# Guessing random additive noise decoding (GRAND) in concatenated schemes: An investigation of complexity and BER behavior

Giovani Chrestani

Department of Electrical and Information Technology
Lund University

Supervisor: Michael Lentmaier

Examiner: Thomas Johansson

October 16, 2024

Printed in Sweden
Tryckeriet i E-huset, Lund

# Abstract

Recently, a newly developed family of decoders named as Guessing Random Additive Noise Decoder (GRAND) has been gaining attention. The key idea of the decoder is to look for the most probable error vectors that interacted with the transmitted message, producing the received block. Among the different variations of the algorithm, the soft-input soft-output (SISO) decoder, known as SO-GRAND, emerges as a good candidate for using with iterative schemes by employing list decoding. In this work, SO-GRAND's complexity and performance are analyzed against a trellis-based soft output decoder. These decoders are used as concatenated components of a generalized LDPC code. Results show that, while the soft-input hard-output version of GRAND (known as ORBGRAND) demonstrates good complexity qualities, the same is not true for the SO-GRAND that, in general, does not match the amount of computations of the trellis algorithm for typical scenarios of list size. Nevertheless, SO-GRAND has the nice property of trading off complexity in the expense of performance. This approach may look interesting when performance is good enough. Better performance of SO-GRAND algorithms are expected for higher order approximations in the ORBGRAND algorithm. The study also features several simulations with different types of codes (CRC, Hamming and random), demonstrating the universality quality of both trellis-based and SO-GRAND algorithms.

# Popular Science Summary

Error-correcting codes are part of standardized protocols found in everyday technologies, such as QR codes, the 5G standard, and compact discs. These codes function by adding redundancy to data, enabling the correction of errors at the receiver. In communication systems, various strategies are employed to infer the correct set of transmitted symbols (or bits). The optimal solution is called maximum likelihood (ML) decoder. Although this strategy gives the best possible performance, it is often the case that the ML decoder algorithm is not efficient in terms of computational complexity.

In the look for these efficient decoder algorithms, researches have rediscovered, at the end of the 90's, the "low-density parity-check" (LDPC) codes. This capacity achieving coding scheme is nowadays one of the most used options for providing high throughput with relatively long code. A testament to its success is the incorporation of LDPC codes into the 5G standard. The generalized version of this decoder, known as GLDPC, allows for flexibility as it supports any type of component code in its design.

This thesis investigates the performance and complexity trade-offs of the recently developed SO-GRAND (Soft-Output Guessing Random Additive Noise Decoder), compared to a trellis-based algorithm. Both decoders analyzed are used as concatenated schemes of GLDPC codes. The analysis shows that, while both SO-GRAND and trellis algorithms share similar proportional complexity with respect to block length and redundancy, SO-GRAND demands more computations due to factors like list size and the calculation of probabilities for noise vectors. However, GRAND algorithms are more flexible, offering configurable trade-offs between complexity and bit-error rate (BER) performance, making them promising candidates for future research. Additionally, the soft-input version of the algorithm shows competitive complexity compared to the trellis approach.

By detailing the strengths, weaknesses, and practical challenges of GRAND algorithms, this study hopefully contributes to advancing the understanding of these decoders and their potential in modern communication systems.

# Table of Contents

# List of Figures

Chapter 1

# Introduction

Since Shannon's pioneering work that opened the way for the information era [1], researchers look for methods for transmitting information in a noisy environment. One of the key strategies used in communication devices we interact with daily relies on the addition of redundant information, or channel coding. This additional information is crucial for recovering damaged data upon reception, enhancing the reliability of the message, at the expense of overall throughput.

## Background and Motivation

In 2018, Duffy, Li and Médard [2] developed a new channel decoding technique named Guessing Random Additive Noise Decoder (GRAND). As the name suggests, the decoder focuses on finding the noise vector that, when combined with the transmitted block, produces the hard-decoded received block. Soon after this first paper, GRAND's performance was tested using the 5G CA-Polar codes [3]. The algorithm resulted in low computational complexity in such short-length, high-rate codes. Although the first version was limited to hard decoding techniques, the authors extended the concept for soft input [4] and, at last, the soft output decoder [5] [6]. The "code-agnostic" property of the algorithm positions it as a viable alternative for a universal decoder of block codes. The authors also claimed that a good trade-off between performance and complexity could be achieved, as some versions of the algorithm provide maximum likelihood (ML) performance.

## Contributions

By leveraging the apparent advantages of GRAND in high-rate codes, this thesis aims at investigating the performance and complexity trade-off of GRAND, particularly as a concatenated scheme of generalized low-density parity-check (GLDPC) codes. The LDPC codes, initially introduced by Gallager in [7], gained increasing prominence after the 1990s and have since become part of modern communication standards, including 5G and Wi-Fi. The key idea of LDPC is to reduce a large low-rate code to several high-rate concatenated blocks, iterating over the soft output produced by these concatenated schemes. The iterative scheme has the advantage of stopping the decoding process according to processing time restrictions in the communication system. With a more generic formulation, the GLDPC code

accepts any kind of component decoder, as opposed to the parity-check equations of the LDPC.

The main element of a decoder that can dictate whether it is good, apart from the bit error rate (BER) performance, is the computational complexity. This metric determines whether the decoder can execute its tasks within the maximum processing time allowed in the communication system. Therefore, the evaluation of a good decoder often relies on the trade-off between BER performance and computational complexity. However, an in-depth analysis of this trade-off for various versions of the GRAND decoder remains lacking, and this thesis addresses that gap, focusing primarily on the soft output version known as SO-GRAND. If a good performance of GRAND can be found with a reasonable complexity, then this algorithm may be a good option for competing with other technologies already in use, such as Chase-Pyndiah decoding, used mostly in the context of product codes [8]; ordered statistics decoding [9] [10], that provides an interesting BER-complexity trade-off by approximating ML performance and the reduced trellis-based algorithm M*-BCJR [11] [12].

## Methodology

For comparison with soft-output GRAND (SO-GRAND), another universal decoder utilizing a trellis-based algorithm has been also implemented. The analysis of both trellis and SO-GRAND decoders starts with an inspection, trying to map the dependencies of each variable on the complexity of the algorithm. This analysis has been verified by simulations that show how the variables relate empirically. After presenting the complexity analysis, the BER/BLER performance of both decoders is shown for one and several iterations within the GLDPC scheme. Throughout the study, different types of codes are presented (Hamming, CRC, and random codes), showcasing the flexibility of the "code-agnostic" quality of the decoders. The scope of this work is limited to the analysis of the basic ORBGRAND version in the concatenated scheme. The basic ORBGRAND relies on an approximation and does not deliver ML performance, especially for higher SNR values. Nevertheless, the basic ORBGRAND presents the lowest complexity when compared to the generic ORBGRAND formulation, unless parallel computing could be employed. In that case, a detailed investigation should be carried on the generic ORBGRAND, which is out of the scope of this work.

## Overview

The text follows from the introduction to a theoretical background in Section 2, which explains the standard channel coding theory required throughout the rest of the text. Section 3 presents an overview of GRAND and its variations. In the following section, the computational complexity of the trellis and GRAND decoders is studied and compared, together with BER/BLER behavior. Although the implementation of a general form of ORBGRAND was left out of this work, some insights of this approach are given in Section 5. The last section discusses the results and presents other possible approaches for studying GRAND complexity, as well as some insights on what are the bottlenecks of the algorithm.

# Chapter 2
# Theoretical Background

This chapter presents the background needed in order to understand the GLDPC codes and the trellis-based decoder, one of the decoding methods studied in this thesis, used as a performance metric for comparison with GRAND. At first, the reader is introduced to the communication system model and the basic assumptions used throughout the work. An introduction to block codes and CRC follows, aiming at describing some of the properties of these encoding schemes. Random codes are presented as a nice way of analyzing the decoders studied. The trellis-based block code is introduced in the following section, followed by the theory behind Low-Density Parity-Check Codes (LDPC). Finally, the generalized LDPC is presented.

## 2.1 Communication System Model

A wireless communication system can be efficiently described in the form of a block diagram. Figure 2.1 illustrates the blocks and their connections.

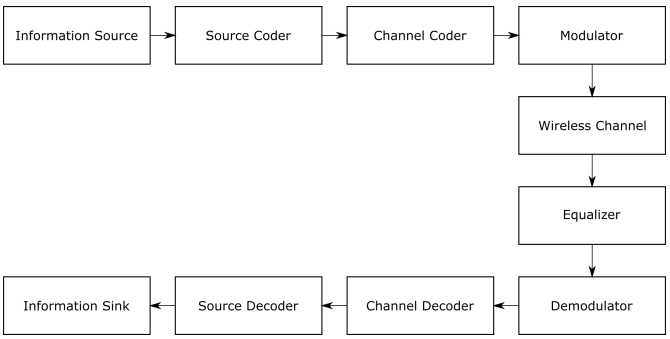


**Figure 2.1:** General block diagram of a wireless communication system

The information source provides a digital or analog source signal with the original information to be transmitted. If it is an analog signal, like a voice signal from a microphone, it will pass through an analog to digital conversion (ADC). The source coder is responsible for compressing the digital information to an adequate

level, considering the quality required when restoring the original information. The idea of this block is to reduce the use of bandwidth for a particular message transmitted. The channel coder, on the other hand, adds structured redundancy to the "pure" information, so that it can be decoded at the receiver side with tolerable bit error rates. The modulator then converts the digital information into analog waves, shaping them according to the required use of spectrum. These analog waves are then converted to a radio frequency band and transmitted over the antenna. The wireless channel distorts these radio frequency waves according to the boundary conditions, attenuates the signal by distance, and adds noise and interference. Noise is also introduced by the radio components that are part of the receiver radio. In some cases, the wireless channel is frequency selective, meaning that the signal transmitted is attenuated differently for different frequencies. At the receiver side, the equalizer compensates for the frequency selective fading but can also deteriorate the signal-to-noise ratio for some frequency ranges. The role of channel decoder, source decoder and information sink is simply the opposite of their counterparts at the transmitter side.

A block diagram can describe several layers of details, from the high to the low level structure. In Figure 2.1, for example, the analog to digital conversion is not distinguished from the blocks shown, as well as the multiple access control layer or header structures of the messages. This work is mainly focused on the channel coder/decoder block, so many simplifications can be applied to the general block diagram of Figure 2.1. The source coder/decoder blocks do not affect any operation of the channel coder/decoder, so they are just eliminated. Instead, a random bits generator is used for creating the stream of bits to be simulated. The wireless channel can be simplified by an Additive White Gaussian Noise (AWGN) block. This implies that the channel has no memory and channel fading is not considered. Given that, the equalizer block can also be eliminated. The resulting reduced block diagram of the system is shown on Figure 2.2.

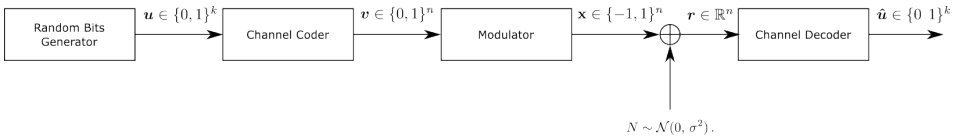


**Figure 2.2:** Reduced block diagram of the wireless communication model used

As shown in the scheme, the channel decoder is also responsible for demodulating the signal from soft input, in the domain of real numbers, to the actual set of bits estimated. The modulation scheme utilized is the Binary Phase Shift Keying (BPSK) and the mapping is given as in Table 2.1.

The noise is a random variable that depends on the standard deviation $\sigma$ and has zero mean. The standard deviation is proportional to the noise power, which can be defined by the Signal-to-Noise Ratio (SNR). The idea is to sweep the SNR values over a certain range and measure the Bit Error Rate (BER) and the Block Error Rate (BLER). The BER and BLER are two common figures of merit that define the performance of a decoder.

| Bit | Symbol |
|---|---|
| 0 | 1 |
| 1 | -1 |

**Table 2.1:** BPSK mapping

## 2.2 Linear Block Codes

The idea that a communication link can be established without errors for a given signal-to-noise ratio (SNR) and a capacity limit is known since [1]. From that moment on, people have tried to come with different methods for encoding the original information so that the bit rate approaches the capacity limit, or "Shannon limit". The process of encoding requires the addition of redundant bits. The correlation between the bits of the encoded message can strengthen the likelihoods of bits received with poor signal quality. Two different classes of codes have been established: block and convolutional codes. Block codes add redundancy to blocks of data while convolutional codes add redundancy continuously throughout all the message transmitted [18]. In this work, the focus is on a newly proposed family of decoders called GRAND, that applies for block codes. The encoding schemes of such codes can be easily described by a matrix multiplication in which the set of bits to be encoded, described by the row vector $\mathbf{u}$, is transformed to the codeword $\mathbf{v}$, as shown in the equation below

$$\mathbf{v} = \mathbf{u}\mathbf{G} \tag{2.1}$$

in which $\mathbf{G}$ is the generator matrix. The original vector $\mathbf{u}$ has $K$ bits, while the encoded vector $\mathbf{v}$ has $N$ bits, including the redundancy bits previously mentioned. Therefore, the generator matrix is $K \times N$. It is defined as a codeword, a vector $\mathbf{v}$ in $\{0,1\}^N$ that is part of the subspace generated by the rows of matrix $\mathbf{G}$. Such subspace is denoted by $\mathcal{C}$, the code-book.

Throughout this work, the word redundancy (denoted by $m$) is referred to the amount of bits added to the codeword $\mathbf{v}$, in relationship to the original set of bits $\mathbf{u}$, as in the following equation: $m = N - K$.

A block code is usually specified by the parameters length $N$, dimension $K$ and redundancy $m$. For example, a (7,4,3) Hamming code has $N = 7$, $K = 4$ and $m = 3$. Other important definitions of coding theory used throughout the text are summarized below:

- Rate: The rate of a code is defined as the ratio of input bits to the encoded bits, $R = K/N$. A high-rate code is meant by a code with $R \to 1$, meaning that the amount of redundancy is low.
- Hamming distance: The Hamming distance between two binary vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ is defined by

$$d_H(\mathbf{v}_1, \mathbf{v}_2) = \sum_{j=1}^{N} |v_{1,j} - v_{2,j}| \tag{2.2}$$

A minimum Hamming distance $d_{min}$ is determined by the smallest distance between any two codewords of a given code-book $\mathcal{C}$. The larger this value, the easier it is to distinguish between two different codewords and, therefore, the more robust to errors is the encoding scheme. On the other hand, increasing $d_{min}$ often implies increasing the amount of redundant bits $m$, resulting in a reduced bit rate.

A code with minimum Hamming distance $d_{min}$ can guarantee a correction of, at maximum, $t$ wrong bits per block, where $t$ is given by

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \tag{2.3}$$

- Hamming weight: This is the Hamming distance from the all-zero codeword. It is the equivalent of counting the amount of ones in the binary vector.

$$w_H(\mathbf{v}) = \sum_{j=1}^{N} v_j \tag{2.4}$$

- Logistic weight: This is defined by the sum of each bit "1", weighted by its position in the binary vector

$$w_L(\mathbf{v}) = \sum_{j=1}^{N} j \cdot v_j \tag{2.5}$$

- Linear codes: This family of codes obey the following rules
  - Any linear combination of two codewords is a codeword.
  - The all-zero vector is a codeword.
  - All codewords, except the all-zero one, have a Hamming weight equal or larger than the minimum distance $d_{min}$.
- Perfect codes: For all types of codes, the amount of redundant bits for a given minimum distance is bounded by

$$m \geq \log_2 \left( 1 + \binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{t} \right), \text{with } t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \tag{2.6}$$

  The perfect code satisfies the condition above with equality.
- Parity-check matrix: The parity-check matrix $\mathbf{H}$ is the matrix that satisfies $\mathbf{G}\mathbf{H}^{\mathbf{T}} = \mathbf{0}$ for a given generator matrix $\mathbf{G}$
- Systematic encoding: A code in which the original bits of the input vector $\mathbf{u}$ can be distinguished from the redundant bits added by the generator matrix. These "original bits" of length $K$ are then at fixed locations, usually at the first positions. In that case, the generator matrix can be represented by

$$\mathbf{G} = (\mathbf{I}_K \quad \mathbf{P}), \tag{2.7}$$

  in which $\mathbf{I}_K$ is the identity matrix of length $K \times K$. The parity-check matrix is then given by

$$\mathbf{H} = (-\mathbf{P}^T \quad \mathbf{I}_m) \tag{2.8}$$

- Hamming Codes: These are codes represented by a parity-check matrix whose columns cover all distinct nonzero vectors of length $m$. As an example, a Hamming (7,4,3) parity-check matrix could be

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

  Hamming codes are perfect codes.

- Syndrome: The syndrome of a received vector $\hat{\mathbf{v}}$ is defined by

$$\mathbf{s} = \hat{\mathbf{v}}\ \mathbf{H}^T \tag{2.9}$$

  If the received vector has errors, it follows that

$$\mathbf{s} = (\mathbf{v} + \epsilon)\mathbf{H}^T = \mathbf{0} + \epsilon\mathbf{H}^T = \epsilon\mathbf{H}^T \tag{2.10}$$

  and the syndrome depends only on the error vector. The second equality follows from the fact that codewords have an all-zero syndrome vector. For a given transmitted codeword, there are multiple error vectors that could lead to the same syndrome. These error words are said to be part of the same coset. Considering the probability of receiving a correct bit higher than the opposite, it is then reasonable to think that the most probable error vector is the one with minimum Hamming weight. This is the principle behind hard decision GRAND, that will be further developed later on.

  As an example, consider a (7,4) Hamming code. For this code, it is possible to correct all error vectors with weight 1 (refer to (2.3)). For any error vector with unitary weight $\epsilon$ there is a syndrome vector $\mathbf{s}$ that corresponds to one of the columns of the parity-check matrix (see (2.10)). The index of this column corresponds to the wrong bit and, therefore, it is possible to simply flip back the bit and get the most probable transmitted vector.

- Code extension: A $(N, K, d_{min})$ code can be extended to a $(N+1, K, d_{min}+1)$ code by adding a parity-check symbol to each codeword

$$v_{N+1} = v_1 + v_2 + \cdots + v_N.$$

  The parity-check matrix can be written as

$$\mathbf{H}_{\text{ext}} = \begin{bmatrix} & & & & & 0 \\ & & \mathbf{H} & & & 0 \\ & & & & & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

## 2.3 Cyclic Redundancy Check Codes

Cyclic redundancy check (CRC) codes have been used extensively on communication and storage systems. Usually, CRC codes are employed as a way of checking the integrity of the message transmitted or stored. In this work, the idea is to

explore the use of CRC codes not only for detecting failed communication but also correcting the wrong bits, whenever it is possible. In this section, an introduction to CRC codes is presented together with its properties. The polynomial and matrix formulations of CRC codes are explored, as they are both used in the simulation MATLAB code.

CRC codes are defined as follows,

> A linear block code with codebook $\mathcal{C}$ and $\mathbf{v} \in \mathcal{C}$ is defined as a cyclic code if the vector $\mathbf{v}^i = (v_{N-i}, v_{N-i+1}, \cdots, v_{N-1}, v_0, v_1, \cdots, v_{N-i-1}) \in \mathcal{C}$

Usually, CRC codes are presented in the polynomial form. An input vector $\mathbf{u}$ can be described as

$$u(X) = u_0 + u_1 X + u_2 X^2 + \cdots + u_{K-1} X^{K-1}.$$

The coefficients of the polynomial in $X$ are binary. As an example, the six-bit vector $\mathbf{u} = [0100110]$ is represented by the polynomial

$$u(X) = X + X^2 + X^5.$$

The CRC encoded vector $\mathbf{v}$ is defined by

$$v(X) = u(X)X^m + c(X), \tag{2.11}$$

where $u$ is the input vector, $c(X)$ represents the appended bits by the CRC encoder and $m$ is the number of bits added to the original vector.

The encoding process can be also given by the use of a generator polynomial $g(X)$ of degree $m$, as shown below:

$$v(X) = g(X)z(X). \tag{2.12}$$

> A polynomial $g(X)$ of degree $m$ generates an $(N, K)$ cyclic code if it is a factor of $X^N - 1$

In that way, the validity of a codeword can be checked by dividing it by the generator polynomial. Valid codewords give a null remainder. By putting together equations (2.11) and (2.12) and rearranging, the equation below follows:

$$u(X)X^m = g(X)z(X) + c(X). \tag{2.13}$$

The result above follows from the property that addition and subtraction are equal operations in the binary field. Finally, the appended bits encoded by the CRC are given as

$$c(X) = \text{remainder}\left[\frac{u(X)X^m}{g(X)}\right]. \tag{2.14}$$

Equivalently to the matrix formulation of block codes, it is possible to define a parity-check polynomial $h(X)$

$$g(X)h(X) = X^N + 1 \tag{2.15}$$

The equation above shows that the degree of $h(X)$ is $N - m = K$, so the polynomial has $K + 1$ coefficients, as the 0th order coefficient is also counted.

Finally, the parity-check matrix of a CRC code is defined as

$$\mathbf{H} = \begin{bmatrix} \bar{h}(X) \\ X\bar{h}(X) \\ X^2\bar{h}(X) \\ \vdots \\ X^{m-1}\bar{h}(X) \end{bmatrix}, \tag{2.16}$$

with $\bar{h}(X) = h_m + h_{m-1}X + h_{m-2}X^2 + \cdots + h_1X^m$.

As an example, consider $K = 4$, $m = 3$ and $h(X) = 1 + X^2 + X^3 + X^4$. According to the definition just presented,

$$\bar{h} = 1 + X + X^2 + X^4.$$

In vector form, it can be presented as $\bar{\mathbf{h}} = [10111]$. The resulting $\mathbf{H}$ matrix will be $m \times N$, with $N = K + m = 4 + 3 = 7$,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

In both trellis-based decoding and GRAND methods, the validity of a vector is checked by the use of a parity-check matrix. For cyclic codes, when the parity-check matrix multiplication results in a null syndrome, it is easy to simply take out the last $m$ bits of the codeword and get the decoded vector.

## 2.4 Random Codes

Another encoding method used throughout this work is the random block code. These codes can be constructed by assigning 1s and 0s randomly in a parity-check matrix $\mathbf{H}$. In order to use systematic encoding, it is easy to create a matrix $\mathbf{R}$ with random entries and define the parity-check matrix as

$$\mathbf{H} = (\mathbf{R} \quad \mathbf{I}_m).$$

The generator matrix is then given by $\mathbf{G} = (\mathbf{I}_K \quad \mathbf{R}^T)$, in accordance with (2.7) and (2.8).

Random codes, as block length increases, statistically have codewords that are well-spread out, which helps in distinguishing between different transmitted messages. When using maximum likelihood (ML) decoding (refer to Section 2.6), which is the optimal decoding method, random codes can achieve capacity for $N \to \infty$.

Although the performance of these codes in terms of $d_{\min}$ for a given $m$ is not optimal for short block lengths, it is interesting to use this encoding scheme to analyze how the complexity of the trellis-based decoder and GRAND depend on the variables $N$ and $m$.

## 2.5 Decoder types

After covering the essential encoding methods used throughout this study, here it is presented the decoder types. Decoders can be categorized based on the type of input information they use and the type of output they provide. This section presents the terminology.

### 2.5.1 Hard Decision Decoder

A **hard decision decoder** uses binary decisions as inputs and outputs. The received signal is first processed by a hard decision demodulator that converts the noisy received signal into a binary sequence (0s and 1s). The hard decoder then processes this binary sequence to detect and correct errors based on the code. The output of the hard decoder is also a binary sequence.

- **Input:** Binary sequence (hard decisions).
- **Output:** Binary sequence (corrected codeword).

As an example, the (7,4) Hamming decoding scheme, that generates the syndrome vector and corrects the bit that matches the corresponding column of $\mathbf{H}$, is a hard decision decoder.

### 2.5.2 Soft Input Decoder

A **soft input decoder** uses soft decisions as inputs, which provide more detailed information about the received signal. Soft decisions typically come in the form of probabilities or confidence levels for each bit or symbol. The decoder uses this probabilistic information to make more informed decisions about the transmitted data, potentially improving error correction performance.

- **Input:** Soft decisions (e.g., log-likelihood ratios, probabilities).
- **Output:** Binary sequence (corrected codeword).

Later on in the text, the trellis-based soft input decoder and the ORBGRAND are presented. Both of them receive log-likelihood ratios and output the estimated binary block.

### 2.5.3 Soft Output Decoder

A **soft output decoder** not only uses soft decisions as inputs but also provides soft decisions as outputs. This means that the decoder produces probabilistic information or confidence levels about the correctness of each decoded bit or symbol.

This additional information can be useful for subsequent stages of the receiver, such as iterative decoding or further error correction processes.

- **Input:** Soft decisions (e.g., log-likelihood ratios, probabilities).
- **Output:** Soft decisions (e.g., updated log-likelihood ratios, probabilities).

The SO-GRAND and the soft-output trellis-based decoder (Johansson-Zigangirov algorithm), introduced later, are examples of soft output decoder schemes. In both cases, the decoder outputs the soft information in terms of extrinsic and intrinsic log-likelihoods ratios (LLRs). The extrinsic information is used in later iterations and represents, for each bit being analyzed, the information provided by all the other bits in the block, excluding the influence of its own LLR. At the last iteration, the intrinsic information is put together with the extrinsic, providing the final posterior probability.

Each type of decoder has its own advantages and use cases. Hard decoders are simpler and faster but may not perform as well as soft input or soft output decoders. Soft input decoders improve performance by utilizing more information about the received signal. Soft output decoders provide the most information, facilitating advanced decoding techniques and iterative processing.

## 2.6 Maximum Likelihood Receivers

A Maximum Likelihood (ML) receiver aims to determine the most likely transmitted signal given the received signal. The decision rule for an ML receiver is to choose the codeword that maximizes the likelihood function.

The ML receiver for soft inputs selects the codeword $\mathbf{c}$ that maximizes the likelihood given the received soft sequence $\mathbf{r}$,

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}\in\mathcal{C}} \Pr(\mathbf{r}|\mathbf{v}).$$

From the equation above, it can be shown that the ML receiver for hard decisions can be simplified. It selects the codeword $\mathbf{v}$ from the codebook $\mathcal{C}$ that minimizes the Hamming distance to the received binary sequence $\mathbf{r}$,

$$\hat{\mathbf{v}} = \arg\min_{\mathbf{v}\in\mathcal{C}} d_H(\mathbf{r}, \mathbf{v})$$

where $d_H(\mathbf{r}, \mathbf{v})$ is the Hamming distance between $\mathbf{r}$ and $\mathbf{v}$.

## 2.7 Trellis-based Block Codes

In this section one of the soft input approaches studied in this thesis is presented: the trellis-based block decoder. These decoders are used in the decoding of block codes by representing the code as a trellis diagram, which allows for efficient decoding algorithms. The most commonly used decoding algorithm for trellis-based decoders is the **Viterbi algorithm**. This algorithm finds the most likely sequence of states (and corresponding codeword) by efficiently searching through the trellis.

The choice of such algorithm for comparison is motivated by its universal characteristics, as it is a code "agnostic" decoder. This feature is one of the key motivations behind GRAND, brought forward by its inventors.

### 2.7.1 Decoder Input

As stated in Section 2.6, the optimal soft input receiver can be obtained by choosing the codeword that maximizes the probability of receiving the soft vector $\mathbf{r}$. In order to analyze the decoder performance in terms of log-likelihoods, the memoryless channel is assumed,

$$\Pr(\mathbf{r}|\mathbf{v}) = \prod_{i=0}^{N-1} p(r_i|v_i). \tag{2.17}$$

As the AWGN channel is assumed, the density function $p(r_i|v_i)$ is given by

$$p(r_i|v_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(r_i - x_i)^2/2\sigma^2}, \tag{2.18}$$

where the $x_i$ values are the constellation mapping of the input binary vector $\mathbf{v}$ (see Figure 2.2). The log-likelihood ratios are defined as

$$\ell_i = \ln \frac{p(r_i|v_i = 0)}{p(r_i|v_i = 1)}. \tag{2.19}$$

Considering the BPSK convention of Table 2.1, 2.18 and 2.19 simplify to

$$\ell_i = \frac{-(r_i - 1)^2 + (r_i + 1)^2}{2\sigma^2} = \frac{2r_i}{\sigma^2} = \frac{4r_i}{N_0}. \tag{2.20}$$

The noise power spectral density varies with the signal-to-noise ratio (SNR), which will be swept in a range specified later on.

### 2.7.2 The State Variable

Every trellis-based decoder is built with two important variables: the state and the branch metric. The path metric can be derived from the branch metric. The diagram displayed on Figure 2.3 helps to understand the concept. In that diagram, there are three different states at level 2: $\sigma_2 = \{0, 1, 3\}$.

In the context of linear block codes, a syndrome vector with all-zero entries characterizes a codeword (a valid received vector), as demonstrated previously on Section 2.2. Consider the representation of the parity-check matrix in terms of its column vectors:

$$\mathbf{H} = [\mathbf{h}_1 \quad \mathbf{h}_2 \quad \cdots \quad \mathbf{h}_N] \implies \mathbf{H}^T = \begin{bmatrix} \mathbf{h}_1{}^T \\ \mathbf{h}_2{}^T \\ \vdots \\ \mathbf{h}_N{}^T \end{bmatrix}.$$

The syndrome vector $\mathbf{s}$, defined in (2.9), can be written as

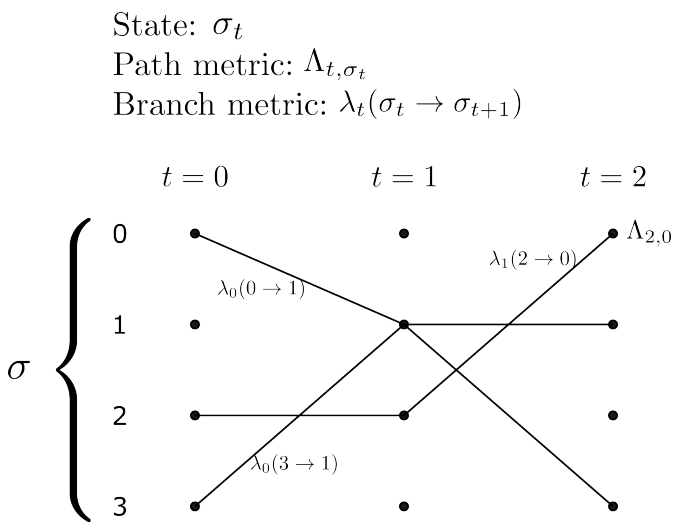


**Figure 2.3:** Trellis variables

$$\mathbf{s} = [\hat{v}_1 \quad \hat{v}_2 \quad \cdots \hat{v}_N] \begin{bmatrix} \mathbf{h}_1{}^T \\ \mathbf{h}_2{}^T \\ \vdots \\ \mathbf{h}_N{}^T \end{bmatrix} = \sum_{i=1}^{N} \hat{v}_i \mathbf{h}_i{}^T. \tag{2.21}$$

It is then possible to define a state variable $\sigma_t$ that is the truncated sum of the syndrome vector, as shown below

$$\sigma_t = \sum_{i=1}^{t} \hat{v}_i \mathbf{h}_i{}^T. \tag{2.22}$$

The initial state is defined as state 0, $\sigma_0 = 0$. At each level of the trellis $t$, there are two possible paths to be taken: one for bit 0 and another for bit 1. The paths that result from the bit 0 (same state) are going to be represented by dashed lines, while the continuous segments are given to the paths corresponding to bit 1. For example, let us define the parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}.$$

The corresponding trellis is presented in Figure 2.4.

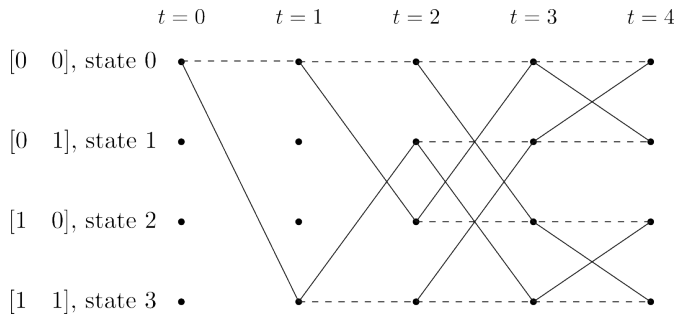


**Figure 2.4:** Trellis on block code example

Note that, due to (2.21), valid codewords end up in the zero state, after all levels of the trellis ($t = N$). It is then more effective to filter the paths in the trellis that do not end up in the zero state. This new trellis is called "expurgated" trellis, and is presented in Figure 2.5.

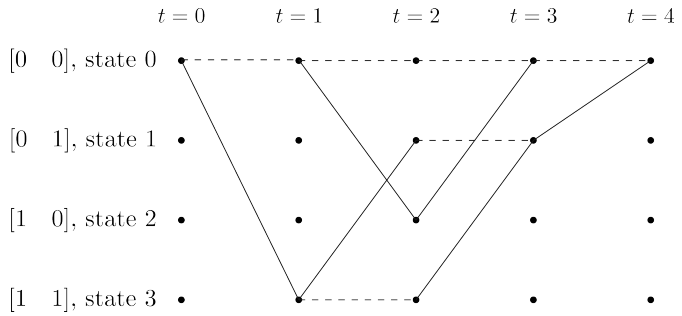


**Figure 2.5:** Expurgated trellis

### 2.7.3 The Branch Metric

Under the ML receiver assumption, the interest is in finding the branch metric that maximizes the probability of finding the correct codeword transmitted, for a given block. The key idea in the Viterbi algorithm is to decide, for several paths that merge to the same state, which one is more probable. This decision is based on the accumulated path metric, which is the summation of all branch metrics from the zero state until the point of analysis.

The memoryless assumption states that maximizing the likelihood of a block is the same as maximizing the likelihood for each individual bit that is part of the block. From 2.18, it is seen that maximizing $p(r_i|v_i)$ is equivalent to minimizing the quantity $(r_i - x_i)^2$, called Euclidean distance between $r_i$ and $x_i$. The value of $x_i$ can only be -1 or $+1$, according to the BPSK defined. The Euclidean distance can be expanded to

$$(r_i - x_i)^2 = r_i^2 - 2r_i x_i + 1.$$

Note that $r_i$ is independent of the path in the trellis, so the minimization of Euclidean distance can be simplified to maximizing $r_i x_i$, for all $i \in \{0, 1, 2, 3, \cdots, N-$

1}. The equation below synthesizes these ideas:

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}\in\mathcal{C}} \sum_{i=0}^{N-1} r_i x_i. \tag{2.23}$$

The truncated sum of 2.23 in a particular state $\sigma_t$ is the path metric up to level $t$

$$\Lambda_{t,\sigma_t} = \sum_{i=0}^{t} r_i x_i. \tag{2.24}$$

### 2.7.4 Soft Output Decoder

One of the main goals of this work is to analyze the performance of both GRAND and trellis decoders with concatenated schemes, in GLDPC codes. As it is presented in Section 2.9, the GLDPC's component code has to output soft information, so that it can be used in the iterative scheme.

The soft output trellis scheme used in this work is known as Johansson-Zigangirov algorithm, after its original authors. A detailed description can be found in [13] and [17]. The algorithm consists in calculating the extrinsic LLRs after an intermediate step, in which a variable called "flow" $\mu$ is computed in all possible paths of the trellis. The paths are determined by the columns of the parity-check matrix, as described previously in this section. The interesting feature of this algorithm is that the flow can be computed recurrently in the trellis, by performing simple sums and XOR operations. The pseudo code in page 16 shows the Johansson-Zigangirov algorithm.

As it will be discussed later, it is possible to take out the invalid paths at the beginning of the trellis by inspecting the columns of the parity-check matrix. This approach may reduce a bit the complexity of the algorithm, but not significantly.

## 2.8 Low-Density Parity-Check Codes

Low Density Parity-Check (LDPC) codes were first introduced by Robert G. Gallager in his 1960 doctoral dissertation. Gallager's pioneering work demonstrated the potential of these codes to approach the Shannon limit. Despite their promise, LDPC codes were not widely adopted at the time, primarily due to the computational complexity involved in their implementation, which exceeded the capabilities of the technology available in the 1960s. Interest in LDPC codes returned in the late 1990s and early 2000s. Researchers realized that LDPC codes could be efficiently implemented using modern hardware, making them practical for a wide range of applications. Today, LDPC codes are widely used in various fields, including satellite communications, Wi-Fi (IEEE 802.11), among others.

In this work, both trellis-based soft output decoding and SO-GRAND are used to decode component codes of a generalized form of LDPC codes, the generalized LDPC (GLDPC) codes. In order to lay out the ground for such topics, this section introduces some important concepts of LDPC codes.

**Trellis-based block decoder, soft output (Johansson-Zigangirov algorithm):**

Inputs: $\mathbf{H}$ and $L_{in,i}$, with $L_{in,i} = p(r_i|v_i = 0)/p(r_i|v_i = 1)$ and $i = 0, ..., N-1$

**a.** Initialize: $\mu(\mathbf{0}, 1) = L_{in,0}^{1/2}$ and $\mu(\mathbf{h}_1, 1) = L_{in,0}^{-1/2}$

**b.** **for** $l = 2, ..., N$

**for** $\mathbf{s} = [000]^T, ..., [111]^T$, where $\mathbf{s}$ is the state in the trellis

$$\mu(\mathbf{s}, l) = \mu(\mathbf{s}, l-1)L_{in,l-1}^{1/2} + \mu(\mathbf{s} \oplus \mathbf{h}_l, l-1)L_{in,l-1}^{-1/2}$$

**c.** **for** $i = 0, ..., N-1$

$$\ell_{ext,i} = \ln \frac{\mu(\mathbf{0}, N)L_{in,i} - \mu(\mathbf{h}_{i+1}, N)}{\mu(\mathbf{h}_{i+1}, N)L_{in,i} - \mu(\mathbf{0}, N)}, \text{ extrinsic LLRs}$$

$$\ell_{intr,i} = \ln L_{in,i} + \ell_{ext,i}, \text{ intrinsic LLRs}$$

**d.** Return $\ell_{ext,i}$ and $\ell_{intr,i}$, for $i = 0, ..., N-1$

From the previous sections, the most probable transmitted block that gives an all-zero syndrome vector is given by:

$$\hat{\mathbf{v}}\mathbf{H}^T = \mathbf{0}. \tag{2.25}$$

Therefore, for each row of parity-check matrix ($\mathbf{h}_{\text{row},i}$) there is a parity-check equation

$$\begin{bmatrix}\hat{\mathbf{v}}\mathbf{h}_{\text{row},1}^T & \hat{\mathbf{v}}\mathbf{h}_{\text{row},2}^T & \cdots & \hat{\mathbf{v}}\mathbf{h}_{\text{row},N}^T\end{bmatrix} = \mathbf{0}. \tag{2.26}$$

Consider the matrix $\mathbf{H}$ below

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Considering only the first row of the parity-check matrix, a parity-check equation can be written as follows:

$$\hat{v_1} + \hat{v_2} + \hat{v_4} + \hat{v_5} = 0. \tag{2.27}$$

Another form of representing it is by the diagram in Figure 2.6, known as Tanner graph. In the figure it is indicated what is the check node and the variable node.

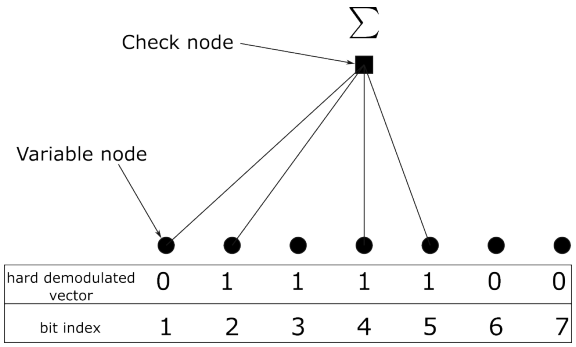


**Figure 2.6:** Tanner graph representation of parity-check (2.27)

In this example, the parity-check equation from variable nodes 2, 4 and 5 indicate that the first bit of the codeword should be 1, instead of 0, as the demodulator provided. This information is know as "extrinsic", as it depends on the other bits of the received vector. The "intrinsic" information about a particular bit is its output from the demodulator or, in the case of soft input, the log-likelihood received from the channel. In practice, all $m$ parity-check equations will play a role in this estimation. The estimated codeword will be the one that satisfies all parity-check equations simultaneously. The full Tanner graph of the parity-check matrix used as example would look like displayed on Figure 2.7.

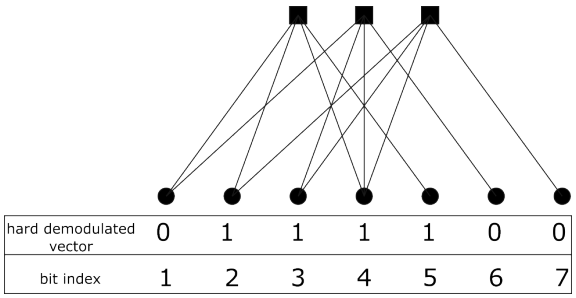


**Figure 2.7:** Full Tanner graph representation of the parity-check matrix used as example

## 2.9 Generalized Low-Density Parity-Check Codes

The generalized LDPC have the same idea of reducing big block length to smaller pieces with higher rate. The difference to LDPC is that, instead of utilizing only parity-check matrices on the component code, GLDPC allow for more complex decoders, such as the ones studied in this text: SO-GRAND and soft output trellis decoder.

The principle of operation of such codes is based on matrices $\mathbf{H}$ and $\mathbf{H}_{inn}$, the outer and inner parity-check matrices, respectively. The $\mathbf{H}$ is the parity-check

matrix associated with the LDPC code, of length $N$. The $\mathbf{H}_{inn}$ is associated with the inner code, or component code, that checks the validity of a specific subset of bits from the whole block. The values $N_{inn}, m_{inn}$ and $K_{inn}$ are attributes of $\mathbf{H}_{inn}$, while $N$, $m$ and $K$ are attributes of $\mathbf{H}$.

In this work an iterative scheme based on the principles of the previous chapter was applied to both SO-GRAND and trellis-based decoder. The idea was to substitute the parity-check equations by this generic decoder. The algorithm is described at page 19.

The generic_decoder() mentioned in the algorithm can be any soft output decoder. More information on GLDPC can be found in [14], [15] and [16].

**Iterative GLDPC decoder:**

Inputs: $\mathbf{H}$, $\mathbf{H}_{inn}$, $I$, , $\ell_l$ with $l \in (1, 2, \cdots, N)$, where $\ell_l$ is the channel LLR of $l$th bit and $I$ is the maximum iteration counter.

**a.** Find, for each row $t$ of $\mathbf{H}$, the indices that are equal to 1 and store them in variable $\mathcal{L}(t)$. For instance, if only $\mathbf{H}_{5,6}$ and $\mathbf{H}_{5,10}$ are 1 in the $5^{th}$ row, then $\mathcal{L}(5) = \{6, 10\}$.

**b.** Find, for each column $l$ of $\mathbf{H}$, the indices that are equal to 1 and store them in variable $\mathcal{T}(l)$. For instance, if only $\mathbf{H}_{5,2}$ and $\mathbf{H}_{7,2}$ are 1 in the $2^{nd}$ column, then $\mathcal{T}(2) = \{5, 7\}$.

**c.** Initialize the $m \times N$ matrix $\mathbf{L}^i$, that contain the updated LLRs for each iteration.

**for** $l = 1$ to $N$:

**for** $t$ in $\mathcal{T}(l)$:

$\mathbf{L}^i(t, l) = \ell_l$

**d.** Calculate extrinsic LLRs for each subset of bits in a row of $\mathbf{H}$:

**for** $t = 1$ to $m$:

$\mathbf{L}^i_{ext}(t, \mathcal{L}(t)) = \text{generic\_decoder}(\mathbf{L}^i(t, \mathcal{L}(t)), \mathbf{H}_{inn})$

**e.** Combine the LLRs on each column but not counting its own extrinsic LLR, obtained from the previous step:

**for** $l = 1$ to $N$:

**for** $t$ in $\mathcal{T}(l)$:

**if** $i < I$:

$\mathbf{L}^i(t, l) = \ell_l \cdot \prod_{t=\mathcal{T}(l)\setminus t} \mathbf{L}^i_{extr}$

**if** $i = I$:

$\mathbf{L}^i_{out}(l) = \ell_l \cdot \prod_{t=\mathcal{T}(l)} \mathbf{L}^i_{extr}$

**f.** Test the validity of all rows of $\mathbf{L}^i(t, \mathcal{L}(t))$:

**if** $\mathbf{L}^i(t, \mathcal{L}(t)) \cdot \mathbf{H}_{inn} = \mathbf{0}$ for all $t = (1, 2, \ldots, m)$:

return $\mathbf{L}^i_{out}$

Repeat the iterative steps until $i = I$.

Chapter 3

# Guessing Random Additive Noise Decoding Principles

The recently developed family of decoders known as GRAND has several variations, according to hard input-output, soft input and soft output. The common principle that underlies each of GRAND versions is that it is efficient to decode received vectors by choosing the most probable error vector. Due to the fact that, in a reasonable channel, the probability of receiving a correct bit is higher that a wrong bit, it makes sense to guess that the error vector is the one with smallest Hamming weight that produces a valid codeword. In case soft input is considered, it may not be as simple as that. The most probable error sequences are then obtained after some processing and a mechanism called "landslide algorithm". Finally, for the soft output case, a list decoding approach is applied, based still on the ORBGRAND algorithm. The following sections will provide a more detailed explanation of how these decoders work.

## 3.1 Hard Input GRAND

In order to test the hard input GRAND, it is sufficient to assume the binary symmetrical channel (BSC), which determines if a bit will be flipped with a probability $p$. Figure 3.1 illustrates the idea.

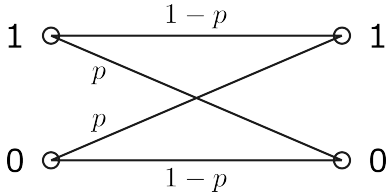


**Figure 3.1:** Binary symmetrical channel

Consider the demodulated received vector $\mathbf{r}$ and the transmitted vector $\mathbf{v}$. It can be defined that the error vector $\mathbf{z}$ is given by

$$\mathbf{z} = \mathbf{r} \oplus \mathbf{v}. \tag{3.1}$$

The idea in the GRAND algorithm is to rank-order noise vectors by their probability and check, in that order, if they provide a valid codeword, for the given received vector. This procedure is repeated until the first valid codeword is found. As the information is hard decoded (binary), the most likely noise sequences are the ones with lower Hamming weight. In order to rank these noise vectors, all $N$ sequences of weight 1 are considered at first, then all $\binom{N}{2}$ sequences of weight 2, and so on.

This method is proven to give ML decoding. More information on that can be found in [2]. The following pseudo algorithm synthesizes the GRAND (hard decoder) idea.

**GRAND (Hard Decoder):**

Inputs: $\mathbf{r}$, $\mathbf{H}$

**a.** Sort error vectors $\mathbf{z}_i$ by their probabilities, prioritizing lower Hamming weights.

**b.** Initialize: $i = 1$ and set $\mathbf{z}_1$, the most likely sequence.

**c.** While $(\mathbf{r} \oplus \mathbf{z}_i) \notin \mathcal{C}$, increase $i$ and update the noise vector candidate $\mathbf{z}_i$ to the next most likely vector.

**d.** Return $\hat{\mathbf{v}} = \mathbf{r} \oplus \mathbf{z}_i$.

## 3.2 Ordered Reliability Bits GRAND

The soft input version of GRAND is the so-called Ordered Reliability Bits GRAND, or ORBGRAND. The core idea of the algorithm is still the same, but now, with the soft input, the ranking process must be changed in order to account for the different reliabilities of each bit in the sequence. In order to rank these sequences, it is required the probability of them being correct. As stated previously, the output of the channel, for each bit of the sequence, is given in terms of log-likelihood ratios:

$$\ell_i = \ln \frac{p(r_i|v_i = 0)}{p(r_i|v_i = 1)}. \tag{3.2}$$

By using Bayes' rule, it is possible to invert the conditional probabilities:

$$\ell_i = \ln\left(\frac{P(v_i=0|r_i)p(r_i)}{P(v_i=0)}\frac{P(v_i=1)}{P(v_i=1|r_i)p(r_i)}\right) = \ln\frac{P(v_i=0|r_i)(1-P(v_i=0))}{(1-P(v_i=0|r_i))P(v_i=0)}. \tag{3.3}$$

The letter $P$ denotes a probability while the letter $p$ denotes a probability density function. The value $P(v_i)$ is known as a-priori probability of the bit $i$. The assumption used here is that the transmitted codewords are picked at random, so that the a-priori probability of being either 0 or 1 is 0.5.

By putting $P(v_i = 0|r_i)$ in evidence in (3.3) and assigning $P(v_i = 0) = 0.5$ it follows that

$$P(v_i = 0|r_i) = \frac{e^{\ell_i}}{1+e^{\ell_i}}. \tag{3.4}$$

and, reciprocally

$$P(v_i = 1|r_i) = \frac{e^{-\ell_i}}{1+e^{-\ell_i}}. \tag{3.5}$$

By considering the BPSK mapping of Table 2.1, it is possible to derive that the probability that the hard decoding of a bit is correct is the probability that the bit was originally 0 originally, for a positive $r_i$, or the bit was originally 1, for a negative $r_1$:

$$P(v_i \text{ right}) = \begin{cases} \frac{e^{\ell_i}}{1+e^{\ell_i}} & \text{if } r_i \geq 0 \\ \\ \frac{e^{-\ell_i}}{1+e^{-\ell_i}} & \text{if } r_i < 0. \end{cases} \tag{3.6}$$

From the LLR definition in (3.2), it is observed that $\ell_i$ is negative for negative $r_i$ and positive for positive $r_i$. Therefore, (3.6) can be rewritten as

$$P(v_i \text{ right}) = \frac{e^{|\ell_i|}}{1+e^{|\ell_i|}}. \tag{3.7}$$

Let us denote the probability of a wrong bit $(1 - P(v_i \text{ right}))$ as

$$B_i = \frac{e^{-|\ell_i|}}{1+e^{-|\ell_i|}}. \tag{3.8}$$

The probability that a whole sequence of $n$ bits results in a particular noise vector, is given by

$$P(\mathbf{Z} = \mathbf{z}|\mathbf{r}) = \prod_{i:z_i=0}(1-B_i)\prod_{i:z_i=1}B_i = \prod_{i=0}^{n}(1-B_i)\prod_{i:z_i=1}\frac{B_i}{1-B_i} \tag{3.9}$$

$$\propto \prod_{i:z_i=1}\frac{B_i}{1-B_i} = \prod_{i:z_i=1}e^{-|\ell_i|} = \exp\left(-\sum_{i=1}^{N}|\ell_i|z_i\right), \tag{3.10}$$

where the $\mathbf{Z}$ denotes the random variable of length $n$ and $\mathbf{z}$ corresponds to the realization of that random variable, the noise vector obtained from the hard decoder.

From this, it is possible to define a quantity $Q(\mathbf{z})$ that encodes how unreliable the decoded sequence $\mathbf{y}$ is, which is conceptually the opposite of $P(\mathbf{Z} = \mathbf{z})$:

$$Q(\mathbf{z}) = \sum_{i=1}^{n} |\ell_i| z_i. \tag{3.11}$$

The smaller the value of $Q(\mathbf{z})$, the more reliable the decoded sequence $\mathbf{y}$ and the decoded noise vector $\mathbf{z}$ are. Note how this value is proportional to the Hamming weight if the hard decoder GRAND is considered

$$|\ell_i| = C \quad \forall i \in (1, 2, \cdots, n) \implies Q(\mathbf{z}) \propto w_H(\mathbf{z}), \tag{3.12}$$

which is exactly how the hard input GRAND works: noise sequences are evaluated and ranked according to their Hamming weights.

Now that there is a metric of how reliable (or unreliable) any block sequence is, for the given LLRs received from the channel, it is possible to order these noise sequences by their reliability. In order to do that, first the LLRs of each individual bits are ordered in increasing order and encoded in a variable that stores the original bit positions in the sequence $\pi = (\pi_1, \pi_2, \cdots, \pi_n)$. For instance, the variable $\pi_i$ records the original position of the $i$th bit in the ordered sequence of LLRs. For the BPSK case, the ordered LLRs can be simplified to $|y_i|$, as $|\ell_i| \propto |y_i|$. It is observed [4] that, for low SNR values, these approximated LLR values of the individual bits in a received block can be satisfactorily approximated by a straight line that starts at the origin:

$$|\ell_{i,\text{ord}}| \approx \beta i. \tag{3.13}$$

By using this approximation in (3.11), it follows that

$$Q(\mathbf{z}) \approx \beta \sum_{i=1}^{n} i z_i. \tag{3.14}$$

As constants of proportionality have no impact on the reliability order, it is possible to define the unreliability of a bit sequence $\mathbf{z}$ in terms of the logistic weight of vector $\mathbf{z}$:

$$w_L(\mathbf{z}) = \sum_{i=1}^{n} i z_i. \tag{3.15}$$

From that metric, the ORBGRAND algorithm tests out the validity of each noise vector $\mathbf{z}$ in increasing order of logistic weight. It starts with $w_L(\mathbf{z}) = 0$, the hard decision decoder on the received vector $\mathbf{r}$, than tests out $w_L(\mathbf{z}) = 1$ by flipping the least reliable bit (the first one, after the permutation), than tries $w_L(\mathbf{z}) = 2$ by flipping the second bit of the permuted sequence. At $w_L(\mathbf{z}) = 3$, the decoder can either flip the third bit or both first and second bits, as they have the same logistic weight. The process goes on accordingly.

The key point of this step is to find the noise sequences that correspond to a specific logistic weight. An algorithm called **landslide** is used by the original ORBGRAND authors to provide the set of all noise sequences for a specific Hamming and logistic weights. When all these sequences have been tested, the Hamming

weight is increased by one until it is maximum, for the specific logistic weight that the algorithm is running. In case the logistic weight is larger than $n$, the minimum Hamming weight cannot be one. The maximum logistic weight $W = w_L(\mathbf{z})$ for a given Hamming weight $w = w_H(\mathbf{z})$ is obtained by flipping the last bits of the codeword,

$$w_L(\mathbf{z}) \leq W = \sum_{i=n-w+1}^{n} i = \frac{w(2n-w+1)}{2}. \tag{3.16}$$

The second order equation above can be put in terms of $w$. Considering that $w < n$ and that the Hamming weight is an integer, it follows that

$$w_{min} = \left\lceil \frac{2n+1-\sqrt{(2n+1)^2-8W}}{2} \right\rceil. \tag{3.17}$$

In the same manner, the smallest logistic weight considering a specific Hamming weight is obtained by flipping the first bits of the error vector:

$$w_L(\mathbf{z}) \geq W = \sum_{i=1}^{w} i = \frac{w(w+1)}{2}. \tag{3.18}$$

Finally, by putting $w$ in evidence and choosing the solution that has $w > 0$, it follows that

$$w_{max} = \left\lfloor \frac{-1+\sqrt{1+8W}}{2} \right\rfloor, \tag{3.19}$$

where the floor function is used to provide an integer value that produces a logistic weight less or equal to the value required $W$. For example, if the algorithm is looking for noise vectors with logistic weight equal to 7, then (3.19) without the floor function gives a maximum Hamming weight of about 3.27. If one would choose to use 4, for example, one could get to a logistic weight that is larger than 7, so that demonstrates why the floor is important.

The focus of this work is on the basic ORBGRAND, a version of ORBGRAND that approximates the ordered reliabilities by a straight line passing through the origin. Figure 3.2 has been taken from [4] and shows how the approximation is compared to the actual reliabilities for different values of SNR.

More information on the general ORBGRAND is given in Section 5. The basic ORBGRAND algorithm implemented is described in details in page 27.

The details of the landslide algorithm can be found in [4] and are out of the scope of this work. It is important to emphasize that the algorithm just presented is an approximation, due to the linear assumption taken, that results in the logistic weight metric. For larger SNR values this approximation is not valid anymore and the linearization does not produce ML decoding performance. The authors propose that, for that cases, the reliability function can be divided into several segments, each of which can be approximated linearly. The idea is that such segments can be computed in parallel, making it feasible in terms of computational complexity. This approach is also out of the scope of this work but is described briefly in Section 5.

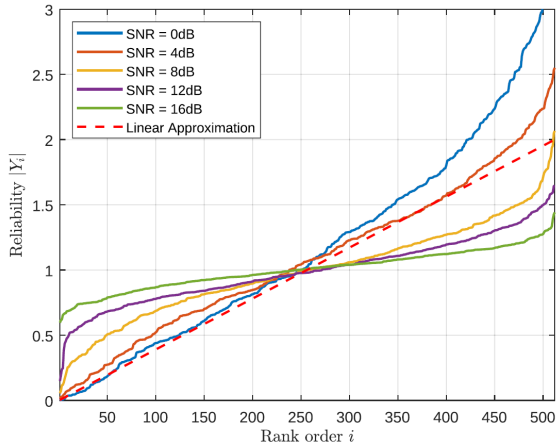


**Figure 3.2:** Ordered reliabilities and the basic ORBGRAND approximation. Figure taken from [4].

To illustrate the BER performance of ORBGRAND when compared to the **soft input** trellis decoder of Section 2.7, a (32,26) Hamming code was simulated. The complexity analysis is carried out in Section 4, where the ORBGRAND can be approximated as a special case of the SO-GRAND, with list size $L = 1$. The results of the simulation are depicted in Figure 3.3.

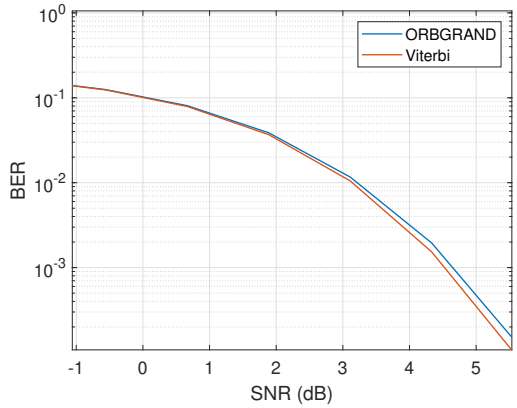


**Figure 3.3:** BER performance of ORBGRAND and trellis-based decoder with the (32,26) Hamming code

Due to the basic ORBGRAND approximation, it is seen that the ORBGRAND does not produce ML results, while the Viterbi do.

**Basic ORBGRAND:**

Inputs: $k$, $\mathbf{H}$, $\ell_i$ with $i \in (1, 2, \cdots, n)$ and $k$ is the maximum number of queries to find a codeword

**a.** Demodulate the soft information received $\mathbf{y}$ and test the codeword validity with $\mathbf{H}$. If $\mathbf{y}\mathbf{H}^T = \mathbf{0}$, **return** $\mathbf{y}$.

**b.** Sort all $\ell_i$ in ascending order and record the original positions in $\pi_i$.

**c.** Create a new parity-check matrix $\bar{\mathbf{H}}$ with the columns of matrix $\mathbf{H}$ sorted according to $\pi_i$:

$$\bar{\mathbf{H}} = [\bar{\mathbf{h}}_1 \quad \bar{\mathbf{h}}_2 \quad \cdots \quad \bar{\mathbf{h}}_n] = [\mathbf{h}_{\pi_1} \quad \mathbf{h}_{\pi_2} \quad \cdots \quad \mathbf{h}_{\pi_n}].$$

**d.** Initialize: $i = 1$, $W = 0$ (logistic weight)

**while** $i < k$

$W = W + 1$

$w = w_{min}$, from (3.17)

**while** $w \leq w_{max}$, from (3.19)

$\mathcal{Z}$ = landslide$(W, w, n)$, where the set $\mathcal{Z}$ contains all error vectors $\bar{\mathbf{z}}$ with $w_L(\mathbf{z}) = W$ and $w_H(\mathbf{z}) = w$

**for** all $\bar{\mathbf{z}} \in \mathcal{Z}$

$i = i + 1$

if $\bar{\mathbf{z}}\bar{\mathbf{H}} = \mathbf{0}$

$\mathbf{z} = [\bar{z}_{j:\pi_j=1} \quad \bar{z}_{j:\pi_j=2} \quad \cdots \quad \bar{z}_{j:\pi_j=n}]$, the process of inverting the permutation

$\hat{\mathbf{v}} = \mathbf{y} \oplus \mathbf{z}$

**return** $\hat{\mathbf{v}}$

## 3.3 Soft Output GRAND

As discussed previously, the soft output decoder delivers soft information to be used in an iterative scheme. Figure 3.4 presents the inputs and outputs of a generic soft output block in a concatenated scheme. In this work, SO-GRAND and trellis-based soft output decoder are used in the place of the soft-input soft-output (SISO) decoder block.

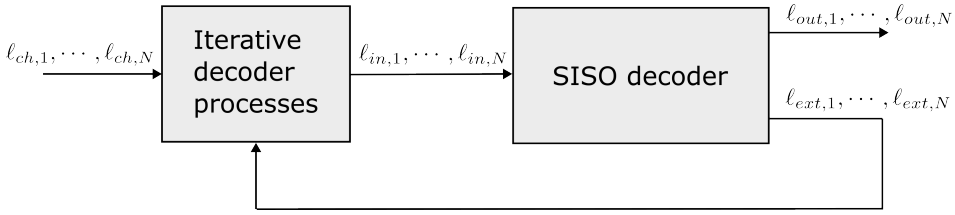


**Figure 3.4:** SISO decoder block connections

In general, soft output decoders utilize the channel LLR with the a-priori LLR in the first iteration ($\ell_{in,i} = \ell_{ch,i} + \ell_{A,i}$). The a-priori LLR determines what is the probability of a bit in a certain position of the sequence be one or zero, regardless of what was received by the channel. In the simulations performed in this work, the first a-priori LLRs were considered as zero, due to the equal distribution of symbols assumption. After executing the SISO decoder, the extrinsic LLRs are combined again with the channel output by a processing stage that depends on the type of iterative decoder used. The extrinsic LLRs $\ell_{ext}$ are generated with basis on the other bits on the sequence. For example, for the calculation of $\ell_{ext,i}$, all bits with an index different than $i$ are considered. Finally, the a-posteriori LLR $\ell_{APP}$ are used by the demodulator to convert the soft output to binary digits.

Ideally, the $\ell_{APP}$ is represented in terms of sequence probabilities. The numerator is related to the codewords with the $i$th bit equal to 0, and the denominator relates to the codewords with the $i$th bit equal to 1,

$$\ell_{APP} = \ln \frac{\sum_{\mathbf{c}\in\mathcal{C}:c_i=0} P(\mathbf{c}|\mathbf{r})}{\sum_{\mathbf{c}\in\mathcal{C}:c_i=1} P(\mathbf{c}|\mathbf{r})}. \tag{3.20}$$

It is clear that such extensive method is not feasible for large blocks. The SO-GRAND strategy is to use list decoding, meaning that only some of the codewords are considered accurately in the calculation (ideally, the most probable ones). The probabilities associated with codewords out of the list are only approximated. For a given list of most probable codewords $\mathcal{L}$, the equation below shows the idea:

$$\ell_{APP} = \ln \frac{\sum_{\mathbf{c}\in\mathcal{L}:c_i=0} P(\mathbf{c}|\mathbf{r}) + P(\mathbf{c}\in\mathcal{C}\setminus\mathcal{L}|\mathbf{r})P(c_i=0|r_i)}{\sum_{\mathbf{c}\in\mathcal{L}:c_i=1} P(\mathbf{c}|\mathbf{r}) + P(\mathbf{c}\in\mathcal{C}\setminus\mathcal{L}|\mathbf{r})P(c_i=1|r_i)}. \tag{3.21}$$

In order to execute this calculation, the decoder requires the list $\mathcal{L}$, the probability of the sequence not being a part of $\mathcal{L}$ for a given received vector $P(\mathbf{c}\in\mathcal{C}\setminus\mathcal{L}|\mathbf{r})$ and, finally, the probability of a particular output on bit $i$, for a given received demodulated bit, $P(c_i|r_i)$.

The list of most probable sequences $\mathcal{L}$ can be obtained by the ORBGRAND algorithm, as described in details in Section 3.2. The difference here is that, instead of looking for only the most probable codeword, the decoder requires the $L$ most probable codewords as an output. By simply letting the algorithm continue (instead of returning at the first valid codeword), it is possible to stop the code when all list of size $L$ is filled with codewords.

The probability that the list decoding does not contain the correct codeword is approximated in [6] by

$$P(\mathbf{c} \in \mathcal{C} \setminus \mathcal{L}|\mathbf{r}) \approx \frac{\left(1 - \sum_{j=1}^{q_L} P(\mathbf{z}_j|\mathbf{r})\right) \frac{2^k-1}{2^n-1}}{\sum_{i=1}^{L} P(\mathbf{z}_{q_i}|\mathbf{r}) + \left(1 - \sum_{j=1}^{q_L} P(\mathbf{z}_j|\mathbf{r}_j)\right) \frac{2^k-1}{2^n-1}}. \tag{3.22}$$

The $q_i$ is the query order of the $i$th noise vector that produces a valid codeword. For example, if $L = 4$ the output list from the ORBGRAND has 4 valid noise vectors and each of these sequences will be obtained after testing all noise vectors generated by the landslide algorithm before. If $q_5 = 15$, it means that the 5th valid noise vector was obtained after 14 other noise sequences generated by the landslide algorithm.

The probability of a noise vector $P(\mathbf{z}|\mathbf{r})$ can be obtained by (3.9), displayed here again in short:

$$P(\mathbf{z}|\mathbf{r}) = \prod_{i:z_i=0} (1 - B_i) \prod_{i:z_i=1} B_i. \tag{3.23}$$

The summation term over the probabilities of noise vectors in (3.22) can be obtained by simply accumulating the probabilities given in (3.23), for all noise vectors generated by the landslide algorithm. The first term in the denominator of (3.22) is obtained by summing up the probabilities of the valid noise vectors, as given by the ORBGRAND algorithm.

Finally, the $P(c_i|r_i)$ in (3.21) is given by (3.4) and (3.5), using the channel LLR.

From the block in Figure 3.4, there is also the extrinsic LLR $\ell_E$ to be defined. The point of this extrinsic values is to get rid of the dependence on the analyzed bit, so that it can be used on the next iteration, as the a-priori LLR $\ell_A$. The extrinsic LLR is given by the equation below:

$$\ell_{ext,j} = \ell_{APP,j} - \ell_{A,j} - \ell_{Ch,j}, \quad i = 1, \cdots, n. \tag{3.24}$$

The a-priori probability starts as 0.5 for all bits, as the assumption is that the codewords are picked at random. As a result, the first set of $n$ a-priori LLRs is an all-zero vector.

Chapter 4

# GRAND and Trellis-Based Approach Comparison

The ultimate goal of this work is to compare the performance and trade-offs between GRAND and the trellis approach. Performance is evaluated in two aspects: bit error rate (or block error rate) over SNR, and computational complexity.

One of the key advantages of GRAND, as previously mentioned, is its potential as a "universal" decoder due to its code-agnostic nature. To keep this same argument, GRAND is compared with the trellis approach, which, while traditionally associated with convolutional codes, has been used here for block codes. Both GRAND and the trellis-based decoder for block codes only require knowledge of the parity-check matrix, supporting the universal applicability of these decoders. To demonstrate this, both decoders are tested with CRC, Hamming, extended Hamming, and random block codes.

As discussed earlier, generalized LDPC codes can incorporate generic codes into the outer parity-check matrix, creating a flexible decoder that optimizes the balance between BER performance and computational complexity. With that in mind, both trellis and GRAND approaches are evaluated within concatenated schemes.

The complexity analysis begins with a logical inspection of the algorithms to deduce complexity order and computational demands. When exact analytical formulas are not feasible, statistical estimates are provided. Finally, simulations are conducted to validate the analysis.

The final computational complexity of a concatenated scheme is a function of its BER performance and the complexity associated with each iteration of the inner code. It is then interesting to separate the analysis into two parts. In the first level of analysis, the individual soft output blocks are studied for both decoders. Later on, their performance is evaluated in the context of a complete concatenated scheme.

## 4.1 Soft Output Block Performance

Each SO block is analyzed in terms of its inherent complexity and output BER. The resulting BER is evaluated with histograms that show how the original bits transmitted, which are all-zero, distribute after the decoding procedure. These

histograms can help provide insight on how the decoding is acting on the soft information received and what is the quality of the soft output delivered to the next iteration.

### 4.1.1 SO trellis Decoder

A simple inspection of the algorithm presented in Section 2.7.4 shows that the number of computations is dominated by the concatenated for loops in the item b of the Johansson-Zigangirov algorithm. One can estimate the amount of computations $C$ required for calculating the flow $\mu$ by

$$C = N \cdot 2^m - 2^m = 2^m(N-1), \tag{4.1}$$

where $2^m$ is the number of states in the trellis and $N$ is the block length. The $(N-1)$ factor results from the fact that the outer for loop starts at $l = 2$, as the first two flow variables $\mu(\mathbf{0}, 1)$ and $\mu(\mathbf{h}_1, 1)$ were initiated outside the loop. One could count that too but what matters in the end of the analysis is the proportionality to $N$ and $m$ and not the absolute value of computations.

By using the expurgated trellis approach demonstrated in Section 2.7.2, one can reduce the amount of possible states in the trellis, thus reducing the final complexity of the algorithm. As the Johansson-Zigangirov algorithm utilizes all state variables at the end of the trellis, it is only possible to take out the invalid states at the beginning of the trellis algorithm. The amount of states reduced depends on the parity-check matrix $\mathbf{H}$ and cannot be exactly calculated. Nevertheless, it is possible to find an upper bound for the reduced number of states traversed by the algorithm. This upper bound is obtained by considering that, at each next bit analyzed in the block, twice as many states are visited when compared to the previous bit. Normally, for some transitions from one bit to the next, there would be repeated states visited, so this assumption is the worst case scenario.

This reduced trellis approach could potentially reduce the number of times the function enters the concatenated for loop by $(m-3)2^m + 4$. Figure 4.1 shows an example with $N = 7$ and $m = 3$.

The formulas give the total number of states visited in the for loop as:

$$\begin{aligned} C &= 2^m(N-1) = 48 \text{ states} \\ C_{red} &= 2^m(N-1) - (2^m(m-3)+4) = 2^m(N-m+2) - 4 = 44 \text{ states.} \end{aligned} \tag{4.2}$$

From the equations above it is observed that the complexity is $\mathcal{O}(2^m)$ and $\mathcal{O}(N)$, regardless if the reduced trellis approach is used or not. Figures 4.2 and 4.3 show how the estimations of computation vary with the input parameters $m$ and $N$.

Finally, the Figures 4.4 and 4.5 show the empirical proportionality of complexity to block length $N$ and redundancy ). The estimated computational complexity is here taken as CPU time, which is not a very reliable metric but, statistically, can provide some useful insight for the analysis.

From the results, it is possible to conclude that the analysis correctly associates the computational demand to the variables $m$ and $N$, i.e. the relation is indeed exponential and linear, respectively.

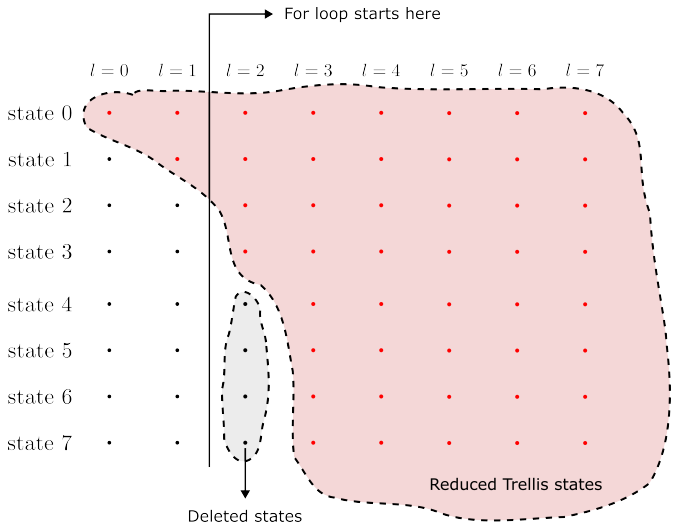


**Figure 4.1:** Reduced trellis for N=7 and m=3

A very important aspect of the trellis' complexity is that is does not depend on the SNR, meaning that, even in situations with low BER, the algorithm provides the same complexity.

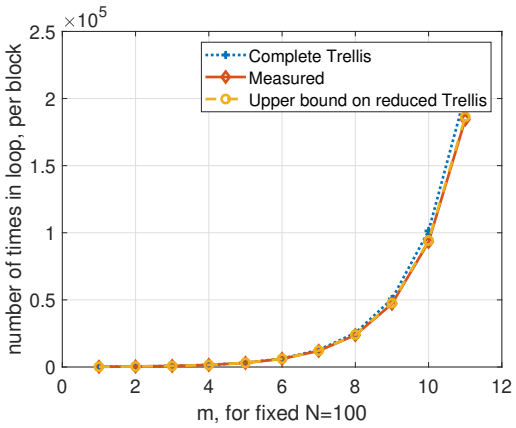


**Figure 4.2:** Predicted and empirical computations with increasing redundancy

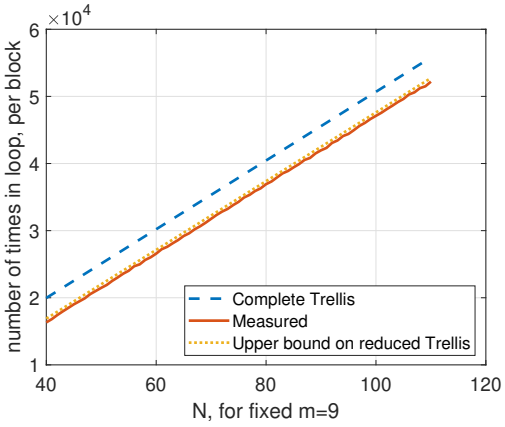


**Figure 4.3:** Predicted and empirical computations with increasing block length

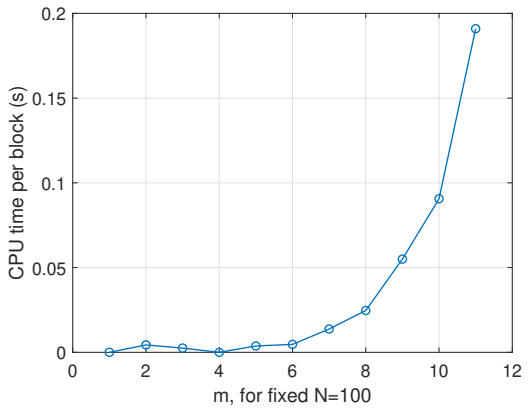


**Figure 4.4:** CPU time versus redundancy $m$

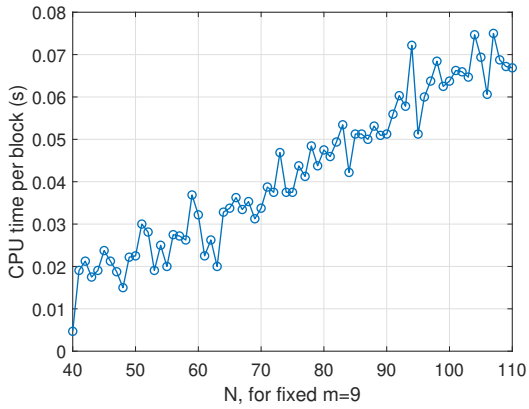


**Figure 4.5:** CPU time versus block length $N$

### 4.1.2 SO-GRAND

The SO-GRAND algorithm is arguably more complicated to be analyzed than the trellis as there are several mechanisms that run in a stochastic manner. The inspection of simulation time, with the help of a code profiler, pointed out that the complexity is dominated mainly by the landslide algorithm. The complexity of the overall algorithm is then a function of the landslide algorithm complexity times the number of times this function is called,

$$C_{GR} \propto n_{land} \cdot C_{land}. \tag{4.3}$$

Although it would be possible to carry the analysis with these two variables, they can be concatenated to a single one: the number of guesses. In order to see that more clearly, it is observed that the larger the number of guesses the longer the algorithm spends on a single execution of the landslide, due to the fact that the Hamming and logistic weights are increased. In addition, the larger the number of guesses, the less times the landslide function is called per output noise vector. This is shown empirically in Figure 4.6.

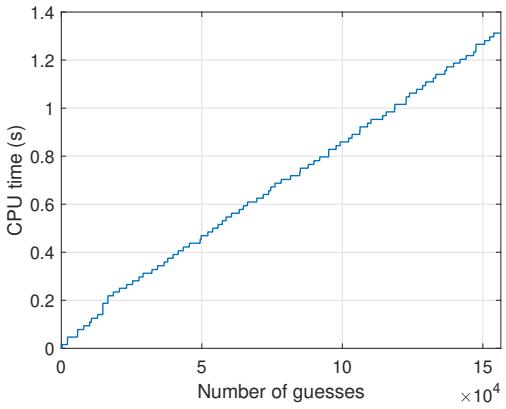


**Figure 4.6:** CPU time versus number of guesses

Given that, it is convenient to state that

$$C_{GR} \propto n_{guesses} \tag{4.4}$$

and move on with the analysis on the number of guesses.

The first step is to map out the influences of each entry of the SO-GRAND function. The essential inputs to the algorithm are the parity-check matrix $\mathbf{H}$, to check the validity of the noise vectors; the LLRs of each bit $i$ in the received vector $\ell_i$ and the list size $L$, that determines the amount of codewords used in the list decoding. Other inputs may be used to stop the algorithm like maximum number of guesses and minimum list BLER to return. This last input refers to the probability that the correct codeword is part of the list, calculated according to (3.22). These latter inputs mentioned do not affect our analysis as their impact can be obtained from the other inputs. For example, the minimum list BLER

affects the overall complexity because it changes the value of $L$ which, in turn, affects the total number of guesses.

From that inputs it is possible to conclude that the variables are:

- List size $L$;
- Block length $N$, encoded in the LLRs and $\mathbf{H}$;
- Redundancy $m$, encoded in the LLRs and $\mathbf{H}$;
- SNR, encoded in the values of the LLRs.

The rest of this section analyzes each of these variables considering random block codes as the encoding procedure.

### Signal-to-Noise Ratio (SNR)

Naturally, the SNR affects how close (by Euclidean distance) the received vector is in relationship to the transmitted vector. By the process of flipping the least reliable bits first and ranking the noise vectors in terms of reliability, the GRAND algorithm optimally finds the ML noise vector. It is important to note that the basic ORBGRAND, that approximates the reliability curves by a straight line through the origin, does not give ML receiver performance. Nevertheless, this approximation still is expected to find a valid codeword faster when the SNR is higher. So, in conclusion, pure ORBGRAND algorithm should be faster for higher SNRs.

However, for list decoding SO-GRAND, that should be true only for the first candidate codeword found, as the rest of the $2^K - 1$ codewords are, in general, distributed randomly over the $N$-dimensional space that they are part of. This is shown in Figure 4.7, where the average number of guesses until the algorithm finds the respective candidate codeword is plotted against the SNR.

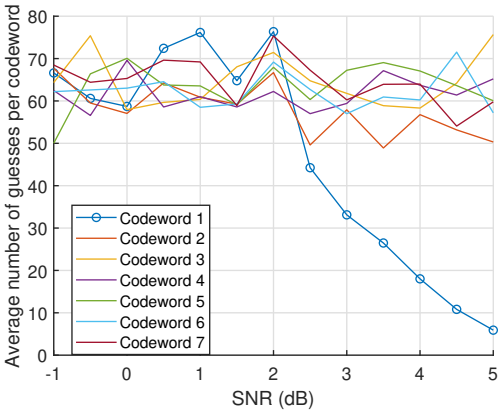


**Figure 4.7:** Average number of guesses until a candidate codeword is found versus SNR. $L = 7, N = 100, m = 6$

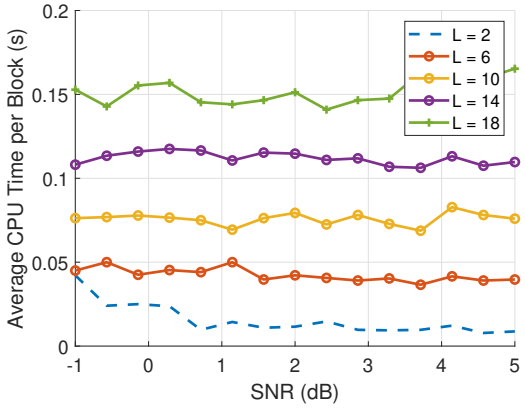


**Figure 4.8:** CPU time versus SNR, for several values of $L, N = 60$ and $m = 10$

As a result of that, the overall processing time does not depend much on the SNR, except for when the list size is very small. This can be seen at the empirical evidence of Figure 4.8.

### List size

A hint of how list size affects processing has already been shown in Figures 4.7 and 4.8. Essentially, by assuming random coding, the codewords are distributed almost evenly throughout the code space. Their probabilities are not dependent on one another. The final computational complexity then should be proportional to the list size and the number of guesses per codeword,

$$C_{GR} \propto n_{guess} = L \cdot n_{gpc}. \tag{4.5}$$

As the number of guesses per codeword $n_{gpc}$ is constant for each new codeword added to the list, the processing complexity should be linearly proportional to the list size. This is finally observed empirically, on Figure 4.9.

### Amount of redundancy bits $m$

In the same manner that the amount of redundancy bits influence the trellis-based algorithm, there are strong reasons to believe that there is a significant influence also in GRAND algorithm. The redundancy bits are a measure of the amount of codewords compared to all possible vectors in $\mathbb{F}_2^N$. As discussed previously, random codes place codewords randomly over the $\mathbb{F}_2^N$. For any received vector with low enough SNR, the probability of decoding directly a codeword is $2^K/2^N = 2^{-m}$. However, for each noise guess checked with the parity-check matrix, the probability of finding a codeword on the next guess is altered. For example:

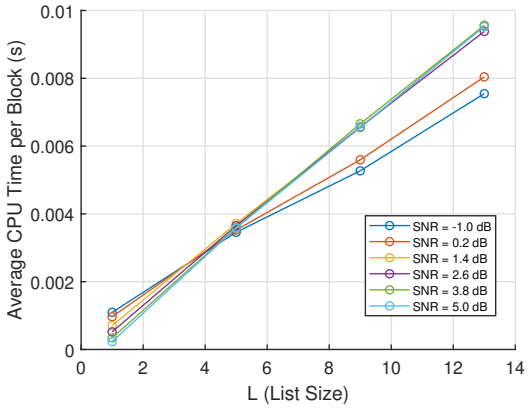


**Figure 4.9:** Average CPU processing time versus list size $L$, for different SNR levels, $m = 6$ and $N = 30$

$$P(1) = 2^K/2^N$$
$$P(2) = (1 - P(1))\frac{2^K - 1}{2^N - 1}$$
$$P(3) = (1 - (P(1) + P(2))\frac{2^K - 3}{2^N - 2}.$$

Thus, in general:

$$P(n) = \left(1 - \sum_{i=1}^{n-1} P(i)\right) \frac{2^K - (n-1)}{2^N - (n-1)}. \tag{4.6}$$

The assumption for the formulas above is that the SNR is very low or the algorithm is looking for a candidate codeword after the first one (the closest) has already been found. A closed formula in terms of $K$, $N$ and $m$ is hard to obtain. It is possible though to analyze it with a recurrent formula in the computer. The quantity of interest here is the average number of guesses per codeword. This quantity can be obtained by

$$\langle n_{guess} \rangle = \sum_{n=1}^{2^N - 2^K} n \cdot P(n). \tag{4.7}$$

The computations show that the average number of guesses until a candidate codeword is found does not depend on the block length $N$, but only on the redundancy bits $m$, in powers of two. Figure 4.10 shows how the average number of guesses depend on $m$, for $N = 100$, although the block length does not alter the result. From the plot it is also obtained that, for $m = 6$, the average number of guesses is $63.9578 \approx 64$, which is what has been observed empirically in Figure 4.7.

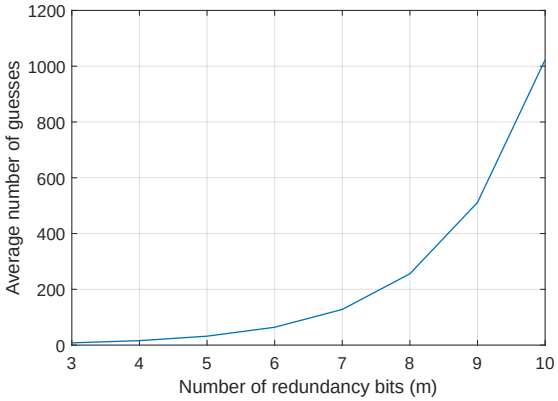


**Figure 4.10:** Calculated average number of guesses as a function of redundancy bits, for $N = 100$ (although the plot is invariant to $N$) and low SNR or candidate codeword number larger than one.

Naturally, these results leads one to postulate that

$$\langle n_{guess} \rangle = 2^m. \tag{4.8}$$

Probably a more in depth mathematical analysis on (4.7) and (4.6) would lead to that, but a proof that this holds for all $n$ is not presented here and it is out of the scope of this work. Finally, Figure 4.11 shows how the average CPU processing time varied with $m$, for different values of $N$, with the GRAND code built in MATLAB. Similarly, the plot from Figure 4.12 shows how the algorithm behaves with increasing $N$.

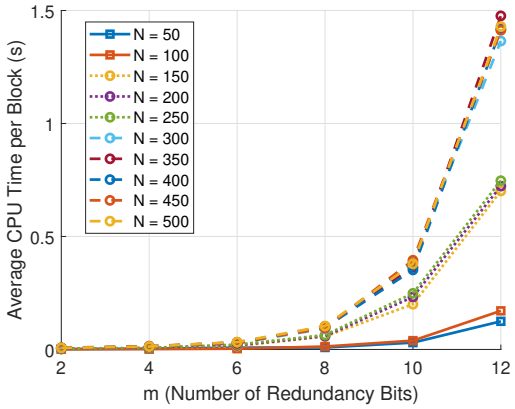


**Figure 4.11:** Empirical CPU time versus redundancy bits, for $SNR = 1.5$ dB, $L = 4$ and several values of $N$.

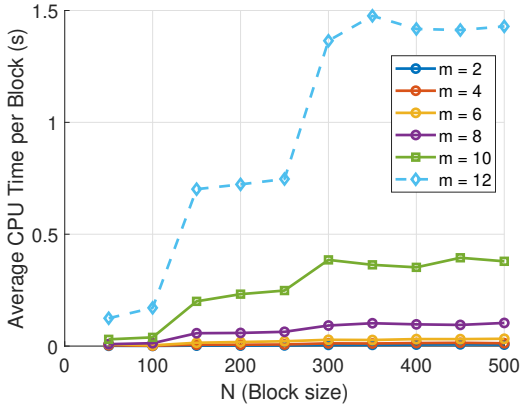


**Figure 4.12:** Empirical CPU time versus block length, for $SNR = 1.5dB$, $L = 4$ and several values of $m$.

The results show apparently that the value of N adds complexity to the algorithm. The reason is that, for larger and larger N, the lines of code outside the landslide algorithm start to become important too. Some of these computations are, for example, calculating probabilities for each noise vector generated, inverting the permutations done for reliability ordering and checking, with parity-check matrix, if the noise vector generated produces a codeword. All these operations grow at least linearly with the number of bits per codeword $N$. The conclusion is that, although the number of guesses is not affected by the block length, the number of computations per individual guess increases linearly with $N$. This can be seen in Figure 4.13, that shows the CPU time for guessing 300 noise vectors, not counting the time spent at the landslide function.

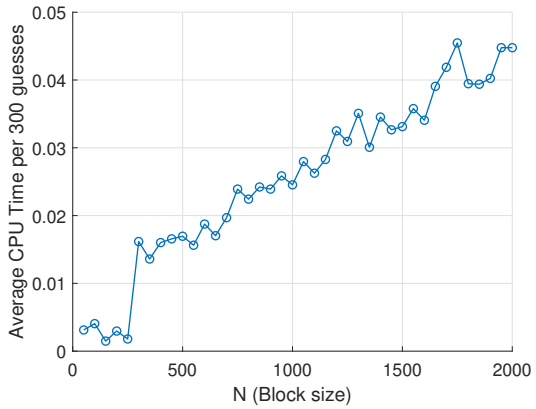


**Figure 4.13:** CPU processing time for computing 300 guesses (for $m = 5$), as a function of the block length

Summing up the complexity analysis for each run of the list decoder SO-GRAND, it is concluded that:

$$C_{GR} = C_{per\_guess} \cdot n_{guesses} + C_{per\_run} \approx kN \cdot L2^m. \tag{4.9}$$

Here, $k$ is a constant representing the number of computations per guess and per bit in a block of length $N$, while $C_{per\_run}$ denotes the number of computations performed during each algorithm run that are independent of individual guesses (e.g., sorting reliability, calculating overall list BLER, etc.). At best case, $C_{per\_run}$ is $\mathcal{O}(N \log N)$, due to the sorting algorithm. The approximation above considers that the first term dominates for typical values of $N$, $L$ and $m$. Additionally, the equation assumes the list $L$ is sufficiently large, making the dependency of the SNR on the first codeword in the list negligible.

## 4.2 Component Code Performance

The basic trade-off compared is complexity versus performance. The performance of each decoder is shown using bit error rate in a single iteration, relating to the previous complexity analysis. The encoded bits are all-zero vectors, allowing soft output LLRs to be presented in a histogram at different SNR levels. A CRC with generator polynomial `0xbae` has been implemented, with $N = 63$ and $K = 51$, as in [19]. The polynomial excludes the 0th order bit, so the least significant bit of value 1 is added. A list size of $L = 4$ and minimum list BLER of $10^{-4}$ have been set to reduce complexity at higher SNRs. Figure 4.15 presents the output histograms for the SO-GRAND algorithm, while 4.14 shows the trellis algorithm results.

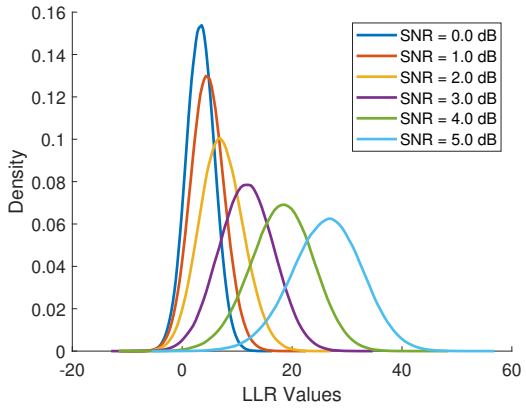


**Figure 4.14:** Histogram of output LLR from the trellis algorithm, for different levels of SNR

Both methods show an increasing variance over the higher SNR values but the SO-GRAND method seems to achieve a steady peak before, as between SNR values of 4 and 5 dB the distribution only shifts the mean to higher LLR values.

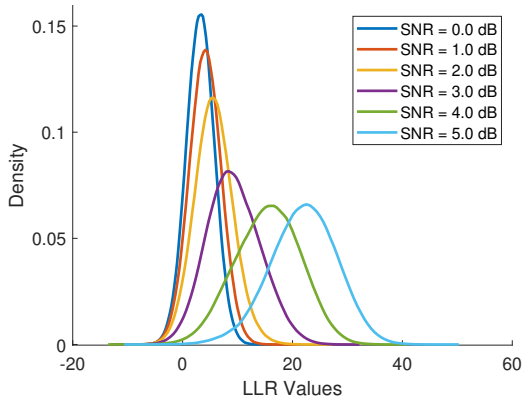


**Figure 4.15:** Histogram of output LLR from the SO-GRAND algorithm, for different levels of SNR

By taking the soft output LLRs and hard decoding with threshold at zero, the overall BER for both methods is obtained. The results are shown at Figure 4.16.

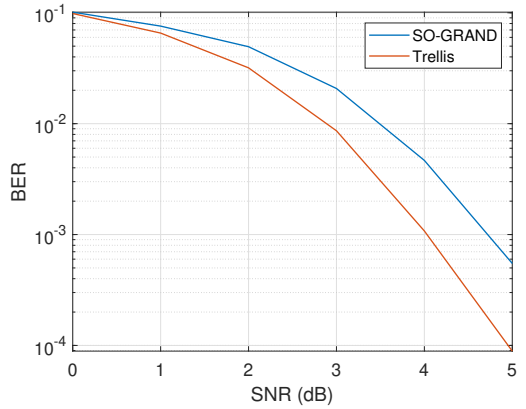


**Figure 4.16:** Bit error rate comparison for the hard decoded soft output from trellis and SO-GRAND algorithms

As the plot shows, the SO-GRAND has a significant performance gap in terms of BER when compared to the trellis, especially for higher SNRs. This happens due to the approximation of reliability values after ordering, which has been considered as a straight line through the origin. This simplification is embedded in the logistic and Hamming weights approach and it basically may not give the correct order of block reliability. As discussed previously, other approximations were already studied and presented in [4]. These other versions of the ORBGRAND algorithm would provide better results and would possibly be closer to the trellis algorithm, especially in higher SNR values. Although the other versions of ORBGRAND

were not implemented, some insights are given in Section 5.

At last, the performance of both decoders is tested with the GLDPC. The component code utilizes a Hamming (32,26) code that generates a quasi-cyclic GLDPC [16] like below

$$\mathbf{H} = \begin{bmatrix} I_n^0 & I_n^0 & \cdots & I_n^0 \\ I_n^0 & I_n^1 & \cdots & I_n^{N_{inn}-1} \end{bmatrix},$$

where the subscript "inn" indicates the inner code (component code) parameter. In this case $N_{inn} = 36$, $K_{inn} = 26$ and $m_{inn} = 6$. The overall code rate of the GLDPC is given by $R = 1 - 2 \cdot 6/36 = 2/3$. For the code implemented in this work, the generic decoder mentioned in the algorithm of Section 2.9 is substituted by the SO-GRAND and the trellis decoders. The results are displayed at Figure 4.17 and 4.18, in terms of BER and BLER.

As seen from the figures, there is a big gap in performance between both methods. This is again explained by the rough estimation of block reliability based on a straight line through the origin, called basic ORBGRAND in [4].

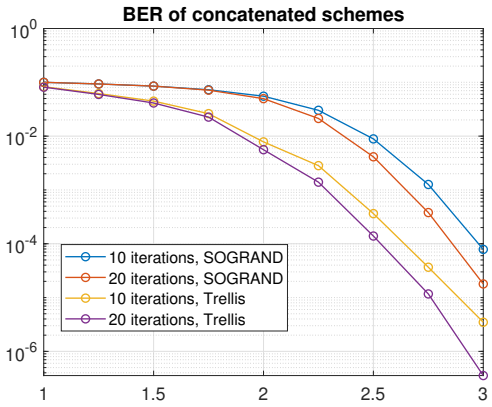


**Figure 4.17:** BER performance of SO-GRAND and trellis in the concatenated scheme of GLDPC

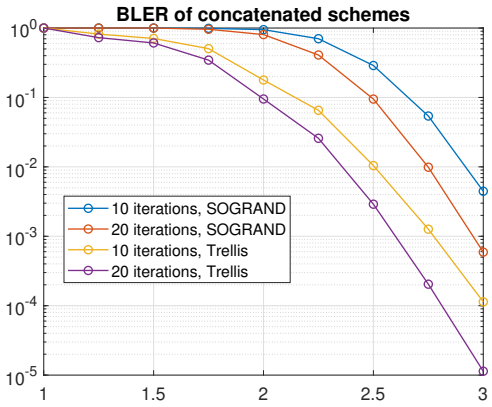


**Figure 4.18:** BLER performance of SO-GRAND and trellis in the concatenated scheme of GLDPC

# Chapter 5

# Generic ORBGRAND principles

The implementation of the more complex generic ORBGRAND is out of the scope of this work, and all the complexity analysis has been carried with the best case scenario for ORBGRAND: the basic ORBGRAND. The basic ORBGRAND approximation can be seen in Figure 3.2.

However, it is already known and stated by the original authors that such approximation leads to a performance degradation, specially for higher SNR. In order to account for that, the authors developed an ORBGRAND that approximates the ordered reliabilities with more than one line. Figure 5.1 was extracted from [4] and shows how the generalized ORBGRAND approximates the reliability by a series of straight lines.

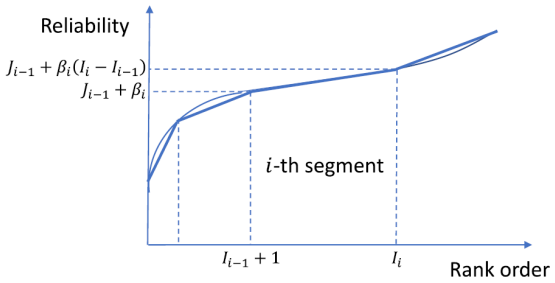


**Figure 5.1:** Generalization of ORBGRAND by fitting the ordered reliabilities with several segments. Figure taken from [4]

The slope of each segment of the approximation is given by

$$\lambda_j = J_{i-1} + \beta(j - I_{i-1}), \text{ for } I_{i-1} < j \leq I_i \text{ and } 1 \leq i \leq M. \tag{5.1}$$

The reliability from (3.11) is then approximated by the equation below:

$$Q(\mathbf{z}) \approx \sum_{i=1}^{M} \sum_{j=I_{i-1}+1}^{I_i} \lambda_j z_j = \sum_{i=1}^{M} W_i, \tag{5.2}$$

where $W_i$ is the generalized weight, given by

$$W_i = \sum_{j=I_{i-1}+1}^{I_i} [J_{i-1} + (j - I_{i-1})\beta_i] z_j \tag{5.3}$$

$$= \sum_{j=I_{i-1}+1}^{I_i} J_{i-1} z_j + \sum_{j=I_{i-1}+1}^{I_i} (j - I_{i-1}) z_j \beta_i = J_{i-1} w_{H,i} + \beta_i w_{L,i}, \tag{5.4}$$

where $w_{H,i}$ and $w_{L,i}$ are the Hamming and logistic weights of the i$th$ segment. Note that the logistic weight is defined relative to the first bit of the segment $i$. The **high level** idea of the algorithm is described as follows:

- First test out the hard decoder and check if it is a codeword, adding to the list in case it is;
- Approximate the ordered reliability by $M$ segments;
- Start with $W = 1$;
- Find the combinations of $W_i$ that gives the overall $W$. This is an integer partition problem;
- For each $W_i$, check which $w_{H,i}$ are valid, obeying the maximum and minimum logistic weights and if $W_i - w_{H,i} J_{i-1}$ is divisible by $\beta_i$, as shown in (5.4);
- For each valid combination of $W_i$ and $w_{H,i}$, take the logistic weight $w_{L,i}$ and run the landslide algorithm, finding all possible noise vectors with that inputs for the respective segment;
- Take all resulting partial noise vectors and perform a cartesian product for all segments in the block;
- Test all resulting noise vectors with the parity-check matrix and add the codewords to the list;
- Increase the overall general weight $W$ and repeat the procedures until $L$ codewords are found.

Although the algorithm above improves the performance in terms of BER, it certainly aggregates more complexity per each guess, as there are several new mechanisms involved. The question is whether a significant part of these computations are not relevant because are calculated prior to the sections where the landslide algorithm is called, meaning that they would only account for a constant amount of computations for a particular call of the SO-GRAND function. An important point brought by the original authors is that, once the partial generalized weights $W_i$ are defined, the algorithm can operate each segment in parallel, decreasing the overall processing time. Further studies with simulations and detailed analysis of the generalized ORBGRAND should be conducted to provide more insights on that.

# Chapter 6
# Conclusion

The GRAND algorithm, in its various forms, represents a new and creative way of decoding block codes. This study attempted to analyze the complexity and BER/BLER performance of GRAND applied to concatenated schemes. The soft output version of the algorithm (SO-GRAND) is suitable for such application, being that the main theme of this work.

Based on the universality feature of SO-GRAND, the trellis-based block decoder was chosen as a basis for comparison and showed itself as memory efficient, requiring only that $m_{inn}$ flow values are saved at each iteration of the concatenated for loops. The state variables can easily be converted to integers and the XOR operations are efficiently computed with a lower level coding. The algorithm is elegant and relatively simple to implement.

With regards to SO-GRAND, the inspection of the algorithm and the simulations show that, for a list size larger than 1, SO-GRAND complexity has the same proportionality to the block length $N$ and redundant bits $m$ as the trellis-based algorithms. However, the overall number of computations of SO-GRAND does not seem to match the trellis. This means that, for fixed $N$ and $m$ and $L > 1$, there are more computations in SO-GRAND than the trellis. This happens because, for a specific $N$ and $m$, the trellis is dominated by the one line of code in item b of the algorithm in Section 2.7.4 (that contains computations like XOR and square roots) while SO-GRAND relies on list size $L$ and the amount of computations per guess, which involves finding integer partitions in the landslide function and also calculating (and summing) probabilities for each noise vector generated. On the other hand, GRAND algorithms are more flexible for trading off complexity and BER performance, as the maximum number of guesses and minimum list BLER are possible inputs to the decoder. These strategies come with the price of having to test the conditions for all noise sequences generated, adding to the complexity of each guess. However, for higher SNR values, it pays off, decreasing significantly the processing time and saving power.

A key observation from the results is that one of GRAND's main advantages, related to its ability to efficiently search for the closest codewords to a given received vector, diminishes when a list decoding approach is employed. That is well observed in Figure 4.7. A drawback of the simulated SO-GRAND is that its BER/BLER performance, even without complexity reduction, shows a significant gap compared to the trellis approach. This leaves little room for trade-offs in the number of computations.

The scope of this work was limted to the use of the basic version of ORBGRAND, in the concatenated scheme. It is expected that the generic form of ORBGRAND, as it can approach ML performance, provides a much better BER/BLER performance. The Section 5 was dedicated to motivate why the generic version seems to increase the processing demand, although it is required a more detailed study on how the number of partitions of the received vector affects the feasibility of the algorithm, including the possibility of parallel computing.

Apart from investigating and comparing other universal decoders performance with GRAND, there are other approaches on the GRAND itself that could be tried. One of the bottlenecks of the algorithm is the landslide function. An in depth research on other algorithms that could perform the same task is important. It is interesting to note how the noise sequence of bits could be summarized by a function that receives a noise vector and the SNR, giving as output the closest noise sequence that was not tested yet. The SNR is required because, according to Figure 3.2, the reliability sequence looks different for different SNR values. Maybe an alternative with AI would be worth trying, in case it is possible to converge.

The newly developed GRAND algorithms offer a promising area for further research, with their strengths and weaknesses yet to be fully explored. This study aims to put light on several aspects of these decoders by comparing them with established methods. One of the key challenges confronted in this work was the intricacy and complication of GRAND algorithms, which should not be underestimated when considering practical implementation. The generic ORBGRAND, in particular, would certainly bring even more challenges. By detailing the implementation process, this work also hopes to assist future researchers interested in studying and advancing these algorithms.

# References


[1] C. E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[2] K. R. Duffy, J. Li, and M. Médard, "Guessing noise, not code-words", in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 671–675, 2018.

[3] K. R. Duffy, A. Solomon, K. M. Konwar, and M. Médard, "5G NR CA-Polar Maximum Likelihood Decoding by GRAND", in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–5, 2020.

[4] K. Duffy, W. An, and M. Médard, "Ordered Reliability Bits Guessing Random Additive Noise Decoding", Feb. 2022.

[5] 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 *2023 IEEE International Symposium on Information Theory (ISIT)*, pp. 1090–1095, 2023.

[6] P. Yuan, M. Médard, K. Galligan, and K. R. Duffy, "Soft-output (SO) GRAND and Iterative Decoding to Outperform LDPCs", 2024.

[7] R. Gallager, "Low-density parity-check codes", in *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[8] A. Straßhofer, D. Lentner, G. Liva and A. G. i. Amat, "Soft-Information Post-Processing for Chase-Pyndiah Decoding Based on Generalized Mutual Information," *2023 12th International Symposium on Topics in Coding (ISTC)*, Brest, France, 2023, pp. 1-5

[9] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, Sep. 1995.

[10] Y. Wu and C.N. Hadjicostis, "Soft-Decision Decoding Using Ordered Recodings on the Most Reliable Basis," *IEEE Trans. Inf. Theory*, vol. 53, no. 2, pp. 829-836, Feb. 2007.

[11] M. Loncar, R. Johannesson, I. Bocharova and B. Kudryashov, "A Comparison of Some Trellis- and Tree-Based SISO Algorithms for Decoding and Equalization," *2007 IEEE International Symposium on Information Theory*, Nice, France, 2007, pp. 1696-1700.

[12] M. Sikora and D. J. Costello, "A new SISO algorithm with application to turbo equalization," *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005.*, Adelaide, SA, Australia, 2005, pp. 2031-2035

[13] T. Johansson and K. Zigangirov, "A simple one-sweep algorithm for optimal APP symbol decoding of linear block codes", in *IEEE Transactions on Information Theory*, vol. 44, no. 7, pp. 3124-3129, Nov. 1998.

[14] M. Lentmaier and K. S. Zigangirov, "Iterative decoding of generalized low-density parity-check codes", in *Proceedings of the 1998 IEEE International Symposium on Information Theory*, Cambridge, MA, USA, 1998, pp. 149–.

[15] J. Boutros, O. Pothier, and G. Zemor, "Generalized low-density (Tanner) codes", in *1999 IEEE International Conference on Communications*, vol. 1, pp. 441–445, 1999.

[16] M. Lentmaier, G. Liva, E. Paolini, and G. Fettweis, "From product codes to structured generalized LDPC codes", in *2010 5th International ICST Conference on Communications and Networking in China*, pp. 1–8, 2010.

[17] M. Lentmaier, "Soft iterative decoding of generalized low-density parity-check codes based on MAP decoding of component Hamming codes," Diploma Thesis, 1997, Lund University, Sweden/Univeristy of Ulm, Germany.

[18] A. F. Molisch, *Wireless Communications*, 2nd ed. Wiley Publishing, 2011.

[19] W. An, M. Médard, and K. R. Duffy, "CRC Codes as Error Correction Codes", *CoRR*, vol. abs/2104.13663, 2021.