

Error-correction on non-standard communication channels

Edward A. Ratzer
Christ's College
Cambridge

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

Inference Group
Cavendish Laboratory
University of Cambridge



September 2003

DECLARATION

I hereby declare that my dissertation entitled “Error-correction on non-standard communication channels” 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:

Edward A. Ratzer
Christ’s College
Cambridge
September, 2003

ABSTRACT

Many communication systems are poorly modelled by the standard channels assumed in the information theory literature, such as the binary symmetric channel or the additive white Gaussian noise channel. Real systems suffer from additional problems including time-varying noise, cross-talk, synchronization errors and latency constraints. In this thesis, low-density parity-check codes and codes related to them are applied to non-standard channels.

First, we look at time-varying noise modelled by a Markov channel. A low-density parity-check code decoder is modified to give an improvement of over 1dB.

Secondly, novel codes based on low-density parity-check codes are introduced which produce transmissions with $\Pr(\text{bit} = 1) \neq \Pr(\text{bit} = 0)$. These non-linear codes are shown to be good candidates for multi-user channels with crosstalk, such as optical channels.

Thirdly, a channel with synchronization errors is modelled by random uncorrelated insertion or deletion events at unknown positions. Marker codes formed from low-density parity-check codewords with regular markers inserted within them are studied. It is shown that a marker code with iterative decoding has performance close to the bounds on the channel capacity, significantly outperforming other known codes.

Finally, coding for a system with latency constraints is studied. For example, if a telemetry system involves a slow channel some error correction is often needed quickly whilst the code should be able to correct remaining errors later. A new code is formed from the intersection of a convolutional code with a high rate low-density parity-check code. The convolutional code has good early decoding performance and the high rate low-density parity-check code efficiently cleans up remaining errors after receiving the entire block. Simulations of the block code show a gain of 1.5dB over a standard NASA code.

ACKNOWLEDGEMENTS

My interest in error correction started in my last undergraduate year in Cambridge while attending David MacKay's course "Information Theory, Pattern Recognition and Neural Networks" and being supervised by him for my MSci project. For the last three years David MacKay has been a PhD supervisor always full of ideas and encouragement.

EPSRC and Schlumberger have generously funded me for the duration of my studies. Their funding has allowed me to attend several conferences which have kept me up-to-date with the academic side of the field. Discussions with Astrium, Azea, IBM and Schlumberger have been invaluable in keeping in touch with industrial applications. Work in Appendix H was done while visiting the IBM Zurich Research Laboratory.

I would like to thank all the members of my research group in Cambridge: Seb Wills, Sanjoy Mahajan, David Ward, James Miskin, Phil Cowans, John Winn, John Barry and Hanna Wallach, for being ready to bounce ideas off and looking over draft papers. The members of Cambridge URNU also deserve thanks for being willing volunteers for experimental work.

LIST OF PUBLICATIONS

Edward A. Ratzert. Convolutional code performance as a function of decoding delay. In *6th International Symposium on Communication Theory and Applications*, July 2001.

Edward A. Ratzert. Low-density parity-check codes on Markov channels. In *Second IMA Conference on Mathematics in Communications*, December 2002.

Edward A. Ratzert and David J.C. MacKay. Sparse low-density parity-check codes for channels with cross-talk. In *IEEE Information Theory Workshop*, pages 127–130, April 2003.

Edward A. Ratzert. Intersected low-density parity-check and convolutional codes. In *Post-graduate Research Conference in Electronics, Photonics, Communications and Software*, April 2003.

Edward A. Ratzert. Sparse data blocks and multi-user channels. In *International Symposium on Information Theory*, July 2003.

Edward A. Ratzert. Intersected low-density parity-check and convolutional codes. In *7th International Symposium on Communication Theory and Applications*, July 2003.

Edward A. Ratzert. Marker codes for channels with insertions and deletions. In *3rd International Symposium on Turbo Codes and Related Topics*, September 2003.

Edward A. Ratzert. Marker codes for channels with insertions and deletions. *Annals of Telecommunications*, August 2004. To appear.

CONTENTS

Chapter 1	Introduction	1
	1.1 Forward Error Correction	1
	1.2 Random codes	2
	1.3 Algebraic coding theory	2
	1.4 Modern Coding	4
	1.5 Thesis outline	5
Chapter 2	Low-Density Parity-Check Codes	6
	2.1 History	6
	2.2 Belief Propagation Decoding	7
	2.3 Construction	10
	2.4 Basic simulations	12
	2.5 Codes over $\text{GF}(q)$	12
Chapter 3	Markov Channels	15
	3.1 Introduction	15
	3.2 Markov channels	15
	3.3 Modifications to the decoding algorithm	17
	3.4 Performance on the Markov channel	20
	3.5 Discussion	26
	3.6 Conclusion	27
Chapter 4	Multi-User Channels	28
	4.1 Introduction	28
	4.2 Channel Models	29
	4.3 Capacity	30
	4.4 Time-sharing	31
Chapter 5	Sparse-Dense Codes	32
	5.1 Introduction	32
	5.2 Sparsifiers	32
	5.3 Sparse-Dense Codes	37

	5.4 Conclusion	39
Chapter 6	Sparse LDPC Codes	40
	6.1 Introduction	40
	6.2 Binary Low-Density Parity-Check Codes	40
	6.3 Sparse LDPC codes	42
	6.4 Simulation results	45
	6.5 Discussion	45
	6.6 Conclusion	48
Chapter 7	Insertion-Deletion Channels	49
	7.1 Introduction	49
	7.2 Probabilistic Resynchronization	51
	7.3 Comparison with Watermark Codes	54
	7.4 A Complete Iterative System	55
	7.5 EXIT charts	58
	7.6 Deletion Channel	60
	7.7 Conclusion	62
Chapter 8	Convolutional Codes	63
	8.1 Introduction	63
	8.2 Forward-backward algorithm	65
	8.3 Experiments	66
	8.4 Discussion	71
	8.5 Conclusion	71
Chapter 9	Intersected LDPC and Convolutional Codes	72
	9.1 Introduction	72
	9.2 Code Construction	72
	9.3 EXIT chart analysis	75
	9.4 Threshold Optimization	75
	9.5 Code construction results	77
	9.6 Comparison with a deep-space standard	79
	9.7 Discussion	80
	9.8 Conclusion	81
Chapter 10	Conclusion	82
	10.1 Summary	82
	10.2 Key Insights	82
	10.3 Future directions	83

Appendix A	Information Theory	85
A.1	Definitions	85
A.2	Source coding	86
A.3	Channel coding	86
Appendix B	Simulations	88
B.1	Basic channels	88
B.2	Simulation of one block	90
B.3	Calculation of error probabilities	91
Appendix C	Finite Field Arithmetic	93
Appendix D	Linear Block Codes	94
D.1	Definition	94
D.2	Properties	94
D.3	Generator Matrix	95
Appendix E	Convolutional Codes	96
Appendix F	Factor Graphs	98
Appendix G	The Hat Problem	101
G.1	Introduction	101
G.2	The code	103
G.3	Experiment	103
G.4	Postscript	105
Appendix H	Complexity of LDPC belief propagation decoding	106
H.1	Complexity of the GF(2) Non-Fourier algorithm	106
H.2	Fourier Representation	110
H.3	Complexity of the Fourier Decoding algorithm	112
H.4	The best algorithm for binary codes	114
H.5	Conclusion	116
Appendix I	EXIT Charts	117
I.1	Introduction	117
I.2	Log-likelihood messages	117
I.3	Variable nodes	119
I.4	Check nodes	119
I.5	Irregular ensembles	120
I.6	Combined sections	120
	Bibliography	121

CHAPTER 1

INTRODUCTION

1.1 Forward Error Correction

We are all familiar with noise on the radio. If the noise level gets so high that we miss something, we can not get back what we missed. This is a common problem in many areas of communication and has led to the development of “forward error-correction” or “channel coding”. In forward error-correction we add extra redundancy to our original signal and then hope we can reconstruct the original transmission from this new signal when it is corrupted by noise. In 1948 Shannon [93] showed that there was a theoretical solution that would allow an arbitrarily small probability of error under certain constraints. This thesis will look at situations where these constraints do not hold or are not suitable.

We will look at the field of binary communication – where we transmit a sequence of 0s and 1s over a system like that illustrated in figure 1.1. For example, on a fibre optic channel 1s could correspond to the presence of light and 0s to no light. This is one possible type of modulation; many modulation schemes exist for different types of channel [81]. The receiver observes the original sequence corrupted by noise. There are two main classes of receivers. “Hard-decision” receivers report a best guess of each transmitted symbol (for the fibre-optic channel this could be obtained from a threshold of the observed light level). “Soft-decision” receivers report analogue information, ideally $\Pr(\text{received signal}|\text{transmitted bit} = 1)$ and

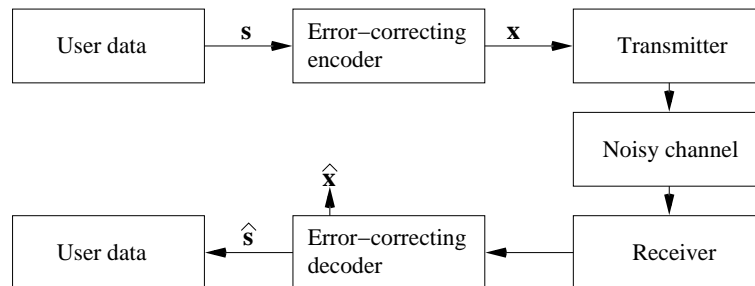


Figure 1.1: Schematic of a communication system

$\Pr(\text{received signal} | \text{transmitted bit} = 0)$. For the fibre-optic channel this could be obtained by observing the light intensity and using a knowledge of the channel characteristics. In Appendix B.1 simple standard channel models that approximate some real systems are presented.

In forward-error correction we are interested in forming an error-correcting code – a set of allowed sequences, E . Redundancy for forward-error correction can then be obtained as not all sequences are allowed. We define a one-to-one mapping from input sequences \mathbf{s} to transmitted sequences $\mathbf{x} \in E$. In this thesis we will look at codes where there is a fixed ratio between the length of these sequences, $R = \frac{\text{length } \mathbf{s}}{\text{length } \mathbf{x}}$. R is a measure of redundancy called the code rate and corresponds to the number of information bits carried per symbol transmitted on the channel.

There are two main classes of error-correcting codes. The first are block codes for which lengths \mathbf{s} and length \mathbf{x} are fixed. Length \mathbf{x} is the blocksize of the code, commonly given the symbol N . The second class are stream or convolutional codes for which only R is fixed and there are no block boundaries. Commonly stream codes are implemented by an encoder which, for example, outputs 2 bits for every 1 bit of input. This would then define a stream code of $R = 1/2$.

1.2 Random codes

Shannon [93] showed that on a channel with stationary ergodic noise of a fixed power there exists a rate C called the channel capacity. For $R < C$ transmission at an arbitrarily small error probability is possible with a large random block code (meaning that the set of codewords is chosen at random). For $R > C$ arbitrarily reliable transmission is not possible.

To get a particular error probability with a random block code one needs to choose the blocksize N . To get a smaller error probability, N needs to be larger. For a code to be practical we would like an $\mathcal{O}(N)$ complexity encoder and decoder so we can choose a larger blocksize without a penalty on the amount of computation per bit. Random block codes are not practical. One could imagine a simple encoder and decoder based on the set of codewords. Such an encoder and decoder are both $\mathcal{O}(Ne^N)$ in space as they need to store all the codewords (there are $\mathcal{O}(e^N)$ codewords each of length N). The maximum-likelihood decoder is $\mathcal{O}(Ne^N)$ in time – the decoder looks through every codeword and evaluates the probability of the received signal having been produced by the channel given that codeword. More elaborate algorithms could be used, for example with decision trees, but no algorithm is known that is better than exponential [69].

So instead more structured codes with tractable encoders and decoders were studied.

1.3 Algebraic coding theory

Algebraic coding theory mainly deals with noise from the binary symmetric channel (as defined in appendix B.1) which flips bits with probability p_f . The simplest family of error-

correcting codes for such a channel are the repetition codes R_N . For the binary case R_N consists of two codewords: the codeword of N 1s and the codeword of N 0s. The code has $R = 1/N$. To encode, we take one input bit and repeat it N times. On receiving an R_N codeword corrupted by a binary symmetric channel, the maximum likelihood decoder returns the bit that forms the majority of the received noisy codeword (if $p_f < 1/2$) [71].

In [50] the idea of looking at error-correcting codes from the point of view of the “minimum distance”, d_{\min} , was introduced. One can define a distance (the Hamming distance) between two codewords as the number of positions where the codewords have different bits. d_{\min} is the minimum distance between any two codewords in the code. For the R_N code $d_{\min} = N$ as the only pair of codewords differ in every position.

One can characterise the maximum number of bit flips that a code can *always* correct: a code can always correct $\lfloor \frac{d_{\min}-1}{2} \rfloor$ bit flip errors [45]. For example the R_3 code can correct 1 bit-flip error as one wrong “vote” can always be overruled in a set of three “votes”. The R_2 code can not correct a single bit-flip error.

Algebraic coding focused on the development of codes with high values of d_{\min} for particular values of R and N . Many clever codes have been developed [10, 44, 52, 84, 85] with “bounded-distance” decoders that allow the one codeword (if it exists) within a distance of $\lfloor \frac{d_{\min}-1}{2} \rfloor$ or less of the received word to be found with tractable algorithms (polynomial time). For a summary of algebraic coding with decoding up to $d_{\min}/2$ see [7]. Decoders for algebraic codes which can decode slightly beyond $d_{\min}/2$ are starting to appear [47, 75].

The Gilbert-Varshamov lower bound [43] (which is believed to be tight [71]) says that for any binary code

$$R \approx 1 - H_2(d_{\min}/N) \quad (1.1)$$

where $H_2(\cdot)$ is the binary entropy (see appendix A for a definition of $H_2(\cdot)$ and other information theory concepts). If we set the expected number of bit flip errors to be the same as the best we hope to be able to successfully decode with a bounded-distance decoder ($Np_f \approx d_{\min}/2$), we get a maximum code rate of

$$R \approx 1 - H_2(2p_f) \quad (1.2)$$

whereas Shannon says the capacity of a binary symmetric channel is

$$C = 1 - H_2(p_f) \quad (1.3)$$

So a bounded-distance decoder has a fundamental problem: if one decodes a code with such a decoder it is impossible to reach Shannon’s channel capacity for the BSC. To achieve capacity we need to generally be able to decode to twice the minimum distance.

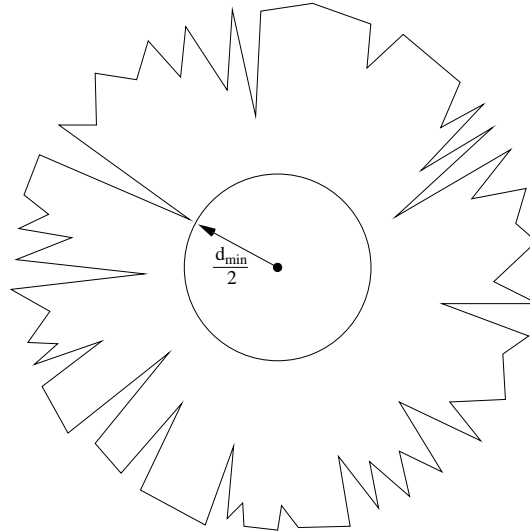


Figure 1.2: A schematic illustration of the space around a codeword (after [71]). The jagged edge is the boundary inside which the chosen codeword is the closest. The space where a bounded distance decoder works takes up a small fraction of the allowed space. The diagram has to be schematic as a codeword lives in N -dimensional Hamming space, not the 2-dimensional Euclidean space of the page.

1.4 Modern Coding

At low noise levels it is important to look at the number of codewords at the minimum distance – for many codes this is low. In a high dimensional space the probability of a random direction from a codeword heading towards a particular codeword is low. Berlekamp visualises the space around a codeword as a “bat cave” [6]. One can draw the idea schematically in 2-dimensions, figure 1.2. A bat heading off in a random direction has a low chance of hitting the edge of the cave in a distance of approximately $d_{\min}/2$. If the noise level is such that about $d_{\min}/2$ errors occur, decoding errors are caused by codewords at small distance. If the number of these codewords is low the error probability is small.

At higher noise levels, errors are caused by codewords at larger distance as there are more of them within reach. To create a code with good general performance we need to worry about the number of codewords at all distances (the distance spectrum). For a random code an excursion in most directions can reach a distance of more like d_{\min} without hitting the edge.

With the development of Turbo codes in 1993 [8, 9], it was realised that practical codes that allow capacity-approaching performance were possible. At about the same time low-density parity-check codes were rediscovered [67] and have been shown to get within a whisker of the capacity of the additive white Gaussian noise (AWGN) channel with large block size [18]. Both of these families of codes are random-like block codes [3].

The AWGN channel is not a good model for many real systems and the large block sizes

of these modern codes can be a problem. In this thesis we will look mainly at low-density parity-check codes, but apply them to systems with more difficult noise models and blocksize constraints.

1.5 Thesis outline

In Chapter 2 low-density parity-check codes will be presented in detail.

We will then break away from the symmetric memoryless channel. Many channels suffer from noise whose power varies with time so we will test the performance of low-density parity-check codes on channels with bursts in Chapter 3. In Chapter 4 channels for which the noise varies depending on other users' behaviour will be presented, and in Chapters 5 and 6 two different coding solutions for these multi-user channels using non-linear modifications to low-density parity-check codes will be shown. A further type of channel will be presented in Chapter 7, a channel that suffers from synchronization errors. A concatenated coding scheme will be shown to be effective on this channel.

Finally we examine the issue of latency; we present a method to get good performance early, without needing to wait to receive a large block. Convolutional codes will be presented in Chapter 8. In Chapter 9 these codes will then be used in conjunction with low-density parity-check codes to develop a system with good latency properties as well as good performance over a larger block size.

Appendices A to F present background information on topics in communication theory. Appendix G presents an interesting application of Hamming codes. Appendix H has a detailed complexity evaluation of low-density parity-check decoding. Appendix I describes the theory behind the Gaussian approximation in EXIT charts.

member of this sequence for $N = 100$.

Gallager showed that the codes were “good” on a symmetric memoryless channel – sequences of codes exist that achieve arbitrarily reliable communication at a rate below the channel capacity with a maximum likelihood decoder. However the codes were largely forgotten for the next thirty years.

After the invention of turbo codes [8], low-density parity-check codes were “rediscovered” in 1995 [67]. In [70] it was proved that low-density parity-check codes were “very good” – sequences of codes exist that achieve reliable communication at rates up to the channel capacity with a maximum likelihood decoder. This was proved for any symmetric stationary ergodic noise.

In a special issue of the IEEE Transactions in Information Theory [56] many important remaining questions on the practicality of the codes were answered. Firstly low-density parity-check codes were shown to be “good” with a practical decoding algorithm [87]. Secondly a way of choosing the sequence of codes to achieve capacity-approaching performance was shown [86]. Thirdly an efficient encoding algorithm was presented [88]. Simulations of codes have since shown performance within 0.04dB of the Shannon Limit at a bit error probability of 10^{-6} on the additive white Gaussian noise (AWGN) channel [18].

These codes are all based on a random-like construction. Work on more structured approaches continues [60, 72]. In most work the codes have been defined over $\text{GF}(2)$, but work on codes defined over larger fields has been promising [23]. We will mainly deal with binary random-like codes.

Low-density parity-check codes are now starting to break out of academia with current attempts to incorporate them in standards for video transmission [29] and space communications [21]. Text books covering the codes are starting to appear [37, 71, 89].

2.2 Belief Propagation Decoding

In this thesis we will use a technique for inference called belief propagation [79]. The technique allows one to take a graph describing the inter-relationships between variables in a system and obtain approximate marginalised probabilities of each variable. Belief propagation can then be used to approximately infer the state of hidden variables from observed variables. To be more specific we will look initially at belief propagation decoding of low-density parity-check codes on a memoryless channel [39]; this is the best known practical decoding algorithm [18]. In this case, the observed variables are as received from the channel and the hidden variables represent the true transmitted codeword.

For a low-density parity-check code each row of the parity-check matrix provides a relationship. We form a graph by starting with nodes corresponding to each variable (x_i) which we call the variable (or symbol) nodes. For each row we add a check node. We then add an arc (or link) for each non-zero member of \mathbf{H} connecting the variable node corresponding to the column and the check node corresponding to the row of the non-zero member. The

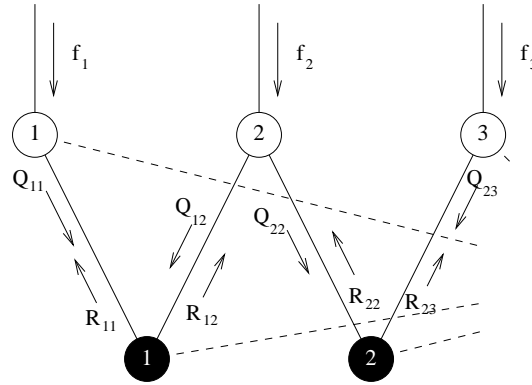


Figure 2.2: A graph fragment corresponding to Equation 2.2. Filled and open circles represent the check nodes and symbol nodes respectively. In this example the row and column weights are 3 and 2 respectively.

graph has the form that each check node provides the constraint that the sum of the variables corresponding to the connected nodes is 0. A valid codeword is formed from a setting of the variable nodes such that the constraints of all the check nodes are satisfied. A parity-check matrix fragment:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 & \cdots \\ 0 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (2.2)$$

creates the bipartite graph fragment shown in figure 2.2. This type of graph is called a factor graph; further details on factor graphs are in appendix F.

A decoder typically receives as its input a set of probabilities $f_i = \Pr(\text{bit } i = 1)$. Belief propagation then gives us an estimate of the most likely setting for each bit \hat{x}_i given the constraints of the code. Belief propagation is an iterative message-passing algorithm which involves local computation in each node and messages passed on the arcs of the graph between nodes. New messages are based on the received channel signal and existing messages. It is hoped that, as we iterate, the messages stabilise. This stable state should ideally have the best guesses of the symbol nodes forming a codeword and that this codeword would be the same as that given by a maximum likelihood decoder. At a fixed noise level the algorithm typically takes the same number of iterations regardless of block size [70]. Therefore the algorithm is computationally $\mathcal{O}(N)$ and is practical. See appendix H for a more exact complexity evaluation.

We approach the algorithm from a local viewpoint by presenting a self-consistent set of message update formulas for each type of node. In appendix F we present an alternative viewpoint.

In the following algorithm description \mathcal{G}_{ij} is the condition that a link exists between check node i and symbol node j (ie \mathcal{G}_{ij} is true if and only if $H_{ij} = 1$).

2.2.1 Check Node Computation

Each check node i sends to a connected symbol node j the message R_{ij}^a that is an approximation to the probability that check i is satisfied given symbol j is a :

$$R_{ij}^a = \Pr(\sum_l H_{il}x_l = 0 | x_j = a) \quad (2.3)$$

$$= \sum_{\mathbf{x}:x_j=a} \Pr(\sum_l H_{il}x_l = 0 | \mathbf{x}) \Pr(\mathbf{x} | x_j = a) \quad (2.4)$$

where \mathbf{x} is the transmitted vector. We observe that the first term is an indicator as to whether a setting of \mathbf{x} is in the set of the valid codewords of the check node (C_i). This indicator is used to restrict the summation. The second term can be approximately expanded in terms of the messages from connected variable nodes Q_{ij}^a (an approximation to the probability that symbol j is a according to the check nodes other than i and the channel probabilities) to give:

$$R_{ij}^a \approx \sum_{\mathbf{x} \in C_i: x_j = a} \prod_{k: k \neq j, \mathcal{G}_{ik}} Q_{ik}^{x_k} \quad (2.5)$$

Equation 2.5 can be evaluated by the forward-backward algorithm [2].

2.2.2 Symbol Node Computation

We want to evaluate both the messages to be passed from a symbol node j to a connected check node i , Q_{ij}^a , and a tentative decoding for that symbol node. Q_{ij}^a is an approximation to the probability that symbol j is a according to the channel probabilities and the check nodes other than i – the symbol S_i is used below to indicate that check node i is satisfied:

$$Q_{ij}^a = \Pr(x_j = a | \{S_1 \dots S_{i-1}, S_{i+1} \dots S_M\}, \mathbf{f}) \quad (2.6)$$

where \mathbf{f} is the “prior” probability that symbol j is a (this in general includes the channel likelihoods). We condition on only the connected check nodes:

$$Q_{ij}^a \approx \Pr(x_j = a | \{S_k : \mathcal{G}_{kj}, k \neq i\}, \mathbf{f}) \quad (2.7)$$

We can then use Bayes’s Theorem to get:

$$Q_{ij}^a \approx Z_{ij} \Pr(x_j = a | \mathbf{f}) \Pr(\{S_k : \mathcal{G}_{kj}, k \neq i\} | x_j = a) \quad (2.8)$$

where Z_{ij} is a normalizing constant. Finally we expand the latter term as a product of messages from the check nodes and select just the individual “prior” needed:

$$Q_{ij}^a \approx Z_{ij} f_j(a) \prod_{k: k \neq i, \mathcal{G}_{kj}} R_{kj}^a \quad (2.9)$$

The tentative decoding for symbol j is:

$$\hat{x}_j = \arg \max_a (\Pr(x_j = a | \{S_1 \dots S_M\}, \mathbf{f})) \quad (2.10)$$

$$\approx \arg \max_a (\Pr(x_j = a | \{S_k : \mathcal{G}_{kj}\}, \mathbf{f})) \quad (2.11)$$

$$\approx \arg \max_a \left(f_j(a) \prod_{k: \mathcal{G}_{kj}} R_{kj}^a \right) \quad (2.12)$$

2.2.3 Iteration

Q_{ij}^a is initialized to $f_i(a)$. Messages are then passed on the bipartite graph. We typically update all the check nodes, followed by all the symbol nodes and then repeat until a successful decoding results or a maximum number of steps has been taken. In simulations this maximum was typically taken to be 500 iterations.

2.3 Construction

To create a code we start by choosing parameters for a sequence of codes over increasing N . These parameters are the proportions of columns (and rows) of fixed weight – this set of proportions is called the column (or row) profile. The term “degree sequence” is also used for these parameters. If all the columns are the same weight, then the code is called regular – having a column weight $j = 3$ is common. Otherwise the code is called irregular. The exact row profile in general is less important to the performance of the code than the column profile [66, 86]. We choose a sequence and then create an instance of it at a particular N .

There are two problems to be addressed in creating good practical codes [37]:

1. We want the code to give excellent performance under maximum likelihood decoding,
2. We want the code to give good performance when used with a practical decoder.

Both of these affect the choice of sequence and how we choose a member of the sequence.

The first proofs of the performance of low-density parity-check codes were done with a maximum likelihood decoder. In [70] MacKay proved that almost-regular codes with $j \geq 3$ could obtain “very good” performance (sequences of codes exist that allow an arbitrarily small probability of error, arbitrarily close to the channel capacity).

More recently performance under a practical decoding algorithm has been analyzed. Density evolution [87] follows the messages passed in belief propagation decoding. An assumption of infinite block size underlies density evolution. Infinite block size corresponds locally to a graph with no cycles (or loops) called a tree. See figure 2.3 for an illustration of cycles. Under density evolution, a “threshold” is found; at noise levels higher than the threshold the decoding does not converge and below the threshold the decoding successfully converges. One can then optimize the row and column profiles to achieve the best threshold [86].

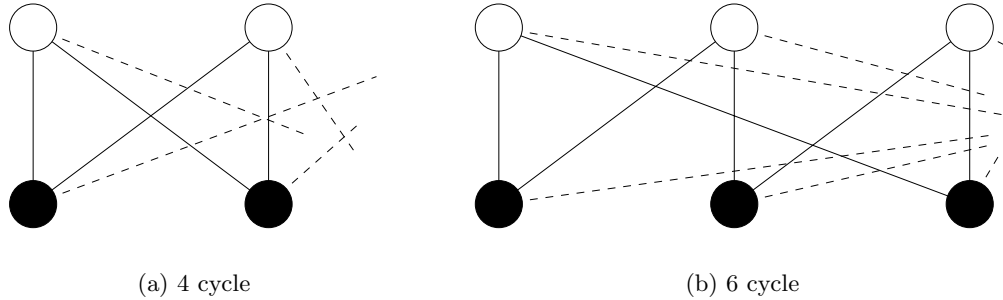


Figure 2.3: Short cycles (or loops) in a graph

Approximate techniques for density evolution exist: Gaussian approximations [19], EXIT charts (section 7.5) and Monte Carlo approximations (section 6.3.2). Applets to carry out Gaussian approximations are available on the web [16, 108].

The infinite code has no local cycles – with a finite-size code cycles change the behaviour. The best thresholds found with density evolution have both columns of weight 2 and very high weight. For a finite code, the weight-2 columns do not give excellent performance even with maximum-likelihood decoding. One gets a poor minimum distance with cycles in weight-2 columns – a cycle in weight-2 columns leads to a codeword of weight half the cycle length [113]. One therefore needs to be careful how the weight-2 columns are arranged and possibly limit the number used. Belief propagation decoding works well with a sparse parity-check matrix; the presence of very high weight columns leads to a large performance difference between infinite and finite block sizes. One needs to impose a maximum column weight to maintain good performance with a finite block size.

Once we have chosen a sequence with good asymptotic properties we then have the task of creating a particular member at the desired block size. We need to arrange the 1s in the $M \times N$ parity-check matrix whilst trying to approximate an infinite code by avoiding short cycles.

We start by placing the weight-2 columns. To avoid low-weight codewords, we create the columns so they are loop-free. Up to $\frac{M}{2}$ non-overlapping weight-2 columns can be created (where M is the number of rows in \mathbf{H}). If further columns are required, up to $\frac{M}{2}$ further columns can be created that together with the first columns form a single chain, see figure 2.4. We also ensure that if any weight-2 check nodes are present they are not connected solely to weight-2 variable nodes.

To ensure no short cycles, the remaining links are arranged using a construction similar to MacKay’s Construction 1A [70]. This involves randomly filling in columns and rows with the correct number of ones and the constraint that no two columns have an overlap of more than 1. The construction ensures the presence of no cycles of length 4.

This process creates an ensemble of codes. The best one was chosen from a random sample using the metric of the largest “girth average” [73]. This metric is the average size of the

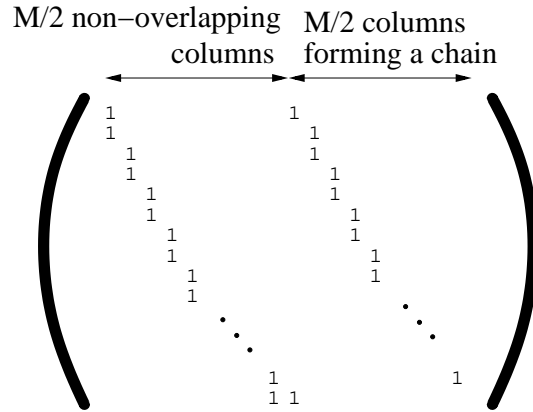


Figure 2.4: Arrangement of M weight-2 columns. The first $M/2$ are loop-free, the rest link the others as a chain. If fewer than M weight-2 columns are needed we take as many as we need, starting from the left-hand side.

smallest cycle in which each variable node is involved.

More complex sequences can be used, for instance the “multi-edge” approach [89, ch. 7]. Alternative techniques for laying out the 1s in a parity-check matrix exist, for example [54]. It is also possible to optimize a finite size code directly [106].

2.4 Basic simulations

To show the behaviour of low-density parity-check codes, simulations are presented on the AWGN channel and the BSC. The simulation procedure is described in appendix B.

Firstly we show the gain from using a soft-decision rather than a hard-decision receiver for the AWGN channel. If we threshold a AWGN channel we obtain a BSC. A low-density parity-check code can in general use any soft information that is available. The simulations, figure 2.5, are carried out with one regular $j = 3$ low-density parity-check code. It can be seen that a gain of approximately 2dB from using soft decisions is obtained.

Good column and row profiles (from [16]) with a maximum column weight of 10 were used to create irregular codes. Figure 2.6 shows the performance of two irregular codes, one with a limit of M weight-2 columns (c_2) and the other with a limit of $M/2$ weight-2 columns. The irregular codes have an error floor where the probability of block error drops slowly at a high signal-to-noise ratio. It can be seen that as we allow more weight-2 columns the performance in terms of the waterfall region improves but the error floor worsens.

2.5 Codes over $\text{GF}(q)$

One can define low-density parity-check codes over $\text{GF}(q)$ [23]. See appendix C for details of finite fields. Each symbol in \mathbf{H} and hence \mathbf{s} and \mathbf{x} (the message and the transmitted codeword respectively) is then a member of the finite field. By using an \mathbf{H} defined over $\text{GF}(2^q)$ we can

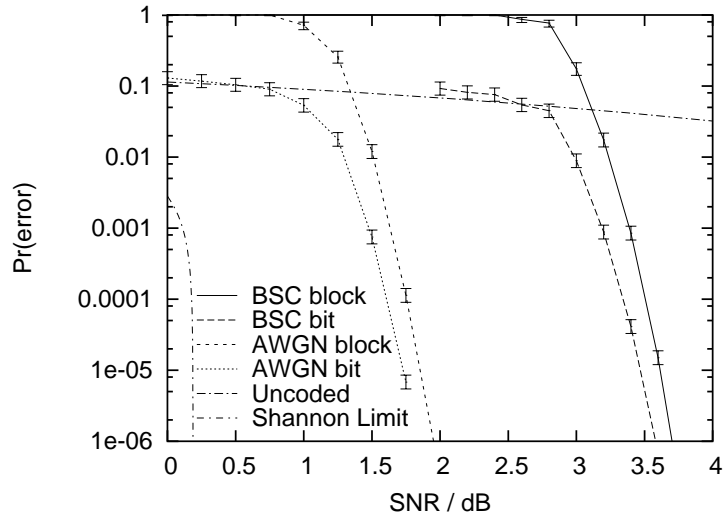


Figure 2.5: The gain from hard to soft decisions with Gaussian noise. An $N = 5000$, $R = 1/2$, $j = 3$ regular low-density parity-check code was simulated on the AWGN channel and on a BSC derived from the AWGN channel. As the signal to noise ratio (SNR) increases the probability of error decreases. With a strong error-correcting code a waterfall region is seen where the probability of error rapidly decreases for a small change in signal to noise ratio.

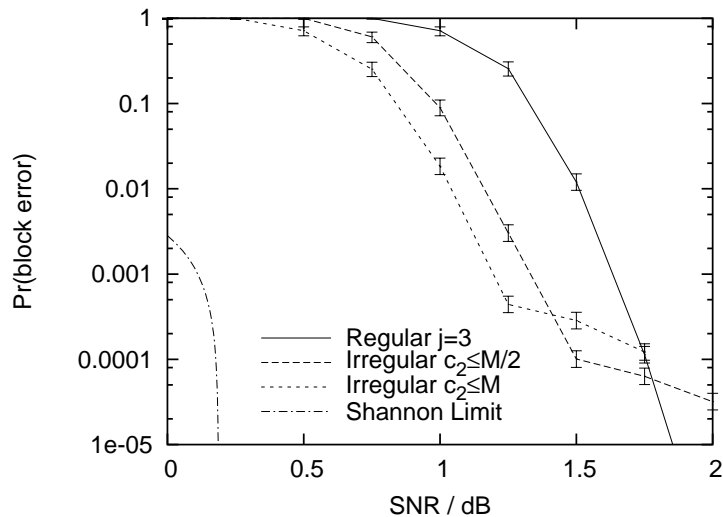


Figure 2.6: Comparison between regular and irregular codes. Three $N = 5000$, $R = 1/2$ codes are simulated: one a regular $j = 3$ code (from figure 2.5), one irregular code with $c_2 = 2394$ and one irregular code with $c_2 = 1250$. The irregular codes show a flattening out at high SNR called an “error floor”.

eliminate some cycles that would otherwise be present in the equivalent \mathbf{H} defined over $\text{GF}(2)$ [22]. This is expected to improve the performance of the decoding algorithm.

Techniques from binary code construction can be used to arrange the non-zero entries of the parity-check matrix. In this thesis the values of the non-zero elements were chosen uniformly at random. The non-zero elements can be chosen so that they maximise the entropy of the syndrome with the channel model [22].

For $\text{GF}(2)$, if any cycle can be made of weight-2 columns then a codeword of weight half the cycle length exists [113]. For larger finite fields we can look at the “weight” as the number of non-zero elements in a codeword. With randomly chosen matrix elements, a cycle in columns of weight 2 causes $q - 1$ codewords of weight half the cycle length with probability $\frac{1}{q-1}$.

The computational cost per iteration of decoding increases with field size. Fourier transform decoding [87] reduces the complexity. See appendix H for details of the complexity of the algorithms. Even for the binary case using a Fourier transform decoding algorithm can often reduce the complexity.

CHAPTER 3

MARKOV CHANNELS

3.1 Introduction

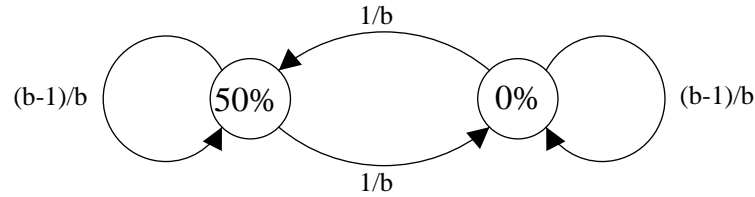
The study of low-density parity-check codes over channels with memory is a less well developed field than the application to memoryless channels. Worthen and Stark have described extending belief propagation to include channel inference [116] and experimental work on channels with block memory [115]. Eckford has since shown conditions for low-density parity-check codes to be “good” on Markov channels [31].

Markov channels are a useful bursty channel model because with a sufficient number of states they can model many noise characteristics [112, 122]. However even a two-state system shows enough complexity to evaluate how a coding system would perform [101]. Worthen [114] describes initial work with Markov channels and Wadayama [111] and Garcia-Frias [42] both have presented results on channels with hard-decision receivers (bit-flipping noise). In this chapter we will extend these results by looking at both Gaussian noise and bit-flipping noise two-state Markov channels.

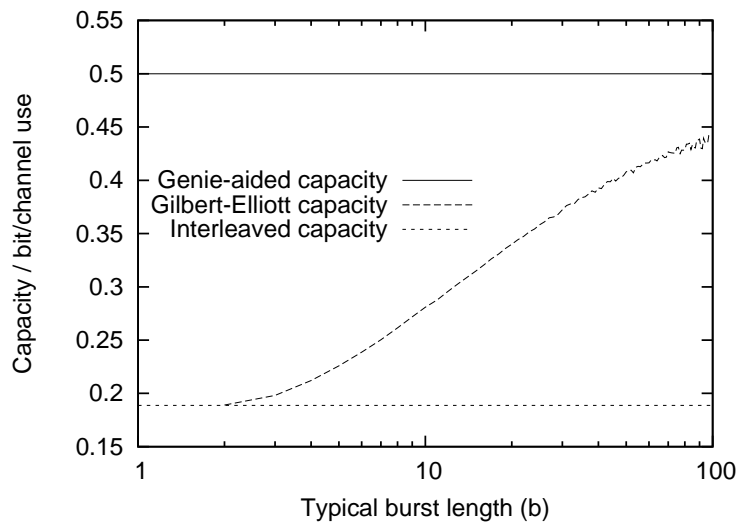
3.2 Markov channels

A Markov channel is defined by a set of states, the transition probabilities between these states and a noise level for each state. In this chapter this noise will either be bit-flipping noise with a symmetrical bit-flip probability p_f or Gaussian noise with a particular standard deviation. Each bit transmission has a state associated with it. The channel state for each transmission is based on the previous state and the state transition probabilities. In the sequence of state labels, each state has a typical run length; if a particular state has a probability p_c of changing to any different state then:

$$\begin{aligned} \text{E}(\text{run length}) &= \sum_{i=1}^{\infty} i(1-p_c)^{i-1}p_c && (3.1) \\ &= 1/p_c && (3.2) \end{aligned}$$



(a) An example channel model. Each state is represented by a circle indicating p_f and the arrows are labelled with state transition probabilities.



(b) Capacity

Figure 3.1: An example Gilbert-Elliott channel and its capacity

The capacity of the two-state bit-flipping channel (called the Gilbert-Elliott channel) has been calculated in [77]. Using a simple channel model, figure 3.1(a), the capacity and two bounds on it were calculated, figure 3.1(b):

Gilbert-Elliott capacity This is calculated by the iterative method in [77]. 1000 bins were used. It can be seen that for long typical burst lengths the accuracy of the calculation starts to reduce, causing a deviation from a smooth curve.

Genie-aided capacity This is the capacity when the state of the channel is available to the decoder. For the example illustrated, the channel then splits into two channels, an erasure channel (when $p_f = 50\%$), and a noiseless channel (when $p_f = 0\%$). The capacity is then 0.5 since each channel is used half the time.

Interleaved capacity If the Markov properties of the channel state are ignored then the channel can be considered as a BSC with noise level p_f equal to the time-average bit

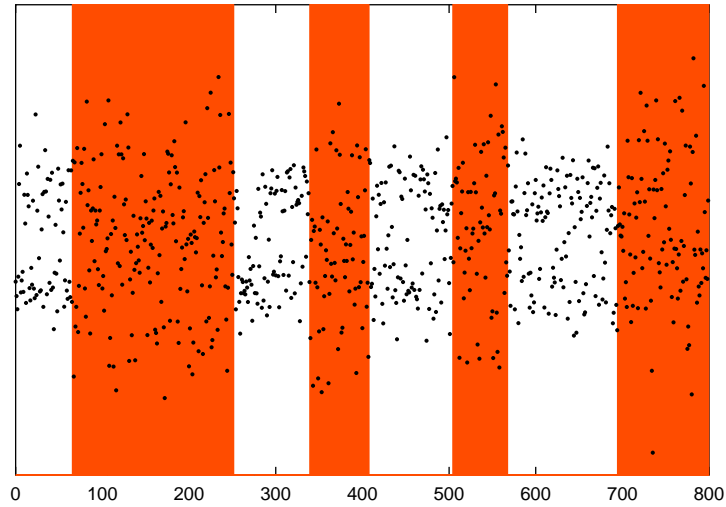


Figure 3.2: A sample set of received amplitudes from the transmission of i.i.d. bits over a Markov channel with two Gaussian noise states separated by 7dB. The average SNR is 0.57dB with a typical run length of 100 in each state. The transmissions occurring during the noisy state are shaded.

flip rate. For the example shown this is $p_f = 25\%$.

When the bursts are very small they are hard to distinguish when decoding, hence as the typical burst length decreases the Gilbert-Elliott capacity tends to the interleaved capacity. Conversely as the burst length increases one can detect more easily where the bursts are and the capacity tends towards the genie-aided capacity.

The capacity for two-state Gaussian noise channels has not been calculated to our knowledge (more complicated finite alphabet channels are addressed in [46]).

3.3 Modifications to the decoding algorithm

The most basic decoding algorithm ignores the time-dependent properties of the channel and uses the decoding algorithm from section 2.2 with the input likelihoods f_i calculated using the average noise characteristics (like the interleaved capacity). We call this the “No state estimation” algorithm.

An improvement would be to estimate the channel state history based on the string of amplitudes received (an example set of received amplitudes is shown in figure 3.2). We can use the different characteristic forms of the received signal in each of the states to estimate the state sequence using a standard HMM algorithm, the forward-backward algorithm [28].

We would like to calculate:

$$\Pr(x_i = 1 | \mathbf{r}) = \sum_{j \in \mathcal{S}} \Pr(x_i = 1 | \mathbf{r}, s_{ij}) \Pr(s_{ij} | \mathbf{r}) \quad (3.3)$$

$$= \sum_{j \in \mathcal{S}} \Pr(x_i = 1 | r_i, s_{ij}) \Pr(s_{ij} | \mathbf{r}) \quad (3.4)$$

where \mathbf{r} is the received signal, \mathcal{S} is the set of states and s_{ij} represents being in state j at time i . The first factor can be calculated using Bayes's theorem and the noise model. The second factor can be calculated as follows:

$$\Pr(s_{ij} | \mathbf{r}) \propto \Pr(s_{ij}, \mathbf{r}) \quad (3.5)$$

$$\propto \Pr(s_{ij}, r_1, \dots, r_i) \Pr(r_{i+1}, \dots, r_N | r_1, \dots, r_i, s_{ij}) \quad (3.6)$$

$$\propto \Pr(s_{ij}, r_1, \dots, r_i) \Pr(r_{i+1}, \dots, r_N | s_{ij}) \quad (3.7)$$

$$\propto \alpha_{ij} \beta_{ij} \text{ by definition} \quad (3.8)$$

α and β are called the forward and backward probabilities and can be evaluated recursively:

$$\alpha_{ij} = \sum_{k \in \mathcal{S}} \Pr(s_{ij} | s_{i-1,k}) \Pr(r_i | s_{ij}) \alpha_{i-1,k} \quad (3.9)$$

$$\beta_{ij} = \sum_{k \in \mathcal{S}} \Pr(s_{i+1,k} | s_{ij}) \Pr(r_{i+1} | s_{i+1,k}) \beta_{i+1,k} \quad (3.10)$$

using the following boundary conditions:

$$\alpha_{0,i} = \Pr(s_{0,i}) \quad (3.11)$$

$$\beta_{N,i} = 1 \quad (3.12)$$

This algorithm can be seen as adding extra nodes to the decoding graph which we call channel nodes, as illustrated in figure 3.3. We call the whole decoding algorithm the “one-off state estimation” algorithm. This algorithm does not work for bit-flipping channels as $\Pr(r_i | s_{ij})$ is independent of s_{ij} for i.i.d. transmitted bits.

A further improvement would be to expand the low-density parity-check belief propagation over these new nodes [116]. In the traditional belief propagation style we have messages g_i coming up from the symbol nodes (using the notation of section 2.2):

$$g_i^a = \Pr(x_i = a | \{S_k : \mathcal{G}_{ki}\}) \quad (3.13)$$

$$= Z_i \Pr(\{S_k : \mathcal{G}_{ki}\} | x_j = a) \quad (3.14)$$

$$\approx Z_i \prod_k R_{ki}^a \quad (3.15)$$

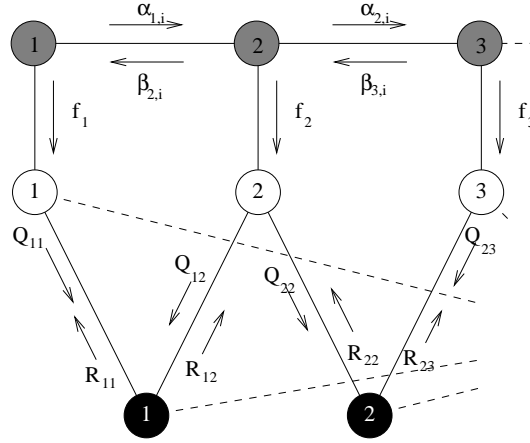


Figure 3.3: The one-off state estimation algorithm with channel nodes shown in grey

and the computation at each channel node has to be changed:

$$\Pr(x_i = 1 | \mathbf{r}, \{\forall g_k : k \neq i\}) = \sum_{j \in \mathcal{S}} \Pr(x_i = 1 | r_i, s_{ij}) \Pr(s_{ij} | \mathbf{r}, \{\forall g_k : k \neq i\}) \quad (3.16)$$

$$\propto \sum_{j \in \mathcal{S}} \Pr(x_i = 1 | r_i, s_{ij}) \alpha'_{ij} \beta'_{ij} \quad (3.17)$$

where α' and β' are modified versions of α and β to include \mathbf{g} :

$$\alpha'_{ij} = \sum_{k \in \mathcal{S}} \Pr(s_{ij} | s_{i-1,k}) \Pr(r_i | s_{ij}, g_i) \alpha_{i-1,k} \quad (3.18)$$

$$\beta'_{ij} = \sum_{k \in \mathcal{S}} \Pr(s_{i+1,k} | s_{ij}) \Pr(r_{i+1} | s_{i+1,k}, g_{i+1}) \beta_{i+1,k} \quad (3.19)$$

We call this decoding algorithm the “iterative state estimation” algorithm and it is illustrated in figure 3.4. This algorithm can now be used over bit-flipping channels unlike the one-off state estimation algorithm. The computation order used in the following experiments was that the channel nodes were first updated as a chain. Next the symbol nodes were initialized in the traditional belief propagation manner. Then iteration over the graph was started. The nodes were repeatedly updated in the following order: the check nodes, the symbol nodes, the channel nodes and then the symbol nodes.

A further improvement would be if one had additional information of the true channel state for each bit received – the traditional low-density parity-check decoding algorithm can then be used with input probabilities derived using the state information. This is the “genie-aided” algorithm.

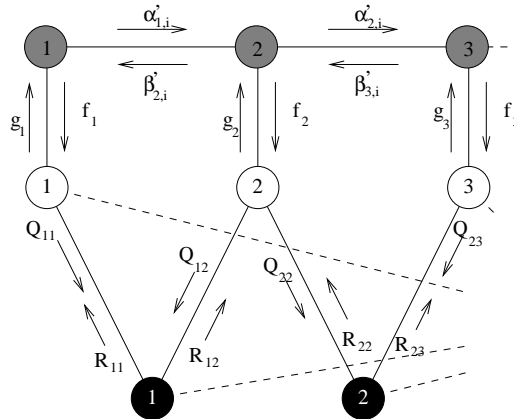


Figure 3.4: The iterative state estimation decoding algorithm

3.4 Performance on the Markov channel

An $R = 1/2$, $N = 8000$ regular low-density parity-check code with column weight 3 from [68] was used to test the above algorithms.

3.4.1 Bursts of total data loss

A channel with bit flipping noise was first studied. Two states were used: one with $p_f = 50\%$ and one with $p_f = 0\%$; the former had a typical burst length of 10 and the latter had a variable typical burst length t , figure 3.5(a). Varying the typical run length in the quiet state varies the average noise on the channel. The results of experimental simulations with three different decoding algorithms and the corresponding Shannon limits are shown in figure 3.5(b).

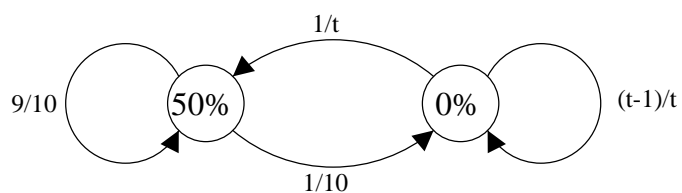
3.4.2 Gaussian noise

A channel with Gaussian noise was also studied. Two channel states were used with a separation in noise level of 7dB (to match [115]). The typical run-length in the two states were both set to 3, 10 or 30. The average SNR was varied and the empirical block error rate was measured using all four algorithms described above. The results are shown in figure 3.6.

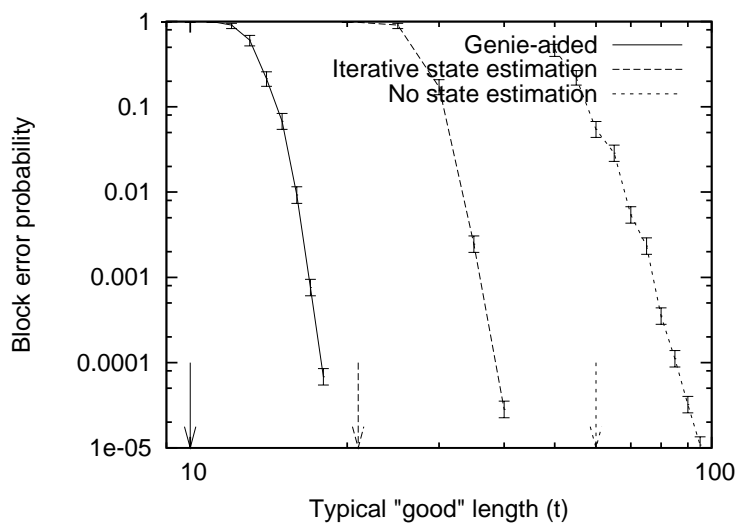
3.4.3 Bit-flipping noise

The same experiment was then repeated using bit-flipping noise. To make the experiment more easily comparable to the previous experiment this noise was assumed to come from the same channel as before but with a hard-decision receiver. One would expect a 2dB drop from taking hard decisions [81]. Empirical results are shown in figure 3.7.

The capacity of this type of channel is shown in figure 3.8.



(a) The channel studied



(b) Performance of three decoding algorithms with corresponding Shannon Limits shown with arrows

Figure 3.5: Bit-flipping channel with a variable length in the "good" state

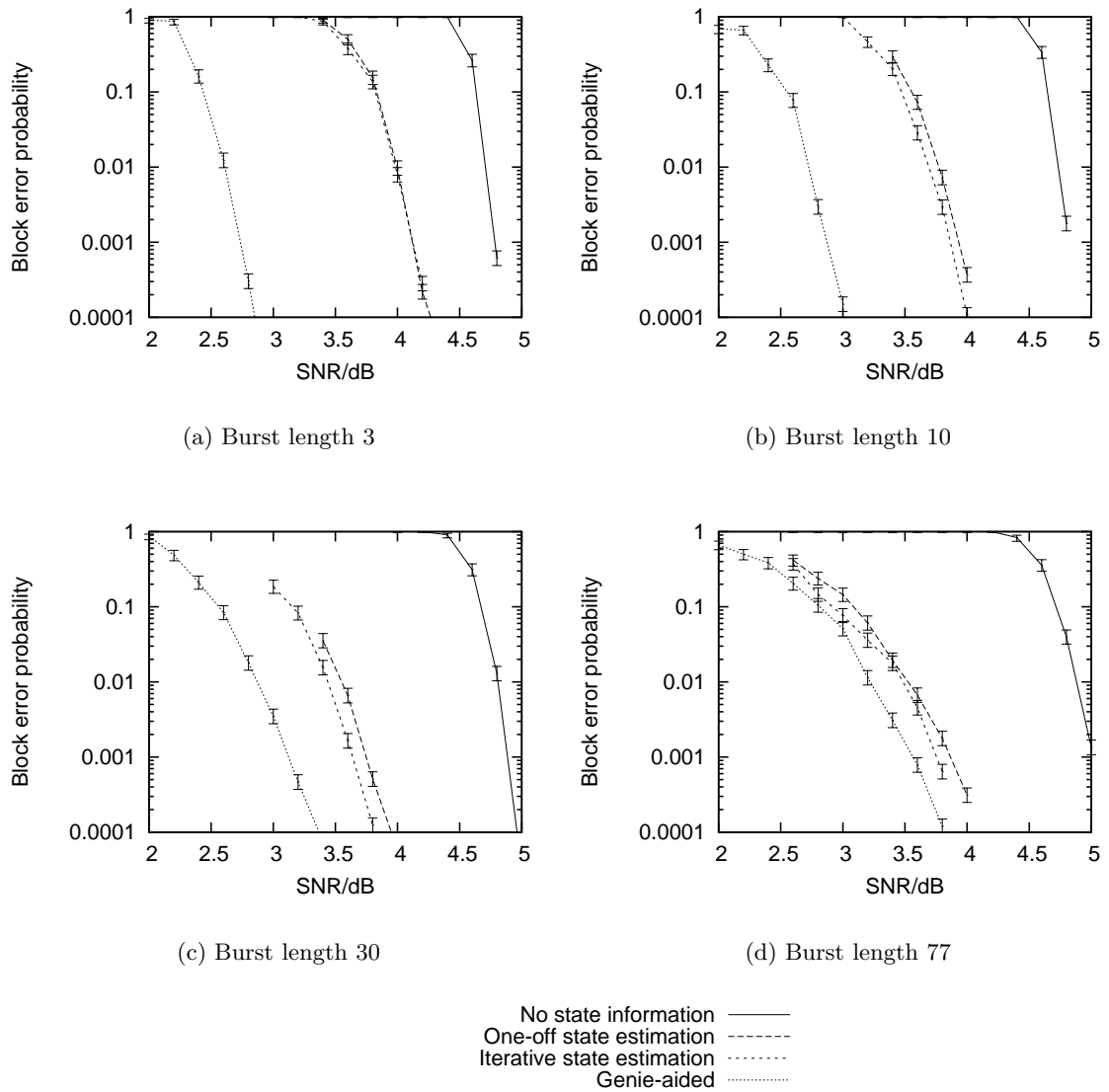
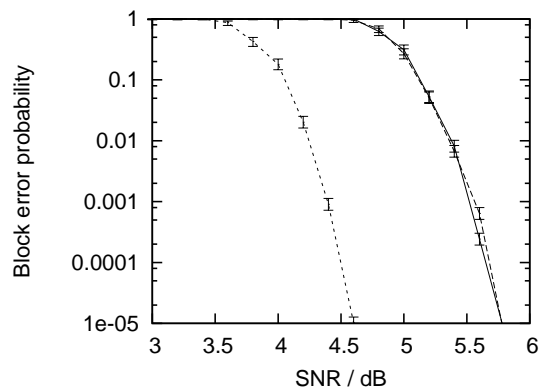
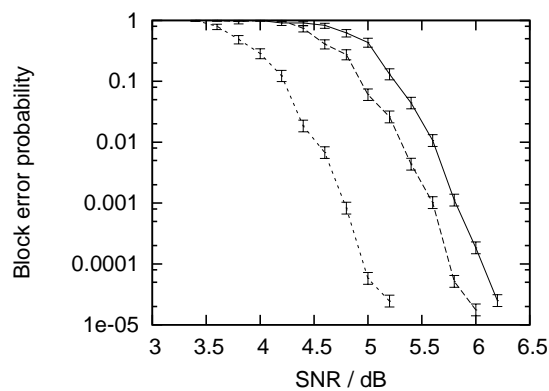


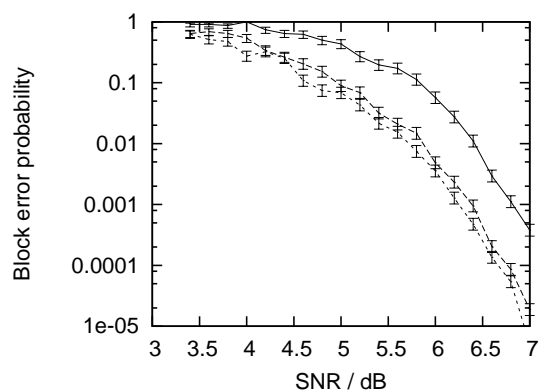
Figure 3.6: Gaussian noise with 7dB separation between two states with equal typical run-lengths



(a) Burst length 3



(b) Burst length 30



(c) Burst length 300

No state estimation ———
Iterative state estimation - - - - -
Genie-aided ·····

Figure 3.7: Bit-flipping noise with 7dB separation between two states with equal typical run-lengths

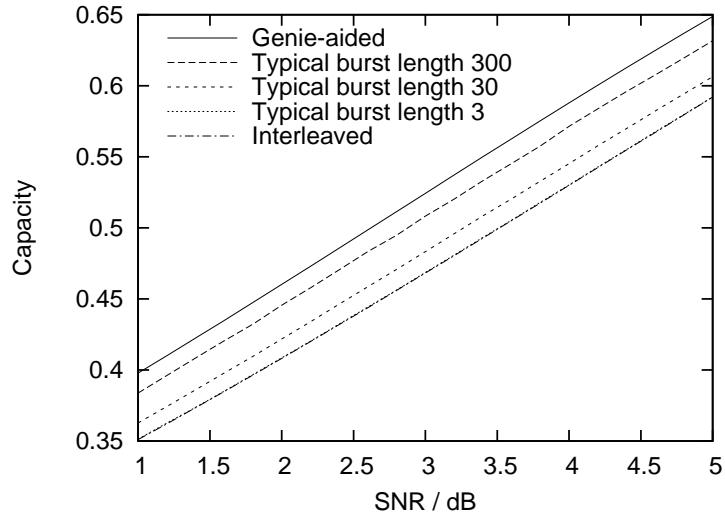


Figure 3.8: Capacity of two-state bit-flipping Markov channels with 7dB noise separation. The lines representing the interleaved and typical-burst-length-3 capacities fall on top of each other at this scale.

3.4.4 Typicality of the channel

It can be seen that as the burst length increases the performance curves become flatter. It is thought that this is due to the variation of the total noise energy of the channel, over the studied block size of 8000, increasing as the typical burst length increases. A low-density parity-check code can handle noise up to a fixed power, so an increased variability of total noise power leads to a blurring of the waterfall region. For a burst length of 30 an estimate of the probability density function of the number of bits transmitted in the noisy state is shown in figure 3.9(a). The standard deviation of this distribution against typical burst length is shown in figure 3.9(b).

3.4.5 Iterations

For the latter bit-flipping experiment (section 3.4.3) the evolution of $\Pr(s_{ij}|\mathbf{r}, \{g_k : k \neq i\})$ was studied. For the first iteration the estimate of the probability of being in the noisy state starts with a 50:50 distribution. It is expected to tend towards the true state distribution as the iterations progress. An example of the evolution is shown in figure 3.10. Correct decoding was achieved in three iterations for the example shown. The state estimate is close to the true value after only a few iterations. Large errors in iteration 2 can be seen around bit numbers 1200 and 1700 but these are corrected by iteration 3.

The number of iterations before the iterative state estimation algorithm reached a decoding was also recorded for the Gaussian noise experiment (section 3.4.2) and is shown in figure 3.11. It can be seen that fewer iterations are needed for a successful decoding as the typical burst length increases. This suggests that decoding becomes easier as the typical burst length

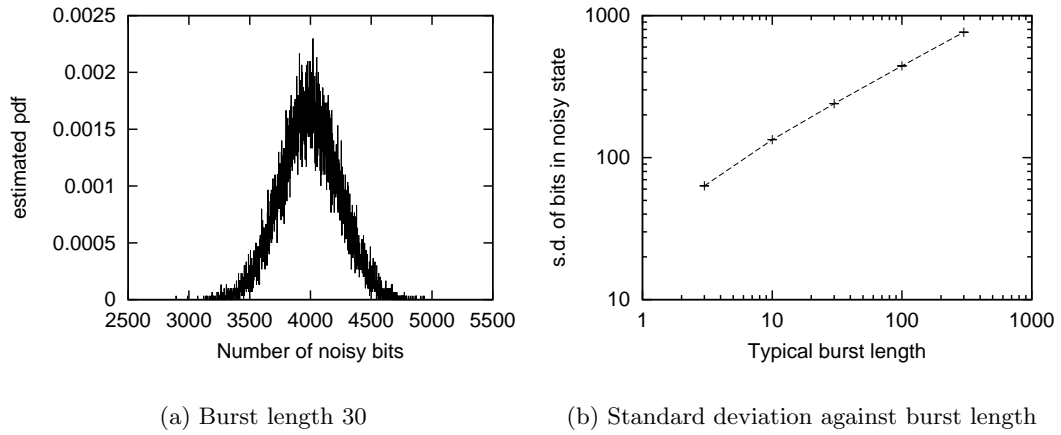


Figure 3.9: Distribution of number of transmissions in the noisy state for an $N = 8000$ block over a 2-state Markov channel

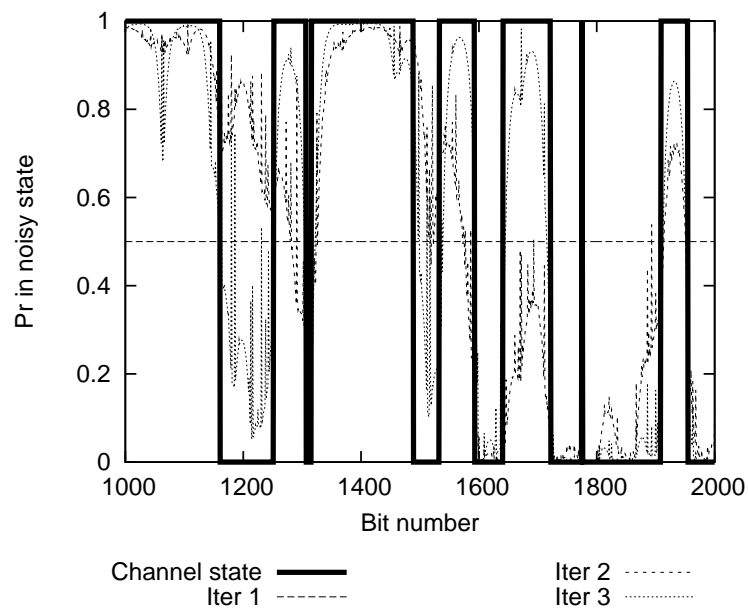


Figure 3.10: Evolution of the inference of state ($\Pr(s_{ij} | \mathbf{r}, \{g_k : k \neq i\})$) for a two-state bit-flipping channel with average noise of 4.8dB, separation of 7dB and typical burst length of 100.

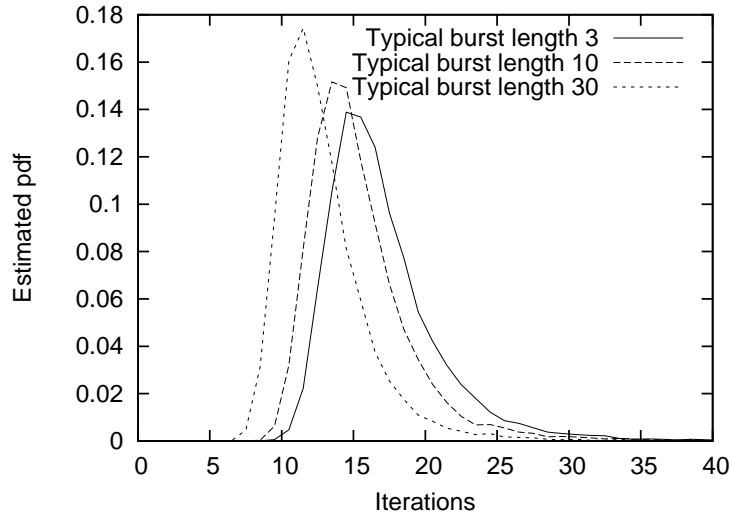


Figure 3.11: Histogram of the probability density function of the number of iterations until the iterative state estimation algorithm terminates. Gaussian noise was used with an SNR difference of 7dB and the average SNR was chosen so the block error probability was 10^{-3} .

increases. The most difficult region for the channel nodes to infer the true state is near a state transition. As the typical burst length increases these regions become less significant and hence the decoding process can be expected to be easier.

3.5 Discussion

For bit-flipping noise, the performance with the iterative state estimation algorithm should be compared with the Gilbert-Elliott capacity as this represents the best possible performance over the bit-flipping channel with no channel state information. The first experiment (section 3.4.1) shows that the iterative state estimation algorithm, despite not reaching this capacity with the chosen code, is producing the expected gain over the no state information case. The further experiments on the bit-flipping channel (section 3.4.3) show that the iterative state estimation algorithm is producing similar gains. We can further see the codes are at least 2dB from capacity at a block error probability of 10^{-3} .

The performance of the algorithms over the Gaussian noise channel is hard to assess as the channel capacity has not been calculated to the author's knowledge. It can be seen that behaviour similar to the bit-flipping case is observed: the code performance increases as the typical burst length increases. However this increase happens faster than for the bit-flipping case, probably due to the extra information in the received amplitudes. It is interesting to compare the one-off state estimation algorithm and the iterative state estimation algorithm. For very short bursts iterative state estimation can not offer any significant advantage over one-off state estimation. When the typical burst length is 30 the gain is only 0.1dB.

Earlier work on a block bursty channel [115] with a hop length of 10 bits (one spends a

fixed “hop length” in a particular state) and a code of the same rate suggested that larger gains might be found between one-off state estimation and iterative state estimation. We can compare the entropy per symbol (H) of the state sequences of the two channels:

$$H(\text{block bursty channel}) = \frac{1}{\text{hop length}} \quad (3.20)$$

$$H(\text{Markov channel}) = H_2\left(\frac{1}{\text{burst length}}\right) \quad (3.21)$$

where $H_2(\cdot)$ is the binary entropy (as defined in appendix A). For a hop length of 10 one needs a typical burst length of 77 for these two expressions to be equal. However the experiments with a burst length of 77, figure 3.6(d), suggest that a gain of about 0.1dB (rather than 0.2dB for the block bursty channel as reported in [115]) is being seen.

Many iteration schemes could be used on the belief propagation graph. It would be worth investigating other schemes and comparing their expected computational complexity. Quicker decodings may be achieved in some cases by, for example, only evaluating the channel nodes after a few iterations over just the symbol and check nodes. An analysis of the balance between expected computational time and performance is possible [13].

No attempt was made to optimize the code. It is hoped that a larger block-size irregular code could perform better as seen with the AWGN channel [86]. Confirmation of this would be useful further work. Further criteria could also be used, for example the removal of “near” codewords (codewords that lead to low weight syndromes when a set of bits transmitted at close times are corrupted) could be advantageous as a burst of noise is likely to corrupt several nearby bits.

3.6 Conclusion

We have shown that we can include channel estimation in a low-density parity-check code’s decoder to get a gain when the code is used over a channel with memory.

For the bit-flipping noise Markov channel we get a gain of the same order as the difference between the interleaved capacity and the Gilbert-Elliott capacity. With Gaussian noise the most significant gain comes from estimating the channel state once. As the typical burst length increases, iterative state estimation produces an increasing gain over one-off state estimation.

CHAPTER 4

MULTI-USER CHANNELS

4.1 Introduction

Often multiple signals are transmitted in the same medium at the same time. For instance, to increase the amount of information carried by an optical fibre, signals are transmitted in several different colours (or bands) at the same time. This is called wavelength division multiplexing. Light in one band can cause noise or “cross-talk” in other bands. As one can alter the characteristics of the noise on one band by regulating transmissions on other bands, coding for the system is an interesting problem.

There are four ways one could approach such a channel which we shall classify in terms of whether one can do joint or independent encoding or decoding:

Joint encoding and decoding All the bands are treated together as a non-binary channel.

An optimal non-binary input distribution could be found and a coding scheme used similar to the one which will be presented in chapter 6.

Joint decoding If one is able to process all the received signals together, a better estimate of the characteristics of the noise can be obtained. This field is called multiuser detection [109].

Joint encoding Beam-forming [61] can be used on some multi-user channels to focus information towards particular receivers.

Independent encoding and decoding Each band is treated independently with conventional error-correcting encoders and decoders.

We will study this last scenario. This situation is common as often neither joint encoding nor joint decoding are possible, either for computational reasons or due to transceivers not being co-located.

In this chapter we introduce two simple channel models in which noise originates from cross-talk. Then in the following two chapters coding schemes will be developed which work well on these channel models.

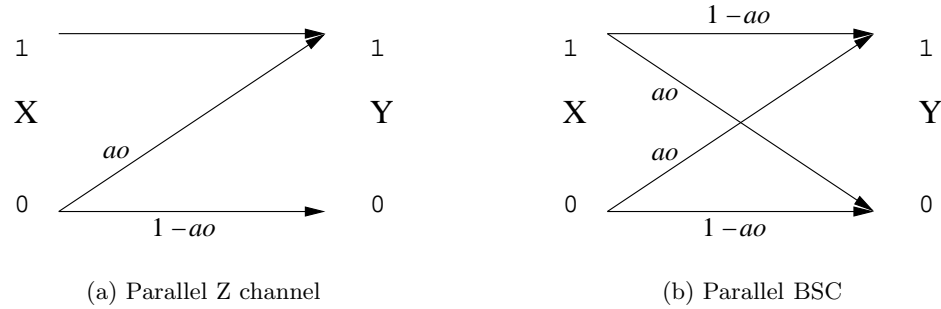


Figure 4.1: Subchannels for the two channel models studied. o is the number of ones being transmitted on the other channels at the current time and a is the noise level parameter.

4.2 Channel Models

In this thesis we will look at two simple multi-user channel models:

Parallel Z Channel Each user's channel is a Z-channel where the probability of corrupting 0s is proportional to the number of 1s being transmitted on the other channels, figure 4.1(a). This is a model of crosstalk in optical channels.

Parallel Binary Symmetric Channel Each user's channel model is a BSC where the flip probability is proportional to the number of 1s being transmitted on the other channels, figure 4.1(b). This example is similar to noise on some ultra-wideband channels.

For both channels, a is the constant of proportionality, o is the number of ones being transmitted on the other channels and u is the number of users.

We assume the constraint that we treat each subchannel independently and identically. This constraint leads to each subchannel being a channel with flip probability $= ap_1 \cdot (u - 1)$, where p_1 is the probability of transmitting 1. To achieve the maximum rate of communication, the symbols should not be used with equal probability. For the parallel BSC, $p_1 \leq \frac{1}{2}$ for optimal communication. For the parallel Z channel this same condition applies for high noise levels. For a single user binary channel the maximum gain over a traditional code by using an asymmetric input distribution is 6% [96], but for a multi-user channel larger gains can be achieved.

We will look at three approaches to improve communication on this channel beyond using a plain low-density parity-check code that gives $p_1 = 1/2$. These are:

Time-sharing Each user only transmits for some fraction of the time, thereby lowering the noise experienced by each user.

Sparse-Dense codes In chapter 5 a scheme whereby p_1 is lowered for a fraction of the bits (with the rest of the bits having $p_1 = 1/2$) is presented. This again lowers the average

noise level, but one might expect superior performance over the timesharing approach as the channel is not being left idle.

Sparse LDPC Codes In chapter 6 a way of lowering p_1 for all the bits will be shown, thus achieving the optimum input ensemble.

4.3 Capacity

We calculate the “capacity” for each user’s subchannel under three different transmission regimes:

Dense transmissions The capacity achievable if each bit transmitted is i.i.d. with $p_1 = 0.5$. This capacity can be approached using linear error-correcting codes.

Sparse-Dense transmission This is the capacity achievable if each data bit is transmitted in a sparse form (with density p_{d1}) and each check digit is dense, corresponding to Sparse-Dense codes. R_d is used as the ratio of number of source sparse bits to block size.

Sparse transmission This is the capacity achievable if each bit is i.i.d. with $p_1 \neq 0.5$.

When calculating the capacities for the example channels we take the average codeword weight and then use this weight to calculate an average noise level. This technique assumes a large number of users each with a random independent order of bit transmission and is a consequence of the noise being linear.

To calculate the dense data capacity we evaluate $I(X;Y)|_{p_1=0.5}$ (where $I(X;Y)$ is the mutual information between the input ensemble X and the output ensemble Y as defined in appendix A). For the sparse calculation we evaluate $\max_{p_1} I(X;Y)$, remembering that the noise level depends on X .

Calculation of the sparse-dense capacity is more complicated as we have two expressions for the information reliably transmitted per channel symbol $f(p_{d1}, R_d)$, one from the channel coding theorem and one from the source coding theorem (the constraint that all the user data is contained within the sparse bits):

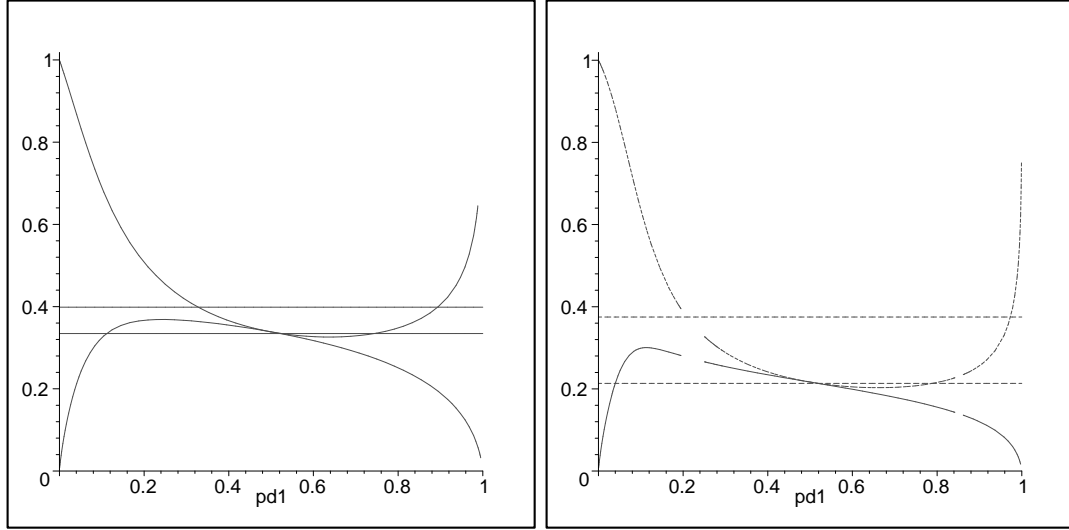
$$f(p_{d1}, R_d) = I(X;Y) \quad (4.1)$$

$$f(p_{d1}, R_d) = R_d H_2(p_{d1}) \quad (4.2)$$

where $H_2(\cdot)$ is the binary entropy (as defined in Appendix A). We obtain $f(p_{d1})$ by elimination of R_d in the above simultaneous equations with a symbolic maths package. The capacity is then:

$$C = \max_{p_{d1}} f(p_{d1}) \quad (4.3)$$

For the parallel Z channel with 128 users and $a = 0.0074$, $f(p_{d1})$ is shown in figure 4.2(a). Similarly figure 4.2(b) shows $f(p_{d1})$ for the parallel BSC with $a = 0.0037$.



(a) Parallel Z channel with 128 users and $a = 0.0074$

(b) Parallel BSC with 128 users and $a = 0.0037$

Figure 4.2: Plots $f(p_{d1})$ for a Sparse-Dense code. The lower curve shows $f(p_{d1})$ (the sparse-dense capacity is at the maximum) and the upper curve shows the corresponding ratio of number of source sparse bits to block size, R_d . The lower line indicates the dense transmission capacity and the upper line the sparse transmission capacity.

4.4 Time-sharing

Time-sharing can be described by a single parameter: the fraction of time a transmitter is active, t_f . We would like to work out the optimum time-sharing proportion t_f^* for a particular code rate R . Assuming a linear capacity-approaching code we can write

$$R = t_f I(X; Y) = t_f (1 - H_2(a(u-1)t_f/2)) \quad (4.4)$$

where the expansion of $I(X; Y)$ has been done for the parallel BSC. We can numerically solve for $a = a(t_f)$ which allows a solution for

$$t_f^* = \arg \max_{t_f} a(t_f) \quad (4.5)$$

Time-sharing at this optimum proportion will be used in the next two chapters as a benchmark.

CHAPTER 5

SPARSE-DENSE CODES FOR MULTI-USER CHANNELS

5.1 Introduction

Sparse data consisting of independent identically distributed (i.i.d.) bits with $p_1 \equiv \Pr(\text{digit} = 1) \neq 1/2$ is frequently used in information theory. Applications in communication theory include MN codes [70] and watermark codes [24].

This chapter first addresses the problem of mapping dense data to sparse data blocks of fixed size. This is often solved for small block sizes by using a look-up table [24]. An algorithm is presented in [70] for larger blocks and is studied further below. A new, more efficient, algorithm is then introduced.

We then present a new application for sparse data blocks in conjunction with linear error-correcting codes for multi-user channels which we call Sparse-Dense codes.

5.2 Sparsifiers

5.2.1 A sparsifier based on arithmetic coding

Arithmetic coding [76] is a source coding technique that represents a sequence of outcomes by an interval in the range $[0,1)$. To encode the next outcome this interval is uniquely divided into subintervals for each of the possible outcomes of the experiment. The size of each subinterval is proportional to the probability of the outcome. The previous interval is replaced with the subinterval to which the outcome corresponds. This type of interval division can be seen in left-hand side of figure 5.1. We usually then compare the final interval with intervals produced in the same manner with an i.i.d. binary equiprobable ensemble. For more information on compression with arithmetic coding, [71] is recommended.

An algorithm for the generation of sparse blocks based on arithmetic coding is presented in [70]. The algorithm uses standard arithmetic coding but reverses the usual role of compression

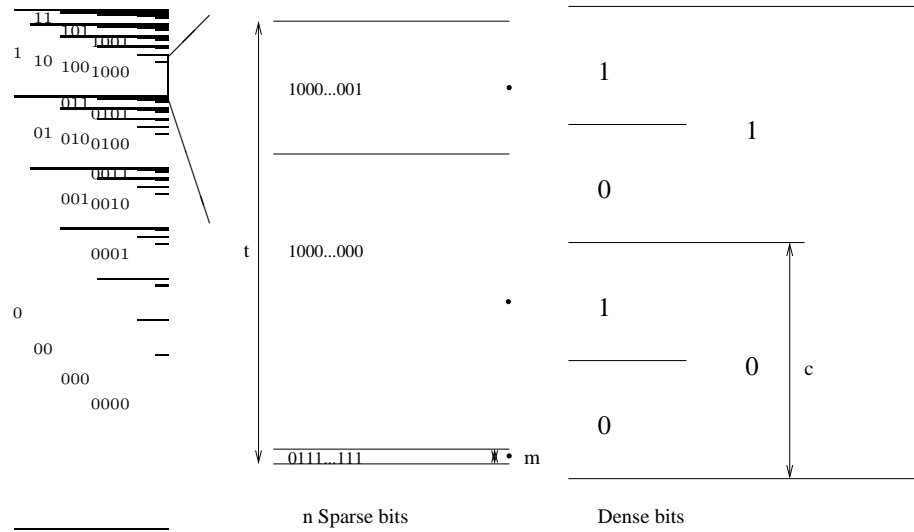


Figure 5.1: An example state of the arithmetic coding based sparsifier. The sparse output bits on the left hand side are read from left to right (the current area is shown as a magnification of the initial state). The dense input bits are on the right hand side and are read from right to left. The centre of each sparse interval is shown with a dot.

and decompression. The probabilistic model used is:

$$\Pr(s_i = 1 | s_1, \dots, s_{i-1}) = \Pr(s_i = 1) = p_1 \quad (5.1)$$

where s_i is the i th bit of the sparse data.

The algorithm is presented without details of initialization or termination. For initialization one sets the standard arithmetic coding start conditions. The algorithm can be used without termination if there is an infinite stream of data being transmitted. If a single sparse bit is corrupted, on decoding, all following data bits are corrupted in a catastrophic fashion.

This property is not desirable in a communication theory context. We are often transmitting blocks; if a single block is corrupted we do not want the following blocks to be corrupted in addition. To avoid it we need each block to be created independently. Different weight blocks have different probabilities of occurring, hence for fixed size sparse blocks the input sequence length is variable (with longer sequences for less probable blocks). We therefore need to ensure that the termination of the arithmetic coder has the property of forming a complete prefix code in the dense data. A complete prefix code has the property that every possible infinite dense string can be divided into codewords without punctuation [71]. We check for the existence of a valid sparse block as if the next input digit is 0 and 1 (to maintain the prefix code tree structure) before reading in each binary digit. If there is not such a block for both 0 and 1 we stop reading for that block and transmit the sequence corresponding to a valid interval.

Two different termination algorithms were tried. The first, Algorithm I, ensured the

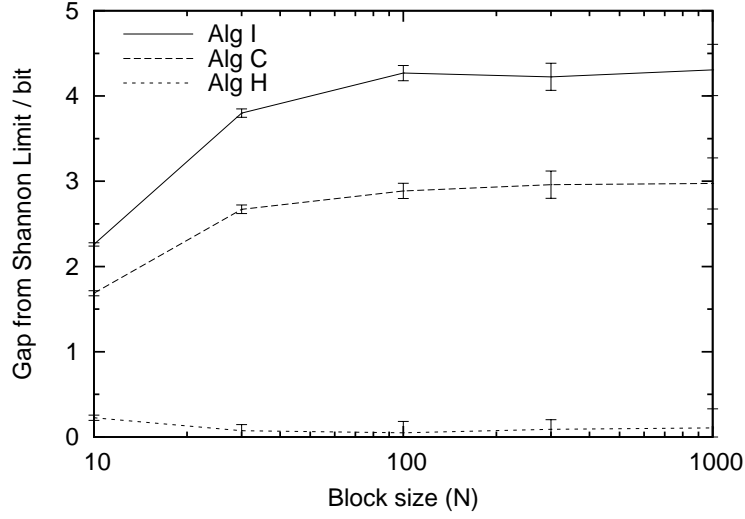


Figure 5.2: The gap from the Shannon Limit for the sparsifiers discussed in this chapter. The difference between the number of bits encoded and $NH_2(\hat{p}_1)$ is shown. p_1 was requested to be 0.1.

presence of the entirety of a sparse interval within the dense interval. For example if the arithmetic coder is in the situation illustrated in figure 5.1, only one more dense bit would be able to be encoded. If the next dense digit is either 0 or 1, neither of their two subintervals both contain a sparse interval. The second, Algorithm C, ensured the presence of the centre of an interval within the dense interval. For the situation in figure 5.1 this would allow the sequences 1, 00 and 01 to be encoded. The encoding of more information by Algorithm C carries across in general as simulations in figure 5.2 show.

Arithmetic coding algorithms are efficient for compressing data as, with a suitable probabilistic model, the overhead is at worst 2 bits over the entire compression. The same efficiency does not carry over to sparsifiers. The problem stems from the termination procedure not coping well with large intervals near small intervals. For the case of figure 5.1 the bottom magnified sparse interval is much smaller than the other two. Under Algorithm I this small interval is transmitted if the dense digit is 0. For this example using the notation in figure 5.1 the information loss (δI) scales with the block size (N):

$$t = p_1^2 p_0^{N-2} + p_1 p_0^{N-1} + p_0 p_1^{N-1} \quad (5.2)$$

$$c = t/2 \text{ for worst case} \quad (5.3)$$

$$m = p_0 p_1^{N-1} \quad (5.4)$$

$$\delta I = \log_2(c) - \log_2(m) \quad (5.5)$$

$$= \log_2(3) - 1 + \frac{4(2N-5)}{3 \ln(2)} (1/2 - p_1) + \mathcal{O}((1/2 - p_1)^2) \quad (5.6)$$

For Algorithm C this information loss happens less often (it needs the next two dense digits

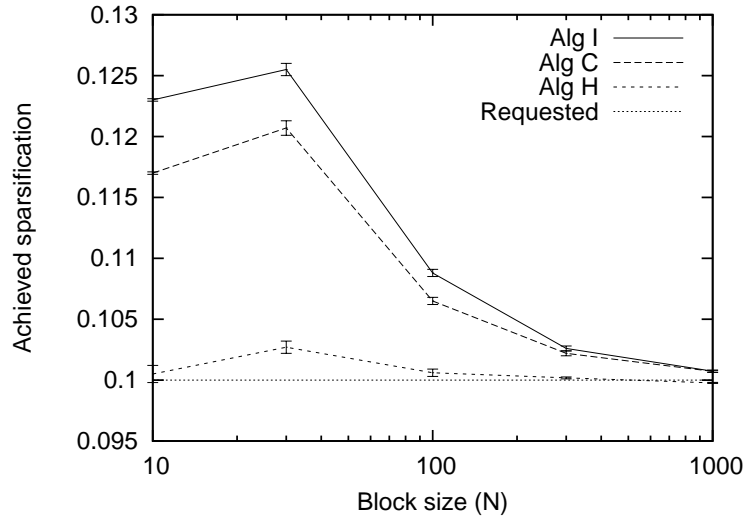


Figure 5.3: The sparsification achieved as a function of block size for the algorithms described in this chapter. A sparsification of $p_1 = 0.1$ was requested.

to be 00) and, in this case, a 1 bit smaller information loss than Algorithm I would be seen.

One can also evaluate the sparsifier's performance in terms of the sparsification achieved, \hat{p}_1 , as a function of blocksize. Figure 5.3 shows \hat{p}_1 for the two sparsification algorithms described so far.

5.2.2 The new sparsifier

A new sparsifier is now developed with the aim of better asymptotic performance.

A set of sparse data blocks, with a given p_1 , is indistinguishable from a set of blocks created by a fixed weight block generator, if the fixed weight (w) is sampled from the correct binomial distribution:

$$\Pr(\text{block}|p_1) = \Pr(\text{block}|w, p_1) \Pr(w|p_1) \quad (5.7)$$

$$= \Pr(\text{block}|w) \Pr(w|p_1) \quad (5.8)$$

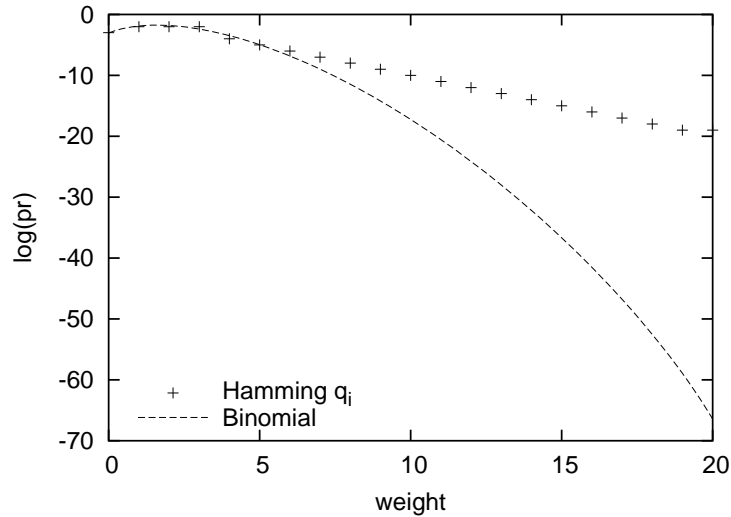
$$= \Pr(\text{block}|w) p_1^w (1 - p_1)^{N-w} \quad (5.9)$$

We can therefore construct a sparsifier which operates in two steps: choosing a weight and generating a block of that weight.

If we take a prefix code decoder and feed it a stream of i.i.d. bits, it outputs members of the code's alphabet with probabilities $q_i = 2^{-l_i}$ (where l_i is the codeword length of alphabet member i). A Huffman code [55] was constructed with an alphabet consisting of a subset of weights, each assigned a probability according to the binomial distribution. This code's decoder was used to choose the weight of a block.

If we choose the subset to be all the weights, we get the code shown in the middle column

Weight	$\log_2 1/\text{Pr}$	Full Code	l_i	Truncated Code	l_i
0	3.04	010	3	010	3
1	1.89	10	2	10	2
2	1.81	11	2	11	2
3	2.4	00	2	00	2
4	3.48	0111	4	0111	4
5	4.97	01101	5	0110	4
6	6.82	011001	6		
\vdots	\vdots	\vdots	\vdots		
18	52.5	011000000000000001	18		
19	58.9	0110000000000000001	19		
20	66.4	0110000000000000000	19		

Table 5.1: Two Huffman codes generated for $N = 20$, $p_1 = 0.1$ Figure 5.4: The binomial distribution and the implicit probabilities for the Hamming code formed from the set of all weights with $N = 20$, $p_1 = 0.1$

of Table 5.1. Figure 5.4 shows a problem with this subset – the very low probability sequences occur more often than expected. If we are asking for sparse data we probably do not want high weight sequences to occur more often than expected. So a subset was chosen that was as large as possible, centred around Np_1 , such that the q_i were within a factor of two of the binomial distribution. A code chosen in this way is shown in the last column of Table 5.1. Such a code stops extreme events happening which can be an advantage in a cooperative system.

To generate a block of a particular weight, once a weight has been chosen, a constant weight code (also known as an M -out-of- N code) was used. These codes satisfy the constraint that for each codeword exactly M bits are 1 and the other $N - M$ bits are 0. The codes are widely used as they can detect all unidirectional errors, such as from a Z-channel [5]. Applications

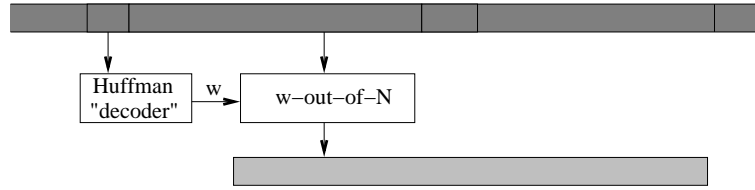


Figure 5.5: Algorithm H

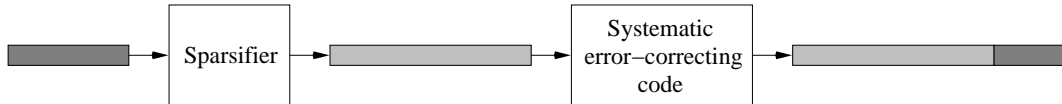


Figure 5.6: A Sparse-Dense Code. Sparse data is indicated by light grey and dense data by dark grey.

include VLSI circuits and memories and optical channels [119]. Efficient creation of both small and large blocks has been investigated [119, 82].

In [82] an algorithm based on a technique similar to arithmetic coding is presented. The algorithm maps a fixed length source block to a M -out-of- N codeword with at worst a one bit inefficiency. We extend the algorithm by creating a prefix code in the dense data to reduce the inefficiency. The idea is to use a similar type of arithmetic coding algorithm to Algorithm C, but instead use it to generate constant weight codewords. The probabilistic model now becomes dependent on the history:

$$\Pr(s_i = 1 | s_1, \dots, s_{i-1}) = \frac{M - \text{weight}(s_1, \dots, s_{i-1})}{N - i + 1} \quad (5.10)$$

This probabilistic model makes all blocks of weight M equiprobable. Unlike many models with large history, this model is easily tractable as the history can be summarized by the weight of the proceeding bits.

Algorithm H is the combination of the Huffman decoder and the constant weight encoder. From the input stream we read off a Huffman codeword to set the weight and then read off one codeword of the corresponding fixed weight sparsifier as illustrated in figure 5.5. The performance of this algorithm is illustrated in figures 5.2 and 5.3. It can be seen that on both efficiency and accuracy of sparsification Algorithm H outperforms Algorithms I and C.

5.3 Sparse-Dense Codes

The method of creation of sparse data blocks above does not include the facility for any error correction. But by feeding sparse bits to a systematic error-correcting code encoder we obtain an error-correcting code word with lower weight than usual. The final block has sparse systematic bits and dense parity bits (for a linear code), figure 5.6.

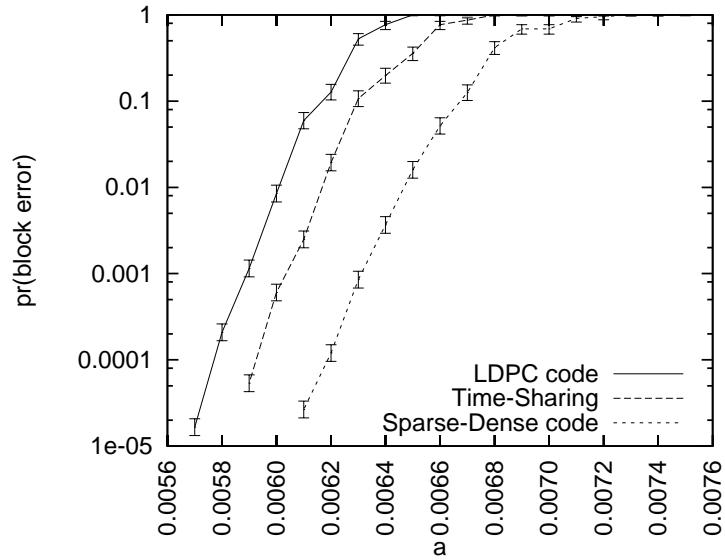


Figure 5.7: The performance of $R = 0.3$, $N = 10000$ codes on a 128-user parallel Z channel

The performance of Sparse-Dense codes on the two channel models presented in chapter 4 was studied by simulation.

5.3.1 Parallel Z Channel

Codes with $R = 0.3$ and $N = 10000$ were constructed. Firstly a Sparse-Dense code was created using a $p_1 = 0.25$ sparsifier combined with an $N = 10000$, $R_{\text{LDPC}} = 0.37$, $j = 3$ low-density parity-check code so that the overall code rate was 0.3.

For comparison, two low-density parity-check coding schemes were created. One $R = 0.3$, $j = 3$ low-density parity-check code used directly on the channel and the other an $R = 0.37$, $j = 3$ low-density parity-check code used with the optimum timesharing proportion $t_f^* = 81\%$ to give a user perceived code rate of 0.3.

The results from simulations on the parallel Z channel are plotted in figure 5.7. It can be seen that the Sparse-Dense code outperforms the other benchmark schemes.

5.3.2 Parallel BSC

A similar simulation was carried out with the parallel BSC and three $N = 10000$, $R = 0.2$ codes. The three codes were a low-density parity-check code, a low-density parity-check code used with timesharing and a Sparse-Dense code created with $p_{d1} = 0.11$. The simulation results are shown in figure 5.8. A gain over both the benchmark systems can again be observed for the Sparse-Dense code.

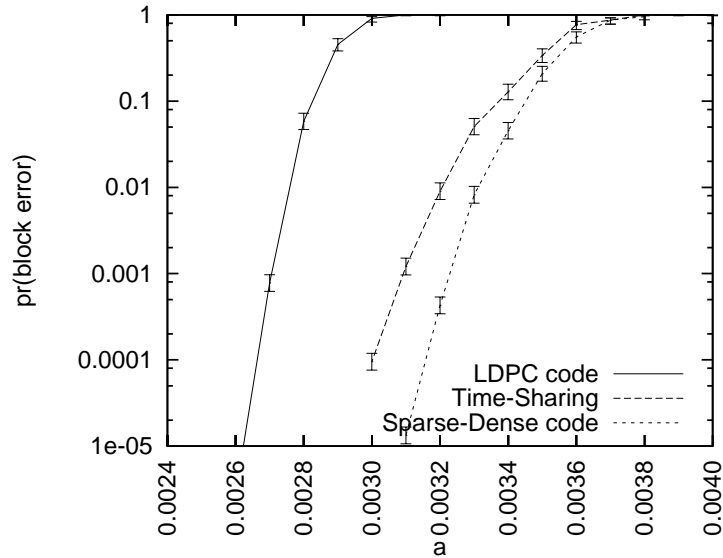


Figure 5.8: The performance of two $R = 0.2$, $N = 10000$ codes on a 128 user parallel BSC

5.3.3 Discussion

For the parallel BSC, sparse codes are more advantageous than for the parallel Z channel; for the single user Z channel the optimum input distribution has $p_1 > 1/2$. For the multiuser channel, the pressure to reduce the noise and the single-user optimum input distribution work against each other. The smaller gain from sparse transmission can be seen in both the simulations and capacity calculations. It is expected that Sparse-Dense codes on a parallel Z channel with the opposite asymmetric error (so 0s are transmitted reliably and 1s corrupted) would show a larger gain.

The codes chosen were not optimized in detail for the channel model. The capacity graphs were used as a guide for the parameters to use. Optimization of R_d and p_{d1} could be carried out. Also irregular low-density parity-check codes could be used with degree sequences optimized for the two different channel models.

It is probable that a further gain could be achieved by timesharing with a Sparse-Dense code to counter some of the effects of the dense parity bits.

5.4 Conclusion

We have presented a technique for creating sparse blocks of data and demonstrated a coding scheme using such a sparsifier to produce a gain over traditional linear codes on multi-user channels.

CHAPTER 6

SPARSE LDPC CODES FOR MULTI-USER CHANNELS

6.1 Introduction

In this chapter we present another coding solution for the multi-user channels presented in chapter 4. For a Sparse LDPC code, every codeword has the property that all the bits are sparse.

6.2 Initial Experiments with Binary Low-Density Parity-Check Codes

The work started by looking at binary low-density parity-check codes with a mapper similar to McEliece [74]. This mapper takes a set of equiprobable bits from a binary low-density parity-check encoder and converts them to an output bit with a biased distribution.

The mapper function ideally needs to have the following properties:

- The function is such that given bits with uniform input probabilities the probability of the output being 1 is p_1 .
- The input bits should be used symmetrically. This avoids problems where one input bit often does not affect the output bit. For example if the mapper took the binary representation of the input bits and performed a threshold, the least significant bit would often not affect the output.
- For belief propagation it is useful that given the output and all the inputs bar one, the remaining input digit can be inferred. One idea was to use the output bit to represent the function “are the inputs a valid codeword of a particular code?” In particular a Hamming code would have good properties due to being a perfect code, appendix G. However given received bits this mapper is not capable of passing any information to

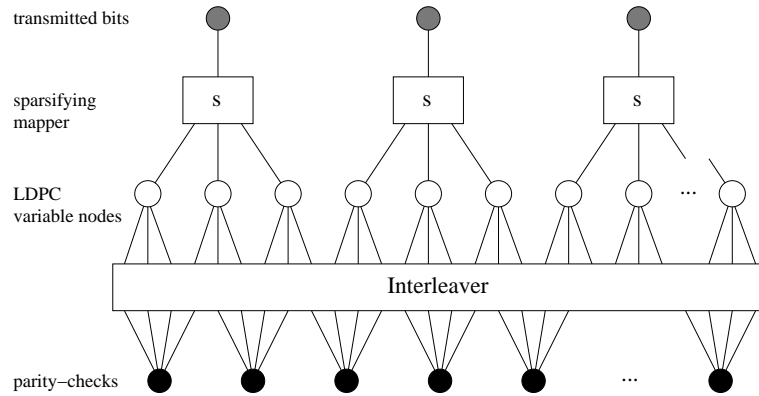


Figure 6.1: A binary sparse LDPC code with non-overlapping sparsifying mappers

a code initially, therefore it does not decode with a belief propagation decoder. This holds for any linear code.

A “voting” function was tested – output 1 if the number of 1 inputs is less than or equal to a threshold value. This function can be tuned to give the correct p_1 and is symmetric with respect to its inputs. However one can only infer a last remaining input bit when the number of input bits being 1 is near the threshold value. A 7-input function was used with a threshold value of 2 to give $p_1 = 0.23$.

The first configuration studied was a binary low-density parity-check code with non-overlapping chunks of bits as inputs to the mapper, as illustrated in figure 6.1. Unfortunately not enough information could be passed down to the low-density parity-check decoder to start decoding for code rates close to capacity. The binary low-density parity-check decoder only deals with marginalised bit-by-bit probabilities – the mutual information between the bits in each chunk is significant. The performance under belief propagation decoding is disappointing; under maximum likelihood decoding, it has been shown that codes constructed similarly are capable of achieving capacity [4, 40].

It was hoped that by linking each variable node to several different mappers the channel beliefs passed down to the rest of the code could be strengthened. The connection properties were chosen so that the information content of the low-density parity-check code bits was the same as the sparsified bits, $N_{\text{LDPC}} = NH_2(p_1)$. The construction is shown in figure 6.2. The interleaver between the variable nodes and sparse mapper nodes was constructed using Algorithm 1A [70] to ensure no short loops which might impair belief propagation. Unfortunately performance of this construction was not good due to poor distance. It was possible for two distinct low-density parity-check codewords to map to identical transmitted sequences.

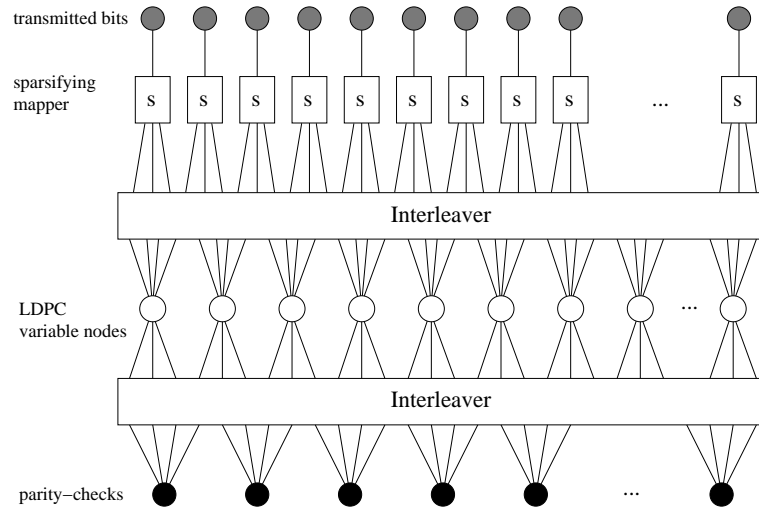


Figure 6.2: A binary sparse LDPC code with overlapping sparsifying mappers

6.3 Sparse LDPC codes

The mutual information problems found with binary low-density parity-check codes with non-overlapping mappers can be handled by low-density parity-check codes defined over $\text{GF}(q)$.

6.3.1 Mapper

To define a Sparse LDPC code we take a low-density parity-check code over $\text{GF}(q)$ and use a time-varying function to translate individual code symbols to channel bits, as illustrated in figure 6.3. To obtain sparse transmissions, most of the symbols in $\text{GF}(q)$ are mapped to 0. The fraction of symbols mapped to 1 is p_1 . We hope that, with a function chosen at random for each symbol node, all the codewords of the code are mapped to distinct sparse sequences.

If we treat the mappers as randomly assigning codewords to sparse sequences, a pigeon-hole approach can be used to determine whether the construction of Sparse LDPC codes is possible. The probability of a codeword colliding with another can be shown to tend to 0 as $N \rightarrow \infty$ if $R < H_2(p_1)$ (where $H_2(\cdot)$ is the binary entropy defined in Appendix A).

A similar construction, called GQC-LDPC codes, has been independently developed and have been shown to have good maximum-likelihood decoding performance [4].

The decoder can view the binary channel and mapper as a single noisy channel, as illustrated in figure 6.4.

6.3.2 Degree sequence optimization

Binary LDPC code constructions with a fixed column weight, called regular codes, are popular. It is known that a column weight of 3 performs well on many standard channels. For low-density parity-check codes over larger finite fields a column weight of three does not lead

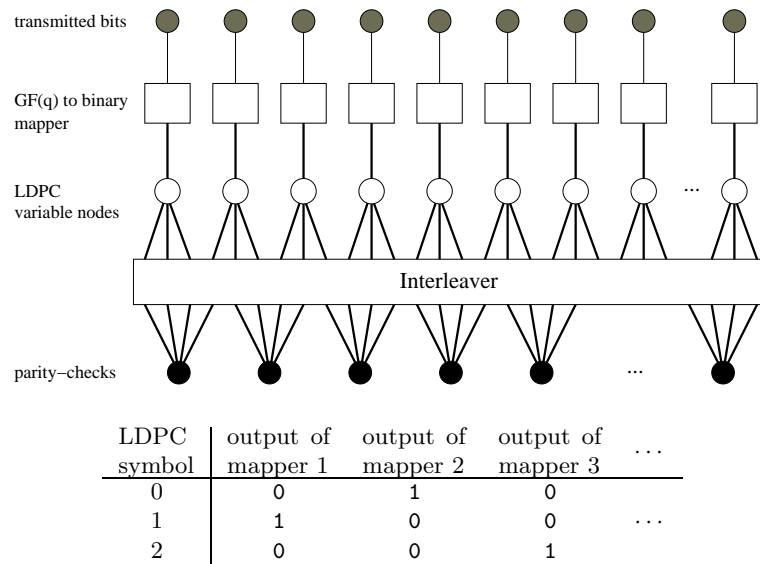


Figure 6.3: A Sparse LDPC Code. The thick lines represent constraints over $GF(q)$, the thin lines represent binary constraints. The example mappers over $GF(3)$ give $p_1 = 1/3$.

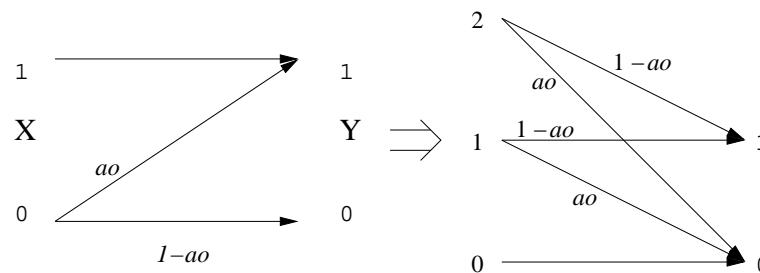


Figure 6.4: A noise model over $GF(3)$ corresponding to the parallel Z channel. The mapper $\{0\} \rightarrow 1, \{1, 2\} \rightarrow 0$ is used in this example. If applied to a $GF(3)$ LDPC code the output bits would have $p_1 = 1/3$.

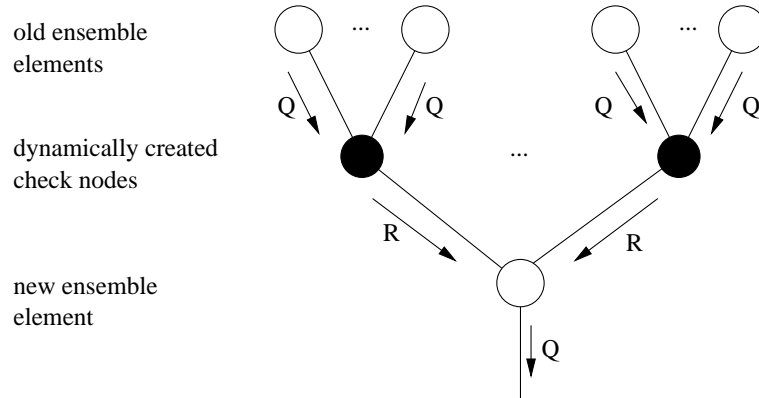


Figure 6.5: An example of the dynamically created tree around an element of the new ensemble. The downward messages from elements of the old ensemble are connected via dynamically created check nodes to the new ensemble element. In general all the Q and R messages illustrated are different.

to optimal code performance. A simple mixture of weight-2 and weight-3 columns can perform better [22].

To optimize the profile of the irregular code, we used the Monte Carlo approach of [22], modified to include the non-linear channel. We started with an ensemble of variable nodes that represented the state of variable nodes of an infinite loop-free code graph in a particular decoding iteration. We then created a new ensemble of variable nodes from the former ensemble to represent the state one iteration later. This process was repeated for a fixed number of iterations to see whether the decoding converges. By binary division, we found a noise threshold below which decoding converges and above which it does not.

In detail, we associated each variable node with a “correct” $\text{GF}(q)$ symbol, and a vector, Q , of messages from it. As the infinite code had a tree structure, each symbol node had one message coming out of it. To create an element of the new ensemble we first chose a “correct” symbol from $\text{GF}(q)$ and used the channel noise model to simulate the received signal. Then we created a tree fragment above it, for example figure 6.5, by first choosing a column weight, c , as a sample from the degree distribution. Then we created $c - 1$ incoming check nodes. For each of these we chose a row weight, r , by sampling from the row distribution weighted by the row weight. We then connected it to $r - 1$ incoming symbol nodes from the old ensemble. These nodes were sampled from the column distribution weighted by the column weight with the constraint that the chosen nodes led to the check node being satisfied with their correct transmissions. Belief propagation was then carried out down the tree to evaluate Q for the new node. This process was repeated for each element in the new ensemble.

For each of the two channel models we assumed 128 users and a line search was carried out to find the best combination of weight-2 and weight-3 columns for particular code parameters, figure 6.6. The best average column weight was 2.35 for the parallel BSC channel for codes defined over $\text{GF}(5)$ with $p_1 = 1/5$ and $R = 0.2$. Similarly 2.35 was the best average column

Col Weight	Proportions		
2	0.65	0.715	0.635
3	0.35	0.193	0.302
5		0.092	
10			0.063
Threshold/0.001	8.48	8.74	8.91

Table 6.1: Thresholds of three degree sequences for codes defined over $\text{GF}(3)$ with $p_1 = 1/3$ and $R = 0.3$ over the parallel Z channel with 128 users.

weight for the parallel Z channel with codes defined over $\text{GF}(3)$, $p_1 = 1/3$ and $R = 0.3$.

A search for better fully irregular degree sequences using a global optimization package (DIRect [38]) was carried out. Some degree sequences for the parallel Z channel are shown in table 6.1.

The thresholds found in figure 6.6 may be compared with those of the simple time-sharing scheme presented in chapter 4. For the above parallel BSC, the Shannon limit for time-sharing is $a = 0.00463$ with $t_f^* = 54\%$ (the optimal fraction of time for which each transmitter is active). Similarly, for the parallel Z channel, the Shannon limit is $a = 0.00832$ with $t_f^* = 81\%$. Both of these Shannon limits are smaller than the respective Sparse LDPC code thresholds found above.

6.4 Simulation results

An $N = 10000$, $R = 0.2$, $p_1 = 1/5$ code was created for testing on the parallel BSC. These parameters were chosen to match simulations in chapter 5. The base code was defined over $\text{GF}(5)$ with $N = 10000$, $M = 9139$ with 60% weight-2 columns and 40% weight-3 columns. Random data symbols were chosen, encoded, transmitted and then decoded. In figure 6.7(a) the simulation results are compared with those from a normal low-density parity-check code, a low-density parity-check code with time-sharing and a Sparse-Dense code.

An $N = 10000$, $R = 0.3$, $p_1 = 1/3$ code was created for testing on the parallel Z channel. The base code over $\text{GF}(3)$ had $N = 10000$, $M = 8107$ and two degree sequences from table 6.1 (the “semi-regular” degree sequence with only weight-2 and weight-3 columns and the degree sequence with weight 2, 3 and 10 columns were used). Simulation results with the parallel Z channel are shown in figure 6.7(b).

For the two channels studied above, the Sparse LDPC codes can be seen to outperform the other coding schemes tested by 1.6dB and 1.1dB respectively.

6.5 Discussion

Even in the event of an uncorrupted transmission being received, a decoding is required to recover the original $\text{GF}(q)$ symbols. This approach is best suited for use with good, large

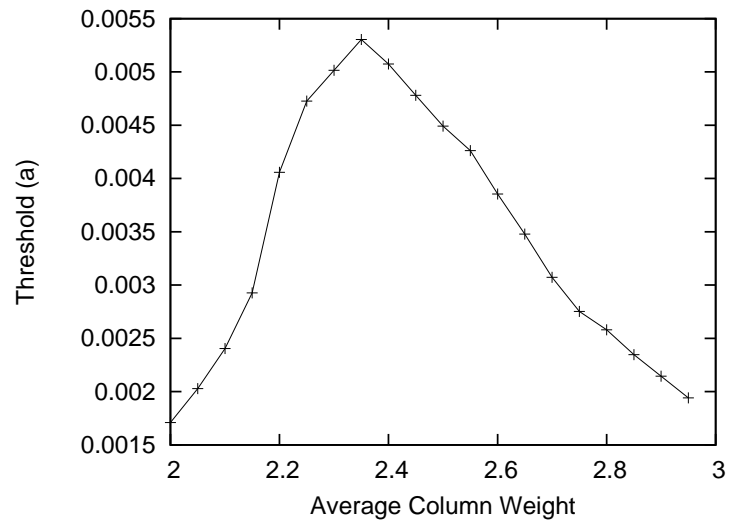
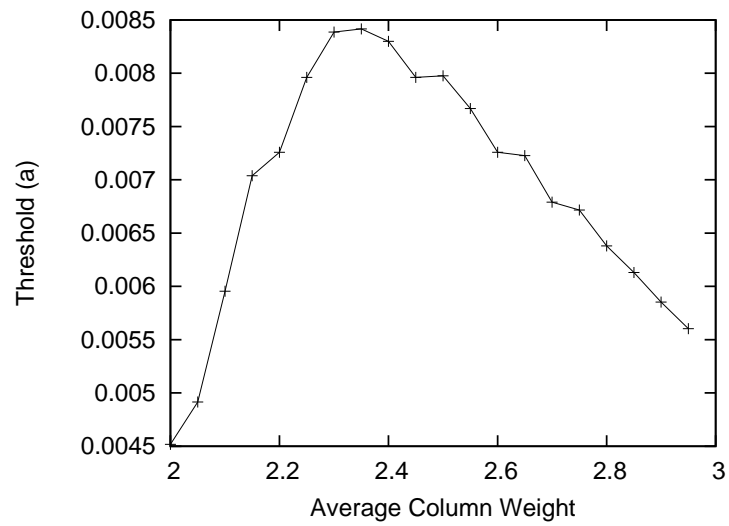
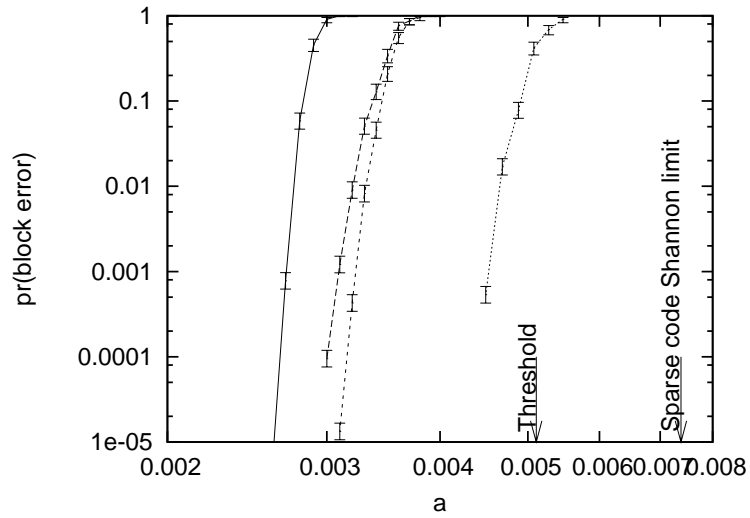
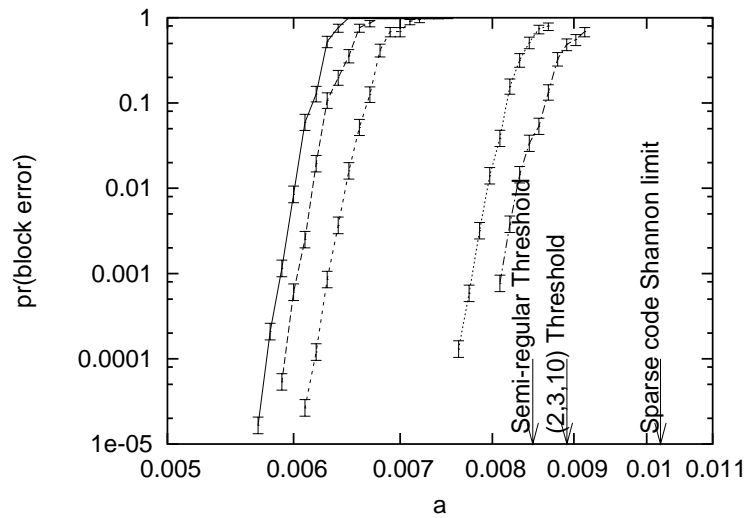
(a) Parallel BSC, GF(5), $R = 0.2$, $p_1 = 1/5$ (b) Parallel Z channel, GF(3), $R = 0.3$, $p_1 = 1/3$

Figure 6.6: Threshold values



(a) Parallel BSC, $R = 0.2$



(b) Parallel Z Channel, $R = 0.3$

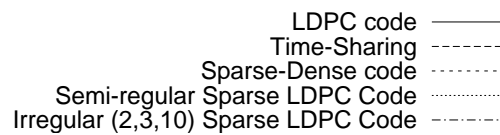


Figure 6.7: Codes with $N = 10000$ on 128-user channels. The low-density parity-check codes, time-sharing schemes and Sparse-Dense codes are the same as in chapter 5. The time-sharing schemes are low-density parity-check codes being used with the optimum time-sharing fraction.

blocksize codes. For small blocksize codes, block errors might still be possible at zero noise level.

The complexity of decoding at the check nodes increases with the size of the finite field used, appendix H. If a value of p_1 is needed that can not be obtained with smallish fields then the coding technique presented may become computationally infeasible. We have used a uniform value of p_1 throughout a codeword – it is possible that using a different value for each codeword symbol might help the decoding process and lead to better code performance.

The method of code construction could be studied further. It may be possible to choose the non-zero values in \mathbf{H} to create a code with better distance properties.

The performance on single-user asymmetric channels could be tested – Sparse LDPC codes might be able to outperform linear codes on these channels.

6.6 Conclusion

The codes presented here show a substantial gain over linear codes for a multi-user channel. The codes have a modified linear encoder and a message-passing decoder and thus are easier to implement than many non-linear codes.

CHAPTER 7

INSERTION-DELETION CHANNELS

7.1 Introduction

Examples of channels with synchronization errors include:

Serial line The clock speed of the transmitter may not be accurately known (for instance due to temperature variations in the clock) so the time of arrival of each transmitted bit is not known.

Hard disc Variations in the rotation speed (for instance due to mechanical vibrations or shock) mean the position of the head relative to the platter may be uncertain.

DAT tape Tape stretch leads to problems similar to those suffered by a hard disc.

In this chapter, such a channel is modelled by random uncorrelated insertion or deletion events at unknown positions. A flowchart of the channel model is shown in figure 7.1. The capacity for channels of this kind is not known exactly. A capacity lower bound from [120] and an upper bound from [107] are used in this chapter.

Marker codes [58] were originally designed to be able to deal with single insertion or deletion error events. The bit stream to be transmitted has a regular marker (or header) inserted in it. For example the marker ‘001’ may be inserted between every 4 data bits:

$$01101100101010 \mapsto 01100011100001101000110$$

The decoder can look for the markers and use any shift in their position to deduce bit loss or gain. Errors in the matched sequence can then be corrected with a conventional code. With advances in computer power probabilistic sequence matching, as described below, can be carried out. The coding system is shown in figure 7.2.

Watermark codes [24] are a similar scheme, but rather than having bursts of synchronization information and bursts of data, the information is distributed uniformly. To encode, the data bits are uniformly sparsified and then added to a watermark sequence. To decode, probabilistic resynchronization can be carried out with the watermark sequence.

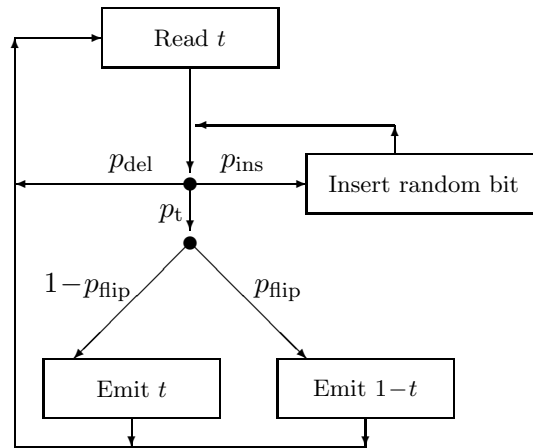


Figure 7.1: Flow chart (from [83]) describing the binary insertion-deletion channel with insertion probability p_{ins} , deletion probability p_{del} , transmission probability $p_t = 1 - p_{\text{ins}} - p_{\text{del}}$, and substitution probability p_{flip} . For simplicity $p_{\text{flip}} = 0$ for the simulations in this thesis.

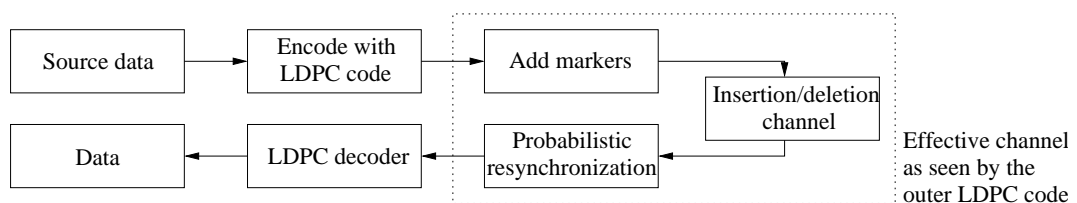


Figure 7.2: The code system used, with serial decoding illustrated

In the literature, watermark codes appear to present the best simulation results. Other coding schemes have been presented which may be of interest for computational or theoretical reasons. A few are listed below:

Comma-free codes The codewords are constructed so that they have the property that no overlap between the codewords can be confused as a codeword [99]. If a codeword is corrupted with an insertion or deletion it is possible to regain synchronization after the error, however error correction within the corrupted codeword is not generally possible [11].

Convolutional coding In [27] a standard convolutional code encoder with a set of normal decoders is used. This leads to a computationally efficient scheme, but the results are disappointing. A modified convolutional encoder and decoder are used in [100] to achieve promising results.

Levenshtein codes In a similar manner to the Hamming distance, the Levenshtein distance [63] between words is the minimum number of insertions or deletions necessary to get from one word to another. Codes exist which try to optimize the minimum Levenshtein distance between codewords but practical codes with good distance properties have not yet been developed.

Schulman and Zuckerman codes Concatenated codes that are asymptotically good are presented in [91] but experimental results are not developed. The codes appear to mainly be of theoretical interest [24].

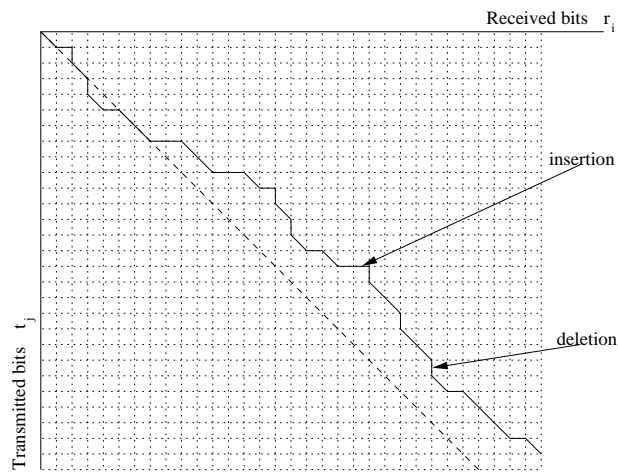
See [98] for good coverage of older schemes.

Earlier work on marker codes [83] presented an experimental way to optimize the markers based on the capacity of the effective channel illustrated in figure 7.2. Algebraic optimization of markers has also been attempted [33, 59]. In this chapter the results from [83] are extended by comparing complete marker-code-based systems with watermark codes. The benefit of iterative probabilistic resynchronization is also studied. We show simulation results that outperform the best known results.

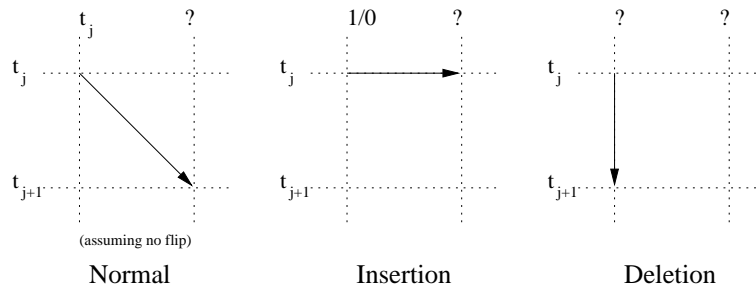
7.2 Probabilistic Resynchronization

We use the forward-backward algorithm to find the conditional probability distribution of the transmitted bits given the received bits. We follow the presentation of [83], modified to include iterative probabilistic resynchronization.

Figure 7.3(a) shows a representation of synchronization errors. The solid line indicates one particular sequence of insertions and deletions which then defines a mapping from the transmitted bits to the received bits. If there were no insertions or deletions the mapping would follow the dashed line – this represents a one-to-one mapping between the received



(a) An example path



(b) The allowed moves

Figure 7.3: Synchronization represented by a path on a two dimensional grid

and transmitted bits. To correspond to the three operations of the channel there are three different types of moves possible on the grid, illustrated in figure 7.3(b):

Normal transmission (possibly including a bit flip) A diagonal move in which the top left-hand end of the move lies on the intersection between the row and column corresponding to the transmitted and received bits respectively.

Insertion This corresponds to a horizontal move in which an extra random bit appears in the received stream at the left hand end of the move.

Deletion This is a vertical move such that the transmitted bit corresponding to the top of the move does not have a position in the received stream.

We assume prior probabilities, $g_j = \Pr(t_j = 1)$, on each transmitted bit. The probability of a particular path and set of received of data is:

$$\begin{aligned} \Pr(\text{path}, \mathbf{r}|\mathbf{g}) &= P(\text{path}) \cdot P(\mathbf{r}|\text{path}, \mathbf{g}) \\ &= \prod_{\text{insertions}} p_{\text{ins}} \prod_{\text{deletions}} p_{\text{del}} \prod_{\text{normal}} (1 - p_{\text{del}} - p_{\text{ins}}) \cdot \prod_{\text{insertions}} \frac{1}{2} \prod_{\text{normal}} \kappa(r_i, g_j) \end{aligned} \quad (7.1)$$

$$(7.2)$$

where

$$\kappa(r, g) = \Pr(r|g, \text{normal transmission}, p_{\text{flip}}) \quad (7.3)$$

For non-iterative resynchronization g_j is 0 or 1 in a marker and $\frac{1}{2}$ otherwise. In iterative resynchronization these “prior” probabilities outside the markers are the messages passed back from the outer low-density parity-check code. The probability of the path is given by a product of the probabilities for each insertion (p_{ins}), deletion (p_{del}) and normal transmission. The probability of the received data for an inserted bit is $\frac{1}{2}$ (in our channel model it is equally likely to be a 0 or 1). Deletions do not give any received data and hence do not contribute towards the probability of the received data.

When decoding we use the forward-backward algorithm [28] to marginalize across all paths. We define a forward probability at a position (i, j) :

$$p_f(i, j) = \Pr(\text{path goes through } (i, j), r_1 \dots r_{i-1} | \mathbf{g}) \quad (7.4)$$

To keep the same notation as [83], $p_f(i, j)$ has been used as a forward probability – this should not be confused with other uses of p_f in this thesis. There are only three ways the path can get to (i, j) – it can arrive by a horizontal, vertical or diagonal move from an adjacent space. So $p_f(i, j)$ can be found recursively:

$$p_f(i, j) = p_{\text{ins}} \frac{1}{2} p_f(i-1, j) + p_{\text{del}} p_f(i, j-1) + (1 - p_{\text{del}} - p_{\text{ins}}) \kappa(r_{i-1}, g_{j-1}) p_f(i-1, j-1) \quad (7.5)$$

as shown in figure 7.4. The boundary conditions, assuming that the synchronization error is known to be zero initially, are $p_f(0, 0) = 1$, and $p_f(i, j) = 0$ for all points not on the grid.

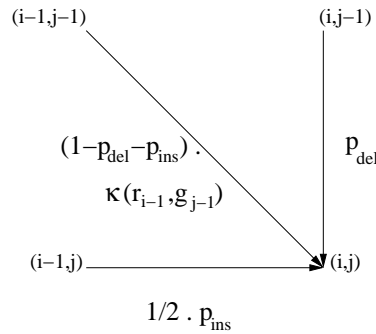


Figure 7.4: The recursive evaluation of the forward probabilities

Similarly, we define the backward probabilities:

$$p_b(i, j) = \Pr(r_i \dots r_N | \text{path through } (i, j)) \quad (7.6)$$

The backward message-passing algorithm for computing $p_b(i, j)$ is similar to the forward algorithm above. The boundary conditions are $p_b(i, j) = 0$ for (i, j) not on the grid, and $p_b(N, i) = 1$ for all rows i , where N is the last column of received data.

The forward and backward messages p_f and p_b are used to infer the posterior probability of each user bit, $P(t_j | \mathbf{r})$, by marginalizing over the diagonal and vertical edges in row j .

7.3 Comparison with Watermark Codes

In [83] it was shown that for a code system like that in figure 7.2 the capacity of the effective channel seen by an outer code is higher for marker codes than for watermark codes at low noise levels. The highest rate results published for watermark codes [24] are at $R = 0.71$ with a block size of 4995. To match this a marker code of the same overall rate and block size was created, code A. The inner code was chosen to be good at a noise level $p_{\text{ins}} = p_{\text{del}} = 0.005$ using the effective capacity technique outlined in [83]. The outer code was a low-density parity-check code with weight-2 and weight-3 columns. The code parameters are shown in table 7.1 on page 61. Decoding was as figure 7.2.

A comparison of the watermark and marker codes is shown in figure 7.5. The marker code outperforms the watermark code, despite the watermark code having been constructed with a code defined over a larger field ($\text{GF}(16)$) than the binary codes considered here.

The performance of the marker code is not close to the channel capacity bounds. To try to approach the bounds, the outer code was optimized. The threshold of an infinite loop-free low-density parity-check code as a function of column weight distribution was evaluated using a Monte Carlo approach [22]. In this procedure it is necessary to know the distribution of messages sent by the channel. Statistics of the messages received by the outer code were collected, figure 7.6. The distribution is almost symmetrical if given a data sequence

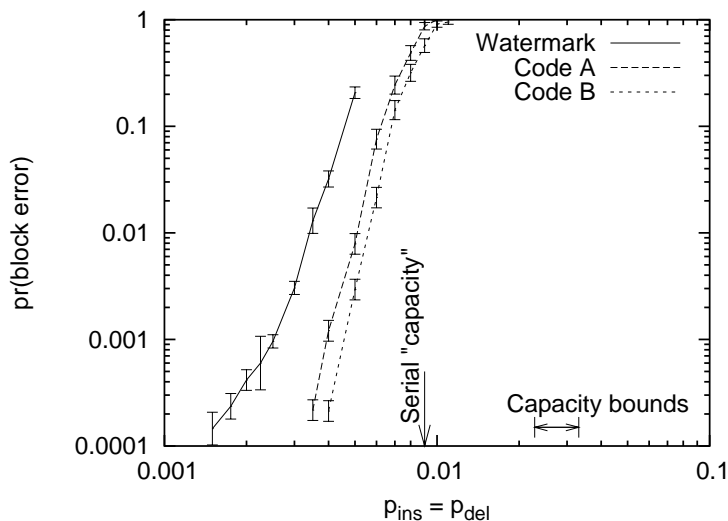


Figure 7.5: $R = 0.71$, $N = 4995$ codes over an insertion/deletion channel with serial decoding. The watermark code result is from [24]. Marker code A is with a low-density parity-check code with with weight-2 and weight-3 columns. Marker code B has weight 10 columns in addition, table 7.1.

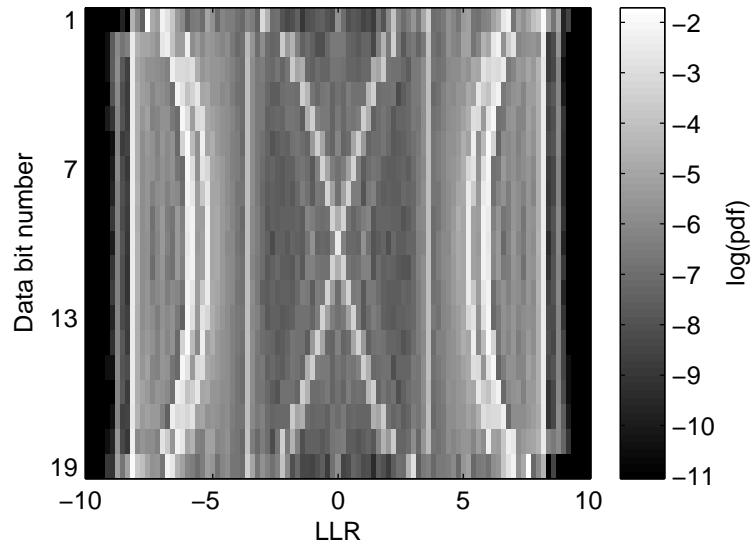
of i.i.d. bits. The simulation was carried out with an all zero-transmission but with noise statistics as if they were i.i.d. data bits. Despite the non-Gaussian form of the messages, degree sequences could not be found that significantly outperformed good degree sequences obtained from optimizations on the Gaussian channel [16]. This optimization was carried out with the message histograms marginalized over all bits, figure 7.6(b). Perhaps if a different degree sequence for each bit between markers were allowed, a further gain could be achieved; as figure 7.6(a) shows, bits received near markers are more reliable than bits far away from markers.

Simulations using a good degree sequence from the Gaussian channel, code B, are also shown in figure 7.5. The waterfall region is not much closer to the channel capacity bounds, but it is close to the serial “capacity” (defined from the effective capacity of the channel seen by the outer code [83]).

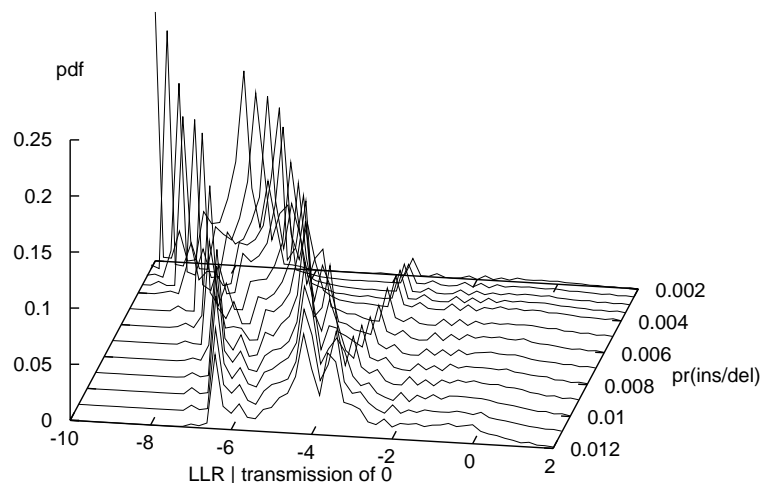
7.4 A Complete Iterative System

In chapter 3 we saw that the extension of loopy belief propagation to include estimating channel state can be beneficial. For the insertion-deletion resynchronization phase it is expected that if we give the resynchronization algorithm more information about the likely transmission the accuracy of the resynchronization will increase.

Watermark and regular marker codes were empirically discovered to have similar performance near $R = 0.5$ by looking at the capacity of the effective channel [83]. Therefore $R = 0.5$ codes with a blocksize of 4000 (to match [24]) were studied.



(a) At a noise level of $p_{\text{ins}} = p_{\text{del}} = 0.5\%$. The vertical axis ranges over each data bit between two markers. Close to a marker the messages are more confident as a larger value of the absolute LLR is more likely.



(b) Marginalized over all data bits whose transmitted bit was '0'

Figure 7.6: Histograms of the probability density functions of messages received by an outer code where the inner code has a marker sequence of '01' between every 19 data bits. The log-likelihood ratio (LLR) is $\frac{\Pr(\text{bit}=1)}{\Pr(\text{bit}=0)}$.

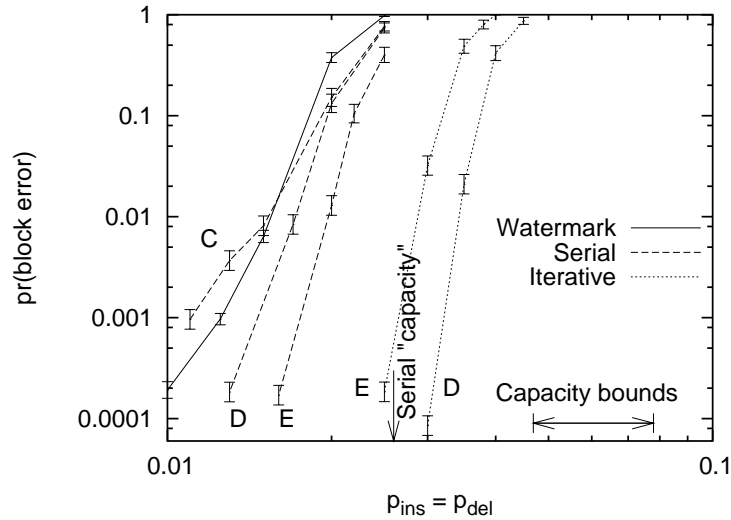


Figure 7.7: The benefit of full iterative decoding with $R = 0.5$, $N = 4000$ codes. Identical marker codes are shown simulated with iterative and non-iterative resynchronization. Also the reduction in error floor from identical markers (C) to non-identical markers (D) is shown. The outer codes in codes C and D have only weight 2 and 3 columns, code E has weight 10 columns in addition. The watermark code result is from [24] and the details of the marker codes are in table 7.1.

As benchmarks, marker codes were decoded with the serial decoding algorithm from section 7.3. The first simulation (code C) was with identical markers of length 3 and a low-density parity-check outer code with weight-2 and weight-3 columns. Figure 7.7 shows that the watermark code has better error floor behaviour. With identical markers catastrophic error propagation is possible in a marker code as the resynchronization can be shifted by a multiple of a marker interval. Code D was similar to code C however each marker was pseudo-randomly chosen from a set of two different markers. This improved the error floor and led to better performance than the watermark code. An irregular low-density parity-check outer code (chosen to be good on the Gaussian channel) was also tested and a small further improvement found, code E.

Simulations were carried out with codes D and E using iterative resynchronization. The algorithm is similar to the serial resynchronization algorithm but with extrinsic information from the low-density parity-check decoder fed back into the resynchronization stage (updating the values for \mathbf{g} in equation 7.2). The resynchronization was carried out every five iterations of the low-density parity-check decoder to increase the decoding speed as a low-density parity-check decoder iteration is faster than probabilistic resynchronization.

The simulation results are shown on figure 7.7. The figure shows that the iterative approach significantly outperforms the serial approach and that the waterfall region can be close to the channel capacity. It is worth noting that the ranking of the codes is reversed between serial and iterative decoding. This suggests that the choice of code to be used should be made

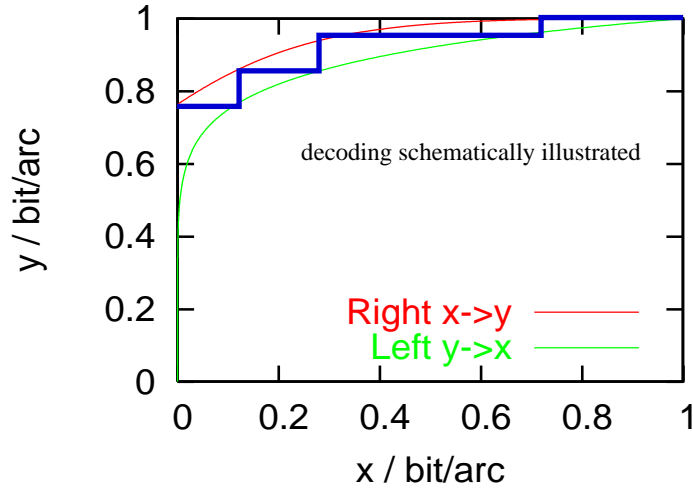


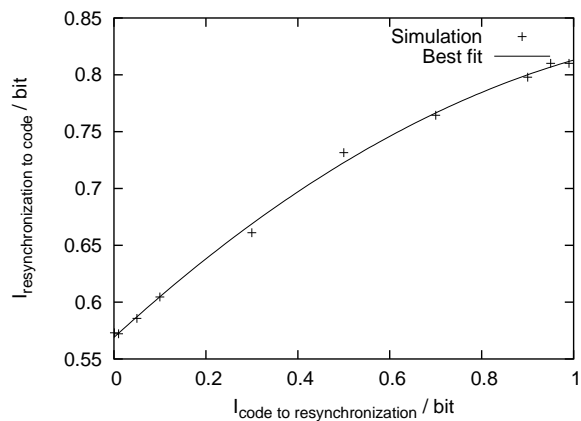
Figure 7.8: An example of an EXIT chart of a graph split into two sections (“right” and “left”). A condition where decoding is expected to converge is shown with a schematic progress of decoding shown with a thick line.

in conjunction with the decoding algorithm.

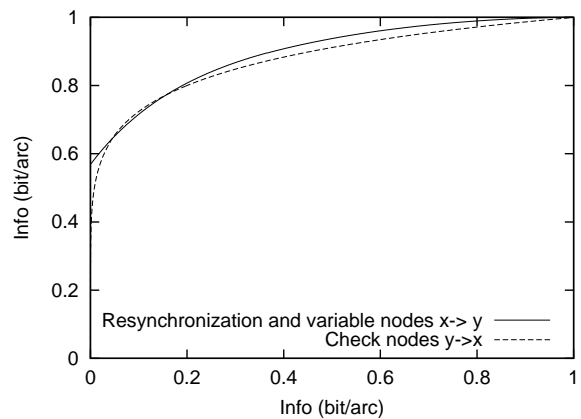
7.5 EXIT charts

To look into this swap in performance between serial and iterative decoding, extrinsic information transfer (EXIT) charts [104] for the system were studied at a noise level of $p_{\text{ins}} = p_{\text{del}} = 0.04$. EXIT charts allow a visualization of the messages passed during decoding between two loop-free sections of the graph describing the decoding. When a section is loop-free an “exact” inference can be carried out in that subsection of the graph. At the limit of large block size we can then look at the average statistical properties of the messages into that section versus the messages coming out of that section. We assume a Gaussian form to the distribution of messages. The transfer function of the two graph sections can be put on opposing axes and then the expected progress of infinite-blocksize decoding can be seen, as shown in figure 7.8. A “staircase” is formed between the two curves with a step per iteration. To decode, the staircase needs to reach the top right-hand corner (which indicates totally confident messages). If there is an intersection between the two curves this is not possible and a code is not expected to decode.

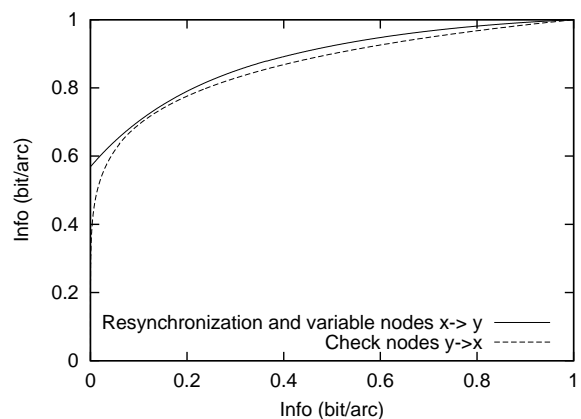
The EXIT chart of the resynchronization was obtained by a Monte Carlo approach and a quadratic function fitted, figure 7.9(a). With no information passed from the code to the resynchronization stage the maximum rate of the outer code with serial decoding is shown at the intercept with the y -axis. As the input information is increased the output information increases but does not reach 1. This is due to remaining uncertainty in the exact synchronization path and whether the bit in question may have been deleted (and possibly



(a) Insertion deletion channel



(b) Code E



(c) Code D

Figure 7.9: EXIT charts at $p_{\text{ins}} = p_{\text{del}} = 0.04$

reinserted).

The behaviour of the decoding algorithm can be seen in terms of messages being passed between the variable nodes and check nodes; a section made up of check nodes and a section made up of variable nodes and resynchronization are each individually loop-free. The transfer functions of the check and variable nodes were as in appendix I. The code that performs well on the Gaussian channel leads to an EXIT chart with an intersection, figure 7.9(b), so decoding is not expected to converge. For the code with only weight-2 and weight-3 columns no intersection is observed, figure 7.9(c), and therefore decoding is expected to converge. A code with a softer response often performs better as part of an iterative decoding scheme; more evidence of this will be seen in chapter 9.

The width of the swath between the curves can be used as a metric to choose a better degree sequence. We only have the resynchronization EXIT chart at one noise level. To create a better code we want to find a form of curves that is less likely to have an intersection at a higher noise level. As the noise increases the top curve moves down the chart, so keeping a wide swath between the curves allows a larger noise level to be reached until an intersection occurs. Better degree sequences were searched for using a global optimization package [38]. Better thresholds could only be found by increasing the number of weight-2 columns above the number of rows in the parity-check matrix. This is not expected to produce a good code as short cycles in weight-2 columns lead to low weight codewords.

7.6 Deletion Channel

Interest in the academic community has recently focused on the bit-deletion channel (insertion-deletion channels where $p_{\text{ins}} = 0$). Researchers have generally used similar codes to those used on the insertion-deletion channel [97].

Another example of a deletion channel is the packet-deletion channel. Whole data packets can be randomly lost on the internet. The channel model is similar to figure 7.1 however instead of looking bit-by-bit we look on a packet-by-packet basis. Intelligent ways of coping with this loss are of commercial interest. For systems with large packets, the packet-deletion channel capacity is close to a packet erasure channel [25]. The deletion capacity shows an asymptotic loss of 1 bit per packet compared to the erasure capacity. The protocol currently used for most internet transmissions (TCP) uses 32 bits to number the packets [80] – however more efficient schemes to convert the deletion channel to an erasure channel should be possible. Packet-erasure channels are handled well by Raptor Codes [95].

Simulations of the marker codes developed in this chapter for the insertion-deletion channel are shown on the bit-deletion channel in figure 7.10. The codes significantly outperform other known deletion codes (for example allowing approximately twice the transmission rate of [15]).

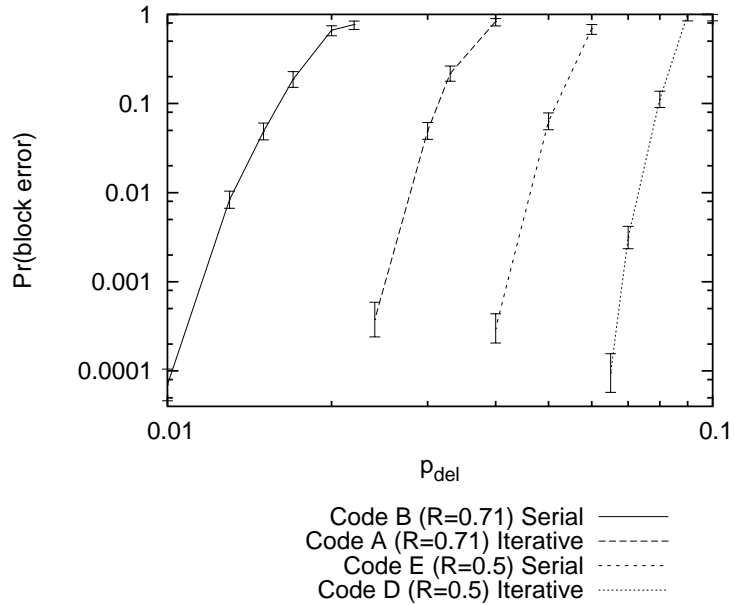


Figure 7.10: Codes of $R = 0.5$ and $R = 0.71$ simulated on the deletion channel ($p_{\text{ins}} = 0$). The lower bounds on the Shannon Limit for $R = 0.71$ and $R = 0.5$ are $p_{\text{del}} = 0.0509$ and $p_{\text{del}} = 0.110$ respectively.

	A	B	C	D	E
R	0.71	0.71	0.5	0.5	0.5
N	4995	4995	4000	4000	4000
d	19	19	9	9	9
m	01	01	001	001/110	001/110
N_{LDPC}	4521	4521	3001	3001	3001
M_{LDPC}	969	969	1001	1001	1001
c_2	968	963	1000	1000	927
c_3	3553	2785	2001	2001	1572
c_{10}	0	773	0	0	502

Table 7.1: The codes used in this chapter. Markers (m) are inserted between every d data bits. If more than one marker is available, the marker is chosen pseudo-randomly. The low-density parity-check outer code was constructed with c_i columns of weight i .

7.7 Conclusion

We have shown that marker codes outperform watermark codes at rates above 0.5. To avoid catastrophic decoding errors at higher noise levels the use of pseudo-random markers from a set of different markers is necessary.

Iterative resynchronization provides better performance than serial resynchronization, bringing the waterfall region close to the bounds on the channel capacity.

CHAPTER 8

CONVOLUTIONAL CODES

8.1 Introduction

Large block sizes are impractical for channels with very slow data rates if the sender and receiver wish to communicate without substantial average delay. A code that allows early decoding (decoding before an entire block is received) with good performance is needed.

Infinite random tree codes (figure 8.1) can reach the Shannon Limit [32]. They seem appropriate for low bandwidth channels: a decoding can be attempted at any time and the data earlier on in the transmission becomes progressively better protected by data received later. Unfortunately infinite random tree codes have infinite complexity. Convolutional codes [57] approximate this structure and are simple to both encode and decode. An encoder can be visualised as a linear sequential circuit, for example figure 8.2. The encoding can alternatively be viewed in a similar manner to a tree code, in which the number of states of the system remains bounded, forming a trellis, for example figure 8.3. For information on the parameters describing a convolutional code and their systematic form see appendix E.

Convolutional codes are often used as block codes by terminating the input stream with dummy zeros. This termination forces the system to a known state at the end of a block and hence provides equal protection to the bits at the beginning and end of the block. However if one is decoding a continuous stream or attempting a decoding in the middle of a block, one is dealing with an unterminated code. In this chapter we study this case.

To test the performance of unterminated convolutional codes, many blocks were encoded, transmitted over a simulated noisy channel, and then decoded. Various algorithms for the decoding stage were considered. Three algorithms that fully exploit the error-correcting abilities of the code are widely used:

- The Viterbi algorithm [35, 110] provides the maximum likelihood decoding of a block by visiting every branch.
- Sequential decoding [117] provides an approximation to maximum likelihood decoding. In general it uses less resources than the Viterbi algorithm as it aims to explore only branches close to the maximum likelihood path.

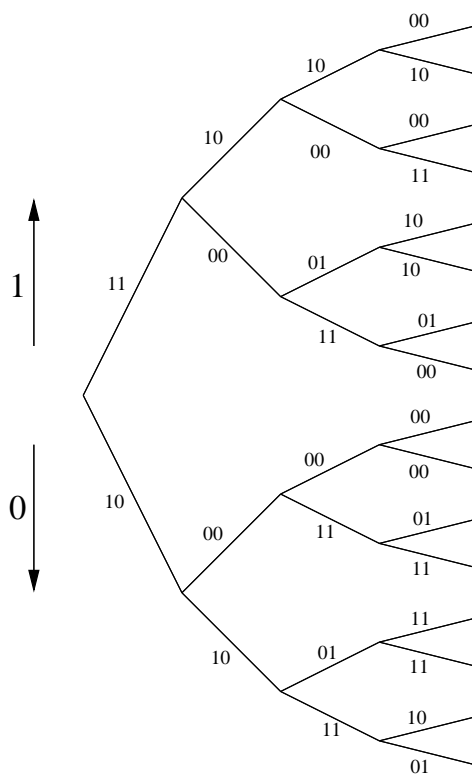


Figure 8.1: A diagram of the beginning of a $R = 1/2$ random tree code. The encoding of a stream can easily be read off the tree – start at the left and for each input bit move one state to the right and up for 1 or down for 0. The output stream consists of the labels (made up of random i.i.d. bits) along the route.

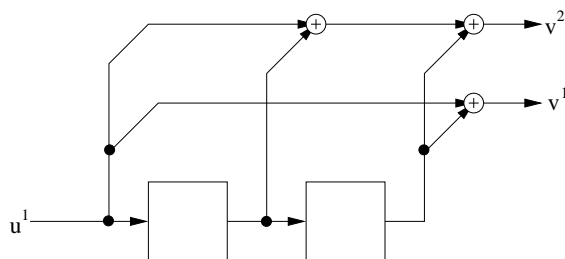


Figure 8.2: A diagram of a $R = 1/2$, $m = 2$ convolutional code in controller-canonical form. The boxes represent memory elements and \oplus modulo 2 binary addition. Each input bit leads to two output bits.

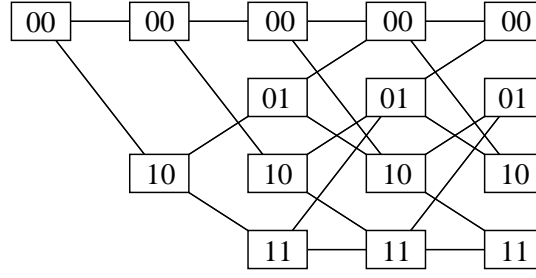


Figure 8.3: A trellis for the system shown in figure 8.2 (with four states). Only the content of the memories are shown, other labels are omitted for clarity.

- The forward-backward algorithm [2] provides the marginal posterior probabilities for each bit – this is called an *a posteriori* probability decoding. The algorithm is slower than the Viterbi algorithm as it visits every branch twice.

The maximum likelihood path produces the minimum block error probability but not necessarily the minimum bit error probabilities. The forward-backward algorithm was chosen for further study as its *a posteriori* probability decoding behaviour minimises the bit error probability.

8.2 Forward-backward algorithm

We start by defining a coordinate system on the trellis (i, j) where i is a time coordinate and j a coordinate over instantaneous state. b_i^{km} represents the path from state (t, k) to $(t+1, m)$. The set \mathcal{L} contains all the valid branches on the trellis:

$$\mathcal{L} \equiv \{(k, m) : \text{a branch links state } (i, k) \text{ to } (i+1, m)\} \quad (8.1)$$

If we receive the sequence of binary vectors $\mathbf{R} \equiv \mathbf{r}_1 \dots \mathbf{r}_N$, we then want to infer the original sequence of vectors $\mathbf{v}_1 \dots \mathbf{v}_N$. We calculate for each bit in the original data stream:

$$\Pr(v_i^l = 1 | \mathbf{R}) = \sum_{(k,m) \in \mathcal{L}} \Pr(v_i^l = 1 | b_i^{km}) \Pr(b_i^{km} | \mathbf{R}) \quad (8.2)$$

$$\propto \sum_{(k,m) \in \mathcal{L}} \Pr(v_i^l = 1 | b_i^{km}) \Pr(\mathbf{R} | b_i^{km}) \quad (8.3)$$

where we have used Bayes's Theorem with a uniform prior across the branches. The first term in equation 8.3 is either 0 or 1 depending on the output corresponding to the branch. We can evaluate the second term efficiently using the trellis of the convolutional code. We

define forward probabilities and backward likelihoods as

$$\alpha_{i,j} = \Pr(\mathbf{r}_1 \dots \mathbf{r}_{i-1}, \text{path through } (i, j)) \quad (8.4)$$

$$\beta_{i,j} = \Pr(\mathbf{r}_i \dots \mathbf{r}_N | \text{path through } (i, j)) \quad (8.5)$$

These can be evaluated recursively:

$$\alpha_{i,j} = \sum_{k:(k,j) \in \mathcal{L}} \alpha_{i-1,k} \Pr(\mathbf{r}_{i-1} | b_{i-1}^{kj}) \quad (8.6)$$

$$\beta_{i,j} = \sum_{k:(j,k) \in \mathcal{L}} \beta_{i+1,k} \Pr(\mathbf{r}_i | b_i^{jk}) \quad (8.7)$$

with the following boundary conditions for an unterminated block:

$$\alpha_{1,j} = \delta_{0,j} \quad (8.8)$$

$$\beta_{N+1,i} = 1 \quad (8.9)$$

We can then evaluate

$$\Pr(\mathbf{R} | b_i^{km}) = \alpha_{i,k} \Pr(\mathbf{r}_i | b_i^{km}) \beta_{i+1,m} \quad (8.10)$$

8.3 Experiments

A computer simulation of the communication system was tested. The performance of rate $R = 3/4$ convolutional codes over the binary symmetric channel with flip probability $p_f = 0.01$ was studied. The capacity of this channel is 0.92 and the computational cut-off rate [121] is 0.74. The process of collecting the data took the following stages:

1. A random binary data sequence of k independent identically distributed (i.i.d.) bits was created.
2. This data sequence was encoded using the chosen generator matrix.
3. All but the first $k/R = t$ bits of the encoded stream were discarded to simulate the effect of being in part of a longer data stream.
4. Bits were randomly flipped with probability p_f to simulate the received sequence.
5. The forward-backward algorithm was used to decode this received sequence and a hard decision returned.
6. This decision was compared to the originally transmitted sequence and the location of errors noted.

By repeating these tests many times and averaging, estimated error probabilities for each bit with effective decoding delays up to k could be obtained. For each bit, each sample is

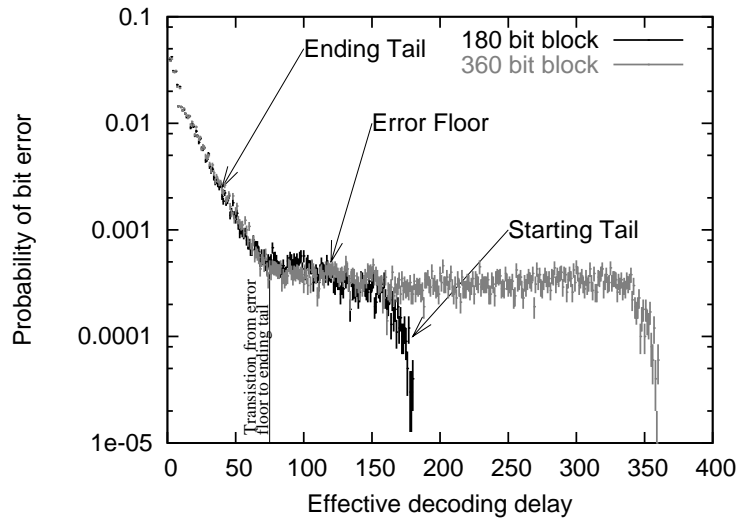


Figure 8.4: A graph of the bit error probability for transmissions using a $R = 3/4$ code over a 1% binary symmetric channel. The vertical lines represent 1σ error bars. The 180 bit block has been annotated with the regions described in the main text.

independent; however, there are correlations in errors between neighbouring bits as errors for convolutional codes tend to occur in bursts.

8.3.1 A sample set of errors

Two experiments were conducted on identical $R = 3/4$ convolutional codes (from [14] with $\nu = 9$) to determine the characteristic form of results. The code was non-catastrophic (a finite number of channel corruptions can not lead to an infinite number of errors on decoding). The transmission of 100,000 unterminated blocks of 180 data bits and of 100,000 unterminated blocks of 360 data bits were simulated. The estimated probability of error for each bit in the blocks is shown in figure 8.4.

Each graph has three distinct regions as marked:

Ending Tail. The probability of error decreases as the decoding delay increases. The protection of the bits increases as the encoder's state becomes more well known in this region.

Error Floor. For data bits in the middle of the transmission there is a constant probability of error. These bits are equally well protected and the end-effects have become unimportant. This can be confirmed by seeing that the differing length transmissions have the same error floor. It is expected that for a catastrophic code this flat region would not exist.

Starting Tail. Early in the code block the state is more well known as the encoder starts in a known state, hence the probability of error decreases.

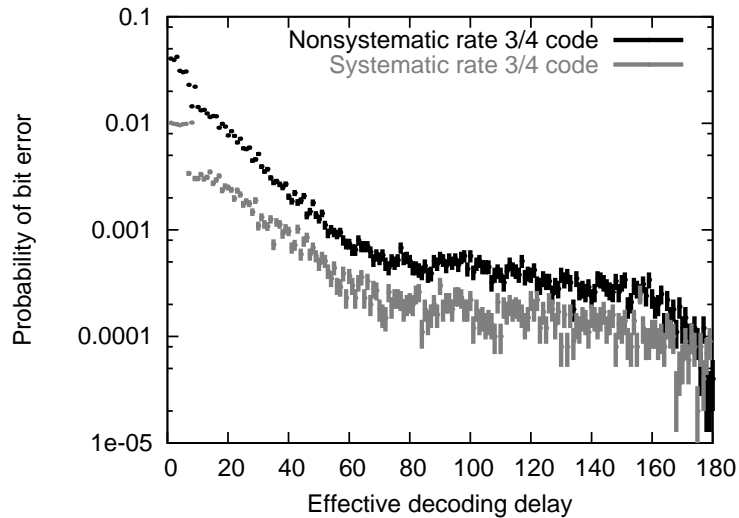


Figure 8.5: A graph of bit error probability for systematic and non-systematic generator matrices of the same $R = 3/4$ code over a 1% binary symmetric channel

8.3.2 Comparison of non-systematic and systematic generator matrices for a “good” code

An optimal $R = 3/4$, $m = 3$, $\nu = 9$ generator matrix was used from [14]. The transmission and decoding of 100,000 unterminated blocks of 180 data bits was simulated using this generator matrix. Then a systematic generator matrix (a generator that replicates the input stream within the output stream) for the same code was created and the experiment repeated. The probability of bit error for the two generator matrices is shown in figure 8.5. The systematic code has two desirable features:

- The probability of bit error is in general lower than the non-systematic code – a deviation from the correct path in the trellis of the code is more likely to decode to the originally transmitted bits for a systematic code.
- The bit error probability does not increase above the channel error probability. In the worst case no information on the state is known and the decoder just reports back the systematic bits as received. The bit error probability becomes “unclamped” from the channel error probability only when the Hamming distance between trellis paths becomes great enough for error correction to occur.

8.3.3 Comparison of “good” and random systematic generator matrices

A random systematic generator matrix ($R = 3/4$, $m = 9$, $\nu = 9$) realizable in observer-canonical form was created. This generator was then compared to the systematic version of the “good” code used in section 8.3.2. The results are shown in figure 8.6. These codes show

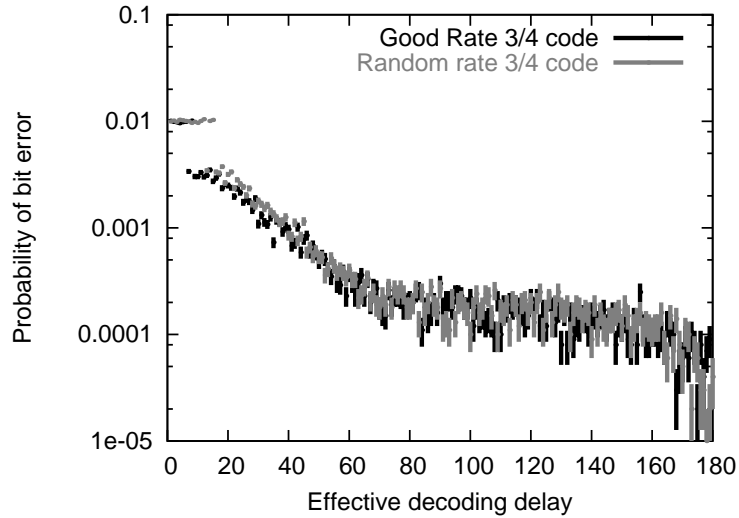


Figure 8.6: Comparison of a random code and a “good” code

similar performance; the main difference is at very short decoding delays, where the random code performs worse. In the random construction the penultimate memory element did not have any connections to it, therefore no more protection was offered to the bits at the end of a transmission.

8.3.4 Dependence on memory size

Random systematic codes with different memory orders were investigated and are shown in figure 8.7. The gradient of the ending tails is similar in each graph; the main variation between them is the level of the error floor. This variation is as expected as a more complicated code generally has a better error correcting capability. The Zigangirov upper bound on the contribution to bit error probability by finite decoding delay [121] is also shown on the graph.

The position of the transition from the error floor to the ending tail region (as shown in figure 8.4) effectively defines the decoding delay one should use to fully exploit the error correcting abilities of the code. The section of the graph from decoding delay 20 to 140 (these ranges were chosen so as to exclude the starting tail and remove the effect of the few “clamped” bits) was modelled as two sections. The error floor was taken to be a constant error probability and the ending tail a log-scale straight line. A maximum likelihood best-fit was carried out with the gradient of the ending tail line kept fixed at the gradient found for the maximum memory size point (otherwise the length of the tail was too small for the procedure to work reliably). The results with a line of best fit (with a fixed zero intercept) are shown in figure 8.8. One can therefore deduce that the decoder should have a delay of at least $7m$ to fully exploit the error-correcting ability of these codes.

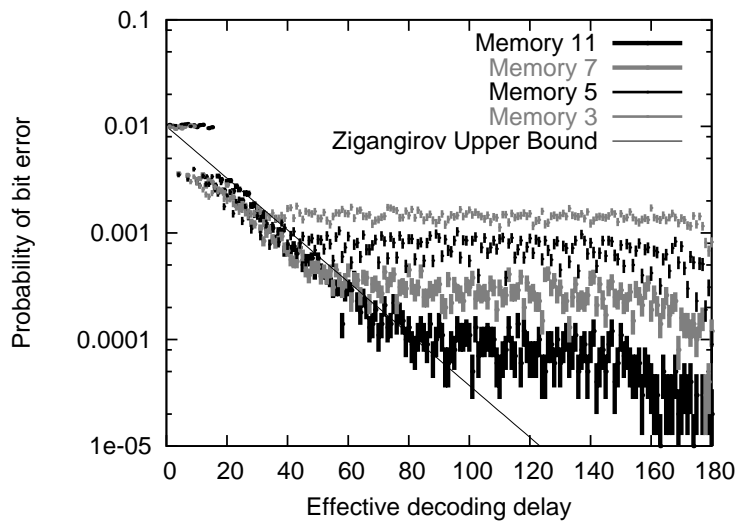


Figure 8.7: Comparison of different memory size

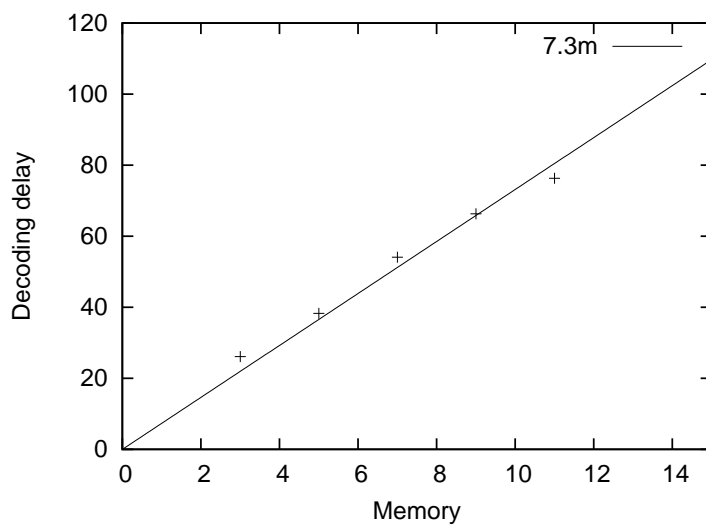


Figure 8.8: Transition from error floor to ending tail for a $R = 3/4$ convolutional code as a function of code memory m

8.4 Discussion

The Zigangirov upper bound for bit error probability against decoding delay as shown in figure 8.7 happens to be the same as the random-coding bound for bit error probability against block size. The following of this bound in the ending tail region shows that a convolutional code offers good performance as function of decoding delay; with every bit received we have similar performance to a block code stretching from the bit being decoded to the end of the block.

It would be desirable to lower the error floor further. The obvious way of doing this, by increasing the memory size, is not immediately feasible in the set-up described in this chapter; the computational complexity of the forward-backward algorithm scales exponentially with the memory size. Sequential decoding would allow the memory size to be increased further as the decoder does not visit all trellis branches. However, it is only efficient below the computational cut-off rate – a lower rate code would have to be used over the example channel above. Sequential decoding is a near maximum likelihood technique, so exact *a posteriori* probability decodings are not reached – this is a particular concern in the ending tail. Also sequential decoding can lose any computational advantage when faced with bursty channels or loss of synchronization.

8.5 Conclusion

We have shown that a convolutional code provides good performance for a low latency application until the code reaches its error floor. In the next chapter we will look at reducing the error floor by adding global structure to the local structure of a convolutional code.

CHAPTER 9

INTERSECTED LDPC AND CONVOLUTIONAL CODES

9.1 Introduction

In current communication systems it is common to use a serial concatenation of a Reed-Solomon code with a convolutional code (SCRSCC) [20, 34]. Commonly the convolutional code is decoded to a maximum likelihood codeword and this is then used as an input to a Reed-Solomon decoder. Information is not used effectively in this process since the convolutional code is returning a hard decision. Most Reed-Solomon decoders are not capable of using soft information.

In general large block codes need a complete block to be received before decoding can be attempted (this is a problem for low-speed telemetry links). To keep encoders close to linear time and allow early decoding, it is useful to keep the structure of a convolutional code combined with a high-rate block code. We can decode the convolutional code at any time to get performance as shown in chapter 8; after a complete block has been received we can decode both the codes to get better performance.

Convolutional codes with a soft-input soft-output decoder [2, 48] have been shown to perform well in concatenated coding schemes [8]. This chapter looks at replacing the Reed-Solomon component code of the SCRSCC with a low-density parity-check code [70].

Iterative decoding will be used. We will use extrinsic information transfer (EXIT) charts [103] to analyse a code's performance in terms of properties of the messages being passed during decoding.

9.2 Code Construction

To maintain a simple graph structure we look at the intersection of a low-density parity-check code with a convolutional code. We call this code an ICG code (intersected convolutional

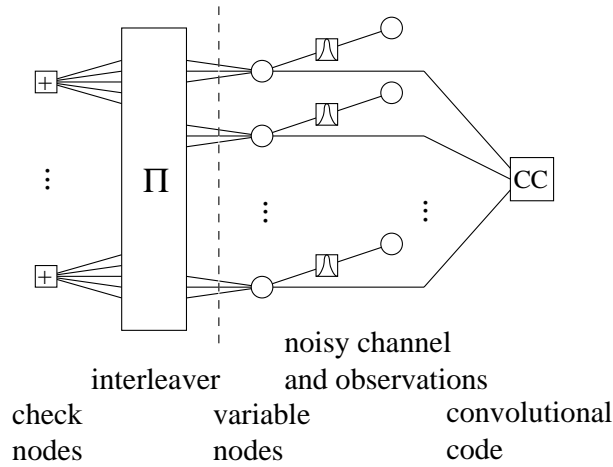


Figure 9.1: The factor graph of a ICG code, shown with a Gaussian noise channel. The point of message analysis, section 9.3, is shown with a dotted line.

Gallager code). We use the term intersection as the codewords are defined as:

$$E_{\text{ICG}} = E_{\text{conv}} \cap E_{\text{LDPC}} \quad (9.1)$$

The intersection means that the factor graph of the complete code is the factor graphs of the two component codes connected by common variable nodes. An example factor graph showing the constraints satisfied is shown in figure 9.1. Decoding is carried out by belief propagation on the factor graph. The iterative belief propagation is terminated when a tentative decoding is found that is a codeword of both the low-density parity-check code and the convolutional code (or a maximum number of iterations is reached).

The intersection can also be seen in the parity-check matrix. The parity-check matrix of the intersected code is the parity-check matrix of the component codes vertically concatenated as shown in the top panel of figure 9.2. With the low-density parity-check component kept to be high rate, the majority of the encoding can happen in linear time with the convolutional code. The final few bits are encoded by matrix multiplication. The encoder can be obtained by Gaussian elimination in the parity-check matrix of the code as shown in figure 9.2.

After Gaussian elimination has been carried out one could view the system as a convolutional outer code serially concatenated with a general linear inner code. Various results [1, 90] suggest that having a high rate inner code in a concatenated system is advantageous. The rate of the general linear inner code (R_{GLI}) increases if the low-density parity-check component code's rate is increased. Looking at the size of the pivot square one can show that

$$R_{\text{GLI}} = R_{\text{LDPC}} \left(1 + \frac{1 - R_{\text{conv}}}{R_{\text{conv}}} \right) - \frac{1 - R_{\text{conv}}}{R_{\text{conv}}} \quad (9.2)$$

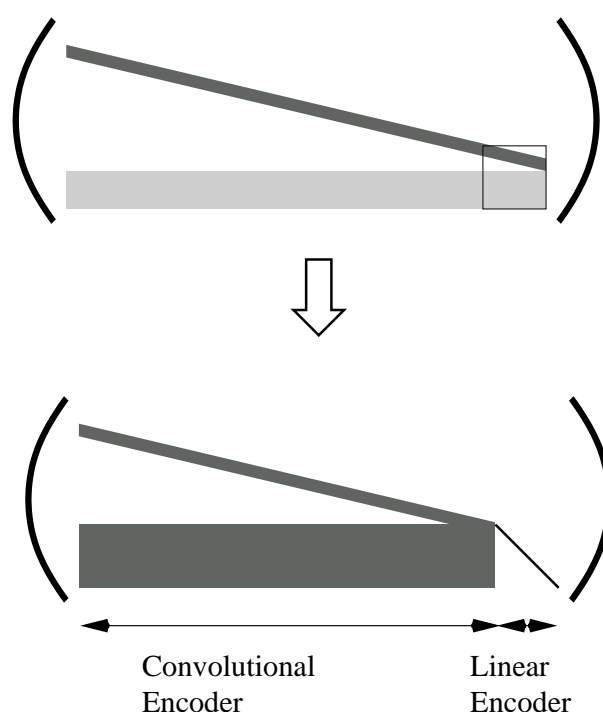


Figure 9.2: The rearrangement of the parity check matrix of an ICG code to create an encoder. In the upper matrix, the upper dark band is the area of the parity check matrix used for the convolutional code. The light rectangle at the bottom of the matrix is the low-density parity-check matrix. The square indicates the pivot area for Gaussian elimination. The lower matrix shows that most of the sparse area of the parity-check matrix has been replaced with dense entries.

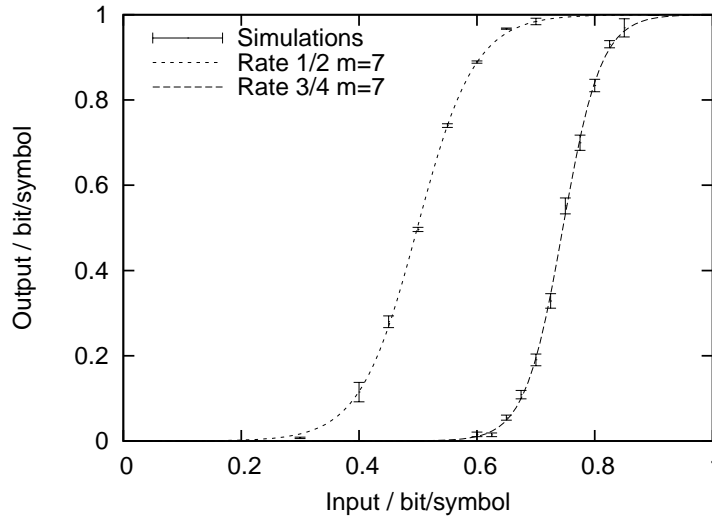


Figure 9.3: EXIT charts of $\nu = 7$, $R = 3/4$ and $\nu = 7$, $R = 1/2$ convolutional codes showing simulation results and tanh fits.

9.3 EXIT chart analysis

The behaviour of the iterative decoding can be predicted using an EXIT chart. In figure 9.1 the graph has been shown split into two loop-free sections; the left section consists of the check nodes and the right section consists of the variable nodes and the convolutional code. The transfer function of the right section depends on the channel noise level; the left section does not. An advantage of ICG codes over turbo codes [8] is that the form of the transfer functions can be altered by adjusting the degree sequences of the nodes.

Approximate forms for the transfer function of the variable nodes and check nodes were used as detailed in appendix I. The transfer function of the convolutional code was simulated and then a tanh function fitted to it. The fit was tried for a range of different convolutional codes and good fits were generally found (for $1/4 \leq R \leq 3/4$, $\nu \geq 3$). Some examples are shown in figure 9.3.

An example EXIT chart is shown in figure 9.4. In the limit of large block size a code decodes if a swath exists between the two curves. The channel noise level was altered by binary division to find the threshold where the curves just intersect.

9.4 Threshold Optimization

The degree sequence of the low-density parity-check nodes was optimized using a global optimization package (DIRect [38]) with “fractional phantom distributions” [86]. For high rate component codes, the optimization results suggested that the largest gain in performance for high rate low-density parity-check component codes could be achieved by a simple mixture of weight-0 and weight-1 columns – equivalent to having only some of the nodes of the convolu-

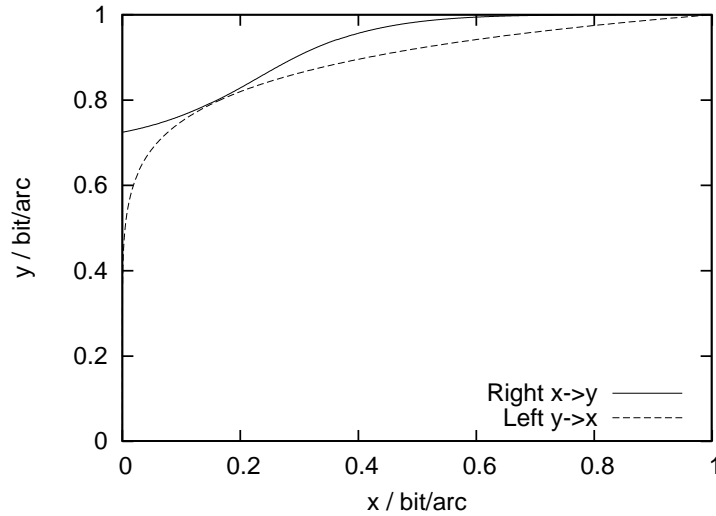


Figure 9.4: EXIT chart of an ICG code ($j = 1$, $R_{\text{LDPC}} = 0.9$, $R_{\text{conv}} = 0.75$) at threshold

tional code involved with the intersection with the low-density parity-check code. Intuitively this corresponds to the low-density parity-check code having a lower rate and being able to give better help to the bits to which it is connected. When the rates of the convolutional and low-density parity-check component codes become similar then a more complex irregular sequence with higher weight columns is preferred.

The gap between the threshold and the Shannon Limit for codes with a mixture of weight-1 and weight-0 columns is shown in figure 9.5. It can be seen that there is a minimum in these graphs.

With weight-1 columns there is local structure in the parity-check matrix as there is no overlap between each row of the parity-check matrix. Uniform interleaving of the parity checks spreads out the structure. However a burst of errors from the convolutional code can easily knock out a large number of parity checks. So we applied an S -random interleaver [26] to the parity-checks. An S -random interleaver is a random-like interleaver with the constraint that no input symbols within distance S appear within a distance of S in the output.

With no weight-0 columns this strategy works well. For a code with weight-0 columns an error floor is found. Many sequential symbols of the convolutional code can be disconnected from the low-density parity-check code. These disconnected symbols can be at the same position as a burst of errors from the convolutional code which leads to an error floor. Better results are found if the weight-0 columns are regularly spaced through the code and then an S -random interleaver applied to the other columns.

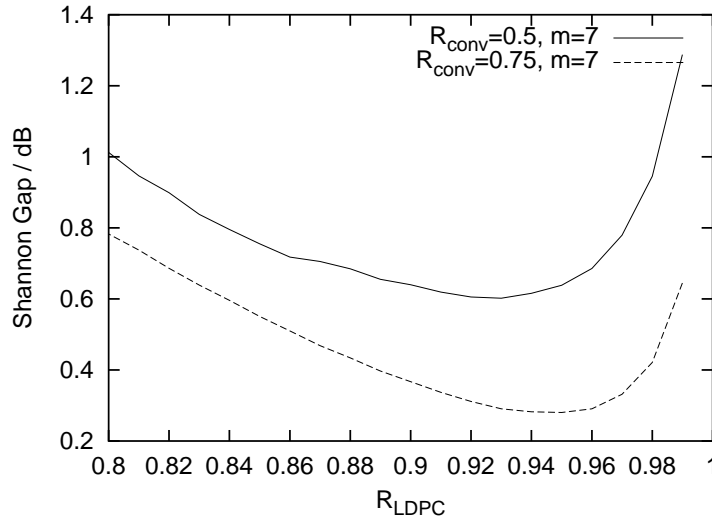


Figure 9.5: The gap from the Shannon Limit for ICG codes with two different convolutional component codes and the low-density parity-check codes made from a mixture of weight-1 and weight-0 columns.

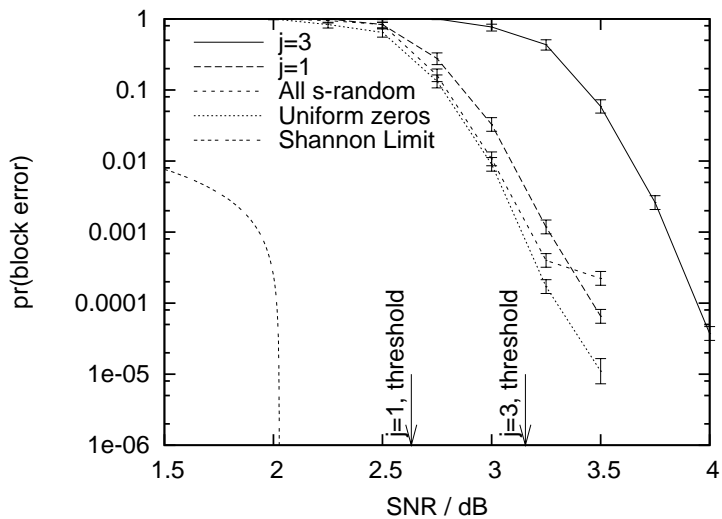
9.5 Code construction results

Ideally we would like the low-density parity-check component code rate to be close to 1 to keep the encoding complexity close to linear. For a slow telemetry application keeping the low-density parity-check component code rate high means we do not need to suspend data transmission to transmit a large number of parity bits.

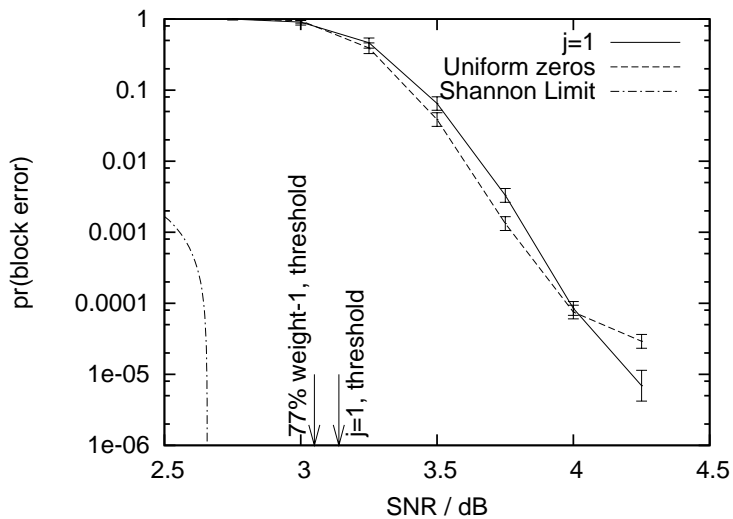
Simulations are shown with two different rates of low-density parity-check component codes in figure 9.6.

In figure 9.6(a) an $R = 0.9$ low-density parity-check component code is used. Two regular codes with $j = 1$ and $j = 3$ are shown with two irregular codes (both with 80% weight-1 columns and 20% weight-0 columns). Firstly one can see the improvement of the $j = 1$ over the $j = 3$ regular low-density parity-check component code. When used alone, $j = 3$ low-density parity-check codes are good codes, however strong codes are often not good constituent codes in an iterative decoding scheme. When the waterfall region is steep (as with a low-density parity-check code) the constituent decoder gives an “all-or-nothing” answer; this does not help iterative decoding. Also shown is the effect on the error floor of having uniformly distributed weight-0 columns and having them as part of the S -random permutation. A significant reduction in the error floor can be seen.

The parameter S of the interleaver generally satisfies $S < \sqrt{N/2}$ [26]. However to widely distribute the parity checks we need $S > \text{row weight}$. So the S -random interleaver imposes a maximum rate on R_{LDPC} . $R_{LDPC} = 0.95$ was approximately the highest rate that could be simulated with $N = 3000$. With a higher rate the S -random interleaver could not spread the parity bits widely enough. The simulation results are shown in figure 9.6(b). The irregular



(a) Two regular and two irregular $R = 0.9$ low-density parity-check codes to give $R \approx 0.65$ ICG codes



(b) One regular and one irregular $R = 0.95$ low-density parity-check code to give $R \approx 0.7$ ICG codes

Figure 9.6: A convolutional code ($N = 3000$, $R = 3/4$, $m = 7$) intersected with low-density parity-check codes

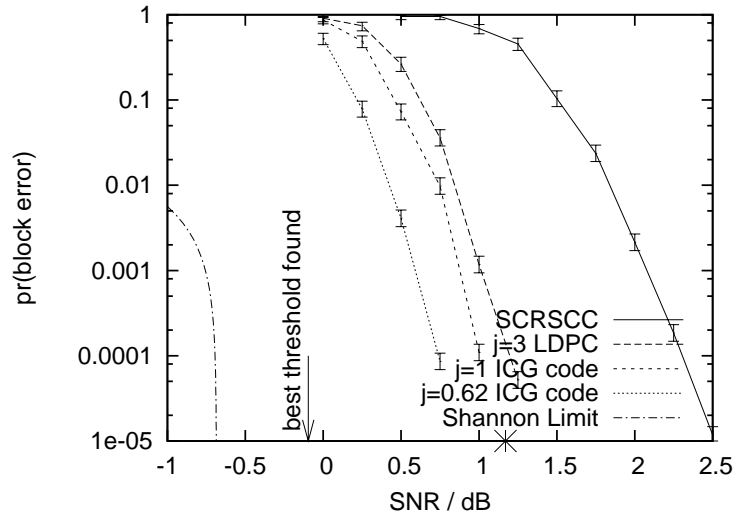


Figure 9.7: A comparison of ICG with SCRSCC and low-density parity-check codes. All codes have $N = 4092$ and $R = 0.4360$. The constituent convolutional codes are identical. One point of the *bit* error rate of a larger block size ($N = 130572$) iteratively decoded SCRSCC is shown with an asterisk [49].

code had 23% weight-0 columns and 77% weight-1 columns. It can be seen that there is an error floor with weight-0 columns despite having the weight-0 columns uniformly distributed.

9.6 Comparison with a deep-space standard

A standard for deep-space communications (for example on the NASA Voyager mission) is a rate 1/2 convolutional code serially concatenated with (255, 223) Reed-Solomon codes [20]. This was compared to two ICG codes (one regular with $j = 1$ and the other with 38% of the columns being of weight 0 and the rest of weight 1). Simulation results are shown in figure 9.7. For comparison a regular $j = 3$ low-density parity-check block code is shown. It can be seen that a gain of 1.5dB over the deep-space standard is achieved with the ICG code and that the ICG code compares favourably with a non-optimized sparse graph block code.

Recently some iterative techniques for decoding SCRSCC codes using state pinning have been introduced [49]. With multiple interleaved Reed-Solomon codes a gain of about 1dB can be found by conducting iterative decoding. This is smaller than the gain found at $N = 3000$ (equivalent to no interleaving) by using ICG codes. As N is increased the ICG code performance should further approach its threshold. A value for the *bit* error rate of an iteratively decoded SCRSCC with interleaver depth $I = 32$ ($N = 130572$) from [49] is shown in addition in figure 9.7.

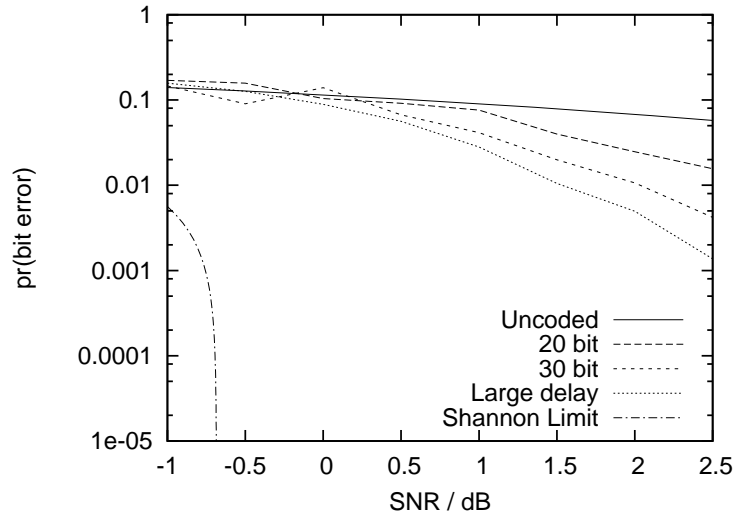


Figure 9.8: Decoding the NASA standard $R = 1/2$ convolutional code at various decoding delays. The bit error probability drops with increasing decoding delay reaching a limit indicated by the “large delay” line. The Shannon Limit is shown for $R = 0.4360$ to match the codes in figure 9.7.

9.7 Discussion

In figure 9.8 the performance of the same convolutional code as used in the ICG codes in figure 9.7 is shown at various decoding delays. For this example, at signal-to-noise ratios above 1dB the ICG code structure works well. On decoding the convolutional code alone the bit error probability drops as the decoding delay increases. Then, as seen in figure 9.7, decoding the entire block of the ICG code cleans up most remaining errors.

To get the very best waterfall performance, evidence of an error floor is seen due to the presence of weight-0 columns. It may be possible to ensure that the weight-0 columns do not match very low weight error patterns of the convolutional code to reduce this error floor.

Thresholds could be further investigated. Firstly a theoretical model for the threshold of an ICG code could be worked upon. Secondly a study of different block sizes could be carried out to find out whether the approximate threshold values are approached as the block size increases.

We have used a decoding algorithm in which the entire convolutional code has been decoded once per iteration. As convolutional codes often have bursts of errors it may be possible to speed up decoding by only redecoding the sections of the convolutional code that are in error.

9.8 Conclusion

Intersected convolutional and low-density parity-check codes have been presented and shown to compare favourably with other coding schemes whilst showing computational advantages. Approximate threshold values have been found, some less than 0.3dB from the Shannon Limit. Simulations of medium size blocks have shown code performance within 1.5dB of the Shannon Limit at a block error probability of 10^{-5} .

CHAPTER 10

CONCLUSION

10.1 Summary

We have shown that low-density parity-check codes and related codes can be successfully applied to a variety of non-standard channels.

For channels without synchronization errors, an ordinary low-density parity-check code system can be used. But if we keep the same code and allow modifications of the decoder, we may obtain a performance gain on channels with time-varying noise. In chapter 3, we looked at Gaussian noise and bit-flipping noise two-state Markov channels, and showed that iterative message passing on a graph representing the code and channel could lead to gains of as much as 1dB on the examples studied. For the Gaussian noise case, a similar gain could be obtained by inferring the channel state initially and then carrying out message passing on the graph of the code alone.

If we allow encoder and decoder modifications, we can obtain a gain for asymmetric channels (chapters 5 and 6) and allow channels with synchronization problems to be handled (chapter 7). We introduced two new codes designed for use on a multi-user asymmetric channel: Sparse-Dense Codes and Sparse LDPC Codes. The Sparse LDPC Codes showed better performance on the channels studied than Sparse-Dense codes but at a higher computational cost. On the bit insertion-deletion channel, marker codes were studied and a full iterative decoding approach experimentally shown to have better performance than other known codes.

Finally it was shown in chapters 8 and 9 that a low-density parity-check code can be used in combination with a convolutional code to create a useful telemetry system over a slow channel. A high rate low-density parity-check code is used to add global structure to a convolutional code. This gives good block code performance whilst the early decoding properties of a convolutional code are maintained.

10.2 Key Insights

Throughout this thesis “soft decisions” have been used; the messages passed are real numbers rather than discrete. Taking these messages to be beliefs allows a wide variety of algorithms

to be constructed. Furthermore a better decoding can usually be obtained if we receive soft information from the channel. For a code designer who understands the transfer characteristics of a channel it is often a simple matter to design a belief-propagation decoder which exploits that knowledge to obtain a better decoding. Care should be taken as the optimal code depends on the message passing schedule in the decoder.

EXIT charts can often be used as a code optimization tool. They allow an easy visualisation of the behaviour of a belief-propagation decoder. The technique is a Gaussian approximation to density evolution; it was seen that even with non-Gaussian inputs the code optimization results are useful. In a few cases EXIT charts are not suitable; for example, when our messages became vectors of probabilities, we instead used Monte Carlo techniques.

Random codes are good codes. To achieve a random-like code, global structure is needed; a change in one bit affects many other bits throughout the code. A convolutional code has local structure due to the finite history in the encoder. We have seen that we can add global structure to a convolutional code by intersecting it with a high-rate low-density parity-check code. This addition could be used with other codes, for example to join multiple small block codes.

Asymmetric channels seem to be largely ignored as, for the single-user binary case, symmetric signalling does not lead to a large hit in performance. We have shown that, for multi-user channels, significant gains can be obtained with a code designed to give the correct input distribution. The common assumption of the adequacy of symmetric signalling needs revision.

10.3 Future directions

Iterative message-passing algorithms which include channel state estimation lead to a useful gain in performance. However there is a cost in decoding speed which needs to be addressed. A lowering in complexity may be possible with a different message-passing schedule. Design and implementation of the final algorithm in hardware should lead to a further increase in speed.

Codes designed for asymmetric channels have received very little study. The work presented on multi-user channels suggests that further study of such codes would be profitable. Physical systems could be surveyed to obtain accurate channel models and tests carried out with asymmetric signalling. Sparse-Dense Codes are promising for fibre-optic applications as their serial-style encoder and decoder could lead to a high-speed implementation. The hardware design for such a system would be an interesting challenge. Sparse LDPC codes are currently more of theoretical interest. The experiments presented show near-capacity performance; whether Sparse LDPC codes can approach capacity with a practical decoding algorithm remains an important open question.

The capacity of the insertion-deletion channel is still unknown. We have presented codes that are close to the bounds on the capacity. If the capacity can be evaluated, we will be

better able to assess whether marker codes provide a satisfactory coding solution or whether alternative code designs should be studied.

Intersected convolutional and low-density parity-check codes have shown good performance. Only initial analysis has been conducted of the threshold and error floor of these codes and further study could be invaluable. Also optimization of the message passing schedule could be studied; it is likely an entire forward-backward pass on the trellis of the convolutional code is not needed every iteration.

Low-density parity-check codes have spent 40 years in academia; I hope this thesis contributes to the take-up of the codes in the real world.

APPENDIX A

INFORMATION THEORY

A.1 Definitions

Shannon [93] introduced the concept of the information content of the outcome of an experiment. If the experiment has an outcome randomly drawn from the ensemble X (the set of all possible outcomes with a probability $\Pr(x)$ associated with each outcome x), then the information content of outcome x is:

$$h(x) = -\log_2 \Pr(x) \tag{A.1}$$

where h is measured in bits. A logarithmic approach as observed in [51, 78] seems sensible; when an experiment is repeated the information content adds:

$$h(x, y) = -\log_2 \Pr(x, y) = -\log_2 \Pr(x) - \log_2 \Pr(y) = h(x) + h(y) \tag{A.2}$$

The entropy of an ensemble $H(X)$ is defined to be expected information content:

$$H(X) = \sum_{x \in X} \Pr(x) h(x) = - \sum_{x \in X} \Pr(x) \log_2 \Pr(x) \tag{A.3}$$

The entropy of an ensemble with two symbols (one occurring with probability p_1) has the form:

$$H_2(p_1) = -p_1 \log_2 p_1 - (1 - p_1) \log_2 (1 - p_1) \tag{A.4}$$

$H_2(\cdot)$ is called the binary entropy function.

Other types of entropy can be defined:

$$H(X, Y) = - \sum_{x \in X, y \in Y} \Pr(x, y) \log_2 \Pr(x, y) \quad (\text{A.5})$$

$$H(X|y) = - \sum_{x \in X} \Pr(x|y) \log_2 \Pr(x|y) \quad (\text{A.6})$$

$$\begin{aligned} H(X|Y) &= \sum_{y \in Y} \Pr(y) H(X|y) \\ &= - \sum_{x \in X, y \in Y} \Pr(x, y) \log_2 \Pr(x|y) \end{aligned} \quad (\text{A.7})$$

The final important definition is that of the mutual information between two ensembles:

$$I(X; Y) = I(Y; X) = H(X) - H(X|Y) \quad (\text{A.8})$$

This measures the reduction in uncertainty of x once y is learnt (or vice versa). So it measures the amount of information that y conveys about x .

A.2 Source coding

Shannon's source coding theorem [93] tell us the length of the shortest string in which some data can be expressed. If one has N draws from an ensemble X with entropy $H(X)$ then as $N \rightarrow \infty$ one can describe the outcomes in $NH(X)$ bits. If fewer bits are used an inaccurate description will be obtained.

When the model of the source is known perfectly, arithmetic coding [76] is able to efficiently reach within 2 bits of the bound. Huffman codes [55] can achieve within 1 bit of the bound (and often less [41]) but are computationally infeasible with large N .

The source coding theorem describes a compression process. For instance, it has been found that English has an entropy of about 1 bit per character [94]. If English text is stored in the standard representation used in a computer then it takes 7 bits per character [92]. The source coding theorem says that we ought to be able to store a long string using 1 bit per character, achieving compression by a factor of 7.

A.3 Channel coding

Shannon also presented his channel coding theorem in [93]. Unlike the source coding theorem which deals with removal of redundancy this theorem specifies how much redundancy a source needs to allow reliable communication over a noisy channel (with stationary ergodic noise).

We will present the case of a discrete memoryless channel (DMC). A DMC defines a probabilistic map of elements from an input ensemble X to an output ensemble Y . The mapping does not vary with time. For examples of basic channels see section B.1. The

capacity of such a channel is:

$$C = \max_{\{\Pr(x)\}} I(X;Y) \quad (\text{A.9})$$

In other words the input distribution $\{\Pr(x)\}$ is chosen to maximize the mutual information between X and Y .

We can construct a code E , a set of allowed transmission sequences. Each sequence is of length N . The code rate $R = \frac{\log_2 |E|}{N}$. R specifies the number of source bits encoded per output symbol. For a binary (n, k) code which encodes k bits into n bits, $R = k/n$. The channel coding theorem says that as $N \rightarrow \infty$ we can communicate with an arbitrarily small probability of error over a DMC if $R < C$. If $R > C$ then reliable communication is not possible.

For more details on Information Theory, [71] is recommended.

APPENDIX B

SIMULATIONS

B.1 Basic channels

There are four well-known noisy communication channels we will use. Each of these channels defines a probabilistic map from an input set to an output set. They are all memoryless; each time the probabilistic map is conducted it is done independently and identically.

For more information on any of the following channels the reader is referred to [71].

B.1.1 Binary erasure channel

The binary erasure channel (BEC) has a binary input $\{0, 1\}$ and a ternary output $\{0, 1, ?\}$. Each input bit is either copied identically to the output with probability $1 - p_e$ or changed to ‘?’ with probability p_e , figure B.1(a).

The capacity for this channel is $1 - p_e$.

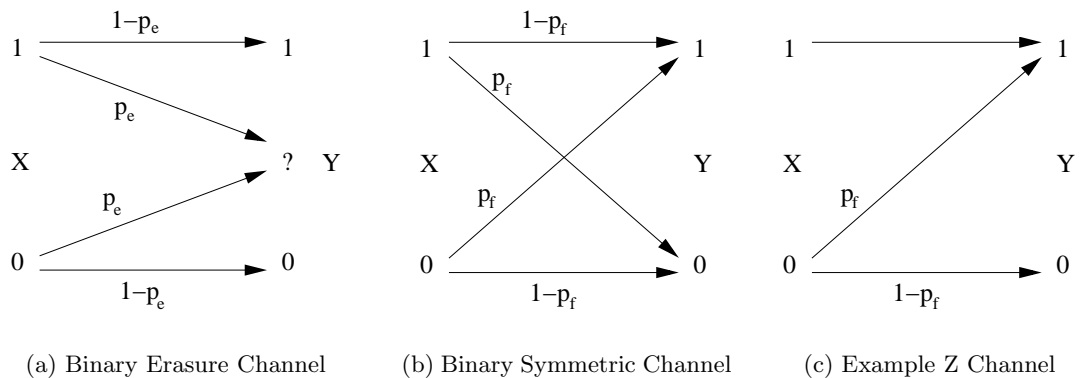


Figure B.1: Three discrete memoryless channels

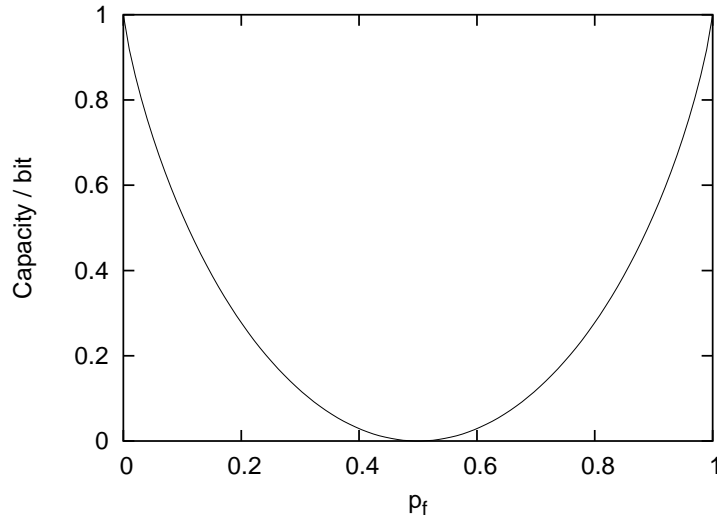


Figure B.2: The capacity of the binary symmetric channel

B.1.2 Binary symmetric channel

A binary symmetric channel (BSC) has a binary input $\{0, 1\}$ and output $\{0, 1\}$. Each bit input is either copied identically to the output with probability $1 - p_f$ or flipped with probability p_f , figure B.1(b).

The capacity for a BSC is $1 - H_2(p_f)$ (where $H_2(\cdot)$ is the binary entropy function as defined in appendix A). The capacity is plotted in figure B.2.

B.1.3 Z channel

The Z channel also has a binary input and output. One of the input set is transmitted reliably and the other flipped with probability p_f . Figure B.1(c) shows the case with 1 transmitted reliably and 0 subject to corruption. The channel is not symmetric and it is optimal to use an input distribution that is asymmetrical.

B.1.4 Additive White Gaussian Noise

The additive white Gaussian noise (AWGN) channel has a real input X and a real output Y . Y has a Gaussian distribution:

$$\Pr(y|x) \propto \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right) \quad (\text{B.1})$$

Is it optimal to use Gaussian signalling (where X has a Gaussian distribution) but this is hard to achieve in practice. Digital modulation is common and in this thesis we will assume binary signalling where $X = \{-1, 1\}$. For low code rates the difference in capacity between binary and Gaussian signalling is small [37].

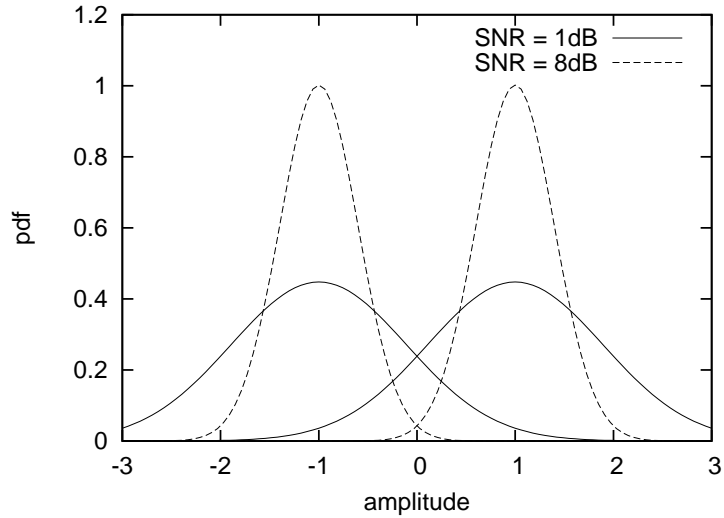


Figure B.3: Example received Gaussian amplitudes for two different noise levels with bipolar signalling

It is common to quote a signal-to-noise ratio (SNR) rather than σ . We will use the following definition of SNR:

$$\text{SNR} / \text{dB} \equiv 10 \log_{10}(1/\sigma^2) \quad (\text{B.2})$$

In many contexts a statistic of E_b/N_0 is quoted where E_b is the energy per data bit and N_0 is the noise spectral density. This gives:

$$E_b/N_0 / \text{dB} \equiv 10 \log_{10}(1/(2R\sigma^2)) = \text{SNR} / \text{dB} + 10 \log_{10}(1/2R) \quad (\text{B.3})$$

Examples of the resultant amplitude distributions with Gaussian noise are shown in figure B.3. In simulations, samples from a Gaussian distribution were obtained using the Box-Muller transform technique [12].

The capacity of the AWGN channel does not have a simple analytical form. When presented the capacity has been calculated using the Monte-Carlo technique of Frey [37].

Many BSCs are derived from thresholding the output of an AWGN (or similar real-output channel) channel at 0. The electronics for a thresholding receiver can be simpler than a real-output receiver – however this leads to approximately a 2dB loss in performance [57].

B.2 Simulation of one block

To test the performance of a code, the transmission of blocks over the chosen channel was simulated. In general a random data input sequence (\mathbf{s}) was chosen and then encoded using the code's encoder. The resultant bits (\mathbf{x}) were corrupted randomly by the chosen channel and then fed to the code's decoder. The final decoder bits ($\hat{\mathbf{x}}$) were compared to \mathbf{x} . The number of bit differences were reported as the number of bit errors n_b . If n_b was greater than

zero a block error was declared.

In some cases \mathbf{s} and \mathbf{x} were set to zero. For a linear code and linear channel all source blocks will behave the same and therefore computational effort can be saved. In this case, tests were done with a random \mathbf{s} first to check for any systematic errors in the simulation.

B.3 Calculation of error probabilities

Many blocks n_{sim} were simulated at a particular noise level and a total number of bit errors n_{bit} and block errors n_{block} obtained. Best estimates of bit and block error probabilities could then be obtained:

$$\Pr(\text{block error}) = \frac{n_{\text{block}}}{n_{\text{sim}}} \quad (\text{B.4})$$

$$\Pr(\text{bit error}) = \frac{n_{\text{bit}}}{N n_{\text{sim}}} \quad (\text{B.5})$$

Error bars on the block error probability were obtained by using the feature that in logit space, $l = \log_e \frac{p}{1-p}$, the posterior distribution is generally bell-shaped and hence a Laplace approximation can work well [71]. Logit space is also a sensible space in which to have a uniform prior as large block code simulations often lead to probabilities of error roughly uniformly distributed in logit space.

Abbreviating $n_{\text{sim}} = n$, $n_{\text{block}} = r$ and $\Pr(\text{block error}) = p$, we have for the likelihood:

$$\Pr(n, r|p) = p^r (1-p)^{n-r} \quad (\text{B.6})$$

With an improper uniform prior over logit space we then get the following posterior:

$$\Pr(l|n, r) \propto \frac{(\exp(l))^{n-r}}{(1 + \exp(l))^n} \quad (\text{B.7})$$

Differentiating the log-likelihood allows one to find the peak in the posterior at:

$$\exp(-\hat{l}) = \frac{r}{n-r} \quad (\text{B.8})$$

When $r \neq 0$ and $n \neq r$, we find the second differential of the log-likelihood at the peak. Using the Laplace approximation we get:

$$\sigma_l^2 = \frac{n}{n-r} \quad (\text{B.9})$$

We can then give 1- σ error bars:

$$p_{\pm} = \left(1 + \frac{n-r}{r} \exp \mp \sqrt{\frac{n}{r(n-r)}} \right)^{-1} \quad (\text{B.10})$$

When $r = 0$ and $r = n$, we get $\hat{l} = \pm\infty$ which requires special treatment due to the improper prior. We will look at where the probability falls by a factor of $\exp(-1/2)$ from the

peak (the $1\text{-}\sigma$ point on a Gaussian distribution). For $r = 0$ we get:

$$p_+ = 1 - \exp\left(-\frac{1}{2n}\right) \quad (\text{B.11})$$

$$p_- = 0 \quad (\text{B.12})$$

and for $r = n$:

$$p_+ = 1 \quad (\text{B.13})$$

$$p_- = \exp\left(-\frac{1}{2n}\right) \quad (\text{B.14})$$

In [70] $2\text{-}\sigma$ error bars are calculated in log space. The $1\text{-}\sigma$ error bars here agree well in form except at large p where the log space approximation is not expected to be good.

In general at each noise level a code was simulated until 20 block errors had occurred.

APPENDIX C

FINITE FIELD ARITHMETIC

A field (F) is a set over which the binary operations $+$ and \times are defined and the operations satisfy the following constraints:

- F forms a commutative group over $+$ with identity 0
- $F \setminus 0$ (F without the 0 element) forms a group over \times
- \times is distributive over $+$

The well known infinite sets of all real numbers, all rational numbers and all integers each form fields. In coding we commonly look at finite fields (where F has a finite number of elements).

The basic finite field is the field consisting of integers modulo a prime number, p . The field is called the Galois Field of size p , $\text{GF}(p)$. Most commonly used is $\text{GF}(2)$:

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} \times & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array} \qquad (\text{C.1})$$

The only other finite fields are extension fields of size p^q where p is prime and q an integer. These are not explicitly used in this thesis.

For more details on finite fields see [64].

APPENDIX D

LINEAR BLOCK CODES

D.1 Definition

The codewords of a linear block code are defined as:

$$E = \{\mathbf{x} : \mathbf{H}\mathbf{x} = 0\} \tag{D.1}$$

\mathbf{x} and \mathbf{H} are a vector and a matrix respectively over the same finite field. The field in this thesis will in general be GF(2) (so $\mathbf{H}\mathbf{x} = 0 \pmod{2}$) however larger finite fields will be used in chapter 6 and appendix H.

D.2 Properties

The “bat caves” (see section 1.4) around every codeword in a linear code are equivalent:

$$\mathbf{H}(\mathbf{x}_1 + \epsilon) = \mathbf{H}\epsilon = \mathbf{H}(\mathbf{x}_2 + \epsilon) \tag{D.2}$$

As the all zero sequence is a codeword, the terms “weight” and “distance” are often used interchangeably for a linear code. The minimum codeword weight (excluding the all-zero codeword) is the minimum distance d_{\min} .

The parity-check matrix makes it easy to check whether a received block is a codeword. If the received string has noise added to it: $\mathbf{r} = \mathbf{t} + \mathbf{n}$ we can calculate the syndrome $\mathbf{z} = \mathbf{H}\mathbf{r} = \mathbf{H}\mathbf{n}$ as $\mathbf{H}\mathbf{t} = 0$. If $\mathbf{z} = 0$ then \mathbf{r} is a codeword. Otherwise the best estimate of the noise for a linear channel can be calculated knowing the syndrome alone.

It has been shown that one can achieve the capacity of a symmetric channel with a random linear block code [32].

D.3 Generator Matrix

We need a way to map user source bits to a codeword block. For a linear code one typically uses a “systematic” generator that adds parity bits after the source bits to form the codeword.

A first stage to obtain this generator is to use row operations to rearrange the parity-check matrix into a form similar to row-reduced echelon form (this is equivalent to conducting Gaussian elimination on $\mathbf{H}\mathbf{x} = 0$):

$$\mathbf{H}_{\text{enc}} = (\mathbf{P} \ \mathbf{I}_M) \quad (\text{D.3})$$

so the $M \times N$ parity check matrix is now expressed by a $M \times (N - M)$ matrix (\mathbf{P}) and a $M \times M$ identity matrix. This is not always possible and sometimes the columns might need to be rearranged to find a suitable form. If the columns are reordered a new code is defined with the same distance properties as the original code (it is a permutation of the original code). With \mathbf{H}_{enc} the last M values of a codeword can be deduced from the first $N - M$ values. We have therefore found a way of generating codewords.

Typically the operation of finding the parity bits is written as a matrix multiplication:

$$\mathbf{x} = \mathbf{G}^T \mathbf{s} \quad (\text{D.4})$$

where \mathbf{G} is the “generator matrix” defined as:

$$\mathbf{G}^T = \begin{pmatrix} \mathbf{I}_{N-M} \\ -\mathbf{P} \end{pmatrix} \quad (\text{D.5})$$

For binary linear block codes the minus sign in the above equation can be ignored.

For low-density parity-check codes this rearrangement will lead to a dense \mathbf{P} matrix. An alternative rearrangement can be carried which will often lead to a sparse generator matrix [88].

APPENDIX E

CONVOLUTIONAL CODES

Convolutional codes, as used in chapters 8 and 9, are described in terms of several parameters. We will follow the notation and examples of [57]. The first parameter is the rate, R . It is traditionally written in a fractional form with the number of input symbols to the sequential circuit over the number of output symbols; an $R = 2/4$ code hence has a different form to an $R = 1/2$ code. The other generally cited parameters are the memory order, m , and the total encoder memory, ν (called the overall constraint length in [57]). If each input has ν_i memory elements (for example in figure 8.2 on page 64, $\nu_1 = 2$), we then define $m = \max_i \nu_i$ and $\nu = \sum_i \nu_i$.

Instead of using a sequential circuit form for describing convolutional codes, we can use a matrix representation of the system under a D -transformation (where operations are carried out modulo 2). Then $\mathbf{v}' = \mathbf{u}'\mathbf{G}$, where \mathbf{u}' and \mathbf{v}' are the vectors representing the D -transforms of the input time sequences \mathbf{u}_t and output time sequences \mathbf{v}_t respectively. For the example in figure 8.2:

$$\mathbf{G} = \begin{pmatrix} 1 + D^2 & 1 + D + D^2 \end{pmatrix} \quad (\text{E.1})$$

Convolutional codes can be catastrophic: a finite number of channel corruptions can lead to an infinite sequence of errors after decoding. To avoid this situation, the convolutional encoder needs to be chosen appropriately. Systematic convolutional codes (codes where the input stream appears in the clear in the output stream) are known to be non-catastrophic.

The codes so far described have not necessarily been systematic. It is known that if you create a systematic generator matrix from a non-systematic generator matrix by replacing the left hand side of the matrix with the identity matrix, a bad code is produced [57]. Alternatively we can create a systematic matrix from a non-systematic generator matrix, \mathbf{G} , by multiplying

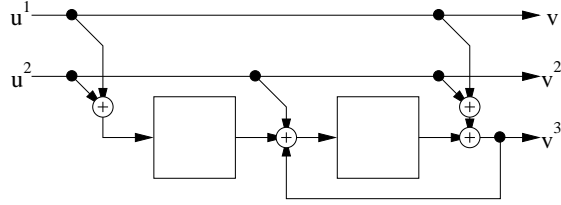


Figure E.1: Observer-canonical representation of the generator matrix shown in equation E.5

by the inverse of a square-matrix component, \mathbf{T} , for example:

$$\mathbf{G} = \begin{pmatrix} 1 + D^2 & D + D^2 & 1 + D \\ D^3 & D^3 + 1 & D^2 \end{pmatrix} \quad (\text{E.2})$$

$$\mathbf{T} = \begin{pmatrix} 1 + D^2 & D + D^2 \\ D^3 & D^3 + 1 \end{pmatrix} \quad (\text{E.3})$$

$$\mathbf{T}^{-1}\mathbf{G} = \begin{pmatrix} 1 & 0 & \frac{1+D}{1+D^2+D^3+D^4} \\ 0 & 1 & \frac{D^2+D^3}{1+D^2+D^3+D^4} \end{pmatrix} \quad (\text{E.4})$$

If we use $\mathbf{T}^{-1}\mathbf{G}$ as our new generator matrix it forms the same code as \mathbf{G} ; \mathbf{T}^{-1} only defines a convolutional scrambling of the input bits. If \mathbf{T} is selected so that its determinant is in the form $1 + f(D)$, then we can realise it in a sequential circuit form with the denominator corresponding to feedback in the delay line.

It is important to be able to represent a convolutional code as a sequential circuit with a minimal set of memory elements. The contents of the memories define the state; if we can minimize the number of memories we can reduce the trellis size and hence the complexity of decoding. In general for systematic codes this is the hard problem of finite-state-machine minimization; however for $R = n/(n + 1)$ systematic codes the problem becomes trivial. We can use the distributive and associative properties of the finite field to reverse the order of the delay and addition properties to come up with another simple representation called observer-canonical form. For example we can represent the systematic encoder

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & \frac{1+D^2}{1+D} \\ 0 & 1 & \frac{1+D+D^2}{1+D} \end{pmatrix} \quad (\text{E.5})$$

using two memory elements rather than four as we would have in controller-canonical form. This form is shown in figure E.1. The memory order m of such a matrix is the same as the total encoder memory ν of the minimal non-systematic matrix.

APPENDIX F

FACTOR GRAPHS

Factor graphs [37] show the decomposition of a complex joint distribution into simple factors. For example:

$$\Pr(x_1, x_2, x_3, x_4) = \frac{f_1(x_1)f_2(x_2)f_3(x_3)f_4(x_4)f_5(x_1, x_2, x_3)f_6(x_3, x_4)}{Z} \quad (\text{F.1})$$

where Z is a normalizing constant that is not necessarily known. The factorization can be illustrated in the manner shown in figure F.1. Each variable and factor has a node and each factor node is joined to the variable nodes on which the factor depends.

In the context of linear codes [102], each of the variables is from the finite field over which the code is defined and corresponds to a column of the parity-check matrix. Each of the rows describes a factor which would be a constraint (f_5 and f_6 in this case). A constraint is an indicator $\{0, 1\}$ and says whether that parity-check is satisfied. In the example above, this would for example be $f_6(x_3, x_4) = 1$ if $x_3 + x_4 = 0$ and $f_6(x_3, x_4) = 0$ otherwise. A codeword has all the constraints satisfied. Each factor of only one variable is a prior which generally includes the receiver symbol likelihood.

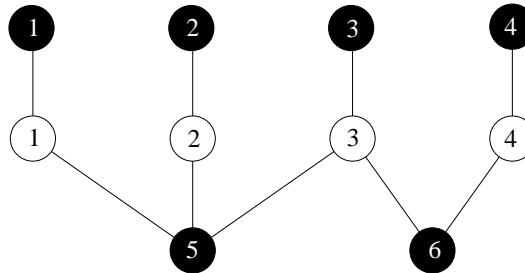


Figure F.1: A factor graph formed from equation F.1. The factors are shown as filled circles and the variables as empty circles.

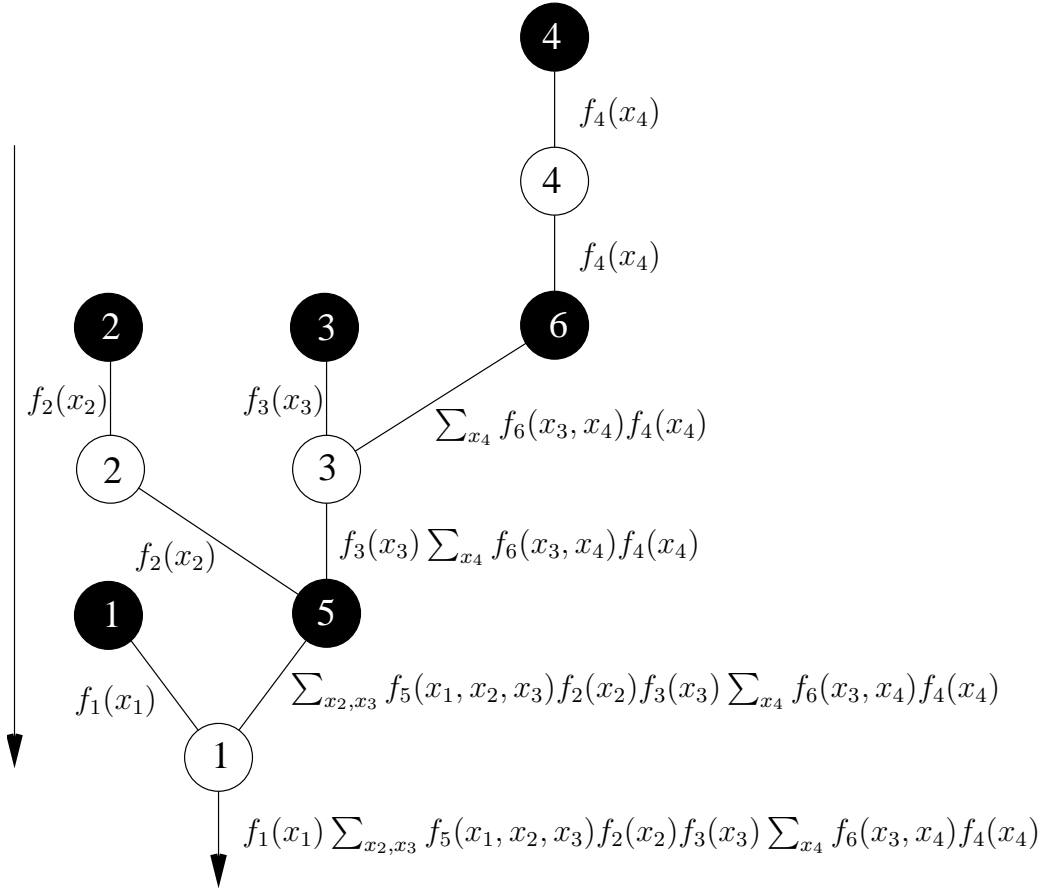


Figure F.2: Message evaluation in a tree. Messages are shown being passed from top to bottom.

The decoding problem is then to marginalise out the other variables, for example:

$$\widehat{x}_1 = \arg \max_{x_1} \Pr(x_1) = \arg \max_{x_1} \sum_{x_2, x_3, x_4} \Pr(x_1, x_2, x_3, x_4) \quad (\text{F.2})$$

We could calculate the sum in equation F.2 directly, but this can be computationally complex as the sum is over all combinations of all variables. For a code this would involve $\mathcal{O}(e^N)$ terms in the sum.

Alternatively, the summation can be split into smaller steps:

$$\Pr(x_1) = f_1(x_1) \sum_{x_2, x_3} f_5(x_1, x_2, x_3) f_2(x_2) f_3(x_3) \sum_{x_4} f_6(x_3, x_4) f_4(x_4) \quad (\text{F.3})$$

This evaluation of this equation can be implemented using computation local to the nodes and message-passing on a tree. A tree is shown in figure F.2 which topologically is identical to figure F.1. We will look at passing messages from top to bottom.

We set the check node message passing equation to be:

$$R_i^a = \sum_{\{x_k\}} f_i(\dots) \prod_j Q_j^{x_j} \quad (\text{F.4})$$

where the sum is over the variables connected on the incoming links and the product is over the incoming connected symbol nodes. We set the variable node message passing equation to be:

$$Q_j^a = \prod_i R_i^a \quad (\text{F.5})$$

where the product is over the incoming connected factor nodes. By expanding out the messages (as shown in figure F.2) we reach the same formula as equation F.3. This algorithm is called the sum-product algorithm [113].

We will normalize the messages used in decoding. For more details on the sum-product algorithm and normalization please see [71].

The graph in this example has no cycles (called a loop-free graph or a tree). In a cycle-free graph, the message update equations are exact [37]. If a graph has cycles then one can carry out the same message update equations but the update equations are approximate. The update equations assume the incoming messages are independent; this is only generally the case for a tree.

Yedidia [118] showed that the fixed points of belief propagation on a graph with cycles correspond to stationary points of the Bethe approximation to the free energy of a factor graph. This gives support to using belief propagation on graphs with cycles. Yedidia also presented algorithms that minimize better approximations to the free energy, however these have higher computational complexity.

APPENDIX G

THE HAT PROBLEM

G.1 Introduction

Todd Ebert presented the following puzzle with a solution [30]:

Each player on a team of seven is randomly and independently assigned a colored hat (either red or blue) to wear, and then tries to guess the color of his hat by viewing the hats of the other teammates. No communication is allowed except for a strategy session before the game begins. [Each player may either guess red, guess blue or pass.] The team wins a prize if at least one player guesses and all the guesses are correct. The team loses if no one guesses or some player guesses incorrectly. What strategy should the team adopt to maximize their chance of winning?

No-one knows their own hat colour, so if a person guesses they have a 50% chance of guessing correctly. One might think that the best strategy is to nominate a single person to guess the colour of their hat and tell everyone else to pass. The team would have a 50% chance of winning. But they can do better because the rules are not symmetric; one wrong answer can rule out any number of right answers. One can spread out all the right answers and clump the wrong ones together.

It is helpful to consider the case of how a team of three might play. The optimal strategy is for a player to pass unless they see two hats of the same colour worn by the other team members. In this event they say the opposite colour to what they see. The outcomes are listed in table G.1. It can be seen that each player's guesses are correct half the time but the strategy is such that everyone is wrong together and right separately. The odds of winning are $3/4$.

One can plot the situation on cube, figure G.1. We have defined "strategy points" where everyone is wrong. Assuming we're not close to another strategy point, then at a distance of one from the strategy point (along the edges of the cube) one person is right and everyone else passes. One hat colour has changed. The person whose hat colour has changed sees the same

Hats worn			Guesses			Win?
R	R	R	B	B	B	no
R	R	B			B	yes
R	B	R		B		yes
R	B	B	R			yes
B	R	R	B			yes
B	R	B		R		yes
B	R	R	B			yes
B	R	B		R		yes
B	B	R			R	yes
B	B	B	R	R	R	no

Table G.1: The outcome of 3-players playing an optimal strategy with the hat game

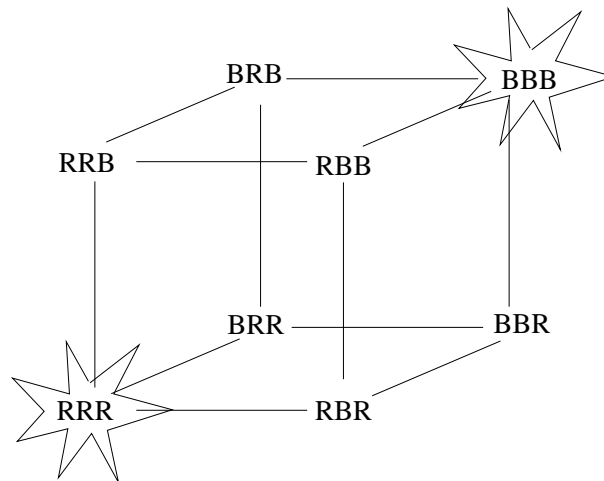


Figure G.1: The possible states of the three-person hat problem. The states are arranged on the corners of a cube with an edge between states where exactly one hat colour is different. The “strategy points” have been marked with a star.

state as at a strategy point and is now right; everyone else does not recognise the situation and passes.

To be an optimal strategy every point in space needs to be either a strategy point or an a distance of one from a single strategy point. Therefore all strategy points need to be at least a distance three from each other.

So to solve the 7-player case we need to design a set of strategy points on a seven-dimensional cube such that the points are all at least a distance of three from each other and every other point is within a distance of one of the strategy point. If we map this “hat space” to Hamming space the problem can be seen as the specifications for an error-correcting code.

G.2 The code

One of the first family of error-correcting codes presented were Hamming codes [50]. A Hamming code is created by first choosing the number of rows $M \geq 2$ in the parity-check matrix. The columns are then filled with all the distinct M -tuples apart from the the all-zero tuple. For example for $M = 3$ we get the Hamming (7,4) code:

$$\mathbf{H}_{(7,4)} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad (\text{G.1})$$

One can easily prove two useful properties. Firstly $d_{\min} = 3$. To find a minimum distance codeword from the all-zero codeword we need to find the smallest set of columns of the parity-check matrix such that their vector sum is zero. We can see there are no sets of size 1 (as the all-zero column does not exist) or of size 2 (as there are no identical columns). However combinations of three columns can be found; in the example above columns $\{1, 2, 3\}$ for instance.

Secondly we can show the code is “perfect”. A perfect code has the Hamming spheres of radius $\lfloor \frac{d_{\min}-1}{2} \rfloor$ completely filling the space. In this case we are interested in spheres of radius 1. We can start by working out how many codewords there are. There are M independent equations with N unknowns. Therefore there are 2^{N-M} solutions. Each sphere takes up $N + 1$ points in space (the central point and 1 point off in each dimension). The total space filled by spheres of radius 1 is $(N + 1)2^{N-M} = 2^N$. All space is filled. The Hamming code is therefore a perfect code.

As all space is filled with the spheres, all points in space are either codewords or within a distance of 1 of a codeword. If we map the colours red and blue to $\{0, 1\}$ then the codewords of a (7,4) Hamming code are located at the strategy points needed to solve the 7-player hat problem. The probability of winning is 7/8.

G.3 Experiment

The Hamming code may give the best answer in theory, but can non-mathematicians be trained to play with this strategy? To get 7 people to recognise Hamming codewords by working out the parity-checks is liable to lead to mistakes. We can simplify the problem by using the cyclic form of a Hamming code [65]. To be cyclic, the columns of the parity-check

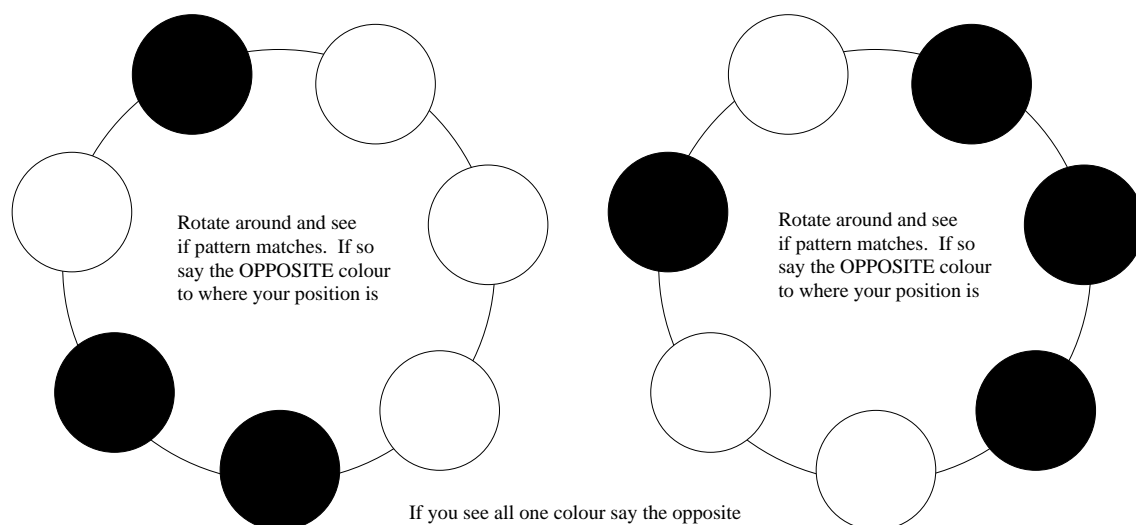


Figure G.2: A black and white version of the rule sheet issued. The filled circles were red and unfilled circles were blue. Players were told to stand facing each other in a circle and try and match the patterns. If no match could be found they were to pass.

matrix are arranged so that the rows are cyclic shifts of each other:

$$\mathbf{H}_{\text{cyclic}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (\text{G.2})$$

Four extra redundant rows have been added below (eg row 4 is the sum of rows 1 and 2). The symmetry of all the columns means that any cyclic shift of a codeword is another codeword. For the parity-check matrix above all the codewords are formed from cyclic shifts of members of the following set:

$$\{0000000, 1011000, 0100111, 1111111\} \quad (\text{G.3})$$

We therefore reduce the problem of recognising 16 patterns to recognising cyclic shifts of four patterns.

Within a ten minute talk on the topic, seven undergraduate students were given sheets as figure G.2 and told how to use them. They were then given randomly chosen coloured hats. The group then all correctly passed apart from one member who said what colour hat he was wearing.

G.4 Postscript

A Hamming code can be used to solve this problem for groups of size $(2^n - 1)$ where n is an integer. For instance for a group of size 127 the chance of winning is higher than 99%. For other size groups it would be possible to nominate the largest possible subgroup of size $(2^n - 1)$ to play the above strategy and the rest to pass. Other strategies and games with more than two hat colours have been studied in [62].

APPENDIX H

COMPLEXITY OF LDPC BELIEF PROPAGATION DECODING

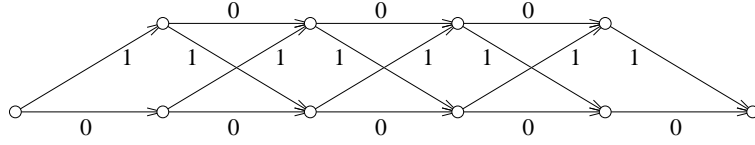
In this appendix the complexity of belief propagation decoding for low-density parity-check codes will be studied. We will look at the complexity per iteration of two different algorithms. Firstly an algorithm based on the equations in chapter 2 and secondly an equivalent algorithm based on a Fourier (or dual) representation. An alternate equivalent algorithm involves passing log-space messages [19] but this can be slow on a computer due to the use of hyperbolic functions. Lower complexity algorithms which pass discrete messages can be used with a drop in code performance [39, 87].

We will express the complexity in terms of parameters of the $M \times N$ parity-check matrix \mathbf{H} . The graph formed has M check nodes and N symbol nodes. We will look at regular codes with column and row weights of j and k respectively. Each check node is connected to k symbol nodes and each symbol node is connected to j check nodes. In each case we will calculate the complexity for each node's message evaluations first so the results could be carried over to irregular codes.

H.1 Complexity of the GF(2) Non-Fourier algorithm

H.1.1 Check Node

We will first evaluate the complexity of the computation in a single check node y . The simplest known algorithm for the computation is the forward-backward algorithm [2] on a trellis like figure H.1. Each possible path from the beginning to the end of the trellis represents a codeword; the power of the algorithm comes from the trellis having fewer branches than paths.


 Figure H.1: A trellis for a single parity-check over $\text{GF}(2)$

Forward and backward probabilities, α and β respectively, will be defined as follows:

$$\alpha_{yz}(a) = \Pr \left(\sum_{i \leq z} \mathbf{H}_{yi} x_i = a \right) \quad (\text{H.1})$$

$$\beta_{yz}(a) = \Pr \left(\sum_{i > z} \mathbf{H}_{yi} x_i = a \right) \quad (\text{H.2})$$

a can be seen as representing the level on the trellis in this case. These probabilities can be calculated recursively on the trellis:

$$\alpha_{yz}(1) = Q_{yz}^0 \alpha_{y,z-1}(1) + Q_{yz}^1 \alpha_{y,z-1}(0) \quad (\text{H.3})$$

$$\beta_{ij}(1) = Q_{i,j+1}^0 \beta_{i,j+1}(1) + Q_{i,j+1}^1 \beta_{i,j+1}(0) \quad (\text{H.4})$$

$\alpha_{yz}(0)$ and $\beta_{yz}(0)$ do not need to be evaluated explicitly due to the normalization condition. We can express equations H.3 and H.4 as:

$$\alpha_{yz}(1) = (Q_{yz}^0 - Q_{yz}^1) \alpha_{y,z-1}(1) + Q_{yz}^1 \quad (\text{H.5})$$

$$\beta_{yz}(1) = (Q_{y,z+1}^0 - Q_{y,z+1}^1) \beta_{y,z+1}(1) + Q_{y,z+1}^1 \quad (\text{H.6})$$

The boundary conditions are:

$$\alpha_{y0}(0) = \beta_{yk}(0) = 1 \quad (\text{H.7})$$

$$\alpha_{y0}(1) = \beta_{yk}(1) = 0 \quad (\text{H.8})$$

We need the α and β values on $(k-2)$ non-trivial states, this takes $2(k-2)$ multiplications and $4(k-2)$ additions in total.

We can then obtain the R_{yz} values by multiplication and addition:

$$R_{yz}^1 = \alpha_{y,z-1}(1) \beta_{yz}(0) + \alpha_{y,z-1}(0) \beta_{yz}(1) \quad (\text{H.9})$$

$$= \alpha_{y,z-1}(1) + \beta_{yz}(1) - 2\alpha_{y,z-1}(1) \beta_{yz}(1) \quad (\text{H.10})$$

$$R_{yz}^0 = 1 - R_{yz}^1 \quad (\text{H.11})$$

To do this efficiently we only need to apply the full equation H.10 to $k-2$ states – over the two remaining states the boundary conditions make the equation trivial. Equation H.11 needs to

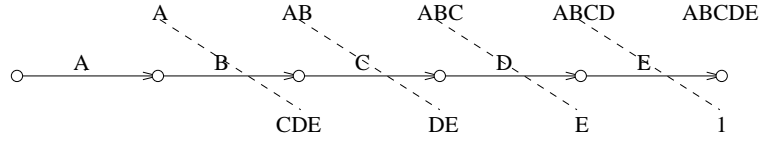


Figure H.2: The forward-backward algorithm at an $j = 4$ symbol node. The linear structure for just one particular member of $\text{GF}(q)$ is shown. ‘A’ represents the additional state for f_z and ‘B’-‘E’ the R_{iz}^a messages. The top row shows the α' values and the bottom row the β' values. The dotted lines represent the needed multiplications to obtain values of Q_{yz}^a from these.

be applied to all k states. Calculation of all R_{yz} values hence takes $2(k - 2)$ multiplications and $3k - 4$ additions. It should be noted though that $(k - 2)$ of these multiplications are easy as they are just multiplication by 2 – on a computer this can quickly be done just using a bit shift or addition of 1 to the exponent of a floating point number.

Each check node in total requires $4(k - 2)$ multiplications and $7k - 12$ additions.

H.1.2 Symbol Node

We will again use forward-backward algorithm to evaluate Q_{yz} and \hat{x}_z . An additional state will be used to deal with the f_z term, as shown in figure H.2. We define the forward and backward probabilities to be:

$$\alpha'_{yz}(a) = f_z(a) \prod_{i \leq y} R_{iz}^a \quad (\text{H.12})$$

$$\beta'_{yz}(a) = \prod_{i > y} R_{iz}^a \quad (\text{H.13})$$

Again we will break the computation down into steps:

1. **Evaluation of α' and β'** We need to know all the values of α' (taking $2j$ multiplications) and $j - 2$ values of β' (taking $2(j - 2)$ multiplications). This takes a total of $4(j - 1)$ multiplications.
2. **Evaluation of unnormalized Q_{yz}** We simply multiply values of α' and β' separated by one, as shown in figure H.2, this takes $2(j - 1)$ multiplications.
3. **Normalization** This is done by adding up each pair of probabilities and then dividing each probability by the result – this takes j additions, j multiplicative inverses and $2j$ multiplications (note the inverses and multiplications could be replaced by $2j$ divisions but we will again assume that multiplication is preferable).
4. **Evaluation of \hat{x}_z** We just need to compare $\alpha'_{jz}(1)$ and $\alpha'_{jz}(0)$ and find the maximum. This takes 1 comparison.

Operation	Number
Real multiplications	$4M(k - 2) + 2N(4i - 3)$
Real additions	$M(7j - 12) + Nk$
Binary multiplications	Mj
Binary additions	$M(k - 1)$
Multiplicative inverses	Nj
Comparisons	$N + M$

(a) Complexity in terms of parameters of the \mathbf{H} matrix

Operation	Number
Real multiplications	$4(3j + 2R) - 14$
Real additions	$8j + 12(R - 1)$
Binary multiplications	j
Binary additions	$j + R - 1$
Multiplicative Inverses	j
Comparisons	$2 - R$

(b) Complexity per transmitted bit in terms of rate

Table H.1: Complexity per iteration for non-Fourier algorithm

This comes to a total of $2(4j - 3)$ multiplications, j additions, j multiplicative inverses and 1 comparison.

Alternatively one could multiply all the R_{iz} terms and then use division to remove the unneeded ones – this takes fewer operations in total however it is likely that division will be significantly slower than multiplication and hence be slower overall.

H.1.3 Syndrome check

After calculating all the \hat{x}_z we then want to check whether this is a valid decoding. To do this we just test whether

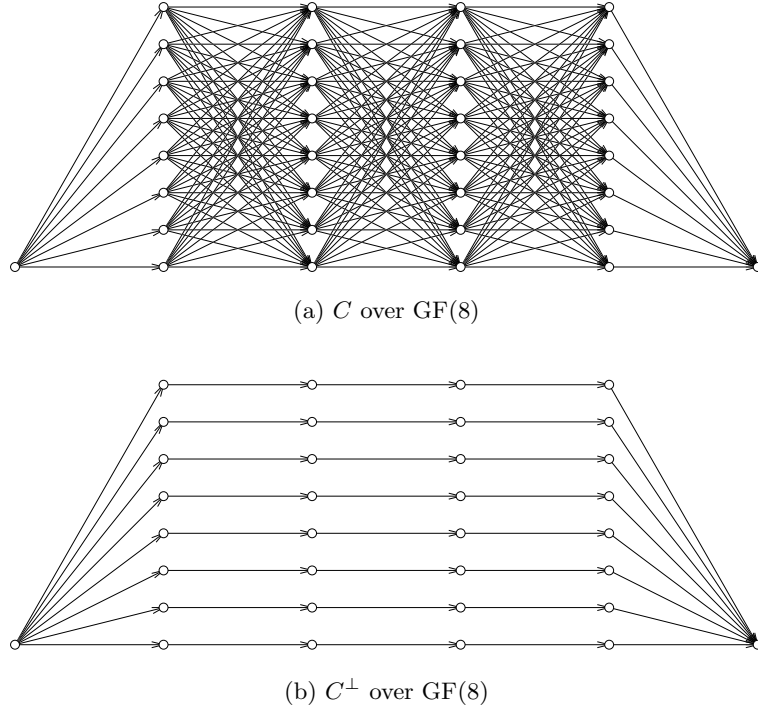
$$\mathbf{H}\hat{\mathbf{x}} = \mathbf{0}.$$

\mathbf{H} is sparse and hence the entire matrix multiplication does not need to be carried out, instead we can just concentrate on the non-‘0’ entries. This will take Mk binary multiplications, $M(k - 1)$ binary additions and M comparisons.

H.1.4 Total complexity per iteration

In each iteration we will update all the check nodes, all the symbol nodes and then do a syndrome check. The total number of operations per iteration is shown in table H.1(a).

It is useful to express the complexity in terms the rate R of the resultant code. Assuming


 Figure H.3: Trellises over $\text{GF}(q)$

linearly independent rows in \mathbf{H} this gives:

$$R = \frac{N - M}{N} \Rightarrow M = N(1 - R) \quad (\text{H.14})$$

The total number of connections in the graph must be the same from the point of view of the check nodes and the symbol nodes, so we can remove the dependence on the row weight:

$$Mk = Nj \Rightarrow k = \frac{j}{1 - R} \quad (\text{H.15})$$

The complexity per transmitted bit per iteration is shown in table H.1(b).

H.2 Fourier Representation

We will now look at codes defined over $\text{GF}(q) = \text{GF}(2^p)$ using Fourier transform decoding in the check nodes. We can define an isomorphism between $\text{GF}(2)^p$ and $\text{GF}(2^p)$ so codes defined over $\text{GF}(2^p)$ are convenient for binary digital data.

H.2.1 Check Node Computation

Unlike in the binary case the trellis for a check node computation is complicated, figure H.3(a). The number of branches scales as q^2 making the forward-backward algorithm computationally

demanding. By using a Fourier representation one can reduce the complexity as follows:

$$R_{ij}(a) = \sum_{\mathbf{x} \in C_i} \delta(x_j = a) \prod_{k \neq j} Q_{ik}(x_k) \quad (\text{H.16})$$

$$= \frac{|C_i|}{|\text{GF}(q)^n|} \sum_{\mathbf{x}' \in C_i^\perp} \mathcal{F}[\delta(\cdot = a)](x'_j) \prod_{k \neq j} \mathcal{F}[Q_{ik}](x'_k) \quad (\text{H.17})$$

$$\begin{aligned} &= \frac{1}{q} \sum_{\mathbf{x}' \in C_i^\perp} \sum_{x_j} \langle x_j, x'_j \rangle \delta(x_j = a) \prod_{k \neq j} \mathcal{F}[Q_{ik}](x'_k) \\ &= \frac{1}{q} \sum_{\mathbf{x}' \in C_i^\perp} \langle a, x'_j \rangle \prod_{k \neq j} \mathcal{F}[Q_{ik}](x'_k) \\ \mathcal{F}[R_{ij}](a') &= \frac{1}{q} \sum_a \langle a, a' \rangle \sum_{\mathbf{x}' \in C_i^\perp} \langle a, x'_j \rangle \prod_{k \neq j} \mathcal{F}[Q_{ik}](x'_k) \\ &= \frac{1}{q} \sum_{\mathbf{x}' \in C_i^\perp} q \delta(x'_j = a') \prod_{k \neq j} \mathcal{F}[Q_{ik}](x'_k) \\ \Rightarrow \mathcal{F}[R_{ij}](a') &= \sum_{\mathbf{x}' \in C_i^\perp: x'_j = a'} \prod_{k \neq j} \mathcal{F}[Q_{ik}](x'_k) \quad (\text{H.18}) \end{aligned}$$

where in going from equation H.16 to equation H.17 we have applied the Poisson summation formula [105] as suggested by Forney [36] to get to a summation over the dual code (C_i^\perp) and we can split up the Fourier transform as each term is in a separate dimension. Note equation H.18 has the same structure as equation 2.5 and hence can similarly be evaluated by the sum-product algorithm, but instead over the dual code with Fourier transformed messages.

The trellis for the dual code of each check node is trivial (the dual code is one-dimensional), figure H.3(b). Hence the addition in equation H.18 is only over one term. The number of branches now scales as q . This leads to an important reduction in complexity as the field size increases.

H.2.2 Fourier transformations

The Fourier transformation is defined as:

$$F(h_1, h_2, \dots) \equiv \mathcal{F}[f](h_1, h_2, \dots) \triangleq \sum_{(g_1, g_2, \dots) \in \text{GF}(2)^p} (-1)^{h_1 g_1 + h_2 g_2 + \dots} f(g_1, g_2, \dots) \quad (\text{H.19})$$

where we have mapped $\text{GF}(2^p) \mapsto \text{GF}(2)^p$ and used the “character” of $\text{GF}(2)$ that maps the elements on to $\{-1, 1\}$.

We can view this process as the application of a matrix similar to a Hadamard matrix, for example figure H.4.

$$\mathbf{F} : F(h_1, h_2) = \sum_{g_1, g_2} \mathbf{F}_{(h_1, h_2), (g_1, g_2)} f(g_1, g_2)$$

$$\begin{matrix} & (1, -1) & (1, 1) & (-1, 1) & (-1, -1) \\ \begin{matrix} (1, -1) \\ (1, 1) \\ (-1, 1) \\ (-1, -1) \end{matrix} & \begin{pmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{matrix}$$

 Figure H.4: Fourier transformation matrix \mathbf{F} for $\text{GF}(2^2)$

H.3 Complexity of the Fourier Decoding algorithm

In a similar manner to section H.1 we will calculate the complexity of the Fourier decoding algorithm. The major changes are the Fourier transformation and the normalization of the probability distributions. All of the messages will be unnormalized and the normalization will be done once per iteration in the check nodes. This leads to a reduction in complexity as explained below. The notation will be the same as used before.

H.3.1 Check Node

The forward-backward algorithm will be used to carry out the calculations in Fourier space. Our trellis now looks like figure H.3(b). The trellis is not time-invariant and hence one needs to be careful with how the forward and backward probabilities are defined. We label the i th symbol of dual code word z for check node y as d_{yz}^i . The forward and backward probabilities then are defined as:

$$\alpha''_{yz}(a) = \prod_{i \leq z} \mathcal{F}[\mathbf{Q}_{yi}](d_{ya}^i) \quad (\text{H.20})$$

$$\beta''_{yz}(a) = \prod_{i > z} \mathcal{F}[\mathbf{Q}_{yi}](d_{ya}^i) \quad (\text{H.21})$$

We can then recursively evaluate these in the similar manner to section H.1.2. To obtain the Fourier transformed \mathbf{R}_{ij} we then just have to multiply these:

$$\mathcal{F}[\mathbf{R}_{yz}](d_{ya}^z) = \alpha''_{y,z-1}(a) \beta''_{yz}(a) \quad (\text{H.22})$$

The algorithm breaks down into several steps:

1. **Fourier Transformation** Using the method in figure H.4 we are multiplying a $k \times q$ matrix (there are k rows as we will Fourier transform the messages from all connected symbol nodes at the same time) by a $q \times q$ matrix (\mathbf{F} , which can be pre-calculated) and so we would expect that kq^2 multiplications and $kq^2 - kq$ additions are needed. However

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \quad \begin{aligned} F(0) &= [f(0) + f(1)] + [f(2) + f(3)] \\ F(1) &= [f(0) - f(1)] + [f(2) - f(3)] \\ F(2) &= [f(0) + f(1)] - [f(2) + f(3)] \\ F(3) &= [f(0) - f(1)] - [f(2) - f(3)] \end{aligned}$$

Figure H.5: An example Fourier transform (after [22])

the matrix \mathbf{F} consists of only 1 and -1 entries so we can expand the calculation out into instead $kq^2 - kq$ additions or subtractions only.

A big saving can be gained by seeing that our matrix is a permutation of a Hadamard matrix. We can then split the calculation up into pairs and additions and subtractions. For example in figure H.5 it can be seen that four additions or subtractions are repeated and hence need only be done once. In general this gives us $nq \log_2 q$ additions or subtractions. This result stems from the fact that at each level of pairing there are q additions or subtractions and there are $\log_2 q$ levels of pairing.

It should be noted that this does not increase the memory usage as the inputs (or previous level of calculations) can be discarded once each pair of an addition and subtraction is done.

2. **Normalization** We then normalize each row in the resultant $n \times q$ matrix. This is an efficient time as the ‘0’ element of each Fourier transform is just the required normalization factor for that row. This takes k multiplicative inverses and $k(q - 1)$ multiplications.
3. **Evaluation of α'' and β''** The forward and backward probabilities each require $(k - 2)(q - 1)$ multiplications (the first and last state is either known or not needed in later calculations and the ‘0’ row of the trellis is trivial as all α'' and β'' values are just 1).
4. **Calculation of Fourier Transformed Probabilities** This multiplication of α'' and β'' requires $(k - 2)(q - 1)$ multiplications (the first and last states are just β'' or α'' respectively and the ‘0’ row is again trivial).
5. **Inverse Fourier Transform** The Fourier transformation used is proportional to an involution (a self-inverse). Thus to obtain unnormalized probabilities this requires the same $kq \log_2 q$ additions or subtractions as we did before. The normalizing factor is q for all the probabilities – this scaling factor does not affect anything apart from the scale of the symbol node calculations and can be left in the resultant messages.

This leads to a total of $2kq \log_2 q$ additions or subtractions, $2(2k - 3)(q - 1)$ multiplications and k multiplicative inverses per check node.

H.3.2 Symbol Node

We will use the same technique as before (figure H.2 and section H.1.2) but now over q field elements rather than just 2. The probabilities are now unnormalized but this does not change the overall calculation.

1. **Evaluation of α' and β'** This takes $2(j - 1)q$ multiplications.
2. **Evaluation of unnormalized \mathbf{Q}_{ij}** This takes $(j - 1)q$ multiplications.
3. **Evaluation of $\hat{\mathbf{x}}_z$** $q - 1$ comparisons are needed.

This comes to a total of $3(j - 1)q$ multiplications and $q - 1$ comparisons per symbol node.

H.3.3 Syndrome check

We will do exactly the same as section H.1.3. Our additions and multiplications are now over the field $\text{GF}(q)$ rather than being binary. This comes to Mk $\text{GF}(q)$ multiplications, $M(k - 1)$ $\text{GF}(q)$ additions and M comparisons.

H.3.4 Total complexity per iteration

As in the binary case the total complexity per iteration is shown in table H.2.

Meaningful comparisons are only possible if we assume the number of iterations are similar for all codes.

H.4 The best algorithm for binary codes

We can directly compare the Fourier algorithm with the non-Fourier algorithm in the case of $q = 2$. This is shown in table H.3. It can be seen that the only difference occurs for real multiplications and additions. For the Fourier algorithm to be more efficient we respectively require:

$$j > 1 - R \tag{H.23}$$

$$j > 3(1 - R) \tag{H.24}$$

It is likely that both of these will be satisfied. Equation H.23 will be satisfied as j in general is greater than 1. Equation H.24 will often be satisfied as j is often greater than or equal to 3. Therefore the Fourier decoding algorithm is liable to be more efficient even over $\text{GF}(2)$.

As q is increased we can hope for similar behaviour as the Fourier algorithm is $\mathcal{O}(q)$ whereas a traditional algorithm is $\mathcal{O}(q^2/\log_2 q)$ (as shown by the complexity of the trellis).

Operations	Number
Real multiplications	$2M(2k - 3)(q - 1) + 3N(j - 1)q$
Real additions	$2Mkq \log_2 q$
GF(q) multiplications	Mk
GF(q) additions	$M(k - 1)$
Multiplicative inverses	Mk
Comparisons	$N(q - 1) + M$

(a) Complexity per iteration in terms of parameters of the \mathbf{H} matrix

Operation	Number
Real multiplications	$\frac{(7j-9)q+6(R(q-1)+1)-4j}{\log_2 q}$
Real additions	$2jq$
GF(q) multiplications	$\frac{j}{\log_2 q}$
GF(q) additions	$\frac{j+R-1}{\log_2 q}$
Multiplicative Inverses	$\frac{j}{\log_2 q}$
Comparisons	$\frac{q-R}{\log_2 q}$

(b) Complexity per iteration per transmitted bit in terms of rate

Table H.2: Complexity for the Fourier algorithm

Operation	Non-Fourier number	Fourier number
Real multiplications	$4(3j + 2R) - 14$	$10j - 12 + 6R$
Real additions	$8j + 12(R - 1)$	$4j$
Binary multiplications	j	j
Binary additions	$j + R - 1$	$j + R - 1$
Multiplicative inverses	j	j
Comparisons	$2 - R$	$2 - R$

Table H.3: Comparison of the complexity of the Fourier and non-Fourier decoding algorithms over GF(2)

H.5 Conclusion

Moving to Fourier transform decoding reduces the complexity of the decoding algorithm in the binary case and also allows easier use of larger finite fields as the decoding algorithm scales as q .

APPENDIX I

EXIT CHARTS WITH A GAUSSIAN APPROXIMATION

I.1 Introduction

Following [104] we will construct extrinsic information transfer (EXIT) charts with the assumption that the messages are Gaussian distributed. We want to obtain a Gaussian approximation to the transfer function of an ensemble of nodes given that the input messages are samples from a Gaussian distribution.

I.2 Log-likelihood messages

A variable node adds incoming log-likelihood messages to produce an output message [39]. This simple function means it is sensible to look at log-likelihood messages. We will use the approach of [104].

First we need need to look at the form of log-likelihood ratio (LLR) messages from sources with Gaussian noise. Following [53] let L be the LLR ensemble from the Gaussian distributed real space ensemble Y :

$$\Pr(y|\mu = +\mu_y)dy = \frac{1}{\sqrt{2\pi}\sigma_y} \exp\left(-\frac{(y - \mu_y)^2}{2\sigma_y^2}\right) dy \quad (\text{I.1})$$

By definition the LLR from a bipolar input signal is defined as follows:

$$l \equiv \log(\Pr(y|\mu = +\mu_y)) - \log(\Pr(y|\mu = -\mu_y)) \quad (\text{I.2})$$

$$= \frac{2\mu_y}{\sigma_y^2} y \quad (\text{I.3})$$

We can then use this substitution to find the conditional distribution of L :

$$\Pr(l|\mu = +\mu_y)dl = \frac{dy}{dl} \frac{1}{\sqrt{2\pi}\sigma_y} \exp\left(-\frac{((\sigma_y^2/2\mu_y)l - \mu_y)^2}{2\sigma_y^2}\right) dl \quad (\text{I.4})$$

$$= \frac{1}{\sqrt{2\pi} \frac{2\mu_y}{\sigma_y}} \exp\left(-\frac{\left(l - \frac{2\mu_y^2}{\sigma_y^2}\right)^2}{2\left(\frac{2\mu_y}{\sigma_y}\right)^2}\right) dl \quad (\text{I.5})$$

which leads to the result that

$$L \sim N(\mu_l, \sigma_l) \quad (\text{I.6})$$

$$\text{where } \sigma_l^2 = 2\mu_l \quad (\text{I.7})$$

We can therefore parameterise a LLR message from a Gaussian source by σ_l alone.

We can now look at the information between a LLR message with standard deviation σ_l and a bipolar distribution in real space, X . To follow [104] we will call this function $J(\sigma_l)$.

$$J(\sigma_l) = I(X; L) = H(X) - H(X|L) \quad (\text{I.8})$$

$$= 1 - \sum_{x=\pm 1} \int_{-\infty}^{+\infty} \Pr(x, l) \log_2 \frac{1}{\Pr(x|l)} dl \quad (\text{I.9})$$

$$= 1 - \sum_{x=\pm 1} \int_{-\infty}^{+\infty} \frac{1}{2} \Pr(l|x) \log_2 \frac{1}{\Pr(x|l)} dl \quad (\text{I.10})$$

which by symmetry gives:

$$J(\sigma_l) = 1 - \int_{-\infty}^{+\infty} \Pr(l|x = +1) \log_2 \frac{1}{\Pr(x = +1|l)} dl \quad (\text{I.11})$$

For our bipolar X we can easily calculate

$$\Pr(x = +1|l) = \frac{\Pr(l|x = +1)}{\Pr(l|x = +1) + \Pr(l|x = -1)} \quad (\text{I.12})$$

$$= \frac{1}{1 + \exp(-l)} \quad (\text{I.13})$$

Substituting equation I.13 and equation I.5 into equation I.11 we get

$$J(\sigma_l) = 1 - \int_{-\infty}^{+\infty} \frac{\exp\left(-\frac{(l - \sigma_l^2/2)^2}{2\sigma_l^2}\right)}{\sqrt{2\pi}\sigma_l} \log_2(1 + e^{-l}) dl \quad (\text{I.14})$$

This sigmoid-like function can be evaluated numerically. The approximate piecewise closed forms for J and J^{-1} from [104] were used in calculations in this thesis.

I.3 Variable nodes

Variable nodes add log-likelihood messages [39]. The transfer function in terms of σ_l^2 can be calculated as a simple sum by the basic laws of probability theory. Hence the extrinsic information transfer function of a variable node with an incoming extrinsic information I and degree d_v is

$$f_v(I) \approx J \left(\sqrt{(d_v - 1)J^{-1}(I)^2 + \sigma_{ch}^2} \right) \quad (\text{I.15})$$

where σ_{ch} is the log-likelihood standard deviation of the message from the channel.

I.4 Check nodes

For check nodes we can use a duality property which holds for a binary erasure channel [1, 17]. With a binary erasure channel the set of messages passed during decoding is $\{0, 1, ?\}$. The extrinsic information of a message $I = 1 - p_{e,\text{mesg}}$ (where $p_{e,\text{mesg}}$ is the probability that a message is ‘?’). A variable node knows its state if any incoming message is not a ‘?’:

$$\Pr(\text{unknown}) = \Pr(\text{all erased}) = p_{e,\text{mesg}}^{d_v-1} = (1 - I)^{d_v-1} \quad (\text{I.16})$$

so

$$f_v^{\text{BEC}}(I) = 1 - (1 - I)^{d_v-1} \quad (\text{I.17})$$

A check node has to send a ‘?’ message unless all the other inputs are known:

$$\Pr(\text{known}) = \Pr(\text{all known}) = (1 - p_{e,\text{mesg}})^{d_c-1} = I^{d_c-1} \quad (\text{I.18})$$

where d_c is the check node degree, so

$$f_c^{\text{BEC}}(I) = I^{d_c-1} \quad (\text{I.19})$$

The following duality can be seen:

$$f_c^{\text{BEC}}(I) = 1 - f_v^{\text{BEC}}(1 - I) \quad (\text{I.20})$$

We then apply this duality to Gaussian messages to get:

$$f_c(I) \approx 1 - J \left(\sqrt{d_c - 1} J^{-1}(1 - I) \right) \quad (\text{I.21})$$

Simulations were carried out with Gaussian messages to find out how accurate this approximation is. In figure I.1 it can be seen that the approximation is reasonable.

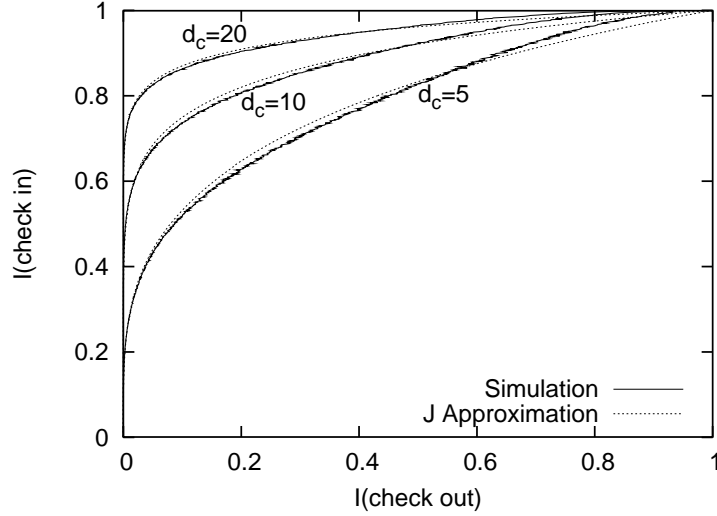


Figure I.1: EXIT charts of check nodes of different degrees

I.5 Irregular ensembles

When optimizing codes we have various different degree variable nodes and check nodes. To calculate the transfer function of an ensemble of nodes we take a weighted average of the transfer functions of each of the different degree nodes [1]. The weights are the proportions of edges coming from nodes of that type (proportional to the number of nodes of that type multiplied by number of outgoing edges from such a node).

I.6 Combined sections

In chapters 7 and 9 we combine the transfer function of the variable nodes with the additional transfer functions of the resynchronization and a convolutional code respectively. We will give this additional transfer function the symbol $f_a[I]$ (we use square brackets solely to clarify the equation below). The information flows through a variable node, to the additional transfer function and then back through a variable node. The overall transfer function can be evaluated by concatenating the transfer functions [104]. The overall transfer function for the variables nodes and convolutional code is:

$$J \left(\sqrt{(d_v - 1)J^{-1}(I)^2 + J^{-1} \left(f_a \left[J \left(\sqrt{d_v J^{-1}(I)^2 + \sigma_{ch}^2} \right) \right] \right)^2 + \sigma_{ch}^2} \right) \quad (\text{I.22})$$

For the variable nodes and resynchronization the transfer function is the same but with $\sigma_{ch} = 0$ as there is no channel input directly to the variable nodes.

BIBLIOGRAPHY

- [1] Alexei Ashikhmin, Gerhard Kramer, and Stephan ten Brink. Extrinsic information transfer functions: A model and two properties. In *36th Annual Conference on Information Sciences and Systems*, March 2002.
- [2] Lalit R. Bahl, John Cocke, Frederick Jelinek, and Josef Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, 20(2):284–287, March 1974.
- [3] Gérard Battail. On Gallager’s low-density parity-check codes. In *International Symposium on Information Theory*, page 202, June 2000.
- [4] Amir Bennatan and David Burshtein. On the application of LDPC codes to arbitrary discrete-memoryless channels. In *IEEE International Symposium on Information Theory*, page 293, July 2003.
- [5] J.M. Berger. A note on error detection codes for asymmetric channels. *Information and Control*, 4:68–73, 1961.
- [6] Elwyn R. Berlekamp. The technology of error-correcting codes. *Proceedings of the IEEE*, 68(5):564–592, May 1980.
- [7] Elwyn R. Berlekamp. *Algebraic Coding Theory*. Aegean Park Press, 1984.
- [8] Claude Berrou and Alain Glavieux. Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on Communications*, 44(10):1261–1271, October 1996.
- [9] Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *IEEE International Conference on Communications*, May 1993.
- [10] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error-correcting binary group codes. *Information and Control*, 3:68–79, March 1960.
- [11] P.A.H. Bours. *Codes for Correcting Insertion and Deletions Errors*. PhD thesis, Eindhoven Technical University, June 1994.

- [12] G.E.P. Box and M.E. Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29:610–611, 1958. Summary at <http://mathworld.wolfram.com/Box-MullerTransformation.html>.
- [13] Fredrik Brännström, Lars K. Rasmussen, and Alex Grant. Optimal scheduling for iterative decoding. In *International Symposium on Information Theory*, page 350, July 2003.
- [14] Jinn-Ja Chang, Der-June Hwang, and Mao-Chao Lin. Some extended results on the search for good convolutional codes. *IEEE Transactions on Information Theory*, 43(5):1682–1697, September 1997.
- [15] Johnny Chen, Michael Mitzenmacher, Chaki Ng, and Nedeljko Varnica. Concatenated codes for deletion channels. In *IEEE Interational Symposium on Information Theory*, page 218, July 2003.
- [16] Sae-Young Chung. LDPC code design applet. On the web at <http://lids.mit.edu/~sychung/gaopt.html>.
- [17] Sae-Young Chung. *On the Construction of Some Capacity-Approaching Coding Schemes*. PhD thesis, Massachusetts Institute of Technology, September 2000. <http://lids.mit.edu/~sychung/thesis/>.
- [18] Sae-Young Chung, Forney G. David, Jr., Thomas J. Richardson, and Rüdiger Urbanke. On the design of low-density parity-check codes within 0.0045dB of the Shannon limit. *IEEE Communications Letters*, 5(2):58–60, February 2001.
- [19] Sae-Young Chung, Thomas J. Richardson, and Rüdiger L. Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. *IEEE Transactions on Information Theory*, 47(2):657–670, February 2001.
- [20] Consultative Committee for Space Data Systems. *Telemetry Channel Coding*, October 2002. CCSDS 101.0-B-6.
- [21] Consultative committee for space data systems sub-panel 1B report, April 2003.
- [22] Matthew C. Davey. *Error-Correction Using Low-Density Parity-Check Codes*. PhD thesis, University of Cambridge, 1999. Available at <http://www.inference.phy.cam.ac.uk/mcdavey>.
- [23] Matthew C. Davey and David J.C. MacKay. Low-density parity check codes over $GF(q)$. *IEEE Communication Letters*, 2(6):165–167, June 1998.
- [24] Matthew C. Davey and David J.C. MacKay. Reliable communication over channels with insertions, deletions, and substitutions. *IEEE Transactions on Information Theory*, 47(2):687–698, February 2001.

- [25] Suhas N. Diggavi and Matthias Grossglauber. Bounds on the capacity of deletion channels. In *IEEE International Symposium on Information Theory*, 2002.
- [26] Dariush Divsalar and Fabrizio Pollara. Multiple turbo codes for deep-space communications. Technical report, Jet Propulsion Laboratory, May 1995.
- [27] M.P.F. dos Santos, W.A. Clarke, H.C. Ferreira, and T.G. Swart. Correction of insertions/deletions using standard convolutional codes and the Viterbi decoding algorithm. In *IEEE Information Theory Workshop*, pages 187–190, April 2003.
- [28] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.
- [29] DVB-S2 standard development, June 2003.
- [30] Todd Ebert. Why mathematicians now care about their hat color. *New York Times*, 10 April 2001.
- [31] Andrew W. Eckford, Frank R. Kschischang, and Subbarayan Pasupathy. Characterizing the Gilbert-Elliott parameter space under LDPC decoding. In *40th Annual Allerton Conference on Communication, Control, and Computing*, October 2002.
- [32] Peter Elias. Coding for noisy channels. *IRE Conv. Rec.*, 3(4):37–46, 1955.
- [33] H.C. Ferreira, W.A. Clarke, A.S.J. Helberg, K.A.S. Abdel-Ghaffar, and A.J. Han Vinck. Insertion/deletion correction with spectral nulls. *IEEE Transactions on Information Theory*, 43(2):722–732, March 1997.
- [34] Jeff Foerster and John Liebetreu. FEC performance of concatenated Reed-Solomon and convolutional coding with interleaving. Technical Report 802.16.1pc-00/33, IEEE 802.16, June 2000.
- [35] G. David Forney, Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3), March 1973.
- [36] G. David Forney, Jr. Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, 47(2):520–548, February 2001.
- [37] Brendan J. Frey. *Graphical Models for Machine Learning and Digital Communication*. The MIT Press, 1998.
- [38] Joerg M. Gablonsky and C. Tim Kelley. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21:27–37, 2001.
- [39] Robert G. Gallager. *Low-Density Parity-Check Codes*. MIT Press, 1963. Available at <http://www.inference.phy.cam.ac.uk/mackay/gallager/papers/>.

-
- [40] Robert G. Gallager. *Information Theory and Reliable Communication*. Wiley, 1968.
- [41] Robert G. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24:668–674, November 1978.
- [42] Javier Garcia-Frias. Decoding of low-density parity check codes over finite-state binary Markov channels. In *IEEE International Symposium on Information Theory*, 2001.
- [43] E. N. Gilbert. A comparison of signalling alphabets. *Bell Systems Technical Journal*, 31:504–522, May 1952.
- [44] Marcel J. E. Golay. Notes on digital coding. *Proceedings of the IEEE*, 37:657, June 1949.
- [45] Charles M. Goldie and Richard G.E. Pinch. *Communication Theory*. Cambridge University Press, 1991.
- [46] Andrea J. Goldsmith and Pravin P. Varaiya. Capacity, mutual information, and coding for finite-state Markov channels. *IEEE Transactions on Information Theory*, 42(3):868–886, May 1996.
- [47] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon codes and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, September 1999.
- [48] Joachim Hagenauer and Peter Hoeher. A Viterbi algorithm with soft-decision outputs and applications. In *IEEE Globecom*, 1989.
- [49] Joachim Hagenauer, Elke Offer, and Lutz Papke. Matching Viterbi decoders and Reed-Solomon decoders in a concatenated system. In Stephen B. Wicker and Vijay K. Bhargava, editors, *Reed-Solomon Codes and their Applications*. IEEE, 1994.
- [50] Richard W. Hamming. Error detecting and error correcting codes. *Bell Systems Technical Journal*, 29(2):147–160, April 1950.
- [51] R.V.L. Hartley. Transmission of information. *Bell System Technical Journal*, 7:535–563, July 1928.
- [52] A. Hocquenghem. Codes correcteurs d’erreurs. *Chiffres*, 2:147–156, September 1959.
- [53] Peter Hoeher, Ingmar Land, and Ulrich Sorger. Log-likelihood values and Monte Carlo simulation – some fundamental results. In *2nd International Symposium on Turbo Codes and Related Topics*, September 2000.
- [54] Xiao-Yu Hu, Evangelos Eleftheriou, and Dieter-Michael Arnold. Regular and irregular progressive edge-growth Tanner graphs, 2003. Submitted to *IEEE Transactions on Information Theory*.

- [55] David A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.
- [56] *IEEE Transactions on Information Theory Special Edition on Codes and Graphs and Iterative Algorithms*, volume 47(2). February 2001.
- [57] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE, 1998.
- [58] Frederick F. Sellers Jr. Bit loss and gain correction code. *IEEE Transactions on Information Theory*, 8(1):35–38, January 1962.
- [59] Stavros Konstantinidis, Steven Perron, and L. Amber Wilcox-O’Hearn. On a simple method for detecting synchronization errors in coded messages. *IEEE Transactions on Information Theory*, 49(5):1355–1363, May 2003.
- [60] Yu Kou, Shu Lin, and Marc P. C. Fossorier. Low-density parity-check codes based on finite geometries: A rediscovery and new results. *IEEE Transactions on Information Theory*, 47(7):2711–2736, November 2001.
- [61] Erik G. Larsson and Petre Stoica. *Space-Time Block Coding for Wireless Communications*. Cambridge University Press, 2003.
- [62] Hendrik W. Lenstra, Jr. and Gadiel Seroussi. On hats and other covers. In *International Symposium on Information Theory, Lausanne*, page 342, July 2002.
- [63] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics – Doklady*, 10(8):707–710, February 1966.
- [64] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their applications*. Cambridge University Press, revised edition, 1994.
- [65] Shu Lin and Daniel J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.
- [66] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2):585–598, February 2001.
- [67] David J. C. MacKay and Radford 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.
- [68] David J.C. MacKay. LDPC code database. On the internet at <http://www.inference.phy.cam.ac.uk/mackay/codes/data.html>.
- [69] David J.C. MacKay. Personal communication.

- [70] David J.C. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.
- [71] David J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. CUP, 2003.
- [72] David J.C. MacKay and Michael S. Postol. Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes. *Electronic Notes in Theoretical Computer Science*, 74, 2003. <http://www.elsevier.nl/locate/entcs/volume74.html>.
- [73] Yongyi Mao and Amir Banihashemi. A heuristic search for good LDPC codes at short block lengths. In *IEEE International Conference on Communications*, June 2001.
- [74] Robert J. McEliece. Are turbo-like codes effective on non-standard channels? *IEEE Information Theory Society Newsletter*, 51(4):1–8, December 2001. Based on 2001 ISIT Plenary Lecture.
- [75] Robert J. McEliece. On the average list size for the Guruswami-Sudan decoder. In *7th International Symposium on Communication Theory and Applications*, pages 2–6, July 2003.
- [76] Alistair Moffat, Radford M. Neal, and Ian H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, July 1998.
- [77] Mordechai Mushkin and Israel Bar-David. Capacity and coding for the Gilbert-Elliott channels. *IEEE Transactions on Information Theory*, 35(6):1277–1290, November 1989.
- [78] Harry Nyquist. Certain factors affecting telegraph speed. *Bell System Technical Journal*, 3:324–346, April 1924.
- [79] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.
- [80] J. Postel. *Transmission control protocol*, September 1981. RFC 793.
- [81] John G. Proakis. *Digital communications*. McGraw-Hill, 1995.
- [82] Tenkasi V. Ramabadran. A coding scheme for m-out-of-n codes. *IEEE Transactions on Communications*, 38(8):1156–1163, August 1990.
- [83] Edward A. Ratzner and David J.C. MacKay. Codes for channels with insertions, deletions and substitutions. In *2nd International Symposium on Turbo Codes and Related Topics*, 2000.
- [84] Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IEEE Transactions on Information Theory*, 4:38–49, September 1954.

- [85] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8(10):300–304, 1960.
- [86] Thomas J. Richardson, M. Amin Shokrollahi, and Rüdiger L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, February 2001.
- [87] Thomas J. Richardson and Rüdiger L. Urbanke. The capacity of low-density parity check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, February 2001.
- [88] Thomas J. Richardson and Rüdiger L. Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):638–656, February 2001.
- [89] Thomas J. Richardson and Rüdiger L. Urbanke. *Modern Coding Theory*. In preparation, 2003. <http://lthcwww.epfl.ch/papers/ics.ps>.
- [90] Jossy Sayir. Why Turbo codes cannot achieve capacity. In *3rd International Symposium on Turbo Codes and Related Topics*, pages 459–462, September 2003.
- [91] Leonard J. Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions and transpositions. *IEEE Transactions on Information Theory*, 45(7):2552–2557, November 1999.
- [92] 7-bit coded character set for information interchange. ISO 646:1991. International standardisation of the American Standards Association’s American Standard Code for Information Interchange, 1963.
- [93] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.
- [94] Claude E. Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64, 1951.
- [95] M. Amin Shokrollahi. Raptor codes. In *IEEE International Symposium on Information Theory*, July 2003. Recent Results Session, US Patent Application 20030058958.
- [96] Richard A. Silverman. On binary channels and their cascades. *IEEE Transactions on Information Theory*, 1(3):19–27, December 1955.
- [97] Neil J. Sloane. On single-deletion-correcting codes. In K.T. Arasu and A. Seress, editors, *Codes and Designs, Ohio State University, May 2000*, pages 273–291, 2002.
- [98] J.J. Stiffler. *Theory of synchronous communications*. Prentice-Hall, 1971.
- [99] L.R. Welch S.W. Golomb, B. Gordon. Comma free codes. *Canadian Journal of Mathematics*, 10(2):202–209, 1958.

-
- [100] T.G. Swart and H.C. Ferreira. Insertion/deletion correcting coding schemes based on convolution coding. *Electronics Letters*, 38(16):871–873, August 2002.
- [101] Peter Sweeney. Discussion at the International Symposium on Communication Theory and Applications, July 2001.
- [102] R. Michael Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.
- [103] Stephan ten Brink. Convergence of iterative decoding. *Electronics Letters*, 35(10):806–808, May 1999.
- [104] Stephan ten Brink, Gerhard Kramer, and Alexei Ashikhmin. Design of low-density parity-check codes for multi-antenna modulation and detection. Submitted to *IEEE Transactions on Communications*, June 2002.
- [105] Audrey Terras. *Fourier Analysis on Finite Groups and Applications*. Number 43 in London Mathematical Society Student Texts. Cambridge University Press, 1999.
- [106] Jeremy Thorpe. A randomized design algorithm for protograph LDPC codes. In *IEEE International Symposium on Information Theory*, July 2003. Recent results session.
- [107] Jeffrey D. Ullman. On the capabilities of codes to correct synchronization errors. *IEEE Transactions on Information Theory*, 13(1):95–105, January 1967.
- [108] Rüdiger L. Urbanke. A fast and accurate degree distribution optimizer for LDPC code ensembles. <http://lthcwww.epfl.ch/research/ldpcopt/>.
- [109] Sergio Verdú. *Multuser Detection*. Cambridge University Press, 1998.
- [110] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:751–772, 1967.
- [111] Tadashi Wadayama. An iterative decoding algorithm of low density parity check codes for hidden Markov noise channels. In *International Symposium on Information Theory and Its Applications*, November 2000.
- [112] Hong Shen Wang and Nader Moayeri. Finite-state Markov channel – a useful model for radio communication channels. *IEEE Transactions on Vehicular Technology*, 44(1):163–171, February 1995.
- [113] Niclas Wiberg. *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, 1996.
- [114] Andrew P. Worthen. *Codes and iterative receivers for wireless communication systems*. PhD thesis, University of Michigan, 2001. Available on the web at <http://www.eecs.umich.edu/~worthena/>.

-
- [115] Andrew P. Worthen and Wayne E. Stark. Low-density parity check codes for fading channels with memory. In *Proceedings of the 36th Annual Allerton Conference on Communication, Control and Computing*, 1998.
- [116] Andrew P. Worthen and Wayne E. Stark. Unified design of iterative receivers using factor graphs. *IEEE Transactions on Information Theory*, 47(2):843–849, February 2001.
- [117] John M. Wozencraft and Barney Reiffen. *Sequential Decoding*. MIT Technology Press and Wiley, 1961.
- [118] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR-2002-35, Mitsubishi Electric Research Laboratories, August 2002.
- [119] Jong-Hoon Youn and Bella Bose. Some improved encoding and decoding schemes for balanced codes. In *IEEE Pacific Rim International Symposium on Dependable Computing*, pages 103–109, December 2000.
- [120] Kamil Sh. Zigangirov. Sequential decoding for a binary channel with drop-outs and insertions. *Problemy Peredachi Informatsii*, 5(2):23–30, 1969.
- [121] Kamil Sh. Zigangirov. On the error probability of sequential decoding on the BSC. *IEEE Transactions on Information Theory*, 18(1):199–202, January 1972.
- [122] Michele Zorzi, Ramesh R. Rao, and Laurence B. Milstein. On the accuracy of a first-order Markov model for data transmission on fading channels. In *ICUPC'95, Tokyo, Japan*, November 1995.