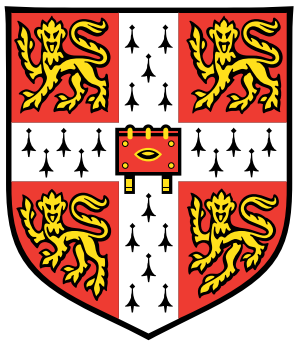
Error-correction using Low-Density Parity-Check Codes

Matthew C. Davey
Gonville and Caius College
Cambridge

A dissertation submitted in candidature for the degree of Doctor of Philosophy,
University of Cambridge

Inference Group
Cavendish Laboratory
University of Cambridge



December 1999

Declaration

I hereby declare that my dissertation entitled "Error-correction using Low-Density Parity-Check Codes" is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other University.

I further state that no part of my dissertation has already been or is being concurrently submitted for any such degree or diploma or other qualification.

Except where explicit reference is made to the work of others, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. This dissertation does not exceed sixty thousand words in length.

Date: ..... Signed: .....

Matthew C. Davey
Gonville and Caius College
Cambridge
December, 1999

Abstract

Gallager's low-density parity-check codes are defined by sparse parity-check matrices, usually with a random contruction. Such codes have near Shannon limit performance when decoded using an iterative probabilistic decoding algorithm. We report two advances that further improve the error-correction performance of these codes. First, defining the codes over non-binary fields we can obtain a 0.6 dB improvement in signal to noise ratio for a given bit error rate. Second, using irregular parity-check matrices with non-uniform row and column weights we obtain gains of up to 0.5 dB. The empirical error-correction performance of irregular low-density parity-check codes is unbeaten for the additive white Gaussian noise channel.

Low-density parity-check codes are also shown to be useful for communicating over channels which make insertions and deletions as well as additive (substitution) errors. Error-correction for such channels has not been widely studied, but is of importance whenever synchronisation of sender and receiver is imperfect. We introduce concatenated codes using novel non-linear inner codes which we call 'watermark' codes, and LDPC codes over non-binary fields as outer codes. The inner code allows resynchronisation using a probabilistic decoder, providing soft outputs for the outer LDPC decoder. Error-correction performance using watermark codes is several orders of magnitude better than any comparable results in the literature.

Acknowledgements

During my three short years in Cambridge I have often benefited from the expertise of others. At various points I have had cause to be thankful to Johannes Blocher, John Bridle, Sae-Young Chung, Hendrik Ferreira, Dave Forney, Brendan Frey, George Hart, Ralf Kötter, Bob McEliece, James Miskin, Radford Neal, David Neuhoff, Tom Richardson, Dan Spielman, David Ward and Simon Wilson. Thank you to Prof. John Lewis of DIAS for introducing me to Information Theory and to Sanjoy Mahajan for constructive criticism during the writing of previous work

My stay at Cambridge has been supported by the Engineering and Physical Sciences Research Council (EPSRC), the George and Lillian Schiff Foundation, the Robert Gardiner memorial scholarship and the Cavendish laboratory. My work was also funded by the Gatsby charitable foundation through their support of the Inference Group. The IEEE and Gonville and Caius college supported my attendance at relevant conferences. Thank you all.

I consider myself privileged to have been supervised by David MacKay. His formidable insight and unfailing enthusiasm have proved a fertile source of stimulation and encouragement. He has always been generous with his time, listening carefully and criticising fairly. I could spend another few years following up the ideas he has passed my way.

Of course, life at Cambridge has not entirely revolved around the Cavendish laboratory. Thanks to the 1 Strange road crew (including hangers on and alumni) for salsa football, good coffee and a great planetarium. Thanks to Jeremy and Sheila, Jim, Bernard, John, Ed, Andy, Emily and all the others in the CUCrC windsurfing section for some all-important quality time.

This thesis is dedicated to my father, my mother and Hélène, whose importance to me I shall not try to put into words.

Contents

Chapter 1 Summary **1**

1.1 Improved error-correction 2
1.2 Error-correction coding 4
1.2.1 Random block codes 4
1.2.2 Textbook codes 5
1.2.3 Practical near Shannon limit error-correction 6
1.3 Outline of thesis 7

Chapter 2 Low-Density Parity-Check Codes **8**

2.1 Background 8
2.2 Construction of regular Gallager codes 10
2.3 Decoding 10
2.3.1 Initialisation 12
2.3.2 Updating R_{ij}^a 12
2.3.3 Updating Q_{ij}^a 13
2.3.4 Tentative decoding 13
2.4 Constructing LDPC matrices 14
2.4.1 Gallager's prescription 14
2.4.2 MacKay's constructions 14
2.4.3 Ultra-light matrices 15
2.4.4 Irregular matrices 17

Chapter 3 Non-Binary Low-Density Parity-Check Codes **18**

3.1 Decoding over $GF(q)$ 19
3.2 Initial experiments with non-binary LDPC codes 20
3.2.1 Description of experiments 20
3.2.2 Confusing results 21
3.3 Monte-Carlo analysis 22
3.3.1 Method 22
3.3.2 MC results 23
3.4 Fourier transform decoding 25

3.5 Regular LDPC code results 26

Chapter 4 Irregular Low-Density Parity-Check Codes 29

4.1 Finding good irregular codes over $GF(q)$ 30
4.1.1 Cost functions 30
4.1.2 Search method 32
4.1.3 Alternative search methods 32
4.2 Results 33
4.2.1 Binary irregular LDPC codes 33
4.2.2 Non-binary irregular LDPC codes 33
4.3 Unequal error protection 35

Chapter 5 Behaviour of the LDPC decoding algorithm 37

5.1 Required iterations for decoding 37
5.2 Decoding failures 39
5.2.1 Confidence of tentative decoding 39
5.2.2 Convergence and oscillation 39
5.2.3 Evidence of chaos 42

Chapter 6 Codes for correcting synchronisation errors 45

6.1 Synchronisation errors 45
6.2 Codes for synchronisation 46
6.2.1 Run-length limited codes 46
6.2.2 Slip-correcting codes 47
6.2.3 Insertion and deletion correcting codes 48
6.3 Probabilistic frameworks 49

Chapter 7 Watermark codes 51

7.1 Outline of watermark codes for synchronisation 51
7.2 Channel model 52
7.3 Construction 54
7.4 Decoding 55
7.4.1 Hidden Markov Model 56
7.4.2 Synchronisation 57
7.4.3 LDPC decoding 58
7.4.4 Multiple blocks 58
7.5 Limits of watermark codes 58
7.5.1 Limits of synchronisation 59
7.5.2 Probabilistic resynchronisation 60
7.5.3 Limits of decoding 64
7.6 Experiments and results 68

7.6.1 Simulation method . . . 68
7.6.2 Choice of outer code rate . . . 70
7.6.3 High-rate watermark codes . . . 70
7.6.4 Lower rate watermark codes . . . 71
7.7 Complexity and implementation considerations . . . 73
7.7.1 Faster decoding . . . 73
7.7.2 How sparse is a sparse vector? . . . 74
7.7.3 Choice of watermark vector . . . 76
7.8 Future directions . . . 77

Chapter 8 Conclusions **79**

Appendix A Linear Block Codes **82**
A.1 Generator matrix . . . 83
A.2 Parity-check matrix . . . 83

Appendix B Arithmetic in a Finite Field **84**

Appendix C Channel Models **85**
C.1 Additive white Gaussian noise channel . . . 85
C.2 Binary symmetric channel . . . 87
C.3 q-ary symmetric channel . . . 87
C.4 Simple Insertion/Deletion Channel . . . 88

Appendix D Hidden Markov Models **90**
D.1 Markov Chains . . . 90
D.2 Hidden Markov Models . . . 91
D.2.1 Definitions . . . 91
D.2.2 Problems . . . 91
D.2.3 Solutions . . . 92
D.2.4 Notes . . . 93

Appendix E Viterbi decoding of watermark codes **94**

Appendix F Tables of LDPC codes **96**
F.1 Codes in figure 1.1 . . . 96
F.2 Codes in figure 4.2 . . . 98
F.3 Codes in figure 4.3 . . . 98
F.4 Code in figure 5.1 . . . 98
F.5 Regular Codes . . . 98
F.6 Miscellaneous . . . 99

Chapter 1

Summary

In teh bginning thwre was the word, anjd rhe werd wa 'Aardvark'.

In reading the above sentence you have performed error-correction. The above sentence has been corrupted by five substitution errors, two deletion errors and one insertion error. You were able to perform error-correction because the English language contains much redundancy. The redundancy of natural languages allows them to be compressed very effectively – text compression works by removing redundancy, but error-correction relies on its presence.

With digital communication, it is less clear how to detect and correct errors. How could you tell if the message "0011101001010" had been corrupted by errors? The solution is to artificially add a small amount of redundancy to a (typically compressed) message. If errors occur during transmission then the added redundancy can be used to detect and correct the errors. The goal of error-correction coding is to maximise the typical number of errors that can be corrected, while keeping the added redundancy to a minimum.

Almost all known error-correcting codes are designed to correct *substitution* errors: errors in which a transmitted character is received as a different character ("there" becomes "thwre"). In this thesis we present significant improvements of one class of error-correcting codes known as low-density parity-check (LDPC) codes, and produce performance close to the theoretical limit for the most widely studied communication channel.

The opening message was also corrupted by *insertion* and *deletion* errors. Such errors occur when the transmitter and receiver are not perfectly synchronised. Very few methods are known for correcting such errors, and most solutions discard some information when regaining synchronisation. We show how LDPC codes may be used to correct insertion and deletion errors, improving on known methods by several orders of magnitude.

In this chapter we review error-correcting codes, putting low-density parity-check codes in context with other known codes, and finally we give an overview of subsequent chapters. We start by summarising the improvements to LDPC codes that make it possible for their practical performance to surpass all other known codes.

1.1 Improved error-correction

The performance of error-correcting codes was bounded by Shannon in 1948 [79]. However, until the arrival of turbo codes in 1993 [6], practical coding schemes for most non-trivial channels fell far short of the Shannon limit. Turbo codes marked the beginning of near Shannon limit performance for the additive white Gaussian noise channel, spectacularly improving on all previous schemes. Two years later MacKay and Neal [61] rediscovered Gallager's long neglected low-density parity-check codes and showed that, despite their simple description, they too have excellent performance. In this section we outline our recent improvements of low-density parity-check codes that allow them to surpass the performance of turbo codes.

Low-density parity-check codes are a class of linear error-correcting block codes. The reader unfamiliar with such codes is referred to the overview in Appendix A. Introduced by Gallager in 1962 [25], LDPC codes are defined in terms of a sparse parity-check matrix: each codeword satisfies a number of linear constraints, with each symbol of the codeword participating in a small number of constraints (*e.g.* 3). Gallager provided constructions, a description of an iterative probabilistic decoding algorithm and theory that in some aspects goes beyond what is known today for turbo codes. Arriving before the computing power that was to prove their effectiveness, LDPC codes were largely forgotten until 1995 (notable exceptions to this rule include Zyablov and Pinsker [94] and Tanner [83]).

We have developed and combined two improvements to Gallager's original codes. First, we consider non-binary versions of the codes. By encoding our messages using symbols from a finite field with more than two elements, each parity-check becomes more complex but decoding remains tractable. Although the non-binary codes have an alternative representation as binary codes, the non-binary decoding algorithm is not equivalent to the binary algorithm. These changes can help if we construct the parity-check matrix carefully. Second, Gallager considered codes whose parity-check matrix had fixed row and column weights (a construction we refer to as 'regular'). We relax this constraint and produce 'irregular' LDPC codes which have a variety of row and column weights. High weight columns help the decoder to identify some errors quickly, making the remaining errors easier to correct.

Figure 1.1 compares the performance of these improved LDPC codes with state-of-the-art codes and with the original Gallager codes. All codes shown have rate 1/4, and we plot signal to noise ratio against the bit error rate. The aim is to achieve the lowest bit error rate for a given signal to noise ratio. That is, the best codes lie towards the bottom-left of the figure.

On the right is a good regular binary LDPC code, as reported by MacKay [58]. Such codes were introduced by Gallager in 1962 but their quality was not recognised until computing power allowed sufficiently long blocklength versions to be implemented. The curve labelled 'Galileo' shows the performance of a concatenated code developed at NASA's Jet Propulsion Laboratory based on a constraint length 15, rate 1/4 convolutional code (data courtesy of R.J. McEliece.). This code was developed for deep space communication and requires an extremely computer intensive decoder. Until it was eclipsed by turbo codes, it represented

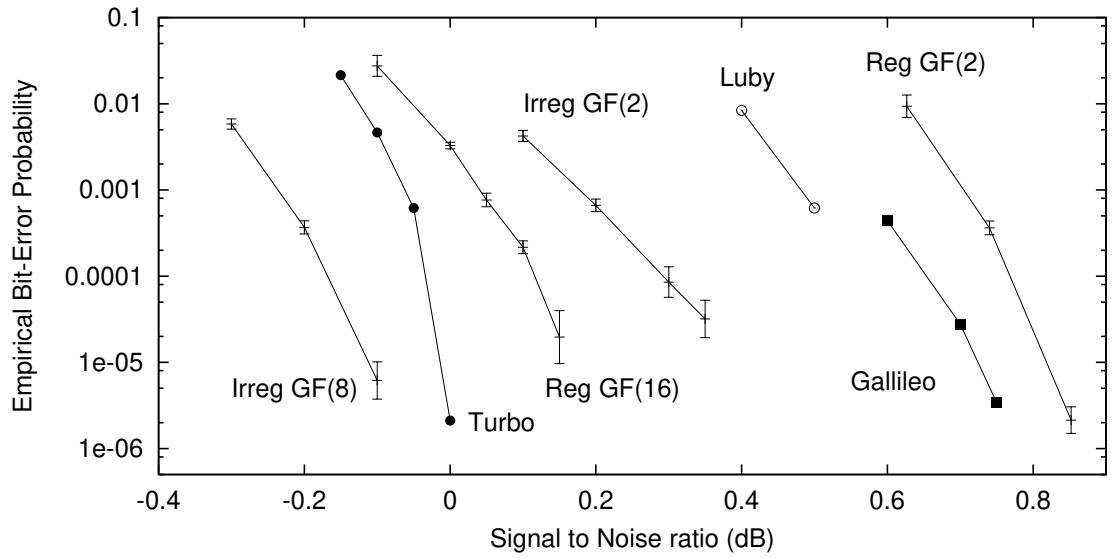


Figure 1.1: Comparison of empirical results for rate 1/4 improved low-density parity-check codes over the Gaussian Channel. From left to right: Irregular LDPC over $GF(8)$, blocklength 48000 bits; JPL turbo code [45] blocklength 65536; Regular LDPC over $GF(16)$, blocklength 24448 bits; Irregular binary LDPC, blocklength 16000 bits; Luby *et al.* [56] irregular binary LDPC, blocklength 64000 bits; JPL concatenated code based on constraint length 15, rate 1/4 convolutional code; Regular binary LDPC: blocklength 40000 bits [58]. The Shannon limit is at about -0.79 dB.

the state of the art in error-correction.

Luby, Mitzenmacher, Shokrollahi and Spielman [56] first investigated irregular constructions of LDPC codes and reported the results labelled 'Luby'. Their methods for choosing matrix parameters are not directly applicable to non-binary codes so we develop alternative construction methods. The binary irregular code labelled 'Irreg GF(2)' was constructed using our alternative methods for finding construction parameters. Although the blocklength is just 1/4 the length of the 'Luby' code, the performance is considerably better.

Regular LDPC codes defined over non-binary fields can outperform the binary irregular codes, as shown by the code labelled 'Reg GF(16)', a regular code defined over the finite field with 16 elements.

The code 'Irreg GF(8)' was constructed by combining both modifications. It beats the best known turbo codes, at least for bit error rates above 10^{-5}, making it the best error correcting code of rate 1/4 for the Gaussian channel currently known. Not only is the error-correction performance better than that of the turbo code, the blocklength is less than that of the turbo code. Another key difference between LDPC codes and turbo codes is that, empirically, all errors made by the LDPC decoding algorithm are *detected* errors. That is, the decoder reports the fact that a block has been incorrectly decoded.

Recent results by Richardson, Shokrollahi and Urbanke [74] have shown that extremely long blocklength (10^6 bits) irregular LDPC codes can perform within 0.1 dB of the Shannon limit. Empirical results were presented for rate 1/2 codes. Related work by Chung [12] using

matrices with some extremely high-weight columns (maximum column weight 1000), suggests that performance within 0.01 dB of the Shannon limit may be achievable for long blocklength rate 1/2 LDPC codes over the binary-input additive white Gaussian noise channel.

1.2 Error-correction coding

In this section we briefly review several known constructions of error-correcting codes, putting low-density parity-check codes in context.

We start by quoting a version of Shannon's noisy channel coding theorem:

Theorem 1.1 *Associated with each discrete memoryless channel is a nonnegative number C (called the channel capacity) with the following property. For any $\epsilon > 0$ and $R < C$, there exist codes of rate $\geq R$ and a decoding algorithm such that the frequency of decoding errors is less than ϵ. Error-free communication at rates above capacity is not possible.*

Shannon proved his theorem by showing that, at rates below capacity, the *average* frequency of errors made by all codes in a certain class can be made arbitrarily small. It follows that certain codes within the class must satisfy the requirements of the theorem. The codes that Shannon used in his proof were random block codes, which we now define.

1.2.1 Random block codes

Consider a channel with input alphabet $\mathcal{A}_X$ and output alphabet $\mathcal{A}_Y$. We make the following definitions:

An (N,K) block code is a mapping from $\{0,1\}^K$ to $\mathcal{A}_X^N$. A binary input message $\mathbf{x}$ of length K is mapped to a codeword $\mathbf{t} \in \mathcal{A}_X^N$ (*i.e.* of length N).

The rate of communication is $R = K/N$. K bits of information are sent in N channel uses.

A decoder is a mapping from $\mathcal{A}_Y^N$ to $\{0,1\}^K$. Received channel outputs $\mathbf{r} \in \mathcal{A}_Y^N$ are mapped to decoded codewords $\hat{\mathbf{x}} \in \mathcal{A}_X^N$.

The probability of block error of a code, given a distribution over input messages and a channel model, is:

$$p_{\mathrm{B}} = \sum_{\mathbf{x}} P(\mathbf{x}) P(\hat{\mathbf{x}} \neq \mathbf{x} | \mathbf{x}).$$

The optimal decoder is the decoder that minimises p_{B}.

Figure 1.2 summarises the operation of an (N, K) block code.

Shannon proved that for rates below capacity and any $\epsilon > 0$, there are block codes for which the probability of block error (using the optimal decoder) is less than ϵ. Furthermore,

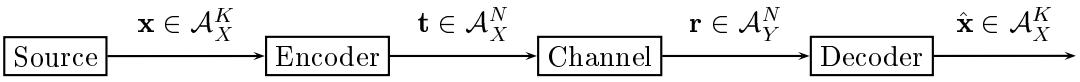


Figure 1.2: Outline of (N, K) block code.

for some channels (*e.g.* the binary symmetric channel – defined in appendix C.2) almost all sufficiently long random block codes are close to ideal.

In the discussion of his now celebrated theorem, Shannon [79] points out the practical difficulties of achieving capacity:

> An attempt to obtain a good approximation to ideal coding by following the method of the proof is generally impractical. In fact, apart from some rather trivial cases and certain limiting situations, no explicit description of a series of approximation to the ideal has been found. Probably this is no accident but is related to the difficulty of giving an explicit construction for a good approximation to a random sequence.

The search for families of codes that approach the Shannon limit and can be encoded and decoded with practical algorithms (requiring time and space polynomial in the blocklength) has been of consuming interest for information theorists.

1.2.2 Textbook codes

For several years, the codes with the best practical performance for the Gaussian channel were Reed-Solomon [72, 63] outer codes concatenated with convolutional codes [22].

Like block codes, convolutional codes map an input block of K symbols to an output block of N symbols. The output block depends not just on the inputs, but also on m previous input blocks. Thus, an output symbol can depend on up to $K(m+1)$ input symbols. The quantity $K(m+1)$ is known as the *constraint length* of the code. Constraint length 7 convolutional codes have been the *de facto* standard for satellite communication since the 1970's [53]. Convolutional codes are easily encoded.

In principle, convolutional codes can approach the Shannon limit as the constraint length increases, but the complexity of the best known (Viterbi) decoding algorithm [53] is exponential in the constraint length. Alternatively, sequential decoding of convolutional codes [53] has complexity largely independent of the constraint length, but effective error-correction is only possible up to a cutoff rate $R_0 < C$ at which point the average computational load becomes infinite. In the case of a binary symmetric channel, for example, for rate 1/2 codes to operate below the cutoff rate the noise level must be below 4.5% whereas the Shannon limit for rate 1/2 codes is 11% noise.

Reed-Solomon codes are a class of cyclic codes[1] over fields $GF(q)$ of length $q-1$, which

[1] A cyclic code is one in which all cyclic shifts of a codeword are themselves codewords.

can correct up to t errors and require not more than $2t$ parity checks. Reed-Solomon codes defined over fields $GF(2^p)$ are often used as outer codes for binary inner codes: information is first encoded using a Reed-Solomon code and the resulting codeword is encoded using a second code. Such constructions are known as *concatenated* codes [23].

The concatenated code included in figure 1.1 comprises a Reed-Solomon code concatenated with a constraint length 15 convolutional code, and requires an extremely computer intensive decoding algorithm.

1.2.3 Practical near Shannon limit error-correction

Turbo codes, introduced by Berrou *et al.* in 1993 [6], heralded the beginning of near Shannon-limit performance for the additive white Gaussian noise channel. As originally presented, the turbo encoder consists of two binary rate 1/2 convolutional encoders in *parallel*. The input to one of the encoders is a pseudo-random permutation of the input to the other. The constituent convolutional codes are *systematic*: half of the bits they produce are copies of the input bits. When used in turbo encoding, the systematic bits produced by *one* of the convolutional codes are discarded.

The decoding algorithm consists of the modified Viterbi decoding algorithm applied to each constituent code, with the output *a posteriori* estimates from one decoder being used as input to the other. Decoding consists of several iterations of this message passing algorithm.

In his PhD thesis, Wiberg [91] developed a general framework for iterative decoding based on Tanner graphs [83]. Intended as a generalisation of trellis coding and Gallager's low-density parity-check codes, Wiberg found that the turbo decoding algorithm fitted naturally into his framework. McEliece, MacKay and Cheng [65] showed that the turbo decoding algorithm is an instance of Pearl's belief propagation algorithm [69], an inference algorithm well established in the artificial intelligence community[2]. Furthermore, McEliece *et al.* pointed out that several other known iterative decoding algorithms could be derived from Pearl's algorithm.

The earliest description of an iterative decoding algorithm was by Gallager in 1962, for his low-density parity-check codes [25]. Low-density parity-check codes have a simple description and a largely random structure. MacKay [58] proved that sequences of low-density parity-check codes exist that, when decoded with an optimal decoder, approach arbitrarily close to the Shannon limit. The iterative decoding algorithm makes decoding practical and, as we shall see in later chapters, is capable of near Shannon limit performance.

Arguably simpler to describe than low-density parity-check codes are the *repeat and accumulate (RA)* codes studied by Divsalar, Jin and McEliece [18, 44]. An information block of length K is repeated q times and scrambled by a pseudo-random permutation of size Kq. The permuted block is encoded by a rate 1 accumulator: the output block $[y_1, \ldots, y_N]$ is related to the input block $[x_1, \ldots, x_N]$ by the formula $y_i := \sum_{j=1}^{i} x_i \bmod 2$. The code may be

[2] Pearl's algorithm is a graph-based message passing algorithm which caculates exact *a posteriori* probabilities when applied to tree-like graphs. In the coding applications we consider, the relevant graphs have many loops but we find that Pearl's algorithm performs very well.

decoded using a belief propagation decoder.

Despite the fact that the repetition code is virtually useless on its own, the random permutation creates surprisingly good codes: as the rate of RA codes approaches zero and the blocklength is increased, the average signal to noise level required for arbitrarily small error probability with an optimal decoder approaches the Shannon limit. Practical performance less than 1 dB from capacity has been reported [18].

Low-density parity-check codes, repeat-accumulate codes, and turbo codes have several features in common. All have a strong pseudo-random element in their construction. All may be decoded using an iterative belief propagation algorithm. All have been shown to achieve near Shannon limit error-correction performance.

1.3 Outline of thesis

The following three chapters describe Gallager's original low-density parity-check codes and the improvements that enable record-breaking performance. In chapter 2 we give a brief review of low-density parity-check codes. Gallager's codes attracted little attention prior to 1995, but there has been a recent surge of interest since their performance was recognised. We describe the iterative decoding algorithm and several methods of constructing suitable low-density matrices. In chapters 3 and 4 we develop non-binary and irregular versions of Gallager's codes, respectively. We show that to obtain the best performance we must choose our construction parameters carefully. Monte Carlo methods are used to simulate infinite blocklength codes, providing results which help in choosing good parameters for finite blocklength codes. In chapter 4 we construct binary irregular LDPC codes that improve on Luby *et al.*'s results.

The decoding algorithm is known to be sub-optimal. Although the practical performance is very good, in general it is not guaranteed to converge for all initial conditions (*i.e.* received blocks). Loops in the graph on which belief propagation messages are passed induce feedback into the decoding algorithm. In chapter 5 we present evidence that such feedback can lead to chaotic dynamics in the decoding algorithm. We also analyse the number of iterations required for decoding.

Chapters 6 and 7 consider the problem of error-correction for channels that cause insertions and deletions. We first review previous work in this area, which has focused largely on number-theoretic constructions for the correction of a small number of errors. We introduce a novel concatenated coding scheme which uses a non-binary low-density parity-check code as the outer code, and a simple non-linear code as the inner code. The inner code, which we call a 'watermark code' provides protection against insertion and deletion errors. Remaining (substitution) errors are corrected by the outer decoder. Few practical codes have been developed for the general insertion/deletion channel that we consider. Watermark codes correct several orders of magnitude more errors than comparable codes in the literature.

Finally, chapter 8 contains our conclusions.

Chapter 2

Low-Density Parity-Check Codes

What most experimenters take for granted before they begin their experiments is infinitely more interesting than any results to which their experiments lead.

— Norbert Wiener

2.1 Background

Low-density parity-check codes are a class of linear error-correcting codes. Linear codes use a generator matrix $\mathbf{G}$ to map messages $\mathbf{s}$ to transmitted blocks $\mathbf{t}$, also known as *codewords*. They have an equivalent description in terms of a related parity-check matrix $\mathbf{H}$. All codewords satisfy $\mathbf{Ht} = 0$. Appendix A contains an introduction to linear block codes.

As their name suggests, low-density parity-check codes are defined in terms of parity-check matrices $\mathbf{H}$ that consist almost entirely of zeroes. Gallager defined (n, p, q) LDPC codes to have a blocklength n and a parity-check matrix with exactly p ones per column and q ones per row, where $p \geq 3$. Figure 2.1 shows a (20,3,4) code constructed by Gallager [25]. If all the rows are linearly independent then the rate of the code is $(q - p)/q$, otherwise the rate is $(n - p')/n$ where p' is the dimension of the row space of $\mathbf{H}$.

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

Figure 2.1: Example of a parity-check matrix for a $(20, 3, 4)$ LDPC code.

In Gallager's construction in figure 2.1, the lower two sections are column permutations

of the upper section. By considering the ensemble of all matrices formed by such column permutations, Gallager proved several important results. These include the fact that the error probability of the optimum decoder decreases exponentially for sufficiently low noise and sufficiently long blocklength, for fixed p. Also, the typical minimum distance[1] increases linearly with the blocklength.

Gallager's codes, introduced in 1962, made little impact on the information theory community despite the fact that they would have broken all practical coding records prior to 1993. It is likely that the storage requirements of encoding, and the computational demands of decoding made their immediate adoption infeasible.

There were few citations of Gallager's work in the literature prior to 1996 (see MacKay [58, p.422] for a comprehensive list). The most significant, in terms of coding theory, are those of Zyablov and Pinsker (1975), Tanner (1981) and Wiberg (1996).

Zyablov and Pinsker [94] showed that Gallager's codes can correct a number of errors linear in the blocklength with a decoding algorithm of complexity $n \log n$. The proof only worked for codes of rather low rate, but showed that the codes could achieve vanishing probability of error at non-zero rates with a practical decoding algorithm.

Tanner [83] generalised Gallager's constructions, using bipartite graphs to represent the relationships between codeword symbols and general constraints. Gallager's codes are obtained when all constraints are binary parity checks. Tanner proposed iterative decoding using independent decoders at each constraint, a generalisation of Gallager's algorithm.

Wiberg [91] rediscovered and extended Tanner's ideas, developing generic decoding algorithms (which he called 'min-sum' and 'sum-product') for codes based on Tanner graphs. Wiberg showed that the Viterbi algorithm, and Tanner's algorithm 'B' are special cases of min-sum decoding. Also, forward-backward, turbo and Gallager algorithms are special cases of the sum-product algorithm. Wiberg carried out some performance analysis for iterative decoding, and calculated some weak bounds on the performance of low-density parity-check codes.

For a comprehensive treatment of standard low-density parity-check codes see MacKay [58]. MacKay showed that LDPC Codes are 'very good' when decoded with an optimal decoder. That is, they can be used to communicate at any rate below the Shannon limit with vanishing probability of error. Unfortunately, as is the case with most codes, optimal decoding of low-density parity-check codes is an NP-complete problem [5]. MacKay also demonstrated that Gallager's decoding algorithm has excellent empirical performance.

Davey and MacKay [15] introduced non-binary versions of Gallager's codes, and showed that significant improvements could be made. Luby, Mitzenmacher, Shokrollahi and Spielman [56] introduced parity-check matrices with highly non-uniform column-weight distributions. Such *irregular* LDPC codes were demonstrated to give improved performance. In 1998, Davey and MacKay [16] presented irregular non-binary codes which outperformed the best

[1] The *minimum distance* of a linear code is the smallest Hamming weight (number of non-zero symbols) of a non-zero codeword.

known turbo codes.

The most significant theoretical work, since MacKay's proof that LDPC codes can approach capacity with an optimal decoder, is that of Richardson, Shokrollahi and Urbanke [74]. Their work (discussed in section 4.1.3) allows the calculation of upper bounds on decoding performance. Empirical results suggest the bounds are approached as the blocklength increases, and results were presented showing performance within 0.1 dB of the Shannon limit.

The superb performance of low-density parity-check codes has sparked great interest, and the number of publications is rising quickly. Recent work includes [17, 41, 59, 60, 62, 65, 74, 80, 93].

2.2 Construction of regular Gallager codes

We relax Gallager's constraints of fixed row and column weights, using instead a mean column weight $t > 2$. The *weight* of a vector is the number of non-zero components in that vector. The parity-check matrix $\mathbf{H}$ is constructed randomly, while constraining the distributions of row and column weights to be as uniform as possible. We also require that no two columns of $\mathbf{H}$ have an overlap greater than 1 (where we define the overlap of two binary vectors $\mathbf{x}$ and $\mathbf{y}$ to be the number of coordinates for which $x_i = y_i$) to reduce the probability of low weight codewords. For details of the construction rules, see section 2.4.

Since $\mathbf{H}$ is not in systematic form (see Appendix A) we perform Gaussian elimination using row operations and reordering of columns to derive a parity-check matrix $\mathbf{H}' = [-\mathbf{P}|\mathbf{I}_m]$ where the notation $[\mathbf{A}|\mathbf{B}]$ indicates the concatenation of matrix $\mathbf{A}$ with matrix $\mathbf{B}$ and $\mathbf{I}_m$ represents the $m \times m$ identity matrix. We now redefine the original $\mathbf{H}$ to include the column reordering as per the Gaussian elimination. The corresponding generator matrix is then $\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$.

$\mathbf{G}$ is not in general sparse, so the encoding complexity is $O(n^2)$ per block. However, with simple modifications of the structure of $\mathbf{H}$ the encoding complexity can be reduced significantly [62]. Rate 1/2 codes with encoding time linear in n have been shown to have good performance.

2.3 Decoding

Having encoded our source vector $\mathbf{s}$ using the generator matrix we transmit the message $\mathbf{t} := \mathbf{G}^\mathsf{T}\mathbf{s}$. The channel introduces noise and we receive the vector $\mathbf{r} := \mathbf{t} + \mathbf{n}$, where $\mathbf{n}$ is the noise vector. We multiply our received vector by $\mathbf{H}$ to form the syndrome vector $\mathbf{z} := \mathbf{Hr} = \mathbf{HG}^\mathsf{T}\mathbf{s} + \mathbf{Hn} = \mathbf{Hn}$. All arithmetic is performed over a finite field. Appendix B gives a brief review of arithmetic in finite fields.

We perform syndrome decoding. That is, we want to find the most probable vector $\mathbf{x}$ (according to the channel model) that explains the observed syndrome vector $\mathbf{Hx} = \mathbf{z}$. $\mathbf{x}$ is then our estimate of the noise vector. The exact decoding problem is known to be NP-complete even when the column weight is fixed to be 3 [5], so we must settle for an approximate

algorithm. The algorithm we now describe proves to be highly effective.

We use an iterative probabilistic decoding algorithm known variously as a sum/product [91] or belief propagation [69] algorithm. At each step we estimate the posterior probability of the value of each noise symbol, given the received signal and the channel properties. The process is best viewed as a message passing algorithm on the bipartite graph defined by $\mathbf{H}$ in which we have two sets of nodes: nodes representing noise symbols, and nodes representing check symbols (figure 2.2). Nodes z_i and x_j are connected if the corresponding matrix entry H_{ij} is non-zero. The directed edges show the causal relationships: the state of a check node is determined by the state of the noise nodes to which it is connected. We refer to the neighbours of a noise node x_j as its *children* and to the neighbours of a check node as its *parents*.

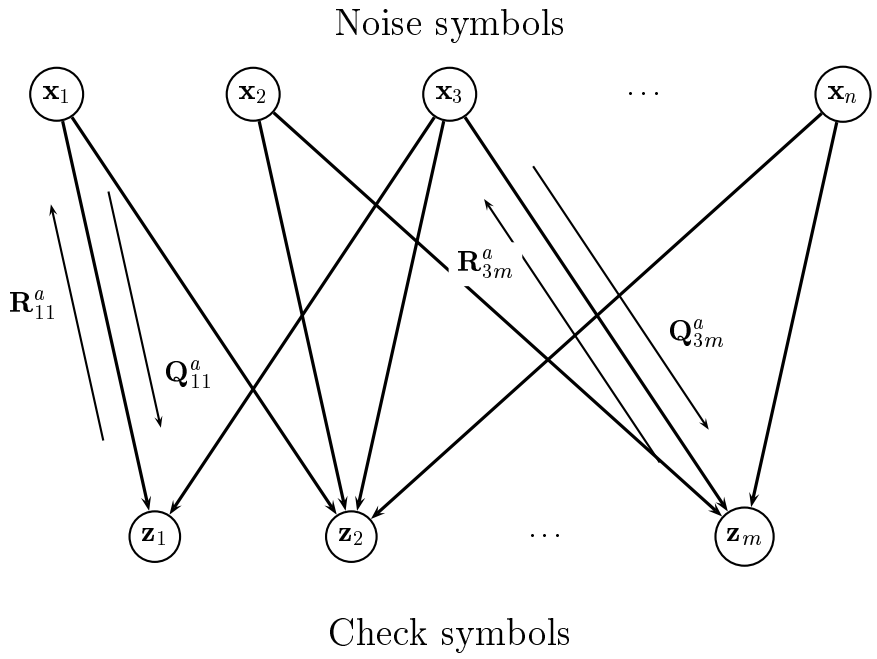


Figure 2.2: Message passing on the bipartite graph representing a parity-check matrix.

At each step of the decoding algorithm each noise node x_j sends messages Q^a_{ij} to each child z_i which are supposed to approximate the node's belief that it is in state a, given messages received from all its other children. Also, each check z_i sends messages R^a_{ij} to each parent x_j approximating the probability of check i being satisfied if the parent is assumed to be in state a, taking into account messages received from all its other parents. After each step we examine the messages and produce a tentative decoding. The decoding algorithm consists of iteratively updating these messages until the tentative decoding satisfies the observed syndrome vector (declare a success) or a preset maximum number of iterations is reached (declare a failure). The maximum number of iterations may be set to perhaps ten times the typical number, improving the success rate while imposing little overhead on the average decoding time. Although it is in principle possible for the decoder to converge to the wrong

noise vector, we do not observe this in practice. That is, (empirically) all decoding errors are detected.

If the underlying graph has a tree structure the algorithm is known to converge to the true posterior distribution after a number of iterations equal to the diameter of the tree. In our problem there are many cycles in the graph and occasionally the algorithm fails to converge at all. We can take care to avoid short cycles in our graph, but in practice this has little impact on the decoding performance.

2.3.1 Initialisation

We initialise the algorithm by setting each message Q_{ij}^a to f_j^a, the prior probability that the jth noise symbol is a. In the case of a binary symmetric channel f_j^1 would be equal to the crossover probability. In this thesis we mainly consider the binary-input additive white Gaussian noise channel, outlined in Appendix C.1.

2.3.2 Updating R_{ij}^a

The messages R_{ij}^a that check i sends to parent j should be the probability of check i being satisfied if the parent was in state a. In the sense it is used here, check i is 'satisfied' if it agrees with the corresponding syndrome symbol z_i. In syndrome decoding, z_i is not necessarily zero. The laws of probability tell us:

$$\mathbf{P}(z_i|x_j = a) = \sum_{\mathbf{x}:x_j=a} \mathbf{P}(z_i|\mathbf{x})\,\mathbf{P}(\mathbf{x}|x_j = a) \tag{2.1}$$

Hence, we sum over all configurations $\mathbf{x}$ for which the check is satisfied *and* the parent is in state a and add up the probability of the configuration (product of associated $\mathbf{Q}$ messages). For node z_i we update the outgoing message to node x_j for each value a as follows:

$$R_{ij}^a = \sum_{\mathbf{x}:x_j=a} \mathbf{P}(z_i|\mathbf{x}) \prod_{k\in\mathcal{N}(i)\backslash j} Q_{ik}^{x_k} \tag{2.2}$$

where $\mathcal{N}(i)$ denotes the set of indices of the parents of node z_i and $\mathcal{N}(i)\backslash j$ denotes the indices of all parents except node j. The probability $\mathbf{P}(z_j|\mathbf{x})$ of the check being satisfied is either 0 or 1 for any given configuration $\mathbf{x}$.

$\mathbf{R}$ can be calculated efficiently by treating the partial sums of a parity check as the states in a Markov chain, with transition probabilities given by the appropriate $\mathbf{Q}$ values. The forward-backward algorithm (See appendix D) is used to calculate the forward and backward

probabilities

$$F_{ij}(a) = \mathbf{P}\left(\sum_{k<j} \mathbf{H}_{ik} x_k = a\right) \tag{2.3}$$

$$B_{ij}(a) = \mathbf{P}\left(\sum_{k>j} \mathbf{H}_{ik} x_k = a\right) \tag{2.4}$$

according to the probabilities given by the $\mathbf{Q}$ messages. The calculation of R_{ij}^a is then straightforward:

$$R_{ij}^a = \sum_{\{s,t:H_{ij}a+s+t=z_i\}} F_{ij}(s) B_{ij}(t). \tag{2.5}$$

2.3.3 Updating Q_{ij}^a

The messages Q_{ij}^a that noise node j sends to check i should be the belief the parent has that it is in state a, according to all the other children. Applying Bayes' theorem:

$$\mathbf{P}(x_j = a | \{z_i\}_{i \in \mathcal{M}(j)\setminus i}) = \frac{\mathbf{P}(x_j = a)\, \mathbf{P}(\{z_i\}_{i \in \mathcal{M}(j)\setminus i} | x_j = a)}{\mathbf{P}(\{z_i\}_{i \in \mathcal{M}(j)\setminus i})} \tag{2.6}$$

Treating the symbols of $\mathbf{z}$ as independent, we take the product of all the other childrens' votes for state a, weighted by the prior. For node x_j we update the outgoing message to node z_i for each value a as follows:

$$Q_{ij}^a = \alpha_{ij} f_j^a \prod_{k \in \mathcal{M}(j)\setminus i} R_{kj}^a \tag{2.7}$$

where $\mathcal{M}(j)$ denotes the set of indices of the children of node x_j and f_j^a is the prior probability that x_j is in state a. The normalising constant α_{ij} ensures $\sum_a Q_{ij}^a = 1$. The update may be implemented using a forward-backward algorithm.

2.3.4 Tentative decoding

After updating the $\mathbf{Q}$ and $\mathbf{R}$ messages we calculate, for each index $j \in \{1, \ldots, n\}$ and possible states a, the quantity

$$\hat{n}_j = \operatorname*{argmax}_a f_j^a \prod_{k \in \mathcal{M}(j)} R_{kj}^a \tag{2.8}$$

The vector $\hat{\mathbf{n}}$ is our tentative decoding. If this satisfies the syndrome equation $\mathbf{H}\hat{\mathbf{n}} = \mathbf{z}$ then we terminate decoding, declaring a success. Otherwise we iterate, updating $\mathbf{R}$ and $\mathbf{Q}$ again until either decoding is successful or we declare a failure after a fixed number of iterations (we use 500).

2.4 Constructing LDPC matrices

One of the attractions of LDPC codes is their simple description in terms of a random sparse parity-check matrix, making it easy to construct LDPC codes of any rate. Many good codes can be built by specifying a fixed weight for each row and each column, and constructing at random subject to those constraints. However, the best LDPC codes use further design criteria. We review the basic constructions.

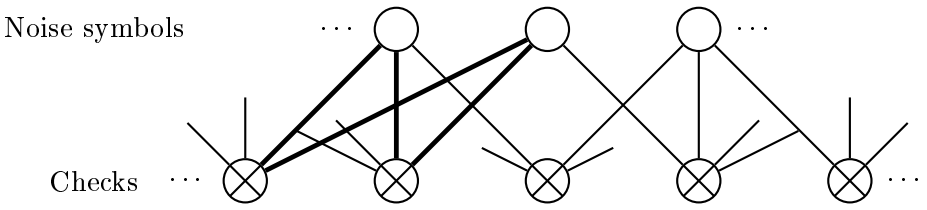
2.4.1 Gallager's prescription

In his original work Gallager imposed a fixed column weight j and a fixed row weight k. The parity-check matrix was divided horizontally into j equal size submatrices, each containing a single '1' in each column. Without loss of generality the first submatrix was constructed in some predetermined manner (eg. k concatenated identity matrices). The subsequent submatrices were random column permutations of the first. An example of a matrix constructed in this way was shown in figure 2.1.

It was for the ensemble in which all such matrices were assigned equal probability that Gallager proved his results. A schematic representation of Gallager's construction is shown in figure 2.3(c).

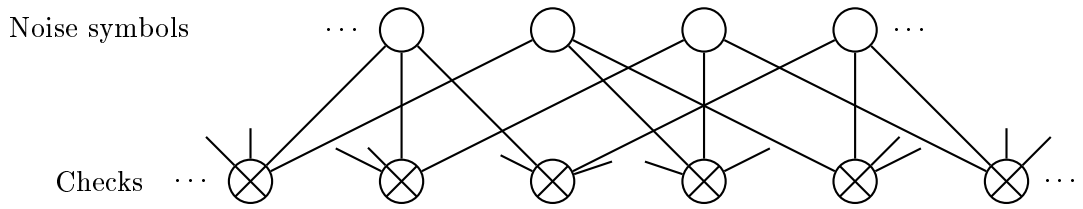
2.4.2 MacKay's constructions

MacKay [58] was interested in keeping to a minimum the number of short cycles in the bipartite graph representing the parity-check matrix. The presence of short cycles can create difficulties for the belief propagation decoder. Consider the following fragment of a graph representing a parity-check matrix:



In this case, there is a cycle of length 4 (indicated by the bold lines). If the state of the three noise symbols x_i are changed to $(x_i + c)$ for arbitrary $c \in GF(2^p)$, then only one of the five checks is affected. The decoder can get stuck in such a configuration, finding the evidence of four satisfied checks difficult to contradict.

By ensuring that any pair of columns in the parity-check matrix has an overlap of at most one we avoid any cycles of length 4. In principle it is possible for a code to have a minimum distance of 4 even when the minimum cycle length is 6, as shown by the following fragment in which there are $(2^p - 1)$ codewords that take some non-zero value $c \in GF(2^p)$ on the four indicated noise symbols and are 0 elsewhere:



For randomly constructed codes, as the blocklength increases so the minimum distance increases (linearly with the blocklength, for almost all codes [27]), and configurations such as the one above become increasingly unlikely.

Reducing the overall weight of the matrix by introducing some columns of weight 2 can improve decoding, but steps must be taken to reduce the probability of low weight codewords. MacKay describes the following construction methods for matrices with no cycles of length 4:

Construction 1A This is the basic construction, in which we have a fixed weight per column t (*e.g.* , $t = 3$) and construct the matrix at random keeping the weight per row (t_r) as uniform as possible, and overlap between any two columns no greater than 1. Shown in figure 2.3(a).

Construction 2A As per 1A, except up to $m/2$ of the columns have weight 2. These weight 2 columns are constructed in the form of two (possibly truncated) identity matrices of size $m/2 \times m/2$, one above the other. Shown in figure 2.3(d).

Constructions 1B, 2B Some carefully chosen columns from a 1A or 2A matrix are deleted, so that the bipartite graph of the matrix has no short cycles of length less than some length l, (*e.g.* , $l = 6$).

MacKay also generalised Gallager's construction method, using superposed permutation matrices to build up regular [58] and irregular [62] matrices.

2.4.3 Ultra-light matrices

When reducing the weight of his binary matrices, MacKay used a maximum of $m/2$ weight-2 columns. Any more makes low weight codewords unacceptably likely, leading to increased probability of decoding errors (undetected errors). With non-binary codes we can include more weight-2 columns before encountering such problems. The constructions we use are a development of the '2A' construction mentioned above, using a succession of smaller stacked identity matrices.

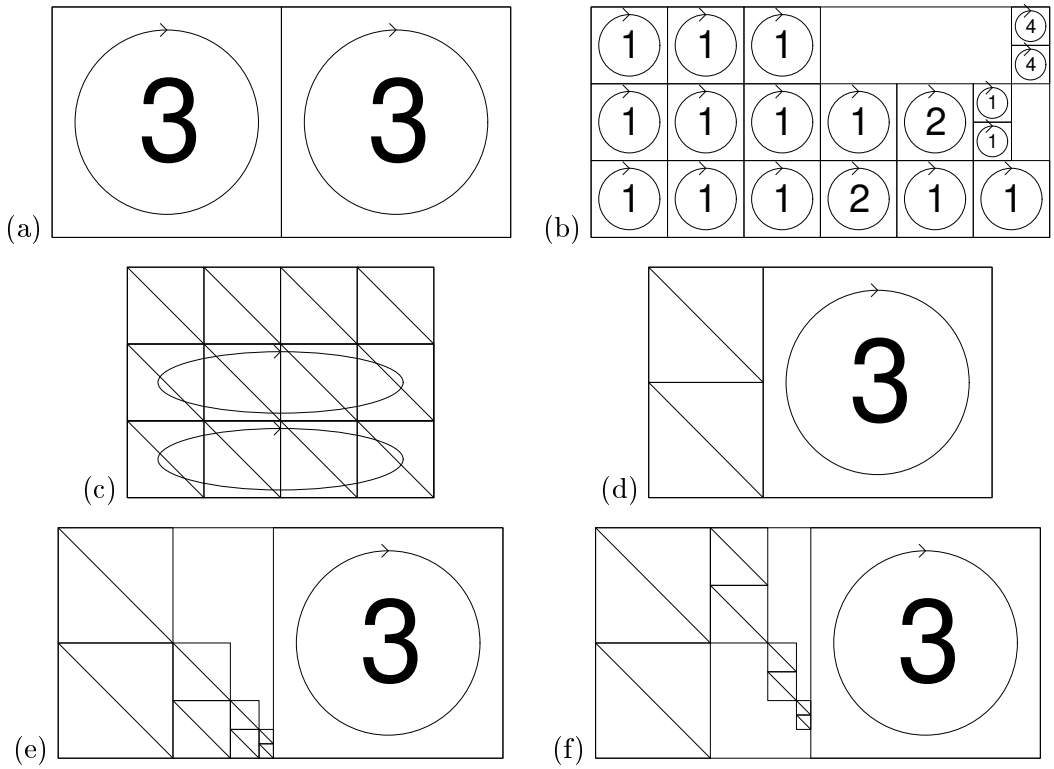


Figure 2.3: Schematic illustration of some constructions of LDPC codes. (a) construction 1A for a code with $t = 3$, $t_r = 6$ and rate 1/2; (b) variant of construction IR for a code with rate 1/2; (c) Gallager's construction for a code with rate 1/4; (d) construction 2A for a code with rate 1/3; (e) construction UL-A for a code with rate 15/31; (f) construction UL-B for a code with rate 15/31; (Adapted from diagrams by MacKay [62]).

Notation: an integer in a circle represents a number of permutation matrices superposed on the surrounding square. A diagonal line represents an identity matrix. An ellipse imposed on a block within the matrix represents a random permutation of all the columns in that block.

UL-A Having constructed $m/2$ weight 2 columns we construct a further $m/4$ by stacking two $m/4 \times m/4$ identity matrices beside one of the previous ones. This process continues until a maximum of m weight 2 columns have been constructed, each time placing the new blocks beside the previous ones so that rows may have weights of 3, 4 or higher depending on how many times the process is repeated. We then continue filling the remaining columns, keeping the row weight as uniform as possible. Shown in figure 2.3(e).

UL-B Similar to method UL-A, except the smaller identity blocks are placed so that each row has weight at most 2 before the higher weight columns are filled. Shown in figure 2.3(f).

2.4.4 Irregular matrices

Irregular matrices are those for which there are several row and column weights. The matrix parameters are described in terms of row and column profiles, as discussed in Chapter 4. To build an irregular matrix we use a construction described by Luby, Mitzenmacher, Shokrollahi and Spielman [56].

Construction IR Let T be the total number of non-zero entries in our matrix. In the notation of Chapter 4, T is given by:

$$T = n \sum_i i\lambda_i = m \sum_i i\rho_i \tag{2.9}$$

where n is the blocklength, m is the number of parity-checks, λ_i is the fraction of columns with weight i and ρ_i is the fraction of rows with weight i.

We consider a bipartite graph with T 'left nodes' $\{L_1, \ldots, L_T\}$ and T 'right nodes' $\{R_1, \ldots, R_T\}$. For each column of weight j in our matrix we label j left nodes with that column's index. Similarly we label i right nodes with the index of each row of weight i. We connect each node L_i to R_i.

Finally, we permute the labels on all the right nodes. This defines our parity-check matrix. When permuting the right nodes, we must avoid defining duplicate edges: we check that the ρ_i right labels belonging a given row of weight i are all permuted to match left nodes of different columns.

Constructions IR-UL-A, IR-UL-B If our column profile specifies some columns of weight 2 then we use a variation of the method IR. In these variations IR-UL-A and IR-UL-B, we choose a permutation for the row labels such that the weight two columns are constructed according to constructions UL-A and UL-B respectively. In all other respects the constructions are identical to IR.

Chapter 3

Non-Binary Low-Density Parity-Check Codes

Base 8 is just like base 10, if you are missing two fingers.

— Tom Lehrer

So far we have considered binary low-density parity-check codes, represented by sparse binary matrices or by corresponding bipartite graphs. We can generalise LDPC codes by using the same bipartite graphs but allowing the noise nodes to take values from some finite alphabet and allowing the check nodes to impose constraints more complex than binary parity-checks. Such constructions were first suggested by Tanner [83]. For example, using noise nodes defined over the finite field $GF(q)$, check m could require:

$$\sum_{j \in \mathcal{N}(m)} f_{mj}(x_j) = 0 \tag{3.1}$$

where $\mathcal{N}(m)$ is the set of noise nodes connected to check m and each f_{mj} is one of the $(q-1)!$ permutations of the field elements. In this chapter we restrict ourselves to codes over $GF(q)$ with linear constraints, in particular to codes defined by sparse non-binary parity-check matrices.

We consider codes over the finite fields with q elements (denoted $GF(q)$) where $q = 2^p$ for some integer p. We choose powers of two because we continue to use binary channels, transmitting one q-ary symbol for every p uses of the channel. A symbol from the field $GF(2^p)$ may be represented as a binary string of p bits. For example, we might represent the symbol 3 in $GF(8)$ as the binary string 011. See Appendix B for details.

The decoder interprets p bits $(x_1, \ldots, x_p)$ from the channel as a single q-ary symbol and sets the prior distribution for that symbol by assuming a product distribution for the values of each constituent bit. That is to say:

$$f^a := \prod_{i=1}^{p} f_{x_i}^{a_i} \tag{3.2}$$

where $f_{x_i}^{a_i}$ is the likelihood the i'th constituent bit (x_i) is equal to a_i, where $(a_1, \ldots, a_p)$ is the binary representation of the symbol a.

We may construct the parity-check matrix $\mathbf{H}$ as before, but we now have $q-1$ choices for each non-zero entry. Formally, the decoding algorithm has precisely the form described in equations (2.2, 2.7, 2.8) although the symbol a now ranges over q possible values instead of just two. The updating of the $\mathbf{R}$ messages takes of order q^2 per iteration using the forward-backward algorithm, so there is a practical limit on the size of field that is computationally feasible. A faster implementation of the decoding algorithm is discussed in section 3.4.

With each symbol $a \in GF(2^p)$ we can associate a $p \times p$ binary matrix. Multiplication of a symbol b by a is equivalent to matrix multiplication (mod 2) of the binary string for b by the matrix associated with a. By replacing each symbol in the q-ary matrices $(\mathbf{G}_q, \mathbf{H}_q)$ by the associated binary $p \times p$ blocks we obtain binary matrices $(\mathbf{G}_2, \mathbf{H}_2)$ that are p times as large in each direction. To multiply a q-ary vector $\mathbf{x}_q$ by $\mathbf{G}_q$ we can form the binary representation of $\mathbf{x}_q$, multiply by $\mathbf{G}_2$, and take the q-ary representation of the resulting binary vector.

In particular, the codewords of the binary code generated by $\mathbf{G}_2$ are precisely the binary representations of the q-ary codewords generated by $\mathbf{G}_q$. To understand why q-ary codes can outperform the related binary codes, we must examine the decoding algorithm in greater detail.

3.1 Decoding over $GF(q)$

MacKay [58] proved that, given an optimal decoder, LDPC codes can approach arbitrarily close to the Shannon limit if we choose sufficiently high (fixed) column weight of $\mathbf{H}$ and then choose a sufficiently long blocklength. However, as we increase the column weight the performance of our iterative decoding algorithm decreases because the number of cycles in the associated bipartite graph increases drastically.

By moving to $GF(q)$ we manage to increase the mean column weight t of the equivalent binary matrix $\mathbf{H}_2$ while retaining the same bipartite graph on which we perform the decoding. The drawback is that the decoding complexity is increased.

In figure 3.1 we compare the graphs of two equivalent matrix fragments. We can see that the q-ary code contains no cycles, whereas the binary code has a cycle of length 4 (highlighted). Hence we would expect the decoding behaviour to depend on our choice of field $GF(q)$ even though the transmitted (binary) messages are identical.

Another way of viewing the difference between binary and q-ary codes is that we increase the state space of each node in the decoding graph by decoding over $GF(q)$. This increase allows us to track correlations in the true posterior distribution that are not detectable by the binary algorithm. Increasing the field order q for LDPC codes is comparable to increasing the memory of convolutional codes.

It is worth pointing out the importance of the choice of non-zero elements in a parity-check matrix defined over $GF(2^p)$. For example, if we choose them all to be ones then the graph of

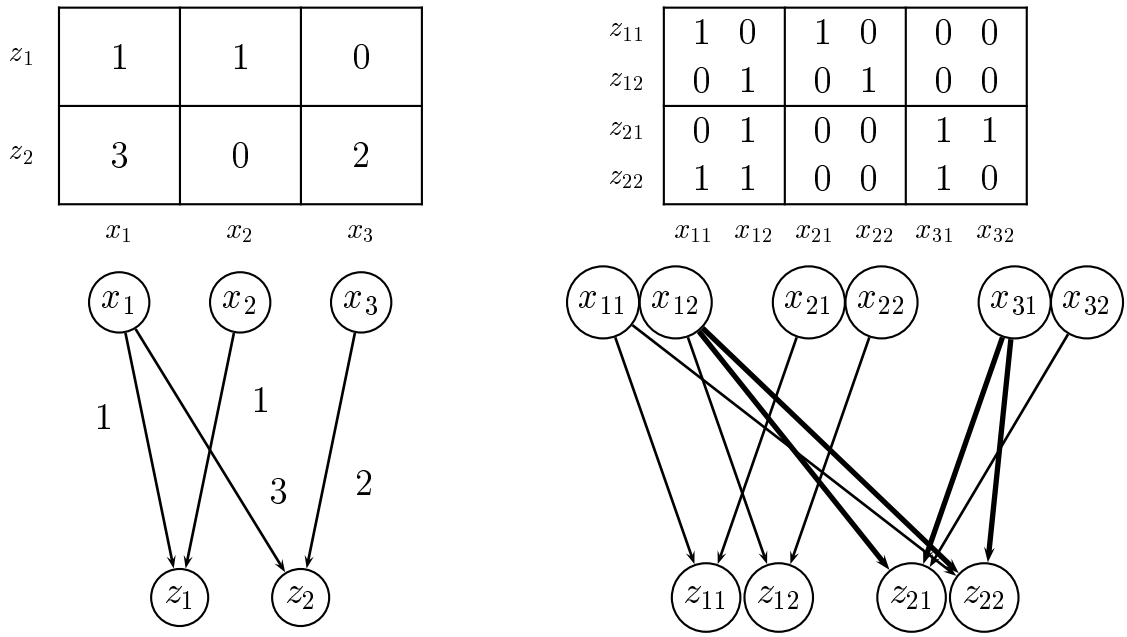


Figure 3.1: Upper panels: fragments of equivalent parity-check matrices over (left) $GF(4)$ and (right) $GF(2)$. Lower panels: comparison of corresponding graph structure. Note the presence of a short cycle in the graph for the binary code.

the equivalent binary code splits into p disjoint subgraphs. This split occurs because all the non-zero $p \times p$ blocks in the equivalent binary parity-check matrix are identity matrices. In this case the decoding is equivalent to that of the binary algorithm applied to each subgraph.

One sensible choice of non-zero entries is to maximise the expected entropy of one symbol of the syndrome vector $\mathbf{z}$ using the channel model. As we shall see in section 3.5, this gives a small but noticeable improvement over a purely random choice.

3.2 Initial experiments with non-binary LDPC codes

3.2.1 Description of experiments

Having constructed a parity-check matrix, we test its performance by numerical simulation. We simulate a binary Gaussian channel as described in Appendix C.1 and examine the success of decoding several thousand blocks. Each q-ary symbol is transmitted using p channel uses, and the likelihood of each corresponding noise symbol depends on p channel outputs, as per equation 3.2.

The performance is gauged by plotting the bit error rate (*i.e.* the fraction of bits that are incorrectly decoded) against the signal to noise ratio. As mentioned in section 2.3, we can distinguish between detected and undetected errors. Not one of our simulations suffered undetected errors.

In all the experiments presented in this thesis, we allow the decoding algorithm to run for a maximum of 500 iterations before announcing a decoding failure. The average number of iterations required depends on the noise level, but is typically between 15 and 40 for bit error rates below 10^{-2}. All our results include a one sigma confidence interval for the bit error rate.

In our initial investigations of non-binary codes, we constructed parity-check matrices with a fixed column symbol weight of three, defined over fields $GF(2)$, $GF(4)$ and $GF(8)$. To compare like with like we compare codes over $GF(2^p)$ of length N with binary codes of length Np. We refer to Np as the binary blocklength.

3.2.2 Confusing results

The results shown in figure 3.2 are, at first sight, rather strange. They show results for codes with a fixed column weight of 3 defined over fields $GF(2), GF(4)$ and $GF(8)$ for two different rates. We see that for rate 1/2 codes we get an improvement going from $GF(2)$ to $GF(4)$ but the code over $GF(8)$ is worst of all. For the rate 1/4 codes the best results are for the binary codes, and the codes over $GF(4)$ and $GF(8)$ perform worse. Note that the binary blocklength was fixed for each group of codes.

What is going on? To answer this we must examine the dependence of decoding performance on the matrix weight, the topic of the next section.

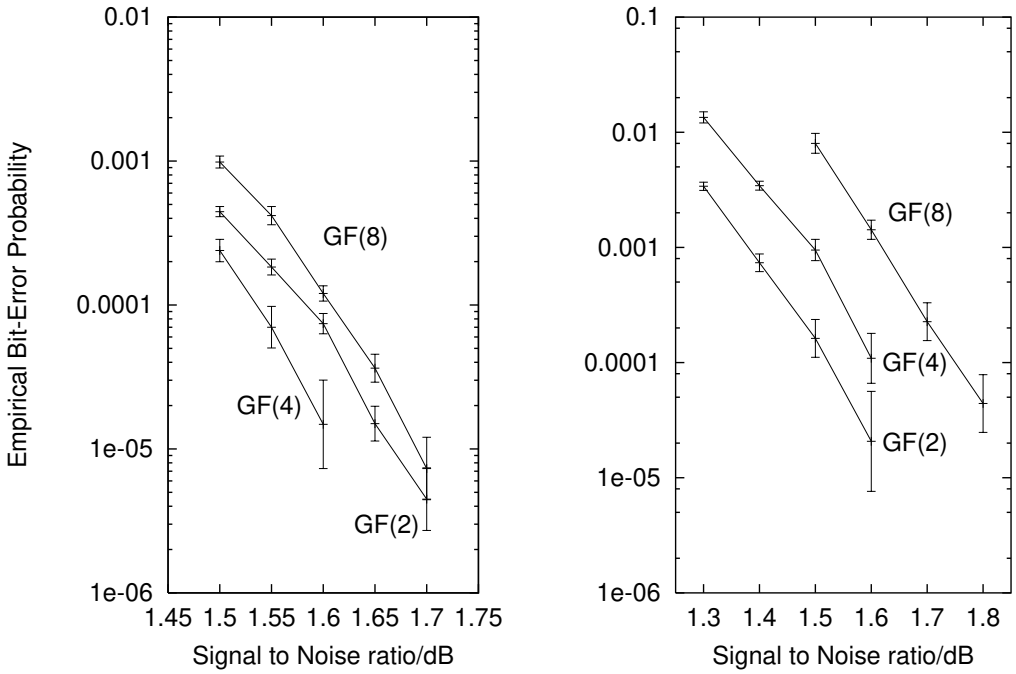


Figure 3.2: Performance of codes over fields $GF(2), GF(4), GF(8)$ with fixed column weight 3. x-axis: signal to noise ratio. y-axis: bit error rate. Left: Rate 1/2 codes, blocklength 3000 bits. Right: Rate 1/4 codes, blocklength 8000 bits.

3.3 Monte-Carlo analysis

How should we choose good parameters for the parity-check matrix? It would be useful to measure the effect of changes of field order q and mean column weight t on the decoding performance, in order to construct the best codes. We can accomplish this by using Monte Carlo methods to simulate infinite LDPC codes whose associated graphs have a tree structure and hence the decoding algorithm is known to be exact.

3.3.1 Method

We start by producing a large number of noise symbols with associated prior distributions over their states (an 'ensemble') according to the channel model. To simulate an infinite code we update the messages $\mathbf{Q}$ for each noise symbol by creating a local neighbourhood populated with elements chosen at random from our ensemble and then running one step of the decoding algorithm. This creates a new ensemble representing noise symbols after one iteration. Since each local neighbourhood is cycle-free, the overall effect is of simulating the decoding algorithm on an infinite tree. We now describe this method in more detail.

We form an ensemble $\mathcal{S}$ of S noise symbols and associated Q messages according to the channel model:

$$\mathcal{S} = \left\{ n_i, \{Q_i^a\}_{a=0}^{q-1} \right\}_{i=1}^{S} \tag{3.3}$$

where Q_i^a is set to f_i^a, the likelihood that noise symbol i is equal to a. The size, S, of the ensemble is large, typically about 100,000.

We now want to form an ensemble $\mathcal{S}'$ representing S noise symbols after one iteration of the decoding algorithm. To generate one entry in $\mathcal{S}'$ we proceed as follows. We create a pair $\left\{ n', \{f_{n'}^a\}_a \right\}$ according to the noise model, generate a row weight $(r+1)$ and r column weights $\{(c_1+1), \ldots, (c_r+1)\}$. These values are determined by our choice of mean column weight and code rate. We now create a fragment of an infinite code as shown in figure 3.3. Our new noise symbol n' is at the root of the tree, connected to r checks. Check z_i is connected to n' via the matrix entry $H_{in'}$, and to c_i other noise symbols $\{n_{i1}, \ldots, n_{ic_i}\}$ (chosen at random from $\mathcal{S}$) via the matrix entries $\{H_{i1}, \ldots, H_{ic_i}, H_{in'}\}$. This allows us to calculate the values of the checks $\{z_i\}$:

$$z_i = H_{in'} n' + \sum_{j=1}^{c_i} H_{ij} n_{ij}. \tag{3.4}$$

We can now calculate the incoming messages $R_{in'}^a$ to the root according to the rule in equation 2.2. Given these messages, the root can produce the outgoing messages $\{Q_{n'}^a\}_{a=0}^{q-1}$. This defines one entry $\left\{ n', \{Q_{n'}^a\}_a \right\}$ in the new ensemble $\mathcal{S}'$.

Once we have created a new ensemble $\mathcal{S}'$ we iterate, replacing the old ensemble $\mathcal{S}$ with $\mathcal{S}'$. In this way we produce successive ensembles containing approximations to the distribution

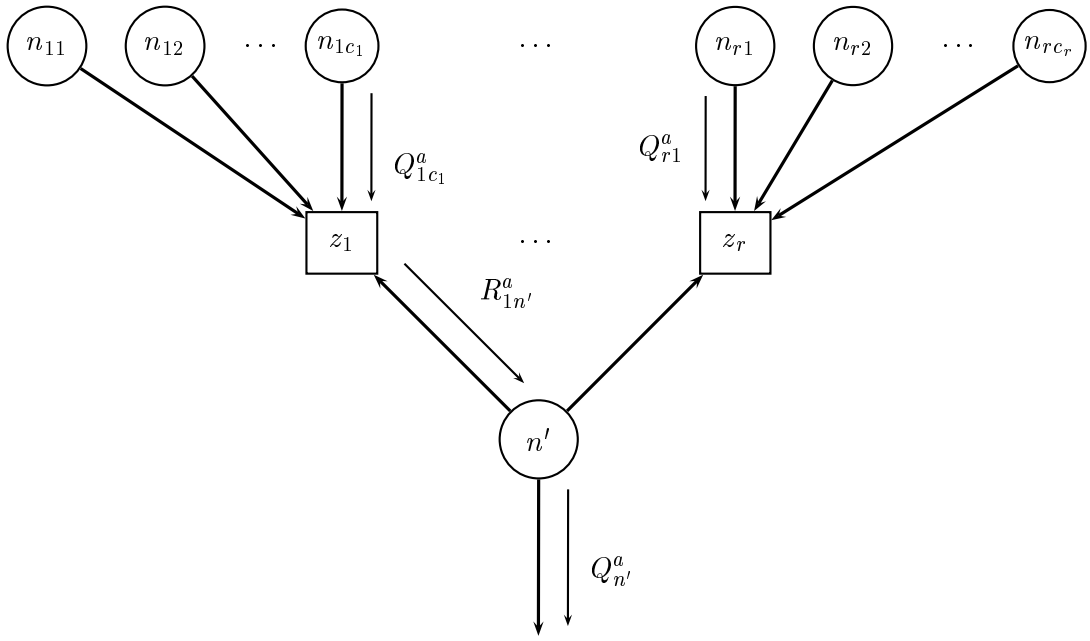


Figure 3.3: Fragment of the graph for an infinite LDPC code.

of Q messages in an infinite network after an arbitrary number of iterations. For successful decoding the average Shannon entropy of the Q messages

$$H = \frac{1}{S} \sum_{i=1}^{S} \prod_{a=0}^{q-1} Q_i^a \log_q Q_i^a \tag{3.5}$$

should become arbitrarily small as decoding progresses.

Comparing the average entropy per bit as calculated by the Monte Carlo simulation with results taken from averaging comparable finite LDPC codes, we verify that we produce a good approximation to the real behaviour (figure 3.4) well beyond the number of iterations at which cycles render the algorithm inexact (4 or less, for these codes). As the blocklength increases the finite-blocklength results appear to approach the Monte Carlo predictions.

3.3.2 MC results

We may declare a decoding 'success' if the average entropy of the Q messages in our ensemble drops below a chosen threshold. With this approach it is possible to investigate the effect of changes in field order, matrix weight and noise level on the decoding performance.

In figure 3.5 we plot the minimum signal to noise ratio for which average bit entropy reaches 10^{-5} within 80 iterations as a function of the mean column weight of the parity-check matrix. We see that there is an optimum mean column weight which decreases as the order of the field increases. The results suggest that for best performance (neglecting complexity issues) we should choose the highest order field that is feasible and then choose an appropriate

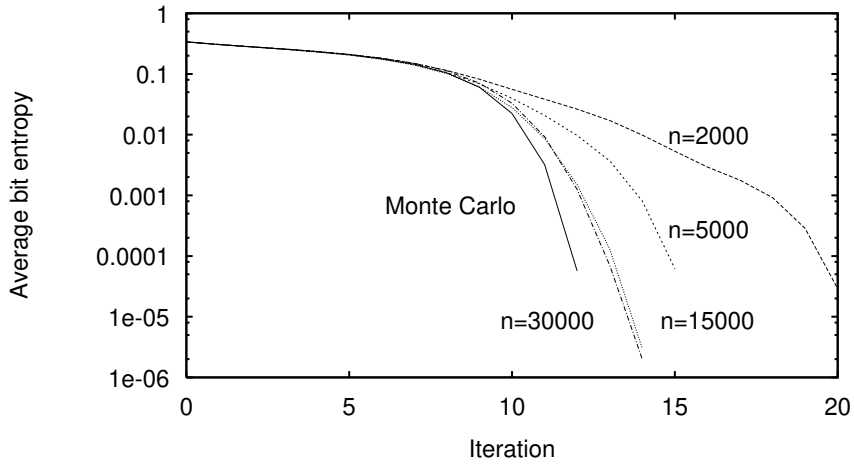


Figure 3.4: Comparison of Monte Carlo and empirical results for several blocklength rate 1/2 codes over $GF(8)$ with column weight 3 at SNR 1.6 dB. x-axis: iteration number. y-axis: mean entropy of one bit of the tentative decoding.

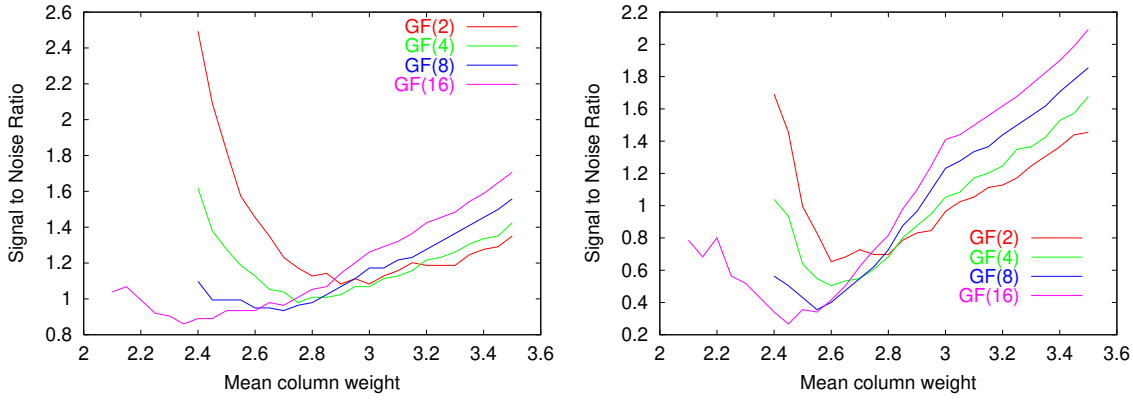


Figure 3.5: Monte Carlo simulation of (left) rate 1/2 and (right) rate 1/4 infinite LDPC codes over the Gaussian Channel. Vertical axis: Minimum signal to noise ratio for which average bit entropy reaches 10^{-5} within 80 iterations. Horizontal axis: mean column weight.

mean column weight.

If we compare figures 3.5 and 3.2 we can now understand our initial 'confusing' results. For codes with a column weight of 3 and rate 1/2, the Monte Carlo results predict that the best codes are those defined over $GF(4)$, followed by $GF(2)$ and $GF(8)$. This is borne out by the empirical results. The Monte Carlo simulations rank column weight 3 codes of rate 1/4 in the order $GF(2)$, $GF(4)$, $GF(8)$, again in agreement with the finite codes.

Intuitively, we can understand this as follows: whatever field order we use, the decoder performs worse as the matrix weight increases because each check node in the network, having more neighbours, is less confident about each neighbour's state. The effect of increasing the field order is similar because each neighbour then has more possible states. This soon counteracts the improved strength of each parity-check.

3.4 Fourier transform decoding

The complexity of the decoding algorithm outlined in section 2.3.2 scales as q^2 for non-binary LDPC codes. We now describe how this complexity can be reduced using a Fourier transform of the probabilities, as suggested by Richardson and Urbanke [75].

In the following, we use the notation of section 2.3. Let $\mathcal{N}(m) := \{n : H_{mn} \neq 0\}$ be the set of noise symbols that participate in check m. The decoder needs to update quantities R^a_{mn}, the probability of check m being satisfied if symbol n of the message $\mathbf{x}$ is considered fixed at $a \in GF(q)$ and the other noise symbols have a separable distribution given by the probabilities $\{Q^b_{mn'} : n' \in \mathcal{N}(m)\backslash n\}$. The new value of R^a_{mn} is:

$$R^a_{mn} = \sum_{\mathbf{x}:x_n=a} \delta\left(\sum_{n'\in\mathcal{N}(m)} H_{mn'}x_{n'} = z_m\right) \prod_{j\in\mathcal{N}(m)\backslash n} Q^{x_j}_{mj} \tag{3.6}$$

This represents a convolution of the quantities Q^a_{mj}, and so the summation can be replaced by a product of the Fourier transforms (taken over the additive group of $GF(q)$) of Q^a_{mj} for $j \in \mathcal{N}(m)\backslash n$, followed by an inverse Fourier transform. The Fourier transform F of a function f over $GF(2)$ is given by $F^0 = f^0 + f^1, F^1 = f^0 - f^1$. Transforms over $GF(2^p)$ can be viewed as a sequence of binary transforms in each of p dimensions. Hence for $GF(4)$ we have

$$F^0 = [f^0 + f^1] + [f^2 + f^3] \tag{3.7}$$
$$F^1 = [f^0 - f^1] + [f^2 - f^3] \tag{3.8}$$
$$F^2 = [f^0 + f^1] - [f^2 + f^3] \tag{3.9}$$
$$F^3 = [f^0 - f^1] - [f^2 - f^3] \tag{3.10}$$

The inverse transform is the same, followed by division by 2^p.

With a slight abuse of notation, let $(\tilde{Q}^0_{mj}, \ldots, \tilde{Q}^{q-1}_{mj})$ represent the Fourier transform of the vector $(Q^0_{mj}, \ldots, Q^{q-1}_{mj})$, after permuting the components to take account of the matrix

entry H_{mj}. Now R^a_{mn} is the a'th coordinate of the inverse transform of

$$\left(\left(\prod_{j \in \mathcal{N}(m) \backslash n} \tilde{Q}^0_{mj} \right), \ldots, \left(\prod_{j \in \mathcal{N}(m) \backslash n} \tilde{Q}^{q-1}_{mj} \right) \right) \tag{3.11}$$

The update of the quantities Q is unchanged.

Each fast Fourier transform takes $(q/2) \log q$ additions and q multiplications. Assuming a column weight $j = 3$ and taking $q = 16$, the total cost per iteration is 48 additions and 72 multiplications per bit. All these operations can be implemented in low precision arithmetic with a small loss in performance [75]. In comparison, to implement the forward-backward algorithm would require 672 multiplications and 612 additions per bit per iteration. In this example, the saving is more than $\log(q)$, because the fast Fourier transform implementation incorporates the normalisation of the Q messages.

3.5 Regular LDPC code results

The Monte Carlo results suggest that carefully constructed non-binary LDPC codes are likely to outperform their binary counterparts. Figure 3.6 shows the performance of three LDPC codes of rate 1/3 with mean column weight 2.5, over fields $GF(2), GF(4)$, and $GF(8)$. Note that each of these codes has a blocklength of 18000 bits. For comparison we include the performance of a rate 1/3 turbo code of blocklength 49152 [45].

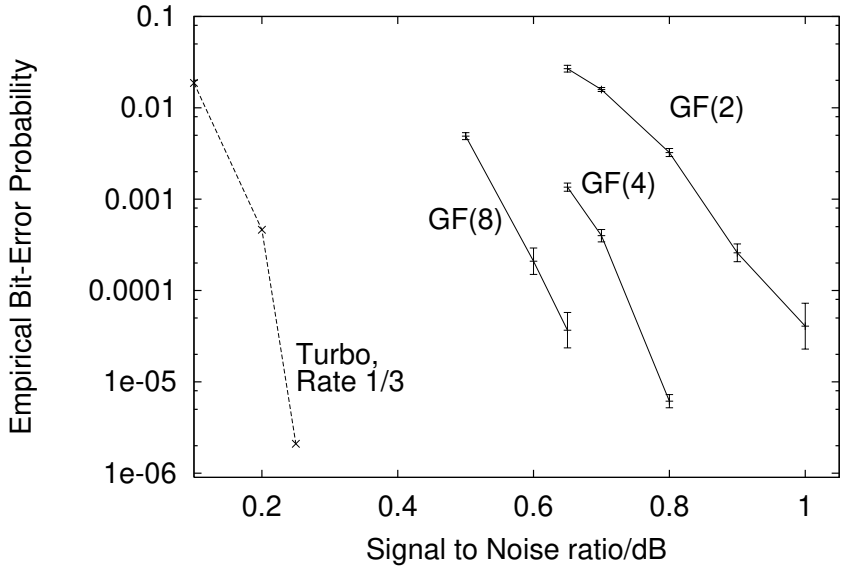


Figure 3.6: Comparison of LDPC codes over a Binary Gaussian Channel over $GF(2), GF(4)$, and $GF(8)$. From left to right: JPL turbo code, rate 1/3, blocklength 49152; LDPC codes with rate 1/3, blocklength 18000 bits, mean column weight 2.5: $GF(8)$, $GF(4)$, $GF(2)$.

These results show that LDPC codes over higher order fields can significantly outperform

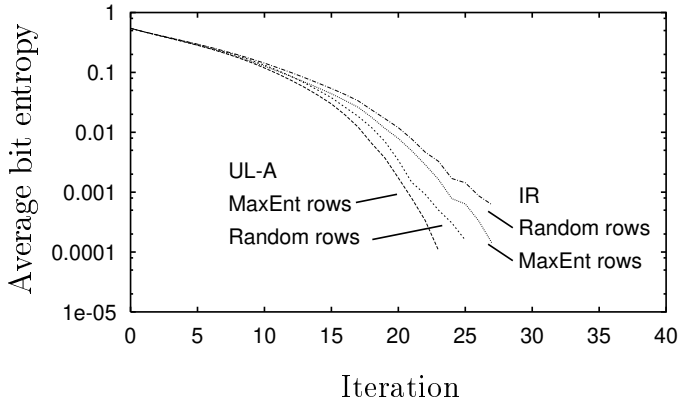


Figure 3.7: Comparison of non-binary construction methods (see section 2.4) for rate 1/4 codes over $GF(8)$ with mean column weight 2.25. Horizontal axis: decoding iteration. Vertical axis: average entropy per bit of the tentative decoding. Construction methods from left to right: UL-A (weight-2 columns constructed using blocks of identity matrices) with non-zero entries in each row of the parity-check matrix chosen to maximise the expected entropy of one symbol of the syndrome vector; UL-A with random non-zero matrix entries; Construction IR (disordered weight-2 columns) with entropy maximising matrix entries; Construction IR with random matrix entries.

binary LDPC codes of similar blocklength. An improvement of 0.4 dB is shown for the rate 1/3 code moving from binary to $GF(8)$ construction, halving the distance to the turbo code performance.

Comparison of construction methods

Figure 3.7 compares several different constructions of LDPC codes. The entropy of the tentative decoding is shown as a function of the number of iterations of the decoding algorithm, for codes defined over $GF(8)$ with a mean column weight of 2.25. The best constructions are those for which the weight-2 columns are laid out in identity matrices – construction UL-A of section 2.4. The codes with non-zero row entries chosen to maximise the entropy of one symbol of the syndrome vector $\mathbf{z}$ decode faster than similar codes with random entries.

High rate results

It is easy to construct LDPC codes with good distance properties for rates less than 2/3. For higher-rate LDPC codes, especially those with shorter blocklengths (*e.g.* less than 2000), it is important to impose constraints on the parity-check matrix to avoid low weight codewords. For example, a rate 2/3 binary LDPC code of length 540 and column weight 3, constructed with Gallager's original construction, has an expected minimum distance of 5 [59].

With some effort, however, it is possible to construct good high-rate LDPC codes. Figure 3.8 shows the performance of a column weight 3 rate 8/9 LDPC code over $GF(16)$ applied to the 16–ary symmetric channel. The code had a blocklength of 3996 bits, and had no cy-

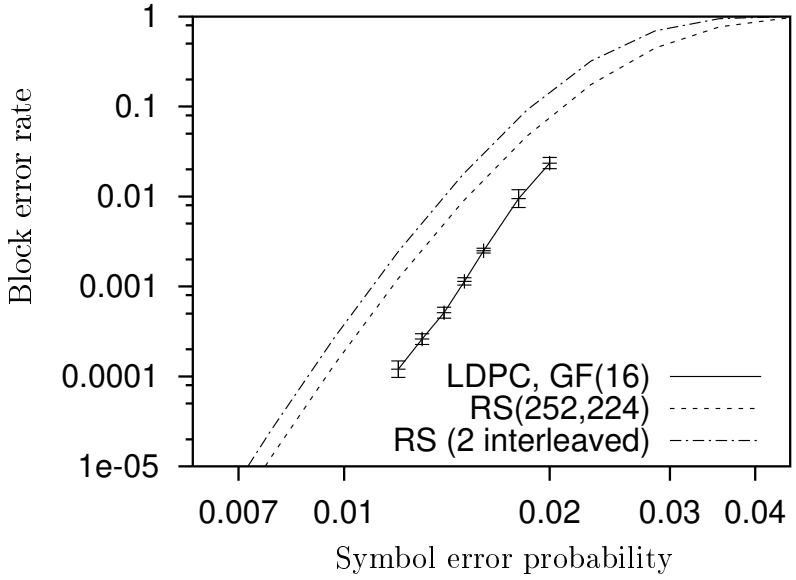


Figure 3.8: Rate 8/9 LDPC code over GF(16) applied to 16–ary symmetric channel. Weight per column $t = 3$. The code is compared with two Reed–Solomon codes similar to those used in discdrives. (Reed-Solomon calculation by MacKay [59])

cles of length 4. The code is compared with two Reed–Solomon codes of blocklength 2016 bits, similar to those used in discdrives. The horizontal axis shows the fraction of symbols corrupted by the channel. The LDPC code is shown to have a smaller block error rate for a given noise level than the Reed-Solomon codes.

In this chapter we have shown how LDPC codes defined over fields $GF(q)$ can outperform comparable binary LDPC codes. In the next chapter we consider irregular LDPC codes – codes defined by parity-check matrices with non-uniform column weights. We will show that such codes outperform all other known codes for Gaussian channels.

Chapter 4

Irregular Low-Density Parity-Check Codes

Oh, I can't stand scientists. Have they nothing better to do with their time than invent stuff and discover things?

— Ardal O'Hanlon

Gallager's original exposition of LDPC codes considered parity-check matrices with a single column weight. In the previous chapter we relaxed this constraint slightly, allowing non-integer mean column weight but keeping the column weight as uniform as possible.

Another possibility is to allow a variety of column weights, constructing *irregular* LDPC codes. Luby, Mitzenmacher, Shokrollahi and Spielman [56, 57] showed that carefully constructed irregular binary LDPC codes outperform regular ones.

To gain intuition into this result, consider again the graph-based view of the decoding algorithm. Ideally a noise node should be connected to many check nodes, to gain most information about its state. Conversely, a check node prefers few parents, so that it can provide more confident estimates of each parent's state. A few 'elite' noise nodes with many connections can determine their state with high confidence early in decoding, becoming a source of accurate information for their neighbours. The introduction of a few elite nodes turns out to be a useful strategy, when used carefully.

To describe an irregular code we require column and row *profiles*. The column profile λ is a vector such that a fraction λ_i of the matrix columns have weight i where i ranges from 1 to t_c, the maximum column weight[1]. For example, a matrix of fixed column weight 3 would have $\lambda = (0, 0, 1)$ whereas a matrix of mean column weight 2.5 might have $\lambda = (0, 0.5, 0.5)$. Similarly, we define a row profile ρ with t_r components, where t_r is the maximum row weight.

Luby *et al.* adopted constructions proven to be optimal for erasure channels and applied them to LDPC codes. Their analysis allowed them to choose a column profile and use linear programming to derive a corresponding row profile. One major drawback of this method is

[1] To avoid confusion, we point out that [56] defined λ and ρ in terms of fractions of *edges* in the bipartite graph, whereas our definition is in terms of fractions of *nodes*.

the necessity of an *ad hoc* choice of column profile. In practice, we find decoding performance is more sensitive to the choice of column weights than row weights, so we would rather choose a (pretty uniform) row profile and then optimise the columns. Furthermore, Luby's method does not deal with codes over $GF(q)$ so we develop an alternative approach.

4.1 Finding good irregular codes over $GF(q)$

We describe an empirical approach to the problem of finding good choices of column weight. Fixing the row weights imposes one constraint on the possible column weights. This defines the space in which valid column profiles λ are found. We define a cost function that assigns a performance score to each possible column profile and then extremise this function. Although the method is simple, it helps us find parameters for excellent non-binary codes, and binary codes that outperform the irregular codes found using Luby's method.

We impose the constraint that λ must be a probability vector:

$$\sum_{i=1}^{t_c} \lambda_i = 1, \qquad 0 \leq \lambda_i \leq 1, \quad \forall\, i. \tag{4.1}$$

Similarly for ρ. Given a row profile, the column profile is further constrained by the rate R of the code:

$$\sum_{i=1}^{t_c} i\lambda_i = (1-R) \sum_{i=1}^{t_r} i\rho_i \tag{4.2}$$

Equation 4.2 simply reflects the fact that the number of non-zero entries in the columns must sum to the number of non-zero entries in the rows.

Equations 4.1 and 4.2 define a closed $(t_c - 2)$-dimensional sub-manifold of $\mathbb{R}^{t_c}$ to which valid values of λ belong. For example, if we imposed a maximum column weight of 3 then we would need to search along a line, as shown in figure 4.1.

4.1.1 Cost functions

We require a cost function to assess the quality of a given column profile and then must minimise this function over our $(t_c - 2)$-dimensional manifold. We have tried two numerical cost functions: one based on empirical decoding trials, the other based on Monte Carlo simulation of decoding. We found the empirical cost function to be the more accurate of the two, given sufficient iterations. The Monte Carlo simulations suffered from the fact that finite blocklength effects seem to be more pronounced with the addition of high weight columns. Thus, the agreement between Monte Carlo simulations and finite codes was not as close as in the case of regular codes.

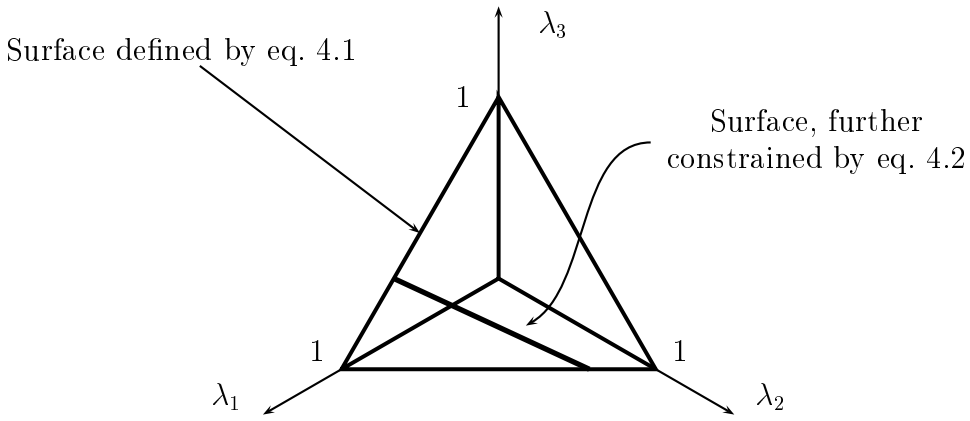


Figure 4.1: Example of the search space for a column profile, when maximum column weight is three. Equation (4.1) defines the simplex $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Adding the constraint (4.2) restricts us to a line.

Empirical cost function

In this approach we first construct a code with the desired parameters. We use the average number of iterations required for decoding to estimate the usefulness of the code. The number of iterations required varies from block to block, so our estimate of the cost function is necessarily fairly noisy.

For the average iteration count to be meaningful, it is necessary that all blocks are decoded. For this reason we use a variant of the decoding algorithm in which we artificially increase the signal to noise level as decoding progresses. This ensures that all blocks can eventually be decoded. For the binary Gaussian channel (see appendix C.1), this is easily accomplished. We keep the noise vector $\mathbf{n}$ and noise amplitude for each bit fixed, but increase the signal amplitude after each iteration.

As an example, take the binary-input Gaussian channel with input signals of $\pm x$ (for some signal amplitude x) and additive Gaussian noise with amplitude 1 and variance 1. If a transmitted signal of $+1.0$ was corrupted by noise of amplitude -1.2, the receiver receives -0.2 and declares a probability 0.4 that $+1$ was the transmitted signal. By increasing the signal amplitude to 1.5, the receiver receives 0.3, and declares a probability 0.7 that $+1$ was the transmitted signal. Hence, by gradually increasing the signal amplitude the decoding problem becomes progressively easier.

Monte Carlo cost function

We can also score parameters using the techniques of the previous chapter, using Monte Carlo methods to simulate an infinite code and calculating the average bit entropy of the tentative decoding after a fixed number of iterations of the decoding algorithm.

Care is required when simulating irregular codes. Consider the construction of the graph fragment of figure 3.3. When we choose a noise node our column profile tells us with what probability that node has i neighbours. However, when we add a check node, the degree of that node must be chosen according to the probability that an *edge* chosen at random is connected to a check node of a given degree. This is not the same quantity as that given by the row profile, just as the fraction of passengers who find themselves on crowded buses is greater than the fraction of buses that are crowded.

In finite irregular codes the presence of high weight columns introduces many cycles of relatively short length and consequently the Monte Carlo simulations agree less well with the empirical results than for regular constructions.

4.1.2 Search method

The problem of minimising a general n-dimensional non-linear function is hard. Several methods exist that are guaranteed to find a local minimum and simulated annealing may be used to help in escaping shallow minima. Our problem is made more acute because not only do we lack any gradient information, but also our evaluations of the function itself are noisy, producing many false local minima.

We implemented the relatively straightforward downhill simplex method (see, for example [70], Chapter 10). We start by choosing $(t_c - 1)$ points in the $(t_c - 2)$-dimensional manifold in which we want to search. These define a simplex[2]. We ensure that the simplex encloses a non-zero volume by finding a basis for our manifold and placing the vertices in the basis vector directions relative to an interior point.

We evaluate the cost function at each vertex, and move the vertex with the highest (worst) score. Usually we reflect it through the centre of the opposite 'face' (average of other vertices), stretching whenever possible to increase the step size. If this results in a higher cost we instead shrink the point towards the opposite face. If this also fails we shrink all vertices towards the best point. Repeating these steps we converge to a local minimum of the cost function. To combat the effect of noise in the estimation of the cost function, a new search can be started from the final point of the previous search.

4.1.3 Alternative search methods

Recent work by Richardson, Shokrollahi and Urbanke [74] offers an analytical approach to the design of LDPC codes based on *density evolution*. Density evolution is a method of analysing the LDPC decoding algorithm by propagating distributions over the LDPC decoding algorithm messages, rather than the messages themselves. For given code parameters, the method allows the expected fraction of incorrectly determined nodes to be calculated as a function of the iteration count.

[2] A *simplex* is the generalization of a tetrahedral region of three dimensional space to k dimensions. The boundary of a k-simplex has $k+1$ 0-faces (vertices), $k(k+1)/2$ 1-faces (edges), and $\binom{k+1}{i+1}$ i-faces.

Although the method assumes a loop-free graph, Richardson *et al.* proved that, with probability that approaches 1 exponentially fast in the blocklength, the decoder will not deviate from the loop-free behaviour by more than ϵ. For a given row/column profile, the supremum over all noise levels for which the fraction of incorrectly determined nodes approaches zero is the "cutoff noise level" for that profile. This can be used as a cost function for empirical searches for good profiles.

4.2 Results

4.2.1 Binary irregular LDPC codes

To check the effectiveness of the search algorithm we searched for good binary irregular codes. These can be compared to the results published by Luby *et al.*. Figure 4.2 shows that the empirical search helps us find binary irregular codes that are significantly better than those found with Luby *et al.*'s analytical method.

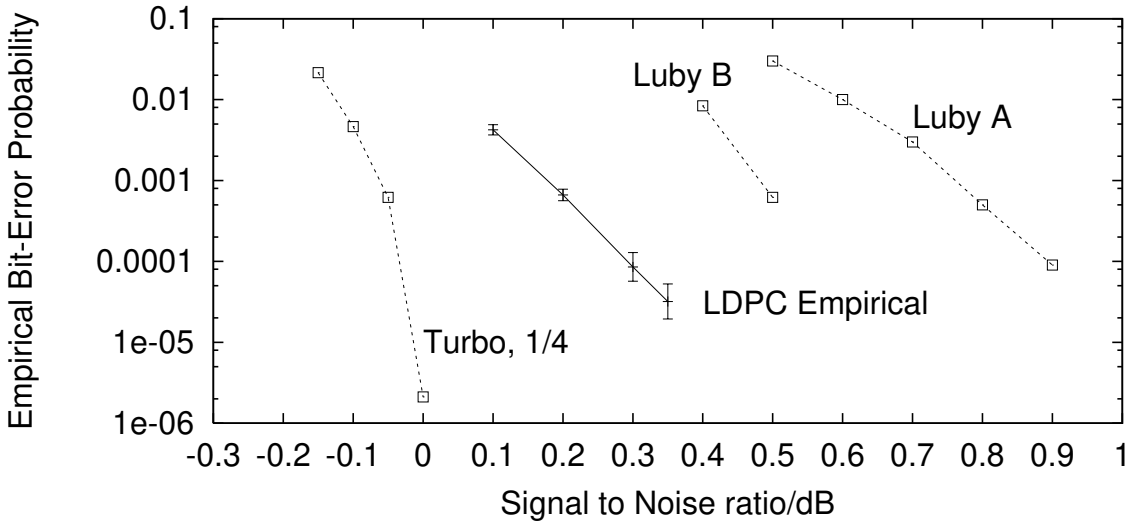


Figure 4.2: Comparison of rate 1/4 irregular binary codes. From right to left: LDPC code reported by Luby *et al.*, blocklength 16000; LDPC code reported by Luby *et al.*, blocklength 64000; LDPC with parameters found using empirical search, blocklength 16000; JPL turbo code [45], blocklength 65536.

4.2.2 Non-binary irregular LDPC codes

We find that the codes presented by Luby *et al.* in [56], with mean column weight 8, perform very poorly over higher order fields. This is not surprising since we found that regular high weight non-binary codes performed worse than the corresponding binary codes. Motivated by the Monte Carlo results for regular codes we tried irregular constructions with quite low

weight.

We present results for rate 1/4 codes. The best code we have found has a mean column weight of 3.3, but has a maximum column weight of 43. The row weight is almost uniform. The column profile is shown in table 4.1. For the parameters of several other good codes, see appendix F.

Col. Weight	2	3	7	9	11	15	19	23	29	43
fraction	0.700	0.191	0.037	0.024	0.013	0.017	0.008	0.005	0.003	0.002

Table 4.1: Parameters of good irregular code for $GF(8)$, rate 0.25

Figure 4.3 shows the performance of the best error correcting codes of rate 1/2 and rate 1/4 found using the search methods of section 4.1. We compare irregular non-binary LDPC codes with the best known turbo codes. An irregular LDPC code of rate 1/2 defined over $GF(4)$ makes fewer errors than a turbo code of similar blocklength. An irregular LDPC code of rate 1/4 defined over $GF(8)$ outperforms the comparable rate 1/4 turbo code. LDPC codes offer other advantages over turbo codes: the decoding algorithm is fully parallelisable and it is straightforward to make codes of arbitrary rate. Furthermore, all errors made by the LDPC codes in our experiments were *detected* errors – the decoder reported the fact that a block had been incorrectly decoded.

Recent results by Richardson *et al.* [74] show that binary LDPC codes with blocklengths of 10^5 and 10^6 can achieve better performance than that shown in figure 4.3. For example, they present a rate 1/2 code with blocklength 10^6 achieving a bit-error rate of 10^{-6} at $E_b/N_0 = 0.3$, just 0.1 dB from the Shannon Limit.

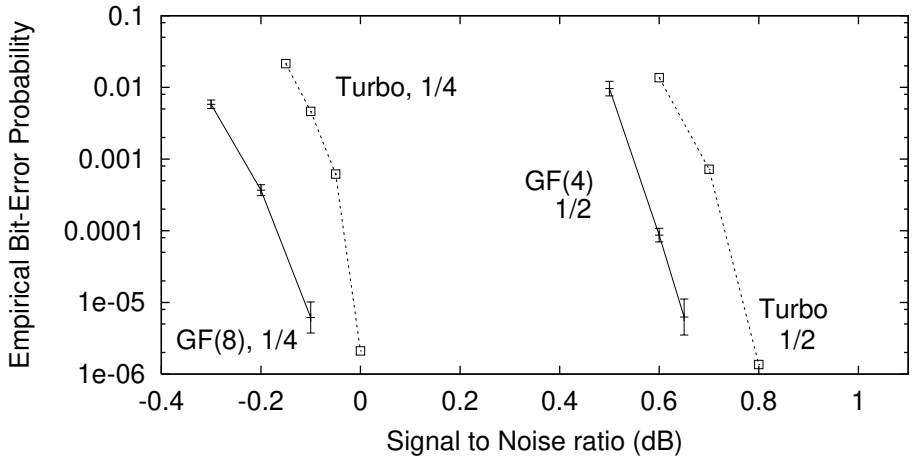


Figure 4.3: Irregular non-binary LDPC codes. Left to right: Rate 1/4 irregular LDPC over $GF(8)$, blocklength 48000 bits; Rate 1/4 JPL turbo code [45], blocklength 65536; Rate 1/2 irregular LDPC over $GF(4)$, blocklength 60000 bits; Rate 1/2 JPL turbo code, blocklength 32768. The Shannon limit for rate 1/4 codes is at -0.8 dB. For rate 1/2, the limit is 0.2 dB.

4.3 Unequal error protection

In an irregular code, symbols do not all participate in the same number of parity-checks. We use the term 'elite' to refer to those symbols which participate in the highest number of parity-checks because they receive the most information during decoding. We expect elite symbols to determine their state earlier and more accurately than 'less privileged' symbols. It is natural to ask whether, in practice, they are less likely to be incorrectly determined when decoding fails.

Comparing the bit error rate of elite symbols with that of standard symbols, we find that when decoding fails elite symbols are indeed more likely to be correctly decoded than standard symbols. For a particular LDPC code defined over $GF(8)$ with column weights 2, 3 and 13 and binary blocklength 4800, when decoding failed the symbols associated with weight-13 columns were roughly half as likely to be in error as symbols associated with weight-2 columns, for a range of noise levels. Table 4.2 shows, for several settings of the signal-to-noise ratio and for each column weight, the average fraction of symbols of that weight that were incorrectly determined when a block failed to decode.

Signal-to-noise ratio (dB)	0			0.2			0.5		
Column weight	2	3	13	2	3	13	2	3	13
Fraction of symbols in error	0.29	0.28	0.17	0.26	0.25	0.15	0.29	0.28	0.19

Table 4.2: Unequal error protection: the fraction of block decoding errors in which a symbol is incorrectly decoded depends on the weight of the column associated with that symbol. The results here are for a rate 1/4 code defined over $GF(8)$, with binary blocklength 4800. The fraction of columns with weight 2, 3 and 13 was 0.68, 0.23, and 0.09 respectively. The code was of construction IR-UL-A (see section 2.4). See also figure 4.4.

Figure 4.4 shows, for each symbol, the fraction of block decoding errors in which that symbol was incorrectly determined. Symbols 1–1086 had degree 2, symbols 1087–1453 had degree 3, and the remainder had weight 13. The elite symbols are correctly determined more often than the other symbols. Also visible is the effect of the IR-UL-A construction (see section 2.4) in which the weight-2 columns are constructed as stacked identity matrices. Columns 1–600 all participate in checks in which they are the only weight-2 columns. Conversely, all checks in which columns 1051–1086 participate contain mostly weight-2 columns. Thus, these later weight-2 columns receive the poorest information, and are most often in error.

Elite symbols receive stronger error protection, so information symbols should be assigned to the higher-weight columns of the parity-check matrix. Although this is certainly good practice, the effect on the overall bit error rate is generally small because the best codes have a modest number of elite columns.

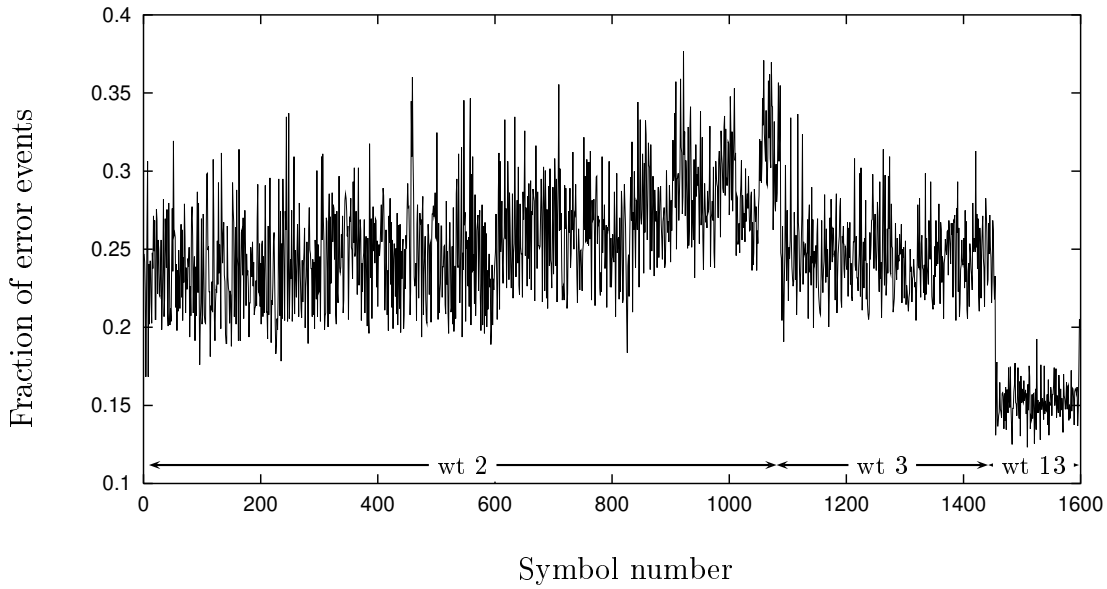


Figure 4.4: Unequal error protection for the code described in table 4.2. Horizontal axis: symbol number. Vertical axis: Fraction of times symbol was incorrectly decoded in 1700 block errors, at signal to noise ratio 0.2 dB.

Chapter 5

Behaviour of the LDPC decoding algorithm

Numeric stability isn't that important when you're guessing.

— Unknown

Until recently, the greatest justification of the iterative sum-product decoding algorithm for LDPC codes was its excellent practical performance. Gallager [25] knew that loops in the graph representing the parity-check matrix render the calculation of the posterior distribution of the transmitted symbols inexact. However, he correctly conjectured that the effect of loops on the decoding algorithm would be minimal: recently, Richardson and Urbanke [75] have proved that, for any $\epsilon > 0$, the average behaviour of the algorithm will not deviate from the loop-free behaviour by more than ϵ with probability that approaches 1 exponentially fast in the blocklength. Their measure of the behaviour of the algorithm is the fraction of symbols correctly decoded after each iteration.

In this chapter we examine the behaviour of the decoding algorithm in more detail. First, we investigate the distribution of the number of iterations required for successful decoding. We verify the findings of Mackay, Wilson and Davey [62], showing that the distribution is well described by a simple power law. Second, we examine the behaviour of the algorithm for blocks that fail to decode. We show that the algorithm does not always converge and displays several characteristics typical of a chaotic dynamical system, including sensitive dependence on initial conditions and quasi-periodic behaviour.

5.1 Required iterations for decoding

The decoder halts when a solution is found that satisfies all the parity-checks, or when an imposed maximum number of iterations has been reached. In many examples 10 iterations suffice, but sometimes over a hundred are required. By allowing a timeout of (say) 500 iterations, but halting early in the majority of cases, the occasional lengthy decoding time has a small impact on the average decoding time. Even with such a high timeout, we find

that some blocks that are declared failures can be decoded by allowing more iterations. In one case, a successful decoding was made after more than 4,000 iterations.

It is an important question, then, how many potentially successful decodings are missed by limiting the number of iterations. Decoding failures usually occur because the decoding algorithm converges to a stable configuration in which several checks remain violated. In such cases, extra iterations never lead to successful decoding. It is, however, possible for the algorithm to fail to converge to a stable state.

We can estimate the number of avoidable block errors by examining the distribution of the iterations required for successful decoding. MacKay *et al.* [62] reported that the tails of the distributions for binary codes are well approximated by a power law, $P(\tau) \sim \tau^{-\alpha}$, where $P(\tau)$ is the probability that a block requires τ iterations for successful decoding.

Figure 5.1 verifies that the distribution of iterations for non-binary codes is also well approximated by a power law. The figure shows results for an irregular code of rate 1/4 over $GF(8)$, of blocklength 4800 bits and mean column weight 3.2. Out of 10000 blocks, 9851 were decoded within 50 iterations, and there were 149 block decoding failures.

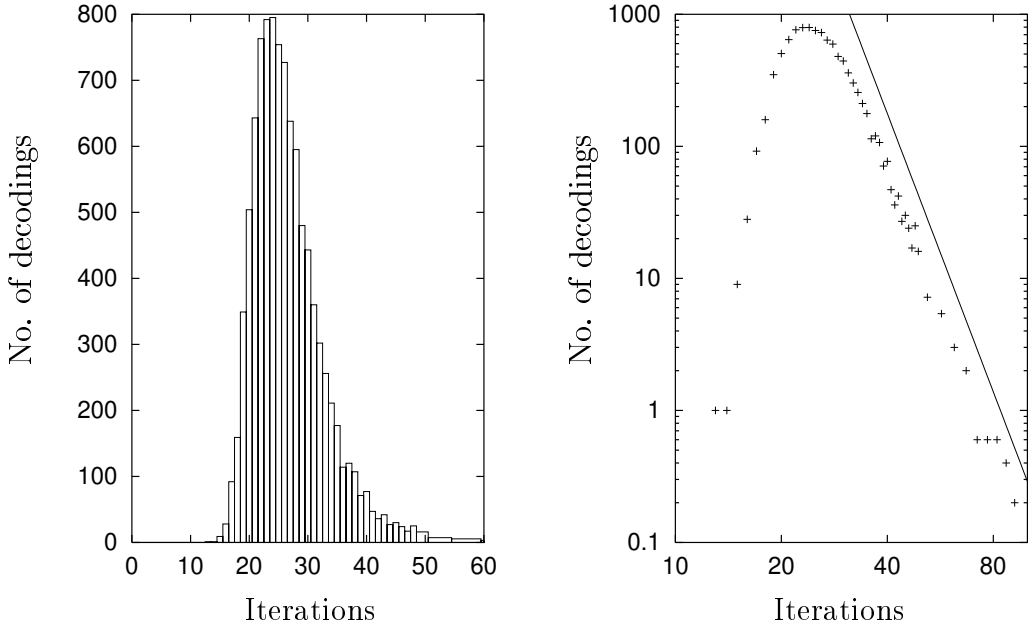


Figure 5.1: Left: Histogram of number of iterations required for decoding. Right: log-log plot of the histogram, showing that the tail is well approximated by a power law. The straight line has slope -7.0. Counts for iterations greater than 50 have been binned into intervals of size 5. Results are for a rate 1/4 code over $GF(8)$, of blocklength 4800 bits and mean column weight 3.2, at $E_b/N_0 = 0.5$.

Using the counts for up to 50 iterations, an estimate of the scaling exponent α was made: $6.7 \leq \alpha \leq 7.3$. Using this estimate of α and the observed counts for iterations 30–50, a prediction was made that between 100 and 137 of the 149 decoding failures could be avoided

by allowing up to 500 iterations. The actual number was 104, in agreement with this rough calculation. A similar calculation suggests that approximately 1 in 10^6 of the blocks that fail to decode within 500 iterations could be decoded within 10^6 iterations. This error rate is much smaller than the overall block error rate of 0.0045 at this signal-to-noise ratio. In contrast, when decoding is halted after 50 iterations, most (about 2/3) of decoding failures are due to the early stopping.

As the signal-to-noise ratio decreases, the variance in the distribution becomes larger. With the above code, at $E_b/N_0 = 1.0$ we have $P(\tau) \sim \tau^{-12}$ and at $E_b/N_0 = 0.2$ we have $P(\tau) \sim \tau^{-5}$.

5.2 Decoding failures

5.2.1 Confidence of tentative decoding

When the decoding algorithm fails to converge to a valid codeword within the maximum number of iterations, we can view the final state as giving a "best guess" of the codeword symbols. The error rate of such partial decodings is generally less than the raw channel error rate.

Further information can be gleaned from the distribution over codeword symbols calculated by the decoder after each iteration, which we refer to as the *pseudo-posterior* (equation 2.8). When decoding fails, there are usually some symbols that are determined quite confidently and others that are wholly undetermined. For example, if the decoder gets stuck in a local minimum because of a bad topology in the decoding graph (see section 2.4.2) then it is possible for most of the symbols to be correctly (and confidently) determined and for just a few incorrect symbols to prevent convergence to a codeword.

We find that the pseudo-posterior provides a good estimate of the probability that a given symbol has been correctly decoded. Figure 5.2 shows that the empirical probability that a symbol has been correctly decoded is proportional to the pseudo-posterior probability calculated by the decoder. Results are shown for codes over $GF(2)$ and $GF(8)$. There does appear to be a slight bias, especially among relatively well-determined symbols, towards over-confidence. Intuitively, we expect loops in the graph to produce feedback such that nodes "listen to their own propaganda", so becoming falsely confident.

Clearly, then, when decoding fails the pseudo-posterior provides a useful distribution over each information symbol. The pseudo-posterior could be used as input to an outer decoder if a concatenated code was in use, or as prior information for a LDPC decoder if a retransmission was requested.

5.2.2 Convergence and oscillation

The exact behaviour of the decoding algorithm is hard to predict. The average and loop-free behaviours are now well described [75], but in practice the deviation from the average

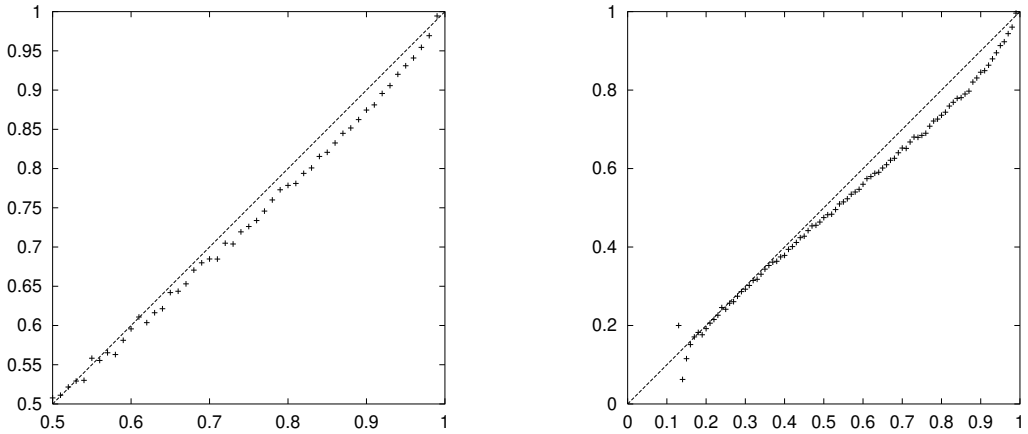


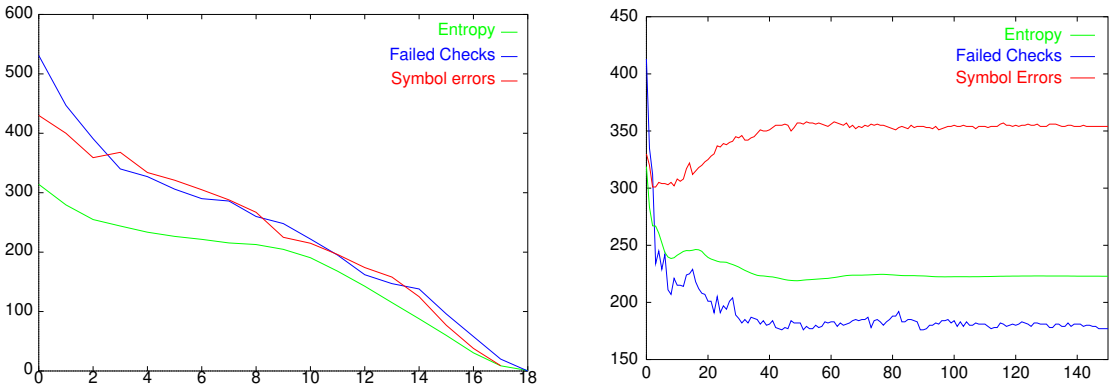
Figure 5.2: Frequency distribution showing the empirical probability that a symbol has been correctly decoded as a function of the pseudo-posterior probability of correct decoding, as calculated by the decoding algorithm. Horizontal axis: pseudo-posterior probability (100 bins). Vertical axis: empirical probability of correct decoding. (a) Binary rate 1/2 regular code, column weight 3, blocklength 2000, $E_b/N_0 = 1.4$, 1000 block errors. (b) Rate 1/2 code over $GF(8)$ as per figure 5.1, $E_b/N_0 = 0.2$, 1000 errors. The straight line shows that the pseudo-posterior provides a good estimate of the probability that a given symbol has been correctly decoded.

behaviour can be large. In this section we first show the typical behaviour, and then show some atypical behaviour. All examples are for regular rate 1/2 codes over $GF(8)$, $GF(4)$ and $GF(2)$ with a column weight of 3. The channel model used is the binary symmetric channel.

One measure of the behaviour of the algorithm is the average entropy of the tentative decoding (equation 2.8) after each iteration. We also report the number of parity-checks violated and the number of symbols incorrectly determined after each iteration.

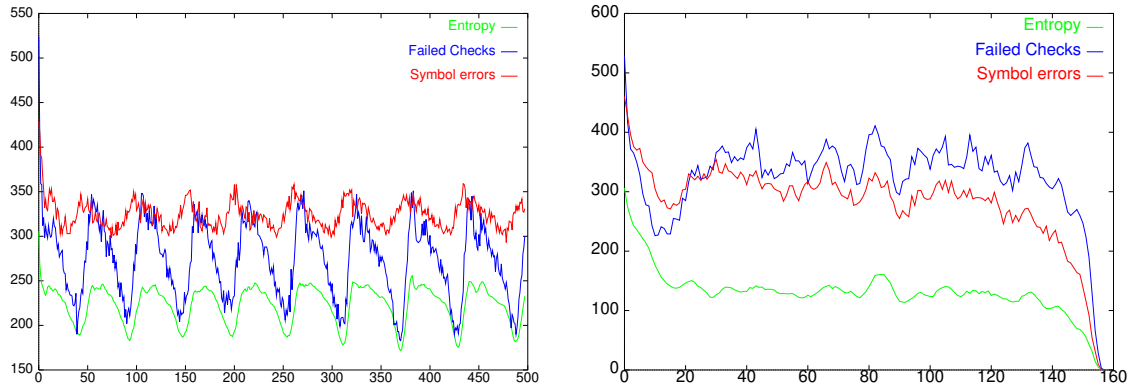
Typically, the entropy, the number of violated checks and the number of symbols incorrectly determined all decrease monotonically until a codeword is found. This behaviour is shown in figure 5.3 (a). When decoding fails, the decoder usually converges to a stable state in which several parity checks remain violated. The state is often made stable by a small number of incorrect symbols that happen to satisfy several parity-checks. Figure 5.3 (b) shows this behaviour: from iterations 5 to 40, decreasing numbers of parity-checks are violated at the expense of increasing numbers of incorrectly determined symbols. Incorrect symbols that participate in several satisfied checks become confidently (yet incorrectly) determined. This confidence encourages nearby symbols to trust their confident neighbour, in turn setting themselves to an incorrect value.

Occasionally the decoder neither makes a decoding nor settles into a steady state. Sometimes the behaviour oscillates quasi-periodically (figure 5.4 (a)), in other cases it may show no obvious pattern at all before suddenly dropping into a decoding (figure 5.4 (b)). Note that



(a) (b)

Figure 5.3: Typical behaviour of the decoding algorithm for a rate 1/2 regular column weight 3 code over $GF(8)$ of length 6000 bits. Horizontal axis: iterations. Vertical axis: number of failed checks/incorrect symbols. The entropy has been scaled by an appropriate factor to appear on the graph. (a) decoding failure, converging to a state with 170 failed checks and 350 incorrect symbols (Binary symmetric channel noise level 0.078). (b) successful decoding (Binary symmetric channel noise level 0.076).



(a) (b)

Figure 5.4: Atypical behaviour of the decoding algorithm. Horizontal axis: iterations. Vertical axis: number of failed checks/incorrect symbols. The entropy has been scaled to appear on the graph. (a) successful decoding after 157 iterations. (b) unsuccessful decoding, showing quasi-periodic behaviour. Results are for the code of figure 5.3, noise level 0.076.

these examples are not for particularly noisy channels: the block error rate is of order 10^{-3}.

For the code studied in this section, a regular rate 1/2 code over $GF(8)$ with blocklength 6000 bits and column weight 3, roughly 40% of blocks that were declared failures (at noise level 0.076) had not converged to a stable state after 500 iterations. At noise levels 0.082, this proportion has dropped to roughly 10%. We conjecture that this proportion also drops as the blocklength increases and the effect of loops in the graph is diminished, but further work is required to confirm this.

5.2.3 Evidence of chaos

The fact that oscillatory behaviour such as that shown in figure 5.4 is generally not perfectly periodic suggests that the dynamics of the decoding algorithm may be chaotic. This is not surprising given that the loops introduce feedback, which is a key component of chaotic systems. There are well known methods for examining time series for evidence of chaos. We briefly introduce the method of embedding to search for attractors.

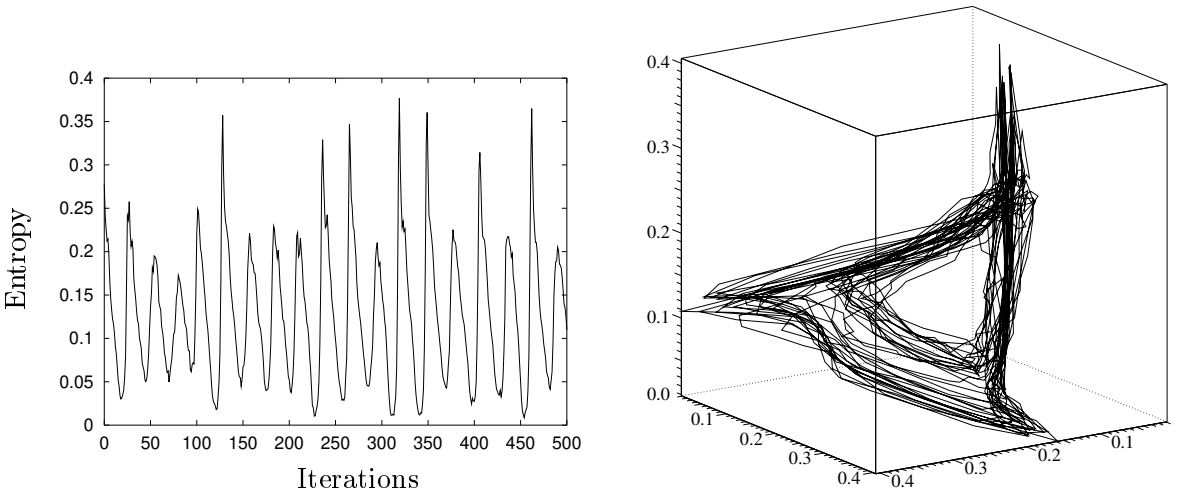
In a dynamical system, an *attractor* is a set of points in the phase space that is invariant under the dynamics. Nearby points, in a so-called basin of attraction, approach the attractor asymptotically in the course of dynamic evolution. *Strange attractors* are bounded regions of phase space having zero measure in the embedding phase space and a fractal dimension. Trajectories within a strange attractor appear to move around unpredictably.

As shown by Packard *et al.* [68], a phase-space picture can be constructed from the observation of a single dynamical variable. The heuristic idea is to use n independent observations of a single dynamical variable to produce a single point in an n dimensional phase space. We will use the evolution of the entropy of the tentative decoding as our 'dynamical variable'. To construct a phase space picture, we choose a *delay time*, t, and an *embedding dimension*, d. From a time series $\{x_i\}$ a sequence $\{s_i\}$ of vectors of dimension d is constructed: $s_i := (x_i, x_{i+t}, \ldots x_{i+(d-1)t})$.

Figure 5.5 (a) shows a short section of a typical quasi-periodic time series of the entropy of the tentative decoding, similar to that shown in figure 5.4. Figure 5.5 (b) shows the embedding of the time series in three dimensions, using a delay time of 8. Note that $d \times t = 3 \times 8$ is approximately the quasi-period of the time series. By choosing $t = 8$, we minimise the mutual information of the three time series created from the original shifted by 0, t and $2t$ steps, as suggested by Fraser *et al.* [24] .

The embedding reveals a strange attractor. There are two main loops to the attractor, corresponding to the two main amplitudes of the oscillating entropy. For this code, time series generated by many different runs (*i.e.* noise vectors) gave rise to similar attractors. The attractor is a feature of the dynamical system defined by the decoding algorithm applied to this particular code. Different codes possess different attractors, and a given code generally possesses several distinct attractors. Of course, in regimes where decoding is reliable the basins of attraction of such attractors are very small, if they exist at all.

Finally, we show that LDPC decoding exhibits another characteristic of chaotic systems,



(a) (b)

Figure 5.5: (a) Evolution of the entropy of the tentative decoding, for a binary rate 1/2 code of length 504, noise level 0.07. The dynamics are quasi-periodic. (b) Embedding of the time series in three dimensions, using a delay time of 8. The embedding reveals a strange attractor of fractal dimension approximately 2.7 (calculated using the Grassberger – Procaccia method [30]).

namely sensitive dependence on initial conditions. During simulations of LDPC decoding, it was noticed that subtly different results were produced by two different machines. Eventually it was discovered that very occasionally a block would be decoded on one machine and not the other, despite identical initial conditions.

On closer examination, the decoding dynamics were seen to diverge after a large number of iterations (the exact reason for the discrepancy does not concern us here, but we suspect slight differences in the system implementation of the numerical functions). For the first 400 iterations, the entropy agreed to 4 significant figures. After about 450 iterations the decodings diverged, and were essentially uncorrelated after 550 iterations (figure 5.6). Both machines eventually managed to decode the block, one taking 3429 iterations, the other 4589, as shown in figure 5.7.

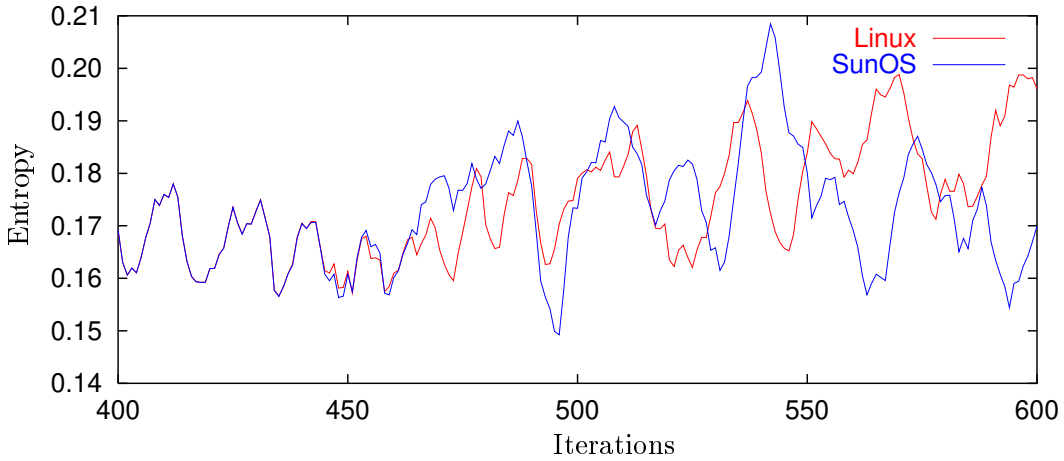


Figure 5.6: Sensitivity to initial conditions: evolution of the entropy of the tentative decoding as calculated by two different machines. After 450 iterations the two calculations have produced slightly different results. These quickly diverge. Code over $GF(8)$ as per figure 5.1, noise level 0.085.

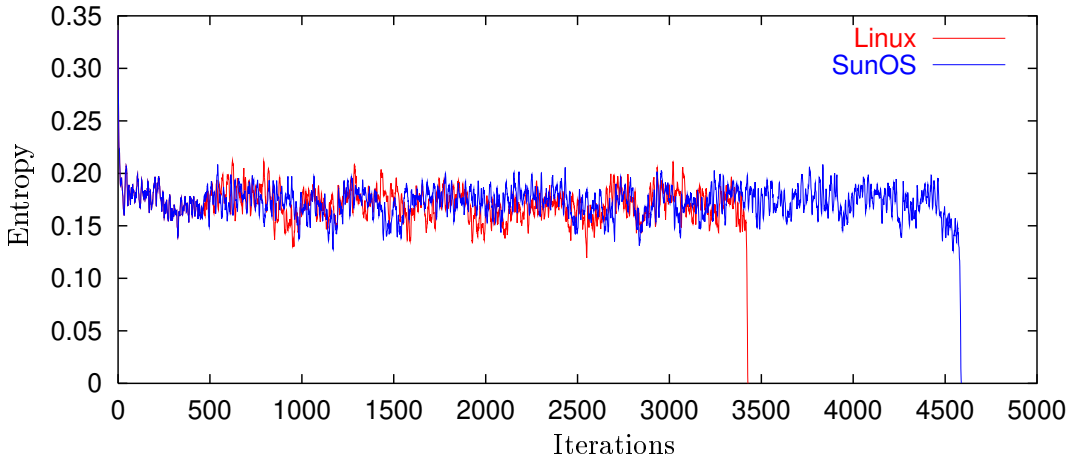


Figure 5.7: Long decoding times. Blocks for which the decoding algorithm does not settle down into a stable state may eventually decode after many iterations. The results here are for the same block as in figure 5.6.

Chapter 6

Codes for correcting synchronisation errors

If I have not seen as far as others, it is because giants were standing on my shoulders.

— Hal Abelson

Most error-correcting codes (including LDPC codes) are designed to correct *substitution* errors, in which the transmitted symbol is replaced by a different received symbol. The decoder must be told (or be able to identify) the beginning of each block in order to decode. Thus, ensuring synchronisation of the encoder and decoder is a crucial element of communication systems.

Channels that make *synchronisation* errors, in which bits may be deleted from and inserted into the transmitted message, disrupt the synchronisation. Alternative coding methods must be used to protect against such errors.

In this chapter we discuss synchronisation errors and review the known constructions of codes that can correct synchronisation errors. Finally, we discuss some probabilistic frameworks that have been proposed for use with insertion/deletion channels. These frameworks are closest in spirit to the new 'watermark' codes for insertion/deletion channels which we present in the next chapter.

6.1 Synchronisation errors

The term 'synchronisation error' to denotes any error that changes the length of the received message. In general, synchronisation errors result in the insertion or deletion of symbols.

Slip, or frame synchronisation error, is one type of synchronisation error. When denoting synchronisation errors as 'slip', it is implicitly assumed that the block is not affected by insertions or deletions but that the start of the block has merely been incorrectly determined. In such cases it is common to use a code that allows the block boundaries of uncorrupted blocks to be identified, and to discard blocks in which insertion/deletion errors occur.

More generally, we may consider the blocks in which bits have been inserted or deleted. To communicate reliably we must use a code such that distinct codewords may still be distinguished after corruption by a certain number of insertion, deletion and substitution errors. Levenshtein [51] introduced a metric to define a distance between strings corrupted by insertions, deletions and substitutions. The *Levenshtein distance* (also known as the *edit distance* [89] or Damerau-Levenshtein metric [14, 50]) $L(\mathbf{x}, \mathbf{y})$ between two strings $\mathbf{x}$ and $\mathbf{y}$ is defined as the smallest number of insertions, deletions and substitutions required to transform $\mathbf{x}$ into $\mathbf{y}$. Table 6.1 shows the strings with Levenshtein distance 1 from the string "01101".

Deletion	Substitution	Insertion of '0'	Insertion of '1'
1101	11101	001101	101101
0101	00101	010101	011101
0111	01001	011001	011011
0110	01111	011010	
	01100		

Table 6.1: All strings with Levenshtein distance 1 from the string "01101".

Levenshtein proved that a fixed length code can correct up to s insertion, deletion and substitution errors if and only if $L(\mathbf{x}, \mathbf{y}) > 2s$ for any two different codewords $\mathbf{x}$ and $\mathbf{y}$. This result assumes the codeword boundaries are known. As table 6.1 suggests, Levenshtein distance constraints are considerably more restrictive than the more common Hamming distance constraints suitable for substitution errors. For example, a code of length 15, which can correct up to 3 insertion/deletion errors (*i.e.* no substitution errors) has a maximum size of 10 codewords [39]. In comparison, a code of length 15 to correct up to 3 substitution errors may contain up to 32 codewords [63].

Levenshtein also calculated asymptotic bounds on the number of codewords in a code capable of correcting up to s insertion/deletion errors as the code length $n \to \infty$. Ullman [86] produced bounds on the rate of a code capable of correcting $n\alpha$ insertion/deletion errors in a block of length n, as $n \to \infty$. In the next chapter we present codes capable of achieving a small probability of error at rates above the lower bounds of Ullman and Levenshtein.

6.2 Codes for synchronisation

6.2.1 Run-length limited codes

The most common method of maintaining synchronisation is to avoid making synchronisation errors in the first place. In channels such as magnetic recording channels and high bit-rate networks, synchronisation errors arise from small discrepancies between the transmitter and receiver clocks. Left unchecked, such discrepancies accumulate. For example, if the receiver's clock is running fast too many bits are detected from the channel, causing insertion errors.

Codes for data storage [64] make use of certain run-length limited sequences, also known as (d, k) constrained sequences, in which the run of '0' symbols between each '1' must have length at least d and at most k. In magnetic storage systems, zeroes and ones are stored by polarising the medium in one of two opposite directions every clock cycle. By limiting the length of a pulse, the receiver is guaranteed to detect a transition in polarisation within k clock cycles. Assuming the clock drifts by less than half a cycle every k cycles, the receiver can reliably resynchronise the clock.

6.2.2 Slip-correcting codes

Comma-free codes were originally proposed by Crick, Griffith and Orgel in 1957 [13] as a possible encoding of protein sequences in DNA sequences. They were introduced into the coding community by Golomb, Gordon and Welch [29] the following year.

Comma-free codes require that if $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_n$ are codewords, then none of the overlaps

$$x_i \ldots x_n y_1 \ldots y_{i-1} \quad 1 < i \leq n$$

is a valid codeword. Hence, if the incorrect block framing is chosen (assuming no insertion, deletion or substitution errors), the data does not form valid codewords. Comma-free codes allow a receiver to resynchronise following a synchronisation error, with a delay of at most two blocks of data, although blocks corrupted by insertions and deletions are not recoverable.

Gilbert [28] investigated a subset of comma-free codes which he called *prefix synchronised* codes. In such codes a prefix sequence is chosen that appears at the start of each codeword, and is forbidden from appearing within the codeword. For example, the following four 5-tuples:

$$11000 \quad 11001 \quad 11010 \quad 11011$$

form a comma-free code, with the prefix sequence '110'. The rate of this code is $\log_2(4)/5 = 0.4$. Gilbert showed that, for long blocklengths and suitable prefix sequences, the rate of prefix synchronised codes can approach that of optimal comma-free codes. Asymptotically, the redundancy required to correct synchronisation slip is very small: the rate of optimal comma-free codes approaches $1 - \log(n)/n$.

Variable-length comma-free codes that proved efficient for encoding natural-languages were examined by Neumann [66]. Eastman [21] proved that optimal comma-free codes of odd length n have the number of codewords conjectured by Golomb, Gordon and Welch.

Several extensions to comma-free codes allow both substitution errors and synchronisation slip to be corrected. Hatcher [36] constructed a subset of a Neumann variable-length code which could resynchronise and correct a limited number of substitution errors. Stiffler [81] and Tavares and Fukada [84] proposed adding a constant vector to binary cyclic codes (a con-

struction known as *coset codes*) to add synchronisability to known error-correcting codes.

We emphasize that none of the codes considered in this section treats synchronisation errors as the loss or gain of bits. Instead, regaining synchronisation is treated as the problem of locating the block boundaries.

6.2.3 Insertion and deletion correcting codes

The first known codes capable of correcting a codeword corrupted by insertions and deletions were introduced by Sellers [78]. The approach taken was to insert a synchronising *marker* sequence (*e.g.* '001') into a burst-error-correcting code at periodic intervals. By searching for the markers in their expected positions, the decoder could detect single insertions or deletions between successive markers. If synchronisation errors were detected, a bit would be inserted or deleted (as appropriate) midway between the markers, giving rise to a burst of substitutions errors of maximum length $\lceil j/2 \rceil$, where j was the separation of the markers. The burst-error-correcting code was then used to then decode the resulting word. Multiple synchronisation errors could be corrected by using longer markers, at the expense of added redundancy.

A similar idea was employed by Bours [9] to construct array codes (commonly used for the correction of burst errors [7]) capable of correcting bursts of insertions and deletions, and uniformly distributed substitution errors. Bours constructed an array such that the rows were comma-free codewords and the columns were Reed-Muller codewords. The comma-free codewords encoded both row numbers and information bits, so that after synchronisation errors the ungarbled comma-free codewords could be assigned to the correct row of the array. Some empirical results using a bursty insertion/deletion/substitution channel were given.

Much of the work on insertion/deletion correcting codes has focused on the number-theoretic constructions employed by Levenshtein [51]. These codes impose congruence constraints on the binary codewords $x = x_1 x_2 \ldots x_n$:

$$\sum_{i=1}^{n} i x_i \equiv a \pmod{m} \tag{6.1}$$

for $m > n$. These codes were first introduced by Varshamov and Tenengolts [87] to correct asymmetric errors (where the probability of $0 \to 1$ and $1 \to 0$ substitution errors are not equal). Levenshtein proved that these codes could correct single insertions and deletions under the assumption that the codeword boundaries were given, and gave a straightforward decoding algorithm.

The codeword separation restriction was removed by Calabi and Hartnett [10], who studied number-theoretic codes that, in every t blocks, could correct either one synchronisation error or $(t-2)$ substitution errors. These codes were extended by Tanaka and Kasai [82] to allow several synchronisation errors to be corrected in the absence of substitution errors. They also deduced Levenshtein distance constraints on codes for the simultaneous correction of

synchronisation and substitution errors and gave a greedy algorithm for constructing such codes. No efficient decoding algorithm is known for these codes.

The first non-binary codes capable of correcting single insertions or deletions were introduced by Tenengolts [85]. A binary string was associated with each codeword. These binary strings satisfied a congruence constraint similar to (6.1), and the codewords satisfied a second congruence constraint. Tenengolts showed the codes could be put in systematic form and provided a simple decoding algorithm.

Helberg [39] investigated subcodes of Levenshtein's codes that satisfy runlength or DC-free constraints[1]. Several new number-theoretic constructions of (binary) multiple insertion and deletion correcting codes were presented, although they had rather low rates as they relied on partitioning the space of codewords.

6.3 Probabilistic frameworks

Channels causing insertions, deletions and substitutions are of practical interest in many disciplines other than coding theory. Prominent fields using models related to insertion/deletion channels include speech recognition, biological sequence analysis, optical character recognition and automated spelling correction.

Solutions to such problems commonly employ a probabilistic source model (often a hidden Markov model) and an observation sequence corrupted by insertions, deletions and substitutions. The decoding problem is to find the most likely source string, or marginal probabilities of some sort.

Bahl and Jelinek [3] developed a general channel model where input sequences could give rise to output sequences of varying length. An efficient dynamic programming procedure for calculating $P(Y|X)$ was given, where X and Y were the input and output of the channel, respectively. $P(Y|X)$ was calculated by marginalising over all possible sequences S of channel insertions, deletions and substitutions: $P(Y|X) = \sum_S P(Y, S|X)$.

Bouloutas, Hart and Schwartz [8] developed similar methods for the case of an unknown input sequence produced by a known finite state machine. They pointed out that such an algorithm would be suitable for decoding convolutional codes over insertion/deletion channels, but did not report results. Convolutional codes proved unsuitable for communicating over such channels [34]. Hart and Bouloutas later extended their work to more general channels [35], allowing an arbitrary number of channel error patterns. More efficient decoding methods for the above problems were developed by Amengual and Vidal [1] by introducing a combination of computational tricks.

The use of hidden Markov models and probabilistic methods for biological sequence alignment has spread widely in the bioinformatics community since being introduced by Anders Krogh, David Haussler and others [38, 49]. At the heart of the models are in-

[1] A DC-free binary word is one which has the same number of zeroes as ones, *i.e.* a word of length n with Hamming weight $n/2$.

sertion/deletion/substitution models of varying complexity. For good reviews of the field, see [20, 37].

Holmes and Durbin [40] made explicit connections between the dynamic programming and hidden Markov model approaches, and provided numerical and analytical estimates of the alignment *fidelities* – in our context this corresponds to the fraction of bits correctly synchronised by the decoder. They used coupled hidden Markov models to compare two corrupted descendants of a single ancestor DNA sequence.

From the dynamic programming community, the analysis of Hwa and Lässig [42, 19] is relevant. Drawing on ideas from statistical physics they investigate the deviation of the inferred alignment from the true alignment for a simple insertion/deletion model. As the insertion/deletion probability increases, they find a phase transition between regimes where asymptotically the fraction of correctly aligned positions is greater than zero, and where asymptotically this fraction is zero. In the context of communication codes, this result suggests that there are non-trivial insertion/deletion channels for which we can expect, almost surely, to retain bounded synchronisation loss with a probabilistic decoder.

In the next chapter we introduce a block code suitable for channels that make insertion, deletion and substitution errors. It bears no relation to the number theoretic constructions. In some ways it can be viewed a generalisation of Sellers' codes, with a probabilistic decoder similar to that of Bahl and Jelinek.

Chapter 7

Watermark codes

Do not seek to follow in the footsteps of the men of old; seek what they sought.

— Matsuo Basho

In this chapter we introduce a new approach to coding for insertion/deletion channels. The design incorporates a significant random element in the construction, and a probabilistic decoding algorithm. These codes can correct hundreds of synchronisation errors in a single block of size 4000 bits.

We break down the decoding problem into two parts. First we identify, as best we can, where the insertions and deletions occurred. We do this using an inner code that is resilient to insertions and deletions. Once this inner code has been decoded we can expect to have several residual errors from mis-identified synchronisation errors and the uncorrected substitution errors. An outer code then corrects the remaining errors.

7.1 Outline of watermark codes for synchronisation

The key idea in these codes is to provide a carrier signal or 'watermark' for the decoder. If there are synchronisation errors then the decoder identifies discontinuities in the carrier signal and deduces the presence of the errors. This idea is similar to that of marker codes [48, 78] which insert a known pattern at regular intervals. Although marker codes allow synchronisation, sections that suffer from any insertion or deletion errors are declared as erasures because there is no synchronisation capability between markers. The approach described in this section distributes the 'marker' smoothly throughout the block, allowing for continuous probabilistic synchronisation. Marker codes can be seen as a special case of watermark codes.

Imagine, first, that the receiver knows the message being transmitted. In this case, the problem of identifying where the insertions and deletions occurred is easier to formulate. This problem is closely related to that of generating alignments of biological sequences and more generally to the notion of string similarity and edit distance. Good reviews of the literature are available [76, 50, 33, 90]. We can solve the problem efficiently using dynamic programming.

Take, as an example, the following transmitted and received strings, where we label the positions with letters:

```
Index:       ABCDEFGHIJKLMNOPQRSTUVWXYZ
Transmitted: 01001010011100101111010001
Received:    01001010111011011111010001
```

In this case the receiver might conclude that there has been a deletion at position 'H' or 'I', a substitution in position 'N', and an insertion somewhere between positions 'Q' and 'U'. Of course, we must assign cost functions to insertion, deletion and substitution events to qualify this conclusion.

If the sender were to make a few substitutions to the agreed string before transmission then the receiver could still retain synchronisation but would infer extra substitution errors. Substitutions in an agreed string thus provide a means of communicating information. Watermark codes exploit this fact. The message to be communicated is converted to a sparse string which is then added (modulo 2) to an agreed '*watermark*' string. The decoder receives the modified watermark string corrupted by channel insertions, deletions and substitutions, and attempts to identify the position of synchronisation errors. Once this is done the decoder can recover a noisy version of the sparse string.

As in the example above, the precise position of the synchronisation errors cannot be determined exactly. Hence, even when the channel makes no substitution errors, there will be some uncertainty in the inferred sparse string. We protect the sparse string with an outer code that allows us to recover the original message.

Before describing watermark codes in more detail, we will introduce a general model of an insertion/deletion channel with substitutions. Like LDPC codes, we expect our codes to be useful over a large class of channels, subject to appropriate modifications of the decoding algorithm.

7.2 Channel model

We consider a binary channel parameterised by three quantities: P_s, P_i and P_d, which control the rate of substitutions, insertions and deletions. We imagine the transmitted bits t_i entering a queue, waiting to be transmitted by the channel. At each channel use, one of three events occurs. With probability P_i a random bit is inserted into the received stream. With probability P_d the next queued bit is deleted. With probability $P_t = (1 - P_d - P_i)$ the next queued bit is transmitted *i.e.* put into the received stream, with a probability P_s of suffering a substitution error. Under this model, the burst length of an insertion event has a geometric distribution. We will generally impose a maximum insertion length I.

In the discussion that follows we will assume that $P_d = P_i$, so that the expected length

of the received string is equal to the transmitted length (except for a correction due to I of order P_i^I).

One possible representation of the channel is shown in figure 7.1. Note that the 'Transmit' state makes a fraction P_s of substitution errors.

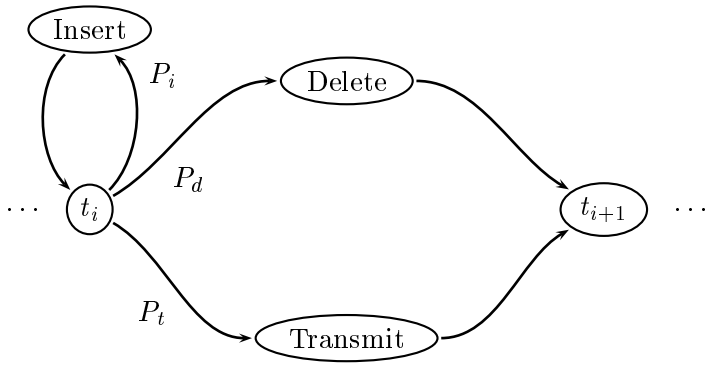


Figure 7.1: Insertion/Deletion channel with probabilities P_i, P_d and P_t of insertions, deletions and transmissions.

Alternatively, if we impose an upper limit I on the number of insertions that can be made during a single event, we can write out the channel model explicitly. Each input bit t_i gets mapped to between 0 and $I+1$ output bits, according to the probabilities in table 7.1:

Emission	Probability
$[\phi]$	P_d
$[t_i]$	$P_t(1-P_s)$
$[t_i \oplus 1]$	$P_t P_s$
$[u, \phi]$	$P_d \frac{P_i}{1-(P_i)^I}$
$[u, t_i]$	$P_t(1-P_s)\frac{P_i}{1-(P_i)^I}$
$[u, (t_i \oplus 1)]$	$P_t P_s \frac{P_i}{1-(P_i)^I}$
$\vdots$	$\vdots$
$[\underbrace{u, \ldots, u}_{I \text{ times}}, \phi]$	$P_d \frac{(P_i)^I}{1-(P_i)^I}$
$[\underbrace{u, \ldots, u}_{I \text{ times}}, t_i]$	$P_t(1-P_s)\frac{(P_i)^I}{1-(P_i)^I}$
$[\underbrace{u, \ldots, u}_{I \text{ times}}, (t_i \oplus 1)]$	$P_t P_s \frac{(P_i)^I}{1-(P_i)^I}$

Table 7.1: Explicit emission probabilities for insertion/deletion channel model with maximum insertion burst length I. ϕ is the null string and u represents a uniformly distributed bit. Hence the probability assigned to the string $[uu\phi]$ is divided evenly between the 4 possible strings.

7.3 Construction

The construction of watermark codes is outlined in figure 7.2. Using a LDPC code we first encode our message $\mathbf{m}$ of length K_L into a vector $\mathbf{d}$ of length N_L. We use LDPC codes defined over the field $GF(q = 2^k)$ because they can easily utilise the soft information provided by the inner code.

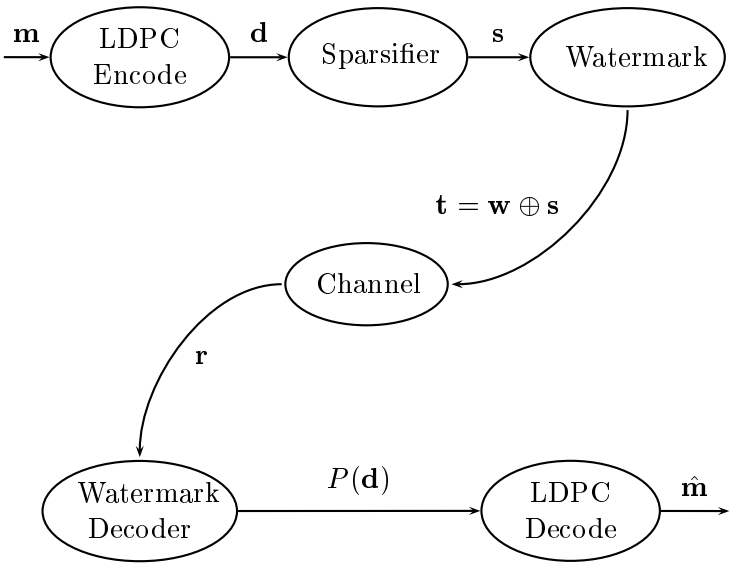


Figure 7.2: Overview of watermark code construction

The LDPC codeword $\mathbf{d}$ is then transformed into a sparse vector $\mathbf{s}$. In general, we could vary the sparseness of $\mathbf{s}$ from symbol to symbol. Here we describe the simplest approach, where the sparseness of $\mathbf{s}$ is uniform. Alternative irregular constructions are discussed in section 7.8.

We choose some $n > k$ and map each symbol of $\mathbf{d}$ into one of the 2^k *sparsest* vectors of length n using a lookup table. We denote the *mean density* of the sparse vectors by f. Section 7.7.2 discusses how n and k can be chosen to minimise f. The overall blocklength of the code is thus $N := nN_L$ and the rate of the code is $\frac{K_L}{N_L} \times \frac{k}{n}$.

The sparse vector $\mathbf{s}$ is added (modulo 2) to the watermark string $\mathbf{w}$ (which is known to both encoder and decoder) to produce the transmitted vector $\mathbf{t}$. The watermark vector provides the synchronisation capability of the code so it is important that local translations of the watermark vector are easily identified. Useful choices include random and run-length limited sequences.

The watermark (inner) decoder processes the noisy received vector $\mathbf{r}$ and produces a distribution over symbols of the LDPC codeword $\mathbf{d}$. This distribution initialises the LDPC decoding algorithm.

7.4 Decoding

We model the received vector as having been produced by a Hidden Markov Model (HMM), *i.e.* we ignore the correlations between substitutions that are present because of the construction of $\mathbf{s}$ and the structure in the outer LDPC code word. We now describe a dynamic programming method to synchronise the noisy received stream, based on the standard Hidden Markov Model (HMM) decoding methods introduced in appendix D.

We define the *excess*, x_i, at position i to be the (number of insertions) − (number of deletions) made by the channel from the start of the block to the point where bit t_i is ready to be transmitted. Explicitly, if bit t_{i-1} is not deleted then it appears in the received stream as r_{i-1+x_i}. The sequence $\{x_i\}$ will form the hidden states of our Markov model. We limit the maximum allowed value of the excess: $|x_i| < x_{\max}$, although this condition can be relaxed (see section 7.7.1). Typically, $x_{\max}$ is several times larger than I, the maximum insertion burst length. For example, we might choose $x_{\max} = 50$ and $I = 10$. $x_{\max}$ limits the size of the HMM, and hence the decoding complexity.

The synchronisation task can be conveniently represented on a lattice. In figure 7.3 the received vector is placed along the top and the watermark vector is placed down the side. In this representation, deletions are indicated by a vertical line, insertions by horizontal, and transmissions by diagonals. A path that passes through the point (i, j) associates received bit r_j with watermark bit w_i (or with a bit inserted between w_i and w_{i+1}. A diagonal (transmission) step from (i_1, j_1) to (i_2, j_2) defines the excess $x_{i_2} = (j_2 - i_2)$.

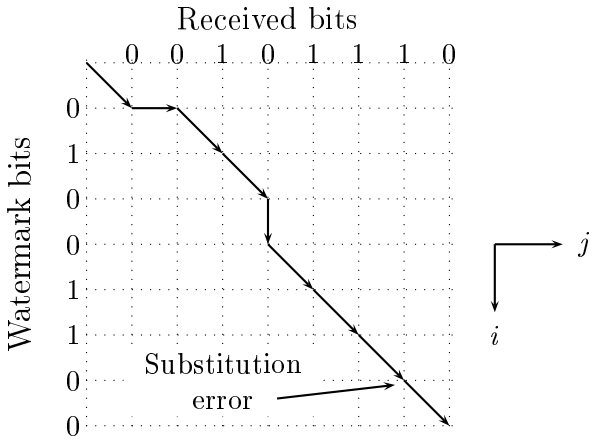


Figure 7.3: Lattice representation of synchronisation. The solid line shows one possible alignment of the received bits with the watermark bits. There is an insertion (horizontal line) after the first bit, a deletion of the fourth bit, and a substitution error of the seventh bit.

7.4.1 Hidden Markov Model

In this section we make explicit the HMM that we use for identifying the position of insertion/deletion errors.

The model is parameterised by the channel parameters P_i, P_d, P_s, by the mean density of the sparse vectors f, by the maximum allowed excess $x_{\max}$ and by the choice of watermark vector $\mathbf{w}$. It will be useful to define $P_f := f(1-P_s) + (1-f)P_s$, the probability that a bit transmitted by the channel is not equal to the corresponding watermark bit. The parameters will collectively be denoted $\mathcal{H}$.

We use the channel model outlined in section 7.2, imposing a maximum insertion burst length I. Furthermore, we limit the number of states in the model by restricting the maximum allowed excess (but see section 7.7.2 for alternatives).

States x_i in the model take values in the alphabet $\mathbf{X} := \{-x_{\max}, -x_{\max}+1, \ldots, x_{\max}\}$. Each transmitted bit produces from 0 to $I+1$ received bits. Hence, if $x_i = a$ then x_{i+1} must take values in $\{\max(a-1, -x_{\max}), \ldots, \min(a+I, x_{\max})\}$, representing the range from a deletion event (change in excess $= -1$) to an insertion of I bits followed by transmission of t_i (change in excess $= I$), subject to the constraints on the extreme values of excess.

The transition matrix entry P_{ab} contains the probability $P(x_{i+1} = b | x_i = a)$. Explicitly:

$$P_{ab} = \alpha_a \times \begin{cases} 0 & : \quad (b-a) < -1, \quad (b-a) > I \\ P_d & : \quad (b-a) = -1 \\ \alpha_I P_i P_d + P_t & : \quad b = a \\ \alpha_I ((P_i)^{b-a+1} P_d + (P_i)^{b-a} P_t) & : \quad 0 < (b-a) < I \\ \alpha_I (P_i)^I P_t & : \quad (b-a) = I \end{cases} \tag{7.1}$$

where $\alpha_I = 1/(1-(P_i)^I)$ and α_a is a normalising factor to ensure $\sum_b P_{ab} = 1$. It is 1 unless $a = -x_{\max}$ or $a > x_{\max} - I$.

The transition from $(x_i = a)$ to $(x_{i+1} = b)$ results in the emission of $(b-a+1)$ bits. Loosely, $P_{ab} \equiv \alpha P_d + \beta P_t$, where α is the probability of making $(b-a+1)$ insertions, and β is the probability of making $(b-a)$ insertions. We can write down $Q^i_{ab}(s)$, the probability the transition $(x_i = a)$ to $(x_{i+1} = b)$ emits the string s, in terms of the coefficients α and β. Note that the Q values are dependent on the watermark bit w_i:

$$Q^i_{ab}(s) = \begin{cases} \alpha/2^{b-a+1} + \beta(1-P_f)/2^{b-a} & : \quad s \in \{0,1\}^{b-a} \times w_i \\ \alpha/2^{b-a+1} + \beta P_f/2^{b-a} & : \quad s \in \{0,1\}^{b-a} \times (w_i \oplus 1) \end{cases} \tag{7.2}$$

This definition of Q approximates the sparse bits of $\mathbf{s}$ as independent and identically distributed.

7.4.2 Synchronisation

The objective of the synchronisation algorithm is to output a distribution over vectors $\mathbf{d}$ that can be used by the LDPC (outer) decoder to produce a valid decoding. We treat each q-ary symbol d_i of $\mathbf{d}$ as independent and calculate the posterior probability of d_i given the received vector $\mathbf{r}$ and the HMM:

$$P(d_i|\mathbf{r},\mathcal{H}) = \frac{P(\mathbf{r}|d_i,\mathcal{H})P(d_i|\mathcal{H})}{P(\mathbf{r}|\mathcal{H})} \tag{7.3}$$

$P(d_i|\mathcal{H})$ is a uniform distribution. The likelihood factor $P(\mathbf{r}|d_i,\mathcal{H})$ is calculated via a forward/backward algorithm using the HMM described in the previous section. Define the forward quantity $F_j(y) := P(r_1,\ldots,r_{j-1+y}, x_j = y|\mathcal{H})$, the probability that the excess at position j is y and that the first $(j-1+y)$ bits emitted by the channel agree with $\mathbf{r}$. Similarly, the backward quantity $B_j(y) := P(r_{j+y},\ldots|x_j = y,\mathcal{H})$ denotes the probability of emitting the tail of $\mathbf{r}$ given an excess of y at position j. Then

$$P(\mathbf{r}|d_i,\mathcal{H}) = \sum_{x_{i_-},x_{i_+}} F_{i_-}(x_{i_-})P(\mathbf{r}^0, x_{i_+}|x_{i_-}, d_i,\mathcal{H})B_{i_+}(x_{i_+}) \tag{7.4}$$

where $i_- = n\times i$ and $i_+ = n\times(i+1)$, so d_i is encoded into the n sparse bits $(s_{i_-},\ldots,s_{i_+-1})$. $\mathbf{r}^0$ denotes the received bits $(r_{(i_-+x_{i_-})},\ldots,r_{(i_++x_{i_+}-1)})$.

Decoding proceeds along the lines detailed in appendix D, with slight changes to allow both for emissions of variable length and the fact that we do not know the position (in the received stream) of the end of the block (*i.e.* x_N). The latter is solved by running the forward pass several (*e.g.* 5) multiples of $x_{\max}$ beyond the expected position of the block boundary, then initialising the backward pass from the final forward values.

Variable length emissions are easily dealt with. Rather than having a single state dependent emission matrix, we have the distributions Q given in equation 7.2. Thus the forward quantities are calculated as follows (compare equation D.7):

$$F_j(y) = \sum_{a=y-I}^{y+1} F_{j-1}(a)P_{ay}Q_{ay}^{j-1}(r_{j-1+a},\ldots,r_{j+y-1}) \tag{7.5}$$

Calculation of the backward quantities is handled similarly.

The bits of the sparse vector $\mathbf{s}$ are not independent: the sparse bits $(s_{i_-},\ldots,s_{i_+-1})$ are determined by the value of d_i. For each possible value of d_i we calculate the likelihood $P(\mathbf{r}^0, x_{i_+}|x_{i_-}, d_i,\mathcal{H})$ by fixing $(s_{i_-},\ldots,s_{i_+-1})$ according to d_i and performing a forward pass between x_{i_-} and x_{i_+}. In this pass $w_i \oplus s_i$ is known, so all substitution errors are due to the channel. Hence the emission probabilities are as per equation 7.2 with w_i replaced with $w_i \oplus s_i$ and P_f with P_s.

7.4.3 LDPC decoding

The posterior distributions (7.3) are used to initialise the LDPC decoder. The uncertainty in the distributions is unevenly spread between regions with well-determined synchronisation and those corrupted by insertions and deletions. This results in a strongly time-dependent effective channel for the LDPC code.

It is interesting to compare the LDPC decoding performance as a function of the entropy of the input distributions to that of previously studied channels. We find that the error-correction performance is similar. This motivates the use of the entropy of the distributions $P(d_i|\mathbf{r}, \mathcal{H})$ as a measure of the expected LDPC code cut-off rate for successful decoding, given the channel parameters. This is explored in section 7.5.3 below.

7.4.4 Multiple blocks

If the decoder is not told the position of the block boundaries, that information must be estimated so that the received bits belonging to the next block can be identified. The most likely value of the excess at the end of the block, $\hat{x}_{N+1} := \text{argmax}_y F_{N+1}(y)B_{N+1}(y)$, is used for this purpose.

We move the zero point of our received stream by $(N + \hat{x}_{N+1})$ positions. The quantities F are shifted accordingly:

$$F_1(y) \leftarrow \alpha \times \begin{cases} F_{N+1}(y + \hat{x}_{N+1}) & : \quad -x_{\max} - \hat{x}_{N+1} \leq y \leq x_{\max} - \hat{x}_{N+1} \\ 0 & : \quad \text{otherwise} \end{cases} \tag{7.6}$$

where α is a normalising constant such that $\sum_y F_1(y) = 1$. The decoder can then proceed to synchronise the next block.

As the channel gets noisier, synchronisation becomes more difficult and the error in $\hat{x}_{N+1}$ increases. If the accumulated errors exceed $x_{\max}$ then the decoder fails with a catastrophic loss of synchronisation. It is important to protect against such catastrophes by providing a mechanism for detecting and correcting gross loss of timing. This is discussed in section 7.5.2.

7.5 Limits of watermark codes

Little is known about the capacity of the insertion/deletion channel under consideration. Bounds [86] on the rates of codes capable of correcting a number of insertion/deletion errors proportional to their length indicate that it is certainly non-zero if P_d and P_i are less than 0.05. Taking an empirical approach we will make conjectures of the achievable communication rates for watermark codes over insertion/deletion channels.

In the following sections we investigate the properties of watermark codes. We start by analysing the ability of watermark codes to identify the position of insertion/deletion errors in the received stream. We show that catastrophic loss of synchronisation is highly unlikely for watermark codes over channels whose insertion/deletion rates are low enough to

admit decoding with block error rates below 10^{-1}. Secondly, we examine the quality of the information passed to the LDPC decoder. This will allow us to conjecture lower bounds on the capacity of specific insertion/deletion channels.

7.5.1 Limits of synchronisation

In this section we measure the ability of watermark codes to resynchronise the received stream. One measure of this ability is the fraction of positions in which a decoder correctly determines the excess (synchronisation drift). By examining how this fraction depends on the channel parameters, we can estimate the maximum density of sparse vector, $\mathbf{s}$, for which reliable synchronisation is possible.

In identifying insertion/deletion errors, the watermark decoder makes no distinction between watermark vector substitutions which are due to the channel and those which are due to $\mathbf{s}$. Hence, for given P_i and P_d, the synchronisability of the received stream depends only on the effective substitution rate P_f, which takes account of the channel substitutions and the density of the sparse vector $\mathbf{s}$.

Consider the model sketched in figure 7.4 using the lattice notation. This is equivalent to the channel of figure 7.1 with no maximum insertion burst length. Transmissions suffer a fraction P_f of substitutions. This representation lends itself to an efficient implementation of a Viterbi decoder in integer arithmetic, making it practical to run experiments over moderately long blocklengths with less restrictive maxima on the excess insertion/deletion counts. As usual, we will take $P_d = P_i$.

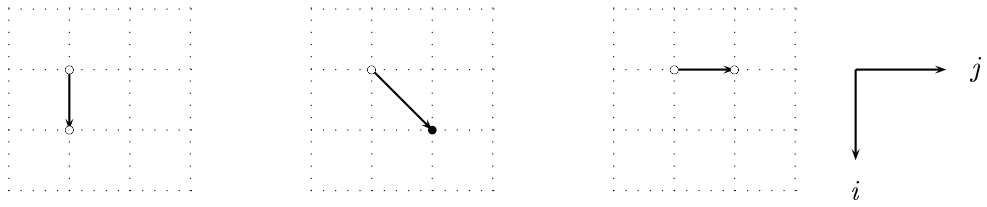


Figure 7.4: A simple insertion/deletion channel. Received bits are in the horizontal direction, watermark bits in the vertical. Costs are given by the log-likelihood of the transitions, as shown. There is a further cost for transmissions (solid circle) arriving at a node (i, j) which is $-\log(1-P_f)$ (or $-\log(P_f)$) if the received bit r_j matches (or doesn't match) the watermark bit w_i.

To estimate the range of channel parameters for which synchronisation errors remain bounded, the following method was used. Blocks of 30,000 random watermark bits were created. These were passed through the simulated channel to produce received bits. A Viterbi

decoder was used to find the maximum likelihood path through the lattice (*i.e.* sequence of channel transitions). The decoder was initialised with the true synchronisation (at the top left of the lattice) but was not given the end point.

The fraction of positions in which the Viterbi path was synchronised with the true path, averaged over multiple blocks, was recorded. The maximum allowed loss of synchronisation (200) was much larger than the expected drift in an unconstrained model (see appendix C.4), so this limit had a negligible effect on the results.

In figure 7.5 the fraction of synchronised positions is plotted as a function of the insertion/deletion and substitution probabilities. It is clear from figure 7.5 that when the insertion/deletion probability is low, good synchronisation is possible even for quite high substitution levels. For example, with $P_d = 0.04$ and $P_f < 0.3$, over half the received bits were correctly resynchronised. This is of importance when designing high-rate watermark codes: to achieve higher rates we must use denser $\mathbf{s}$ which leads to higher effective substitution probabilities P_f (see section 7.7.2). For example, with an LDPC code over $GF(16)$, to construct a rate 4/5 watermark code requires a sparse vector of density 0.3125. That is to say, even if the channel makes no substitutions the effective substitution probability P_f is over 30%. Figure 7.5 shows that it is possible to construct high rate watermark codes that can maintain synchronisation.

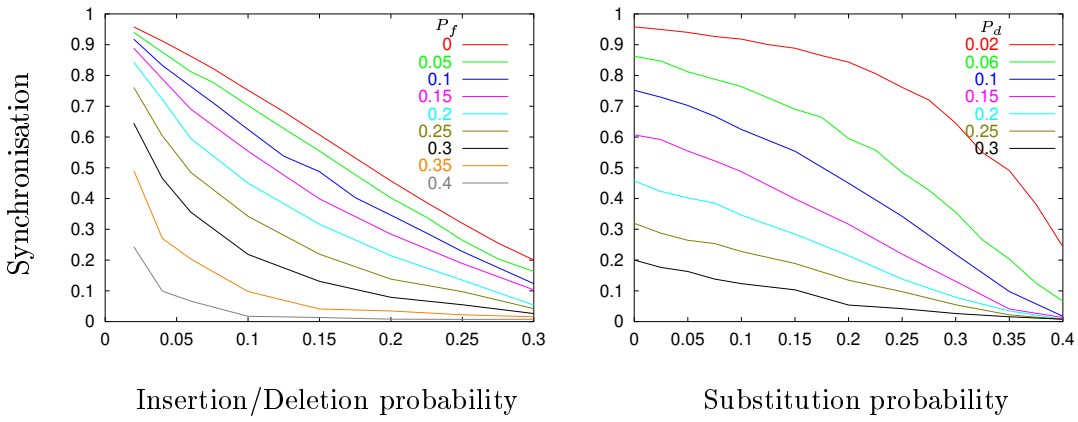


Figure 7.5: Left: fraction of synchronised bits as a function of insertion/deletion probability, for several settings of P_f. Right: fraction of synchronised bits as a function of substitution probability, for several settings of P_d, P_i.

7.5.2 Probabilistic resynchronisation

When decoding watermark codes, it may occasionally be necessary to recover lost synchronisation or to start decoding a stream with unknown synchronisation. In this section we describe the method used to recover synchronisation, and find that maintaining synchronisation is

usually much easier than making a successful decoding.

We use the fact that LDPC codes very rarely make undetected errors. If consecutive blocks fail to decode we should suspect global loss of synchronisation. To retrieve the situation we perform sum/product decoding to find the maximum *a posteriori* synchronisation for the current block. The method is essentially the same as that described in section 7.4.4 for finding the end of a block during normal decoding; the difference lies in the initial conditions for the forward/backward passes.

Define the synchronisation $S(r_i)$ of the received bit r_i to be the index (mod N) of the transmitted bit corresponding to r_i. That is, $S(r_i) = 0$ if r_i marks the start of a block, and $S(r_i) = N/2$ if r_i is associated with the middle of a transmitted block. The synchronisation is trivially related to the excess: $S(r_i) = n \Leftrightarrow x_n = i - n$.

To find the correct synchronisation, we proceed as follows. Define some arbitrary position in the received stream to have index 0. We assign uniform priors to $S(r_0)$ and $S(r_{2N})$:

$$P(S(r_0) = n) = \begin{cases} 1/N & \text{if } -N/2 \leq n < N/2 \\ 0 & \text{otherwise} \end{cases} \tag{7.7}$$

$$P(S(r_{2N}) = n) = \begin{cases} 1/N & \text{if } -N/2 \leq n < N/2 \\ 0 & \text{otherwise} \end{cases} \tag{7.8}$$

We want to find the position in the received stream of the start of the next block. The priors (7.7), (7.8) define initial conditions for the forward/backward passes. For example, (7.7) imposes $F_n(-n) = 1/N$ for $-N/2 \leq n < N/2$ (see section 7.4.2). We perform forward/backward decoding from position 0 to $2 \times N$ in the received stream, to infer the most likely value of x_N:

$$P(x_N = y|\mathbf{r}) \propto F_N(y)B_N(y). \tag{7.9}$$

The method is shown graphically in figure 7.6.

When synchronisation is lost, the decoder can usually restrict attention to a narrow range of possible synchronisations centred on the last known value, rather than using the broad priors described above.

Figure 7.7 shows the logarithm of the quantity $F_N(y)B_N(y)$ as a function of the distance from the inferred start of the next block (*i.e.* $N + y$) to the true position. The results are for a block of length 2000 bits with $P_s = 0$ and a sparse vector $\mathbf{s}$ of density 0.125. As we shall see in section 7.6.4, with these parameters a rate 1/2 LDPC code over $GF(8)$ can communicate reliably for $P_d < 0.05$. In contrast, the correct synchronisation is easily identifiable for $P_d = 0.1$ (the approximate cut-off for LDPC codes of rate 1/10). Even for $P_d = 0.2$, although there is usually a small error in the identification of the block boundary, the errors are bounded to within ≈ 20 by the exponential drop in posterior probability away from the true block boundary.

It is also possible to use a Viterbi algorithm along a narrow corridor in the lattice. If the

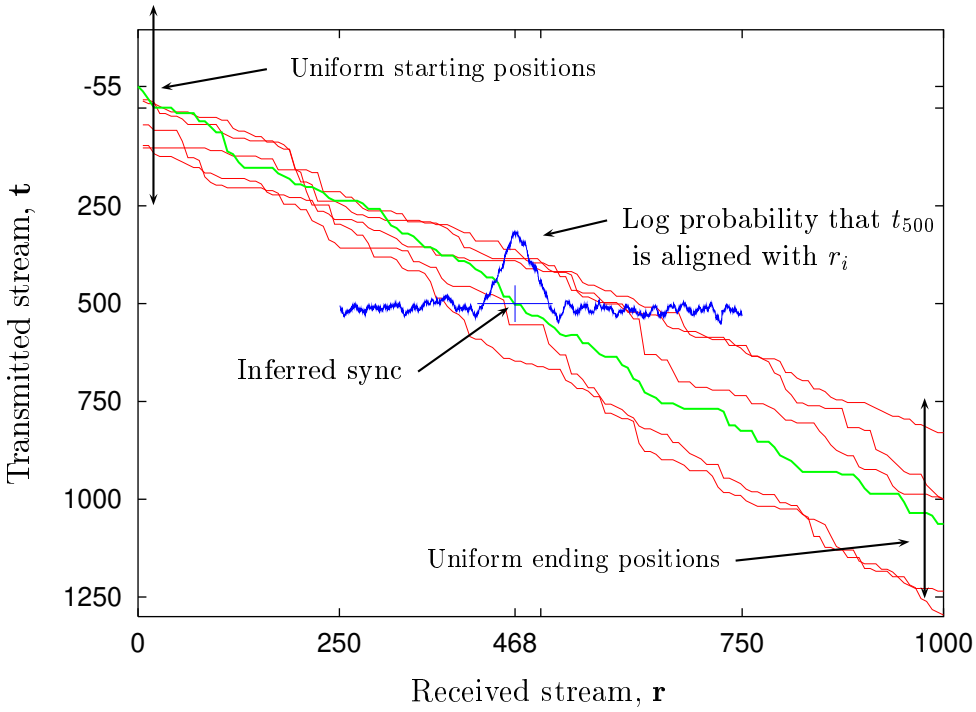


Figure 7.6: Resynchronisation (schematic diagram). We choose an arbitrary zero point in the received stream. We assign a uniform prior $P(S(r_0) = n)$ for $-N/2 \leq n < N/2$, where $S(r_0)$ is the synchronisation of the zero point – *i.e.* the position within a transmitted block to which r_0 belongs. Similarly, we assign a uniform prior to $S(r_{2N})$. Using two blocks of received data, the forward-backward algorithm is used to calculate the posterior probability of the position in the received stream of the start of the next block. In this example the blocklength is 500 and the assumed zero point is close to the true start of block ($S(r_0) = -55$). The true path is shown in green. The red paths represent samples from the prior. The position of the end of the block is correctly identified as position 468 in the received stream.

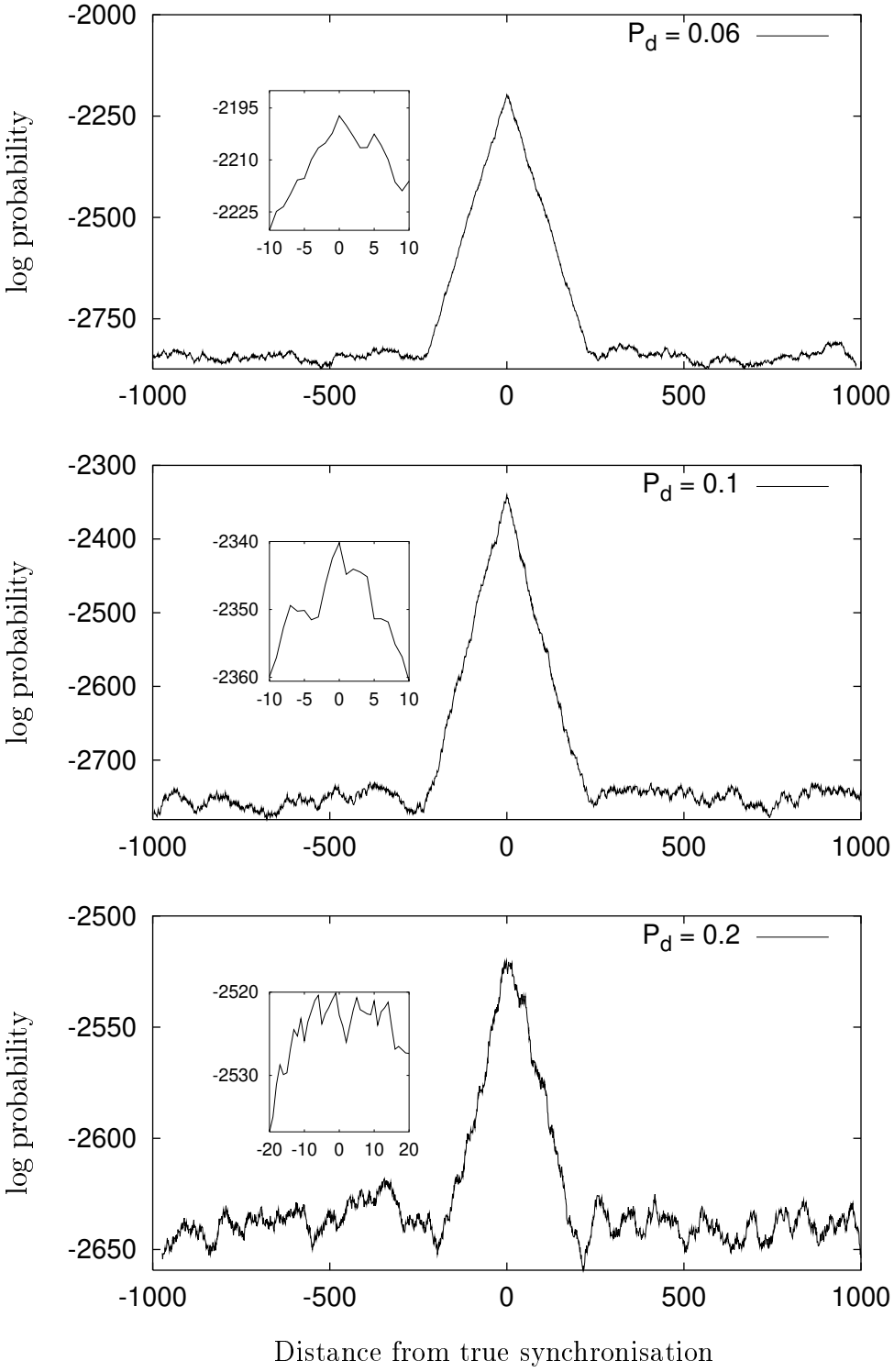


Figure 7.7: Using a uniform prior over possible synchronisation points (*i.e.* start of block), the forward-backward algorithm is used to calculate a posterior distribution. Horizontal axis: distance from inferred start of block to true start of block. Vertical axis: log probability (not normalised) of the inferred start of block. Results are shown for several choices of P_d, increasing from top to bottom. The correct synchronisation is easily determined (with a small margin of error), even when the probability of insertion and deletion is 20%. Density of $\mathbf{s} = 0.125$, channel substitution probability $P_s = 0$, blocklength 2000 bits.

corridor does not contain the true path, the probability of the Viterbi path will be similar to that for a random transmitted stream. On the other hand, if the corridor does contain the true path, the Viterbi path will have increased probability, which is easily detectable. Such methods might be useful for a rough estimate of the synchronisation, but we do not expect the estimates of the Viterbi algorithm to be as stable as the sum/product approach, as the latter calculates a maximum *a posteriori* value of the synchronisation. Viterbi paths are often not 'typical' paths, in that the number of insertions, deletions and transitions may vary significantly from the expected and true counts. In detecting the end of a block it is more appropriate to use the probabilities from the sum/product decoder. However, especially at lower noise levels, the Viterbi results are generally good enough for resynchronisation. Appendix E shows typical errors made by the Viterbi path.

There are noticeable differences between the pointwise synchronisation estimates obtained with the sum/product algorithm and the corresponding Viterbi path. The pointwise synchronisation estimates made by the sum/product algorithm are usually closer to the true synchronisation. In difficult sections the probability density is typically shared between the true path and a competitor. This is illustrated graphically by figure 7.8 in which the probability density generated by the sum/product decoder is compared with the Viterbi lattice path and the true path.

7.5.3 Limits of decoding

Synchronisation is only half the battle. To make a decoding, the LDPC decoder needs a reasonable estimate of the q-ary symbols of the LDPC codeword $\mathbf{d}$. In this section, the factors affecting the quality of this estimate are investigated. Lower bounds on the capacity of insertion/deletion channels are conjectured.

The dominant factors affecting the quality of the estimate passed to the LDPC decoder are the density f of the sparse vector $\mathbf{s}$, and the order of the field $GF(q)$. The density f is a function of q and n, the number of sparse bits used to represent a q-ary symbol. f limits the accuracy with which the watermark decoder determines the synchronisation, while $\log(q)/n$ is the rate of the watermark code. For a given density f we would like to make the rate $\log(q)/n$ as large as possible.

Figure 7.9 plots the entropy per symbol, scaled by $1/\log(q)$, of the distribution over $\mathbf{d}$ calculated by the watermark decoder (equation 7.3) as a function of the watermark code rate $\log(q)/n$. We now discuss the reasons for the observed dependencies shown in the figure.

For a fixed watermark code rate $\log(q)/n$, the density f decreases (albeit non-monotonically) as $\log(q)$ increases. For increasing q and n, $\log(q)/n \to H_2(f)$. Hence, for a given watermark code rate, we can make synchronisation easier simply by increasing q (and n) appropriately. The utility of this approach is limited by the accompanying tradeoff in complexity addressed in section 7.7.2.

A more important advantage of increasing q and n comes from the fact that the uncertainty in the bits of $\mathbf{s}$ is highly correlated over short distances. As an extreme example, take the

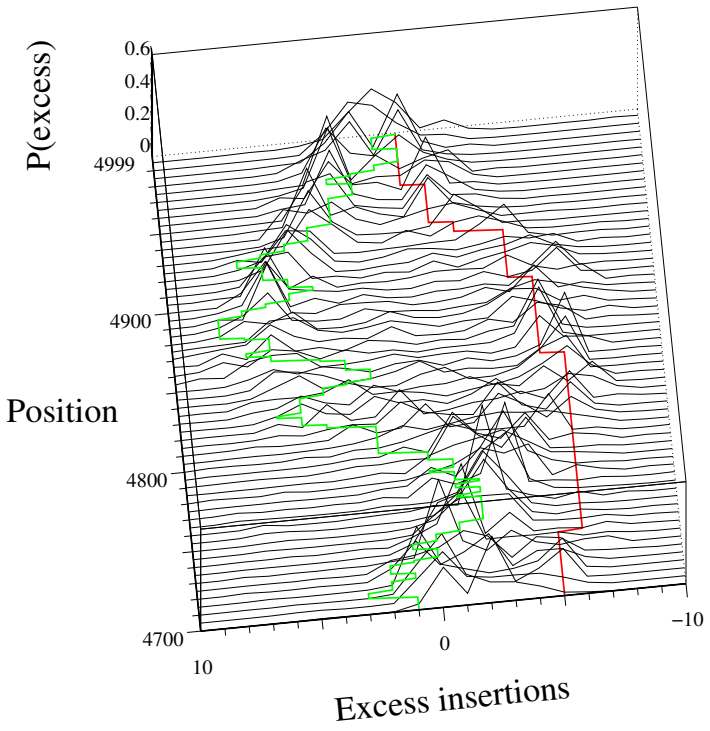


Figure 7.8: Comparison of Viterbi path (red) and true path (green) to the pointwise densities calculated by the sum/product algorithm for a short section of a block. Insertion/deletion probabilities: $P_d = P_i = 0.1$; substitution probability: $P_f = 0.35$.

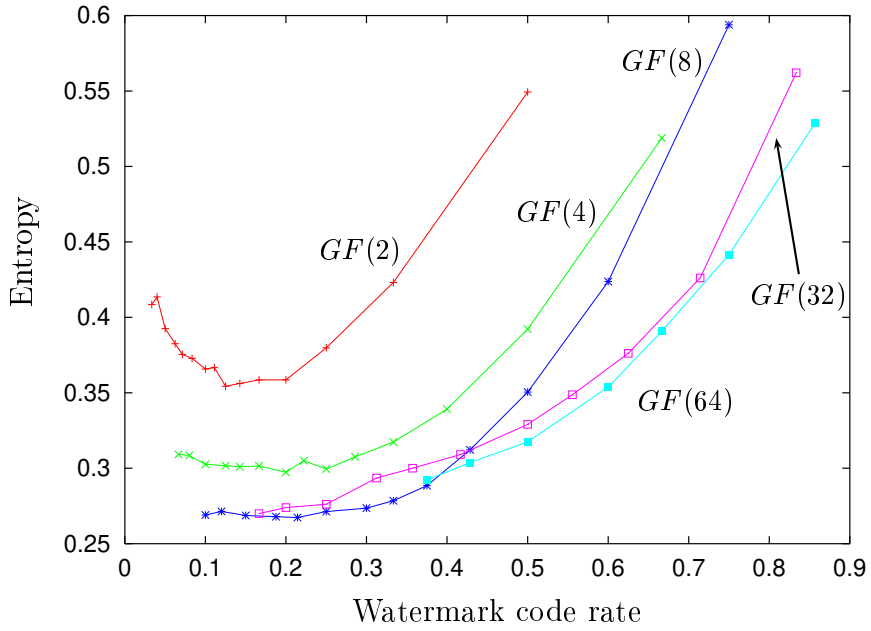


Figure 7.9: Average entropy per 'bit' of the vector $\mathbf{d}$ returned by the watermark decoder, as a function of the watermark code rate. Channel parameters: $P_d = P_i = 0.05$, $P_s = 0$.

received string '0101010101...', and assume that the decoder has determined that there is a 50% chance the first symbol was a spurious insertion, but the following stream is reliable. The entropy of any single bit is then 1 (maximum), but the joint entropy of groups of n following bits is clearly also 1, for all n. In this artificial example, there are only two possible values each q-ary symbol can take, regardless of the size of q. More generally, a region of uncertainty of a characteristic width surrounds the position of insertion/deletion events. Outside these regions symbols are more strongly determined.

Keeping q fixed, increasing n reduces the density of the sparse vector $\mathbf{s}$ – asymptotically the density of s is $(q-1)/nq$. For moderate values of n this reduction improves decoding, but in the limit of large n the increased number of synchronisation errors present in regions of length n outweighs the benefit of sparser $\mathbf{s}$. Eventually the entropy increases with increasing n rendering the estimation of the q-ary symbols more difficult. In the limit of low rate (large n) the entropy increases. This effect is shown most clearly in figure 7.9 for $GF(2)$ and $GF(4)$.

The entropy of the distribution of the vector $\mathbf{d}$ can also be used to infer a lower bound on the capacity of the channel using watermark codes. This entropy depends on the choice of n and q, and the channel parameters. Figure 7.10 shows the average entropy of $\mathbf{d}$ as a function of the channel insertion and deletion parameters. Results have been averaged over 50 blocks.

The entropy surface identifies upper bounds on the rate of the LDPC code for reliable communication, for a given channel and watermark code. Given the success of LDPC codes for Gaussian channels, we conjecture that this upper bound is approachable in the limit of

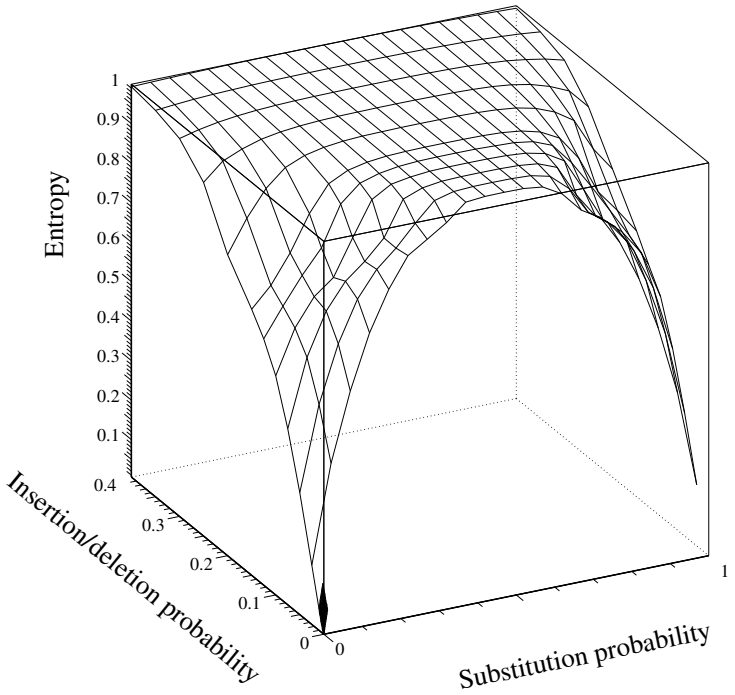


Figure 7.10: Entropy of the LDPC codeword $\mathbf{d}$, as a function of the channel parameters $P_d = P_i$, P_s. Outer LDPC code defined over $GF(8)$, blocklength 666. q-ary symbols mapped to groups of 7 sparse bits ($n = 7$). The per-symbol entropy has been divided by $\log(q)$.

long blocklengths and well designed LDPC codes. If this conjecture holds, then by subtracting the entropy in figure 7.10 from 1 we obtain lower bounds on the capacity, C_{eff}, of the *effective channel* seen by the outer code after watermark decoding, as shown in figure 7.11.

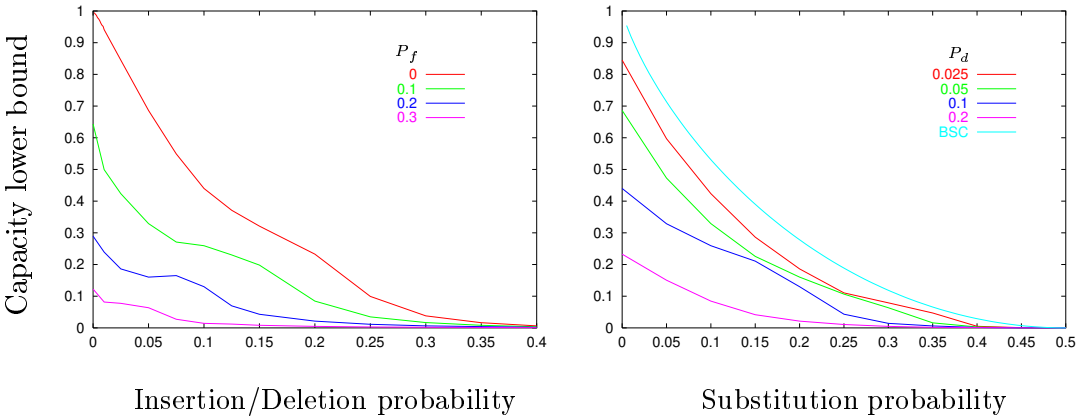


Figure 7.11: Conjectured lower bounds on the capacity of the effective channel seen by the outer code after watermark decoding. Watermark code: blocklength 4662, rate 3/7, sparse vector density of 0.125, outer code over $GF(8)$. Left: lower bound as a function of the insertion/deletion probability, for several settings of the substitution probability. Right: lower bound as a function of the channel substitution error probability, for several settings of the insertion/deletion probability. The watermark code has no substitution error-correction capability, so the channel approximates a binary symmetric channel as $P_d \to 0$. The capacity of the BSC is also shown in the right-hand panel.

A lower bound on the capacity of the insertion/deletion channel proper may be obtained by multiplying C_{eff} by the rate of the watermark code. The rate of the watermark code is 3/7 in this example, so the lower bound is necessarily less than this. Figure 7.12 compares the conjectured lower bound on capacity with the upper and lower bounds calculated by Ullman [86]. The empirical lower bound is consistent with the calculated upper bound, and improves the known lower bound considerably. Note that the calculated upper bound is not considered to be tight.

7.6 Experiments and results

7.6.1 Simulation method

The performance of watermark codes was tested by simulation as follows. A LDPC code of length N_L over $GF(q = 2^k)$ and a sparseness parameter $n > k$ were chosen. A binary pseudorandom watermark vector $\mathbf{w}$ of length $N_L \times n$ was created, as was a lookup table of q of the sparsest vectors of length n.

A long random string $\mathbf{d}$ of q-ary symbols was generated. This string was translated via the

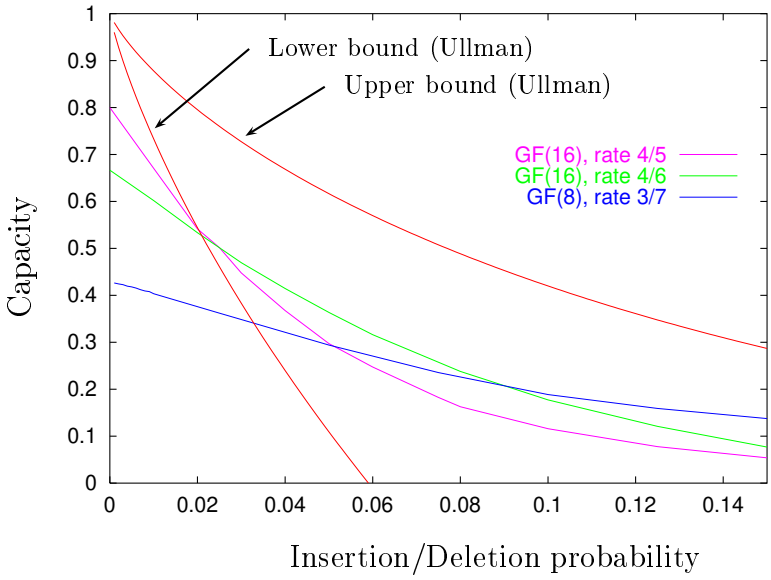


Figure 7.12: Bounds on the capacity of an insertion/deletion channel with no substitutions. Comparison is made between the upper and lower bounds calculated by Ullman [86] and empirical conjectured lower bounds obtained using the following watermark codes: rate 3/7 with outer code over $GF(8)$, rate 4/6 with outer code over $GF(16)$ and rate 4/5 with outer code over $GF(16)$.

lookup table to a sparse string $\mathbf{s}$ and added to the watermark vector (which repeated every nN_L bits) to produce the transmitted string. The watermark decoder returned a distribution for each q-ary symbol. Let $\hat{d}_i : GF(q) \to \mathbb{R}$ be the distribution returned for symbol d_i. The i'th noise symbol for the LDPC code was defined to be $n_i := d_i - \text{argmax}_a(\hat{d}_i(a))$ and the distribution for the i'th noise symbol was then $\hat{n}_i(a) := \hat{d}_i(a + d_i - n_i)$ (arithmetic in $GF(q)$). The LDPC decoder then performed syndrome decoding using the distributions $\{\hat{n}_i\}$ for each symbol i. In this way, we avoid explicitly generating LDPC codewords when simulating watermark codes.

For the first block simulated, synchronisation was given. A continuous stream of received bits was supplied to the decoder, and for subsequent blocks the decoder was fully responsible for retaining synchronisation. Typical synchronisation at the end of blocks was ± 1. For decoding to be possible, synchronisation errors must remain bounded. Very occasionally, synchronisation errors accumulated and several blocks in a row were incorrectly decoded. In all cases the approach outlined in section 7.5.2 successfully regained the lost synchronisation.

For higher rate LDPC codes (*e.g.* rate 0.7) synchronisation was not lost once in many thousands of blocks. In this regime fewer errors are tolerated by the LDPC decoder and hence, for channels over which communication is possible, synchronisation is very well determined. Errors are typically caused by the displacement of insertion/deletion events by one or two positions.

7.6.2 Choice of outer code rate

We have already seen how the rate of the watermark code and the field order q affects the accuracy with which the LDPC codeword is determined by the watermark decoder. In figure 7.13 the effect of varying the rate of the LDPC code, for fixed overall code rate and length, is studied. Within the constraints there is an optimum choice. Lower rate LDPC codes can cope with poorer estimation of $\mathbf{d}$ but result in denser sparse vectors $\mathbf{s}$. Conversely, higher rate LDPC codes allow sparser $\mathbf{s}$ but need $\mathbf{d}$ to be well determined by the watermark decoder. The best choice of LDPC code rate is not independent of the channel parameters. This can be understood by recognising that in the vicinity of insertion/deletion events there is a greater uncertainty in the vector $\mathbf{d}$ – causing a short burst of errors. For higher insertion/deletion rates the ability of a lower rate LDPC code to combat the increased number of these bursts causes the optimum LDPC code rate to drop slightly.

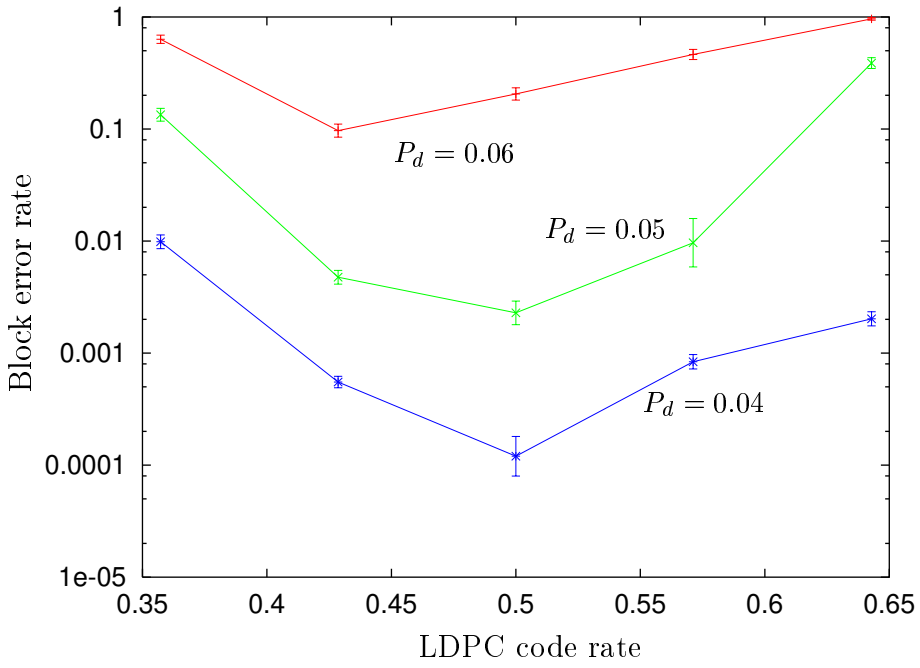


Figure 7.13: Block error rate as a function of LDPC code rate. Overall rate of code was fixed at 3/14, with blocklength 4662 bits. Inner codes: watermark codes with $n = (5, 6, 7, 8, 9)$. Outer codes: regular LDPC codes over $GF(8)$ with length N_L and rate R_L chosen to satisfy the constraints $N_L \times n = 4662, R_L \times 3/n = 3/14$. Results are given for three choices of insertion/deletion probability. $P_s = 0$.

7.6.3 High-rate watermark codes

Figure 7.14 shows the decoding accuracy of a high rate $N = 4995$ watermark code as a function of the channel parameters. The code was constructed from a rate 8/9 LDPC code over $GF(16)$. Each symbol was mapped to 5 bits of $\mathbf{s}$, which made the density of the sparse

vector 0.3125. For decoding to be possible with such a dense $\mathbf{s}$, the insertion/deletion rates are necessarily very low. A block error rate of less than 10^{-3} was achieved for insertion/deletion probability 15×10^{-4} and channel substitution error rate less than 3×10^{-3}. At these noise levels there are an average of 14 synchronisation and 14 substitution errors per block.

Results are shown for several settings of the channel substitution rate P_s. Increasing P_s affects the performance smoothly, as suggested by the entropy surface (figure 7.10). For the common case of channels with substitution rates much less than the density of $\mathbf{s}$, synchronisation performance is not significantly affected. In this case the effect of channel substitutions is to make the estimation of the sparse bits more difficult, causing a progressive increase in the entropy of the input to the LDPC decoder. As a result, the degradation in performance as P_s increases is comparable to that for a binary symmetric channel with increasing noise, with the insertion/deletion errors adding an effective background noise level. Subsequent results are quoted for channels without substitution errors.

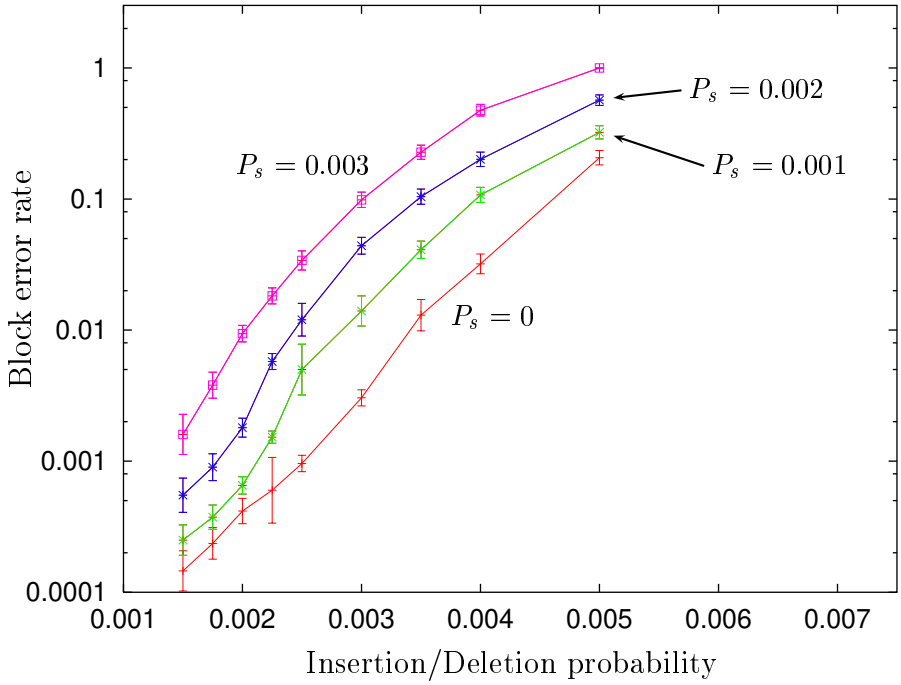


Figure 7.14: Block error rate as a function of insertion/deletion probabilities $P_d = P_i$, for several choices of substitution probability P_s. Inner code: Watermark code with $n = 5$. Outer code: Regular LDPC code over $GF(16)$ of column weight 3, rate 8/9, length 999. Overall rate 0.71, with a total blocklength of 4995 bits.

7.6.4 Lower rate watermark codes

For lower rate watermark codes, much higher noise levels can be tolerated. With an outer code over $GF(16)$ the density of $\mathbf{s}$ can be almost halved (from 0.31 to 0.17) by reducing the rate of

the watermark code from 0.8 to 0.5. Watermark codes of rate 3/14 have been constructed that can routinely correct several hundred insertion and deletion events in blocks a few thousand bits long. In figure 7.15 the performance of various watermark codes is compared. Results are shown for codes of rates approximately 0.7, 0.5, 0.2 and 0.05. For clarity, all results are for channels without substitution errors.

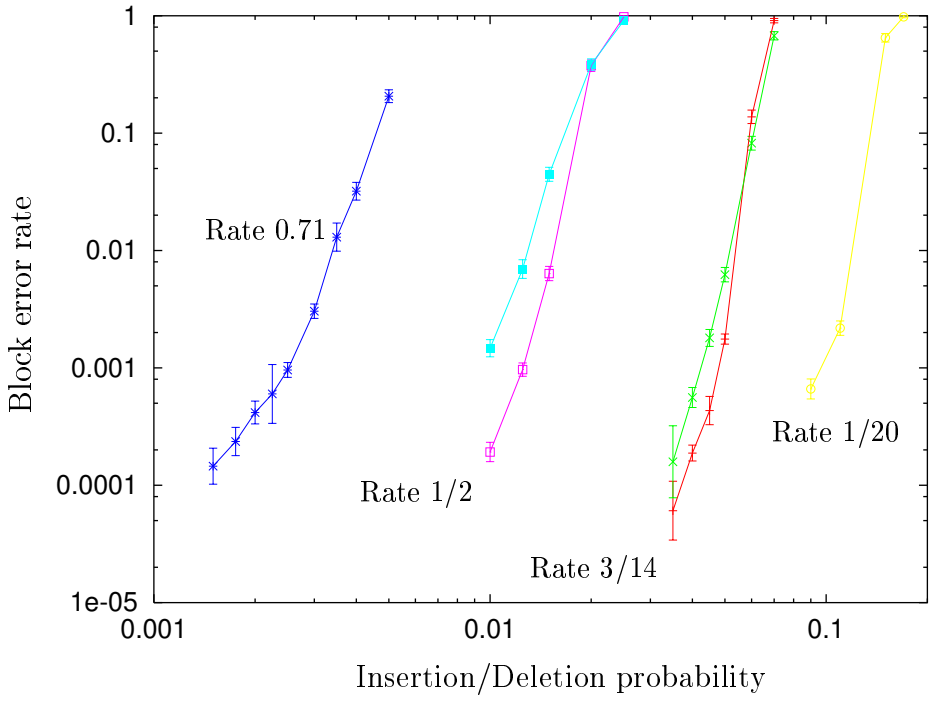


Figure 7.15: Performance of watermark codes. Details of codes from left to right: Rate 0.71 code, blocklength 4995 bits, outer code of rate 8/9 over $GF(16)$; Two rate 1/2 codes, blocklength 4000 bits, with outer codes of rate 5/8 and 3/4 respectively, defined over $GF(16)$; Rate 3/14 codes, blocklength 4662 bits, outer codes of rate 3/7 and 1/2 over $GF(8)$; Rate 1/20 code, blocklength 6000, outer code of rate 1/10 over $GF(8)$. All outer codes were regular LDPC codes with mean column weights between 2.6 and 3. The channel substitution probability P_s was zero. Full details of the outer codes are given in appendix F.

From the figure it is clear that block error rates of less than 10^{-4} can easily be achieved with watermark codes. For the results shown, the length of the LDPC code was no more than 1000 q-ary symbols. For longer codes, we expect to see further improvements in performance. Improvements could also be made by using irregular constructions for the LDPC codes. Nevertheless, the results show watermark codes are extremely effective at communicating over channels with insertions and deletions.

There are few results in the literature that are directly comparable to those presented here. Recently Bours [9, section 3.5.2] developed codes that could correct bursts of insertions and deletions. Rate 0.465 codes of blocklength 15840 bits were presented which had a block error rate 10^{-3} given a channel which made insertion/deletion bursts of expected length 6

on average once every 9 blocks. Watermark codes can do much better even without the synchronisation errors coming in bursts. Figure 7.15 shows watermark codes of length 4662 and rate 0.5 reaching block error rate 10^{-3} with almost 100 synchronisation errors scattered throughout each block.

7.7 Complexity and implementation considerations

7.7.1 Faster decoding

A naïve implementation of watermark decoding would result in a costly decoding algorithm. The forward/backward algorithm scales as X^2N, where X is the number of states in the hidden Markov model and N is the length of the hidden sequence. For watermark codes, X limits the loss in synchronisation that we consider within a block ($X = 1+2x_{\max}$). Assuming the insertion and deletion probabilities are equal, the synchronisation drift in a block of length N is a Gaussian random variable with mean 0 and variance $N \times P_d/(1-P_d)$ (see appendix C.4). For blocks of length 4000, and $P_d = 0.05$, we would need to set $X > 85$ to catch drifts at the 3σ level. The complexity and memory requirements of the resulting decoding algorithm would be prohibitive.

Clearly, an $O(X^2)$ algorithm is wasteful for two reasons. Firstly, most of the entries in the HMM transition matrix represent weak or disallowed transitions. This is because the synchronisation only drifts a small amount at each step. Secondly, the synchronisation is generally very well determined: for all codes and noise levels shown in figure 7.15 the average error at the end of a block was less than 1.

These observations allow great reduction in the complexity of the decoding algorithm, at a slight expense of accuracy. Given our simple channel model, a burst of I insertions followed by a transmission is just as likely as two bursts of $I/2$ insertions separated by a transmission. In practice, in the region of such an event it would be impossible to separate transmitted from inserted bits. Hence, when decoding it is sufficient to consider synchronisation drifts from -1 to +2 (say) at each step. Longer insertion events will then be 'explained' by several shorter events, at minimal cost to decoding accuracy.

Also, by performing short backward passes as the forward pass progresses, the local synchronisation within a block can be estimated to within some uncertainty Δx. Then it is only necessary to propagate paths starting within the identified region. Even without short backward passes, useful speed-ups can be achieved by considering only forward/backward pass states with probabilities above a certain threshold. Combining these changes, the complexity of the decoding algorithm scales as $4\Delta x$ rather than X^2.

Furthermore, by identifying the local synchronisation throughout the block, we obviate the need for the artificial limit X on the synchronisation drift. With appropriate modification of the decoding algorithm, it is possible to perform the forward and backward passes through a corridor of width Δx surrounding the most likely path through the lattice (figure 7.16).

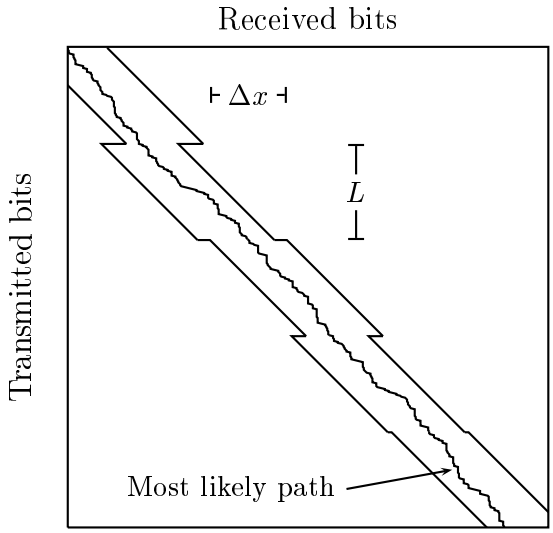


Figure 7.16: Use of local synchronisation to restrict the decoding lattice: Every L bits, the local synchronisation is estimated using a short backward pass. This is used to define a 'corridor' in the lattice of width Δx around the most likely path. Only paths within the corridor are considered during decoding. The expected drift in length L is a factor $\sqrt{L/N}$ smaller than for the whole block, so reducing the number of states required in the HMM decoder.

7.7.2 How sparse is a sparse vector?

In general, for a fixed watermark code rate k/n, the density f of the sparse vector $\mathbf{s}$ decreases as n and k increase. However, this decrease is not monotonic. Define $f(k,n)$ to be the mean density of the 2^k sparsest binary vectors of length n. Then $f(3,6) = 0.167$ but $f(4,8) = 0.172$.

It is easy to calculate the limit $\lim_{\alpha\to\infty} f(\alpha k, \alpha n)$. For large α, we can view the reverse mapping from $\mathbf{s}$ to $\mathbf{d}$ as a compression algorithm, from a block of length αn to a block of uniform bits of length αk. For compression to be possible, the entropy of the sparse block must be at most αk bits. Hence $n \times H_2(f(\alpha k, \alpha n)) \to k$ as $\alpha \to \infty$.

When designing watermark codes, efficient choices of n and k are of greater interest than the asymptotic result. Consider $n = 7$, $k = 3$. In this case, the null vector and all 7 weight-1 vectors are used, giving a particularly efficient encoding. Not until $k = 7$ is a sparser representation possible for comparable k/n. Figure 7.17 shows the sparseness achieved as a function of the watermark code rate k/n for several choices of k. Also shown is the limiting value $H_2(f) = n/k$.

As discussed in section 7.5.3, we expect minimising the density of $\mathbf{s}$ to help decoding by improving the estimate of $\mathbf{d}$ output by the watermark decoder. Figure 7.18 shows that this is indeed the case. The entropy of $\mathbf{d}$ is roughly proportional to the density of $\mathbf{s}$, at least for $n < 2\log(q)$.

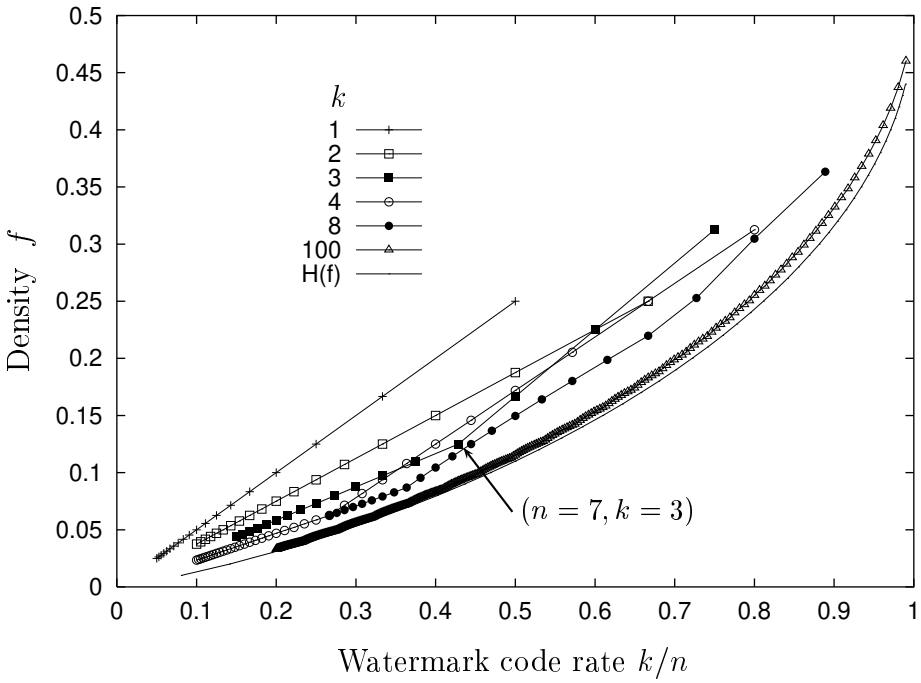


Figure 7.17: Average density of the sparse vector $\mathbf{s}$ as a function of the watermark code rate k/n for several choices of k. Also shown is the asymptotic limit, $H_2(f) = k/n$.

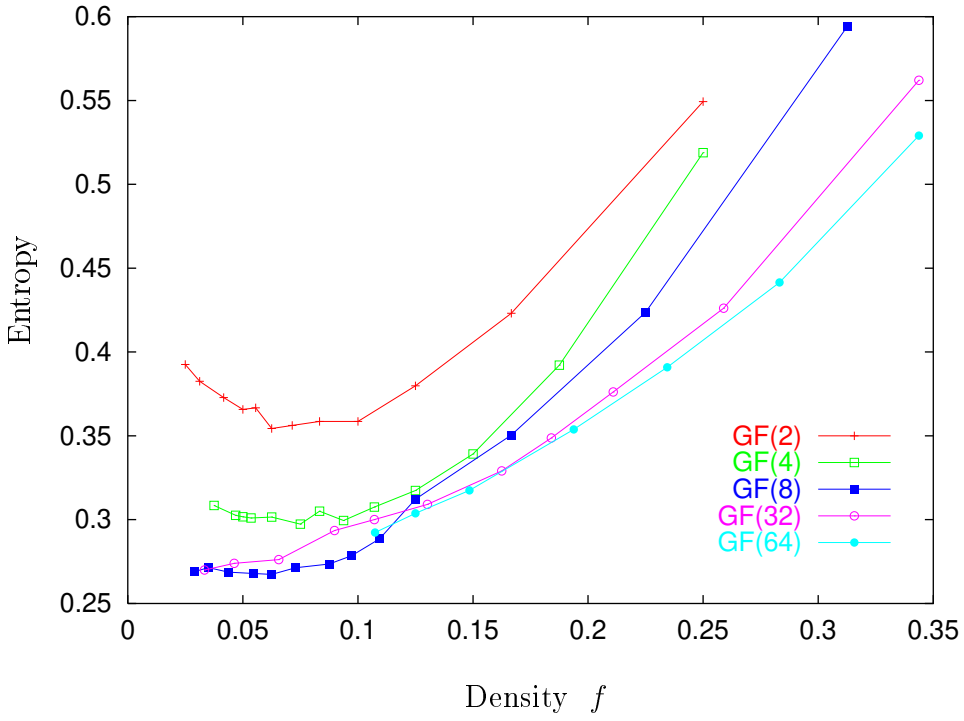


Figure 7.18: The entropy per bit of the LDPC codeword $\mathbf{d}$ as a function of the average density of the sparse vector $\mathbf{s}$, for codes over several fields $GF(q)$, $q \in \{2, 4, 8, 32, 64\}$.

7.7.3 Choice of watermark vector

It is possible that watermark vectors with structure could have useful properties not offered by purely random choices. Long runs of zeroes or ones make it impossible to localise deletions (and insertions that lengthen the runs) to greater accuracy than the length of the run. Hence, it is natural to consider using maximum run-length limited (RLL) sequences as watermark vectors. Constraining the minimum run-length to be greater than 1 is not expected to be useful.

RLL watermark sequences with maximum run lengths of various length have been compared. The sequences were constructed using a uniform random bit generator whenever the next bit in the sequence was not determined by the RLL constraints.

No choice of RLL constraints has been found to improve decoding. Initial investigations, shown in figure 7.19, found that watermark sequences with maximum run-length less than 4 gave significantly poorer performance. As the maximum run-length was increased, the results soon became indistinguishable from those obtained from random watermark sequences.

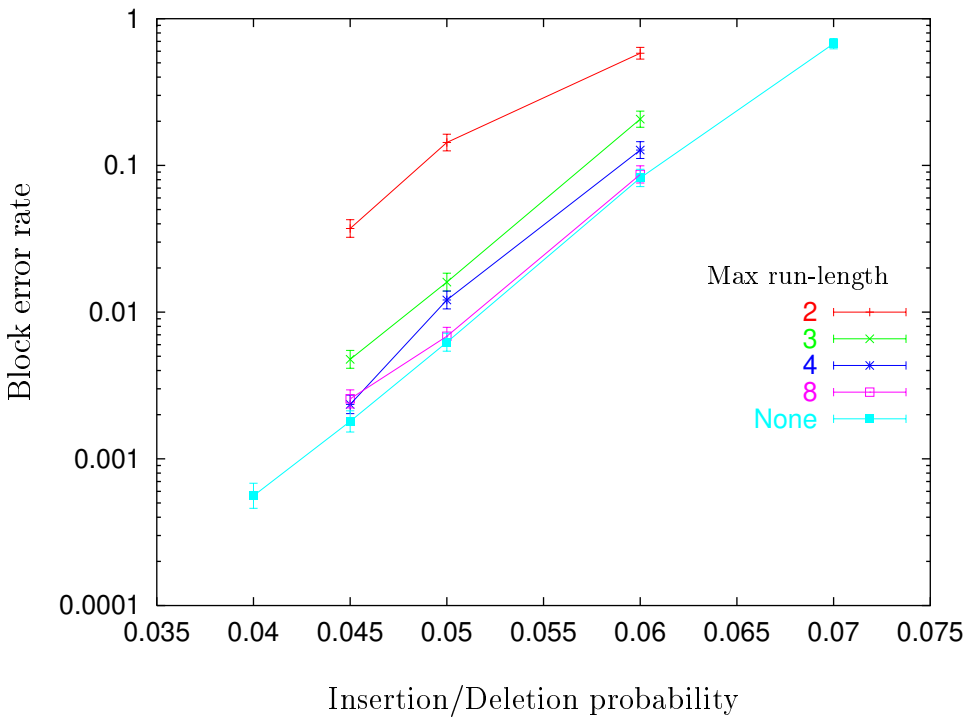


Figure 7.19: Results for codes with maximum run-length limited watermark vectors. Overall rate of code was 3/14, blocklength 4662 bits. Outer code: regular rate 1/2 LDPC code over $GF(8)$.

7.8 Future directions

Channel models

The probabilistic decoding framework makes the development of watermark codes for different channels relatively straightforward. For example, many channels of practical interest (*e.g.* the S-DAT recording channel [73]) exhibit occasional bursts of synchronisation errors as well as more frequent single or double errors. By changing the HMM used by the watermark decoder, watermark codes could be used without further modifications to communicate over such channels.

Towards higher rate watermark codes

Watermark codes are well suited to channels that make large numbers of synchronisation errors. It is often the case that synchronisation errors are much rarer than substitution errors. In such cases, a high rate watermark code is desirable. Clearly, for outer codes defined over $GF(2^k)$, the highest watermark code rate possible is $k/(k+1)$, so high rate codes require large k. Figure 7.17 shows that at high rates $\mathbf{s}$ can be made significantly sparser by increasing k. The range of useful values of k is restricted by the decoding complexity, which scales as 2^k for the watermark decoder because a probability is assigned to each of the 2^k possible values of each symbol of the vector $\mathbf{d}$. The decoding complexity of the outer LDPC codes scales as $2^k \log(2^k)$.

When synchronisation is well determined (as is likely for high-rate applications), the 2^k step in the watermark decoder is wasteful because the likelihood of most of the symbols will be close to zero. Higher values of k (*e.g.* $k = 20$) could be used by modifying the decoding algorithm to follow only those paths whose likelihood is within some chosen factor of the most likely path. Such *beam search* methods are well-developed in the speech recognition HMM community [55, 67], as are more involved techniques such as dealing with HMMs with very large numbers of states, finding the N-best paths, and using multiple passes to speed up the HMM decoding [11, 2, 77].

To deal with large k we would also need an efficient and invertible method of mapping sparse to dense vectors, because a look-up table of sufficient size would require too much storage. A good candidate for this task is arithmetic coding. Arithmetic codes [92] are an excellent compression tool for sources with known statistics, mapping strings to a unique subinterval of $[0, 1)$ of a size equal to their probability. By modelling a source of independent and identically distributed bits of density f and reversing the compression step, dense vectors (identifying a subinterval) can be mapped to sparse vectors. Given a sparse vector, the compression step can be used to find the corresponding dense vector. Some care would be required to ensure the mapping was invertible, and to deal with sparse vectors with no corresponding dense vector.

LDPC codes over fields of order $GF(2^k > 256)$ have not been studied. LDPC codes will not be practical as outer codes for watermark codes of rate greater than 0.9 unless efficient

modifications to the decoding algorithm are found. As with the watermark decoder, it would be necessary to restrict attention to a subset of $GF(2^k)$ for each symbol of $\mathbf{d}$.

Irregular watermark codes

Marker codes [78] can be viewed as a special case of 'irregular' watermark codes. We use the term 'irregular' to denote watermark codes in which the density of the sparse vector $\mathbf{s}$ is not uniform. In Marker codes, information-carrying sections of a block are interspersed with synchronisation-providing marker sections. In an irregular watermark code, marker sections correspond to regions where $\mathbf{s}$ has a density of 0, and information sections to regions where $\mathbf{s}$ has a density 0.5 and the watermark is set to zero.

By varying the density of $\mathbf{s}$ the information can be spread unevenly throughout the block with some sections being used to provide easier synchronisation. Information about variations in the density of $\mathbf{s}$ is easily incorporated into the decoding algorithm. We expect that, just as irregular LDPC codes work better, irregular watermark codes will perform better because occasional low-density sections within a block may make it possible to cope with higher rates of synchronisation errors by preventing drifts in synchronisation from accumulating.

If irregular LDPC codes are used as the outer code, a possible choice is to vary the density of $\mathbf{s}$ according to the weight of the corresponding column of the parity check matrix. Denser sections of $\mathbf{s}$ are harder to synchronise, but this difficulty can be counteracted by connecting them to higher-weight columns, in effect offering them higher error-protection.

Iterative watermark decoding

It is possible to use an iterative decoding scheme for the watermark code. When the watermark decoder first runs, all it knows about $\mathbf{s}$ is the expected density. After one or more steps of LDPC decoding, the current estimate of $\mathbf{d}$ could be used to calculate a refined estimate of the bits of $\mathbf{s}$. This could be used by the watermark decoder to generate a new estimate of $\mathbf{d}$. In this way, information passes between the outer and inner decoders by belief propagation. It is likely that significant improvements could be obtained with such an algorithm. The procedure would be similar to iterative decoding of serially concatenated interleaved codes [4].

Note that the watermark decoder approximates $\mathbf{s}$ as a stream of independent identically distributed bits during the forward and backward passes. This is not immediately compatible with the q-ary distribution provided by the LDPC code. The q-ary distribution can be used in conjunction with the lookup table of sparse mappings from $GF(q)$ to $\{0,1\}^n$ to generate a distribution over each bit of $\mathbf{s}$. The correct solution, of course, is for the watermark decoder to use the q-ary distribution as a prior distribution over each n consecutive bits of $\mathbf{s}$. The use of such information would lead to a significant increase in decoding complexity (of order q), but is likely to result in greatly improved decoding.

Chapter 8

Conclusions

Never make forecasts, especially about the future.

— Sam Goldwyn

Low-density parity-check codes

Gallager's low-density parity-check codes were seldom mentioned in the literature between 1962 and 1995 (notable exceptions being Tanner [83] and Wiberg [91]). MacKay and Neal's serendipitous rediscovery of these excellent codes raises the question: why were they forgotten for so long?

The advent of turbo codes in 1993 demonstrated that near-Shannon limit error correction for the Gaussian channel was practical, and inspired many hundreds of papers. NASA was sufficiently impressed to redesign their deep space communication codes around them. Iterative decoding seemed to be a novel idea.

In fact, Gallager had laid out just such an algorithm in his PhD thesis thirty years earlier [26] – an invention that has been described as "almost clairvoyant" [65] and was certainly ahead of its time. Recent advances, including those presented in this thesis, show that turbo codes are not alone in providing near Shannon limit error-correction. Indeed, iterative sum-product decoding is capable of extracting near Shannon limit performance from codes with disarmingly simple constructions [44]. Had sufficient computing power been available in 1960 to demonstrate the power of Gallager's codes, coding theory would undoubtably have developed along different lines.

In [16] we presented the first evidence that practical low-density parity-check codes could outperform turbo codes. Since then, Richardson *et al.* have demonstrated extremely long blocklength LDPC codes performing less than 0.1 dB from capacity [74]. It is unlikely that approaching closer to capacity will be of practical importance.

Low-density parity-check codes possess notable advantages over turbo codes, making them good candidates for engineering applications. First, in our experience all errors made by LDPC codes are detected errors: the decoder reports the fact that a block has been incorrectly decoded. In contrast, turbo codes make undetected errors due to the presence of relatively

low-weight codewords. Second, LDPC codes are fully parallelisable: each parity-check can perform its own calculation. Although the message passing overhead may be significant (as is common with parallel algorithms), parallel implementations have the potential to speed up decoding considerably. Even without parallelisation, the complexity of binary LDPC codes is slightly lower than that of turbo codes.

LDPC decoding has already been implemented in hardware [47]. Recently, Lucent Technologies announced a new chip for magnetic recording channels, incorporating a Gallager code and an iterative decoder [88], although the details have not yet been made public. Analogue computation, such as that recently implemented for turbo codes by Hagenauer *et al.* [31, 32] and Loeliger *et al.* [54], is likely to prove effective for LDPC codes. It is estimated that analog decoders can outperform digital decoders by two orders of magnitude in speed and/or power consumption.

Our best results (see figure 4.3) have been achieved using non-binary irregular low-density parity-check codes. For a fixed blocklength, we find optimised non-binary codes outperform binary codes, where optimisation is carried out using the methods of section 4.1. The irregular profile optimiser used by Richardson *et al.* has not yet been applied to non-binary codes, but we expect that non-binary codes can outperform the best binary codes for a given blocklength.

For short-blocklength codes, where turbo codes can still outperform LDPC codes, we expect non-binary LDPC codes to be useful. On the JPL 'imperfectness' web page [46] we report a rate 1/3 code over $GF(4)$ of blocklength 1920 bits with performance just 0.2 dB worse than that of the best turbo code with similar blocklength. It is likely that further optimisation of the code profile could improve this result.

The average asymptotic (*i.e.* cycle free) behaviour of the LDPC decoding algorithm is now well described [75]. Long blocklength codes closely approximate the asymptotic behaviour, but for short blocklengths the actual behaviour departs significantly (see figure 3.4). Thus, the design of short block-length codes cannot be considered a solved problem. Chapter 5 showed that the decoding algorithm displays some characteristics of a chaotic dynamical system. It is possible that the theory of non-linear dynamics may lead to useful analyses of finite blocklength low-density parity-check codes. For example, we could estimate the size of the basin of attraction of codewords by varying the signal received from a Gaussian channel in a continuous manner until decoding became impossible.

Watermark codes

The correction of insertion/deletion errors has been an outstanding problem since the 1960s. Recent contributions to the field have included short blocklength codes for the correction of a few (*e.g.* 3) insertion/deletion errors per block [39] and codes whose tolerance to insertions and deletions is several orders of magnitude below that of watermark codes [9]. Watermark codes, presented in chapter 7, make possible the correction of large numbers of insertion/deletion errors per block. Thus, it seems that watermark codes represent a significant new contribution.

Watermark codes can communicate reliably over some channels for which communication

has previously been impossible. For example, the maximum number of channel uses per unit time is often limited by the need to avoid synchronisation errors. If higher bit rates are possible at the cost of a few synchronisation errors, watermark codes may well allow reliable communication with faster communication rates.

The work presented in this thesis represents an initial investigation into watermark codes, proving their effectiveness. Encouraging results have been obtained with only a small attempt to optimise the inner and outer code parameters. The construction of irregular watermark codes and the coupling of the inner and outer decoders are two promising areas for future work, as outlined at the close of chapter 7.

Conclusion

Low-density parity-check codes are now recognised to be outstanding codes, both in versatility and performance. Useful codes of almost any blocklength and rate can be constructed and efficiently decoded. Their profile in the information theory community has been rising steadily since their performance was recognised in 1995. Recent work has shown they can approach capacity under optimal decoding and that practical performance is better than all other known codes for the Gaussian channel. Investigations of the iterative decoding algorithm have helped in the understanding of turbo decoding.

The work in this thesis has shown that carefully constructed non-binary low-density parity-check codes can outperform binary constructions. A family of novel non-linear codes, which we call watermark codes, have been shown to provide excellent protection against insertion and deletion errors when used in conjunction with low-density parity-check codes.

In belated recognition of his overlooked work, Gallager's 1962 paper was recently honoured with an IEEE IT Society Golden Jubilee Paper Award. The work presented in this thesis suggests that his codes are not likely to be forgotten again.

Appendix A

Linear Block Codes

Fifty years ago, in his classic 1948 work "A Mathematical Theory of Communication"[79], Claude Shannon addressed the problem of achieving reliable communication over a noisy channel. He envisioned a system whereby the data is first encoded, then transmitted, and finally decoded to correct errors introduced in transmission (figure A.1). Such an encoding/decoding scheme is an error-correcting code. The aim is minimise the probability of residual errors after decoding, while introducing as little redundancy as possible during encoding.

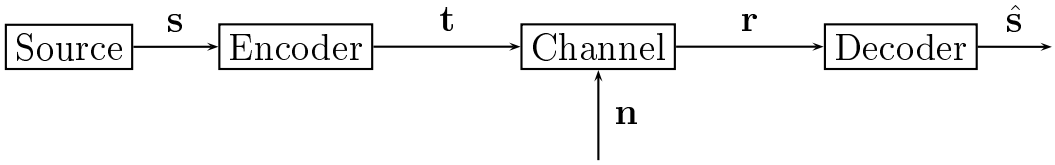


Figure A.1: Diagram of general error-correcting communication system. The encoder maps a message vector $\mathbf{s}$ to a transmitted codeword $\mathbf{t}$ which is received by the decoder after the addition of noise as the vector $\mathbf{r}$. The decoder outputs $\hat{\mathbf{s}}$, its best guess of the input message.

Shannon showed that for any channel with a defined capacity there exist coding schemes that, when decoded with an optimal decoder, achieve arbitrarily small error probability for all transmission rates below capacity. It is a fundamental problem of information theory to make practical codes whose performance approaches the Shannon limit.

Low-density parity-check codes are a class of linear error-correcting block codes. A *block code* is one in which every sequence of k symbols from the source is encoded into a transmitted *codeword* of length n. The *rate* of such a code is then k/n, since we communicate k information bits for every n channel uses. We refer to n as the *blocklength* of the code. A linear code is one in which linear combinations of codewords are themselves valid codewords.

A.1 Generator matrix

A linear block code of blocklength n and rate k/n can be described by a generator matrix $\mathbf{G}$ of dimension $k \times n$ that describes the mapping from source words $\mathbf{s}$ to codewords $\mathbf{t} := \mathbf{G}^\mathsf{T}\mathbf{s}$ (where the vectors $\mathbf{s}$, $\mathbf{t}$ are column vectors). It is common to consider $\mathbf{G}$ in systematic form, $\mathbf{G} \equiv [\mathbf{I}_k|\mathbf{P}]$ so that the first k transmitted symbols are the source symbols. The notation $[\mathbf{A}|\mathbf{B}]$ indicates the concatenation of matrix $\mathbf{A}$ with matrix $\mathbf{B}$; $\mathbf{I}_k$ represents the $k \times k$ identity matrix. The remaining $m = n - k$ symbols are *parity-checks*.

A.2 Parity-check matrix

Such a code is also described by a parity-check matrix $\mathbf{H}$ of dimension $m \times n$, where $m = n - k$. If the corresponding generator matrix is written in systematic form as above, then $\mathbf{H}$ has the form $[-\mathbf{P}|\mathbf{I}_m]$. Note that for codes over finite fields $GF(2^p)$ (see appendix B) $\mathbf{P} \equiv -\mathbf{P}$. Each row of the parity-check matrix describes a linear constraint satisfied by all codewords. $\mathbf{H}\mathbf{G}^\mathsf{T} = \mathbf{0}$ and hence the parity-check matrix can be used to detect errors in the received vector:

$$\mathbf{H}\mathbf{r} = \mathbf{H}(\mathbf{t} + \mathbf{n}) = \mathbf{H}\mathbf{G}^\mathsf{T}\mathbf{s} + \mathbf{H}\mathbf{n} = \mathbf{H}\mathbf{n} := \mathbf{z}. \tag{A.1}$$

Here we have introduced the *syndrome* vector, $\mathbf{z}$. If the syndrome vector is null, we assume there have been no errors. Otherwise, the decoding problem is to find the most likely noise vector $\hat{\mathbf{n}}$ that explains the observed syndrome given the assumed properties of the channel.

The operation of linear error-correcting codes is summarised in figure A.2.

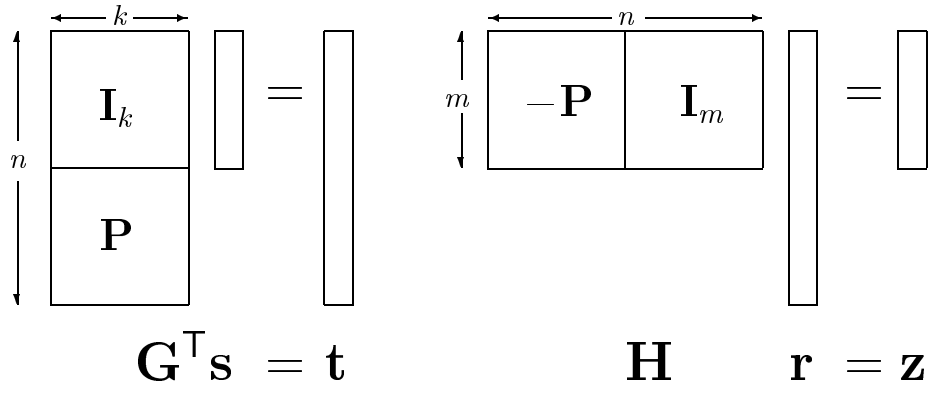


Figure A.2: Linear error-correcting codes: $\mathbf{G}$ maps a message $\mathbf{s}$ to a transmitted codeword $\mathbf{t}$. During transmission the channel adds noise $\mathbf{n}$. $\mathbf{H}$ maps received message $\mathbf{r} := \mathbf{t} + \mathbf{n}$ to syndrome $\mathbf{z}$.

Appendix B

Arithmetic in a Finite Field

In this thesis we consider both binary and non-binary codes. We refer to the words $\mathbf{s}, \mathbf{t}, \mathbf{n}$, *etc.* as *vectors*. These vectors live in a vector space over a finite field of order q, $GF(q)$. The addition and inner product of two vectors is defined in the usual way with respect to the field operations '$\oplus$' and '$\odot$'. In the binary case, addition of vectors reduces to component-wise addition mod(2) (exclusive-OR) and the inner product is the sum of the component-wise multiplications mod(2). For example:

$$\begin{aligned} 01011 \oplus 11001 &= 10010 \\ 01011 \odot 11001 &= 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 0 \end{aligned}$$

The simplest finite fields are $GF(p)$ where p is a prime number. The elements of $GF(p)$ are $\{0, 1, \ldots, p-1\}$ and the operations $\oplus$, $\odot$ are addition and multiplication mod p.

Finite fields of order $GF(p^m)$ have a representation as the set of all polynomials of degree at most $m-1$, with coefficients taken from the finite field with p elements. The field operation $\oplus$ is defined by adding the two polynomials, adding the coefficients using addition in $GF(p)$. Multiplication is performed modulo a polynomial $\pi(x)$ with coefficients from $GF(p)$ which is irreducible over $GF(p)$; *i.e.* $\pi(x)$ is not the product of two polynomials of lower degree with coefficients from $GF(p)$. The choice of $\pi(x)$ defines a representation of $GF(p^m)$, but all such representations are isomorphic: there is essentially only one field of order p^m. A thorough introduction to the theory and use of finite fields is given in [52].

When working with fields $GF(2^p)$ we use the term *symbols* to refer to elements of the field $GF(2^p)$. *Bits* refers to binary representations of symbols. Thus, in $GF(8)$ a symbol consists of three bits.

Appendix C

Channel Models

In this appendix we describe three discrete memoryless channels (Gaussian, binary symmetric and q-ary symmetric) considered in this thesis. We also discuss some simple properties of the insertion/deletion channel introduced in chapter 7.

C.1 Additive white Gaussian noise channel

We consider the binary-input real-output Gaussian channel, also known as the additive white Gaussian noise (AWGN) channel. Our treatment is based on that of MacKay [58, pp425–426].

Transmitted bits $t \in (0, 1)$ map to transmitted signals $x \in (+x_0, -x_0)$ and the output is $y = x + \nu$ where ν is a zero mean normally distributed random variable with variance σ^2. We set $\sigma = 1$ and vary the signal amplitude x_0 to control the signal to noise ratio.

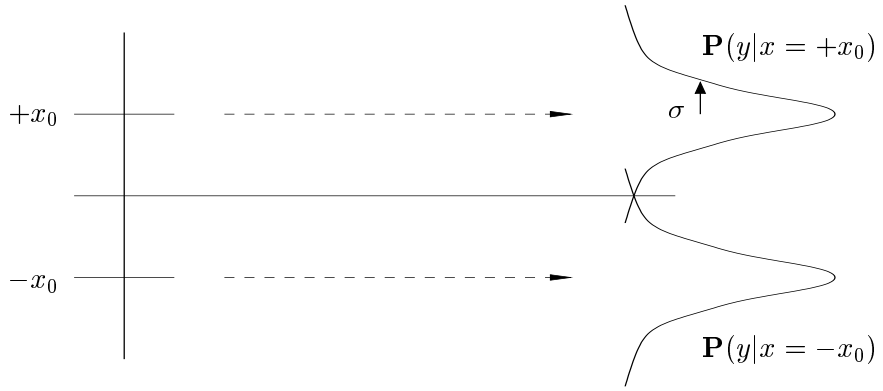


Figure C.1: Gaussian channel

We declare the received bit $r = 1$ if $y > 0$ and $r = 0$ if $y < 0$. The likelihood of this bit being in error is:

$$f^1 := \mathbf{P}(n = 1|y) = (1 + \exp(2x_0|y|))^{-1} \tag{C.1}$$

where $r = t + n \bmod(2)$. We define $f^0 = 1 - f^1$.

If the user is allowed to vary the input signal arbitrarily, constrained only to keep the input power equal to that of the binary inputs $\pm x_0$, then the capacity of the channel is given by:

$$C_{\mathrm{U}} = \frac{1}{2}\log_2\left(1 + \frac{x^2}{\sigma^2}\right). \tag{C.2}$$

For the binary input Gaussian channel, the capacity is somewhat reduced:

$$\begin{aligned} C_{\mathrm{B}} &= H(Y) - H(Y|X) && \text{(C.3)} \\ &= -\int dy\ P(y)\log P(y) + \int dx\ P(y|x=x_0)\log P(y|y=x_0) && \text{(C.4)} \end{aligned}$$

where

$$P(y) = \frac{1}{2\sqrt{2\pi\sigma^2}}\left[e^{-(y+x_0)^2/2\sigma^2} + e^{-(y-x_0)^2/2\sigma^2}\right] \tag{C.5}$$

C_{B} may be evaluated numerically.
For a code of rate R we write the signal to noise ratio as

$$\frac{E_b}{N_0} = \frac{x_0^2}{2R\sigma^2} \tag{C.6}$$

and report this number in decibels as $10\log_{10} E_b/N_0$.

In figure C.2, the Shannon limit of the binary input Gaussian channel is shown as a function of the signal to noise ratio for several different rates R.

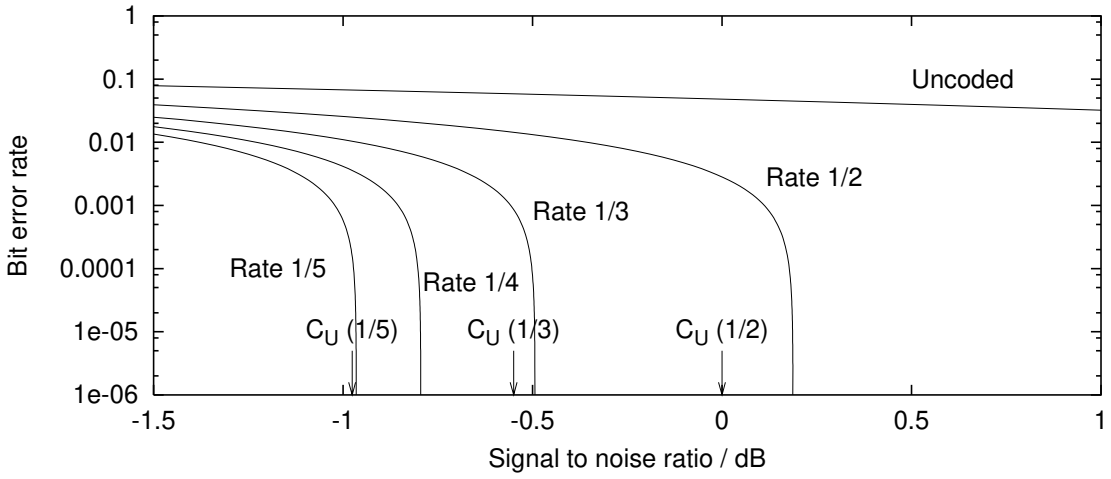


Figure C.2: The Shannon limit for the binary input Gaussian channel. For each rate, the region above and to the right of the Shannon limit is theoretically achievable. The capacity C_{U} of the unconstrained Gaussian channel is also shown (for clarity, C_{U} for rate 1/4 has been omitted).

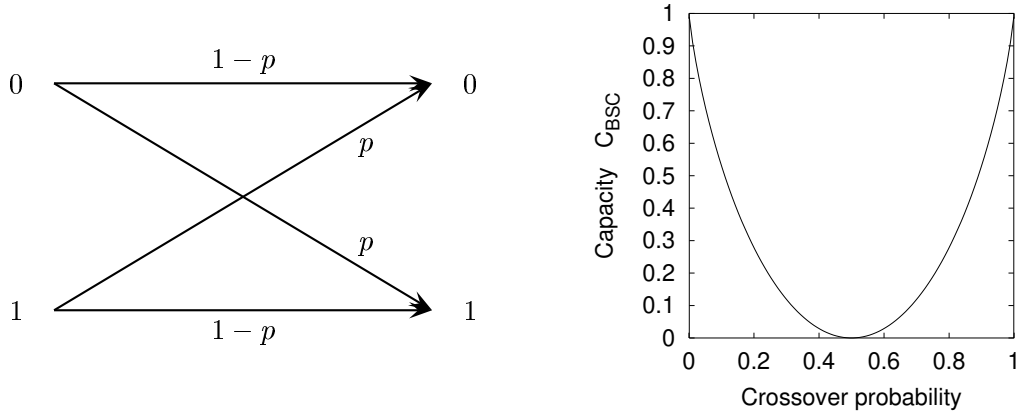
C.2 Binary symmetric channel

The binary symmetric channel is a binary input/binary output channel in which the probability that a 0 is flipped to a 1 is equal to the probability that a 1 is flipped to a 0, as indicated in figure C.3(a). The error probability (also known as the 'crossover' probability) is denoted by p. The capacity of the binary symmetric channel is

$$C_{\mathrm{BSC}} = H(Y) - H(Y|X) \tag{C.7}$$

$$= 1 + (1-p)\log_2(1-p) + p\log_2(p). \tag{C.8}$$

C_{BSC} is shown in figure C.3(b).



(a) (b)

Figure C.3: Binary symmetric channel: (a) Transition probability diagram: $P(0 \to 1) = P(1 \to 0)$. (b) Capacity of the binary symmetric channel, as a function of the crossover probability.

C.3 q-ary symmetric channel

The q-ary symmetric channel is a simple generalisation of the binary symmetric channel. The input and output alphabets both have q elements, and the probability that a symbol is received correctly is $(1-p)$, $\quad p < (q-1)/q$. If a symbol is corrupted, the $(q-1)$ alternative received symbols are all equally probable: $P(y = a|a \neq x) = p/(q-1)$. When $q = 2$ we obtain the binary symmetric channel. The channel capacity is easily calculated:

$$C_{\mathrm{QSC}} = \log_2(q) + (1-p)\log_2(1-p) + p\log_2\frac{p}{q-1} \tag{C.9}$$

C.4 Simple Insertion/Deletion Channel

In section 7.2 we introduced a simple Markov model of a channel with insertion and deletion errors. Unlike the channels considered above, the capacity of insertion/deletion channels is not known.

In this section we sketch the statistical properties of the channel, showing that the excess (*i.e.* synchronisation drift) is a zero-mean Gaussian random variable. Our treatment is based on that of Drasdo *et al.* [19].

For completeness, figure C.4 repeats figure 7.1, showing the channel model under consideration.

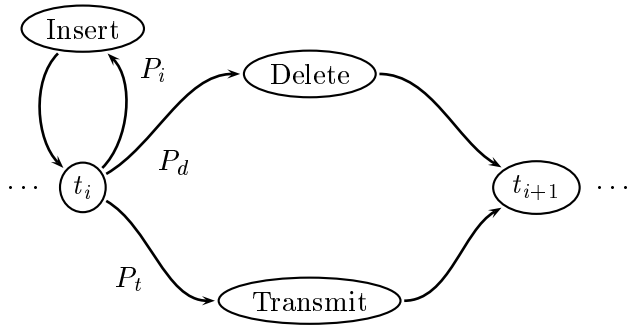


Figure C.4: Insertion/Deletion channel with probabilities P_i, P_d and P_t of insertions, deletions and transmissions.

We take $P_i = P_d = p$, so that the expected length of the received message $\{r_j\}$ is equal to the length of the transmitted message $\{t_i\}$. In the lattice notation of section 7.4, if a path passes through the point (i, j) then the first i message symbols produce j received symbols.

It is useful to use the rotated coordinate system (X, T) defined by $X \equiv j - i$ and $T \equiv i + j$. X is the excess number of received symbols after i source symbols have been transmitted. We refer to T as the time variable. We define $\pi(X|T)$ to be the the probability that a path passes through the point (X, T) at time T.

From the channel model, we obtain the recurrence relation

$$\pi(X|T+1) = (1-2p)\pi(X|T-1) + p\pi(X-1|T) + p\pi(X+1|T). \tag{C.10}$$

Asyptotically, for large X and T we can approximate (C.10) by the differential equation

$$\begin{aligned}\pi(X|T) + \frac{\partial \pi(X|T)}{\partial T} &= (1-2p)\left(\pi(X|T) - \frac{\partial \pi(X|T)}{\partial T}\right) \\ &\quad + p\left(\pi(X|T) - \frac{\partial \pi(X|T)}{\partial X} + \frac{1}{2}\frac{\partial^2 \pi(X|T)}{\partial X^2}\right) \qquad \text{(C.11)} \\ &\quad + p\left(\pi(X|T) + \frac{\partial \pi(X|T)}{\partial X} + \frac{1}{2}\frac{\partial^2 \pi(X|T)}{\partial X^2}\right) \\ &= \pi(X|T) + p\frac{\partial^2 \pi(X|T)}{\partial X^2} - (1-2p)\frac{\partial \pi(X|T)}{\partial T} \qquad \text{(C.12)}\end{aligned}$$

Equation (C.12) reduces to the equation

$$\frac{\partial \pi(X|T)}{\partial T} = \frac{p}{2(1-p)}\frac{\partial^2 \pi(X|T)}{\partial X^2},$$

the solution of which is

$$\pi(X|T) = \frac{1}{\sqrt{2\pi\alpha T}}\, e^{-\frac{X^2}{2\alpha T}}$$

where $\alpha = p/(1-p)$. Hence $X(T)$ is a Gaussian random variable with moments

$$\overline{X(T)} = 0, \qquad \overline{X^2(T)} = \frac{p}{1-p}\, T.$$

In the case of watermark decoding, knowledge of the variance of X allows us to tune the number of states in the hidden Markov model. In the example of figure 7.16, the probability that the path leaves the decoding corridor can be controlled by setting the width Δx to an appropriate multiple of $\sqrt{P_d/(1-P_d)}$.

Appendix D

Hidden Markov Models

In this appendix we give a brief introduction to hidden Markov models, also known as Markov sources. For more detail, the reader is referred to Jelinek [43, chapter 2] or Rabiner and Juang [71], both of which contain extensive bibliographies.

D.1 Markov Chains

Hidden Markov models are flexible and powerful mathematical tools, suitable for modelling many random processes. They are based on the concept of a *Markov chain*, which we now define.

Let $X_1, X_2, \ldots, X_n, \ldots$ be a sequence of random variables all taking values in the finite alphabet $\mathcal{A} = \{a_1, \ldots, a_N\}$. The random variables are said to form a Markov chain if

$$P(X_i|X_1, \ldots, X_{i-1}) = P(X_i|X_{i-1}) \qquad \text{for all } i. \tag{D.1}$$

Hence, for Markov chains,

$$P(X_1, X_2, \ldots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i|X_{i-1}) \tag{D.2}$$

Random processes which produce such a sequence of variables have a minimum amount of memory, depending only on the value at the previous step. We restrict our attention to *time invariant* Markov chains, in which

$$P(X_i = b|X_{i-1} = a) = p(b|a) \qquad \text{for all } i, \quad \text{for all } a, b \in \mathcal{A}. \tag{D.3}$$

If we think of the values of the variables X_i as states, then $p(b|a)$ gives the probability of making a transition from state a to state b. Obviously $\sum_{b\in\mathcal{A}} p(b|a) = 1$ and $p(b|a) \geq 0$.

D.2 Hidden Markov Models

D.2.1 Definitions

In a hidden Markov model the state sequence $\{X_i\}$ of the Markov chain is not observable. Rather, the transitions between states produce observable output. We introduce the following formalism.

1. $\{X_i\}$, state sequence of hidden Markov chain.
2. $\mathcal{A} = \{a_1, \ldots, a_N\}$, state space of Markov chain, $X_i \in \mathcal{A}$. For simplicity, we assume a unique initial state $X_0 = s \in \mathcal{A}$.
3. $p(a_j|a_i)$, state transition probability distribution. $P(X_i = b|X_{i-1} = a) = p(b|a)$.
4. $\{Y_i\}$, observation sequence.
5. $\mathcal{B} = \{b_1, \ldots, b_M\}$, output alphabet.
6. $q(b_i|a_j, a_k)$, output probability distribution associated with transition from state a_j to a_k.

We denote the parameters of a hidden Markov model collectively by $\mathcal{H} = \{\mathcal{A}, p, \mathcal{B}, q\}$. The probability that the hidden Markov model produces the observed sequence $Y_1 Y_2 \ldots Y_n$ is

$$P(Y_1 Y_2 \ldots Y_n|\mathcal{H}) = \sum_{X_1,\ldots,X_k} \prod_{i=1}^{k} p(X_i|X_{i-1})\, q(Y_i|X_{i-1}, X_i) \tag{D.4}$$

D.2.2 Problems

We consider three key problems based on the model defined in the last section.

Problem 1 – Given an observation sequence $Y_1 Y_2 \ldots Y_n$ and a hidden Markov model $\mathcal{H}$, calculate the probability of the observation sequence: $P(Y_1 Y_2 \ldots Y_n|\mathcal{H})$.

Problem 2 – Given an observation sequence $Y_1 Y_2 \ldots Y_n$ and a hidden Markov model $\mathcal{H}$, find the most probable value of X_i for each i.

Problem 3 – Given an observation sequence $Y_1 Y_2 \ldots Y_n$ and a hidden Markov model $\mathcal{H}$, find the state sequence $\{X_i\}$ most likely to have emitted that sequence.

Problem 1 may be thought of as the problem of 'scoring' the model $\mathcal{H}$. Faced with a choice of models, solving problem 1 allows us to find the model that assigns the highest probability to the observed sequence. Problems 2 and 3 are concerned with recovering the hidden state sequence. The solution to problem 2 may not result in a valid state sequence, but finds the most probable value of each state X_i individually. We will see that problems 1 and 2 are closely related. The solution to problem 3 is the best guess of a single coherent state sequence, given the observed sequence.

D.2.3 Solutions

Problem 1

We introduce the *forward probability* $f_j(a)$, defined as the probability of the hidden Markov model emitting $Y_1 \dots Y_j$ and ending up in state a after j steps (*i.e.* $X_j = a$). We initialise f as follows

$$f_0(a) = \begin{cases} 1 & \text{if } a = s \\ 0 & \text{otherwise} \end{cases} \tag{D.5}$$

where s is the initial state. The forward probability is easy to calculate recursively using the Markov property (D.1).

$$\begin{aligned} P(Y_1 \dots Y_j, X_j = a|\mathcal{H}) &= \sum_{a' \in \mathcal{A}} P(Y_1 \dots Y_{j-1}, X_{j-1} = a'|\mathcal{H}) P(X_j = a|X_{j-1} = a', \mathcal{H}) \\ &\quad \times P(Y_j|X_{j-1} = a', X_j = a, \mathcal{H}) \qquad \text{(D.6)} \\ f_j(a) &= \sum_{a' \in \mathcal{A}} f_{j-1}(a')\, p(a|a')\, q(Y_j|a', a) \qquad \text{(D.7)} \end{aligned}$$

Hence, the solution to problem 1 is simply

$$P(Y_1 Y_2 \dots Y_n|\mathcal{H}) = \sum_{a \in \mathcal{A}} f_n(a) \tag{D.8}$$

Problem 2

To solve problem 2 we also require the *backward probability* $b_j(a)$, defined as the probability of emitting $Y_{j+1} \dots Y_n$ given that $X_j = a$. We set $b_n(a) = 1$, for all a. We calculate $b_j(a)$ in a similar fashion to the forward calculation:

$$\begin{aligned} P(Y_{j+1} \dots Y_n|X_j = a, \mathcal{H}) &= \sum_{a' \in \mathcal{A}} P(X_{j+1} = a'|X_j = a, \mathcal{H}) P(Y_{j+1}|X_j = a, X_{j+1} = a', \mathcal{H}) \\ &\quad \times P(Y_{j+2} \dots Y_n|X_{j+1} = a', \mathcal{H}) \qquad \text{(D.9)} \\ b_j(a) &= \sum_{a' \in \mathcal{A}} p(a'|a) q(Y_{j+1}|a, a') b_{j+1}(a') \qquad \text{(D.10)} \end{aligned}$$

It is now straightforward to find the most probable state after j steps by invoking Bayes' theorem, combining the forward and backward probabilities as follows:

$$\begin{aligned} P(X_j = a|Y_1 \dots Y_n, \mathcal{H}) &= \frac{P(Y_1 \dots Y_n, X_j = a|\mathcal{H})}{P(Y_1 \dots Y_n|\mathcal{H})} \qquad \text{(D.11)} \\ &= \frac{f_j(a) b_j(a)}{\sum_{a'} f_n(a')} \qquad \text{(D.12)} \end{aligned}$$

The denominator in equation (D.12) is independent of j, so is not required for the solution of problem 2: $\operatorname{argmax}_a P(X_j = a|Y_1 \dots Y_n, \mathcal{H}) = \operatorname{argmax}_a f_j(a) b_j(a)$.

Problem 3

There is no guarantee that the state sequence found above is valid. That is, the sequence of most probable states might include $X_j = a$ and $X_{j+1} = b$ even though that transition is disallowed (*i.e.* $q(Y_j|a,b) = 0$ or $p(b|a) = 0$). The procedure for finding the most likely valid state sequence given the observation sequence is known as the Viterbi algorithm. It is similar to the forward/backward procedure outlined above. We define

$$v_j(a) = \max_{a' \in \mathcal{A}} v_{j-1}(a')\, p(a|a')\, P(Y_j|a', a) \tag{D.13}$$

$$w_j(a) = \operatorname*{argmax}_{a' \in \mathcal{A}} v_{j-1}(a')\, p(a|a')\, P(Y_j|a', a) \tag{D.14}$$

We initialise v as per f: $v_0(a) = f_0(a)$. The quantity $v_j(a)$ is the probability of the most probable path that outputs $Y_1 \dots Y_j$ and ends up in state $X_j = a$. $w_j(a)$ is used to retrace our steps after the most likely final state is found. Together, v and w allow us to find the maximum likelihood path $X_1^* \dots X_n^*$:

$$X_n^* = \operatorname*{argmax}_{a \in \mathcal{A}} v_n(a) \tag{D.15}$$

$$X_i^* = w_{i+1}(X_{i+1}^*) \qquad 0 < i < n. \tag{D.16}$$

$$\Longrightarrow X_1^* \dots X_n^* = \operatorname*{argmax}_{X_1 \dots X_n} P(X_1 \dots X_n, Y_1 \dots Y_n|\mathcal{H}) \tag{D.17}$$

D.2.4 Notes

It is not necessary to impose a unique initial state $X_0 = s$. It is common to provide a distribution $P(X_0 = a),\ a \in \mathcal{A}$. The required changes to the above algorithms are trivial.

We have defined Markov chains to have one-step memory. It is easy to recast a model with m-step memory as a one-step model by using a change of variables. For example, if Z_i depends on m previous states, we can define a new sequence of variables $\{X_i :\ X_i = Z_{i-m+1} \dots Z_i\}$. The variables X_i have a one-step memory, satisfying equation D.2.

Hidden Markov models are often formulated such that the output symbols are produced by states rather than by transitions. The two formulations are entirely equivalent, as is readily seen. Firstly, if $q(y|a)$ is a state-dependent output distribution, we can immediately obtain a transition-dependent output distribution q' by setting $q'(y|a,b) = q(y|a)$. Secondly, given a model with transition-dependent outputs and state space $\mathcal{A}$, we can form a state-dependent model with state space $\mathcal{A} \times \mathcal{A}$, where each state corresponds to a transition in the original model. The correspondance between transition probabilities is straightforward.

Finally, we mention an implementation issue. The forward-backward probabilities $f_i(a)$ and $g_i(a)$ tend to zero geometrically fast. Hence, for long sequences (often $n > 1000$) it is necessary to perform some sort of scaling to avoid numerical underflow. We refer the reader to Jelinek [43] for an appropriate method.

Appendix E

Viterbi decoding of watermark codes

When decoding watermark codes, a Viterbi min/sum algorithm may be used to scoring synchronisation alternatives when decoding a stream with unknown synchronisation. As mentioned in section 7.5.2, this method is deprecated. The Viterbi path is often not a 'typical' path, in that the number of insertions, deletions and transitions may vary significantly from the expected and observed counts.

Figure E.1 shows the errors made by Viterbi decoding for three choices of channel parameters. The top two panels are fairly well synchronised, but the bottom panel shows a loss of global synchronisation. Some insight into the errors made by Viterbi decoding is given by the central panel. It is clear that a gross underestimate of the number of insertions and deletions has been made. The Viterbi path often deviates significantly from the true path.

When insertion/deletion errors are improbable and substitutions are likely, the Viterbi path favours substitutions over insertions and deletions when explaining errors. In this case, whenever a run of deletions is followed by balancing insertions, the maximum likelihood path favours substitutions over the more expensive insertion/deletion events and takes a 'short cut'. See, for example, the regions 12000 – 14000 and 16000 – 18000 in the central panel.

At the other end of the spectrum, when insertion/deletion errors are more likely the Viterbi path makes large detours (at low cost) if it can reduce the number of substitutions.

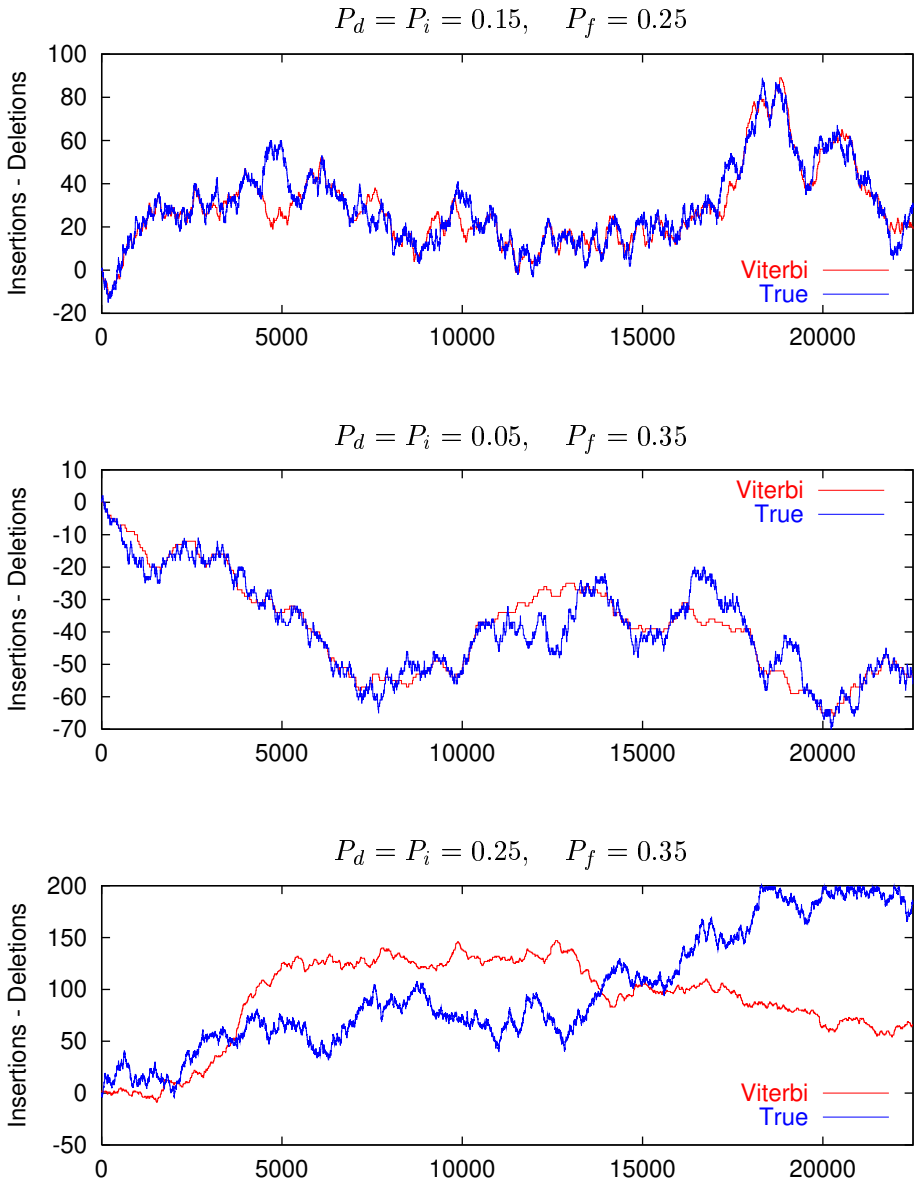


Figure E.1: Excess (insertions − deletions) for several settings of channel parameters. Synchronisation known at start of block. Comparison is between decoded (Viterbi) and true excess, as a function of block position. The fraction of synchronised positions was roughly 0.2 for the top two panels, and 10^{-3} for the bottom panel.

Appendix F

Tables of LDPC codes

This appendix lists the parameters of all the LDPC codes mentioned in this thesis. The parameters are generally given in terms of row and column profiles (see Chapter 4). All blocklengths are given in bits.

F.1 Codes in figure 1.1

'Irreg GF(8)': Blocklength 48000, Rate 0.25, mean column weight 3.26

Column weight	Column fraction	Row weight	Row fraction
2	0.701125	4	0.653167
3	0.1805	5	0.346833
5	0.01375		
7	0.0335		
9	0.02375		
11	0.0105		
13	0.004875		
14	0.000875		
15	0.01425		
17	0.00275		
19	0.00475		
23	0.004625		
29	0.003375		
37	0.001		
43	0.000375		

'Reg GF(16)': Blocklength 24448, rate 0.262, mean column weight 2.308

Column weight	Column fraction	Row weight	Row fraction
2	0.692	3	0.875
3	0.308	4	0.125

'Irreg GF(2)': Blocklength 16000, rate 0.25, mean column weight 3.40

Column weight	Column fraction	Row weight	Row fraction
2	0.7175625	3	0.0084167
3	0.0259375	4	0.454167
5	0.1340625	5	0.5374167
7	0.08175		
9	0.000075		
11	0.012875		
13	0.0163125		
17	0.0011875		
19	0.00425		
23	0.0008125		
29	0.001625		
33	0.000125		
37	0.002125		
43	0.0001875		
53	0.0004375		

'Luby': Blocklength 64000, rate 0.25, mean column weight 8.0

Column weight	Column fraction	Row weight	Row fraction
3	0.445064	4	0.428373
5	0.267039	10	0.432045
9	0.148355	33	0.098185
17	0.078541	34	0.041397
33	0.040460		
65	0.020541		

'Reg GF(2)': Blocklength 40000, rate 0.25, mean column weight 2.625

Column weight	Column fraction	Row weight	Row fraction
2	0.375	3	0.5
3	0.625	4	0.5

F.2 Codes in figure 4.2

Luby A: As per code 'Luby' in figure 1.1.

Luby B: Profile as per 'Luby A', but with blocklength 64000 bits.

LDPC Empirical: Identical to code 'Irreg GF(2)' in figure 1.1.

F.3 Codes in figure 4.3

GF(8), 1/4: This code is the same as code 'Irreg $GF(8)$' of figure 1.1

'GF(4) 1/2': Blocklength 60000, Rate 0.5, mean column weight 3.75

Column weight	Column fraction	Row weight	Row fraction
2	0.478633	7	0.341
3	0.4085	8	0.659
8	0.000067		
11	0.045233		
17	0.067567		

F.4 Code in figure 5.1

GF(8): Blocklength 4800, Rate 0.25, mean column weight 3.22

Column weight	Column fraction	Row weight	Row fraction
2	0.679375	3	0.00667
3	0.229375	4	0.695
5	0.001875	5	0.29833
13	0.089375		

F.5 Regular Codes

Codes in figure 3.2

(left) GF(2) ,GF(4), GF(8): Column weight 3, Blocklength 3000, rate 0.5.

(right) GF(2) ,GF(4), GF(8): Column weight 3, Blocklength 8000, rate 0.25.

Codes in figure 3.6

GF(2), GF(4), GF(8): Blocklength 18000, rate 0.33, mean column weight 2.5

Column weight	Column fraction	Row weight	Row fraction
2	0.5	3	0.25
3	0.5	4	0.75

The three codes $GF(2)$, $GF(4)$ and $GF(8)$ have identical construction, differing only in the field over which they are defined.

Codes in figure 7.13

Figure 7.13 compares watermark codes concatenated with outer low-density parity-check codes over $GF(8)$ with rates 0.643, 0.57, 0.5, 0.43 and 0.36. All are of regular construction, as per the following table.

Rate	Blocklength	Mean column weight	Mean row weight
0.643	1554	3	8.4
0.57	1749	3	7
0.5	1998	3	6
0.43	2331	2.8	4.9
0.36	2796	2.8	4.35

Codes in figure 7.15

Figure 7.15 compares watermark codes of several different rates. We give the parameters of the outer codes from left to right. All are of regular construction.

Field	Rate	Blocklength	Mean column weight	Mean row weight
$GF(16)$	0.89	3996	3	27
$GF(16)$	0.625	3200	2.8	7.4667
$GF(16)$	0.75	2668	3	12
$GF(8)$	0.43	2331	2.8	4.9
$GF(8)$	0.5	1998	3	6
$GF(8)$	0.1	3000	2.6	2.889

F.6 Miscellaneous

The following profile can be used to construct a good short blocklength code over $GF(4)$. A blocklength 1920 bits code constructed according to this profile appears on the JPL code imperfectness webpage [46].

Short blocklength code for $GF(4)$. Rate 1/3, mean column weight 3.34			
Column weight	Column fraction	Row weight	Row fraction
2	0.4677	5	0.99375
3	0.4052	6	0.00625
4	0.0760		
8	0.0042		
11	0.0021		
13	0.0042		
19	0.0406		

Bibliography

[1] J.C. Amengual and E. Vidal. Efficient error-correcting viterbi parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1109–1116, October 1998.

[2] S. Austin, R. Schwartz, and P. Placeway. The forward-backward search algorithm. In *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*, pages 697–700, Cape Cod, MA, May 1991. Institute of Electrical and Electronic Engineers.

[3] L.R. Bahl and F. Jelinek. Decoding for channels with insertions, deletions and substitutions with applications to speech recognition. *IEEE Transactions on Information Theory*, 21(4):404–411, July 1975.

[4] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara. Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding. Technical Report TDA 42-126, Jet Propulsion Laboratory, Pasadena, August 1996.

[5] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. On the intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *Proc. 1993 IEEE International Conference on Communications, Geneva, Switzerland*, pages 1064–1070, 1993.

[7] M. Blaum, P.G. Farrell, and H.C.A. van Tilborg. Multiple burst-correcting array codes. *IEEE Transactions on Information Theory*, 34(5):1061–1066, September 1988.

[8] A. Bouloutas, G.W. Hart, and M. Schwartz. Two extensions of the viterbi algorithm. *IEEE Transactions on Information Theory*, 37(2):430–436, March 1991.

[9] P. A. H. Bours. *Codes for Correcting Insertion and Deletion Errors.* PhD thesis, Eindhoven Technical University, June 1994. Available from `http://www.win.tue.nl/math/dw/pp/wsdwpb/thesis.html`.

[10] L. Calabi and William E. Hartnett. A family of codes for the correction of substitution and synchronization errors. *IEEE Transactions on Information Theory*, 15(1):102–106, January 1969.

[11] Y.-L. Chow and R. Schwartz. The n-best algorithm: An efficient procedure for finding top n sentence hypotheses. In *Proc. of the Speech and Natural Language Workshop*, pages 199–202, Cape Cod, MA, 1989.

[12] Sae-Young Chung. Personal communication. 1999.

[13] F.H.C. Crick, J.S. Griffith, and L.E. Orgel. Codes without commas. *Proceedings of the National Academy of the Sciences of the USA*, 43:416–421, 1957.

[14] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the Association for Computing Machinery*, 7(3):171–176, March 1964.

[15] M. C. Davey and D. J. C. MacKay. Low density parity check codes over GF(q). *IEEE Communications Letters*, 2(6):165–167, June 1998.

[16] M. C. Davey and D. J. C. MacKay. Low density parity check codes over GF(q). In *Proceedings of the 1998 IEEE Information Theory Workshop*, pages 70–71. IEEE, June 1998.

[17] M. C. Davey and D. J. C. MacKay. Monte Carlo simulations of infinite low density parity check codes over $GF(q)$. In *Proceedings of the 2nd international workshop on optimal codes and related topics*. Bulgarian Institute of Mathematics and Informatics, June 1998. Available from `http://wol.ra.phy.cam.ac.uk/is/papers/`.

[18] D. Divsalar, H. Jin, and R. J. McEliece. Coding theorems for 'turbo–like' codes. In *Proceedings of the 36th Allerton Conference on Communication, Control, and Computing, Sept. 1998*, pages 201–210, Monticello, Illinois, 1998. Allerton House.

[19] D. Drasdo, T. Hwa, and M. Lässig. Scaling laws and similarity detection in sequence alignment with gaps. Submitted to the Journal of Computational Biology, 1999.

[20] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK, 1998.

[21] W.L. Eastman. On the construction of comma-free codes. *IEEE Transactions on Information Theory*, 11(2):263–267, April 1965.

[22] P. Elias. Coding for noisy channels. *IRE Convention Record*, Part 4(37–46), 1955.

[23] G. D. Forney, Jr. *Concatenated Codes*. MIT Press, Cambridge, Mass., 1966.

[24] A.M. Fraser and H.L. Swinney. Independent coordinates for strange attractors from mutual information. *Physical Review A*, 33(2):1134–1140, February 1986.

[25] R. G. Gallager. Low density parity check codes. *IRE Transactions on Information Theory.*

[26] R. G. Gallager. *Low Density Parity Check Codes.* PhD thesis, MIT, Cambridge, Mass., September 1960.

[27] R. G. Gallager. *Low Density Parity Check Codes.* Number 21 in Research monograph series. MIT Press, Cambridge, Mass., 1963.

[28] E.N. Gilbert. Synchronization of binary messages. *IRE Transactions on Information Theory*, 6(4):470–477, September 1960.

[29] S.W. Golomb, B. Gordon, and L.R. Welch. Comma-free codes. *Canadian Journal of Mathematics*, 10(2):202–209, 1958.

[30] P. Grassberger and I. Procaccia. Characterisation of strange attractors. *Physical Review Letters*, 50:346–369, 1983.

[31] J. Hagenauer, E. Offer, C. Measson, and M. Mrz. Decoding and equalization with analog non-linear networks. *European Transactions on Telecommunications (ETT)*, October 1999.

[32] J. Hagenauer and M. Winklhofer. The analog decoder. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, page 145, 1998.

[33] P. Hall and G. Dowling. Approximate string matching. *Computing Surveys*, 12:381–402, 1980.

[34] G.W. Hart. Personal communication. 1999.

[35] G.W. Hart and A.T. Bouloutas. Correcting dependent errors in sequences generated by finite-state processes. *IEEE Transactions on Information Theory*, 39(4):1249–1260, July 1993.

[36] T.R. Hatcher. On a family of error-correcting and synchronizable codes. *IEEE Transactions on Information Theory*, 15(5):620–624, September 1969.

[37] D. Haussler. Computational genefinding. Available from `http://www.cse.ucsc.edu/~haussler/pubs.html`, 1998.

[38] D. Haussler, A. Krogh, I.S. Mian, and K. Sjölander. Protein modelling using hidden Markov models: analysis of globins. In T.N. Mudge, V. Milutinovic, and L. Hunter, editors, *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, volume 1, pages 792–802, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[39] A.S.J. Helberg. *Coding for the correction of Synchronization Errors.* PhD thesis, Faculty of Engineering, Rand Afrikaans University, November 1993.

[40] I. Holmes and R. Durbin. Dynamic programming alignment accuracy. *Journal of Computational Biology*, 5(3):493–504, 1998.

[41] S. Hui, J. Bond, and H. Schmidt. Constructing low-density parity-check codes with circulant matrices. In *Proceedings of the 1999 IEEE Information Theory Workshop*, page 50, 1999. Greece, June 17 – July 1, 1999.

[42] T. Hwa and M. Lässig. Similarity detection and localization. *Physical Review Letters*, 76(14):2591–2595, 1996.

[43] F. Jelinek. *Statistical Methods for Speech Recognition.* The MIT Press, Cambridge, Massachusetts, 1998.

[44] H. Jin and R.J. McEliece. RA codes achieve AWGN channel capacity. To appear in Proceedings of the 13th Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, November 1999.

[45] Turbo codes performance. Available from `http://www331.jpl.nasa.gov/public/TurboPerf.html`, August 1996.

[46] JPL. Code imperfectness. `www331.jpl.nasa.gov/public/imperfectness.html`, 1999.

[47] K. Karplus and H. Krit. A semi–systolic decoder for the PDSC–73 error–correcting code. *Discrete Applied Mathematics*, 33:109–128, 1991.

[48] N. Kashyap and D. L. Neuhoff. Codes for data synchronization and timing. In *Proceedings of the 1999 IEEE Information Theory and Communications Workshop*, pages 63–65. IEEE, 1999. IEEE Catalog Number 99EX253.

[49] A. Krogh. Hidden Markov models for labeled sequences. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, pages 140–144, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[50] K. Kukich. Techniques for automatically correcting words in text. *Computing Surveys*, 24:377–439, 1992.

[51] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady*, 10(8):707–710, February 1966.

[52] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, Cambridge, 1994.

[53] S. Lin and D. J. Costello, Jr. *Error control coding: fundamentals and applications.* Prentice-Hall, Englewood Cliffs, N.J., 1983.

[54] H-A Loeliger, F. Tarköy, F. Lustenberger, and M. Helfenstein. Decoding in analog vlsi. To appear in Proceedings of IMA workshop on Codes, Systems and Graphical Models 1999. Available from `http://www.ima.umn.edu/csg/`, 1999.

[55] B. Lowerre. *The HARPY speech recognition system.* PhD thesis, Carnegie-Mellon University, April 1976.

[56] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Improved low-density parity-check codes using irregular graphs and belief propagation. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, page 117, 1998.

[57] M. G. Luby, M. Mitzenmacher, M. Amin Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*, 1997.

[58] D. J. C. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.

[59] D. J. C. MacKay and M. C. Davey. Evaluation of Gallager codes for short block length and high rate applications. To appear in Proceedings of IMA workshop on Codes, Systems and Graphical Models 1999. Available from `wol.ra.phy.cam.ac.uk/mackay/`, 1998.

[60] D. J. C. MacKay and C. P. Hesketh. Performance of low density parity check codes as a function of actual and assumed noise levels. `http://wol.ra.phy.cam.ac.uk/mackay/abstracts/sensit.html`. To appear, IEEE Trans. Communications 1999, 1997.

[61] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In Colin Boyd, editor, *Cryptography and Coding. 5th IMA Conference*, number 1025 in Lecture Notes in Computer Science, pages 100–111. Springer, Berlin, 1995.

[62] D. J. C. MacKay, S. T. Wilson, and M. C. Davey. Comparison of constructions of irregular Gallager codes. *IEEE Transactions on Communications*, 47(10):1449–1454, October 1999.

[63] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes.* North-Holland, Amsterdam, 1977.

[64] B. H. Marcus, P. H. Siegel, and J. K. Wolf. Finite-state modulation codes for data storage. *IEEE Journal on Selected Areas in Communication*, 10(1):5–38, January 1992.

[65] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng. Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, February 1998.

[66] P.G. Neumann. On a class of efficient error-limiting variable-length codes. *IEEE Transactions on Information Theory*, 8(5):260–266, September 1962.

[67] H. Ney, D. Mergel, A. Noll, and A. Paesler. Data driven search organization for continuous speech recognition. *IEEE Transactions on Signal Processing*, 40(2):272–281, February 1992.

[68] N.H. Packard, J.P. Crutchfield, J.D. Farmer, and R.S. Shaw. Geometry from a time series. *Physical Review Letters*, 45(9):712–716, 1980.

[69] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, 1988.

[70] W.H. Press, B.P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C.* Cambridge, 1988.

[71] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[72] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, 8:300–304, 1960.

[73] M.A. Ribeiro. Optimal bit-level synchronization strategy for digital magnetic recorders. In *Proceedings of the 14th Symposium on Information Theory in the Benelux*, pages 222–227, 1993.

[74] T. Richardson, A. Shokrollahi, and R. Urbanke. Design of provably good low-density parity-check codes. Submitted to IEEE Transactions on Information Theory, 1999.

[75] T. Richardson and R. Urbanke. The capacity of low-density parity check codes under message-passing decoding. Submitted to IEEE Transactions on Information Theory, 1998.

[76] D. Sankoff and J. B. Kruskal, editors. *Time Warps, string edits, and macromolecules: the theory and practice of sequence comparisons.* Addison-Wesley, Reading, MA, 1983.

[77] R. Schwartz and S. Austin. A comparison of several approximate algorithms for finding multiple (n-best) sentence hypotheses. In *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*, Cape Cod, MA, May 1991. Institute of Electrical and Electronic Engineers.

[78] F. F. Sellers, Jr. Bit loss and gain correction code. *IRE Transactions on Information Theory*, 8(1):35–38, January 1962.

[79] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.

[80] V. Sorokine, F. R. Kschischang, and S. Pasupathy. Gallager codes for CDMA applications: Generalizations, constructions and performance. In *Proceedings of the 1998 IEEE Information Theory Workshop*, pages 8–9. IEEE, June 1998.

[81] J.J. Stiffler. Comma-free error-correcting codes. *IEEE Transactions on Information Theory*, 11(1):107–111, January 1965.

[82] E. Tanaka and T. Kasai. Synchronization and substitution error-correcting codes for the levenshtein metric. *IEEE Transactions on Information Theory*, 22(2):156–162, March 1976.

[83] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[84] S.E. Tavares and M. Fukada. Matrix approach to synchronization recovery for binary cyclic codes. *IEEE Transactions on Information Theory*, 15(1):93–101, January 1969.

[85] G. Tenengolts. Nonbinary codes, correcting single deletion or insertion. *IEEE Transactions on Information Theory*, 30(5):766–769, September 1984.

[86] J.D. Ullman. On the capabilities of codes to correct synchronization errors. *IEEE Transactions on Information Theory*, 13(1):95–105, January 1967.

[87] R.R. Varshamov and G.M. Tenengolts. One asymmetrical error correcting codes. *Automatics and Telemechanics* (Russian), 26(2):288–292, 1965.

[88] B. Vasic. Personal communication. Lucent Technologies, 1999.

[89] R.A. Wagner. Order-n correction for regular languages. *Communications of the Association for Computing Machinery*, 17(5):265–268, May 1974.

[90] M.S. Waterman. *Introduction to Computational Biology.* Chapman and Hall, London, 1995.

[91] N. Wiberg. *Codes and Decoding on General Graphs.* PhD thesis, Dept. of Electrical Engineering, Linköping, Sweden, 1996. Linköping studies in Science and Technology. Dissertation No. 440.

[92] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the Association for Computing Machinery*, 30(6):520–540, 1987.

[93] A.P. Worthen and W.E. Stark. Low-density parity check codes for fading channels with memory. In *Proceedings of the 36th Allerton Conference on Communication, Control, and Computing, Sept. 1998*, pages 117–125, Monticello, Illinois, November 1998. Allerton House.

[94] V. V. Zyablov and M. S. Pinsker. Estimation of the error–correction complexity for Gallager low–density codes. *Problemy Peredachi Informatsii*, 11(1):23–36, 1975.