Maynooth University
National University of Ireland Maynooth

Universal Error Correction Decoding Algorithms

A dissertation submitted for the degree of
Doctor of Philosophy

By:

Kevin Galligan

Under the supervision of:

Prof. Ken Duffy

Prof. Muriel Médard (Massachusetts Institute of Technology)

Hamilton Institute
National University of Ireland Maynooth
Ollscoil na hÉireann, Má Nuad

Contents

Abstract **iv**

Acknowledgements **vi**

Funding acknowledgements **vii**

Publications **viii**

Acronyms **ix**

1 Introduction **1**

1.1 An overview of the channel coding problem 1

1.2 Outline of thesis 8

2 Background **9**

2.1 Linear codes 9

2.2 Channel models 11

2.3 Parameters and statistics of channels 14

2.4 Guessing Random Additive Noise Decoding 15

2.4.1 Overview 15

2.4.2 Soft-input GRAND algorithms and ORBGRAND 18

2.4.3 GRAND complexity 21

2.5 Product codes 22

3 Iterative GRAND **25**

3.1 Introduction 25

3.2 Background 28
3.3 Iterative GRAND 30
3.3.1 Description 30
3.3.2 Implementation details for efficient decoding 31
3.3.3 Pseudocode 32
3.3.4 Columns-only decoding 32
3.4 Empirical Results 36
3.4.1 Experimental setup 36
3.4.2 Error correction performance 37
3.4.3 Complexity 40
3.5 Discussion 42

4 Block turbo decoding with ORBGRAND **47**
4.1 Introduction 47
4.2 Background 48
4.2.1 List decoding 48
4.2.2 Turbo decoding 49
4.3 List decoding with GRAND 51
4.4 Turbo decoding with GRAND 54
4.4.1 Description and complexity 54
4.4.2 1-line ORBGRAND for turbo decoding 55
4.5 Performance evaluation 57
4.5.1 List decoding 57
4.5.2 Turbo decoding accuracy 58
4.5.3 Turbo decoding complexity 62
4.6 Discussion 64

5 Alternative soft output for GRAND **67**
5.1 Introduction 68
5.2 Background 69
5.2.1 GRAND 70
5.2.2 Previous work on soft output 70
5.3 GRAND soft output 71

5.4 GRAND soft output per bit 77
5.5 Performance evaluation 80
5.5.1 Accuracy of soft output 80
5.5.2 Application to error detection. 85
5.6 Discussion 88

6 Discussion **90**
6.1 Future work 92

A Capacity-achieving proof for random linear product codes **95**

List of Figures **98**

Bibliography **100**

Abstract

There is no perfect communication channel, and any communication necessarily involves some level of noise. Attempting to hold a conversation across a crowded room, for instance, will likely result in miscommunication due to background noise. Channel coding, as a field, is concerned with reducing the rate of error in such noisy communication channels. This can be achieved by encoding messages with channel codes, which allow communication errors to be detected and corrected. The study of channel coding was launched in 1948 [78], and it now underlies critical technology such as the Internet, space communications, and storage of digital information [57].

In this thesis, we develop new algorithms and channel coding techniques based on Guessing Random Additive Noise Decoding (GRAND), a recently introduced family of decoders for channel codes. GRAND algorithms, unusually, can decode any channel code of any length that has a moderate amount of redundancy. Assuming that all messages are equally likely, they achieve maximum-likelihood decoding, which is the best possible outcome of decoding a channel code. GRAND challenges several assumptions of traditional channel coding and asserts a new decoding paradigm in which the particular channel code being used doesn't matter, allowing greater flexibility in the design of communication schemes. Given that an upper bound on GRAND's computational complexity increases exponentially with the amount of redundancy in a code, it is impractical to directly decode arbitrary channel codes that have a large amount of redundancy.

The goal of this thesis is thus to explore if GRAND can be used to decode such high-redundancy codes, which are suitable for the noisiest channel environments. To that end, we introduce and develop two iterative decoding algorithms, Itera-

tive GRAND (IGRAND) and block turbo decoding with GRAND, for a powerful class of channel codes known as product codes. Product codes are, in general, high-redundancy codes formed from a concatenation of low- to moderate-redundancy component codes. The key insight of the algorithms considered here is that GRAND can decode product codes by decoding each of their component codes in turn, circumventing the aforementioned complexity constraint.

Soft information indicates the reliability of a received message and is useful for a wide range of applications, including error detection and turbo decoding. In addition to the goal of decoding high-redundancy codes, this thesis also investigates the question of whether it is possible for GRAND decoding to output accurate soft information. We derive probabilistic soft output formulae for GRAND algorithms, evaluate their accuracy, and explore their application to error detection.

Acknowledgements

Thanks to Professors Ken Duffy and Muriel Médard for being a heroic supervisor team and for teaching me everything I now know about research.

Thanks to all of my wonderful collaborators, including the Network Coding and Reliable Communications group and Rabia's group in Boston. It was a pleasure to spend time and to collaborate with you all!

Thanks to Mam & Dad for reading to me when I was an annoying kid, and for supporting my 20+ years of education.

Thanks to Marie and Rachel for their constant sisterly presence during my 29 years of existence. Special thanks to Marie for making me aware that there was such a thing as a PhD.

Thanks to Sonia for all the emotional support and QT!

Thanks to all my friends at the Hamilton Institute for being there during all the trials and tribulations of the past 4 years. In particular, thanks to Anna for all the hot chocolate and juicy goss. Thanks to Fred for many distracting games of chess. Thanks to Dáire for the thoughtful gifts and sassy jokes. Thanks to my other cohort buddies: Hannah, Niamh, Amit, Ganesh and Bharvi. And thanks to all the other Hamiltonians, who are too numerous to mention by name.

Thanks to Rosemary, Kate and Joanna for keeping the Hamilton Institute afloat and for helping me to navigate the bureaucratic nightmares of Maynooth University.

And thanks to everyone there wasn't space to mention, except Chris.

Funding acknowledgements

All chapters:

- This publication has emanated from research supported in part by a Grant from Science Foundation Ireland under Grant number 18/CRT/6049. The opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Science Foundation Ireland.

Chapter 3:

- This work was supported by the Battelle grant Low-Probability-of-Detect/Intercept Communications Employing Peaky Frequency-Shift-Key Modulation (PO US0011-0000743557).

Chapter 4 and Chapter 5:

- This work was partially supported by Defense Advanced Research Projects Agency contract number HR00112120008.

Publications

- **K. Galligan**, P. Yuan, M. Médard, and K. R. Duffy. Upgrade error detection to prediction with GRAND. In *IEEE GLOBECOM*, 2023.
- **K. Galligan**, M. Médard, and K. R. Duffy. Block turbo decoding with ORBGRAND. In *CISS*, pages 1–6, 2023.
- A. Riaz, A. Yasar, F. Ercan, W. An, J. Ngo, **K. Galligan**, M. Médard, K. R. Duffy, and R. T. Yazicigil. A sub-0.8pJ/b 16.3Gbps/mm^2 universal soft-detection decoder using ORBGRAND in 40nm CMOS. In *IEEE ISSCC*, 2023.
- F. Ercan, **K. Galligan**, D. Starobinski, M. Médard, K. R. Duffy, and R. T. Yazicigil. GRAND-EDGE: A Universal, Jamming-resilient Algorithm with Error-and-Erasure Decoding. In *IEEE ICC*, 2023.
- F. Ercan, **K. Galligan**, K. R. Duffy, M. Médard, D. Starobinski, and R. T. Yazicigil. A General Security Approach for Soft-information Decoding against Smart Bursty Jammers. In *GLOBECOM Workshops*, pages 245 – 251, 2022.
- **K. Galligan**, A. Solomon, A. Riaz, M. Médard, R. T. Yazicigil, and K. R. Duffy. IGRAND: decode any product code. In *IEEE GLOBECOM*, 2021.
- A. Riaz, V. Bansal, A. Solomon, W. An, Q.Liu, **K. Galligan**, K. R. Duffy, M. Médard, and R. T. Yazicigil. Multi-Code Multi-Rate Universal Maximum Likelihood Decoder using GRAND. In *ESSCIRC*, 2021.

Acronyms

AWGN additive white Gaussian noise

BCH Bose-Chaudhuri–Hocquenghem

BER bit error rate

BLER block error rate

BDD bounded distance decoding

BPSK binary phase-shift keying

BSC binary symmetric channel

CA-Polar CRC-assisted polar

CA-SCL CRC-assisted successive cancellation list

CRC cyclic redundancy check

ECC error correction code

ER erasure rate

FER forward error correction

GRAND Guessing Random Additive Noise Decoding

HARQ hybrid automatic repeat request

IGRAND Iterative GRAND

KL Kullback-Leibler

LDPC low-density parity check

LLR log-likelihood ratio

ORBGRAND Ordered Reliability Bits GRAND

RLC random linear code

SGRAND Soft GRAND

SNR signal-to-noise ratio

SRGRAND Symbol Reliability GRAND

UER undetected error rate

URLLC ultra-reliable low-latency communications

CHAPTER 1

Introduction

1.1 An overview of the channel coding problem

Digital communication has become ubiquitous in recent decades and enables many utilities that we now take for granted, such as mobile phones, digital television, optical disk drives, and, of course, the Internet. One of the key technologies and fields of study underlying these applications is channel coding. In this section we will provide an overview of channel coding, including the key concepts and developments of the field since its inception 70 years ago.

The story of channel coding begins in 1948 with the publication of Claude Shannon's *A Mathematical Theory of Communication* [78], in which he set out to provide a mathematical framework for digital communication and to establish its theoretical limits. Shannon defined communication as follows:

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

Channel coding addresses the problem of reliable communication in the presence of a noisy communication channel. The channel in question could be a cable,

air, or outer space. Discrete information, such as a text message or an image, is encoded as a continuous-valued signal (often in the form of electromagnetic radiation or electricity), and is then transmitted through the channel. Noise is the corruption of this continuous signal, and is caused by physical phenomena as diverse as interference from other signals, solar radiation, a faulty wire, or a shark biting an undersea cable. The receiver of the signal must then convert it back to a sequence of discrete-valued symbols.

By this point, channel noise may have sufficiently corrupted the transmitted signal that there are errors in the discrete message at the receiver side. For example, the message "hello" could be corrupted and received as "cello", in which case the first symbol is an error. Such errors must be identified and removed in order to recover the original message. This is the purpose of channel coding. The message is encoded with an error correction code (ECC), or channel code, prior to transmission. In its encoded form, the message contains redundant information that makes it resistant to errors. The simplest channel code is a repeat code, which would encode the sample message as "hellohellohello".

On receiving an encoded message, the receiver has multiple options for handling errors. To continue the example, suppose that the discrete channel output is "cellohellohello". The receiver knows that errors are present because the 3 copies of the message are not in agreement. One possible mitigation is to request a retransmission from the sender, if possible. Alternatively, in what is known as **fec!** (**fec!**), the receiver can use a decoding algorithm to correct the errors, based on the redundant information provided by the channel code. In the case of the repeat code, a majority rule decoding algorithm can be used, in which case the message "hello" would win over "cello" with 2 votes to 1. The benefit of **fec!** is that it avoids a costly back-and-forth between sender and receiver, although the best solution depends on system constraints and may involve a hybrid of these and other approaches.

As a more concrete model of the channel coding problem, consider the communication system in Figure 1.1. This is the same framework that was first established by Shannon in 1948, and it remains the basic model today.

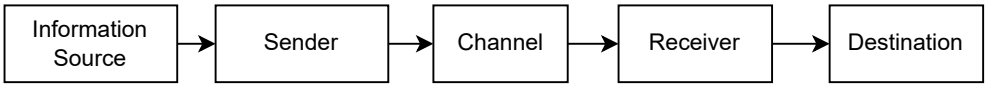


Figure 1.1: Basic model of a communication system.

The parts of the system can be described as follows [73].

1. *Information Source*: Digital messages are produced here. The messages are strings of symbols drawn from an alphabet, typically the binary alphabet $\{0, 1\}$. The symbols are also typically considered to be 1) independent of each other, and 2) drawn uniformly at random from the alphabet. This property can be ensured by compressing the messages from the information source; any correlation or non-uniformity can be exploited to compress the messages further until they satisfy properties 1 and 2.

2. *Sender*: Converts the message from digital (discrete) to analogue (continuous) using some signal modulation scheme, then transmits the analogue signal through the channel. If channel coding is included in the system design, then the message is encoded using a channel code before the digital-to-analogue conversion, which has the effect of adding redundancy to the message. The code rate R is the fraction of an encoded message that contains information and is not redundant. Since there are 5 letters of information in the repeat code considered above, and the remaining 10 letters are redundant, it has a code rate of $R = 5/15 = 1/3$. A higher code rate is generally more desirable, as it means that less energy is spent on transmitting redundant information, though the quality of the channel acts as a constraint on what code rates are possible while maintaining reliable communication.

3. *Channel*: The physical medium through which the message must pass, whether it's a vacuum or a metal wire or a beam of light. The channel adds noise to the signal. The noise is typically modelled as a probabilistic process.

4. *Receiver*: Demodulates the signal, recovering a discrete message that may or may not contain errors. May then make use of a channel code to detect

or correct these errors.

5. *Destination*: The intended recipient of the message.

Figure 1.2 illustrates an example transmission through this system.

Shannon showed that a channel's key property is its capacity, C, which is a measure of the rate at which information can be sent through it. As a channel becomes noisier, its capacity decreases. Shannon proved that arbitrarily reliable communication is possible if $R < C$. That is, if $R < C$, then the probability of error can be made arbitrarily small by choosing appropriate channel codes of rate R. If $R > C$, then no codes exist that can make the error probability arbitrarily small.

Shannon's work launched several fields of study, including channel coding, source coding and information theory. It popularised the term *bit*, short for *binary digit* (0 or 1), and the use of the bit as the standard unit of information content. However, it also left several open questions that would fuel the study of channel coding for the rest of the 20th century and beyond. Firstly, Shannon's proof showed the existence of good codes but did not reveal how to construct them. Secondly, the proof relied on allowing the code length, n, to become infinitely long, and it was unclear whether good codes existed at practical values of n. Thirdly, given a good code, Shannon did not explicitly present an algorithm to efficiently decode it.

We will explore this third challenge in more detail. Assuming that the message consists of k symbols drawn from a binary alphabet, there are 2^k possible input messages. The set of all encoded messages is known as the *codebook*, and the encoded messages themselves are known as *codewords*. Every message is mapped to a unique codeword, as otherwise it would be impossible for the receiver to distinguish between them. For example, with $k = 3$, and using a code that appends a single bit to the end of each message representing its parity, the codebook would be $\mathcal{C} = \{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}$.

Given output from a channel, the best possible decoding outcome is *maximum-likelihood decoding*. As the name implies, this is the codeword with the maximum likelihood of being the intended one. It's impossible to do better than maximising

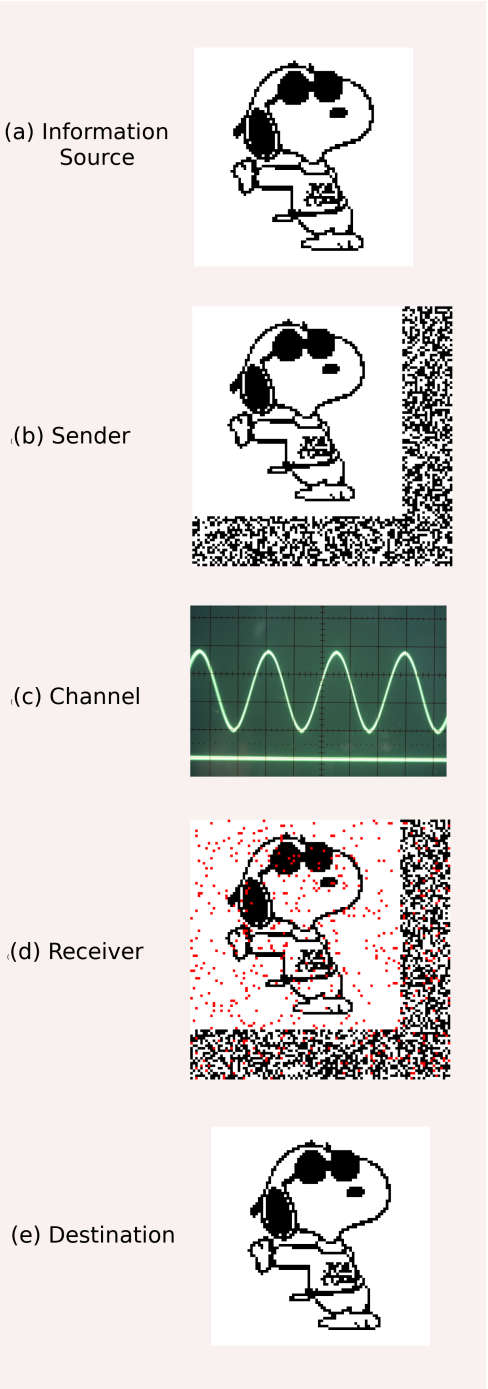


Figure 1.2: An example transmission through the communication system detailed in Figure 1.1, and an application of FEC. (a) The information source produces data to be sent. In this case, the data is a black and white image, which may be encoded in binary so that the white pixels are 0 and the black pixels are 1. (b) The sender adds redundancy to the data. The redundancy is a function of the original message. It could simply be a copy of the original message or it could be the result of something more involved like an algebraic operation on the message bits. (c) The encoded message (which is a codeword in the codebook) is converted to analog, modulated, and transmitted through the channel. (d) The receiver demodulates the channel output. The demodulated output may contain errors as a result of channel noise, represented here as red dots. (e) The receiver applies a decoding algorithm to correct errors and extract the original message from the channel output.

the likelihood of correctness, because given the probabilistic nature of noise, the original message can't be recovered with 100% certainty. The most straightforward algorithm to achieve maximum-likelihood decoding, and the one that arises implicitly in Shannon's proof, is a brute-force search where the likelihood of each of the 2^k codewords is calculated and the maximum-likelihood codeword is returned. The complexity of this approach, $\mathcal{O}(2^k)$, is exponential in the message length and for most purposes is impractical. For this reason it remained an open question to identify efficient decoding algorithms.

An important theoretical development that came later was the proof by Berlekamp et al. in 1978 that maximum-likelihood decoding of arbitrary linear codes is an NP-complete problem [14]. The proof assumed discrete input to the decoder, but the result also holds for more informative probabilistic input [60]. This makes it unlikely that there exists an efficient general-purpose decoder for all binary codes. Researchers have overcome this difficulty by designing codebooks with algebraic structures that, when combined with specialised decoders, allow them to be decoded efficiently. We will see, however, that universal decoding, i.e. the design of a decoder that can decode any binary code, can still be efficient when there are constraints on properties of the code such as n and R.

The design of channel codes, in tandem with decoding algorithms that can decode them efficiently, has become something of an art, and has been a significant focus of research in channel coding over the past 70 years. It involves a trade-off between the codeword length n, the code rate R, and algorithmic considerations. It is desirable for the code length to be short, because long codewords may delay transmission while enough data is accumulated to form an appropriately long message. On the other hand, Shannon's proof of arbitrarily low error rates was asymptotic in the code length n, and in practice many code constructions require large n in order to be effective. The code rate R must be below the channel capacity C to achieve arbitrarily low error rates, but ideally should be as close as possible to C in order to maximise the data rate through the channel while still guaranteeing reliable communication. On the other hand, lower R sometimes leads to faster decoding. It is also common for there to be trade-off in the choice of decoding algorithm. A more complex algorithm may achieve a lower error rate but at the

cost of higher latency, energy or hardware complexity.

The holy grail, so to speak, of channel coding has always been to design an error correction code, and an associated, computationally-tractable decoding algorithm, that *achieves capacity*. This means that the code should be able to achieve arbitrarily low error probability with a code rate equal to, or approaching, channel capacity.

The first reasonably sophisticated error correction code was introduced by Richard Hamming in 1950 [46]. He specified a code, now known as a Hamming code, with 4-bit input messages and 7-bit codewords (hence code rate $R = 4/7 \approx 0.57$) that was guaranteed to correct single-bit errors. This can be generalised to a larger family of Hamming codes, all capable of correcting a single error, with codeword length $n = 2^r - 1$ for $r \geq 2$ and message length $k = 2^r - r - 1$.

Following this came decades of research on the design of codes and decoding algorithms. One significant development was the invention of the *turbo decoding* technique in the 1990s [15][65], named after an analogous technique from the design of car engines. Turbo decoding makes use of probabilistic information from the channel, called *soft information*, which indicates the reliability of each symbol. *Hard information* is the discretised soft output (e.g. "cello" or "101110"). Turbo decoding iteratively refines soft information until it converges to the decoding output. Since the turbo decoding algorithm is given soft information as input and produces updated soft information as output, it is called a *soft-input soft-output* decoding algorithm.

Also in the category of iterative soft-input soft-output decoding techniques are low-density parity check (LDPC) codes and their belief propagation-style decoding methods. LDPCs were introduced in 1960 by Gallagher [37], but their implementation was infeasible at the time due to the constraints of computing technology. They were rediscovered in the 1990s and have since become widely used due to advances in computing capabilities. Both turbo decoding and LDPCs enabled researchers to come very close to channel capacity. That is, they could achieve tiny or arbitrarily small error probabilities at code rates that were almost at the value of channel capacity. Both turbo decoding and LDPCs continue to be widely used

in communications standards [1].

The final development we shall explore in the history of channel coding is the invention in the late 2000s of the polar code [12], the first code to reach the holy grail of provably achieving capacity for binary-input discrete memoryless channels, an important channel type that we will describe in more detail later. Polar codes were later combined with cyclic redundancy check (CRC) codes to form CRC-assisted polar (CA-Polar) codes, which enabled them to be decoded more effectively [82]. Significant effort has since been made to incorporate polar codes into modern communications technology, including the recent 5G 3GPP standard [1].

We have followed the progression of channel coding from its beginnings in 1948 to the present day. While the sought-after goal of finding a capacity-achieving code has been achieved, the trade-off inherent in code design means that channel coding remains a fruitful area of research, as no code or decoding algorithm is perfect for every application. Communication standards are constantly evolving to provide new applications, and the design of codes and decoding algorithms must continue in order to meet these ever-changing requirements. In the background chapter that follows, we will examine the fundamentals of channel coding in more detail and explore recent developments in channel coding that underlie the work in this thesis.

1.2 Outline of thesis

In this chapter we have introduced the channel coding problem. Chapter 2 presents more background detail on channel coding, GRAND, product codes and other prerequisites. Chapter 3 introduces IGRAND, a hard-input iterative decoding algorithm that can decode any product code. Chapter 4 introduces the decoding of product codes with soft-input GRAND algorithms. Chapter 5 introduces new formulae for calculating soft output for GRAND algorithms and discusses applications of this output. Finally, Chapter 6 provides an overarching discussion of the ideas of the thesis.

CHAPTER 2

Background

2.1 Linear codes

In Chapter 1, we introduced a class of error correction codes known as Hamming codes, which were among the first error correction codes to be invented. Hamming codes are part of the wider class of codes called *linear codes* [73]. The codewords of a linear code form a linear space, which, importantly, allows the code to be characterised concisely, as we will now see. If a binary linear code takes k-bit inputs and its codewords are n bits in length, then it is fully characterised by its $n \times k$ *generator matrix* G. If $u^k \in \{0,1\}^k$ is a k-bit input message, then its encoding is Gu^k, where u^k is a column vector and the multiplication takes place over the binary field $\mathbb{F}_2$. In linear algebra terms, the codebook of a linear code is the image of G, while the domain is the set of all possible messages to be encoded.

We use the notation $[n, k]$ to refer to a binary code with 2^k possible input messages, drawn from $\{0,1\}^k$, and 2^k codewords in $\{0,1\}^n$ with a mapping from each input message to a unique codeword. The rate of such a code is $R = k/n$ and $n - k$ is the amount of *redundancy* that the code adds to a message. The codebook of a code, i.e. the set of all codewords, is denoted $\mathcal{C} = \text{Image}(G) \subseteq \{0,1\}^n$.

The generator matrix G of a binary linear code can be stored in nk bits. Storing all 2^k codewords of a general $[n, k]$ code would require $2^k n$ bits, which is impractical

as k becomes large. It is for this reason that the vast majority of error correction codes in practical use are linear codes [73]. It has been proven that there is no performance penalty involved in using linear codes, because channel capacity can also be achieved by picking appropriate linear codes [64]. In fact, all of the codes mentioned in the introduction to this thesis (Hamming, LDPC, CRC, Polar, Turbo-decoded) are linear codes.

Linear codes have another matrix associated with them called their *parity-check matrix*, H. H is an $(n-k) \times n$ matrix with the property that, for all codewords $c^n \in \mathcal{C}$, $Hc^n = 0^{n-k}$, where 0^{n-k} is the all-zero vector. In linear algebra terms, H is the dual matrix of G [57]. H is important because it allows us to check $x^n \overset{?}{\in} \mathcal{C}$ using a single $\mathcal{O}(n^2)$ matrix multiplication Hx^n (known as a *syndrome computation*) instead of having to compare x^n to all 2^k codewords in the codebook. The constraint placed on the codebook by an individual row of the parity-check matrix is called a *parity check*.

A *systematic* code is one where, when forming a codeword, the original message bits are left untouched and redundant bits, or *parity bits*, are added to the end of the message. This is convenient because, following error correction, the original message can be extracted by isolating the first k bits of the message rather than having to invert a complex encoding operation. The generator matrix of a systematic linear code takes the form

$$G = \left[\frac{I_k}{P}\right],$$

where I_k is the $k \times k$ identity matrix and P is some $(n-k) \times k$ matrix. The corresponding parity-check matrix is $H = [P|I_{n-k}]$.

The final property of linear codes that we will explore is their *minimum Hamming distance*, denoted $d_{\min}$. This property is useful because, in the absence of soft information, it completely characterises the error correction capability of a code. In the binary case it's the minimum number of bit flips that must be performed to turn a codeword into any other codeword. More precisely, let the *Hamming*

weight of a binary sequence x^n be $w_H(x^n) = \sum_{i=1}^{n} x_i^n$. The *Hamming distance* between two codewords $x^n, y^n \in \mathcal{C}$ is $d_H(x^n, y^n) = w_H(x^n \ominus y^n)$, where $\ominus$ denotes element-wise subtraction of two vectors. The minimum Hamming distance is $d_{\min} = \min_{x^n, y^n \in \mathcal{C}} d_H(x^n, y^n) = \min_{x^n, y^n \in \mathcal{C}} w_H(x^n \ominus y^n) = \min_{x^n \in \mathcal{C}} w_H(x^n)$. The last equality comes from the linearity of the code, which means that $x^n \ominus y^n \in \mathcal{C}$, and it implies that the minimum Hamming distance of a linear code is in fact equal to the weight of the code's minimum-weight codeword.

Why does $d_{\min}$ characterise a code's error correction capability? If a codeword c^n is perturbed by noise such that fewer than $t = \lfloor d_{\min}/2 \rfloor$ of its bits are changed, then it is guaranteed that c^n is still the closest codeword to the channel output and a maximum-likelihood decoding algorithm should be able to identify it. Hence it is said that the code is t-error-correcting. The codewords can be visualised as being distributed throughout n-dimensional space, each codeword surrounded by a ball of radius t, as in Figure 2.1. As long as noise does not push an encoded message outside the ball that surrounds it, then the errors should be correctable. This all assumes that there is no soft information available, which may enable many more errors to be corrected, but minimum distance is still a useful measure of a code's effectiveness.

An $[n, k]$ linear code of minimum Hamming distance $d_{\min}$ will be denoted $C(n, k, d_{\min})$, where C is the class from which the code is drawn. For instance, LDPC$(128, 90, 5)$ denotes a $[128, 90]$ LDPC code with a minimum Hamming distance of 5. $C(n, k)$ may be used instead if the minimum Hamming distance is unknown or omitted.

2.2 Channel models

Real-world channels are complex physical systems. Some of their distinguishing features include the physical medium through which data is transmitted, the modulation of the transmitted signal, and the level of interference. A vast literature exists for modelling these diverse features [42]. Despite this complexity and variation, channel coding techniques are generally intended to function regardless of channel type. This is achieved by engineering coding systems so that the following simplifying assumption is guaranteed: that all transmitted symbols are affected by

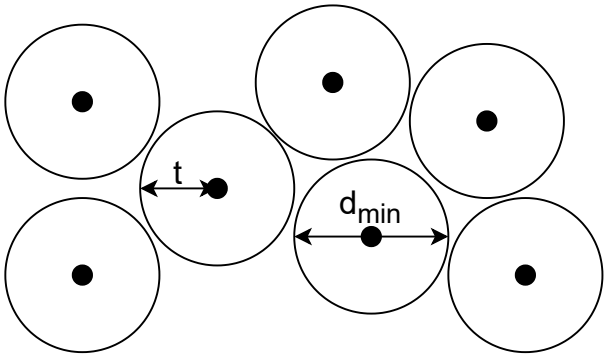


Figure 2.1: The dots are the codewords of a t-error-correcting channel code, distributed throughout n-dimensional space. They are surrounded by so-called Hamming balls of diameter $d_{\min}$ and radius t. A maximum-likelihood decoder would map any channel output within a ball to the codeword at the centre of the ball, since that codeword is most likely to be the intended one (assuming there is no additional soft information that indicates otherwise, and that all codewords are equally likely a priori). An n-dimensional lattice would actually be a more appropriate representation of the space, because the symbols are discrete.

the channel noise independently. Interleaving ensures the independence of the noise by moving adjacent bits away from each other during transmission, thus breaking any correlation. With this assumption in hand, the behaviour of the channel can be effectively captured by a simple probabilistic model; two such models that we will consider here are the binary symmetric channel (BSC) and the additive white Gaussian noise (AWGN) channel.

The BSC is depicted in Figure 2.2. Each bit transmitted through a BSC has an independent probability p of being flipped, thus introducing an error, and a probability $1 - p$ of being unchanged. The channel is called symmetric because 0 and 1 both have the same probability p of being corrupted. The BSC is called *memoryless* because the noise has an independent effect on each transmitted bit.

In the AWGN model, each symbol is modulated to produce a d-dimensional signal that usually is considered to lie somewhere in real space $\mathbb{R}$ ($d = 1$) or in the complex plane $\mathbb{C}$ ($d = 2$). During transmission, the signal is perturbed by the addition of a d-dimensional vector of Gaussian random variables. In binary phase-

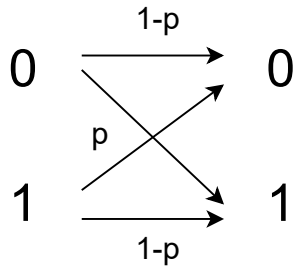


Figure 2.2: A binary symmetric channel with bit-flip probability p.

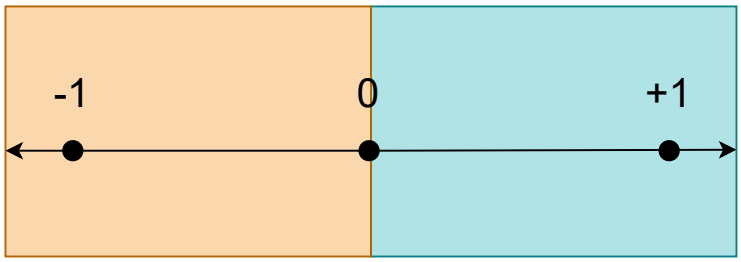


Figure 2.3: An illustration of BPSK modulation. 0 is mapped to -1, 1 is mapped to $+1$ (or the other way around). The receiver demodulates anything below 0 to 0 and anything above 0 to 1. In an AWGN channel, the signal is shifted by Gaussian noise, and a shift past 0 results in an error.

shift keying (BPSK) modulation (Figure 2.3), binary symbols are modulated to 1-dimensional real values: 0 is mapped to a value of -1, while 1 is mapped to $+1$ (alternatively, 0 may be mapped to $+1$ and 1 to -1). The receiver demodulates the received signal $y \in \mathbb{R}$ so that a value of $y < 0$ will be interpreted as 0 and a value of $y > 0$ will be interpreted as 1. If the value of the noise is large enough that it changes the sign of y, then the receiver will misinterpret the value of the symbol and an error will occur. AWGN channels are also memoryless, as there is no correlation between the d Gaussian random variables.

A channel can output *hard information* or *soft information*. A hard-output channel will return a sequence of symbols, some of which may be incorrect. For example, the BSC may receive as input the string 010 and may output 110, in which case the first symbol is received in error. A soft-output channel returns real-valued information that indicates not only the value of a symbol (such as 0 or 1), but also

its reliability. This may take the form of the raw signal value, or a probability of error, or a log-likelihood ratio comparing the likelihood of being in error versus not being in error, or any other measure of reliability. In an AWGN channel with BPSK modulation, the received signal is considered more unreliable as it gets closer to 0, as it is then more likely to have changed sign due to noise. In general, the use of soft information significantly improves the ability of the receiver to correctly decode received messages and can reduce error probability by an order of magnitude [57]. However, it may be expensive or infeasible to extract soft information for each symbol, and the decoder may have to make do with hard information.

2.3 Parameters and statistics of channels

A variety of parameters and statistics are used to describe channels and the performance of coding schemes within those channels. The bit-flip probability p, introduced already in describing the BSC, is one such parameter. The AWGN channel has a parameter σ that describes the standard deviation of the channel noise value. Assuming an AWGN channel and BPSK modulation with unit transmit power, a bit-flip probability of p is given by $\sigma = -1/(\sqrt{2}\text{erf}(2p-1))$ [42], where erf is the error function.

More generally, a quantity called the signal-to-noise ratio (SNR) is used to describe the ratio of the transmitter energy to the energy of the channel noise. When comparing channel codes, however, the code rate R must be considered; due to the overhead of transmitting redundant bits, the energy cost of a bit of information becomes $1/R$ times the cost of transmitting a single bit. For this reason, the quantity E_b/N_0 is typically used instead, where E_b is the energy per bit and $N_0/2$ is the power spectral density of the noise. E_b/N_0 is in fact equivalent to a rate-normalised SNR, given certain assumptions about the bandwidth and frequency of the transmitted signal. Once again assuming an AWGN channel and BPSK modulation, a bit-flip probability of p is given by

$$E_b/N_0 = \frac{1}{2R}\left(Q^{-1}(p)\right)^2,$$

where Q is the Q-function [42]. E_b/N_0 is typically expressed in decibels (dB), so that this equation becomes

$$10\log_{10}(E_b/N_0) = 10\log_{10}\left(\frac{1}{2R}\left(Q^{-1}(p)\right)^2\right).$$

To indicate the performance of a channel coding scheme, two metrics are commonly used: the bit error rate (BER), which is the fraction of bits that are decoded incorrectly by the receiver, and the block error rate (BLER), which is the fraction of messages that are decoded incorrectly by the receiver such that the intended codeword is not recovered. They can be defined more precisely as follows. Suppose that N messages are transmitted, each of which is n bits in length. Let $m_i \in \{0, 1, \ldots, n\}$ denote the number of bit errors in the decoding of the i-th of these transmissions. Then

$$\text{BER} = \sum_{i=1}^{N} m_i/(nN)$$

and

$$\text{BLER} = \left(\sum_{i=1}^{N} \mathbf{1}_{m_i>0}\right)/N,$$

where $\mathbf{1}_{m_i>0}$ is 1 if $m_i > 0$ and 0 otherwise.

2.4 Guessing Random Additive Noise Decoding

2.4.1 Overview

We have described in Section 1.1 how, given that the general decoding problem is NP-complete [14], decoding algorithms are typically designed to work with a single type of code structure, which they exploit to achieve efficient decoding. Efficient decoding algorithms for Hamming codes cannot generally decode any other type of code because they are specialised for the structure of Hamming code. However,

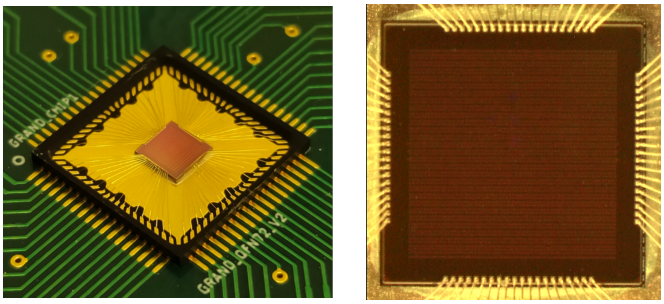


Figure 2.4: Photograph of the GRAND chip implementation from [67] for hard-input decoding of codes of up to 128 bits in length, and photograph of the ORBGRAND chip implementation from [69] for soft-input decoding of codes of up to 256 bits in length.

there has been continued effort to identify effective universal decoders, capable of decoding any code.

A recent addition to the category of universal decoding algorithms is Guessing Random Additive Noise Decoding (GRAND) [4, 5, 10, 11, 24, 27, 29, 30, 67, 80]. GRAND is a class of maximum-likelihood decoding algorithms that can accurately decode any code that has a moderate amount of redundancy. In addition to achieving maximum-likelihood decoding and being able to decode arbitrary codes, GRAND was proven to be capacity-achieving for channels with discrete input and output symbols, with or without memory [27]. Its promise for practical, highly-parallelised decoding has led to the publication of several circuit designs [2, 4, 5] and chip implementations [67, 69], the latter of which are pictured in Figure 2.4.

The key idea behind GRAND is that the effect of the noise is given by the difference between the demodulated channel output and the transmitted codeword. GRAND ignores the code structure and instead attempts to identify this noise effect and thereby infer the transmitted codeword; this is unlike existing decoding methods, which attempt to identify the transmitted codeword directly, often assuming an interleaved channel to do so. GRAND algorithms sequentially generate, from most to least likely, the noise effects that potentially corrupted the original message, based on knowledge of channel statistics or soft information. Given channel input $c^n \in \mathcal{C}$ and demodulated channel output $y^n \in \{0,1\}^n$, the *noise effect* introduced by the channel is $z^n = c^n \ominus y^n$, where $\ominus$ is the element-wise binary difference

operation. GRAND attempts to identify z^n and invert its effect on y^n, thereby yielding c^n. It does so by making a series of guesses $\hat{z}_1^n, \hat{z}_2^n, ...$ and computing $\hat{x}_i^n = y^n \ominus \hat{z}_i^n$ until it finds a sequence $\hat{x}_i^n$ that satisfies $\hat{x}_i^n \in \mathcal{C}$, which is a maximum-likelihood decoding. As described in Section 2.1, $\hat{x}_i^n \in \mathcal{C}$ can be confirmed for a linear code by checking that $H\hat{x}_i^n = 0^{n-k}$ [57], where H is the parity-check matrix and 0^{n-k} represents the length-$(n-k)$ all-zero vector. These core steps of the GRAND algorithm are described more explicitly in Algorithm 1.

Algorithm 1: GRAND decoding of hard channel output y^n, possibly with soft output r^n informing the likelihood of noise effects. Given a codebook $\mathcal{C}$ and code length n.

```
ẑ^n ← 0^n; // start with most likely noise effect
while true do
    ĉ^n ← y^n ⊖ ẑ^n; // undo putative noise effect
    if ĉ^n ∈ C then
        return ĉ^n;
    end
    ẑ^n ← next most likely noise effect;
end
```

As long as the sequence $\{\hat{z}_i^n\}$ is ordered so that $P(N^n = \hat{z}_i^n) \geq P(N^n = \hat{z}_j^n)$ for $i < j$, $N^n : \Omega \to \{0,1\}^n$ being a random variable representing the channel noise effect, this procedure is guaranteed to yield a maximum-likelihood estimate of the original message [27]. To see why, let the random variable $C^n : \Omega \to \mathcal{C}$ be a codeword and let $Y^n : \Omega \to \{0,1\}^n$ be the demodulated channel output. The instantiations of these random variables are c^n and y^n, respectively. A maximum-likelihood decoding is given by $c_{ML}^n \in \text{argmax}_{c^n \in \mathcal{C}} P(C^n = c^n | Y^n = y^n)$. Using Bayes' Rule,

$$\begin{aligned} P(C^n = c^n | Y^n = y^n) &= \frac{P(C^n = c^n, Y^n = y^n)}{P(Y^n = y^n)} \\ &= \frac{P(C^n = c^n) P(Y^n = y^n | C^n = c^n)}{\sum_{c_0^n \in \mathcal{C}} P(C^n = c_0^n) P(Y^n = y^n | C^n = c_0^n)}. \end{aligned}$$

Assuming that all codewords have equal prior probability, and given that the denominator in this expression is constant for a particular y^n, this implies that

$P(C^n = c^n|Y^n = y^n) \propto P(Y^n = y^n|C^n = c^n) = P(N^n = y^n \ominus c^n)$, and hence $\text{argmax}_{c^n \in \mathcal{C}} P(C^n = c^n|Y^n = y^n) = \text{argmax}_{c^n \in \mathcal{C}} P(N^n = y^n \ominus c^n)$. Thus, given that GRAND identifies a codeword $\hat{x}_q^n$ on its q-th guess, $P(C^n = \hat{x}_q^n|Y^n = y^n) = P(N^n = \hat{z}_q^n) \geq P(N^n = \hat{z}_i^n) = P(C^n = \hat{x}_i^n|Y^n = y^n)$ for $i > q$, and hence $\hat{x}_q^n$ is a maximum-likelihood estimate of the true codeword.

The exact ordering of the noise effects guessed by GRAND must depend on the channel, since a noise effect that is probable in one channel may be improbable in another. For this reason a family of GRAND algorithms has been developed, each one specialised for a different channel model. They distinguish themselves by how they generate the sequence of noise guesses, since the other main component of GRAND, the codebook check, is independent of channel type.

For discrete memoryless channels, including the previously-described BSC, noise effects can be generated in maximum-likelihood order by first generating the noise effects of Hamming weight 0, then Hamming weight 1, then Hamming weight 2, and so on, breaking ties arbitrarily. A chip implementation of this GRAND variant has in fact been produced [67]. This algorithm achieves maximum-likelihood decoding in a BSC with bit-flip probability $p < 1/2$. Given two noise effects $a^n, b^n \in \{0,1\}^n$ where $w_H(a^n) = i < j = w_H(b^n)$, their probabilities of occurrence are $P(N^n = a^n) = p^i(1-p)^{n-i}$ and $P(N^n = b^n) = p^j(1-p)^j$. a^n should come before b^n in a maximum-likelihood guessing order because $P(N^n = a^n) > P(N^n = b^n)$, and it does, since $i < j$ and the algorithm orders the noise sequences by Hamming weight.

2.4.2 Soft-input GRAND algorithms and ORBGRAND

Several variants of GRAND have been developed for soft-output channels [26, 29, 80]. Soft GRAND (SGRAND) [80] is a variant that achieves maximum-likelihood decoding in soft-output memoryless channels. It accomplishes this by generating noise effects based on a heap data structure, which grows with each noise guess and which tracks the next-most-likely noise effect. The downside of this approach is that it has high algorithmic complexity and is unsuitable for hardware implementation.

Ordered Reliability Bits GRAND (ORBGRAND) [24, 25] is an alternative soft-input variant of GRAND that comes close to achieving channel capacity [58] while having significantly lower complexity than SGRAND. After a pre-processing step that involves sorting the input bits based on their reliability, ORBGRAND has a fixed sequence of noise guesses, which makes it amenable to efficient hardware implementation, as evidenced by a recent integrated circuit design [5] and a chip implementation [69].

A more detailed explanation of ORBGRAND goes as follows. Given rank-ordered reliability values $\{r_i\}$ for the demodulated bits y^n, where $r_i \geq 0$ and $r_i < r_j$ for $i < j$, the likelihood of a putative noise sequence z^n is proportional to $f(z^n) = \sum_{i=1}^{n} r_i z_i^n$ [25]. The basic version of ORBGRAND approximates the reliability values with a line through the origin, $r_i = \beta i$, $\beta \geq 0$, and based on this model it efficiently generates noise sequences in approximate maximum-likelihood order. Then $f(z^n) = \beta \sum_{i=1}^{n} i z_i^n = \beta w_L(z^n)$, where $w_L(z^n)$ is the *logistic weight* of z^n. Since β is fixed for a particular instantiation of the noise, it can be disregarded when ordering noise effects based on f, and ORBGRAND thus generates noise effects in order of increasing logistic weight w_L. This approximation enables the efficient generation of noise effects in an order that is close to maximum-likelihood order.

ORBGRAND's generation of noise effects can be recast in terms of a combinatorics problem: integer partitioning with unique parts and a maximum part size. Generating the noise effects of logistic weight l for a code length of n is equivalent to constructing the integer partitions of the integer l with a maximum part size of n; the parts in each partition of l corresponds to an error at a particular position in the reliability-ordered hard channel output. For example, given $l = 5$ and $n = 5$, the corresponding partitions of l are $\{5\}$, $\{1, 4\}$ and $\{2, 3\}$, the parts of all of which sum to 5, as required of a partition of the integer 5. These partitions map to noise effects 00001, 10010 and 01100, respectively, assuming that the bits are ordered from least to most reliable; a part value of 2 indicates that the 2nd bit in sorted order should be a 1. The construction of partitions, and by extension the noise effect generation, can be achieved by using the *landslide algorithm* from [25].

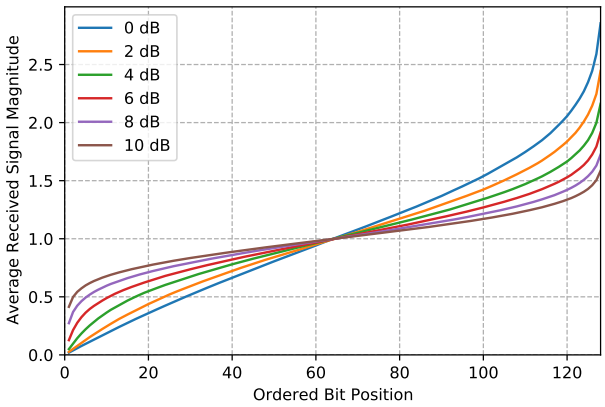


Figure 2.5: The position of a bit in sorted order, versus the average received signal magnitude at that position. The channel SNR (in decibels) is varied. A code length of $n = 128$ is assumed, and an AWGN channel model is used with BPSK modulation. The absolute value of the received signal is used as a proxy for the reliability of a bit, since they are proportional. It can be seen that a linear model does not fit the distribution as well when the channel SNR is higher.

Assuming that this linear approximation, described above, perfectly captures the reliability distribution, basic ORBGRAND generates noise effects in maximum-likelihood order. Particularly in low-noise channels, however, its approximation becomes worse, as seen in Figure 2.5.

The full version of ORBGRAND uses a multiline model to better match the reliability distribution in low-noise channels. Here, we further expand on the 1-line model, which distinguishes itself from basic ORBGRAND by potentially having a non-zero intercept. 1-line ORBGRAND uses the approximation $r_i = \alpha + \beta i$, where $\alpha, \beta \geq 0$. Then $f(z^n) = \sum_{i=1}^{n} (\alpha + \beta i) z_i^n = \alpha \sum_{i=1}^{n} z_i^n + \beta \sum_{i=1}^{n} i z_i^n = \alpha w_H(z^n) + \beta w_L(z^n)$, where $w_H(z^n)$ is the Hamming weight of z^n. $c = \alpha/\beta$ is assumed to be a non-negative integer. We then let $w_T(z^n) = f(z^n)/\beta = c w_H(z^n) + w_L(z^n)$ be the *total weight* of a noise sequence, where $w_T(z^n) \in \{0, 1, 2, ...\}$.

The description of the full ORBGRAND algorithm [25] details how to efficiently generate putative noise sequences in order of their total weight, which is an approximation of their maximum-likelihood order, without the need for dynamic memory.

A simple alternative method will be introduced in Chapter 4, and also a method to find a suitable value for c.

2.4.3 GRAND complexity

There is an upper bound on GRAND's decoding complexity based on the number of parity checks in the code, $n-k$. In general, an erroneous decoding is encountered in a binary code after approximately 2^{n-k} codebook queries [27][Theorem 2]. Intuitively, fewer codebook queries will be required in low-noise channels, as the probability will be concentrated on noise effects that introduce a small number of errors and the true noise effect is likely to appear early in the guessing order, before any erroneous codewords. In contrast, in high-noise channels, the probability will be distributed diffusely across all possible noise effects, and the true noise effect is likely to appear later in the guessing order. When the channel is sufficiently noisy, GRAND is more likely to identify an erroneous codeword than not, and the average number of queries becomes dominated by the term 2^{n-k}.

More precisely, the GRAND algorithm may be understood as a race between two random variables: the number of guesses to find the true channel noise effect $N^n : \Omega \to \{0,1\}^n$, thus recovering the correct codeword, and the number of guesses $U^n : \Omega \to \{1, ..., 2^n\}$ before GRAND identifies an incorrect codeword. GRAND's guessing order is defined by a bijective function $G : \{0,1\}^n \to \{1, ..., 2^n\}$ that maps each noise effect to its position in the guessing order. The number of guesses to identify the true channel noise effect is $G(N^n)$. GRAND identifies the correct decoding when $G(N^n) < U^n$, the asymptotic probability of which as n tends to infinity is derived in [27] for uniform random codebooks.

As n tends to infinity, the average number of guesses to identify the transmitted codeword is $\lim_{n\to\infty} \mathbb{E}[G(N^n)] = 2^{nH}$, where H is the Shannon entropy of the channel [20]. And, as stated previously, $\lim_{n\to\infty} \mathbb{E}[U^n] = 2^{n-k}$. The average number of queries performed by GRAND for asymptotic n and uniform random codebooks is thus upper bounded by $\min(2^{nH}, 2^{n-k}) \leq 2^{n-k}$.

The consequence of this is that, for practical applications, GRAND is suitable for codes with a moderate amount of redundancy. For $n-k = 20$, GRAND is expected

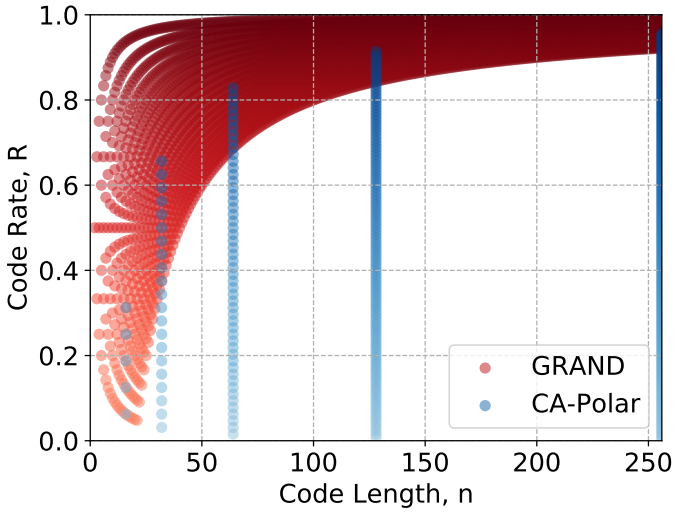


Figure 2.6: The code lengths and code rates at which GRAND can operate when given up to 20 parity checks, and up to a maximum code length of 256. Each dot corresponds to a pair of code length and rate. For comparison, the parameters of CA-Polar codes are shown as well, assuming the use of the 11-bit CRC prescribed by the 5G uplink scenario [1].

to make no more than $2^{20} = 1,048,576$ queries on average. Exponential growth soon makes this number impractical; for $n - k = 30$, it's $2^{30} = 1,073,741,824$. Still, GRAND offers great flexibility to the designers of communication systems, particularly at short code lengths. This is illustrated in Figure 2.6, which shows the decoding region, i.e. the code lengths and rates, at which GRAND can operate with up to 20 parity checks. Compared to CA-Polar codes, which in recent years were included in the 5G communications standard [1] and which exist only at lengths that are powers of 2, GRAND is completely flexible in terms of code length. Much of the focus of this thesis will be on expanding the decoding region of GRAND by applying it to a powerful class of codes called product codes, which are long, low-rate and high-redundancy.

2.5 Product codes

Product codes are a form of concatenated error correction code that were introduced by Elias in 1954 [32]. They allow the construction of long, powerful codes from short component codes. In 2 dimensions they are constructed by arranging

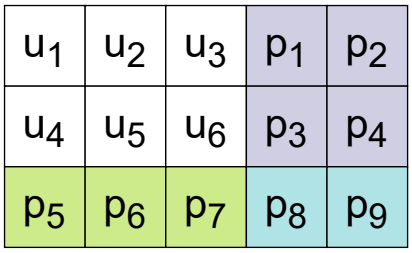


Figure 2.7: A 2-dimensional product code that has a [5, 3] component code $\mathcal{C}_1$ and a [3, 2] component code $\mathcal{C}_2$, giving it a code rate of $(k_1k_2)/(n_1n_2) = 6/15$. Information symbols u_i are structured as a $k_2 \times k_1 = 2 \times 3$ array. Rows are encoded using $\mathcal{C}_1$ and the array is extended with the resulting parity bits p_1, p_2, p_3 and p_4. The extended rows are codewords of $\mathcal{C}_1$. Similarly, the columns are extended by encoding them with $\mathcal{C}_2$. Encoding the parity bits of $\mathcal{C}_1$ produces so-called parity-on-parity bits p_8 and p_9. Note that the product code is both longer than its component codes and has a lower code rate.

information symbols in an array (see Figure 2.7). Each row of the array is extended by encoding it with a systematic code. Then the columns are extended by encoding each of them with another systematic code, possibly distinct from the one that was used for the rows. We use $C(n, k, d_{\min})^2$ to denote a product code with row and column codes of type $C(n, k, d_{\min})$.

More generally, a D-dimensional product code consists of D component codes $\mathcal{C}_1, \ldots, \mathcal{C}_D$, where each component code $\mathcal{C}_i$ is an $[n_i, k_i]$ systematic code of rate $R_i = k_i/n_i$. The product code is constructed by structuring $k = \prod_{i=1}^{D} k_i$ information symbols as a D-dimensional array of shape $k_D \times \cdots \times k_1$. All 1-dimensional slices in the last dimension, which are of length k_1, are extended by encoding them with the component code $\mathcal{C}_1$. The array then has shape $k_D \times \cdots \times k_2 \times n_1$. This procedure is repeated for each of the remaining component codes, giving the array a final shape $n_D \times \cdots \times n_1$.

Appropriately enough, the properties of a product code are all defined in terms of products of component code properties. A D-dimensional product code is length $n = \prod_{i=1}^{D} n_i$, contains $k = \prod_{i=1}^{D} k_i$ information symbols, and has a code rate $R = k/n = \prod_{i=1}^{D} R_i$. If the component codes are linear codes, and the i-th

component code has minimum Hamming distance d_i, then the product code has a minimum Hamming distance of $d_{\min} = \prod_{i=1}^{D} d_i$. Evidently, product codes are longer, lower rate, and have more error correction power than their component codes.

A key property of product codes with linear component codes is that all 1-dimensional slices (in the 2-dimensional case, that would be all rows and all columns) are in fact codewords of a component code, even if they are not explicitly encoded. This is most easily proven in the 2-dimensional case. Take as the encoding input a $k_2 \times k_1$ matrix X whose elements are drawn from a discrete alphabet. Let G_2 be the $n_2 \times k_2$ generator matrix for the column code and let G_1 be the $n_1 \times k_1$ generator matrix for the row code. Assuming the columns are encoded first and then the rows, $(G_1 X)G_2$ is the encoding of X. All rows of X are codewords of the row code because they were explicitly encoded with G_2, while only the first k_1 columns were encoded with G_1. If the rows are encoded first, the encoding is $G_1(XG_2)$, and this time all columns are explicitly encoded as codewords of the column code. By the associativity of matrix multiplication, $(G_1 X)G_2 = G_1(XG_2)$, and hence the outcome is the same regardless of the encoding order, and all rows and all columns are codewords.

CHAPTER 3

Iterative GRAND

This chapter introduces Iterative GRAND (IGRAND), a universal product code decoder. IGRAND applies iterative bounded distance decoding to product codes, correcting errors in their component codes using the code-agnostic GRAND algorithm. This approach overcomes the complexity constraints of GRAND and enables GRAND to be used for decoding high-redundancy codes, a task that would otherwise be infeasible. In addition to describing the algorithm, this chapter contains empirical results that demonstrate IGRAND's accuracy and efficiency, showing gains over comparable algorithms. A summary of the decoding accuracy results can be found in Table 3.2. Parts of this chapter have been published in [39].

3.1 Introduction

In previous chapters it was established that the average guessing complexity of GRAND, a universal maximum-likelihood decoding algorithm for error correction codes, generally has an upper bound of 2^{n-k}, where $n-k$ is the amount of redundancy in the code. This makes it impractical for GRAND to decode long, low-rate codes, which have a large amount of redundancy.

Here we propose to extend the feasible decoding region of GRAND by applying it to a class of codes known as product codes. As described in Section 2.5, product codes are long, high-redundancy codes that are formed from a concatenation of

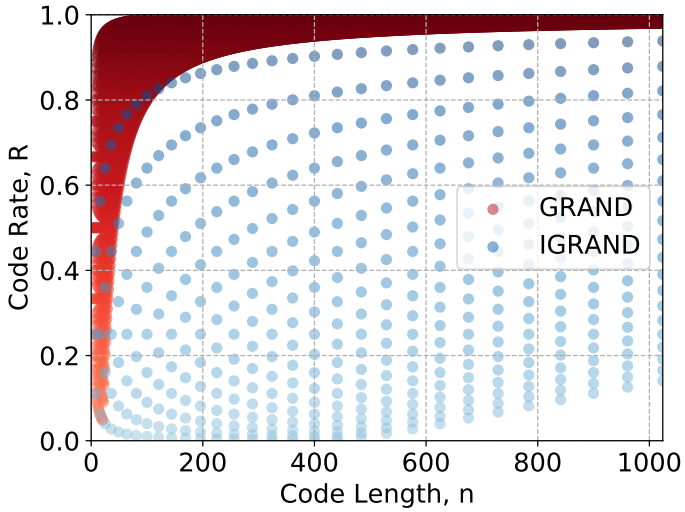


Figure 3.1: The code lengths and code rates at which GRAND and IGRAND can operate, up to a maximum code length of $n = 1024$. Each dot corresponds to a pair of code length and rate. GRAND is assumed to operate on codes of up to 20 parity bits, while for IGRAND it is assumed that the product codes have component codes of up to 20 parity bits (i.e. up to a total of $20 \times 20 = 400$ parity bits). It can be seen that IGRAND greatly expands the decoding region of GRAND, encompassing many long and low-rate codes. The product codes are also assumed to be square, i.e. to have row and column codes of the same length, and non-square product codes would be even more flexible.

shorter codes [32]. Product codes are the target of what we will call Iterative GRAND (IGRAND), a hard-input decoding algorithm that applies GRAND to the component codes of product codes, thus vastly expanding the range of application of GRAND. Figure 3.1 depicts the expanded decoding region that results from using IGRAND.

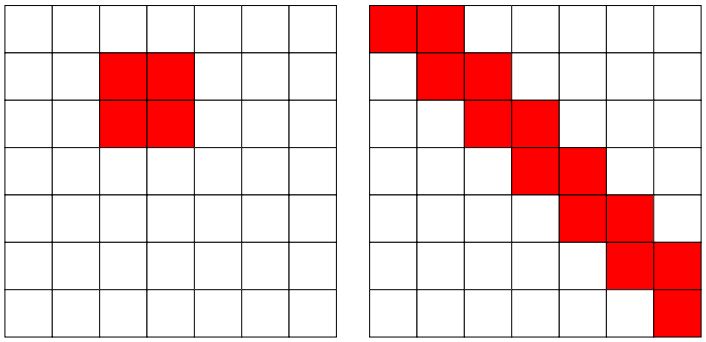
Decoding algorithms for product codes assume that there is a decoder for each component code. Elias, who introduced product codes in 1954 [32], proposed decoding the component codes sequentially: first the columns, then the rows. Performed repeatedly, this process is known as iterative decoding [57]. Subsequent efforts have produced near-optimal soft-input decoders for product codes [65], and we will consider these in the next chapter, but interest in hard-input decoding

continues [8, 9, 45, 50]. For example, high-rate product codes with hard-input decoders have been deployed in optical transport networks [51, 77, 79]. In such networks, where the bit rate can be as high as 100Gb/s, it may be infeasible to derive soft information for each bit, or soft-input decoding may be too slow. In storage applications [83, 88], soft information is simply unavailable.

IGRAND iteratively uses the GRAND algorithm to decode individual component codewords of product codes. IGRAND enforces bounded distance decoding (BDD) on GRAND, which not only improves decoding accuracy [8, 45], but also reduces GRAND's complexity. IGRAND does not define a fixed distance bound. It attempts to keep the distance bound small by initially setting it to the lowest possible value and increasing it only when necessary. We demonstrate through simulation that this approach offers accuracy and efficiency improvements over comparable algorithms.

The components of a product code can be decoded in parallel at no extra energy cost. This feature can be exploited by IGRAND, and we give estimates, based on measurements of a hardware implementation of GRAND [67], that full parallelisation of IGRAND can reduce its decoding latency by orders of magnitude. We also give an estimate of the energy cost of the GRAND chip if used to decode a 16,129-bit product code.

GRAND can reduce the hardware footprint of product code decoding. If product codes are not decoded in parallel, then the hardware footprint of their decoding is already modest because only a single decoder is required for each type of component code. GRAND enhances this flexibility, as only a single decoder is required for all component codes due to GRAND's code-agnosticism. This property also enables IGRAND to decode a particular class of product codes that are otherwise not decodable: those with random linear component codes. In Appendix A, we prove that such codes are capacity-achieving in hard information channels, though simulation results demonstrate that their performance does not live up to this promise at the finite block lengths considered here.



(a) An uncorrectable error pattern.

(b) A correctable staircase error pattern.

Figure 3.2: A 49-bit product code with 1-error-correcting component codes of minimum Hamming distance 3. While iterative decoding of the code cannot correct some 4-error patterns (note that $(\lfloor 3/2 \rfloor + 1)^2 = 4$), it can correct some error patterns with many more than 4 errors, such as the 13-error pattern shown above. This 13-error staircase pattern can be successfully decoded by correcting 1 error at a time over multiple iterations.

3.2 Background

All product code decoders assume the availability of a decoder for the component codes. Elias described how product codes could be decoded by decoding the component codes one after another [32] using the appropriate sub-decoder for each one. At least as early as 1968 [6] it was realised that multiple iterations of this process could improve error correction performance, though there remain error patterns that iterative decoding fails to correct. We refer to this algorithm, parameterised by the maximum number of iterations, as the Elias algorithm.

This iterative approach enables more errors to be corrected than indicated by the minimum Hamming distance of the product code. Consider a product code whose component codes both have a minimum distance of $d_{\min}$ and which therefore itself has a minimum distance of $d_{\min}^2$. Iterative decoding may fail to correct a block of $(\lfloor d_{\min}/2 \rfloor + 1)^2$ errors, as shown in Figure 3.2a, but can correct some error patterns with many more errors than that, such as the staircase pattern illustrated in Figure 3.2b.

It should be noted, however, that product codes are not theoretically optimal

codes in an algebraic sense. Given a particular length and code rate, the Hamming bound [57] is a limit on the best possible minimum Hamming distance. Consider a product code with BCH$(15, 11, 3)$ component codes. Its length is $15^2 = 225$, its code rate is roughly 0.53, and its minimum Hamming distance is $3^2 = 9$, which means it is guaranteed to correct at least 3 errors. The Hamming bound for these code parameters indicates that the maximum possible error correction capability is 23, and a BCH$(255, 143, 29)$, of approximately the same length and code rate, is guaranteed to correct up to 14 errors. Despite this non-optimality, product codes are used in practical settings because the algebraic properties of codes are less important when soft information is available, and due to their ease of decoding. In particular, their components can be decoded in parallel to improve throughput and latency, a property that we will explore later in the context of efficient GRAND decoding.

Justesen [50] used results from random graph theory to estimate the rate of occurrence of uncorrectable error patterns in product codes, placing an upper bound on their decoding accuracy. The bound is estimated by relating product codes to random graphs where the nodes represent component codewords. There's an edge between two nodes when the corresponding codewords have a bit in common and that bit is received in error. If the component codes of the product code can correct up to t errors, then the probability of an uncorrectable error pattern is the probability of a $(t + 1)$-core appearing in the graph, which is known for random graphs. A mitigation for uncorrectable error patterns is to identify their location during decoding, and treat the affected bits as erasures that can then potentially be recovered [9, 16, 45].

Another source of decoding failure is miscorrection, where new errors are introduced by attempting to decode a component codeword when it contains more than t errors. BDD [8, 45] minimizes the frequency of miscorrection events by applying error correction to a codeword only if φ or fewer bits are changed as a result. In a hard-output memoryless channel, the fewer bits that are changed by a decoding, the more likely the decoding is to be correct. If a product code can be decoded by correcting only up to φ bits at a time, then it is more likely that miscorrections will be avoided. Al-Dweik et al. [8] described an iterative decoding algorithm that

applies BDD with a bound of $\varphi = t - 1$ in the first iteration of decoding and discards the bound thereafter. We refer to this as the Al-Dweik algorithm. Like the Elias algorithm, it is parameterised by the maximum number of iterations allowed.

Inspired by a comment of Justesen's in [50], Häger et al.[45] developed another technique to avoid miscorrection called anchor decoding, which is complementary to BDD. It avoids changing bits in already-decoded components until they are contradicted sufficiently many times by the decoding of other components.

3.3 Iterative GRAND

3.3.1 Description

IGRAND decodes a product code's component codes using GRAND. Since GRAND is code-agnostic, IGRAND can decode arbitrary product codes. IGRAND enforces BDD on GRAND, which, in addition to improved decoding accuracy, provides a bound on GRAND's complexity. What distinguishes IGRAND from other iterative decoding algorithms is its particular use of BDD. Rather than allowing GRAND to decode a component codeword and then retroactively applying a distance bound, IGRAND passes the distance bound φ as a parameter to GRAND. If GRAND exhausts all noise sequences of weight up to φ without finding a decoding, it returns a failure. The outcome is the same as a retroactive application of BDD, but reduces complexity by not continuing the search for a decoding when it would in any case fall outside the distance bound. This assumes that GRAND makes noise guesses in order of increasing Hamming weight, which is the case in any hard-input memoryless channel.

In order to minimise complexity and maximise accuracy, IGRAND does not use a fixed distance bound, but instead starts with the lowest possible bound, $\varphi = 1$, and increments the bound only when necessary to make further progress in the decoding. IGRAND continues decoding iteratively until either 1) all components have been decoded successfully; 2) the decoding gets stuck not due to decoding failure, but due to an irreconcilable contradiction between row and column component codes; or 3) $\varphi > t$, where we assume that all component codes can correct t errors. When $\varphi > t$, further decoding is likely to cause miscorrections. If t

is unknown, such as when the component codes are chosen at random, then an optimistic value can be derived from the Hamming bound [70].

IGRAND avoids miscorrection as much as possible by keeping the distance bound as low as possible. This also serves the purpose of controlling GRAND's complexity. In general, GRAND makes up to $\min(2^{n-k}, \sum_{i=0}^{w_H(z^n)} \binom{n}{i})$ codebook queries to decode an $[n, k]$ codeword that has been affected by a binary noise sequence z^n of Hamming weight $w_H(z^n)$. With a distance bound of φ, this is limited to $\min(2^{n-k}, \sum_{i=0}^{\varphi} \binom{n}{i})$ codebook queries.

Figure 3.3 depicts IGRAND in operation, including its use of the status flags described in the next section.

3.3.2 Implementation details for efficient decoding

If a component has been decoded successfully and none of its bits are altered in the decoding of another component, decoding it again would be redundant. Thus, to be efficient, an iterative decoding algorithm should track the decoding status of each component. For this purpose we initialise an array S of status flags, where $S_i \in \{0, 1, 2\}$ denotes the decoding status of the ith component. Similar to the use of status flags in [45], 0 means the component is yet to be decoded, 1 means the decoding was successful, and 2 means the decoding failed.

All flags are initialised to $S_i \leftarrow 0$. At each iteration, the decoder should attempt to decode component i only when $S_i = 0$. If component i is decoded successfully, then its status is set to $S_i \leftarrow 1$. If the decoding of component i causes a bit to change in component j, then the status of j must be reset to $S_j \leftarrow 0$ because any decoding of j is no longer valid. A status flag of 2 is assigned if a component fails decoding due to a distance bound, or if the decoder can detect an error in the component but not correct it. In our implementations, the Al-Dweik algorithm and IGRAND interact with the status flags in a distinct manner to the Elias algorithm, which does not use a distance bound. When the distance bound is increased or discarded, then any component i whose decoding failed ($S_i = 2$) must have its status reset to $S_i \leftarrow 0$, because it may be possible to decode i with the new distance bound.

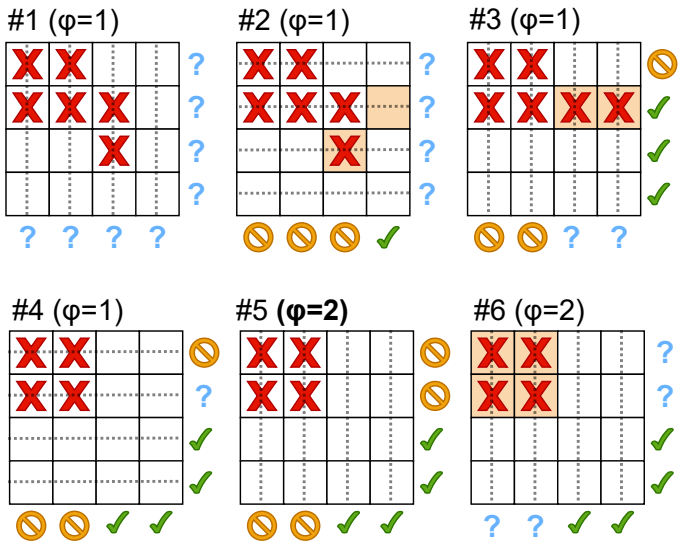


Figure 3.3: IGRAND decodes a 16-bit product code with 2-error-correcting component codes. Red crosses represent errors. Each component has an associated status flag, with a blue question mark beside component i if $S_i = 0$, a green tick if $S_i = 1$ and an orange crossed circle if $S_i = 2$. The direction of the dotted grey lines indicates whether IGRAND is about to decode columns or rows, while an orange background indicates that a bit is about to be changed. In #1, IGRAND decodes the columns. Since it starts with $\varphi = 1$, it fails to correct the first 3 columns, which each contain 2 errors. In #2, IGRAND changes 2 bits while decoding the rows, which leads it to retry the 3rd and 4th columns in #3. By #5, IGRAND is stuck and must increment φ, which allows it to correct the remaining errors following #6.

3.3.3 Pseudocode

Algorithm 2 contains pseudocode for IGRAND. Algorithm 3 describes the GRAND algorithm and how it interacts with the distance bound φ as well as the status flags.

3.3.4 Columns-only decoding

In low-noise channels, there is an opportunity to optimise decoding efficiency: if the columns of a product code appear to be error-free, then the row decoding can be skipped, halving the energy and latency cost of decoding. This comes at the

Algorithm 2: IGRAND decoding.

```
1  foreach component i do                                   // initialise flags
2  |  S_i ← 0;
3  end
4  φ ← 1;
5  while φ ≤ t do
6  |  decode columns with GRAND, distance bound φ;
7  |  decode rows with GRAND, distance bound φ;
8  |  if S_i = 1 ∀ i then                                   // all codewords decoded successfully
9  |  |  return;
10 |  else if no bits changed or S_i ≠ 0 ∀ i then                                   // stuck
11 |  |  if S_i ≠ 2 ∀ i then                                // bound increase won't help, give up
12 |  |  |  return;
13 |  |  else                                               // bound increase might help
14 |  |  |  φ ← φ + 1;
15 |  |  |  foreach component i do
16 |  |  |  |  if S_i = 2 then                             // retry failed codewords
17 |  |  |  |  |  S_i ← 0;
18 |  |  |  end
19 end
```

Algorithm 3: GRAND decodes the ith component, x^n, which should be a codeword of an $[n, k]$ codebook $\mathcal{C}$.

```
1  z^{n,*} ← 0^n;                                           // start with most likely noise effect
2  while w_H(z^{n,*}) ≤ φ do
3  |  c^{n,*} ← x^n ⊖ z^{n,*} ;                             // undo putative noise effect
4  |  if c^{n,*} ∈ C then                                   // found codeword, successful decoding
5  |  |  overwrite x^n with c^{n,*};
6  |  |  foreach affected component j ≠ i do
7  |  |  |  S_j ← 0 ;                                     // decode these components again
8  |  |  end
9  |  |  S_i ← 1;
10 |  |  return;
11 |  z^{n,*} ← next most likely noise sequence;
12 end
13 S_i ← 2;                                                // couldn't find decoding within distance bound
```

cost of occasionally missing an error that would have been corrected by the row decoding. Here we derive the probability that this scheme results in an efficiency gain, and the approximate probability that it results in a decoding mistake. Later we will estimate the effect that this approach has on the efficiency of IGRAND decoding.

Consider the transmission of a 2-dimensional product code through a BSC with bit-flip probability $p \leq 1/2$. Assume that a particular $[n, k, d_{\min}]$ code is used to encode both rows and columns of the product code. Let the Bernoulli random variable $X_{i,j} : \Omega \to \{0, 1\}$, parameterised by p, represent whether the bit at the intersection of the i-th row and j-th column is received in error. Following the assumptions of a BSC, the $\{X_{i,j}\}$ are independent. The number of errors in the j-th column is a binomial random variable $C_j : \Omega \to \{0, 1, ..., n\}$, parameterised by n and p, with $C_j = \sum_{i=1} X_{i,j}$. The $\{C_j\}$ also are independent and identically distributed.

Let G be the event that none of the columns have any errors. The probability of this event is

$$\begin{aligned} P(G) &= P(C_1 = 0, C_2 = 0, ..., C_n = 0) \\ &= \prod_{1 \leq j \leq n} P(C_j = 0) \\ &= \prod_{1 \leq j \leq n} (1 - p)^n \\ &= (1 - p)^{n^2}, \end{aligned}$$

where the second equality is due to the independence of the $\{C_j\}$.

It is also possible that sufficiently many errors occur in certain columns that these columns *appear* to be codewords and error-free, while the remaining columns are genuinely error-free. This will result in an erroneous decoding event, denoted M.

Let $C : \Omega \to \{0, 1, ..., n\}$ be a random variable with the same distribution as any individual C_j, $C = \sum_{i=1}^{n} X_i$ for n Bernoulli random variables $\{X_i\}$ parameterised

by p. Let E be the event $\{C \geq d_{\min}, (X_1, X_2, \ldots, X_n) \in \mathcal{C}\}$, where $\mathcal{C}$ is the codebook of minimum distance $d_{\min}$ and $(X_1, X_2, \ldots, X_n)$ is the error vector affecting the transmitted message. E is the event that a column appears to be a codeword despite containing errors.

Based on the earlier definition of a decoding mistake,

$$\begin{aligned} P(M) &= \sum_{l=1}^{n} \binom{n}{l} P(E)^l P(C=0)^{n-l} \\ &= (P(E) + P(C=0))^n - P(C=0)^n \\ &= (P(E) + (1-p)^n)^n - (1-p)^{n^2}. \end{aligned}$$

To approximate $P(E)$, assume that the codebook was created uniformly at random while ensuring a minimum distance $d_{\min}$, and that, given an error sequence containing $d_{\min}$ or more errors, there is a $2^k/2^n$ probability that it is a codeword. This approximation is supported by the fact that a length-n binary sequence chosen uniformly at random has probability $2^k/2^n$ of being in any given $[n, k]$ codebook. Thus

$$\begin{aligned} P(E) &= P(C \geq d_{\min}, (X_1, X_2, \ldots, X_n) \in \mathcal{C}) \\ &= P(C \geq d_{\min}) P((X_1, X_2, \ldots, X_n) \in \mathcal{C} | C \geq d_{\min}) \\ &\approx P(C \geq d_{\min}) \frac{2^k}{2^n}. \end{aligned}$$

Applying this approximation to the expression for $P(M)$ and expanding the binomial probability $P(C \geq d_{\min})$ gives

$$P(M) \approx \left(\frac{2^k}{2^n} \left(1 - \sum_{i=0}^{d_{\min}-1} \binom{n}{i} p^i (1-p)^{n-i} \right) + (1-p)^n \right)^n - (1-p)^{n^2}, \quad (3.1)$$

and this approximation is denoted by $\hat{P}(M)$.

Figure 3.4 and Figure 3.5 show the values of $P(G)$ and $\hat{P}(M)$ for a selection of product codes with 31-bit and 127-bit component codes, respectively. The bit-flip probability p is varied. Product codes with the same component code parameters will be examined for their error correction capability and efficiency in the sections to come. Together, these figures show that the probability of committing a decoding mistake is negligible with most component codes and under most channel conditions. Using a component code with low minimum distance increases the likelihood of a mistake, but even with $d_{\min} = 3$ the probability reaches a maximum of approximately 6×10^{-6} at the code lengths and noise conditions considered. Meanwhile, columns-only decoding is almost certain to save energy and latency in low-noise channels.

3.4 Empirical Results

3.4.1 Experimental setup

We compare the performance of IGRAND with that of two existing iterative decoding algorithms, using GRAND as their component code decoder: the Elias algorithm, which doesn't apply a distance bound; and the Al-Dweik algorithm, which applies a distance bound in the first iteration of decoding. GRAND enables the Elias and Al-Dweik algorithms to decode any product code. As with IGRAND, the GRAND-adapted Al-Dweik algorithm passes the distance bound as a parameter to GRAND. The Elias and Al-Dweik algorithms are parameterised by the maximum number of decoding iterations. As an upper bound on performance, we also run the Elias algorithm for 5 iterations with the so-called genie from [45] as its component decoder, which has side-channel access to the true channel noise and undoes noise effect z^n from a t-error-correcting component codeword if $w_H(z^n) \leq t$. The genie entirely avoids miscorrections, and the only remaining errors are caused by uncorrectable patterns or patterns that require more than 5 iterations to correct. While not practically realisable, it provides a performance bound.

We run simulations under a BSC model with bit-flip probability p. We relate p to E_b/N_0 using the usual BPSK formula $p = Q(\sqrt{2R\,E_b/N_0})$ [42], where R is the

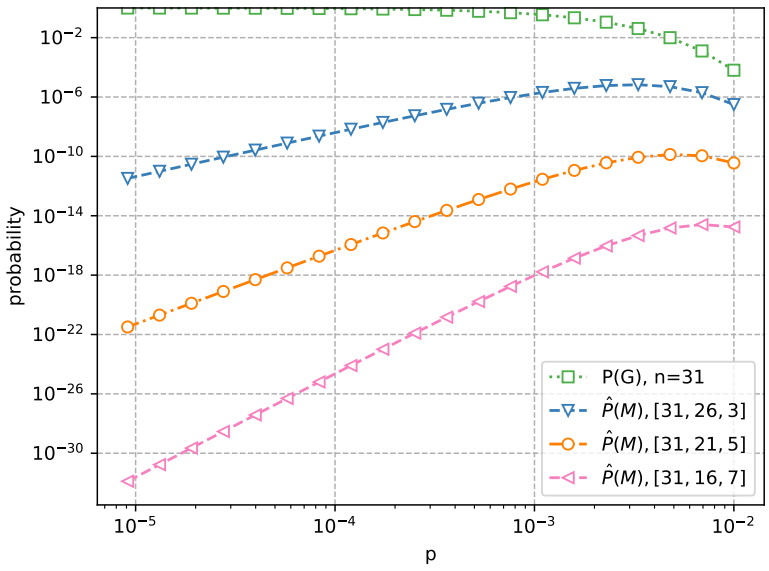


Figure 3.4: The probability of the event G, that columns-only decoding is applied successfully to a product code with 31-bit component codes; and the approximate probability of the decoding mistake event M. The bit-flip probability p is varied. The probability of G does not vary with the code rate, only the code length. The approximate mistake probability varies with the code rate and minimum distance, and so it is shown for a selection of product codes with the following component code parameters $[n, k, d_{\min}]$: $[31, 26, 3]$, $[31, 21, 5]$ and $[31, 16, 7]$. These particular parameters were taken from BCH codes.

code rate. These channel models and parameters were discussed more fully in Chapter 2.

3.4.2 Error correction performance

We first evaluate the decoding accuracy of IGRAND, the Elias algorithm using GRAND, and the Al-Dweik algorithm using GRAND when applied to two types of product code: one with BCH(31,21,5) component codes, and one with CRC(31,21,5) component codes. The corresponding product codes have parameters $[n, k] = [31^2, 21^2] = [961, 441]$ and their code rate is $R \approx 0.46$. The Al-Dweik algorithm was tested with this particular BCH product code in [8]. Though CRCs

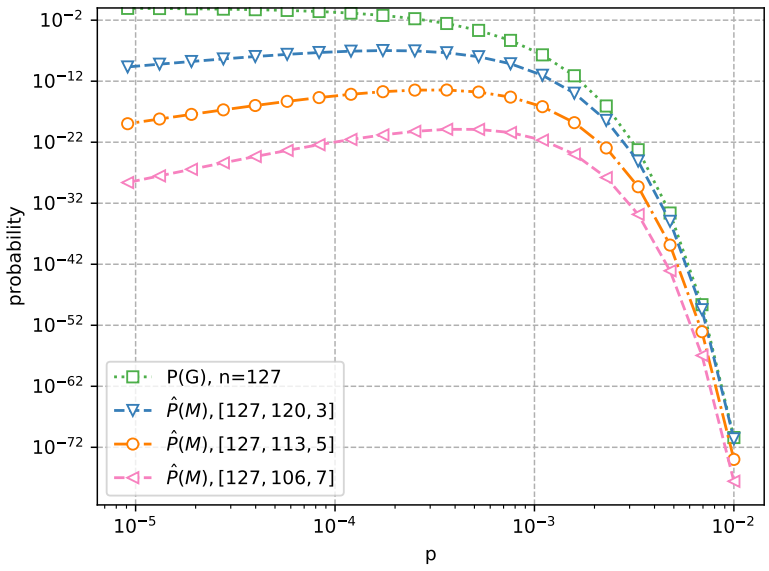


Figure 3.5: Same as Figure 3.4 but for 127-bit product codes with the following component code parameters: $[127, 120, 3]$, $[127, 113, 5]$ and $[127, 106, 7]$.

are traditionally used for error detection, GRAND can use them for error correction [11]. The CRC polynomial we use is represented as 0x2b9 in Koopman notation and was obtained from [55]. Figure 3.6 shows the BER of decoded bits under varying channel conditions. The BCH and CRC product codes have near-identical decoding accuracy. Independent of code type, IGRAND consistently provides a coding gain of more than 0.5dB over the Al-Dweik-GRAND algorithm, and 1dB over the Elias-GRAND algorithm. IGRAND's performance comes within 1dB of the theoretical genie decoder, which has full knowledge of the channel noise. Figure 3.7 depicts the results from the same experiment but with a $\text{BCH}(63, 51, 5)^2$ product code, with IGRAND once again consistently achieving a 0.5dB gain over Al-Dweik-GRAND.

Figure 3.8 shows the decoding accuracy of IGRAND when applied to a broader selection of product codes. It can once again be seen that CRC codes match the performance of BCH codes, despite typically being used for error detection. The random linear code (RLC) performs relatively poorly as a component code, despite

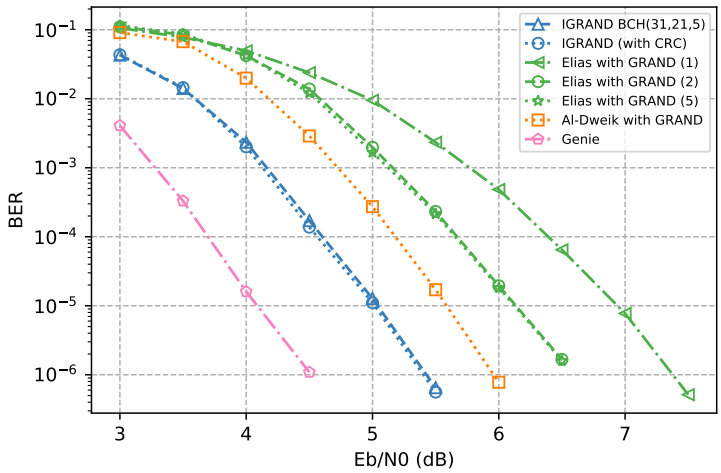


Figure 3.6: BER of a product code with BCH(31,21,5) component codes. In one case, a CRC(31,21,5) component code is used instead, and gives identical performance to the BCH code. The Elias and Al-Dweik algorithms are adapted to use GRAND as their component decoder. The maximum iterations parameter of the Elias algorithm is indicated in the legend. At the code rate of these product codes, $R = (21/31)^2 \approx 0.46$, an SNR per bit of $E_b/N_0 = 3\,\text{dB}$ corresponds to a channel BER of approximately $p = 8.7 \times 10^{-2}$.

product codes with RLC component codes being capacity-achieving in the infinite block length regime (see Appendix A). Of final note is the CA-Polar code, widely used in modern communications standards, which underperforms relative to BCH and CRC codes of similar length and rate. As reported in [11], the error correction power of CA-Polar codes comes primarily from the CRC, and the polar bits can almost be considered to weaken the code.

Further exploring the performance of the RLC class of codes, Figure 3.9 depicts the decoding accuracy of the BCH$(31, 21, 5)$ code considered earlier versus a RLC with a minimum Hamming distance of 4, purposefully constructed to be slightly weaker than the BCH in terms of minimum distance. As standalone codes, their performance is similar, but when used as component codes, the performance gap is magnified. This can be attributed to the properties of product codes: as described

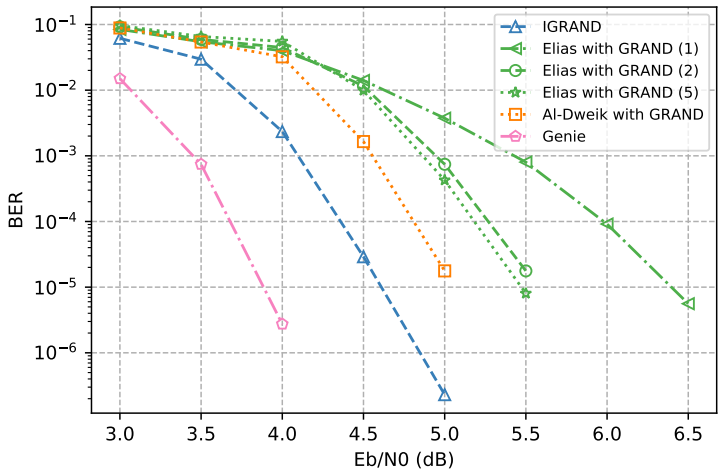


Figure 3.7: A plot of decoding accuracy with the same parameters as Figure 3.6 except with a BCH$(63, 51, 5)^2$ product code, which has a code rate of $R = (51/63)^2 \approx 0.65$.

in Section 2.5, the minimum Hamming distance of a product code is the square of the minimum Hamming distance of its component code (assuming a single type of component code). This exacerbates any weakness in the codebook structure of an RLC, or indeed of any code, as the minimum Hamming distance becomes even smaller relative to the code length.

3.4.3 Complexity

We compare decoding complexity in terms of energy usage and latency, based on measurements from a hardware implementation of GRAND [67]. Table 3.1 gives the average energy and latency required by the GRAND chip to decode a 3-error-correcting code of length $n = 128$.

Based on these hardware measurements, we compare the complexity of IGRAND, the Elias-GRAND algorithm (1, 2 & 5 iterations) and the Al-Dweik-GRAND algorithm (5 iterations) when applied to a 16,129-bit product code of code rate $R \approx 0.7$ with BCH(127,106,7) component codes. Figure 3.10 plots average energy usage as a function of $-log_{10}(p)$, with p denoting the bit-flip probability of the BSC. This scale is used for the x-axis instead of E_b/N_0 because it is typical for hardware

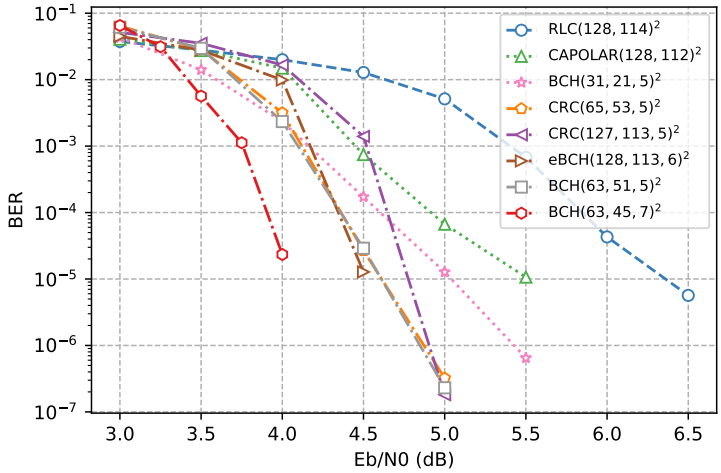


Figure 3.8: The decoding accuracy of IGRAND when applied to a large selection of product codes: RLC$(128, 114)^2$, the class of product codes with random linear component codes, where a new component code was randomly generated for each transmission so that no row or column of the parity matrix was all-zero; CAPOLAR$(128, 112)^2$ with the 11-bit CRC from the uplink scenario of 5G [1]; BCH$(31, 21, 5)^2$, BCH$(63, 51, 5)^2$ and BCH$(63, 45, 7)^2$; CRC$(65, 53, 5)^2$ and CRC$(127, 113, 5)^2$ with CRC polynomials 0xbae and 0x212d [55], respectively; and eBCH$(128, 113, 6)^2$, an extended BCH code with a single additional parity check.

comparisons [67]. The decoding has a base energy cost of 62pJ/bit, as seen in low noise channels. This is the cost of performing a single codebook check for all $2 \times 127 = 254$ components to confirm that they're error-free. In noisier channels, the distance bound becomes significant and IGRAND uses significantly less energy. At $p = 10^{-2.5}$, IGRAND uses roughly half as much energy as Al-Dweik-GRAND, and almost an order of magnitude less energy than Elias-GRAND.

Figure 3.11 plots the average decoding latency, which bottoms out at roughly 18,000 cycles. In noisier channels, IGRAND achieves significantly lower latency than the other algorithms. Latency can be improved further by decoding in parallel, which increases hardware footprint but not energy expenditure. Figure 3.12 plots the average latency and throughput of IGRAND when up to 128 decoding branches are used in parallel. With 128 decoding branches, latency can be as low

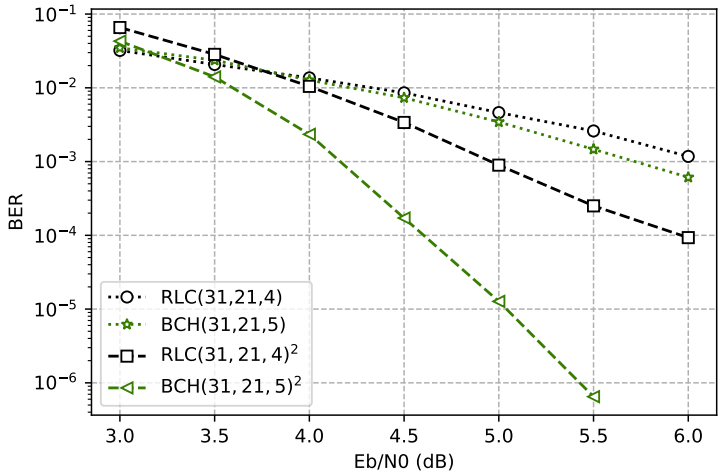


Figure 3.9: The decoding accuracy of BCH$(31, 21, 5)$ compared to a particular random linear code RLC$(31, 21, 4)$, both as the component codes of a product code and as standalone codes. The performance gap is exacerbated by using them in product codes.

as 142 cycles, which is the cost to confirm that the columns (71 cycles) and the rows (71 cycles) are error-free.

Finally, in less noisy channels, the latency (and energy) could be halved again by performing columns-only decoding, which would reduce the latency with 128 decoding branches to 71 cycles. Figure 3.13 shows the estimated reduction in IGRAND decoding latency with columns-only decoding, confirming that the baseline latency is halved.

3.5 Discussion

We have introduced IGRAND, an iterative decoding algorithm that can decode any product code. IGRAND improves decoding accuracy over other BDD-based algorithms by attempting to keep the distance bound as low as possible throughout decoding. This approach to BDD could be adapted by any iterative decoding algorithm, and is complementary to product code decoding techniques such as anchor decoding and erasure-marking. IGRAND also achieves lower complexity

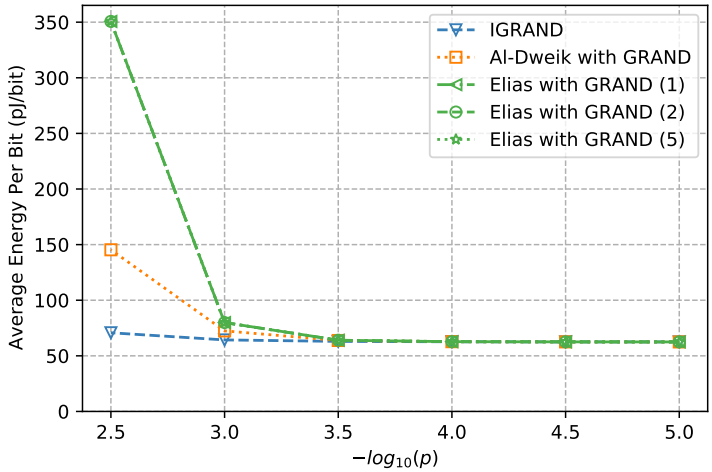


Figure 3.10: Estimated energy cost per bit of decoding a $\mathrm{BCH}(127, 106, 7)^2$ code using GRAND hardware.

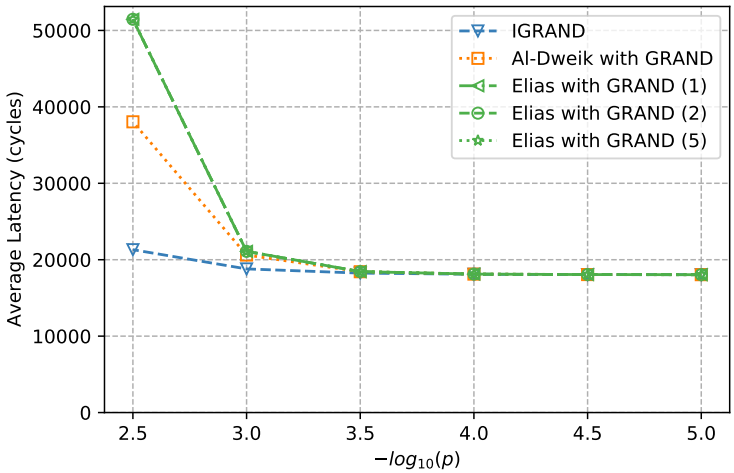


Figure 3.11: Estimated latency of decoding a $\mathrm{BCH}(127, 106, 7)^2$ code with GRAND hardware. The actual latency would be lower due to the hardware's pipelining design.

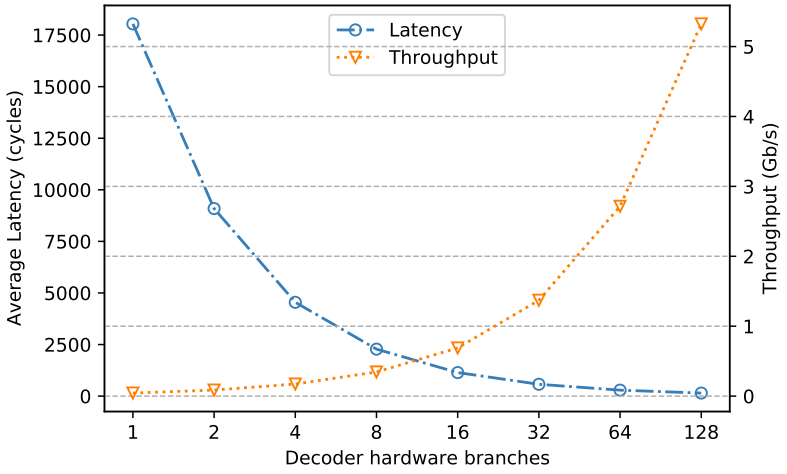


Figure 3.12: Estimated latency and throughput when IGRAND decodes a BCH(127,106,7)^2 code, with an increasing number of parallel decoding branches. Channel noise is fixed at $p = 10^{-5}$, which is the best case in terms of efficiency. Hardware is assumed to run at 70MHz. Full parallelisation is attained at 127 branches.

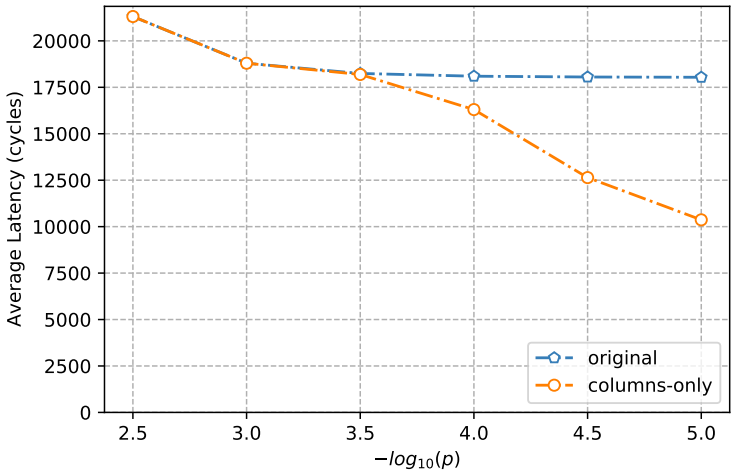


Figure 3.13: IGRAND decoding latency of a BCH$(127, 106, 7)^2$ product code when a columns-only decoding strategy is applied.

	Energy (pJ/bit)		**Latency (cycles)**	
W	Average	Worst	Average	Worst
0	31	31	71	71
1	42.3	50.5	110	142
2	1160	2260	2154	4238
3	26,700	45,940	16,869	29,500

Table 3.1: Average and worst case energy usage and latency of $n = 128$ bit GRAND chip. Quantities are conditioned on W, the Hamming weight of the underlying error pattern. Energy is per bit, and must be multiplied by n to derive the total. In the worst case, GRAND processes *all* noise patterns of weight W rather than half of them, in addition to all patterns of lower weight. This is the cost when GRAND exhausts all error patterns within the distance bound φ and then terminates.

than other GRAND-adapted iterative decoding algorithms, because GRAND is more efficient when the distance bound is kept low.

We have estimated that using a hardware implementation of GRAND to decode a 16,129-bit product code of rate 0.68, with a single decoding branch, would have a base cost of 1 µJ (62pJ/bit) and 18,000 cycles. Product code structure enables a trade-off between latency and hardware footprint, and we have shown that the fully-parallelised decoding of such codes, with a separate decoder for each component, would require as few as 142 cycles, at no extra energy cost. Columns-only decoding would halve this again to 71 cycles in low-noise channels, and we have shown that columns-only decoding adds negligible risk of a decoding mistake.

The components of a product code can be transmitted through heterogeneous coded systems as long as there is a corresponding decoder for each system. GRAND adds to this flexibility, allowing the outputs of all systems to be decoded using a single decoder. This applies even when the codes are picked at random, and to this end we have explored product codes with random linear component codes, which offer a broad selection of high-performing codes, though they underperform in terms of accuracy at the code lengths and rates considered here.

Code	Length	Rate	Decoder	BER 10^{-3}	BER 10^{-5}
BCH$(31, 21, 5)^2$	961	0.46	IGRAND	4.16 dB	5.04 dB
"	"	"	Al-Dweik	4.73 dB	5.59 dB
"	"	"	Elias	5.12 dB	6.12 dB
BCH$(63, 51, 5)^2$	3969	0.65	IGRAND	4.1 dB	4.61 dB
"	"	"	Al-Dweik	4.56 dB	5.06 dB
"	"	"	Elias	4.86 dB	5.47 dB
RLC$(128, 114)^2$	16384	0.79	IGRAND	5.4 dB	6.36 dB
CAPOLAR$(128, 112)^2$	"	0.77	"	4.45 dB	5.52 dB
CRC$(65, 53, 5)^2$	4225	0.66	"	4.12 dB	4.61 dB
CRC$(127, 113, 5)^2$	16129	0.79	"	4.52 dB	4.78 dB
eBCH$(128, 113, 6)^2$	16384	0.78	"	4.17 dB	4.52 dB
BCH$(63, 45, 7)^2$	3969	0.51	"	3.76 dB	4.05 dB

Table 3.2: Summary of results from this chapter. Shows the E_b/N_0 at which BERs of 10^{-3} and 10^{-5} were achieved for various combinations of code and decoder. Linear interpolation between measured datapoints (or, if necessary, extrapolation) was used to estimate the exact E_b/N_0.

CHAPTER 4

Block turbo decoding with ORBGRAND

Block turbo decoding is an iterative soft-input soft-output decoding algorithm for product codes. Similar to hard-input decoding of product codes, it requires a decoder for each component code, with the difference that they must be soft-input decoders that output a list of codewords. In this chapter it is established that, through the use of list decoding, soft-input variants of GRAND can replace the Chase algorithm as the component decoder in the turbo decoding of product codes. In addition to being able to decode arbitrary product codes, rather than just those with dedicated hard-input component code decoders, turbo decoding results show that ORBGRAND achieves a coding gain of up to 0.7dB over the Chase algorithm with an equal list size. A summary of the decoding accuracy results is presented in Table 4.1. Parts of this chapter have been published in [38].

4.1 Introduction

Product codes, as already described in Section 2.5, are a class of long, high-redundancy codes that are constructed by concatenating shorter component codes. The IGRAND algorithm was established in the previous chapter, adapting GRAND for accurate hard-input iterative decoding of arbitrary product codes.

Here we establish that GRAND can decode product codes with the aid of soft information, which enables much greater accuracy than with hard information alone. This is achieved through block turbo decoding, a technique introduced by Pyndiah in the 1990s [65]. Block turbo decoding achieves near-optimal soft-input decoding of product codes by using a soft-input component decoder, customarily the Chase algorithm [18], to generate a selection of candidate decodings for a row or column of the product code. The soft information for each bit in that row or column is then updated using a heuristic devised by Pyndiah.

To achieve turbo decoding with GRAND, we reconsider soft-input GRAND as a list decoding algorithm and use it to replace the Chase algorithm. The decoding list produced by GRAND can be used to update the bit reliabilities as before, with the benefit that GRAND's code-agnosticism allows it to turbo decode any product code. We present results for block turbo decoding with ORBGRAND, a soft-input variant of GRAND that is particularly suited to hardware implementation [5, 69]. We also provide analytical support for list decoding with GRAND algorithms, and consider the standalone list decoding performance of ORBGRAND.

4.2 Background

4.2.1 List decoding

List decoding [33] is a decoding procedure in which the decoder outputs a list of L codewords. The study of list decoding has been reinvigorated in recent years by the success of polar codes [12], which have been incorporated into the control plane protocol of the 5G New Radio standard [1] and which are typically decoded using successive cancellation list decoding [82].

As explored in Chapter 2, a GRAND algorithm achieves maximum-likelihood decoding when its guesses are in order of decreasing noise effect probability. ORBGRAND uses an approximate order that is almost optimal in low-SNR conditions, but sub-optimal at higher SNR. While the basic model used by ORBGRAND is accompanied by an improved statistical model that results in a better guessing order for low-noise conditions [25], other approaches have also been investigated. In particular, in a form of list decoding, Abbas et al. [3] proposed an extension to

basic ORBGRAND in which, after the first codeword has been identified, further codewords are accumulated that have a logistic weight within some distance of the first one. The most likely codeword in the resulting list is then selected as the hard output. This was shown to improve accuracy in low-noise channels.

4.2.2 Turbo decoding

Berrou et al. [15] introduced the turbo decoding technique for convolutional codes in 1993. Turbo decoding gradually converges to the correct decoding output by refining soft information over multiple iterations. Pyndiah [65] extended this idea from convolutional codes to product codes. As envisioned by Pyndiah, soft output from row decoding should inform the soft input of the column decoding, and vice versa.

To do this required a method of extracting soft output from the decoding of an individual component. While the Viterbi algorithm can be adapted to provide soft output from the decoding of convolutional codes [66], no such general method existed for block codes, and this was where Pyndiah made an insightful contribution. Pyndiah's idea was to use a list decoder to produce a list of codewords. The soft output could then be based on the relative distances of these codewords to the received signal. If, when modulated, the most likely codeword in the list is close to the received signal and the second-most-likely codeword is far, then the soft output should indicate that the decoding is very reliable. If, when modulated, the most likely and second-most-likely codewords are about equally far from the received signal, then the decoding is not reliable.

To produce a decoding list, Pyndiah proposed using the Chase [18] algorithm. The Chase algorithm accepts as input a binary sequence $x^n \in \{0,1\}^n$ and associated soft information indicating the reliability of each bit. It produces a list of codewords $\{u^{n,1}, u^{n,2}, \ldots, u^{n,2^\rho}\}$ by trying all combinations of bit flips of the ρ least reliable bits. A hard-input decoder $\mathcal{D} : \{0,1\}^n \to \mathcal{C}$ then maps each of these sequences to a codeword, resulting in a decoding list $\{\mathcal{D}(u^{n,1}), \mathcal{D}(u^{n,2}), \ldots, \mathcal{D}(u^{n,2^\rho})\}$, which may include duplicates.

In Pyndiah's block turbo decoding algorithm, the decoding of a single component

(row or column) of the product code, with accompanying per-bit soft channel output $\mathbf{R} \in \mathbb{R}^n$, goes as follows:

1. Apply Chase decoding to produce a list of codewords $\mathscr{L} \subseteq \mathcal{C}$, where $\mathcal{C}$ is the codebook of the component code and $1 \leq |\mathscr{L}| \leq 2^\rho$ for some small positive integer ρ.

2. Select $\mathbf{D} \in \operatorname{argmin}_{\mathbf{C} \in \mathscr{L}} |\mathbf{R} - \mathbf{C}|^2$ as the new value of the component, where $|\mathbf{R} - \mathbf{C}|^2$ is the Euclidean distance between $\mathbf{R}$ and the modulated form of $\mathbf{C}$.

3. Update the soft information of each bit in the component. If there is a codeword that disagrees with $\mathbf{D}$ on the value of the i-th bit, $\mathbf{C}^* \in \operatorname{argmin}_{\{\mathbf{C} \in \mathscr{L} : \mathbf{C}_i \neq \mathbf{D}_i\}} |\mathbf{R} - \mathbf{C}|^2$, then the soft output for that bit is given by

$$r_i = \frac{(2\mathbf{D}_i - 1)(|\mathbf{R} - \mathbf{C}^*|^2 - |\mathbf{R} - \mathbf{D}|^2)}{4}.$$

4. If $\operatorname{argmin}_{\{\mathbf{C} \in \mathscr{L} : \mathbf{C}_i \neq \mathbf{D}_i\}} |\mathbf{R} - \mathbf{C}|^2$ is empty, then instead use $r_i = \beta$ for some constant $\beta \geq 0$.

Only the next-nearest codeword that disagrees with the decision $\mathbf{D}$ is used to determine the reliability, because comparing $\mathbf{D}$ to all codewords in the codebook is impractical. Step (4) is a further approximation: if Chase decoding does not identify any codeword that disagrees with $\mathbf{D}$, then the nearest one is assumed to be far away and the decision is considered reliable. The parameter β, introduced in step (4), should increase with the number of iterations, since, intuitively speaking, the decoding should converge and become more reliable as more iterations are completed.

The final detail of the block turbo decoding algorithm is how to calculate the soft input for each iteration. Let R denote the $n \times n$ matrix of soft information for an $n \times n$ product code, with each value in the matrix corresponding to a single demodulated bit. The extrinsic information extracted from the decoding process up to the m-th iteration is denoted by the matrix $W^{(m)}$. The soft input for the m-th iteration is

$$R^{(m)} = R + \alpha^{(m)} W^{(m)},$$

where $\alpha^{(m)}$ is a so-called damping factor that weighs the extrinsic information versus the original channel output, and that should increase with the number of iterations to reflect increasing confidence in the decoding outcome. Given that the m-th iteration produces soft output $S^{(m)}$ by following the steps detailed above, the extrinsic information for the $m+1$-th iteration becomes

$$W^{(m+1)} = S^{(m)} - R^{(m)}.$$

Since the introduction of Pyndiah's block turbo decoding algorithm, various attempts have been made to improve its accuracy and efficiency. For instance, accuracy can be improved slightly by dynamically computing the value of the β parameter based on the soft input to each iteration [53]. Mahran et al. [62] propose adapting the Chase algorithm so that candidate codewords outside a certain Hamming radius from the hard channel output are discarded, improving both accuracy and complexity. They also propose [61] to reduce the number of hard decoding operations by not, in the test sequence generation phase of the Chase algorithm, flipping any bits that meet a reliability criteria. Alternative list decoding algorithms besides Chase have also been investigated [23, 81], including Kaneko's algorithm [52], which takes a different approach to generating test sequences for hard-input decoding. Most recently, turbo decoding has been applied to staircase codes [84], a generalisation of product codes that achieve superior decoding accuracy.

4.3 List decoding with GRAND

Here we provide analytical support for list decoding with any GRAND algorithm, based on theorems from [27]. In GRAND's original formulation, it sequentially guesses possible decodings from most to least likely and stops after identifying the first, and thus most likely, codeword that it encounters. Instead of stopping, GRAND can continue this guessing procedure and accumulate codewords until it has a list of the desired size L. This new procedure is described in Algorithm 4.

Algorithm 4: GRAND list decoding of hard channel output y, possibly with soft output r informing the likelihood of noise effects. Given a codebook $\mathcal{C}$, code length n and a target list size L.

$\mathscr{L} \leftarrow \{\}$;
$z^* \leftarrow 0^n$; `// all-zero is most likely`
while $|\mathscr{L}| < L$ **do**
  $c^* \leftarrow y \ominus z^*$; `// undo noise effect`
  if $c^* \in \mathcal{C}$ **then**
    add c^* to $\mathscr{L}$;
  end
  $z^* \leftarrow$ next most likely noise effect;
end
return $\mathscr{L}$;

From an analytical perspective, in Section 2.4 it was explored how GRAND's guessing procedure is a race between two random variables: the number of guesses $G(N^n)$ to find the true channel noise effect $N^n : \Omega \to \{0,1\}^n$, and the number of guesses $U^n : \Omega \to \{1, ..., 2^n\}$ before GRAND identifies an incorrect codeword. GRAND identifies the correct decoding when $G(N^n) < U^n$.

We now examine the case of list decoding with GRAND, and its approximate complexity. Consider a random binary code of length n with 2^k codewords. For list size $L = 2^l$, denote the position of the i-th incorrect codeword in GRAND's guessing order by the random variable $U_i : \Omega \to \{1, ..., 2^n\}$, where $1 \leq i \leq 2^k - 1$. As the codebook is constructed uniformly at random, the $\{U_i\}$ appear uniformly in the guesswork order $\{1, ..., 2^n\}$. Let the codewords be ordered such that $U_1 < U_2 < ... < U_{2^k-1}$, let $Y^n : \Omega \to \{0,1\}^n$ be the hard channel output and let $\mathbf{C}_i \in \mathcal{C}$ be the i-th codeword. Then $U_i = G(Y^n \oplus \mathbf{C}_i)$, since $\mathbf{C}_i = Y^n \ominus (Y^n \oplus \mathbf{C}_i)$. The total number of guesses to accumulate L codewords is $\Upsilon_L = U_1 + \sum_{i=2}^{L}(U_i - U_{i-1})$.

The expected total number of guesses $\mathbb{E}[\Upsilon_L]$ is derived as follows. Since

$$\begin{aligned}\mathbb{E}[U_1] &= \sum_u \mathbb{E}[U_1|U_2 = u]P(U_2 = u) \\ &= (1/2)\sum_u uP(U_2 = u) \\ &= \frac{\mathbb{E}[U_2]}{2},\end{aligned}$$

the expected guesses from the first codeword to the second is

$$\begin{aligned}\mathbb{E}[U_2 - U_1] &= \mathbb{E}[U_2] - \mathbb{E}[U_1] \\ &= \frac{\mathbb{E}[U_2]}{2} \\ &= \mathbb{E}[U_1].\end{aligned}$$

A similar argument proves that $\mathbb{E}[U_i - U_{i-1}] = \mathbb{E}[U_1]$ for all i, and $\mathbb{E}[2^n - U_{2^k-1}] = \mathbb{E}[U_1]$, which is the expected number of guesses from the final codeword to the last binary sequence that GRAND can guess; thus, $\mathbb{E}[\Upsilon_L] = L\mathbb{E}[U_1]$. The expected number of guesses to cover all 2^n possible noise effects is

$$\mathbb{E}\left[\sum_{i=1}^{2^k} U_i - U_{i-1}\right] = 2^k\mathbb{E}[U_1] = 2^n,$$

where $U_0 = 0$ and $U_{2^k} = 2^n$. Hence, $\mathbb{E}[U_1] = 2^{n-k}$ and

$$\mathbb{E}[\Upsilon_L] = L\mathbb{E}[U_1] = 2^{n-k+l}.$$

The above argument informs the choice of list size and code rate in coding scheme design. As n becomes large, $G(N^n) \leq 2^{nH}$ with high likelihood, where H is the Shannon entropy of the channel noise [20]. The correct codeword ends up on the decoding list with high likelihood when $G(N^n) < \mathbb{E}[\Upsilon_L]$, which is true when $2^{nH} < \mathbb{E}[\Upsilon_L] = 2^{n-k+l} = 2^{n(1-R)+l}$. Letting $l = n\theta$ for $\theta > 0$, the requirement

becomes $2^{nH} < 2^{n(1-R+\theta)}$, or $H < 1 - R + \theta$. Stated in a form closer to the noisy-channel coding theorem [78], $R < 1 - H + \theta$. This tells us that by increasing the list size we can perform effective channel coding at higher code rates, as asserted in [33].

Regarding complexity, 2^{n-k+l} is an upper bound on the expected number of GRAND queries for random codebooks. From this arises a design trade-off between list size and the number of parity bits. To keep the complexity bound constant, a parity bit must be removed if the list size is doubled. The bound corresponds to the expected number of queries to identify L incorrect codewords, although in practice the correct codeword will typically be added to the list after a small number of queries and fewer overall queries will be required as a result.

4.4 Turbo decoding with GRAND

4.4.1 Description and complexity

A soft-input list decoding variant of GRAND can replace the Chase algorithm in step (1) of the block turbo decoding algorithm, with the remainder of the algorithm untouched. Indeed, a variant of ORBGRAND has independently been proposed [22] for use in Pyndiah-style soft-input soft-output iterative decoding of OFEC codes, another form of concatenated code. Iterative soft-input soft-output GRAND decoding has also recently been considered in [74].

We have described how 2^{n-k+l} is an upper bound on the expected number of GRAND queries for list decoding of a random n-bit code with list size $L = 2^l$. This leads to a bound on the average decoding complexity of block turbo decoding with GRAND. Given a product code whose row and column codes are random codes of length n with k information bits, and given a maximum of N decoding iterations where $2n$ component codes are decoded in each iteration, an upper bound on the average number of GRAND codebook queries during block turbo decoding is $nN2^{n-k+l+1}$.

An advantage of GRAND as a component decoder is that it can decode any component code, and thus can turbo decode any product code. The Chase algorithm

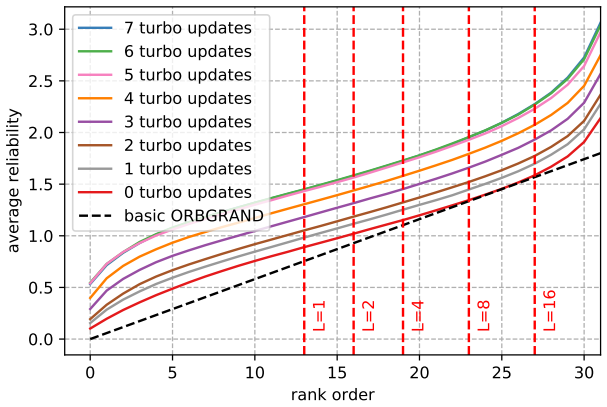


Figure 4.1: The average distribution of rank-ordered soft input throughout turbo decoding of an eBCH$(32, 26, 4)^2$ product code with a Chase component decoder, where $E_b/N_0 = 4$dB and $\rho = 4$. The distribution converges around the 5th update. Also shown is a linear model, as might be used by basic ORBGRAND, which becomes less capable of fitting the reliablility distribution as turbo decoding progresses. The vertical red lines indicate the bit of highest rank that would, on average, be flipped during the guessing process for a given list size L. At $L = 4$, only the top 20 least reliable bits are ever flipped on average.

requires that a specialised hard-input decoder exists for the component codes, which is not the case for RLCs and CRC codes. GRAND also distinguishes itself by populating its list with codewords in maximum-likelihood order, assuming its query order is correct, while Chase makes no such guarantees and may output duplicate codewords.

4.4.2 1-line ORBGRAND for turbo decoding

ORBGRAND is a practical component decoder for turbo decoding, given its accuracy and efficiency. As turbo decoding converges, however, the distribution of reliability values shifts upwards, as in Figure 4.1, which tends to make basic ORBGRAND's linear approximation a poorer fit. We thus propose to turbo decode with the full ORBGRAND algorithm, parameterised to use a single line. This enables ORBGRAND's model to have a non-zero intercept and so better capture the reliability distribution during later iterations of turbo decoding.

Figure 4.1 also indicates that, on average, the most reliable bits will never even be tried as errors in GRAND's guessing process, and so it is not critical to model their reliability values accurately. This is possible to determine because, as already established, GRAND makes no more than 2^{n-k+l} queries on average for list size 2^l. The number of noise guesses at a particular logistic weight w_L corresponds to the number of partitions of the integer w_L, which can be counted using the landslide algorithm. If this count is denoted $g(w_L)$, then, on average, the highest rank of any bit that will be considered a potential error by GRAND will be roughly $\max\{w_L \in \mathbb{Z}^+ : \sum_{m=1}^{w_L} g(m) \leq 2^{n-k+l}\}$, where $\mathbb{Z}^+$ denotes the positive integers.

In Algorithm 5, we present a simple method to construct noise sequences for 1-line ORBGRAND that has similar implementation complexity as basic ORBGRAND. Noise sequences are generated in order of their total weight, w_T. For each w_T, we iterate over all pairs of non-negative integers (w_H, w_L) such that $w_T = cw_H + w_L$, where c is the integer parameter that captures the slope and y-axis intercept of the line, as described in Section 2.4. The landslide algorithm [25] then generates all noise sequences for each pair. Parity constraints [72] may be used to halve the number of codebook queries performed by this algorithm. For even-parity codes, such as eBCH codes, the parity of the noise effect z^n can be inferred from y^n, since, given codeword c^n, $\sum_{i=0}^{n} y_i^n = \sum_{i=0}^{n}(c_i^n + z_i^n) = \sum_{i=0}^{n} c_i^n + \sum_{i=0}^{n} z_i^n \equiv \sum_{i=0}^{n} z_i^n \pmod 2$, which is the parity of the noise effect. Incorrect-parity Hamming weights can then be skipped, avoiding entire branches of noise effect guesses.

We now introduce a simple and effective method to pick the parameter c for a particular instantiation $\{r_i\}$ of sorted reliability values. Fitting the line by regression is inappropriate, since it would give equal weight to each point (i, r_i) when in fact the least reliable bits are the most important to accurately approximate; as explained previously, at practical list sizes, the most reliable bits are rarely even considered as potential errors. For this reason, we propose to fit a line through the points $(1, r_1)$ and $(\lfloor n/2 \rfloor, r_{\lfloor n/2 \rfloor})$, where n is the code length. Then the slope of the line is $\gamma = (r_{\lfloor n/2 \rfloor} - r_1)/(\lfloor n/2 \rfloor - 1)$ and $c = \max(0, [(r_1 - \gamma)/\gamma])$, where $[.]$ rounds to the nearest integer. This gives the best estimate of $(1, r_1)$, which is the least reliable and thus most significant point, and accurately approximates the remaining points if they follow a line-like distribution as in Figure 4.1.

Algorithm 5: Noise effect generation algorithm for 1-line ORBGRAND, given integer parameter $c \geq 0$ and code length n.

yield 0^n; // all-zero is most likely
$w_T \leftarrow c+1$; // minimum possible weight
while $w_T \leq cn + \frac{n(n+1)}{2}$ **do**
  $w_H \leftarrow \max(1, \lceil \frac{1+2(n+c)-\sqrt{(1+2(n+c))^2-8w_T}}{2} \rceil)$;
  while $w_H \leq n$ **do**
    $w_L \leftarrow w_T - cw_H$;
    if $w_L < \frac{w_H(w_H+1)}{2}$ **then**
      break ; // invalid pair
    end
    yield noise effects generated by `Landslide`(w_H, w_L, n);
    $w_H \leftarrow w_H + 1$;
  end
  $w_T \leftarrow w_T + 1$;
end

4.5 Performance evaluation

We begin with a study of ORBGRAND's list decoding performance, since this is critical to its performance as a turbo component decoder. Then we evaluate the accuracy and complexity of ORBGRAND as a turbo component decoder. We run simulations using an AWGN channel model with BPSK modulation.

4.5.1 List decoding

Figure 4.2 shows the list decoding BLER of basic ORBGRAND versus that of Chase decoding for an extended BCH code [57], eBCH$(32, 26, 4)$, which is one of the component codes from [65]. A list decoding block error occurs when, given transmitted codeword $\mathbf{C}$ and output decoding list $\mathscr{L}$, $\mathbf{C} \notin \mathscr{L}$. The list size L ranges from 4 to 16 and $L = 2^l$ corresponds to a Chase parameter of $\rho = l$. At a BLER of 10^{-5} and $L = 16$, basic ORBGRAND provides a coding gain of 1 dB over Chase. That ORBGRAND outperforms Chase as a list decoder is a promising indicator of its potential as a turbo component decoder.

Figure 4.3 shows the basic ORBGRAND list decoding BLER of a BCH code, BCH$(31, 21, 5)$, versus a random linear code, RLC$(31, 21, 4)$. The RLC was pur-

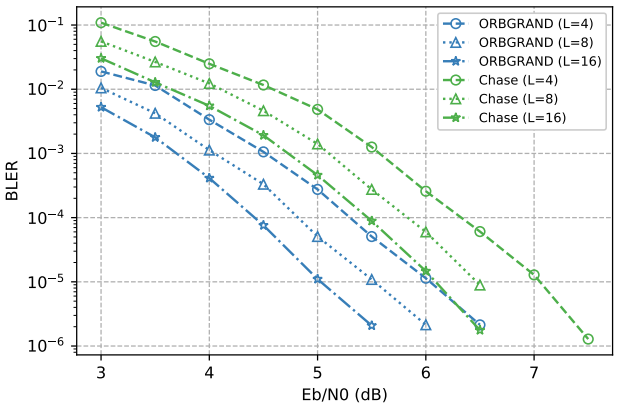


Figure 4.2: List decoding BLER of an eBCH$(32, 26, 4)$ code with ORBGRAND and Chase as list decoders, and list size L. ORBGRAND consistently provides a coding gain over Chase, even with smaller list size.

posely constructed to have a lower minimum distance. Its decoding is only enabled by ORBGRAND's code-agnosticism. As the list size increases, the performance of the two codes converges, which is consistent with Elias's claim that a larger list size should compensate for structural weakness in a code [33]. While this suggests that powerful product codes can be constructed from imperfect component codes, later results will demonstrate that this is not true.

Figure 4.4 shows the list decoding accuracy of a further selection of codes with basic ORBGRAND: eBCH$(32, 26, 4)$, RLC$(32, 26, 3)$ and CRC$(32, 26, 3)$. The CRC polynomial is 0x33 in Koopman notation [55]. The list size ranges from 4 to 16. Performance is essentially equivalent at these list sizes, despite the RLC and CRC having lower minimum distance.

4.5.2 Turbo decoding accuracy

Figure 4.5 and Figure 4.6 show the BER of eBCH$(32, 26, 4)^2$ and eBCH$(64, 57, 4)^2$ product codes with block turbo decoding. 1-line ORBGRAND and Chase are used as component decoders. These codes were tested in [65], and, as in that paper, we run turbo decoding for 4 iterations and use the same set of values for Pyndiah's α and β parameters. The list sizes are 4, 8 and 16. Decoding is halted if the rows

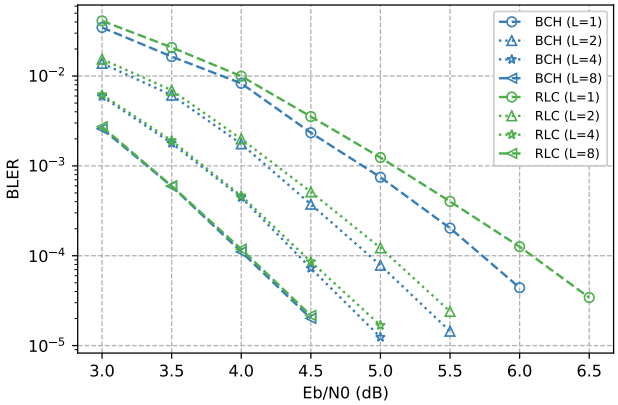


Figure 4.3: List decoding BLER of a BCH$(31, 21, 5)$ code and an RLC$(31, 21, 4)$ code, with basic ORBGRAND decoding and list size L. The RLC was purposely constructed to have a lower minimum distance, demonstrating that its performance converges to that of the BCH regardless of minimum distance.

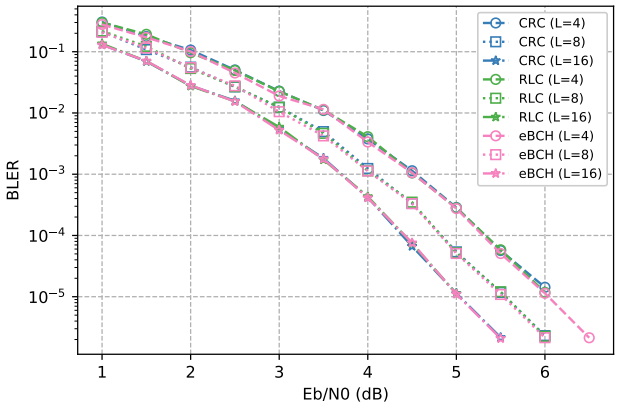


Figure 4.4: List decoding BLER of eBCH$(32, 26, 4)$, RLC$(32, 26, 3)$ and CRC$(32, 26, 3)$ codes, with basic ORBGRAND decoding.

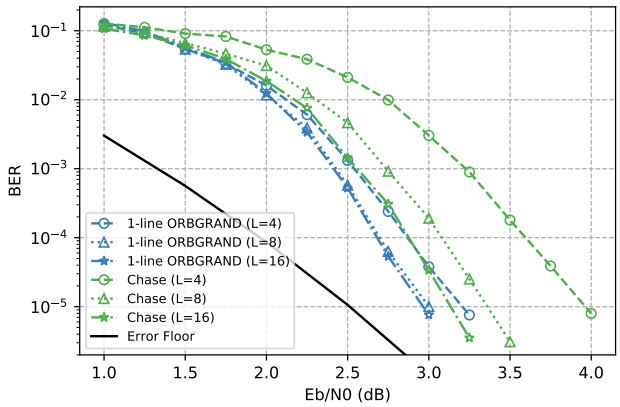


Figure 4.5: BER of an eBCH$(32, 26, 4)^2$ product code ($n = 1024, k = 676, d = 16$) with block turbo decoding. 1-line ORBGRAND and Chase are used as component decoders with list size L.

or columns are error-free at the beginning of any iteration. Also shown are the analytically-derived error floors of these product codes, taken from [19].

Figure 4.5 concerns an eBCH$(32, 26, 4)^2$ code ($n = 1024$, $k = 676$, $R = 0.66$). With $L = 16$, 1-line ORBGRAND provides a coding gain over Chase of approximately 0.15dB at a BER of 10^{-5}. Even with $L = 4$, 1-line ORBGRAND achieves nearly the same performance, whereas the performance of Chase degrades rapidly as the list size decreases. The performance of both decoding algorithms converges to the error floor as the channel becomes less noisy; a gap of 0.5dB remains between the error floor and 1-line ORBGRAND at a BER of 10^{-5}.

Figure 4.6 shows results for an eBCH$(64, 57, 4)^2$ code ($n = 4096$, $k = 3249$ $R = 0.79$). With $L = 16$, 1-line ORBGRAND provides a coding gain of approximately 0.2dB at a BER of 10^{-5}, and Chase performance again degrades more severely with decreasing list size. There remains about 1.5dB between 1-line ORBGRAND and the error floor at a BER of 10^{-6}.

Figure 4.7 shows the turbo decoding accuracy of product codes whose component codes are the same as in Figure 4.4, with 1-line ORBGRAND decoding. Despite having equivalent list decoding performance, the RLC and CRC fare worse than

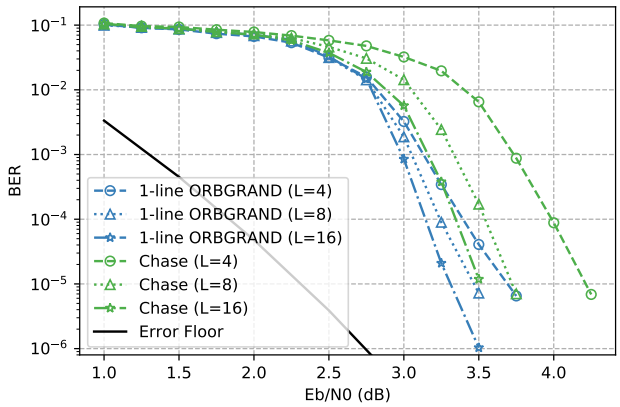


Figure 4.6: BER of an eBCH$(64, 57, 4)^2$ product code ($n = 4096, k = 3249, d = 16$) with block turbo decoding. Decoding parameters are the same as those described in Figure 4.5. With $L = 4$ the gain is 0.7 dB for BLER 10^{-4}.

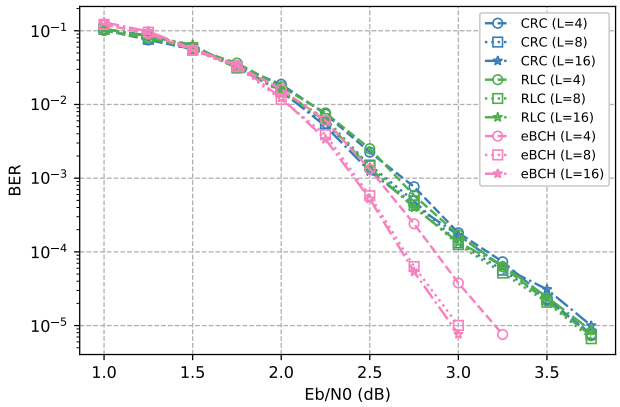


Figure 4.7: BER of CRC$(32, 26, 3)^2$, RLC$(32, 26, 3)^2$, and eBCH$(32, 26, 4)^2$ product codes with 1-line ORBGRAND turbo decoding, list size L.

the eBCH code as component codes, and so product codes appear to compound structural weakness in their component codes.

Figure 4.8 compares the decoding accuracy of IGRAND versus that of turbo decoding with 1-line ORBGRAND, using the eBCH$(32, 26, 4)^2$ product code already considered here and the BCH$(31, 21, 5)^2$ product code from the previous chapter.

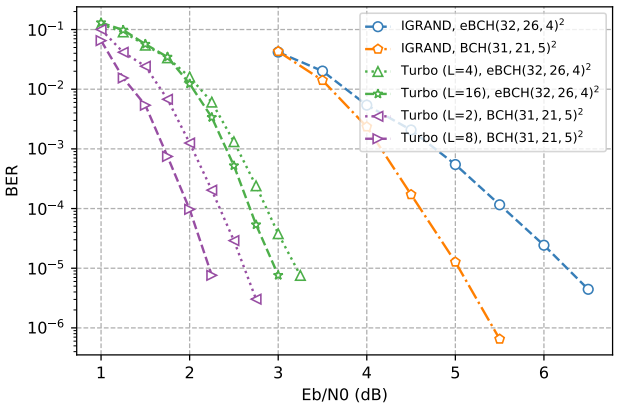


Figure 4.8: Accuracy of IGRAND compared with the accuracy of turbo decoding with 1-line ORBGRAND. The BERs of eBCH$(32, 26, 4)^2$ and BCH$(31, 21, 5)^2$ product codes are considered.

Both codes see a coding gain of 2-2.5dB with turbo decoding, which is typical of the gains offered by soft information [57].

4.5.3 Turbo decoding complexity

Figure 4.9 shows the average number of codebook queries, a standard measure of GRAND complexity [3, 25], performed by 1-line ORBGRAND with parity constraints [72] during turbo decoding of eBCH$(32, 26, 4)^2$ and eBCH$(64, 57, 4)^2$ product codes. The use of parity constraints to reduce complexity was possible because extended BCH codes enforce even parity on their codewords. Figure 4.10 shows the same data but as a ratio over the average codebook queries when $L = 4$. These results support the analysis of Section 4.3, which posited that the complexity should at most double when the list size doubles.

Finally, Figure 4.11 compares the complexity of IGRAND versus that of turbo decoding with 1-line ORBGRAND, using the same product codes from Figure 4.8. When decoding a row or column of a product code, IGRAND makes fewer codebook queries, since it halts the guessing process upon identifying the first codeword. In turbo decoding, the guessing must continue until a full list of codewords has been accumulated. Interestingly, however, turbo decoding of the eBCH$(32, 26, 4)^2$

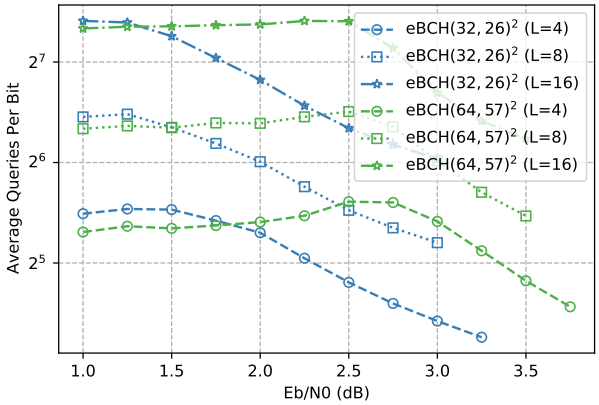


Figure 4.9: Average number of codebook queries per bit performed by 1-line ORBGRAND during turbo decoding of eBCH$(32, 26, 4)^2$ and eBCH$(64, 57, 4)^2$ product codes, conditioned on list size L.

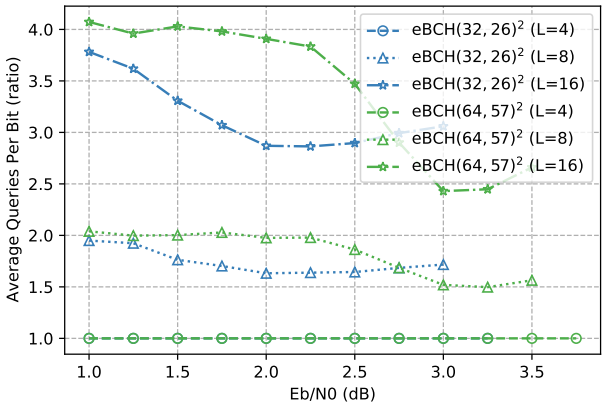


Figure 4.10: Average number of codebook queries per bit, as in Figure 4.9, except as a ratio over the average number of codebook queries per bit when $L = 4$. As predicted by the analysis in Section 4.3, doubling the list size leads to, at most, double the average number of codebook queries.

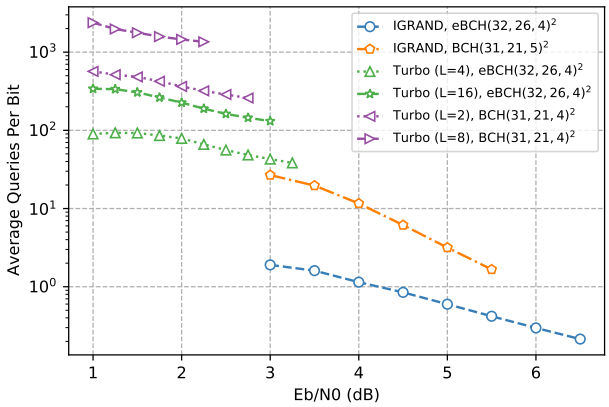


Figure 4.11: Average number of codebook queries per bit when IGRAND and turbo decoding with 1-line ORBGRAND are used to decode eBCH$(32, 26)^2$ and BCH$(31, 21, 5)^2$ product codes.

product code with list size $L = 4$ is shown to have roughly equivalent complexity to IGRAND decoding of the BCH$(31, 21, 5)^2$ product code, despite the former seeing a coding gain of about 1.5dB over the latter. This can be understood through the GRAND complexity bounds discussed throughout this chapter and Chapter 2. GRAND list decoding of the eBCH$(32, 26, 4)$ component code with list size $L = 2^l = 4$ has an upper bound on the average codebook queries of $2^{n-k+l} = 2^{32-26+2} = 2^8$, while the upper bound for regular GRAND decoding of the BCH$(31, 21, 5)$ code is $2^{n-k} = 2^{10}$. While these are upper bounds, they provide some intuition for the similarity of the complexity.

4.6 Discussion

GRAND algorithms can list decode accurately, and soft-input list decoding GRAND algorithms are a viable replacement for Chase as the component decoder in block turbo decoding. We have presented a code for which basic ORBGRAND list decoding gains as much as 1dB over Chase at a list BLER of 10^{-5}. For turbo decoding, the distribution of rank-ordered soft information shifts so that the full version of ORBGRAND is required for effective component decoding. Turbo decoding simulations show that 1-line ORBGRAND gains up to 0.7dB over Chase at a BER of 10^{-5} for two different product codes, an improvement in accuracy

that may be explained by the maximum-likelihood guarantees on GRAND's list decoding output. Turbo decoding with 1-line ORBGRAND is also shown to gain 2-2.5dB over the IGRAND algorithm, as is typical with the incorporation of soft information [57]. A gap still exists between turbo decoding performance and the error floors of the product codes considered here, suggesting that there is potential for further optimisation. One such optimisation would be to dynamically compute the β parameter during turbo decoding. Other improvements to the block turbo algorithm are discussed in [7, 17]. Performance may also be improved by parameterising the ORBGRAND component decoder to use more lines in its multiline model of the reliability distribution.

We have characterised the list and turbo decoding complexity of GRAND, and with the recent low-energy synthesised ORBGRAND chip [69], it will be possible to estimate the energy consumption of block turbo decoding with ORBGRAND. As with any component decoder, the choice of list size is a trade-off between accuracy and complexity. Another way to control complexity is to terminate GRAND's guessing procedure early and increase the reliability of all bits if it identifies a codeword within a small number of guesses, this idea being based on [53] and [22]. GRAND's universality allows list and turbo decoding to be applied to codes without bespoke decoders, such as CRCs, as well as product codes that are concatenations of those codes. This leads to the possibility of new channel coding applications.

Code	**Length**	**Rate**	**Decoder**	**BER 10^{-3}**	**BER 10^{-5}**
eBCH$(32, 26, 4)^2$	1024	0.66	Turbo-ORBGRAND	2.41 dB	2.97 dB
"	"	"	Turbo-Chase	2.56 dB	3.13 dB
eBCH$(64, 57, 4)^2$	4096	0.79	Turbo-ORBGRAND	2.99 dB	3.31 dB
"	"	"	Turbo-Chase	3.16 dB	3.51 dB
CRC$(32, 26, 3)^2$	1024	0.66	Turbo-ORBGRAND	2.55 dB	3.75 dB
RLC$(32, 26, 3)^2$	"	"	Turbo-ORBGRAND	2.56 dB	3.71 dB
eBCH$(32, 26, 4)^2$	"	"	IGRAND	4.76 dB	6.25 dB
BCH$(31, 21, 5)^2$	961	0.46	Turbo-ORBGRAND	1.71 dB	2.22 dB
"	"	"	IGRAND	4.16 dB	5.04 dB

Table 4.1: Summary of results from this chapter, comparing the E_b/N_0 at which BERs of 10^{-3} and 10^{-5} are achieved, as described in Table 3.2. Principally, this is a comparison between block turbo decoding with the Chase algorithm as a component decoder and with 1-line ORBGRAND as a component decoder, with the largest list size assumed (either $L = 8$ or $L = 16$). Some IGRAND results are included as well for convenience.

CHAPTER 5

Alternative soft output for GRAND

In this chapter we develop a method through which any soft-input GRAND algorithm can produce probabilistic soft output. To the best of our knowledge, no prior method exists to provide soft output for block codes without the added complexity of list decoding. Forney [34] and Pyndiah [65] introduced methods to compute block-level and bit-level soft output, but these methods rely on list decoding with a list size of at least 2. Our method, in contrast, computes soft output for decoding lists of any size, including a single codeword. Being probabilistic, the soft output we provide is easily interpretable and integrated with applications, unlike the list decoding approach of Pyndiah. It indicates the likelihood of a codeword, decoding list, or individual decoded symbol being correct. While Forney's method is probabilistic, it is conditioned on the true codeword being in the decoding list; our approach is not conditioned but includes an output for the likelihood that the codeword is not in the list, making it a true probability mass function and making the soft output more accurate. Implementing our approach adds negligible computation and memory overhead to GRAND. Among the many potential applications of the soft output, it permits tuning the balance between undetected errors and block errors for arbitrary moderate-redundancy codes, including CRCs. Simulation results explore the application of the method to error detection and establish that it is accurate for structured codes as well as random codes. Parts of this chapter have been published in [40].

5.1 Introduction

Soft output acts as a measure of confidence in the correctness of a decoded block. In the previous chapter we explored how GRAND algorithms can produce soft output by list decoding and then applying Pyndiah's approximate formula to the decoding list. List decoding, however, is computationally expensive, and so in this chapter we investigate an alternative and more analytical approach to calculating GRAND soft output.

Soft output is a generally desirable feature of error correction decoders. It can be used to make control decisions such as retransmission requests or to tag blocks as erasures for an erasure-correcting code to then rectify [57]. A common method of establishing a binary measure of decoding confidence is to append a CRC to a transmitted message [48, 76] prior to error correction encoding that can then be used post-decoding to assess consistency. When the block length is large, the addition of a CRC has a negligible effect on the code rate. One of the goals of modern communications standards such as 3GPP 5G [1], however, is ultra-reliable low-latency communications (URLLC), which requires the use of short packets [31]. The addition of a CRC to short packets has a significant effect on the code rate, and so alternative solutions to evaluate decoding confidence are a topic of active interest, e.g. [76].

In seminal work on error exponents, Forney [34] proposed an approximate computation of the correctness probability of a decoded block. Forney's approach, like Pyndiah's approach to per-bit soft output, requires the use of a list decoder, which significantly restricts its applicability. In addition to this constraint, we shall show that Forney's approximation provides an inaccurate estimate in channels with challenging noise conditions, which, due to the likely need for retransmission requests, are a primary area of interest. Despite these limitations, the potential utility of Forney's approximation has warranted further investigation, e.g. [49, 76], particularly with the recent introduction of CA-Polar codes and the associated CRC-assisted successive cancellation list (CA-SCL) list decoding algorithm [13, 56, 63, 82] to communications standards [1]. For convolutional or trellis codes, the Viterbi algorithm [35] can be modified to produce soft output at the

sequence level [66], which has been used in coding schemes with multiple layers of decoding [44] and to inform repeat transmission requests [66, 87], but this does not extend to block codes.

The soft output measure we develop for GRAND is an extremely accurate estimate of the a posteriori probability that a decoding is correct or, in the case of list decoding, the probability that the correct codeword is in the list. Based on the block-level probability, a per-bit error probability can be calculated as well. In contrast with the existing methods described above, this GRAND soft output can be readily used with any moderate-redundancy block code, can be evaluated without the need to list decode, and remains accurate in noisy channel conditions. We derive the probabilities for uniform at random codebooks and demonstrate empirically that the resulting formulae continue to provide accurate soft output for structured codebooks. The formulae can be used with any algorithm in the GRAND family so long as soft input is available. Calculating the soft output requires only that the code length and code rate are known, and that the probability of each noise effect query is accumulated during GRAND's normal operation. Thus, it does not increase the decoder's algorithmic complexity or memory requirements. In practical terms, the approach provides accurate soft output for hard-output or list-output decoding of any moderate-redundancy code of any length and any structure.

5.2 Background

We first define notation used in the rest of the chapter. Let $\mathcal{C}$ be a codebook containing 2^k binary codewords each of length n. Let $C^n : \Omega \to \mathcal{C}$ be a codeword drawn uniformly at random from the codebook and let $N^n : \Omega \to \{0,1\}^n$ denote the binary noise effect that the channel has on that codeword during transmission; that is, N^n encodes the binary difference between the demodulated received sequence and the transmitted codeword, rather than the potentially continuous channel noise. Then $Y^n = C^n \oplus N^n$ is the demodulated channel output, with $\oplus$ being the element-wise binary addition operator. Let $R^n : \Omega \to \mathbb{R}^n$ denote soft channel output. Lowercase letters represent realizations of random variables, with the exception of z^n, which is the realization of N^n.

5.2.1 GRAND

Here we revisit a few concepts of GRAND that were given a detailed treatment in Chapter 2, while introducing new notation. Also relevant is Algorithm 4, which describes the GRAND list decoding procedure for generating a decoding list of size L.

First recall that GRAND makes a series of noise effect guesses; in this chapter we denote the i-th of these noise guesses as $z^{n,i} \in \{0,1\}^n$. Also recall that underlying GRAND is a race between two random variables: the number of guesses until the true codeword is identified, and the number of guesses until an incorrect codeword is identified. The guesswork function $G : \{0,1\}^n \to \{1,\ldots,2^n\}$, which may be informed by soft information, maps a noise effect sequence to its position in GRAND's guessing order, so that $G(z^{n,i}) = i$. Thus $G(N^n)$ is a random variable that encodes the number of guesses until the transmitted codeword would be identified. If $W_{(i)} : \Omega \to \{1,\ldots,2^n-1\}$ is the number of guesses until the i-th incorrect codeword is identified, not accounting for the query that identifies the correct codeword, then GRAND returns a correct decoding whenever $G(N^n) \leq W_{(1)}$ and a list of length L containing the correct codeword whenever $G(N^n) \leq W_{(L)}$. Analysis of the race between these two processes leads to the derivation of the soft output.

5.2.2 Previous work on soft output

Forney's work on error exponents [34] resulted in an approximation for probabilistic soft output. Given channel output r^n and a maximum-likelihood decoding output $c^{n,*} \in \mathcal{C}$, the probability that the decoding is correct is

$$P(C^n = c^{n,*}|R^n = r^n) = \frac{P(R^n = r^n|C^n = c^{n,*})}{\sum_{c^n \in \mathcal{C}} P(R^n = r^n|C^n = c^n)}.$$

Based on this formula, Forney derived an optimal threshold for determining whether a decoding should be marked as an erasure. Computing the sum in the formula is infeasible for codebooks of practical size, so Forney suggested that, given the second most likely codeword, $c^{n,**} \in \mathcal{C}$, the correctness probability be approximated by

$$\frac{P(R^n = r^n | C^n = c^{n,*})}{P(R^n = r^n | C^n = c^{n,*}) + P(R^n = r^n | C^n = c^{n,**})}, \tag{5.1}$$

which is necessarily no smaller than 1/2. More generally, given a decoding list $\mathscr{L} \subseteq \mathcal{C}$, the denominator can be replaced by $\sum_{c^n \in \mathscr{L}} P(R^n = r^n | C^n = c^n)$ resulting in an estimate of the correctness probability that is no smaller than $1/|\mathscr{L}|$. Having the codewords of highest likelihood in the decoding list will give the most accurate approximation, as their likelihoods dominate the sum. The resulting approximate probability is conditioned on the true codeword being in the decoding list, which is not necessarily the case, and we will see through our approach how the soft output is improved by removing this conditioning. Another downside of Forney's approach is that it requires a list of codewords, which most decoders do not provide. For this reason, a method has recently been proposed to estimate the likelihood of the second most likely codeword given the first [36]. A variety of alternative schemes have also been suggested for making erasure decisions, a summary of which can be found in [47].

Specifically relating to GRAND, recent work by Sarieddeen et al. [74, 75] investigated how to produce improved per-bit soft output from crude soft input. Their approach used Euclidean distance metrics for each guessed word to inform an improved demodulation of the transmitted bits, enabling multiple iterations of decoding to improve accuracy. In contrast, the soft output method we propose here is based on probabilistic formulae and, crucially, incorporates the likelihood of all codewords in the decoding list, which, out of all the codewords in the codebook, have the greatest impact on the per-bit reliability. The probabilistic soft output is readily interpretable for applications such as error detection.

5.3 GRAND soft output

Throughout this section, we shall assume that the codebook, $\mathcal{C}$, consists of 2^k codewords drawn uniformly at random from $\{0,1\}^n$, although the derivation generalises to higher-order symbols. We first derive exact expressions, followed by readily computable approximations, for the probability that the transmitted codeword is not in the GRAND decoding list and, as a corollary, that a single-codeword

GRAND output is incorrect. In Section 5.5 we demonstrate that the formulae provide excellent estimates for structured codebooks.

Theorem 1 (A posteriori likelihood of an incorrect GRAND list decoding for a uniformly random codebook)**.** *Let $G(N^n)$ be the number of codebook queries until the noise effect sequence N^n is identified. Let $W_1, \ldots, W_{2^k-1}$ be selected uniformly at random without replacement from $\{1, \ldots, 2^n - 1\}$ and define their rank-ordered version $W_{(1)} < \cdots < W_{(2^k-1)}$. With the true noise effect not counted, $W_{(i)}$ corresponds to the location in the guesswork order of the i-th erroneous decoding in a codebook constructed uniformly-at-random. Define the partial vectors $W_{(i)}^j = (W_{(i)}, \ldots, W_{(j)})$ for each $i \leq j \in \{1, \ldots, 2^k - 1\}$*

Assume that a list of $L \geq 1$ codebooks are identified by a GRAND decoder at query numbers $q_1 < \ldots < q_L$. Define the associated partial vectors $q_i^j = (q_i, \ldots, q_j)$ for each $i \leq j \in \{1, \ldots, 2^k - 1\}$, and

$$q_1^{L,\{i\}} = (q_1, \ldots, q_{i-1}, q_{i+1} - 1, \ldots, q_L - 1), \tag{5.2}$$

which is the vector q_1^L but with the entry q_i omitted and one subtracted for all entries from q_{i+1} onwards. Define

$$P(A) = P(G(N^n) > q_L) P(W_{(1)}^L = q_1^L),$$

which is associated with the transmitted codeword not being in the list, and, for each $i \in \{1, \ldots, L-1\}$,

$$P(B_i) = P(G(N^n) = q_i) P(W_{(1)}^{L-1} = q_1^{L,\{i\}}),$$

which is associated with the transmitted codeword being the i-th element of the list, and

$$P(B_L) = P(G(N^n) = q_L) P(W_{(1)}^{L-1} = q_1^{L-1}, W_{(L)} \geq q_L),$$

which is associated with the transmitted codeword being the final element of the list. Then the probability that the correct decoding is not in the list is

$$\frac{P(A)}{P(A) + \sum_{i=1}^{L} P(B_i)}. \tag{5.3}$$

Proof. For $q \in \{1, \ldots, 2^n\}$, define $W_{(i),q} = W_{(i)} + 1_{\{W_{(i)} \geq q\}}$, so that any $W_{(i)}$ that is greater than or equal to q is incremented by one. Note that $W_{(i),G(N^n)}$ encodes the locations of erroneous codewords in the guesswork order of a randomly constructed codebook given the value of $G(N^n)$ and, in particular, $W_{(i),G(N^n)}$ corresponds the number of queries until the i-th incorrect codeword is found given $G(N^n)$.

We identify the event that the decoding is not in the list as

$$A = \left\{G(N^n) > q_L, W_{(1)}^L = q_1^L\right\}$$

and the events where the decoding is the i-th element of the list by

$$\begin{aligned} B_i = \Big\{ & W_{(1)}^{i-1} = q_1^{i-1}, G(N^n) = q_i, \\ & W_{(i)}^{L-1} + 1 = q_{i+1}^L, W_{(L)} \geq q_L \Big\} \end{aligned}$$

where the final condition is automatically met for $i = \{1, \ldots, L-1\}$ but not for $i = L$. The conditional probability that a GRAND decoding is not one of the elements in the list given that L elements have been found is

$$P\left(A \middle| A \bigcup_{i=1}^{L} B_i\right) = P(A) \Big/ P\left(A \bigcup_{i=1}^{L} B_i\right). \tag{5.4}$$

As all of the A and B_i events are disjoint, to compute eq. (5.4) it suffices to simplify $P(A)$ and $P(B_i)$ for $i \in \{1, \ldots, L\}$ to evaluate the a posteriori likelihood that the transmitted codeword is not in the list.

Consider the numerator,

$$\begin{aligned} P\left(A\right) &= P(G(N^n) > q_L, W_{(1)}^L = q_1^L) \\ &= P(G(N^n) > q_L) P(W_{(1)}^L = q_1^L), \end{aligned}$$

where we have used the fact that $G(N^n)$ is independent of $W_{(1)}^L$ by construction. In considering the denominator, we need only be concerned with the terms $P(B_i)$ corresponding to a correct codebook being identified at query q_i, for which

$$\begin{aligned} P(B_i) =& P(G(N^n) = q_i, \\ & W_{(1)}^{i-1} = q_1^{i-1}, W_{(i)}^{L-1} + 1 = q_{i+1}^L, W_{(L)} \geq q_L) \\ =& P(G(N^n) = q_i, W_{(1)}^{L-1} = q_1^{L,\{i\}}, W_{(L)} \geq q_L) \\ =& P(G(N^n) = q_i) P(W_{(1)}^{L-1} = q_1^{L,\{i\}}, W_{(L)} \geq q_L), \end{aligned}$$

where we have used the definition of $q_1^{L,\{i\}}$ in eq. (5.2) and the independence. Thus the conditional probability that the correct answer is not found in eq. (5.4) is given in eq. (5.3). □

Specializing to a list size $L = 1$, the formula in eq. (5.3) for the a posteriori likelihood that decoding is incorrect can be expressed succinctly, as presented in the following corollary.

Corollary 1 (A posteriori likelihood of an incorrect GRAND decoding for a uniformly random codebook)**.** *The conditional probability that a GRAND decoding is incorrect given a codeword is identified on the q-th query is*

$$\frac{P(G(N^n) > q)P(W_{(1)} = q)}{P(G(N^n) = q)P(W_{(1)} \geq q) + P(G(N^n) > q)P(W_{(1)} = q)}.$$

where $W_{(1)}$ is equal in distribution to the minimum of $2^k - 1$ numbers selected uniformly at random without replacement from $\{1, \ldots, 2^n - 1\}$.

In order to compute the a posteriori probability of an incorrect decoding in Theorem 1, we need to evaluate or approximate: 1) $P(G(N^n) = q)$ and $P(G(N^n) \leq q)$; and 2) $P(W_{(1)}^L = q_1^L)$ and $P(W_{(1)}^{L-1} = q_1^{L-1}, W_{(L)} \geq q_L)$. During a GRAND algorithm's execution, the precise evaluation of 1) can be achieved by calculating the likelihood of each noise effect query as it is made, $P(G(N^n) = q) = P(N^n = z^{n,q})$, and retaining a running sum, $P(G(N^n) \leq q) = \sum_{j=1}^{q} P(N^n = z^{n,j})$. For 2), geometric approximations whose asymptotic precision can be verified using the approach described in [27][Theorem 2] can be employed, resulting in the following corollaries for list decoding and single-codeword decoding, respectively.

Corollary 2 (Approximate a posteriori likelihood of an incorrect GRAND list decoding for a uniformly random codebook)**.** *If each $W_{(i)}$ given $W_{(i-1)}$ is assumed to be geometrically distributed with probability of success $(2^k - 1)/(2^n - 1)$, eq. (5.3) describing the a posteriori probability that list decoding does not contain the*

transmitted codeword can be approximated as

$$\frac{\left(1-\sum_{j=1}^{q_L} P(N^n = z^{n,j})\right)\left(\frac{2^k-1}{2^n-1}\right)}{\sum_{i=1}^{L} P(N^n = z^{n,q_i}) + \left(1-\sum_{j=1}^{q_L} P(N^n = z^{n,j})\right)\left(\frac{2^k-1}{2^n-1}\right)} \tag{5.5}$$

Proof. Define the geometric distribution's probability of success to be $\phi = (2^k - 1)/(2^n - 1)$. Under the assumptions of the corollary, we have the formulae

$$P\left(W_{(1)}^{L} = q_1^L\right) = (1-\phi)^{q_L - L}\,\phi^L,$$

for $i \in \{1, \ldots, L-1\}$

$$P\left(W_{(1)}^{L-1} = q_1^{L,\{i\}}\right) = (1-\phi)^{q_L - L}\,\phi^{L-1},$$

and

$$P\left(W_{(1)}^{L-1} = q_1^{L-1}, W_{(L)} \geq q_L\right) = (1-\phi)^{q_L - L}\,\phi^{L-1}.$$

Using those expressions, simplifying eq. (5.3) gives eq. (5.5). □

To a slightly higher precision, the following approximation can be used, which accounts for eliminated queries and is most succintly expressed for a single-codeword decoding.

Corollary 3 (Approximate a posteriori likelihood of an incorrect GRAND decoding for a uniformly random codebook)**.** *If $W_{(1)}$ is assumed to be geometrically distributed with probability of success $(2^k-1)/(2^n-q)$ after $q-1$ failed queries, eq. (5.3) describing the a posteriori probability that a decoding found after q_1 queries is incorrect can be approximated as*

$$\frac{\left(1-\sum_{j=1}^{q_1} P(N^n = z^{n,j})\right)\frac{2^k-1}{2^n-q_1}}{P(N^n = z^{n,q_1}) + \left(1-\sum_{j=1}^{q_1} P(N^n = z^{n,j})\right)\frac{2^k-1}{2^n-q_1}}. \tag{5.6}$$

Proof. Under the conditions of the corollary,

$$P(W_{(1)} = q_1) = \prod_{i=1}^{q_1-1} \left(1 - \frac{2^k - 1}{2^n - i}\right) \frac{2^k - 1}{2^n - q_1},$$

from which eq. (5.3) simplifies to (5.6). □

Figure 5.1 illustrates sample output from (5.6). The probability of incorrect decoding is shown as a function of the number of GRAND codebook queries for a $[128, 116]$ code, assuming a maximum-likelihood guessing order. The log-likelihood ratio (LLR) for an error occurring at the i-th most unreliable bit is set to $-\beta i$, which matches the basic ORBGRAND model. A higher value of β indicates that bits are more reliable. It can be seen that the soft information has a significant effect on the crossover point where a decoding error becomes more likely than not. By abandoning the decoding before this crossover point, a significant amount of computation can be saved while avoiding a probable decoding error. The jaggedness in the curves is explained by the stepped probability distribution of the noise effects; all noise effects of the same logistic weight have the same probability, and a particular noise effect becomes relatively more likely to be the correct one as other noise effects of the same logistic weight are eliminated.

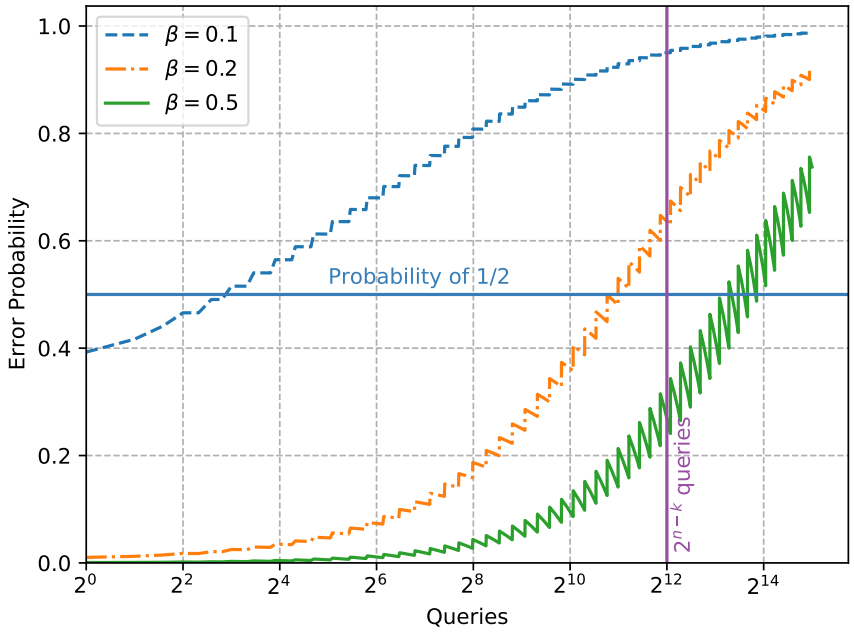


Figure 5.1: Probability of a block error given that a codeword is discovered after a given number of GRAND codebook queries. The LLR of an error occurring at the i-th bit in sorted order, according to the soft output of the channel, is set to $-\beta i$, where β is varied. A higher β value means the channel output is more reliable.

5.4 GRAND soft output per bit

Having derived the probability of error at the granularity of a codeword or decoding list, the probability of error can be derived for individual bits or symbols in the decoding output. For a random codebook, this does not have a uniform effect on the bit reliabilities. Depending on the guessing order, some bits may be affected more than others. To illustrate this point with an extreme example, if $n = 3$ for a binary message and GRAND has eliminated the noise effects $\{000, 001, 010, 011\}$, then the only remaining possible noise effects are $\{100, 101, 110, 111\}$, and it is known with full certainty that the value of the first bit is 1, while the second and third bits remain ambiguous.

Corollary 4 (A posteriori probability of an individual bit in GRAND hard output

being incorrect for a uniformly random codebook.). *Let d^n be the hard decoding output of GRAND list decoding, picked from the decoding list $\mathscr{L}$, and let $z^{n,*} = y^n \ominus d^n$ be the noise guess that yields that hard output. Let $c^{n,i}$ be the i-th codeword in $\mathscr{L}$. Define A and B_i, as before, to be the events that the transmitted codeword is not in the list and that the transmitted codeword is the i-th element in the list, respectively. Then, the probability that the j-th bit of the hard output d^n is incorrect is*

$$\frac{\sum_{i:1\le i\le L, c_j^{n,i}\neq d_j^n} P(B_i)}{P(A)+\sum_{i=1}^{L} P(B_i)} + \frac{P(A)}{P(A)+\sum_{i=1}^{L} P(B_i)} \frac{P(N_j^n \neq z_j^{n,*}) - \sum_{i:i\le q_L, z_j^{n,i}\neq z_j^{n,*}} P(N^n = z^{n,i})}{1-\sum_{i=1}^{q_L} P(N^n = z^{n,i})}. \tag{5.7}$$

Proof. The probability $P\left(C_j^n \neq d_j^n \middle| A \bigcup_{\ell=1}^{L} B_\ell\right)$ of the j-th bit of hard output being incorrect can be expressed using the law of total probability as

$$\begin{aligned} &P\left(\bigcup_{i=1}^{L} B_i, C_j^n \neq d_j^n \middle| A \bigcup_{\ell=1}^{L} B_\ell\right) + P\left(A, C_j^n \neq d_j^n \middle| A \bigcup_{\ell=1}^{L} B_\ell\right) \\ &= \sum_{i=1}^{L} P\left(B_i \middle| A \bigcup_{\ell=1}^{L} B_\ell\right) P(C_j^n \neq d_j^n | B_i) + P\left(A \middle| A \bigcup_{\ell=1}^{L} B_\ell\right) P(C_j^n \neq d_j^n | A). \end{aligned}$$

Since $P(C_j^n \neq d_j^n | B_i) = 1$ only if the i-th codeword in the list agrees with the hard output on the value of the j-th bit, and $P(C_j^n \neq d_j^n | B_i) = 0$ otherwise, this becomes

$$\sum_{i:1\le i\le L, c_j^{n,i}\neq d_j^n} P\left(B_i \middle| A \bigcup_{\ell=1}^{L} B_\ell\right) + P\left(A \middle| A \bigcup_{\ell=1}^{L} B_\ell\right) P(C_j^n \neq d_j^n | A).$$

This is reduced, by the application of (5.3), to

$$\frac{\sum_{i:1\leq i\leq L, c_j^{n,i}\neq d_j^n} P(B_i)}{P(A)+\sum_{i=1}^{L} P(B_i)} + \frac{P(A)}{P(A)+\sum_{i=1}^{L} P(B_i)} P(C_j^n \neq d_j^n|A), \tag{5.8}$$

and it remains only to derive the value of $P(C_j^n \neq d_j^n|A)$. Note that $P(C_j^n \neq d_j^n|A) = P(N_j^n \neq z_j^{n,*}|A)$, since the correctness of the decoding output depends on the correctness of the guessed noise effect. Following the law of total probability and Bayes' Rule gives

$$\begin{aligned} &P(C_j^n \neq d_j^n|A) = P(N_j^n \neq z_j^{n,*}|A) \\ &= \sum_{i:i>q_L, z_j^{n,i}\neq z_j^{n,*}} P(N^n = z^{n,i}|A) \\ &= \frac{\sum_{i:i>q_L, z_j^{n,i}\neq z_j^{n,*}} P(N^n = z^{n,i})}{1-\sum_{i=1}^{q_L} P(N^n = z^{n,i})}. \end{aligned} \tag{5.9}$$

Finally, given that $P(N_j^n = z_j^{n,*}) = \sum_{i:i\leq q_L, z_j^{n,i}=z_j^{n,*}} P(N^n = z^{n,i}) + \sum_{i:i\geq q_L, z_j^{n,i}=z_j^{n,*}} P(N^n = z^{n,i})$ and hence $\sum_{i:i\geq q_L, z_j^{n,i}=z_j^{n,*}} P(N^n = z^{n,i}) = P(N_j^n = z_j^{n,*}) - \sum_{i:i\leq q_L, z_j^{n,i}=z_j^{n,*}} P(N^n = z^{n,i})$, eq. (5.9) becomes

$$\frac{P(N_j^n = z_j^{n,*}) - \sum_{i:i\leq q_L, z_j^{n,i}=z_j^{n,*}} P(N^n = z^{n,i})}{1-\sum_{i=1}^{q_L} P(N^n = z^{n,i})},$$

which can be substituted for $P(C_j^n \neq d_j^n|A)$ in eq. (5.8) to finish the proof. □

This expression for the error probability of an individual bit is expressed in terms of the block error probability and in terms of the probabilities of the noise effects that were guessed throughout GRAND decoding. Like the block error probability itself, it is thus possible to compute the expression with negligible overhead, as long as there is a way to compute the probabilities of individual noise effects.

5.5 Performance evaluation

5.5.1 Accuracy of soft output

Armed with the approximate a posteriori probabilities in eq. (5.5) and (5.6), we investigate their precision for random and structured codebooks. Figure 5.2 depicts the accuracy of formula (5.6) when used for the class of random linear codes RLC$(64, 56)$. For context, Forney's approximation with a list size $L \in \{2, 4\}$ is also shown. Transmissions were simulated using an AWGN channel with BPSK modulation. 1-line ORBGRAND was used for soft-input decoding, which produced decoding lists of the appropriate size for both soft output methods. The codeword yielded by the highest-probability noise effect was selected as the hard output.

Figure 5.2 plots the empirical BLER given the predicted block error probability evaluated using eq. (5.6). If the estimate were precise, then the plot would follow the line $x = y$, as the predicted error probability and the BLER would match. As RLCs are linear, codewords are not exactly distributed uniformly in the guesswork order, but the formula is nevertheless shown to provide an accurate estimate. In contrast, Forney's approximation significantly underestimates the error probability, degrades in noisier channels, and has an estimate of no greater than $1/L$. Moreover, GRAND's prediction has been made having identified only a single codeword.

The same comparison is made in Figure 5.3 and Figure 5.4 but for RLC$(127, 120)$ and RLC$(31, 21)$ codes, respectively. In addition, GRAND soft output is evaluated for $L = 2$. Again, Forney's approximation is shown to underestimate the error probability and to degrade in noisier channels.

Figure 5.5 shows the prediction accuracy of GRAND soft output for list errors, which occur when the transmitted codeword is not in the decoding list. The measured list BLER is plotted against the predicted list BLER given by eq. (5.5). The prediction can be seen to be robust to channel condition, list size, and code structure. A comparison with Forney's approximation is not possible because it provides an estimate only for individual codewords in the list and not the whole list.

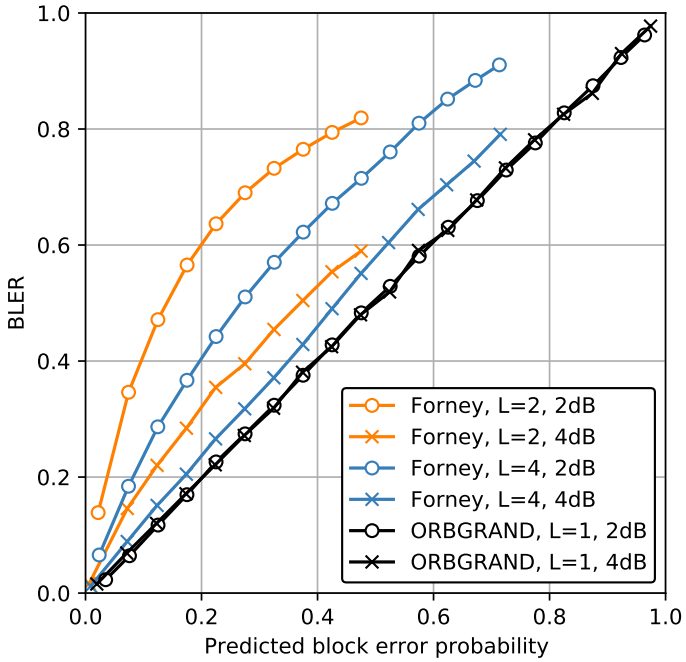


Figure 5.2: The accuracy of soft output when 1-line ORBGRAND is used to decode RLC(64, 57) codes. The predicted block error probability is compared to the measured BLER. If the soft output were perfectly accurate, then the data would follow the line $x = y$.

Figure 5.6 compares the estimated error probability for individual bits, based on eq. (5.7), to their measured BER. Again, the estimate closely follows the line $x = y$, indicating the accuracy of the prediction for a variety of code types, list sizes and channel conditions. Pyndiah's soft output approximation for bits, as explored in Section 4.2.2, is based on Euclidean distance and thus cannot be interpreted as a probability and compared to GRAND soft output.

While the plots shown thus far indicate that the proposed soft output accurately captures the probability of error, alternative evaluations of the accuracy are possible. A longer decoding list must, in some sense, provide more information about the probability of a codeword being correct; after the first codeword has been identified by GRAND, for instance, its probability of correctness will vary greatly depending on whether the next codeword is discovered after 1 guess or 1000 guesses, which can only be determined with a list size of $L > 1$. Yet, the prediction results

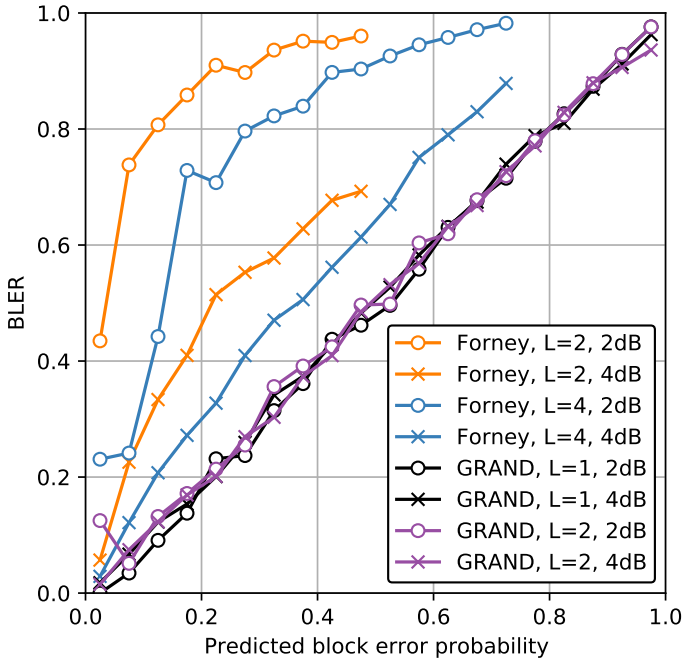


Figure 5.3: The accuracy of soft output when 1-line ORBGRAND is used to decode RLC(127, 120) codes.

shown previously follow the line $x = y$ for both $L = 1$ and $L > 1$. Intuitively, a longer list should lead to more informative predictions, because it clarifies the structure of the codebook and gives a more informative prior.

For this reason, we next compare the soft output methods using a scoring rule. Scoring rules

> ...assess the quality of probabilistic forecasts, by assigning a numerical score based on the predictive distribution and on the event or value that materializes...,

as per [41]. Measures from information theory have been investigated as scoring rules [71]; here, we use the **kl!** (**kl!**) divergence, also called the relative entropy, to capture the difference between the true distribution of error probability and the predicted distribution. Suppose that, after decoding has been performed, the true

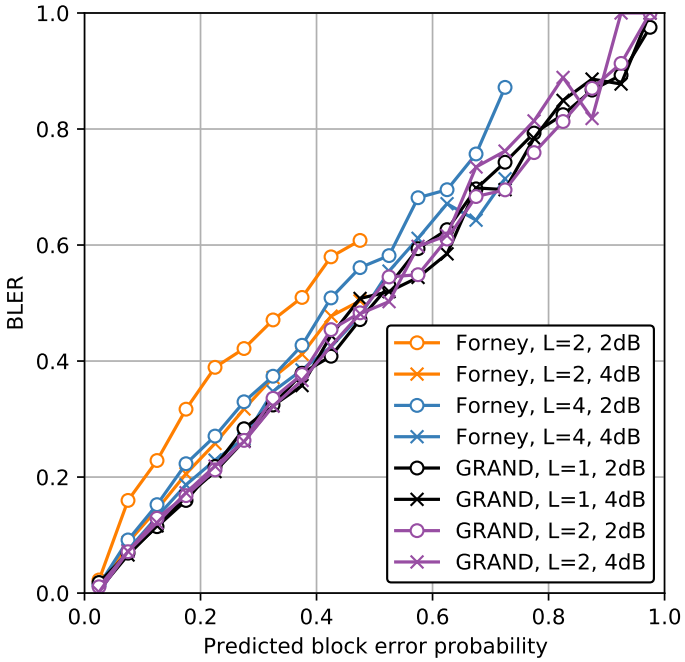


Figure 5.4: The accuracy of soft output when 1-line ORBGRAND is used to decode RLC$(31, 21)$ codes.

error probability is p_i and the predicted error probability is f_i. The KL divergence of these probabilities is

$$-p_i \log_2 \frac{f_i}{p_i} - (1 - p_i) \log_2 \frac{1 - f_i}{1 - p_i}$$

which gets larger as the prediction diverges further from the true probability, and is minimised by $f_i = p_i$ [71].

More specifically, suppose that, in the i-th sample, a codeword is identified by GRAND after q_1 guesses. An estimate f_i of error probability is calculated using one of the GRAND soft output formulae or using Forney's approximation. If the 2^k codewords in the codebook are yielded by noise effects $z^{n,q_1}, z^{n,q_2}, \ldots, z^{n,q_{2^k}}$, then the true error probability for the first codeword is

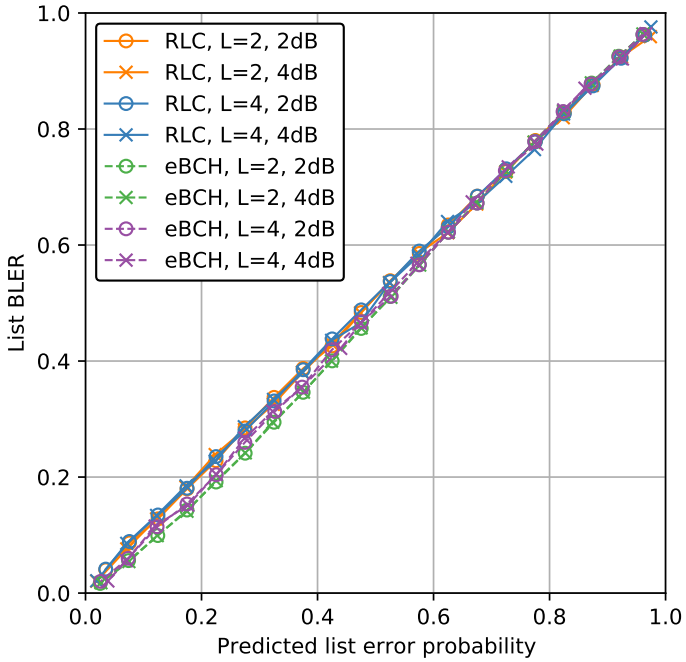


Figure 5.5: The accuracy of the predicted list error probability compared to the measured list BLER. Same parameters as in Figure 5.2, but with varying channel noise, varying list size $L \in \{2, 4\}$, and code types RLC$(64, 57)$ and eBCH$(64, 57)$.

$$p_i = \frac{P(N^n = z^{n,q_1})}{\sum_{j=1}^{2^k} P(N^n = z^{n,q_j})},$$

and from this the KL divergence of p_i and f_i can be determined.

Figure 5.7 illustrates the KL divergence for the class of $[15, 7]$ random linear codes, with both GRAND soft output and Forney soft output used to predict the error probability of the first codeword in the decoding list. The error probability of the first codeword is the only suitable point of comparison because it can be evaluated regardless of list size. 1-line ORBGRAND is used for list decoding. For reasons of complexity, this approach to evaluating prediction accuracy is restricted to small values of k (here, $k = 7$), since its calculation involves all 2^k codewords in the codebook. It can be seen that the performance of both GRAND soft output and

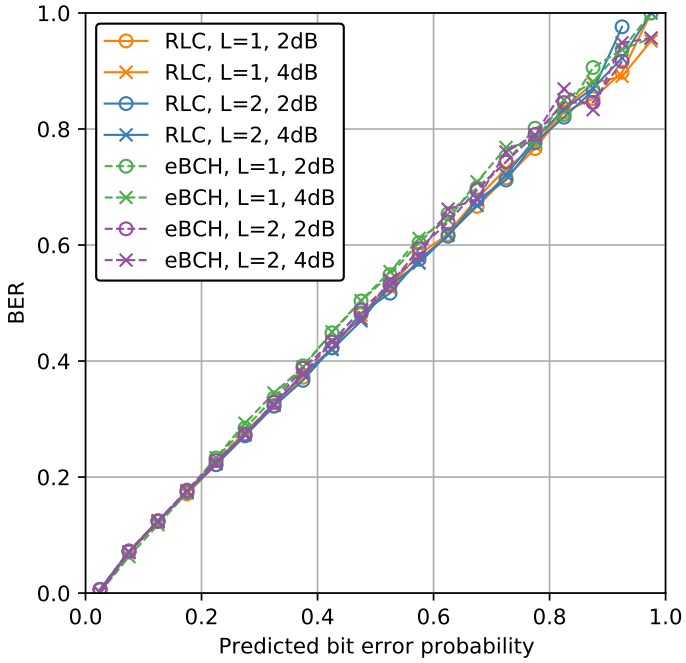


Figure 5.6: Estimated probability of error per bit compared to the measured BER. Same parameters as in Figure 5.5. Varying channel noise, varying list size $L \in \{2, 4\}$, and code types RLC$(64, 57)$ and eBCH$(64, 57)$.

Forney soft output degrades in noisier channels, though this is more severe for Forney's approximation. GRAND soft output scores better or as well as Forney's approximation for same list size, depending on the channel conditions. Under the noisiest channel conditions, the GRAND soft output provides a better prediction than Forney, even with a decoding list of half the size. While these results give a preliminary insight into the accuracy of GRAND soft output, further effort is required to explore the vast range of possible scoring rules and to fully analyse its behaviour.

5.5.2 Application to error detection.

A common method used to detect errors is to append a CRC to a message and declare an erasure at the receiver if there is an inconsistency. As GRAND algorithms can decode any code, one use of GRAND soft output is to upgrade CRCs so that they are used for both error correction and error detection, by decoding the block

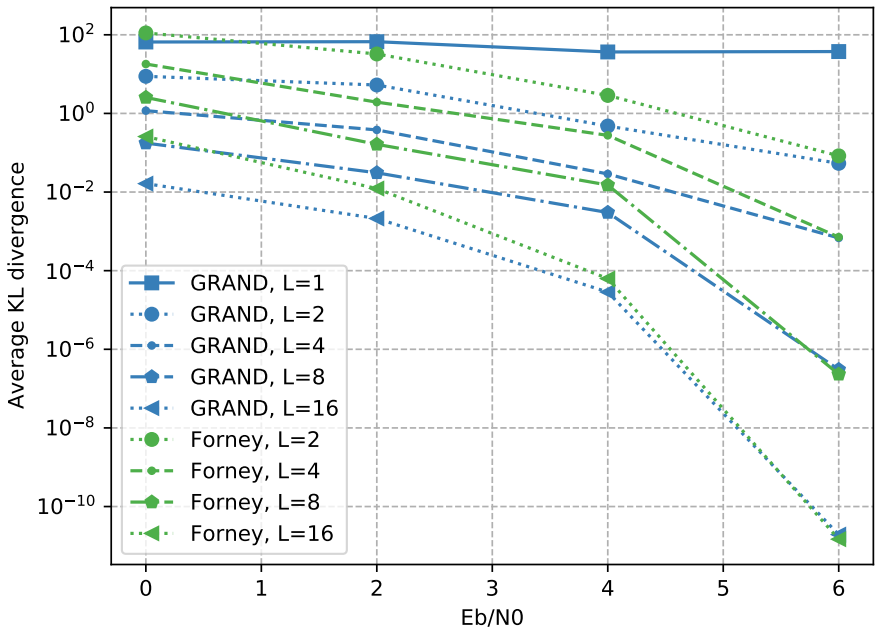


Figure 5.7: KL divergence of per-block probabilistic predictions for $[15, 7]$ RLCs. In a given simulated transmission, probabilistic soft output is calculated by using 1-line ORBGRAND to produce decoding lists of varying size L. Forney's approximation and the GRAND soft output formula in eq. (5.6) are then used to estimate the probability that the first codeword in the decoding list is the correct one. Finally, they are compared in terms of KL divergence to the true correctness probability, which is determined based on an exhaustive evaluation of the full codebook. For $L = 1$ only GRAND soft output can be calculated.

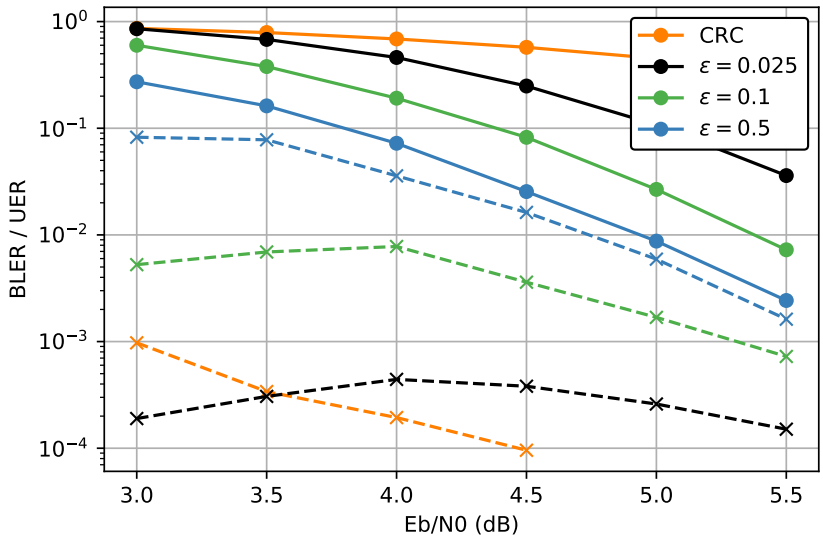


Figure 5.8: The UER (dotted lines) and BLER (solid lines) of a CRC$(64, 56)$ code with two methods of error control: 1) CRC used for error detection; 2) ORBGRAND performs error correction using the CRC, then erasures are declared if the predicted block error probability exceeds ϵ.

but returning an erasure if the decoding is too unreliable. The BLER that results from this process is composed of both undetected block errors and erasures.

Figure 5.8 depicts the undetected error rate (UER) and BLER of a CRC$(64, 56)$ code. Two methods of error control are compared: 1) the CRC is checked for consistency and an erasure is declared if it fails; 2) ORBGRAND is used to correct errors and an erasure is declared if the estimated error probability is greater than a threshold ϵ. The advantage of 2) is that error correction with GRAND results in significantly reduced BLER while tailoring the error detection to a target UER by modifying the threshold accordingly.

As another example, Figure 5.9 depicts the error detection and correction performance of an eBCH$(64, 51)$ code with ORBGRAND decoding when an error probability threshold ϵ is used for erasure decisions. Shown for comparison is CA-SCL decoding [82] of a $(64, 51 + 6)$ 5G polar code [1] concatenated with the 6-bit CRC

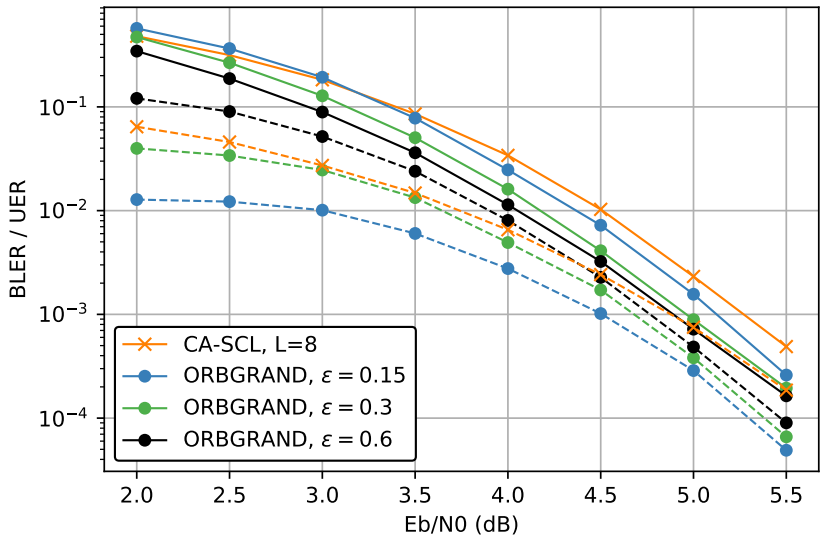


Figure 5.9: The error detection and correction performance of: 1) a $(64, 51 + 6)$ 5G polar code concatenated with the 6-bit CRC `0x30` and with CA-SCL decoding; (2) an eBCH$(64, 51)$ code with ORBGRAND decoding and a threshold-based erasure decision with threshold $\epsilon = 0.15$. Solid lines correspond to BLER, dashed lines to UER.

`0x30`, generating a list of 8 candidates from which the most likely of those whose CRC is consistent is declared to be the decoding. If no element of the CA-SCL list has a CRC that matches, it is treated as an erasure. With an appropriately chosen ϵ, both methods achieve a similar BLER, but ORBGRAND is shown to achieve a UER that is almost an order of magnitude lower than CA-SCL in the 2dB to 3dB range. In less noisy conditions, it still achieves a gain of 0.5dB.

5.6 Discussion

We have established that soft-input GRAND algorithms can, during their execution, evaluate a predicted likelihood that the decoded block, list, or bits are in error. We have derived exact formulae along with readily computable approximations. While the formulae assume random codebooks, we have empirically shown them to be accurate for structured codebooks.

There are many potential applications of this soft output. It can be used to reduce the rate of undetected errors during decoding, or, for URLLC, to do so more cheaply, as GRAND can use a CRC or any other code for both error correction and reduction of undetected errors. In hybrid automatic repeat request (HARQ) schemes, the predicted correctness probability could be used to determine whether to request retransmission, reducing the number of requests. It has been shown [28] that GRAND soft output can be used to compromise the security of wiretap channels. As shown in the previous chapter, soft output can be used in the turbo decoding of product codes. The confidence measure could also be used to determine the most reliable decoding from a collection of decodings, which could help to select a lead channel in noise recycling [21, 68].

CHAPTER 6

Discussion

This thesis began with an introduction to the channel coding problem. Channel coding was launched as a field in 1948 with the publication of Claude Shannon's magnum opus, *A Mathematical Theory of Communication* [78]. Significant effort has been expended since then to discover new channel codes and efficient decoding algorithms for those codes. For many years, the dominant paradigm has been based on long codes with efficient, specialised decoders, such as LDPC codes [73]. New requirements for modern communications systems, however, lead to the possibility of new paradigms. In particular, the requirement of URLLC in modern standards [31] necessitates the use of short, high-rate codes. Such codes are amenable to decoding by universal decoders that are not specific to any particular code.

Throughout this thesis we have investigated a recently-introduced universal decoder called GRAND. GRAND has been established as a practical, accurate decoder for arbitrary codes with a moderate amount of redundancy, as indicated by the development of two efficient hardware implementations for hard- and soft-input decoding [67, 69]. One unexplored topic, however, has been how to apply GRAND to codes with large amounts of redundancy. To that end, in Chapter 3 and Chapter 4 we explored how GRAND can be used to decode a class of high-redundancy codes known as product codes.

Chapter 3 introduced the IGRAND algorithm for hard-input decoding of product

codes. IGRAND adopts the strategy of iterative decoding, in which the rows and columns of a product code are individually decoded over multiple iterations. This approach decomposes the decoding problem into many easier problems, resulting in lower complexity than if the entire product code were decoded by GRAND at once. To further reduce complexity and to improve accuracy, IGRAND undoes errors in a component of the product code only when the number of errors falls within some distance bound, which is kept as low as possible. This distinguishes it from other iterative decoding algorithms and enables it to achieve superior performance in terms of both accuracy and complexity, as shown by experimental results in which IGRAND gained up to 0.5dB over the next-best comparable decoding algorithm for a selection of long product codes. This chapter used real measurements from a hardware implementation of GRAND [67] to estimate the decoding complexity of IGRAND, and explored the latency and throughput improvements from decoding in parallel, which is enabled by the structure of product codes.

Chapter 4 was a natural progression from hard-input to soft-input decoding of product codes. This was achieved by re-conceptualising GRAND as a list decoder, allowing it to produce soft output based on the method of Pyndiah [65]. Given the availability of soft output, GRAND could then be used as a component decoder in the turbo decoding of product codes. ORBGRAND [25] was found to be an effective soft-input list decoder, and in list decoding experiments it was significantly more accurate than the Chase algorithm that is typically used for block turbo decoding. Experiments also validated the theoretical prediction of Elias [33] that larger list sizes compensate for weakness in the structure of a code. Finally, experiments showed the accuracy of ORBGRAND as a turbo component decoder, as it achieved coding gains of up to 0.7dB over the Chase algorithm for a variety of product codes. These simulation-based results were accompanied by an analytical derivation of GRAND list decoding complexity, and it was proven that, for random codes, doubling the list size doubles the upper bound on average decoding complexity.

In Chapter 4, soft output was used in the turbo decoding of product codes. Generally, list decoding is required to compute soft output for block codes, bringing with it a large complexity overhead. Chapter 5 investigated the question of whether

accurate soft output can be calculated from GRAND decoding, with or without a decoding list. This investigation yielded a probabilistic soft output for the GRAND algorithm, indicating the correctness probability of a codeword, decoding list, or individual decoded symbol. The output is computed by a lightweight accounting of the noise guesses, and thus adds negligible overhead. When compared with Forney's approximate method for probabilistic soft output, the GRAND soft output was shown to be more accurate, particularly under the noisiest channel conditions. The availability of soft output opens up many applications, including error detection, turbo decoding and the triggering of retransmission requests. The application to error detection was tested in simulations and compared to the use of a CRC. Simulations showed that the probabilistic soft output permits an easily-tunable balance between undetected errors and decoding errors while avoiding the CRC's impact on code rate.

All chapters considered, this thesis has demonstrated the use of GRAND to decode high-redundancy codes. While these codes, product codes, can also be decoded by specialised component decoders, GRAND has the advantage of being able to decode arbitrary product codes and with high accuracy. The alternative soft output in Chapter 5 offers the potential to calculate soft output without the overhead of list decoding, and enables many new applications of the GRAND algorithm.

6.1 Future work

Continuing the theme of applying GRAND to long, high-redundancy codes, one possible future direction is to apply the algorithms considered here to other types of such codes. In particular, staircase codes [79], a generalisation and improvement of product codes, are an excellent candidate for IGRAND or turbo decoding, and are under consideration for use in the latest standards of optical transport networks [85]. In a recent paper [89], we have also considered the application of iterative soft-input soft-output GRAND decoding to another variant of product codes called generalized LDPCs (GLDPCs) [59].

Besides examining state-of-the-art error correction codes, another avenue is to synthesise the different ideas we have presented here. More specifically, the GRAND

soft-output from Chapter 5 can be used for iterative soft-input soft-output decoding of product codes, as an alternative to the turbo-style decoding seen in Chapter 4. We recently explored this idea in [89], which showed that iterative decoding of product codes with GRAND soft output offers decoding performance that is similar or superior to LDPC codes from the 5G standard. Given that LDPCs are ubiquitous in modern communications standards, from 5G [1] to ATSC 3.0 [54], any ECC and decoding algorithm that perform competitively must therefore be of great relevance to the future of channel coding, and merit further investigation.

To improve on the turbo decoding results of Chapter 4, another approach is to incorporate optimisations of the block turbo algorithm that have been developed since Pyndiah's original work. A summary of some of these improvements can be found in [7]. In particular, it has been shown that the decoding accuracy can be improved by dynamically computing the reliability of a bit if all codewords in the decoding list agree on its value, as opposed to setting the reliability to a constant. It may also be possible to tweak the balance between complexity and accuracy by employing other soft-input GRAND algorithms instead of ORBGRAND, such as SGRAND [80] (higher complexity, higher accuracy) and Symbol Reliability GRAND (SRGRAND) [29] (lower complexity, lower accuracy).

Regarding the GRAND soft output technique, it's possible that alternative formulae can be developed that specialise on particular code structures in order to improve the soft output accuracy. For example, our assumption of a random codebook, while encompassing all possible codes, does not account for relevant properties of linear codes such as the minimum Hamming distance. While this idea may contradict the code-independent philosophy of GRAND, it has already been exploited to reduce GRAND complexity [72], and may yield improvements in decoding accuracy and soft output accuracy. It also remains to investigate alternative scoring rules for evaluating GRAND soft output, such as those detailed in [71].

Due to GRAND's code-agnosticism, its use in coding schemes is extremely flexible, and it can be employed in ways that non-universal decoders cannot, for example in [43]. For this reason, it is possible that entirely new applications of soft output

can be invented. In addition to the applications proposed in Chapter 5, other existing applications of soft output include turbo equalization of single-input single-output Inter-Symbol Interference channels and Multiple-Input Multiple-Output (MIMO) frequency selective fading channels; iterative decoding of MIMO flat fading channels; and multi-user detection and equalization in coded Direct-Sequence Code-Division Multiple-Access systems [86].

APPENDIX A

Capacity-achieving proof for random linear product codes

A.1 Random linear product codes

Chapter 3 demonstrates the efficacy of IGRAND with known structured codes. Here we examine random linear product codes (RLPCs), a class of product codes that IGRAND can decode and that non-universal decoders cannot. RLPCs are product codes whose component codes are RLCs. We prove that RLPCs achieve capacity in hard-decision channels and so offer a wide selection of high-performing product codes.

A.2 Notation

Let x, $\vec{x}$, X denote a scalar, vector and matrix, respectively. Let $\vec{\mathbf{x}}, \mathbf{X}$ denote a random vector and random matrix, respectively. Let $\mathbb{F}_q$ denote a finite field of order q.

A.3 Construction of RLPCs

Recall that a systematic $[n, k]$ linear code over $\mathbb{F}_q$ is defined by a generator matrix $G = [I|P]$, where I is the $k \times k$ identity matrix and P is a $k \times (n-k)$ parity

matrix over $\mathbb{F}_q$. The codebook $\mathcal{C}$ of a given linear code is the image of G, namely $\mathcal{C} = \{\vec{c} \in \mathbb{F}_q^n : \exists \vec{u} \in \mathbb{F}_q^k, \vec{c} = \vec{u}G\}$. A random systematic linear code is one where entries in the parity check matrix, P, are chosen independently and uniformly at random. It is known that every linear code has an equivalent systematic linear code, so we limit the discussion here to systematic codes alone [70]. The encoding of product codes, and hence RLPCs, is described in section 2.5. The component codes of RLPCs are systematic RLCs.

A.4 Proof that RLPCs are capacity-achieving

Since the work of Shannon, it has been known that random codes are capacity-achieving [78]. It was later shown that confining the discussion to RLCs still yields capacity-achieving codes [64]. We prove that RLPCs are capacity-achieving in hard-decision channels. To do so, we use the following lemma, taken from [70], and its corollary.

Lemma 2. *Let $\vec{a}, \vec{x} \in \mathbb{F}_q^l$, $\vec{a}$ being a non-zero vector. Let $f(\vec{x}) = \sum_{i=1}^{l} a_i x_i$. Then each element of $\mathbb{F}_q$ is the image under f of exactly q^{l-1} vectors in $\mathbb{F}_q^l$.*

Corollary 5. *Let $\vec{a} \in \mathbb{F}_q$ be a non-zero vector, let $b \in \mathbb{F}_q$, and let $\vec{\boldsymbol{x}}$ be uniformly distributed over $\mathbb{F}_q^l$. Then $\sum_{i=1}^{l} a_i \boldsymbol{x}_i + b$ is uniformly distributed over $\mathbb{F}_q$.*

To prove that RLPCs achieve capacity, we follow the proof of Theorem 16.2 from Chapter 16 of [64], which establishes that RLCs are capacity-achieving. The steps are unchanged, except that we adapt step 2 for RLPCs.

Theorem 3. *Let $P_{\vec{\boldsymbol{z}}}$ be the probability distribution of additive noise $\vec{\boldsymbol{z}}$ over $\mathbb{F}_q^n$. Then for every k there exists an $[n, k]$ RLPC with error probability:*

$$P_e \leq \mathbb{E}\left[q^{-\left(n-k-\log_q \frac{1}{P_{\vec{\boldsymbol{z}}}(\vec{\boldsymbol{z}})}\right)^+}\right], \tag{A.1}$$

where $x^+ = max(x, 0)$.

Proof. Let $\vec{\mathbf{u}}, \vec{\mathbf{u}}' \in \mathbb{F}_q^n$ be uniformly distributed and independent, let $\mathbf{G}$ be a generator matrix of an $[n, k]$ RLPC, and let $\vec{\mathbf{h}} \in \mathbb{F}_q^n$ be a uniformly distributed dithering vector independent of ($\vec{\mathbf{u}}$, $\vec{\mathbf{u}}'$).

1. Encode with dithering: $\vec{\mathbf{x}} = \vec{\mathbf{u}}\mathbf{G} + \vec{\mathbf{h}}$, $\vec{\mathbf{x}}' = \vec{\mathbf{u}}'\mathbf{G} + \vec{\mathbf{h}}$ are uniformly distributed over $\mathbb{F}_q^n$.

2. We have to show that $\vec{\mathbf{x}}' = \vec{\mathbf{x}} + (\vec{\mathbf{u}}' - \vec{\mathbf{u}})\mathbf{G}$ is uniformly distributed over $\mathbb{F}_q^n$, independent of the realisation of $\vec{\mathbf{u}}$, $\vec{\mathbf{x}}$. In particular, we have to show that the different elements of $\vec{\mathbf{x}}'$ are uniformly distributed over $\mathbb{F}_q$ and independent of each other (and of $\vec{\mathbf{u}}$, $\vec{\mathbf{x}}$). The systematic part is given, as the elements of $\vec{\mathbf{u}}' - \vec{\mathbf{u}}$ are distributed uniformly and independent of each other. It remains to show that this also holds for the parity symbols, which is achieved by corollary 5.

3. Repeat the argument in proving the dependence-testing bound for symmetric and pairwise independent codewords.

4. Compute information density $i(\vec{\mathbf{x}}; \vec{\mathbf{y}}) = log_q \frac{P_{\vec{\mathbf{y}}|\vec{\mathbf{x}}}(\vec{\mathbf{y}}|\vec{\mathbf{x}})}{P_{\vec{\mathbf{y}}}(\vec{\mathbf{y}})}$, where $\vec{\mathbf{y}} = \vec{\mathbf{x}} + \vec{\mathbf{z}}$. This leads to an upper bound $P_e \leq \mathbb{E}\left[q^{-\left(n-k-\log_q \frac{1}{P_{\vec{\mathbf{z}}}(\vec{\mathbf{z}})}\right)^+}\right]$.

5. Remove $\vec{\mathbf{h}}$, as shifting the codebook has no impact on its performance.

□

The proof holds for any number of component codes d and any number of parity symbols, as long as there is at least one non-zero element in each dimension of the parity matrix P. Otherwise, there will be constant zero parity symbols in the codeword. This problem also arises in the original proof of [64] (third note), and can be avoided in a similar fashion. One such way is to draw at least one of the elements of P in each dimension from $\mathbb{F}_q \setminus \{0\}$ rather than $\mathbb{F}_q$.

List of Figures

1.1 Diagram of a digital communication system. 3
1.2 An example transmission through a communication system. 5

2.1 Illustration of minimum Hamming distance concept. 12
2.2 Illustration of a binary symmetric channel. 13
2.3 Illustration of binary phase shift keying modulation. 13
2.4 GRAND and ORBGRAND chip implementations. 16
2.5 Reliability distribution as a function of channel noise. 20
2.6 GRAND decoding region. 22
2.7 Product code diagram. 23

3.1 GRAND decoding region, expanded. 26
3.2 Error patterns affecting a product code 28
3.3 Detailed illustration of IGRAND algorithm. 32
3.4 Probability of efficiency gain and decoding mistake with columns-only decoding, $n = 31^2$. 37
3.5 Probability of efficiency gain and decoding mistake with columns-only decoding, $n = 127^2$. 38
3.6 Bit error rate of IGRAND and comparable decoding algorithms, applied to a product code with $BCH(31, 21, 5)$ component codes. 39
3.7 Bit error rate of IGRAND and comparable decoding algorithms, applied to a product code with BCH$(63, 51, 5)$ component codes. 40
3.8 IGRAND performance on a selection of codes. 41
3.9 RLC vs BCH. 42
3.10 Estimated energy cost per bit of decoding a BCH$(127, 106, 7)^2$ code using GRAND hardware. 43
3.11 Estimated latency of decoding a BCH$(127, 106, 7)^2$ code with GRAND hardware. 43
3.12 Estimated latency and throughput when IGRAND decodes a BCH$(127,106,7)^2$ code, with an increasing number of parallel decoding branches 44

3.13 IGRAND columns-only decoding latency. 44

4.1 Distribution of soft input during turbo decoding. 55
4.2 List decoding BLER of an eBCH$(32, 26, 4)$ code with basic ORB-GRAND and Chase as list decoders. 58
4.3 List decoding BLER of BCH and RLC codes. 59
4.4 List decoding BLER of a selection of codes. 59
4.5 Block turbo decoding BER of an eBCH$(32, 26, 4)^2$ product code. . . . 60
4.6 Block turbo decoding BER of an eBCH$(64, 57, 4)^2$ code. 61
4.7 Block turbo decoding BER of a selection of product codes. 61
4.8 Accuracy comparison of IGRAND versus turbo decoding. 62
4.9 Block turbo decoding complexity in terms of raw number of queries per bit. 63
4.10 Block turbo decoding complexity in terms of ratios. 63
4.11 Complexity comparison of IGRAND and turbo decoding. 64

5.1 Error probability as a function of the number of codebook queries. . . 77
5.2 Accuracy of GRAND soft output for RLC$(64, 57)$. 81
5.3 Accuracy of GRAND soft output for RLC$(127, 120)$. 82
5.4 Accuracy of GRAND soft output for RLC$(31, 21)$. 83
5.5 Accuracy of list-level soft output. 84
5.6 Accuracy of bit-level soft output. 85
5.7 KL divergence of probabilistic predictions. 86
5.8 UER and BLER comparison of CRC and ORBGRAND. 87
5.9 Error detection and correction comparison of CA-SCL and ORB-GRAND. 88

Bibliography

[1] 3GPP. NR; Multiplexing and channel coding. Technical Specification (TS) 38.21, 3rd Generation Partnership Project (3GPP), 2019. Version 15.5.0.

[2] S. M. Abbas, M. Jalaleddine, and W. J. Gross. High-Throughput VLSI Architecture for GRAND Markov Order. In *IEEE Workshop Sig. Proc. Sys.*, 2021.

[3] S. M. Abbas, M. Jalaleddine, and W. J. Gross. List-GRAND: A Practical Way to Achieve Maximum Likelihood Decoding. *IEEE Trans. on VLSI Sys.*, 31:43–54, 2023.

[4] S. M. Abbas, T. Tonnellier, F. Ercan, and W. J. Gross. High-Throughput VLSI Architecture for GRAND. In *IEEE Workshop on Sig. Proc. Sys.*, pages 681–693, 2020.

[5] S. M. Abbas, T. Tonnellier, F. Ercan, M. Jalaleddine, and W. J. Gross. High-Throughput and Energy-Efficient VLSI Architecture for Ordered Reliability Bits GRAND. *IEEE Trans. on VLSI Sys.*, 30(6), 2022.

[6] N. Abramson. Cascade decoding of cyclic product codes. *IEEE Trans. Commun.*, 3(16):398–402, 1968.

[7] P. Adde and R. Pyndiah. Recent simplifications and improvements in Block Turbo Codes. In *Int. Symp. on Turbo Codes & Rel. Topics*, pages 133 – 136, Brest, France, Sept. 2000.

[8] A. J. Al-Dweik and B. S. Sharif. Non-sequential decoding algorithm for hard iterative turbo product codes. *IEEE Trans. Commun.*, 57:1545–1549, 2009.

[9] A. J. Al-Dweik and B. S. Sharif. Closed-chains error correction technique for Turbo Product Codes. *IEEE Trans. Commun.*, 59(3):632–638, 2011.

[10] W. An, M. Médard, and K. R. Duffy. Keep the bursts and ditch the interleavers. In *IEEE GLOBECOM*, 2020.

[11] W. An, M. Médard, and K. R. Duffy. CRC codes as error correcting codes. In *IEEE ICC*, 2021.

[12] E. Arikan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. Inf. Theory*, 55(7):3051–3073, 2009.

[13] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg. LLR-based successive cancellation list decoding of Polar codes. *IEEE Trans. Signal Process.*, 63(19):5165–5179, 2015.

[14] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3):384–386, 1978.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo-codes. In *IEEE ICC*, volume 2, 1993.

[16] G. Bosco, G. Montorsi, and S. Benedetto. A new algorithm for "hard" iterative decoding of concatenated codes. *IEEE Trans. Commun.*, 51(8):1229–1232, 2003.

[17] N. Chapalain, A. Guéguen, D. Castelain, and R. Pyndiah. A Way to Combat the Sub-optimality of Turbo Decoding of Product Codes. *EPMCC*, 2001.

[18] D. Chase. Class of algorithms for decoding block codes with channel measurement information. *IEEE Trans. Inf. Theory*, 18(1):170–182, 1972.

[19] F. Chiaraluce and R. Garello. Extended Hamming product codes analytical performance evaluation for low error rate applications. *IEEE Trans. on Wireless Comm.*, 3(6):2353–2361, 2004.

[20] M. M. Christiansen and K. R. Duffy. Guesswork, Large Deviations, and Shannon Entropy. *IEEE Trans. Inf. Theory*, 59(2):796–802, 2013.

[21] A. Cohen, A. Solomon, K. R. Duffy, and M. Médard. Noise recycling. In *IEEE Int. Symp. on Inf. Theory*, 2020.

[22] C. Condo. Iterative soft-input soft-output decoding with ordered reliability bits GRAND, 2022. arXiv:2207.06691.

[23] S. Dave, J. Kim, and S. Kwatra. An efficient decoding algorithm for block turbo codes. *IEEE Transactions on Communications*, 49:41–46, 01 2001.

[24] K. R. Duffy. Ordered reliability bits guessing random additive noise decoding. In *IEEE ICASSP*, 2021.

[25] K. R. Duffy, W. An, and M. Medard. Ordered reliability bits guessing random additive noise decoding. In *IEEE Trans. Sig. Proc.*, volume 70, pages 4528 – 4542, 2022.

[26] K. R. Duffy, M. Grundei, and M. Médard. Using channel correlation to improve decoding – ORBGRAND-AI, 2023. arXiv:2303.07461.

[27] K. R. Duffy, J. Li, and M. Médard. Capacity-achieving guessing random additive noise decoding. *IEEE Trans. Inf. Theory*, 65(7):4023–4040, 2019.

[28] K. R. Duffy and M. Medard. Soft detection physical layer insecurity, 2023. arXiv:2212.05309.

[29] K. R. Duffy, M. Médard, and W. An. Guessing random additive noise decoding with symbol reliability information (SRGRAND). In *IEEE Trans. Commun.*, volume 70, pages 3–18, 2022.

[30] K. R. Duffy, A. Solomon, K. M. Konwar, and M. Médard. 5G NR CA-Polar maximum likelihood decoding by GRAND. In *Ann. Conf. on Inf. Sci. Sys.*, 2020.

[31] G. Durisi, T. Koch, and P. Popovski. Toward Massive, Ultrareliable, and Low-Latency Wireless Communication With Short Packets. *Proceedings of the IEEE*, 104(9):1711–1726, 2016.

[32] P. Elias. Error-free Coding. *Trans. IRE Prof. Group Inf. Theory*, 4(4):29–37, 1954.

[33] P. Elias. List decoding for noisy channels. In *IRE WESCON Convention Record*, 1957.

[34] G. Forney. Exponential error bounds for erasure, list, and decision feedback schemes. *IEEE Trans. Inf. Theory*, 14(2):206–220, 1968.

[35] G. Forney. The Viterbi algorithm. *Proc. of the IEEE*, 61:268–278, 1973.

[36] J. Freudenberger, D. Nicolas Bailon, and M. Safieh. Reduced complexity hard- and soft-input BCH decoding with applications in concatenated codes. *IET Circ. Device Syst.*, 15(3):284–296, 2021.

[37] R. Gallager. Low-density parity-check codes. *IRE Trans. Inf. Theory*, 8:21–28, 1962.

[38] K. Galligan, M. Médard, and K. R. Duffy. Block turbo decoding with ORB-GRAND. In *CISS*, pages 1–6, 2023.

[39] K. Galligan, A. Solomon, A. Riaz, M. Médard, R. T. Yazicigil, and K. R. Duffy. IGRAND: decode any product code. In *IEEE GLOBECOM*, 2021.

[40] K. Galligan, P. Yuan, M. Médard, and K. R. Duffy. Upgrade error detection to prediction with GRAND. In *IEEE GLOBECOM*, 2023.

[41] T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[42] A. Goldsmith. *Wireless Communications*. CUP, 2005.

[43] J. Griffin, P. Yuan, P. Popovski, K. R. Duffy, and M. Médard. Code at the Receiver, Decode at the Sender: GRAND with Feedback. In *IEEE Inf. Theory Workshop (ITW)*, pages 341–346, 2023.

[44] J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. In *IEEE GLOBECOM*, volume 3, pages 1680–1686, 1989.

[45] C. Häger and H. D. Pfister. Approaching miscorrection-free performance of product codes with anchor decoding. *IEEE Trans. Commun.*, 66(7):2797–2808, 2018.

[46] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[47] T. Hashimoto. Composite scheme LR+Th for decoding with erasures and its effective equivalence to Forney's rule. *IEEE Trans. Inf. Theory*, 45(1):78–93, 1999.

[48] T. Hashimoto and M. Taguchi. Performance of explicit error detection and threshold decision in decoding with erasures. *IEEE Trans. Inf. Theory*, 43(5):1650–1655, 1997.

[49] E. Hof, I. Sason, and S. Shamai. Performance bounds for erasure, list, and decision feedback schemes with linear block codes. *IEEE Trans. Inf. Theory*, 56(8):3754–3778, 2010.

[50] J. Justesen. Performance of Product Codes and Related Structures with Iterated Decoding. *IEEE Trans. Commun.*, 59(2):407–415, 2011.

[51] J. Justesen, K. Larsen, and L. Pedersen. Error correcting coding for OTN. *IEEE Comm. Mag.*, 48:70 – 75, 10 2010.

[52] T. Kaneko, T. Nishijima, H. Inazumi, and S. Hirasawa. An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder. *IEEE Trans. Inf. Theory*, 40(2):320–327, 1994.

[53] S. Kerouedan, P. Adde, and R. Pyndiah. How do we implement block turbo codes? In *Turbo Codes, Error-correcting codes of widening application*, Innovative technology, Inf. sys. and networks, pages 127 – 141. Hermes Penton Science, 2002.

[54] K.-J. Kim, S. Myung, S.-I. Park, J.-Y. Lee, M. Kan, Y. Shinohara, J.-W. Shin, and J. Kim. Low-density parity-check codes for atsc 3.0. *IEEE Trans. on Broadcasting*, 62(1):189–196, 2016.

[55] P. Koopman and T. Chakravarty. Cyclic redundancy code (CRC) polynomial selection for embedded networks. In *Int. Conf. on Dep. Sys. and Net.*, 2004.

[56] X. Liang, J. Yang, C. Zhang, W. Song, and X. You. Hardware efficient and low-latency ca-scl decoder based on distributed sorting. In *IEEE GLOBECOM*, pages 1–6. IEEE, 2016.

[57] S. Lin and D. J. Costello. *Error control coding: fundamentals and applications*. Pearson/Prentice Hall, 2004.

[58] M. Liu, Y. Wei, Z. Chen, and W. Zhang. ORBGRAND Is Almost Capacity-Achieving. *IEEE Trans. on Inf. Theory*, 69(5):2830–2840, 2023.

[59] G. Liva, W. E. Ryan, and M. Chiani. Quasi-cyclic generalized ldpc codes with low error floors. *IEEE Trans. on Comm.*, 56(1):49–57, 2008.

[60] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[61] A. Mahran and M. Benaissa. Adaptive chase algorithm for block turbo codes. *Electronics Letters*, 39(7):1–2, 2003.

[62] A. Mahran and M. Benaissa. Iterative decoding with a hamming threshold for block turbo codes. *IEEE Communications Letters*, 8(9):567–569, 2004.

[63] K. Niu and K. Chen. CRC-aided decoding of Polar codes. *IEEE Commun. Letters*, 16(10):1668–1671, 2012.

[64] Y. Polyanskiy. 6.441 Information Theory.

[65] R. Pyndiah. Near-optimum decoding of product codes: block turbo codes. *IEEE Trans. Commun.*, 46(8):1003–1010, 1998.

[66] A. Raghavan and C. Baum. A reliability output Viterbi algorithm with applications to hybrid ARQ. *IEEE Trans. Inf. Theory*, 44:1214–1216, 1998.

[67] A. Riaz, V. Bansal, A. Solomon, W. An, Q. Liu, K. Galligan, K. R. Duffy, M. Medard, and R. T. Yazicigil. Multi-Code Multi-Rate Universal Maximum Likelihood Decoder using GRAND. In *ESSCIRC*, 2021.

[68] A. Riaz, A. Solomon, F. Ercan, M. Medard, R. T. Yazicigil, and K. R. Duffy. Interleaved Noise Recycling Using GRAND. In *IEEE ICC*, pages 2483–2488, 2022.

[69] A. Riaz, A. Yasar, F. Ercan, W. An, J. Ngo, K. Galligan, M. Médard, K. R. Duffy, and R. T. Yazicigil. A sub-0.8pJ/b 16.3Gbps/mm^2 universal soft-detection decoder using ORBGRAND in 40nm CMOS. In *IEEE ISSCC*, 2023.

[70] R. Roth. *Introduction to Coding Theory*. CUP, 2006.

[71] M. S. Roulston and L. A. Smith. Evaluating probabilistic forecasts using information theory. *Monthly Weather Review*, 130(6):1653–1660, 2002.

[72] M. Rowshan and J. Yuan. Constrained Error Pattern Generation for GRAND. In *IEEE Int. Symp. on Inf. Theory*, 2022.

[73] W. Ryan and S. Lin. *Channel codes: classical and modern*. Cambridge University Press, 2009.

[74] H. Sarieddeen, M. Médard, and K. R. Duffy. Soft-Input, Soft-Output Joint Detection and GRAND. In *IEEE GLOBECOM*, 2022.

[75] H. Sarieddeen, P. Yuan, M. Médard, and K. R. Duffy. Soft-input, soft-output joint data detection and GRAND: A performance and complexity analysis. In *IEEE Int. Symp. on Inf. Theory*, 2023.

[76] A. Sauter, B. Matuz, and G. Liva. Error detection strategies for CRC-concatenated polar codes under successive cancellation list decoding. In *CISS*, 2023.

[77] M. Scholten, T. Coe, and J. Dillard. Continuously-Interleaved BCH (CI-BCH) FEC delivers best in class NECG for 40G and 100G metro applications. *Proc. Opt. Fiber Commun. Conf.*, pages 1–3, 03 2010.

[78] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[79] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge. Staircase Codes: FEC for 100 Gb/s OTN. *J. Light. Technol.*, 30(1):110–117, 2012.

[80] A. Solomon, K. R. Duffy, and M. Médard. Soft maximum likelihood decoding using GRAND. In *IEEE Int. Commun. Conf.*, 2020.

[81] V. Sudharsan, V. Karthik, J. Vaishnavi, S. Abirami, and B. Yamuna. Performance enhanced iterative soft-input soft-output decoding algorithms for block turbo codes. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 8(5):105–110, 2016.

[82] I. Tal and A. Vardy. List Decoding of Polar Codes. *IEEE Trans. Inf. Theory*, 61(5):2213–2226, 2015.

[83] V. Tam Van and S. Mita. A novel error correcting system based on product codes for future magnetic recording channels. *IEEE Trans. Magn.*, 47, 2012.

[84] D. Truhachev, K. El-Sankary, A. Karami, A. Zokaei, and S. Li. Efficient Implementation of 400 Gbps Optical Communication FEC. *IEEE Trans. Circuits and Sys.*, 68(1):496–509, 2021.

[85] G. Tzimpragos, C. Kachris, I. B. Djordjevic, M. Cvijetic, D. Soudris, and I. Tomkos. A survey on fec codes for 100 g and beyond optical networks. *IEEE Comm. Surveys and Tutorials*, 18(1):209–221, 2016.

[86] K. K. Y. Wong. *The soft-output M-algorithm and its applications.* Queen's University, 2006.

[87] H. Yamamoto and K. Itoh. Viterbi decoding algorithm for convolutional codes with repeat request. *IEEE Trans. Inf. Theory*, 26(5):540–547, 1980.

[88] C. Yang, Y. Emre, and C. Chakrabarti. Product Code Schemes for Error Correction in MLC NAND Flash Memories. *IEEE Trans. Very Large Scale Integr. Syst.*, 20(12):2302–2314, 2012.

[89] P. Yuan, M. Medard, K. Galligan, and K. R. Duffy. Soft-output (SO) GRAND and long, low rate codes to outperform 5 LDPCs, 2023. arXiv:2310.10737.