comparison to the 1-D Viterbi algorithm from section 6.1. The 2-D Trellis algorithm has a complexity that is linear with the input dimensions, but exponential with the support of the kernel. Thus the number of operations per bit is of order \( O(2^{K_1 K_2}) \). This is because the graph has \( N_1 N_2 \) region nodes, and each of them has to compute \( 2^{K_1 K_2} \) beliefs. The actual number of operations can be reduced by a sophisticated implementation.

### 6.2.3 Convergence of the 2D-Trellis Algorithm

It is obvious that the 2D-Trellis does not provide an ML or MAP solution to the decoding problem, since the division into constraint regions does not create conditionally independent regions. Therefore, the algorithm may be expected to work only for a subset of all the possible codes of a given kernel size. Unfortunately, for some codes the algorithm cannot converge and fails totally. In this section we try to find conditions on the generator polynomials of the code that characterize the codes for which the 2D-Trellis algorithm works. We are able to produce a necessary condition for the convergence of the algorithm.

Let us begin by analyzing the relationship between a constraint region and the code fragment associated with it. The code fragment for a region of size \( K_1 \times K_2 \) is given by:

\[ c_i = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} r(k_1, k_2) \cdot g_i \left( K_1 - k_1 - 1, K_2 - k_2 - 1 \right) \quad ; \quad i = 1, \ldots, n \] (6.2.15)

We now break the region to two sub-regions: The intersection with a neighboring region (let’s say on the left side), and the rest of the region:
Formally, we define the following divisions of a constraint region, according to its overlap with neighboring constraint region:

With the neighbor to the right:
\[ S_{i,j}^R = R_{i,j} \cap R_{i,j+1} \quad ; \quad T_{i,j+1}^L = R_{i,j+1} \setminus S_{i,j}^R \]

With the neighbor to the left:
\[ S_{i,j}^L = R_{i,j} \cap R_{i,j-1} \quad ; \quad T_{i,j-1}^R = R_{i,j-1} \setminus S_{i,j}^L \]

With the neighbor above:
\[ S_{i,j}^U = R_{i,j} \cap R_{i-1,j} \quad ; \quad T_{i-1,j}^D = R_{i-1,j} \setminus S_{i,j}^D \]

With the neighbor below:
\[ S_{i,j}^D = R_{i,j} \cap R_{i+1,j} \quad ; \quad T_{i+1,j}^U = R_{i+1,j} \setminus S_{i,j}^U \]

The sub-regions denoted by \( S \) correspond to the notion of a state in the 1D trellis, and sub-regions denoted by \( T \) correspond to the notion of a branch in the 1D trellis.

Note that using these divisions, a constraint region may now be represented as an ordered pair of any of those two groups:
\[ R_{i,j} = (T_{i,j}^L, S_{i,j}^R) = (S_{i,j}^L, T_{i,j}^R) = (T_{i,j}^U, S_{i,j}^D) = (S_{i,j}^U, T_{i,j}^D) \]

Using this notation we can re-write the message passing equations of the intersection nodes as
\[ A_{i,j}^{(t)}(s_L, t_R) = \sum_{t_L} p(t_L', s_L, y_{i,j-1}) \cdot A_{i,j-1}^{(t-1)}(t_L', s_L) \cdot H_{i-1,j}^{(t-1)}(t_L', s_L) \cdot Z_{i,j}^{(t-1)}(t_L', s_L) \quad (6.2.16) \]
\[ B_{i,j}(t_L, s_R) = \sum_{t_{i,j}} p(s_{R}, t_{R} | y_{i,j+1}) \cdot B_{i,j+1}(s_R, t'_{R}) \cdot H_{i,j+1}(s_R, t'_R) \cdot Z_{i,j+1}(s_R, t'_R) \] (6.2.17)

\[ H_{i,j}(s_U, t_D) = \sum_{t_{i,j}} p(t'_{U} ; s_U | y_{i-1,j}) \cdot A_{i-1,j}^{(i-1)}(t'_{U} ; s_U) \cdot B_{i,j}^{(i-1)}(t'_U ; s_U) \cdot H_{i-1,j}^{(i-1)}(t'_U ; s_U) \] (6.2.18)

\[ Z_{i,j}(t_U, s_D) = \sum_{t_{i,j}} p(s_{D} ; t'_{D} | y_{i-1,j}) \cdot A_{i-1,j}^{(i-1)}(s_D ; t'_D) \cdot B_{i,j}^{(i-1)}(s_D ; t'_D) \cdot Z_{i-1,j}^{(i-1)}(s_D ; t'_D) \] (6.2.19)

The code fragment may now be represented as a combination of the sub-regions \( s \) and \( t \):

\[ c_i = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-2} s(k_1, k_2) \cdot g_i(K_1-k_1-1, K_2-k_2-1) + \ldots \]

\[ + \sum_{k_1=0}^{K_1-1} t(k_1, 0) \cdot g_i(K_1-k_1-1, K_2-k_2-1) \] (6.2.20)

We can write this in shorthand vector form:

\[ c = \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} g'_0 \\ \vdots \\ g'_{n-1} \end{pmatrix} \cdot s + \begin{pmatrix} g''_0 \\ \vdots \\ g''_{n-1} \end{pmatrix} \cdot t = G's + G''t \] (6.2.21)

Where \( s \) is the column stack of the sub-region \( S \), \( t \) is the column stack of sub-region \( T \), and \( \{g'\} \) and \( \{g''\} \) are row vectors corresponding to the relevant elements of the kernels \( \{g\} \).

For the algorithm to work we have to make sure that the code fragment tells us something about the sub-regions. Specifically we want to know that only some of the possible sub-regions (not all of them) are mapped to a single code fragment. Since the code is linear, we only need to check that not all possible sub-regions \( U \) or \( S \) give \( c=0 \). Let’s assume that \( c=0 \), then:

\[ G's = \begin{pmatrix} g'_{0} \\ \vdots \\ g'_{n-1} \end{pmatrix} \cdot s = - \begin{pmatrix} g''_{0} \\ \vdots \\ g''_{n-1} \end{pmatrix} \cdot t = -G''t \] (6.2.22)

We have to impose some constraints on \( G' \) and \( G'' \) to ensure good message passing. Adjacent regions in the BP graph have overlapping columns or rows, depending on their relative locations. For the BP algorithm to work correctly we need good message passing both on the horizontal and the vertical directions.

We wish to thank Yaron Shani for his help in formulating the following:

**Proposal (Necessary condition for 2D message passing):** Non-trivial message passing occurs in horizontal (vertical) direction if:
\begin{align*}
\text{Im}(G') \neq \text{Im}(G'')
\end{align*}

(6.2.23)

Where \( \text{Im}(A) \) is the image of the matrix \( A \), i.e. \( \text{Im}(A) = \{ y | y = Ax \} \)

**Proof:** Consider messages passed in a certain direction, for example to the right. Assume that \( \text{Im}(G') = \text{Im}(G'') \). Then for every \( s \) there exist a unique \( t \) such that \( G's + G''t = 0 \). Now let us look at the first iteration of the algorithm, assuming the all zero codeword has been transmitted and received correctly. The region beliefs are initialized to:

\[
P(r|c = 0) = \begin{cases} 
1/|\text{Im}(G)| & \text{if } G \cdot r = 0 \\
0 & \text{otherwise}
\end{cases}
\]

The message passed in the right direction is therefore:

\[
A^{(r)}(s_L, s_R) \propto \sum_{r_c} p(r \mid c = 0) = \sum_{r_c} p(t'_L, s_L \mid c = 0) = p(s_L \mid c = 0) = 1/|\text{Im}(G)|
\]

Where the last equation follows from the fact that only a unique \( t \) satisfies \( Gr = 0 \) with any \( s \), so any non-zero value of \( P(r|c=0) \) is summed only once. Since the code is linear, the same holds also for \( c \neq 0 \). This means that the messages passed in the right direction are constant, which is trivial message passing.

In the 2D-Trellis algorithm messages are passed in four directions: up, down, left and right. For the algorithm to perform well we require that non-trivial message passing shall occur in at least one horizontal (left or right) and one vertical (up or down) directions.

**Corollary:** Systematic codes satisfy the 2-D Trellis condition on all 4 sides.

**Proof:** This follows from the fact that on dividing the constraint regions into sub-regions \( S \) and \( T \), the resulting \( G', G'' \) matrices of a systematic code can take one of two forms:

\[
G' = \begin{pmatrix} 0 & \vdots & e_{1,1}^{K_2,M} \\
g'_n \end{pmatrix} ; \quad G'' = \begin{pmatrix} e_{1,1}^{K_2,M} \vdots \\
g''_n \\
g''_n \end{pmatrix} \quad \text{or} \quad G' = \begin{pmatrix} e_{1,1}^{K_2,M} \vdots \\
g'_n \end{pmatrix} ; \quad G'' = \begin{pmatrix} 0 & \vdots \\
g''_n \end{pmatrix}
\]

Where \( e_i^{M \times 1} \) is a unit vector of size \( M \), \((1,0,0,...,0)\). Thus by construction, \( \text{Im}(G') \neq \text{Im}(G'') \), since either \( e_i^{M \times 1} \in G' \) or \( e_i^{M \times 1} \in G'' \) but not both.

**Example I (Code for which the 2D-trellis succeeds):**

Consider the following code with kernel support 2×2.

\[
g_0 = \begin{pmatrix} 0 & 1 \\
1 & 0 \end{pmatrix} \quad g_1 = \begin{pmatrix} 0 & 1 \\
1 & 1 \end{pmatrix}
\]

On horizontal division we have:

\[
G' = \begin{pmatrix} 0 & 1 \\
0 & 1 \end{pmatrix} ; \quad G'' = \begin{pmatrix} 1 & 0 \\
0 & 1 \end{pmatrix}
\]

We can see immediately that \( G' \) is not full rank, and therefore \( \text{Im}(G') \neq \text{Im}(G'') \), so the condition is satisfied on both horizontal directions (since the kernel support is 2×2, the sub-regions \( S \) and \( T \) are inter-changeable between the left and right directions).
The same happens for vertical division, and therefore the condition is satisfied for both vertical directions as well.

Although the condition is only necessary and not sufficient, the 2D-algorithm does converge for this code.

### Example II (Code for which the 2D-trellis fails):

Consider the following code with kernel support 2×2. The:

\[
\begin{pmatrix}
1 & 1 \\
1 & 0 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 1 \\
1 & 1 \\
\end{pmatrix}
\]

Let us construct a table of the code fragments generated by these kernels for each of the 16 possible values of a constraint region. This is shown in Table 6-3.

<table>
<thead>
<tr>
<th>Code bits</th>
<th>Possible region values</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00 01 10 11 11</td>
</tr>
<tr>
<td>01</td>
<td>00 01 10 11 00</td>
</tr>
<tr>
<td>10</td>
<td>00 01 10 11 10</td>
</tr>
<tr>
<td>11</td>
<td>00 01 10 11 11</td>
</tr>
</tbody>
</table>

Let us look at the first row of the table, that shows the constraint region values associated with the code fragment 00. We divide the constraint region to sub-regions \( S \) and \( T \). On vertical division each sub-region corresponds to a single row of the region \( R \), and on horizontal division the sub-regions correspond to the columns of \( R \). No matter which division we choose, we can see that the sub-regions take all the possible binary values for code fragment 00. This means that the code fragment does not give any information on the value of the sub-regions. Since the code is linear, the same happens for all other code fragments. This means that the 2D-Trellis algorithm cannot work for this code.

We can arrive at the same conclusion by testing the necessary condition for message passing. For instance, on horizontal division we get:

\[
G' = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} ; \ G^* = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}
\]

It is easy to verify that these two matrices have the same image, and the condition is not satisfied. The same happens on vertical division.

### 6.3 Results for Sample Codes

We summarize the discussion of the 2D-Trellis algorithm by presenting some results for codes. We compare the following decoding algorithms:

- The 1-D Viterbi algorithm, which is optimal in the ML sense.
- Two variants of the 2D-Trellis algorithm: Sum-Product (SP) and Minimum-Sum (MS)
The codes were simulated over the AWGN channel. The reference point for performance was chosen arbitrarily to be $P_w=10^{-3}$. In this section, best codes refer to the codes with the highest minimum distance. In case there are two codes with the same minimum distance, we consider as “better” the one with the smaller multiplicity of codewords with minimum weight. We do not consider higher weights for this comparison.

### 6.3.1 Kernel Support 2×2 ; Information Support 4×4

We start our discussion with the codes from examples I and II of the previous section. The information support is 4×4, which makes these codes (32,16) codes.

In Figure 6-7 the performance of the code from example I is shown. This code satisfies the 2D-Trellis condition and has $d_{\text{min}}=4$.

As expected the Viterbi algorithm closely follows the union bound asymptote. The 2D-Trellis variants also successfully decode the code, being nearly indistinguishable from the optimum.

In Figure 6-8 the performance of the code from example II is shown. This code has $d_{\text{min}}=6$, but does not satisfy the 2D-Trellis condition. As can be seen, the 2D-Trellis variants fail to decode, giving worse performance than uncoded transmission.

Table 6-4 summarizes the performance of the codes tested in this section.

<table>
<thead>
<tr>
<th>code#</th>
<th>$d_{\text{min}}$</th>
<th>$A(d_{\text{min}})$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>Viterbi $E_b/N_0$ [dB] for $P_w=10^{-3}$</th>
<th>2D-Trellis Min-Sum Degradation [dB]</th>
<th>2D-Trellis Sum-Prod. Degradation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>0 1</td>
<td>1 1</td>
<td>5.40</td>
<td>+0.0</td>
<td>+0.0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>32</td>
<td>1 1</td>
<td>0 1</td>
<td>4.52</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Figure 6-7 Code 1, 2-D Trellis Performance: $K=2x2$, $N=4x4$ ; $d_{\text{min}}=4$

Figure 6-8 Code 2, 2-D Trellis Performance: $K=2x2$, $N=4x4$ ; $d_{\text{min}}=6$
### 6.3.2 Kernel Support 2×2 ; Information Support 6×6

For information support 6×6 there are two codes with $d_{\text{min}}=6$ that are suitable for the 2D-trellis algorithm.

- **Code I:** $g_1=[1 1 ; 0 1]$; $g_2=[1 1 ; 1 1]$.
- **Code II:** $g_1=[1 1 ; 1 0]$; $g_2=[1 1 ; 0 1]$.

The performance of code I is shown in Figure 6-9. The 2D-Trellis variants work successfully, with a degradation of 0.70 dB for the MS variant, and 0.36 for the SP variant.

The performance of code II is shown in Figure 6-10. The 2D-Trellis variants succeed again, with a degradation of 0.28 dB for the MS variant, and only 0.21 dB for the SP variant.

Results for these codes are summarized in Table 6-5.

#### Table 6-5 2D-Trellis performance summary for codes with $K=2\times2$, $N=6\times6$

<table>
<thead>
<tr>
<th>code#</th>
<th>$d_{\text{min}}$</th>
<th>$A(d_{\text{min}})$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>Viterbi $E_b/N_0$ [dB] for $P_w=10^{-3}$</th>
<th>2D-Trellis Sum-Prod. Degradation [dB]</th>
<th>2D-Trellis Min-Sum Degradation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>12</td>
<td>1 1</td>
<td>1 1</td>
<td>4.25</td>
<td>+0.36</td>
<td>0.70</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>48</td>
<td>1 1</td>
<td>1 0</td>
<td>4.62</td>
<td>+0.21</td>
<td>+0.28</td>
</tr>
</tbody>
</table>

Figure 6-9 Code 3, 2-D Trellis Performance: $K=2\times2$, $N=6\times6$ ; $d_{\text{min}}=6$
6.3.3 Kernel Support 3×3 ; Information Support 6×6

To test the 2D-Trellis algorithm for codes with kernel support 3×3 we first have to find suitable candidates for this algorithm. Therefore we have tested each of the 8317 distinct non-degenerate codes for the satisfaction of the 2D-Trellis condition. The results are plotted in Figure 6-11 as a histogram of codes which satisfy the condition for 0,1,2,3 or 4 sides of the constraint regions. In order for a code to be potentially suitable for the 2D-Trellis algorithm the condition has to be satisfied on at least two sides. Since the condition is only necessary and not sufficient this is only a first screening of the codes, and the algorithm will have to be tested in order to verify that it does indeed work.

According the histogram there are 2027 codes that satisfy the condition on at least 2 sides, of which 1049 are (pseudo-)invertible. As expected, all systematic codes satisfy the condition for all 4 sides. Some of the best codes that are potentially suitable for 2D-Trellis decoding are given in Table 6-6.
Figure 6-11 Codes satisfying the 2D-Trellis condition. K=3×3 ; N= 6×6.
In Table 6-7 we tabulate the performance results for several codes for which we tested the 2-D algorithm. The WER curves for these codes are plotted in Figure 6-12 to Figure 6-16. Of the codes tested, code #6 does not satisfy the 2-D trellis condition, and thus the algorithm fails for this code. The last code (#9) has \(d_{\text{min}}=8\) and is the best systematic code found. It can be seen that the 2-D trellis algorithm performs at 1.2 to 2.0 dB away from the optimum maximum likelihood decoder. Additionally, it can be seen that most of the codes operate at \(\sim 3.25\) dB for \(P_{w}=10^{-3}\), which is 0.41 dB for the SPLB for codes of this size.

<table>
<thead>
<tr>
<th>code#</th>
<th>(d_{\text{min}})</th>
<th>(A(d_{\text{min}}))</th>
<th>(g_1)</th>
<th>(g_2)</th>
<th>Viterbi (E_b/N_0) [dB] for (P_w=10^{-3})</th>
<th>2D-Trellis Min-Sum Degradation [dB]</th>
<th>2D-Trellis Min-Sum Degradation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>309</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>+1.74</td>
<td>+1.75</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>78</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>72</td>
<td>0 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>+1.53</td>
<td>+1.53</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>36</td>
<td>1 0 1</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>+1.47</td>
<td>+1.54</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>36</td>
<td>1 0 0</td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>+1.40</td>
<td>+1.40</td>
</tr>
</tbody>
</table>

* This code is also invertible
Figure 6-12 Code 5, 2-D Trellis Performance: \( K=3 \times 3 \); \( N=6 \times 6 \); \( d_{\text{min}}=12 \);

Figure 6-13 Code 6, 2-D Trellis Performance: \( K=3 \times 3 \); \( N=6 \times 6 \); \( d_{\text{min}}=12 \);
Figure 6-14 Code 7, 2-D Trellis Performance: $K=3 \times 3$; $N=6 \times 6$; $d_{\min}=11$

Figure 6-15 Code 8, 2-D Trellis Performance $K=3 \times 3$; $N=6 \times 6$; $d_{\min}=11$
Figure 6-16 Code 9, 2-D Trellis Performance: $K=3\times3$ ; $N=6\times6$; $d_{\text{min}}=8$
7 Decoding in the Parity Check Domain

An alternative approach for soft decoding is to try to estimate the transmitted codeword, and then apply an inverse encoding operation to recover the original information word. This is the technique commonly used for decoding LDPC. This is depicted in Figure 7-1. We refer to these algorithms as *BP in the Parity Check domain*.

![Figure 7-1 Two Stage Decoding: Decoder and Inverse Encoder](image)

For systematic codes, the original information is part of the codeword itself and can be easily extracted. For non-systematic code, a non-trivial inverse encoding has to be performed. Unfortunately, in the case of a decoding error, the inverse encoding operation magnifies the number of errors, such that a single erroneous code bit may cause many erroneous information bits.

Decoding in the parity check domain requires the use of a parity check matrix or *syndrome* formers. We limit ourselves to discussion of codes of rate $1/n$, for which the syndrome formers are readily available, see section 3.3.1.

7.1 Classical Loopy Belief Propagation: PC-LBP

In this approach we construct the classical bi-partite Tanner graph [30] typically used for decoding LDPC. The resulting graph has a very regular structure, which is similar to that of regular LDPC codes. There are however, two important differences with respect to LDPC codes. First, the edges between variable and checks follow a very regular pattern, whereas in LDPC the edges tend to form a random network. Second, the graph is not sparse, and thus has many short cycles. Therefore, it is not guaranteed that the algorithm shall converge to the optimal solution (in the ML/MAP sense), or at all. Surprisingly, however, in several cases the algorithm does provide satisfactory results.

7.1.1 Description of the algorithm

Given the following rate $1/n$ code: $G(x, y) = [g_1(x, y), ..., g_n(x, y)]$

A simple construction for a parity check matrix is given by:

$$H(x, y) = \begin{pmatrix} g_2 & g_1 & 0 & \cdots & 0 \\ g_3 & 0 & g_1 & \vdots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ g_n & 0 & \cdots & 0 & g_1 \end{pmatrix}$$

This induces a semi-regular Tanner graph, where check nodes have degrees $d_{c,1}, d_{c,2}, ..., d_{c,k}$, and variable nodes have degrees $d_{v,1}, d_{v,2}, ..., d_{v,n}$.

**Example:** Let us consider the following 2×2 code over 4×4 sequences:
The syndrome former is:
\[ H(x, y) = (1 + x + y, x + y) \iff h_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, h_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \]

The Tanner graph for this code is shown in Figure 7-2.

In this graph, every check node (squares) has degree \( d_c = 5 \); Variable nodes associated with \( v(1) \) (dark circles) have degree \( d_v = 3 \) and variable nodes associated with \( v(2) \) (white circles) have degree \( d_v = 2 \).

Since all variable nodes in the graph take their values from a binary alphabet, the messages passed can be converted from vectors of probabilities (beliefs) to scalar likelihood ratios or log-likelihood ratios (LLRs).

For the Sum-Product algorithm the message update equations for the LLR are (see [4], chapter 17),

\[
\prod_{a \in N(i) \cup a} m_{d' \rightarrow i} \quad (7.1.1)
\]

\[
m_{a \rightarrow i} = 2 \tanh^{-1} \left( \prod_{j \in N(a) \cup i} \tanh \left( \frac{n_{j \rightarrow a}}{2} \right) \right) \quad (7.1.2)
\]

And for the Minimum-Sum algorithm the message update equations for the LLR are,
\[ n_{i \rightarrow a} = \sum_{a \in N(i) \setminus a} m_{a \rightarrow i} , \quad (7.1.3) \]

\[ m_{a \rightarrow j} = \left( \prod_{j \in N(a) \setminus j} \text{sgn}(n_{j \rightarrow a}) \right) \min_{j \in N(a) \setminus j} \left( n_{j \rightarrow a} \right) \quad (7.1.4) \]

This representation is advantageous in several respects: First, it converts the problem of decoding a 2-D CC into the known Gallager BP algorithm [29], widely used to decode LDPC codes. Next, the fact that messages are scalar LLRs instead of belief vectors reduces the number of computations and memory required to store the messages.

From a complexity point of view, the PC-LBP algorithm has a relatively low computational complexity. The number of operations is linear with the dimensions of the input, and at worst polynomial with the weight of the kernel matrix. Let \( \text{wt}(g) = \text{wt}(g_1) + \text{wt}(g_2) \) be the combined weight of the generator polynomials (Note: for rate \( \frac{1}{n} \) codes one should take \( \text{wt}(g) = \max_{i=2, \ldots, n} \{ \text{wt}(g_1) + \text{wt}(g_i) \} \)). The graph has \((n-k)N_1N_2\) check nodes and \(nN_1N_2\) variable nodes. The complexity is governed by the check nodes, with each check node connected to \(\text{wt}(g)\) variable nodes. Therefore a check node is required to compute \(\text{wt}(g) [\text{wt}(g)-1]\) messages at each iteration. Thus the complexity per information bit is at worst of \(O(\text{wt}(g)^2)\). Note that by construction \(\text{wt}(g) \leq 2K_1K_2\). The actual number of operations may be reduced by a more sophisticated implementation.

Finally, due to the regular structure of the graph, this decoder can be easily implemented in hardware. We can, for example, assign a specific processor for each type of check node and variable node (at most \(k+n\) different types of processors). Each processor may be replicated several times according to the message scheduling policy and the degree of parallelism desired.

However, the graph constructed in this manner has many loops, and is not guaranteed to converge to the MAP/ML solution for any code.

### 7.1.2 Constructing an Inverse BP Network

When the code is (pseudo)-invertible, we add the inverse encoder equations to the parity check and form an extended parity check matrix.

Suppose the encoder has a pseudo-inverse: \( G^*(x, y) = [q_1(x, y), \ldots, q_n(x, y)] \), with:

\[ \sum_{i=1}^{n} q_i(x, y) g_i(x, y) = x^\alpha y^\beta \quad (7.1.5) \]

Then we can form the extended parity check equation:
This equation can be used to reconstruct the information sequence \( u \) from the coded bits \( \{ v_i \} \) and can be integrated into the BP decoding, where the information nodes \( u \) are considered as erased. However, the usefulness of the inverse network deteriorates as the weight of the inverse matrix \( G^\# \) increases. For example, if \( \text{wt}(G^\#)=n \), then \( n \) coded bits are required to reconstruct a single information bit, which can be written as:

\[
\begin{align*}
\sum_{j=1}^{n} v(i,j) &= u_i
\end{align*}
\]  

(7.1.7)

Where \( \{ v(i,j) \} \), \( j=1,...,n \), is the set of coded bits which participate in the reconstruction of information bit \( i \). Now, assume the error probability for a coded bit is \( p_c \). The bit \( u_i \) will be erroneous if an odd number of errors occurs in \( \{ v(i,j) \} \). Since the code is linear, we can assume that the all-zeros codeword has been transmitted, so that an error occurs in \( u_i \) or \( v(i,j) \) if they have the value 1. Then the probability of information bit error probability for inverse encoder of weight \( n \) is given by the well known formula:

\[
\begin{align*}
P_b &= \frac{1}{2} \left( 1 - (1 - 2p_c)^n \right) \quad \text{as } p_c \to 0
\end{align*}
\]  

(7.1.8)

This means that the “heavier” the code inverse, the less effective it is. In Figure 7-3 we plot \( P_b \) as a function of \( p_c \), and in Figure 7-4 we plot \( p_b/p_c \), for code inverses of weights 1 to 6. It can be seen how the error probability increases with the weight of the code inverse. Therefore we should try to find invertible codes with low-weight inverses. We shall refer to this codes as simple invertible codes. We have chosen to define a simple invertible code as one with a pseudo-inverse matrix \( G^\# \) with weight \( \text{wt}(G^\#) \leq 4 \).
Figure 7-3 Code Inverse: Info. Bit error prob. vs. Code bit error prob.

Figure 7-4 Code Inverse: Ratio of Info. Bit error prob. to Code bit error prob.
7.1.3 Other BP Networks

More BP networks may be constructed by combinations of the matrices $G$, $H$, and $G^\#$. The fundamental parity check equations that use each of these matrices may be given as:

$$u \cdot G + v = 0 ; \quad u \cdot 0 + v \cdot H = 0 ; \quad v \cdot G^\# + u = 0$$  \hspace{1cm} (7.1.9)

Combining any two or three of these equation leads to a BP network that is potentially capable of decoding the information sequence $u$. For example, the network described by the matrix:

$$\tilde{H}_1 = \begin{pmatrix} 0 & H \\ I & G^\# \end{pmatrix}$$  \hspace{1cm} (7.1.10)

Is equivalent to the parity-check-pseudo-inverse network described in section 7.1.2.

A generator-pseudo-inverse construction can be achieved by taking the generator matrix and the pseudo inverse:

$$\tilde{H}_2 = \begin{pmatrix} G & I \\ I & G^\# \end{pmatrix}$$  \hspace{1cm} (7.1.11)

Finally we can combine all three equations and get:

$$\tilde{H}_3 = \begin{pmatrix} 0 & H \\ G & I \\ I & G^\# \end{pmatrix}$$  \hspace{1cm} (7.1.12)

While we have experimented with some of these constructions, we do not have definite simulation results. Our preliminary results show that these networks all have similar performance, but improvement may be achieved by some tweaking (e.g., Gaussian elimination). We leave the thorough investigation of these networks for further research.
Generalized Belief Propagation – PC-GBP

Generalized belief propagation seeks to reduce the problems caused by loops in the graph, by grouping nodes together into regions. The algorithm as described in section 5.2 has three parts: The construction of the region graph, initialization of the region beliefs, and the message passing stage. The message passing stage also has several variants, including Sum-Product, Minimum-Sum and Max-Product. In the sequel we describe how this algorithm applies to the Tanner graph of the 2D TBCC, and we demonstrate it on the code shown in Figure 7-2.

7.2.1 Constructing a Region Graph

The first stage of the algorithm is to construct a valid region graph. We have tried three different constructions which we will describe below.

All constructions start by defining the set of largest regions in the graph, from which intersections will be constructed, and denote this set as $R_0$. A large region is defined to contain all variable nodes that are connected to a single factor node. Strictly speaking, the large region contains also the factor node of the parity check function, but we can effectively disregard this by limiting the number of valid states of the nodes in the region only to those states which satisfy the parity check equation.

We then continue by forming intersections of the large regions, and intersections of former intersections, until we cannot find any more intersections, as described in section 5.2.3.

The three constructions differ in the way they treat single variable nodes.

**Construction I:** This is the construction described by [34]. This construction simply stops after no more intersections can be found. If there are single variable nodes that do not form their own regions (i.e., that do not contain only this single variable), then they are not connected to the graph.

**Construction II:** Is described in [35]. Here, all single variable nodes that were not directly formed as regions from intersections of previous regions, are simply attached as children to the smallest region that contains them.

**Construction III:** This is our own construction. After all the intersections have been found in the usual manner, we allow to form more intersections between nodes of different layers. I.e., we allow to create new intersections from the entire set of existing regions $R = R_0 \cup R_1 \cup \ldots \cup R_i$. We do not allow to form an intersection between a region and one of its ancestors. This gives single variable nodes a chance to be formed as intersections of regions in different layers, $R_i$ and $R_j$, were $i \neq j$. Still, if some single variables do not form their own regions they are not connected to the graph.

We have found that construction III is the most consistent in its performance for the codes tested, whereas constructions I and II sometimes performed much worse than the conventional BP.

**Example:** Let us illustrate the different constructions by considering the familiar Hamming (7,4) code:
The region graphs that are formed by the different constructions are shown in Figure 7-5. All constructions start with the large regions that consist of the nodes that participate in each check equation, and the factor node (not listed). They are \{2,3,4,5\}, \{1,3,4,6\} and \{1,2,4,7\}. From these regions we form the second layer of intersections \{3,4\}, \{2,4\} and \{1,4\}. The last layer of intersections consists of the region \{4\}. Construction I terminates here, since there are no more nodes to be found. Construction II, on the other hand, continues to form the single-variable regions \{1\}, \{2\}, \{3\}, \{5\}, \{6\}, and \{7\}, and connects each of them to the smallest region that contains it. Construction III in this case is identical to construction I. In principle, construction III picks up where construction I terminated, and tries to form intersections out of the entire region set: \{4\}, \{1,4\}, \{2,4\}, \{3,4\}, \{2,3,4,5\}, \{1,3,4,6\} and \{1,2,4,7\}, starting by searching for intersections of small regions, and progressing towards large regions. However, in this case, there are no new intersections to be found.

![Figure 7-5 GBP Region Graph Construction for Hamming(7,4) code](image)

The next example demonstrates the different constructions on a 2-D TBCC.

**Example:** Take the code from section 7.1.1, represented by the Tanner graph in Figure 7-2. According to the connectivity between variable and check nodes, we can form the following 16 large regions:

<table>
<thead>
<tr>
<th>(R_0)</th>
<th>(R_1)</th>
<th>(R_2)</th>
<th>(R_3)</th>
<th>(R_4)</th>
<th>(R_5)</th>
<th>(R_6)</th>
<th>(R_7)</th>
<th>(R_8)</th>
<th>(R_9)</th>
<th>(R_{10})</th>
<th>(R_{11})</th>
<th>(R_{12})</th>
<th>(R_{13})</th>
<th>(R_{14})</th>
<th>(R_{15})</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0,3,12,19,28}</td>
<td>{0,4,7,16,23}</td>
<td>{4,8,11,20,27}</td>
<td>{8,12,15,24,31}</td>
<td>{1,13,16,29}</td>
<td>{1,4,5,17,20}</td>
<td>{5,8,9,21,24}</td>
<td>{9,12,13,25,28}</td>
<td>{2,14,17,30}</td>
<td>{2,5,6,18,21}</td>
<td>{6,9,10,22,25}</td>
<td>{10,13,14,26,29}</td>
<td>{3,15,18,31}</td>
<td>{3,6,7,19,22}</td>
<td>{7,10,11,23,26}</td>
<td>{11,14,15,27,30}</td>
</tr>
</tbody>
</table>

The large regions will form the top layer of the region graph. From these regions we can form the following 16 intersection regions, which will form the middle layer of the region graph:

<table>
<thead>
<tr>
<th>(R_{16})</th>
<th>(R_{17})</th>
<th>(R_{18})</th>
<th>(R_{19})</th>
<th>(R_{20})</th>
<th>(R_{21})</th>
<th>(R_{22})</th>
<th>(R_{23})</th>
<th>(R_{24})</th>
<th>(R_{25})</th>
<th>(R_{26})</th>
<th>(R_{27})</th>
<th>(R_{28})</th>
<th>(R_{29})</th>
<th>(R_{30})</th>
<th>(R_{31})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_0\cap R_7)</td>
<td>{3,19}</td>
<td>(R_2\cap R_5)</td>
<td>{1,17}</td>
<td>(R_4\cap R_{11})</td>
<td>{7,23}</td>
<td>(R_8\cap R_{13})</td>
<td>{11,27}</td>
<td>(R_0\cap R_{13})</td>
<td>{12,28}</td>
<td>(R_2\cap R_{15})</td>
<td>{14,30}</td>
<td>(R_4\cap R_8)</td>
<td>{4,20}</td>
<td>(R_9\cap R_{12})</td>
<td>{8,24}</td>
</tr>
</tbody>
</table>
Construction I would terminate here, since there are no new intersections to be found in the last layer! Construction II would continue by attaching a pair of single variable regions to each of the regions found in layer 2.

However, construction III goes on to form a third layer in which we have 16 small regions, composed of single variable nodes, and formed by intersections of nodes from layer 1 and layer 2:

\[
\begin{align*}
R_{32} &= R_{18} \cap R_0 = \{0\} \\
R_{33} &= R_{20} \cap R_1 = \{1\} \\
R_{34} &= R_{22} \cap R_2 = \{2\} \\
R_{35} &= R_{16} \cap R_3 = \{3\} \\
R_{36} &= R_{25} \cap R_4 = \{4\} \\
R_{40} &= R_{20} \cap R_8 = \{8\} \\
R_{44} &= R_{17} \cap R_{13} = \{12\} \\
R_{31} &= R_{26} \cap R_5 = \{5\} \\
R_{41} &= R_{19} \cap R_9 = \{9\} \\
R_{45} &= R_{10} \cap R_{13} = \{13\} \\
R_{38} &= R_{27} \cap R_6 = \{6\} \\
R_{42} &= R_{31} \cap R_{10} = \{10\} \\
R_{46} &= R_{21} \cap R_{14} = \{14\} \\
R_{39} &= R_{24} \cap R_7 = \{7\} \\
R_{43} &= R_{28} \cap R_{11} = \{11\} \\
R_{47} &= R_{23} \cap R_{15} = \{15\}
\end{align*}
\]

Note that there are no single-variable regions that contain variable nodes 16 to 31. In Figure 7-6 the nine central largest regions of this graph are shown (the tail biting regions are omitted for clarity). It can be seen that for each region the two bottom left variable nodes coincide with the two top right variable nodes of an adjacent region. These intersections form the middle layer of the graph.

A part of the resulting region graph is shown in Figure 7-7. The “ancestry” of the variable nodes that are members of the central region is shown.

Figure 7-6 Example: Coverage of the large regions of a 2-D TBCC
7.2.2 Message Initialization and Message Passing

In soft decoding algorithms, the decoder is usually initialized with the conditional probabilities of the received coded bits, based on the channel output observations. That is, if the channel outputs the sequence \( \{ y_i \} \) when the codeword \( \{ c_i \} \) is transmitted, the decoder is fed with the sequence of conditional probabilities \( p_i = p(c_i | y_i) \). We would like to initialize the GBP decoder with the same values.

To correctly initialize the beliefs of the various regions, we have to treat the incoming channel information as factor nodes which are attached to every single variable node. These factor nodes simply perform the function:

\[
f(c_i) = p(c_i | y_i) = p_i
\]

Therefore, each region node contains single variable nodes, and also the factor nodes associated with their channel information. Recall equation (5.2.35) which combines all the factors within a region to a single function:

\[
\tilde{f}_R(x_R) = \left( \prod_{a \in A_R} f_a(x_a) \right)^{c_a}
\]

This means that the correct initialization of the beliefs of each region consists of the product of the channel information for all single variables, raised to the power of that region’s counting number:

\[
b^{(0)}(x_R) = \prod_{i \in R} p_i^{c_a} \tag{7.2.1}
\]

After the initialization phase the message passing starts. The sum-product variation of the algorithm consists of equations (5.2.33) to (5.2.37) which we repeat here for quick reference:
**Sum Product GBP:**

Node beliefs:

\[ b_R(x_R) = \tilde{f}_R(x_R) \prod_{C \in C(R)} n_{C \rightarrow R}(x_C) \prod_{P \in P(R)} m_{P \rightarrow R}(x_P) \]  

(7.2.2)

Pseudo-messages to parents:

\[ n^0_{R \rightarrow P}(x_R) = b_R(x_R) / m_{P \rightarrow R}(x_R) \]  

(7.2.3)

Pseudo-messages to children:

\[ m^0_{R \rightarrow C}(x_R) = \sum_{x_C \in \mathcal{X}_C} b_R(x_R) / n_{C \rightarrow R}(x_C) \]  

(7.2.4)

Messages to parents:

\[ n_{R \rightarrow P}(x_R) = \left(n^0_{R \rightarrow P}(x_R) \right)^{\beta_R} \left(m^0_{P \rightarrow R}(x_R) \right)^{1-\beta_R} \]  

(7.2.5)

Messages to Children:

\[ m_{R \rightarrow C}(x_R) = \left(n^0_{C \rightarrow R}(x_C) \right)^{1-\beta_C} \left(m^0_{R \rightarrow C}(x_C) \right)^{\beta_C} \]  

(7.2.6)

By taking the logarithm of equations (7.2.1) to (7.2.6) and applying the approximation \( \log(\sum_i a_i) \approx \max_i(a_i) \) we arrive at the so-called minimum-sum variant, which operates in the log-probability domain.

**Minimum-Sum GBP:**

Node beliefs:

\[ b_R(x_R) = \log \tilde{f}_R(x_R) + \sum_{C \in C(R)} n_{C \rightarrow R}(x_C) + \sum_{P \in P(R)} m_{P \rightarrow R}(x_P) \]  

(7.2.7)

Pseudo-messages to parents:

\[ n^0_{R \rightarrow P}(x_R) = b_R(x_R) - m_{P \rightarrow R}(x_R) \]  

(7.2.8)

Pseudo-messages to children:

\[ m^0_{R \rightarrow C}(x_R) = \max \{b_R(x_R) - n_{C \rightarrow R}(x_C)\} \]  

(7.2.9)

Messages to parents:

\[ n_{R \rightarrow P}(x_R) = \beta_R \cdot n^0_{R \rightarrow P}(x_R) + (\beta_R - 1) \cdot m^0_{P \rightarrow R}(x_R) \]  

(7.2.10)

Messages to Children:

\[ m_{R \rightarrow C}(x_R) = (\beta_C - 1) \cdot n^0_{C \rightarrow R}(x_C) + \beta_C \cdot m^0_{R \rightarrow C}(x_C) \]  

(7.2.11)
The beliefs of the regions are initialized using the log-probabilities:

$$b^{(0)}(x_R) = \sum_{i \in R} c_i \cdot \log(p_i)$$  \hspace{1cm} (7.2.12)

There is also a third variant, known as Max-Product. This variant is identical in its equations to the Sum-Product algorithm, with only equation (7.2.4) replaced by:

**Max-Product GBP:**

$$m^0_{R \rightarrow C}(x_R) = \max_{x_a \in x_C} \left\{ b_R(x_R) / n_{C \rightarrow R}(x_C) \right\}$$  \hspace{1cm} (7.2.13)

It can be seen that one can arrive at this variant by exponentiating all the equations of the Minimum-Sum variant, and therefore the two are expected to be identical up to numerical accuracy of the computations.

The various GBP variants tend to produce over-confident messages which result in poor performance. Therefore, in [36] a heuristic “forgetting factor” is introduced which slowly updates the messages by combining their new nominal values with their old values from the previous iteration. The update rule proposed in [36] is simply:

$$\{m^{\text{new}}\} = (1-\alpha)\{m^{\text{old}}\} + \alpha\{m^{\text{update}}\}$$  \hspace{1cm} (7.2.14)

Where $\alpha$ is simply set to 0.5, and $m$ represents a message from parent to child or vice versa. We have found this update rule adequate for the minimum-sum variant, but found that many times a lower value of $\alpha$ is required. We have found that it is often useful to set $\alpha=1/M$, where $M$ is the number of the layers in the GBP region graph.

To apply the update rule in (7.2.14) to the sum-product or max-product variants, it has to be exponentiated, which yields:

$$\{m^{\text{new}}\} = \{m^{\text{old}}\}^{1-\alpha} \{m^{\text{update}}\}$$  \hspace{1cm} (7.2.15)

This update rule can be seen as a geometric weighted average of the new and old messages, as opposed to the arithmetic weighted average used in the minimum-sum variant.

The complexity of the PC-GBP algorithm is governed by the size of its largest region. This region contains $\text{wt}(g)$ variable nodes, and therefore has to compute $2^{\text{wt}(g)}$ beliefs at each iteration. In contrast to the PC-LBP algorithm, the complexity is now exponential with $\text{wt}(g)$. This is because the PC-GBP algorithm has to compute the joint beliefs for the entire group of variable nodes, the largest of which contains $\text{wt}(g)$ nodes. Thus the complexity of this algorithm is $O(2^{\text{wt}(g)})$ per information bit. Again, the actual number of operations may be reduced by a more sophisticated implementation.
7.3 Results for Sample Codes

In this section we give performance results for both the LBP and GBP algorithms and their respective variants. In addition to the parity-check construction used for decoding, we have also used a code inverse construction where a code inverse exists. For the code inverse construction an LBP algorithm was used to recover the information bit LLRs from the code bit LLRs generated by the decoder. Where a polynomial code inverse was not available, a recursive tree search algorithm was used to recover the underlying information bits from hard decisions on the code bit LLRs.

For comparison purposes, the results presented are for the same codes of section 6.3.

7.3.1 Kernel Support 2×2 ; Information Support 4×4

For our two K=2×2, N=4×4 codes, the GBP algorithms have a significant advantage over the standard LBP. As usual, our reference point is $P_w=10^{-3}$.

For code #1, the LBP algorithms perform at 0.32 dB away from the optimum, whereas GBP manages to operate at about only ~0.08 dB away. This represents an improvement of 0.24dB. Results for this code are plotted in Figure 7-8.

For code #2, degradation is more severe for both GBP and LBP. However, GBP manages to improve ~0.6 dB over LBP. Results for this code are plotted in Figure 7-9.

The results for these codes are summarized in Table 7-1.

<table>
<thead>
<tr>
<th>code#</th>
<th>$d_{min}$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>Degradation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LBP-SP</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0110</td>
<td>1110</td>
<td>0.32</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1101</td>
<td>0111</td>
<td>0.90</td>
</tr>
</tbody>
</table>
Figure 7-8 Code 1, PC-BP Performance: K=2×2 ; N=4×4; d_{min}=4

Figure 7-9 Code 2, PC-BP Performance: K=2×2 ; N=4×4; d_{min}=6
7.3.2 Kernel Support $2 \times 2$ ; Information Support $6 \times 6$

For our test codes with $K=2 \times 2$, $N=6 \times 6$, we see again that using GBP algorithms yields a significant advantage over using the standard LBP. The results for the test codes are summarized in Table 7-2.

This advantage is very prominent for code #3, where GBP algorithms gain between 1.66-2.43 dB over the standard LBP, and perform only 0.26 dB away from the optimum. Performance curves for this code are plotted in Figure 7-10.

The advantage is smaller, but still significant for code #4, where GBP algorithms gain ~0.7 dB over the standard LBP, and perform 0.44 dB away from the optimum. Performance curves for this code are plotted in Figure 7-11.

<table>
<thead>
<tr>
<th>code#</th>
<th>$d_{\text{min}}$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>Degradation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LBP-SP</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1 1 0 1</td>
<td>1 1</td>
<td>1.96</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1 1 0 0</td>
<td>1 1</td>
<td>1.19</td>
</tr>
</tbody>
</table>

![Figure 7-10 Code 3, PC-BP Performance: $K=2 \times 2$ ; $N=6 \times 6$; $d_{\text{min}}=6$](image-url)
7.3.3 Kernel Support 3×3 ; Information Support 6×6

In the case of K=3x3, N=6x6 codes results are mixed. Performance results for these codes are summarized in, and plotted in to.

There are some codes, for example codes #5 and #7 for which the GBP performs significantly better than the LBP. In fact, for these codes the LBP almost fails to improve anything over uncoded transmission. However, for other codes (#6 and #8), the GBP performs similarly to the LBP, usually giving only a slight advantage. There is one code, #9, for which the GBP performs slightly worse than the LBP, by about 0.5 dB.
Table 7-3 PC-BP Performance summary for codes with $K=3\times3$, $N=6\times6$

<table>
<thead>
<tr>
<th>code#</th>
<th>$d_{\text{min}}$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>Degradation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LBP-SP</td>
<td>LBP-MS</td>
<td>GBP-SP</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>5.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 1 0</td>
<td>1 1 1</td>
<td>1.67</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>1 1 0</td>
<td>0 1 1</td>
<td>2.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 1 1</td>
<td>1 1 1</td>
<td>2.80</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>0 1 1</td>
<td>1 1 1</td>
<td>4.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>3.11</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>1 0 1</td>
<td>1 1 1</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>1.07</td>
</tr>
</tbody>
</table>

Figure 7-12 Code 5, PC-BP Performance: $K=3\times3$ ; $N=6\times6$; $d_{\text{min}}=12$
Figure 7-13 Code 6, PC-BP Performance: $K=3\times3$ ; $N=6\times6$; $d_{\text{min}}=12$

Figure 7-14 Code 7, PC-BP Performance: $K=3\times3$ ; $N=6\times6$; $d_{\text{min}}=11$
Figure 7-15 Code 8, PC-BP Performance: $K=3 \times 3$ ; $N=6 \times 6$; $d_{\text{min}}=11$

Figure 7-16 Code 9, PC-BP Performance: $K=3 \times 3$ ; $N=6 \times 6$; $d_{\text{min}}=8$
8 Conclusion

We conclude this work by a short summary of the work done, and with proposal of future research directions.

8.1 Research Summary

In this dissertation we have combined tools from several fields of research, including commutative algebra, coding theory and probabilistic inference, and have applied them to the subject of two-dimensional convolutional codes. We conclude this work with a summary of the main contributions.

We began this dissertation with a discussion of algebra of multivariate polynomial rings. Previous works have already established that this theory is fundamental to the understanding of multidimensional convolutional codes (m-D CCs). In this framework we have extended the theory of m-D CCs to tail-biting 2-D CCs (2-D TBCCs), and have shown how to find non-degenerate codes in this family (i.e., codes that are bijective, or one-to-one). We have also shown how to use Groebner bases to calculate whether a 2-D CC has a polynomial inverse, and how to derive that inverse. As a side result, we have shown that 2-D CCs are a subset of the larger family of quasi-cyclic codes.

Following our discussion of algebra, we set out to search for powerful codes in the family of 2-D TBCCs. We found an efficient method to find the distance spectrum of 2D-CC, by extending the BEAST algorithm so it could be applied to 2-D CCs. Our search has shown that from a WER perspective, 2-D TBCCs are comparable or slightly better in their performance to 1-D TBCCs of the same constraint size. We have also shown that from a BER perspective, the codes have the potential to exceed 1-D TBCCs in their performance, over similar block sizes. For short information lengths, we have shown that 2-D TBCCs are superior to short LDPCs of the same size.

Next we tackled the problem of soft decoding of 2-D TBCCs. We started by converting the problem to a 1-D CC with non-binary input, so that the conventional Viterbi and BCJR algorithms could be applied. However, since this results in a decoding algorithm whose complexity is exponential with the size of the information sequence, we continued to search for more solutions. Thus, we have constructed a 2-D extension of the Viterbi / BCJR algorithm, dubbed the 2-D Trellis algorithm, that performed reasonably well for our test codes, albeit with some limitations. First, since the 2-D Trellis algorithm does not divide the received sequence into conditionally independent sections, it does not achieve the maximum likelihood solution. For the codes we have tested, the 2-D algorithm performed at 0.0 dB to 2.0 dB away from the optimum, depending on the specific code. Second, the 2-D Trellis is not suitable for the decoding of all 2-D TBCCs. We have formulated a necessary condition for the successful application of the algorithm, albeit a sufficient condition is still missing. Despite these shortcomings, the 2-D Trellis algorithm has the advantage of having a complexity that is linear with the input size, and exponential with the generators’ constraint size, as is the case for the conventional 1-D Viterbi. Furthermore, for many codes we have tested this algorithm showed the best performance of all sub-optimal algorithms, in some cases coming very close to the true ML solution.

Finally, in the last chapter we examined decoding based on the codes parity check matrix. As a baseline, we have used the loopy belief propagation (LBP), commonly used to decode LDPCs. Next we have applied a novel GBP decoder to try and improve upon the basic LBP decoder. We have found that for best results, we needed to modify the original GBP constructions, resulting in
a novel GBP construction. Results from this decoder were mixed. In several cases where the LBP decoder failed completely, the GBP decoder was able to successfully operate with reasonable degradation. On the other hand, there were cases where the GBP decoder failed to significantly improve on the LBP decoder, or even performed slightly worse.

In order to achieve the above results, a variety of tools was used, including: Writing scripts for algebraic computation using CoCoA, writing a full C++ simulation to simulate the various belief propagation algorithms and code searches, and using MATLAB to analyze the results.

8.2 Topics for Further Research

As noted in the introduction, 2-D CCs in general and 2-D TBCCs in particular are still a little-researched branch of coding theory. Currently, a practical application for these codes is lacking. As previous authors have noted, the 2-D nature of these codes may make them suitable for encoding 2-D information.

In this dissertation we have connected 2-D TBCCs with quasi-cyclic codes. Further research needs to be carried out to investigate how concepts from quasi-cyclic code theory can be applied to 2-D TBCCs. We have also briefly touched on the subject of convolutional product codes, and have shown them to be a particular case of 2-D TBCCs. From our few experiments in convolutional product codes, it seems that their performance is degraded relative to true 2-D TBCCs. Further research needs to be carried out to realize if this is indeed the case, and to establish the theory behind these codes.

Finally, a relationship has been suggested with 2-D cyclic codes (a 2-D cyclic code is a code where a shift of the 2-D codeword in any dimension is also a codeword). It is clear that if the codewords of a 2-D TBCC with rate $k/n$ are taken to consist of $n$-tuples of bits, that any cyclic shift of the codeword in $n$-tuples, is also a codeword. However, this requires the definition of the code as a mapping from $\text{GF}(2^k)$ to $\text{GF}(2^n)$, which is not the conventional definition of a cyclic code. A possible direction of research is to establish such a theoretical representation that will allow $m$-D TBCCs to be treated as cyclic codes.

In the context of code search, we have limited ourselves to codes of rate $\frac{1}{2}$, over small information sequences. Better codes may be found by searching over the more general codes of rate $k/n$, or by increasing the dimension. Another line of research lies in the direction of recursive codes, which are described by rational functions, rather than polynomials. The algebraic theory for these codes needs to be established, and since we have seen that division in multi-variate polynomials is non-trivial, the generalization from 1-D recursive codes can be quite interesting.

On the subject of belief propagation decoding, there are several possibilities of extending the research. The 2-D trellis algorithm from section 6.2 needs to be further investigated in order to find a necessary and sufficient condition for its convergence. Also, there may be more equivalent 2-D trellis graphs that may lead to a lower complexity version of the algorithm.

The various belief propagation networks suggested in section 7.1.3 may also be investigated more thoroughly for their decoding capabilities. Finally, the GBP algorithm itself may be refined by finding better region graph constructions and lower complexity message passing techniques, hopefully bringing it closer to the ML solution.
Appendix A. Proofs for GBP algorithms

In this section we give proofs of the equivalence of Loopy Belief Propagation (LBP) with the minimization of the free energy under the Bethe approximation, and the equivalence of Generalized Belief Propagation with the minimization of the free energy under the Kikuchi approximation.

A.1 Bethe Approximation

**Theorem:** The interior stationary points of the Bethe free energy are the same as the fixed points of standard LBP algorithm defined by equations (5.2.25), (5.2.26), and (5.2.27).

**Proof:** Recall the variational average energy and variational entropy of the Bethe approximation:

\[
U_{\text{Bethe}} (b) = - \sum_{a} \sum_{x_a} b_a (x_a) \ln f_a (x_a) \tag{A.1.1}
\]

\[
H_{\text{Bethe}} (b) = - \sum_{a=1}^{M} \sum_{x_a} b_a (x_a) \ln b_a (x_a) - \sum_{i=1}^{N} (1-d_i) \sum_{x_i} b_i (x_i) \ln b_i (x_i) \tag{A.1.2}
\]

We wish to minimize the Bethe free energy \( F_{\text{Bethe}} (b) \),

\[
F_{\text{Bethe}} (b) = U_{\text{Bethe}} (b) - H_{\text{Bethe}} (b) =
\]

\[
= - \sum_{a} \sum_{x_a} b_a (x_a) \ln f_a (x_a) + \sum_{a=1}^{M} \sum_{x_a} b_a (x_a) \ln b_a (x_a) + \sum_{i=1}^{N} (1-d_i) \sum_{x_i} b_i (x_i) \ln b_i (x_i) \tag{A.1.3}
\]

Subject to the marginalization constraints:

\[
\sum_{x_a \supset x_i} b_a (x_a) = b_i (x_i) \tag{A.1.4}
\]

And the normalization constraints:

\[
\sum_{x_a} b_a (x_a) = 1 ; \sum_{x_i} b_i (x_i) = 1 \tag{A.1.5}
\]

From equations (A.1.3), (A.1.4), we form the Langragian:
\[ L_{\text{Bethe}}(b) = F_{\text{Bethe}}(b) - \sum_a \sum_i \lambda_{ai}(x_i) \left[ \sum_{x_{i \setminus i}} b_a(x_a) - b_i(x_i) \right] \]  
(A.1.6)

Differentiating (A.1.6) with respect to some factor belief \( b_a(x_a) \) we get:
\[ \frac{\partial L_{\text{Bethe}}(b)}{\partial b_a(x_a)} = -\ln f_a(x_a) + \ln b_a(x_a) + 1 - \sum_{i \in N(a)} \lambda_{ai}(x_i) = 0 \]  
(A.1.7)

Therefore,
\[ b_a(x_a) \propto f_a(x_a) \prod_{i \in N(a)} \exp\{\lambda_{ai}(x_i)\} \]  
(A.1.8)

Similarly, differentiating with respect to some variable belief \( b_i(x_i) \),
\[ \frac{\partial L_{\text{Bethe}}(b)}{\partial b_i(x_i)} = (1 - d_i)[\ln b_i(x_i) + 1] + \sum_{a \in N(i)} \lambda_{ai}(x_i) = 0 \]  
(A.1.9)

Solving for \( b_i(x_i) \) this yields:
\[ b_i(x_i) \propto \left[ \prod_{a \in N(i)} \exp\{\lambda_{ai}(x_i)\} \right]^{1/d_i-1} \]  
(A.1.10)

Now we set:
\[ \lambda_{ai}(x_i) = \ln n_{i \rightarrow a}(x_i) = \ln \prod_{a \in N(i) \setminus a} m_{a \rightarrow i}(x_i) \]  
(A.1.11)

The variable-to-factor message equation follows immediately from this definition:
\[ n_{i \rightarrow a}(x_i) = \prod_{a \in N(i) \setminus a} m_{a \rightarrow i}(x_i) \]  
(A.1.12)

Which is the first message passing equation, (5.2.25).

If we substitute this in (A.1.10) we get:
\[
\begin{align*}
    b_i(x_i) & \propto \left[ \prod_{u \in \mathcal{N}(i)} n_{i \rightarrow u}(x_i) \right]^{1/d_i - 1} = \left[ \prod_{u \in \mathcal{N}(i)} \prod_{u' \in \mathcal{N}(i) \cup \mathcal{V}} m_{u' \rightarrow i}(x_i) \right]^{1/d_i - 1} = \ldots \\
    \ldots & = \left[ \prod_{u \in \mathcal{N}(i)} m_{a \rightarrow i}(x_j) \right]^{1/d_i - 1} = \prod_{a \in \mathcal{N}(i)} m_{a \rightarrow i}(x_j)
\end{align*}
\] 

Which is the beliefs equation, (5.2.27).

On the other hand, taking substituting (A.1.8) in (A.1.4) we get:

\[
    b_i(x_i) = \sum_{x_a, x_i} b_a(x_a) \propto \sum_{x_a} f_a(x_a) \prod_{j \in \mathcal{N}(a)} \exp \{ \lambda_{ij}(x_j) \} = \sum_{x_a} f_a(x_a) \prod_{j \in \mathcal{N}(a)} n_{j \rightarrow a}(x_j)
\] 

(A.1.14)

Taking (A.1.13) and (A.1.14) together we have:

\[
    \prod_{a \in \mathcal{N}(i)} m_{a \rightarrow i}(x_i) \propto \sum_{x_a} f_a(x_a) \prod_{j \in \mathcal{N}(a)} n_{j \rightarrow a}(x_j)
\] 

(A.1.15)

Now, substituting (A.1.12) on the left hand side we have,

\[
    m_{a \rightarrow i}(x_i) \cdot n_{i \rightarrow a}(x_i) \propto \sum_{x_a} f_a(x_a) \prod_{j \in \mathcal{N}(a)} n_{j \rightarrow a}(x_j)
\] 

(A.1.16)

Which finally yields the factor-to-variable message passing equation (5.2.26),

\[
    m_{a \rightarrow i}(x_i) \propto \sum_{x_a} f_a(x_a) \prod_{j \in \mathcal{N}(a) \cup \mathcal{V}} n_{j \rightarrow a}(x_j)
\] 

(A.1.17)

This completes the proof.

### A.2 Kikuchi Approximation

**Theorem:** The interior stationary points of the region graph free energy are the same as the fixed points of two way GBP (defined by the message and belief equations given above) that have strictly positive beliefs.

**Proof:** Recall the variational average energy and variational entropy of the Kikuchi approximation:

\[
    U_{\text{Kikuchi}}(b) = -\sum_R e_R \sum_{x_a} b_a(x_a) \sum_{a \in R} \ln f_a(x_a)
\] 

(A.2.1)
\[ H_{Kikuchi}(b) = -\sum_{R} c_{R} \sum_{x_{R}} b_{R}(x_{R}) \ln b_{R}(x_{R}) \]  \hspace{1cm} (A.2.2)

Where \( c_{R} \) are the region count numbers which satisfy the condition:
\[ c_{R} = 1 - \sum_{S \in S(R)} c_{S} \]  \hspace{1cm} (A.2.3)

The Kikuchi free energy is given as:
\[ F_{Kikuchi}(b) = U_{Kikuchi}(b) - H_{Kikuchi}(b) = \]
\[ = -\sum_{R} c_{R} \sum_{x_{R}} b_{R}(x_{R}) \sum_{a \in F_{R}} \ln f_{a}(x_{a}) + \sum_{R} c_{R} \sum_{x_{R}} b_{R}(x_{R}) \ln b_{R}(x_{R}) \]  \hspace{1cm} (A.2.4)

The beliefs are subject marginalization constraints which ensure the consistency of a region’s beliefs with the beliefs of its parents and its children:
\[ \sum_{x_{R} \neq x_{P}} b_{R}(x_{R}) = b_{R}(x_{R}) ; \sum_{x_{R} \neq x_{C}} b_{R}(x_{R}) = b_{C}(x_{C}) \]  \hspace{1cm} (A.2.5)

There are also normalization constraints which ensure that the beliefs of each region sum to 1, but we again ignore this constraints, and assume that beliefs will be kept normalized at each step of the algorithm.

From equations (A.2.4) and (A.2.5) we form the Langragian:
\[ L_{Kikuchi}(b) = F_{Kikuchi}(b) - \sum_{R} \sum_{P \in F_{R}} \lambda_{PR}(x_{R}) \left( \sum_{x_{P \neq x_{R}}} b_{P}(x_{P}) - b_{R}(x_{R}) \right) - \]
\[ - \sum_{R} \sum_{C \in C(R)} \lambda_{RC}(x_{C}) \left( \sum_{x_{C \neq x_{R}}} b_{R}(x_{R}) - b_{C}(x_{C}) \right) \]  \hspace{1cm} (A.2.6)

To minimize the Langragian, we differentiate with respect to \( b_{R}(x_{R}) \) and set the derivative to zero:
\[ \frac{\partial L_{Kikuchi}(b)}{\partial b_{R}(x_{R})} = -c_{R} \sum_{a \in F_{R}} \ln f_{a}(x_{a}) + c_{R} \left( 1 + \ln b_{R}(x_{R}) \right) - \sum_{C \in C(R)} \lambda_{RC}(x_{C}) + \sum_{P \in F_{R}} \lambda_{PR}(x_{R}) = 0 \]  \hspace{1cm} (A.2.7)

Isolating the term containing \( b_{R}(x_{R}) \) we get,
\[ c_R \ln b_R(x_R) = -c_R + c_R \sum_{a \in F_R} \ln f_a(x_a) + \sum_{C \in C(R)} \lambda_{RC}(x_C) - \sum_{P \in P(R)} \lambda_{RP}(x_R) = 0 \]  
(A.2.8)

And exponentiating this equation

\[ b_R(x_R)^{\gamma_R} \propto \left( \prod_{a \in F_R} f_a(x_a) \right)^{\gamma_R} \left( \prod_{C \in C(R)} e^{\lambda_{RC}(x_C)} \right) \left( \prod_{P \in P(R)} e^{-\lambda_{RP}(x_R)} \right) \]  
(A.2.9)

For convenience of notation, we group all the factors in a region to a single function:

\[ \tilde{f}_R(x_R) = \left( \prod_{a \in F_R} f_a(x_a) \right)^{\gamma_R} \]  
(A.2.10)

We now set the messages from a region to its parents to be:

\[ n_{R \to P}(x_R) = e^{\lambda_{RP}(x_R)} \]  
(A.2.11)

And the messages from a parent to its children:

\[ m_{P \to R}(x_R) = b_R(x_R)^{\gamma_P} e^{-\lambda_{RP}(x_R)} \]  
(A.2.12)

Substituting (A.2.11) and (A.2.12) in (A.2.9) we have,

\[ b_R(x_R)^{\gamma_R} \propto \tilde{f}_R(x_R) \cdot \left( \prod_{C \in C(R)} n_{C \to R}(x_C) \right) \cdot \left( \prod_{P \in P(R)} m_{P \to R}(x_R) \cdot b_R(x_R)^{-\gamma_P} \right) \]

\[ \propto b_R(x_R)^{-\gamma_P} \cdot \tilde{f}_R(x_R) \cdot \left( \prod_{C \in C(R)} n_{C \to R}(x_C) \right) \cdot \left( \prod_{P \in P(R)} m_{P \to R}(x_R) \right) \]  
(A.2.13)

\[ \propto b_R(x_R)^{\gamma_{-1}} \cdot \tilde{f}_R(x_R) \cdot \left( \prod_{C \in C(R)} n_{C \to R}(x_C) \right) \cdot \left( \prod_{P \in P(R)} m_{P \to R}(x_R) \right) \]

Which finally yields the beliefs equation,

\[ b_R(x_R) \propto \tilde{f}_R(x_R) \cdot \left( \prod_{C \in C(R)} n_{C \to R}(x_C) \right) \cdot \left( \prod_{P \in P(R)} m_{P \to R}(x_R) \right) \]  
(A.2.14)

Which is the first equation of the GBP algorithm.

We now define pseudo-messages between regions and their parents in a manner analogous to the standard BP case:
\[ n^0_{R \rightarrow p}(x_R) = \frac{b_R(x_R)}{m^0_{p \rightarrow R}(x_R)} ; \quad m^0_{p \rightarrow R}(x_R) = \sum_{x_p \in \mathcal{X}_p} \frac{b_p(x_p)}{n^0_{R \rightarrow p}(x_R)} \]  

(A.2.15)

Substituting this in the marginalization constraint (A.2.5) we get,

\[ n^0_{R \rightarrow p}(x_R)m^0_{p \rightarrow R}(x_R) = b_R(x_R) = \sum_{x_p \in \mathcal{X}_p} b_p(x_p) = n^0_{R \rightarrow p}(x_R)m^0_{p \rightarrow R}(x_R) \]

(A.2.16)

From which we can isolate the messages from a parent to a child region:

\[ m_{p \rightarrow R}(x_R) = \frac{n^0_{R \rightarrow p}(x_R)m^0_{p \rightarrow R}(x_R)}{n^0_{R \rightarrow p}(x_R)} \]

(A.2.17)

On the other hand, substituting (A.2.11) in (A.2.12) we have,

\[ m_{p \rightarrow R}(x_R)n_{R \rightarrow p}(x_R) = b_R(x_R) = \left(n^0_{R \rightarrow p}(x_R)m^0_{p \rightarrow R}(x_R)\right)^{q_e} \]

(A.2.18)

From which we isolate the messages from a region to its parent:

\[ n_{R \rightarrow p}(x_R) = n^0_{R \rightarrow p}(x_R)^{q_e} m^0_{p \rightarrow R}(x_R)^{q_e^{-1}} \]

(A.2.19)

Substituting (A.2.17) in (A.2.19) we have,

\[ n_{R \rightarrow p}(x_R)^{2-q_e} = n^0_{R \rightarrow p}(x_R)^{q_e} m^0_{p \rightarrow R}(x_R)^{q_e^{-1}} \]

(A.2.20)

Which finally yields the first message passing equation,

\[ n_{R \rightarrow p}(x_R) = \left(n^0_{R \rightarrow p}(x_R)^{q_e} m^0_{p \rightarrow R}(x_R)^{q_e^{-1}}\right)^{q_e} \]

(A.2.21)

Substituting this in (A.2.17) we get the second message passing equation,

\[ m_{p \rightarrow R}(x_R) = \left(n^0_{R \rightarrow p}(x_R)^{q_e^{-1}} m^0_{p \rightarrow R}(x_R)^{q_e}\right)^{q_e} \]

(A.2.22)

Which completes the proof.
Bibliography


קוד קונגמדיזה ציקליי דו-מימדית

מאת
ליעם אלפנדיר

בהנחיית
ד"ר דני רפאלי
The concept of convolutional codes is a generalization of the traditional convolutional codes. A convolutional code is a linear code where the encoded data sequence is produced by passing the input data sequence through a linear filter. The convolutional codes are defined by a set of linear equations that map the input sequence to the output sequence. The convolutional codes are widely used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.

The convolutional codes can be represented as a set of linear equations that describe the relationships between the input and output sequences. The convolutional codes are also used in various applications, such as digital communications and error correction.
ה理解和 השפה רחבה יותר

PC-LBP

5.2

 GBP - מсхית

61 ..

اتفות אתונרות

6.1

62 ..

אינפונות

6.1.1

63 ..

כיסים

6.1.2

64 ..

כיסים

6.2

65 ..

כיסים

6.2.1

66 ..

כיסים

6.2.2

67 ..

כיסים

6.2.3

68 ..

כיסים

6.3

69 ..

כיסים

6.3.1

70 ..

כיסים

6.3.2

71 ..

כיסים

6.3.3

72 ..

כיסים

7.1

73 ..

כיסים

7.1.1

74 ..

כיסים

7.1.2

75 ..

כיסים

7.1.3

76 ..

כיסים

7.2

77 ..

כיסים

7.2.1

78 ..

כיסים

7.2.2

79 ..

כיסים

7.2.3

80 ..

כיסים

7.3

81 ..

כיסים

8.1

82 ..

כיסים

8.2

83 ..

כיסים

A.1

84 ..

כיסים

A.2

85 ..

בליון

A.3

86 ..

כיסים

A.4

87 ..

כיסים

A.5

88 ..

כיסים

A.6

89 ..

כיסים

A.7

90 ..

כיסים

A.8

91 ..

כיסים

A.9

92 ..

כיסים

A.10

93 ..

כיסים

A.11

94 ..

כיסים

A.12

95 ..

כיסים

A.13

96 ..

כיסים

A.14

97 ..

כיסים

A.15

98 ..

כיסים

A.16

99 ..

כיסים

A.17

100 ..

כיסים

A.18

101 ..

כיסים

A.19

102 ..

כיסים

A.20

103 ..

כיסים

A.21

104 ..

כיסים

A.22

105 ..

כיסים

A.23

106 ..

כיסים

A.24

107 ..

כיסים

A.25

108 ..

כיסים

A.26

109 ..

כיסים

A.27

110 ..

כיסים

A.28

111 ..

כיסים

A.29

112 ..

כיסים

A.30

113 ..

כיסים

A.31

114 ..

כיסים

A.32

115 ..

כיסים

A.33

116 ..

כיסים

A.34

117 ..

כיסים

A.35

118 ..

כיסים

A.36

119 ..

כיסים

A.37

120 ..

כיסים

A.38

121 ..

כיסים

A.39

122 ..

כיסים

A.40

123 ..

כיסים

A.41

124 ..

כיסים

A.42

125 ..

כיסים

A.43

126 ..

כיסים

A.44

127 ..

כיסים

A.45

128 ..

כיסים

A.46

129 ..

כיסים

A.47

130 ..

כיסים

A.48

131 ..

כיסים

A.49

132 ..

כיסים

A.50

133 ..

כיסים

A.51

134 ..

כיסים

A.52

135 ..

כיסים

A.53

136 ..

כיסים

A.54

137 ..

כיסים

A.55

138 ..

כיסים

A.56

139 ..

כיסים

A.57

140 ..

כיסים

A.58

141 ..

כיסים

A.59

142 ..

כיסים

A.60