Titre: Title:	Single-Min LDPC Offset Optimization Methods
Auteur: Author:	Daniel Bowen Dermont
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dermont, D. B. (2022). Single-Min LDPC Offset Optimization Methods [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10295/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Single-min LDPC offset optimization methods

DANIEL BOWEN DERMONT
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Avril 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Single-min LDPC offset optimization methods

présenté par **Daniel Bowen DERMONT**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Christian CARDINAL, président
François LEDUC-PRIMEAU, membre et directeur de recherche
Éric ROY, membre

DEDICATION

for Aunt Nita

ACKNOWLEDGEMENTS

First, I would most like to thank Prof. Leduc-Primeau for this opportunity and his support through the pandemic. Special thanks to Jeremy Nadal for his patience, help and positive attitude. I would like to extend my thanks to my professors and collaborators from class - in particular, Profs. Cardinal, Frigon, Pesant and Cappart. Finally, I would like to thank my colleagues in the office for all the laughs; Hamed, Sebastien, Louis-Normand, and Simon.

RÉSUMÉ

Notre monde est de plus en plus dépendant des technologies à faible consommation d'énergie. Les téléphones mobiles et les ordinateurs portables ont une empreinte énergétique croissante, ce qui nécessite une amélioration constante des technologies employées dans les réseaux de communications. En plus d'augmenter le débit de transmission, les futurs réseaux de communication ont des contraintes en latence, en fiabilité et en consommation énergétique de plus en plus fortes. Bien que ces réseaux évoluent rapidement, les techniques sous-jacentes utilisées pour coder les messages sont restées les mêmes. Si les techniques de codage de canaux ont connu des avancées majeures au cours des 30 dernières années, les changements monumentaux que nous constatons dans la pratique sont le résultat d'innombrables heures d'optimisation et d'adaptation de ces techniques à des technologies comme Ethernet, WiFi et 5GNR. Les codes low-density parity-check (LDPC) sont largement utilisés dans les systèmes de communication.

Nous proposons des méthodes pour améliorer davantage l'efficacité énergétique des décodeurs LDPC, en utilisant des décodeurs single-minimum (SM). Contrairement aux décodeurs Min-Sum (MS), ces décodeurs ne calculent qu'un seul minimum pendant l'opération de mise à jour des messages, et appliquent un facteur de correction pour estimer le second. Bien que ces décodeurs soient beaucoup plus efficaces sur le plan énergétique que les décodeurs utilisés en pratique aujourd'hui, leur taux d'erreur binaire est dégradé en raison de la perte inhérente d'informations générée par l'émulation du second minimum. Les méthodes que nous proposons servent à optimiser les facteurs de correction utilisés dans l'émulation du second-min de manière à atténuer cette dégradation.

Dans ce mémoire, nous présentons une formulation générale du problème d'optimisation des facteurs de correcteur SM. Nous présentons ensuite des approches basées sur les simulations Monte Carlo (MC) pour résoudre ce problème, y compris des solutions obtenues avec une nouvelle méthode heuristique, appelée window search algorithm (WSA). Nous explorons également des solveurs basés sur des fonctions objectives à contraintes relaxées qui optimisent des facteurs de correction non-quantifié (valeur réelle). Ces facteurs sont ensuite quantifiés pour étudier l'effet du nombre de bits de quantifications sur les performances en taux d'erreur binaire. De plus, nous avons adapté l'analyse density evolution (DE) aux deux premières itérations du décodage SM, et nous avons proposé une méthode basée sur cette méthode de DE pour optimiser les facteurs de correction émulant le second-min. Nous évaluons ces méthodes proposées à l'aide de deux codes différents (normes 5G et Ethernet 10G)

et les comparons aux méthodes d'optimisation existantes.

Nous montrons que, pour l'Ethernet 10G, la méthode d'optimisation WSA surpasse significativement, en taux d'erreur binaire, le décodeur SM à décalage fixe pour des valeurs élevées de rapport signal/bruit (SNR). Pour le code 5G, des gains importants de taux d'erreur binaire sont observés, en particulier pour la méthode DE, qui nécessite également moins de temps de calcul. Cette méthode optimise les facteurs de correction SM uniquement pour les deux premières itérations, tandis que les facteurs de correction des itérations restantes sont extrapolés. Par conséquent, nous pensons que l'écart de performance entre les décodeurs MS et SM peut être encore réduit. Cela encourage la poursuite des recherches pour améliorer l'analyse DE pour les décodeurs SM.

ABSTRACT

Our world is increasingly dependent on low power technology. Mobile phones and laptops have a smaller footprint each year, requiring endless improvement on ever-changing technologies. Communication networks require less latency and higher sensitivity to climb to faster transmission speeds and higher bandwidths. However, as fast as these networks change, the underlying techniques used to encode our messages have stayed the same. While there have been major advances in channel coding over the past 30 years, the monumental changes we see in practice are the result of countless hours of optimization and adaptation of those techniques to technologies like Ethernet, WiFi and 5GNR.

Low-density parity-check (LDPC) codes are widely used in communication systems. We propose methods for further improving the energy efficiency of LDPC decoders, specifically using single-minimum (SM) decoders - quantized min-sum (MS) decoders which find a single minimum during the parity check operation and apply an offset to estimate the second. While these decoders are far more energy-efficient than decoders employed in practice today, they generally have impractical error-correction performance due to the inherent loss of information generated from second-min emulation. The methods we propose serve to optimize correction factors used in second-min emulation such that this degradation in performance may be mitigated.

In this thesis, we have formulated the generalized SM offset problem. We then present Monte Carlo-based (MC) approaches to solve this problem, including solutions gathered with a novel heuristic method, called the window search algorithm (WSA). We explore relaxed-constraint objectives and real-valued decoder results to investigate the effect of quantization constraints. We have additionally extended density evolution (DE) analysis to the first two iterations of SM decoding, and have incorporated a DE-based optimization method for second-min emulation.

We evaluate these proposed methods using two different codes (5G and 10G Ethernet standards) and compare with existing optimization methods. We show that, for 10G Ethernet, the WSA optimization method significantly outperform the fixed offset SM decoder at high signal-to-noise ratio (SNR) values. For the 5G code, large BER gains are observed, particularly for the DE method, which also requires less computation time. This method optimizes the SM offsets for only the first two iterations, while the remaining ones are extrapolated. Therefore, we believe that the performance gap between MS and SM decoders can be further reduced. This encourages further investigation to improve DE analysis for SM decoders.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

10GE 10 Gb/s Ethernet

5G 5^{th} generation cellular networks

APP a posteriori probability

ARQ automatic repeat request

BER bit-error rate

BG base-graph

BLER block error rate

BP belief propagation

BPSK binary phase-shift keying

C2V check-to-variable

CC convolutional code

CCDF complementary CDF

CD coordinate descent

CDF cumulative distribution function

CG conjugate gradient

CGU clock gating unit

CN check node

DE density evolution

EVW extended variable weight

FEC forward error correction

FER frame error rate

FF flip-flop

GA genetic algorithm

HARQ hybrid automatic repeat request

KKT Karush-Kuhn-Tucker

LDPC low-density parity-check

LLR log-likelihood ratio

LUT look-up table

MC Monte Carlo

MET multi-edge type

MS min-sum

MSE mean squared error

NMS normalized min-sum

OMS offset min-sum

PCM parity check matrix

PCS physical coding sublayer

PDF probability-density function

PMF probability-mass function

QC quasi-cyclic

RMS relaxed min-sum

RV real-valued

RVQ real-value quantized

SM single-minimum

SNR signal-to-noise ratio

SO soft output

SPA sum-product algorithm

SQP sequential quadratic programming

THz terahertz

V2C variable-to-check

VN variable node

WSA window search algorithm

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Since the discovery of low-density parity-check (LDPC) codes by Gallager in his 1962 dissertation [3], a major question has swirled around the application of these codes: how efficient can a decoding approach be while maintaining a practical performance level?

The goal of our work is to advance the most energy-efficient channel decoding techniques for use in low-power applications. To that end, we have identified an often-overlooked, efficient algorithm for LDPC decoding which is ripe for improvement - single-minimum decoding. The reason we see relatively little activity in this area of research is because of the performance degradation inherent in using only a single minimum in the min-sum parity check calculation. However, the difference in efficiency is hard to ignore; an single-minimum (SM) check node processor is estimated to be 48% the size of a conventional min-sum (MS) check node processor [4]. We propose in this thesis methods for improving the performance of the SM decoder; namely, methods for finding optimized single-min correction factors.

Belief propagation (BP) decoding for LDPC codes has existed since their inception as the best-performing approach to LDPC decoding with respect to error correction [3]. This is the approach used by MacKay and Neal in their work which is credited with reigniting interest in LDPC codes [5]. MacKay's sum-product formulation was done without knowledge of Gallager's previous work. Tanner's work on bipartite graphs in [6] was published in 1981, between the discovery of LDPC codes and their reintroduction by MacKay and Neal. This paper introduces a graphical model for long error-correction codes, notably including LDPC codes by name. More practical BP decoding algorithms have since been introduced, which improve on the efficiency of the decoder in exchange for a degradation in error correction performance - for example, the min-sum algorithm with correction and the reduced-complexity BP approaches presented in [7–9]. We see our work as a continuation of this trend, presenting the feasibility of more efficient approaches in BP approximation algorithms, especially with regards to quasi-cyclic (QC) LDPC codes. QC-LDPC codes are highly parallelizable and simple to implement in hardware using relatively little memory and cyclic shift registers. More information on QC-LDPC implementation is given in section 2.4.

The second goal of this research is to bring attention to the possibilities and difficulties in implementing density evolution analysis in single-min decoding schemes. Investigating the behavior of LDPC decoders generally involves long, costly Monte Carlo simulations. DE allows these analyses to be performed over probability mass functions rather than Monte Carlo results, and is therefore a much more resource efficient tool. DE relies on a few key

assumptions, outlined in Chapter 4. We will present the challenges inherent in extending this analysis tool to single-min decoding schemes, as well as optimization methods for SM decoders using DE.

1.1 Applications of LDPC

LDPC codes have highly parallel architectures and the ability to approach channel capacity. Forward error correction (FEC) refers to the ability of a code to iteratively correct errors during the channel decoding process. This method, as compared to automatic repeat request (ARQ), has a clear advantage in throughput. The two methods are often hybridized, where selective repeat requests are made if error correction is unsuccessful. This is known as hybrid automatic repeat request (HARQ). HARQ increases the average latency of the decoding process, but generally improves error-correction performance. LDPC codes are, like Turbo codes, capacity-approaching, meaning that their performance approaches the Shannon Limit, or capacity of a channel. First outlined in [10], the capacity of a channel refers to the maximum rate of transmission that information may be transmitted over a noisy channel free of errors, or with minimal errors. As explored by Richardson and Urbanke in [11] and [12], it is possible to design LDPC codes such that they perform at rates very close to this limit. LDPC decoding architectures, especially implementations using QC-LDPC codes, are highly parallelizable. Unlike Turbo codes, LDPC codes may be scaled to a desired code rate. That is, a turbo code achieves different code rates through puncturing a parent code, and therefore all decoders at all code rates are of the same complexity, whereas the complexity of LDPC codes may be scaled for specific applications.

Since their reintroduction by MacKay, LDPC codes have made their way from theory into practice - which is made possible in large part due to the efficiency offered by their highly parallel node processing architectures. We see LDPC codes built for a variety of applications - the most visible of which being 5^{th} generation cellular networks (5G) [13]. 10 Gb/s Ethernet (10GE) technology refers to a set of ethernet network standards, many of which are still in use. Published in 2006, 10GBASE-T, or IEEE 802.3an-2006, is one such standard that uses forward error correction using LDPC decoding in the physical coding sublayer (PCS) [14]. The DVB-S2 digital video broadcasting standard's most major improvement over its predecessor, DVB-S, was the incorporation of a much improved channel coding scheme using LDPC codes as opposed to concatenated convolutional and Reed-Solomon codes. The choice was made due the ease of implementation for LDPC codes due to their highly parallelizable architecture, as well as their 35% performance improvement over DVB-S standard Reed-Solomon and convolutional codes [15].

Quasi-cyclic binary LDPC codes are already the standard for 5G data channels, and with terahertz (THz) band technology making an entrance in 6G, LDPC will be the best choice for error correction coding, as it offers high performance with low decoding complexity [16]. Convolutional code (CC) LDPC codes are already being considered for use in impending 6G networks [17]. There already exist limitations in hardware in reaching the higher 5G transmission speeds [18]. The peak data rate proposed for 6G networks will be 1 Tbps, 50 times that of 5G networks [19]. There is a need for more efficient algorithms and hardware architectures to meet the upcoming demand that 6G presents.

1.2 Practical implementation

LDPC decoding architectures have gone through major changes since the first BP decoding algorithms proposed by Gallager in [3]. The highest error-correction performance is achieved using the sum-product algorithm (SPA) decoder, the algorithm which underlies Gallager's A and B algorithms outlined in [3]. SPA is the standard decoding algorithm for capacity-approaching codes. However, SPA includes a parity check process involving a product of hyperbolic tangent operations performed on continuous values. This is simply too costly to implement effectively in hardware. Therefore, reduced-complexity SPA decoding algorithms were developed, including MS decoding. As described in section 2.5.3, MS decoding replaces the costly product operation with a minimum operation - which in practice must find both the global minimum value and the next minimal value (explained in detail in section 2.7). MS decoding architectures have proven practical and may have error correction performance approaching the level of SPA decoding if parameterized correctly. However, as stated above, advancements in practical LDPC decoding must meet very high demands required by emergent network designs.

One way this may be done is by further reducing the complexity of the MS decoder parity check, by implementing the minimum operation to find the global minimum, and using this information to emulate the next minimal value. In the past, this modification was not made due to the inherent performance degradation that these SM decoders present. The goal of this thesis is to reduce the performance gap between MS decoding and SM decoding enough to allow for practical SM implementations.

1.3 DE-based analysis

Usually, we evaluate the error correction performance of an LDPC code for different noise powers using Monte Carlo (MC) simulations. However, these simulations are not time- or memory-resource efficient. In fact, the high execution times of MC simulations were the major roadblock in evaluating SM decoder implementations throughout this research.

Density evolution (DE) analysis is a method where, given a transmission error rate to design for, an upper bound for a channel noise parameter performing at or below this error rate is established. Conversely, DE may be used to find the transmission error rate at a designed-for channel signal-to-noise ratio (SNR) value. DE is an important tool for qualifying empirical results in this space - and doubles as a very useful tool in exploring error performance for LDPC codes without using the same resources as a conventional simulation - supposing the results of DE analysis for an LDPC code ensemble can be shown to be within a permissible range of error when compared to Monte Carlo simulations. In addition, DE analysis is very resource-efficient.

Based on channel model and noise power, we have a channel output distribution from which we can gather a density function. DE begins by assuming a code is of infinite length, and that no cycles exist in the decoder implementation. Though these assumptions are false, they allow analysis to proceed using evolving probabilistic random variables. We obtain an "infinite-length" result from DE applied under these assumptions. The work presented in [20] and [21] introduces a transformation considering finite-length codes, which improves the densities rendered by DE compared to the results of simulations performed on real decoders.

1.4 Thesis Organization

Chapter 2 offers a review of channel coding, with a particular focus on belief propagation for LDPC codes. We begin with a review of the communication channel model, followed by an exploration of Tanner graph representations and code generation techniques. We then review LDPC decoding algorithms, beginning with the sum-product algorithm and moving step-by-step to min-sum decoding with correction. We show how relaxing some of the constraints of sum-product decoding can lead to very practical algorithms for LDPC decoding. A short review of flooding and layered decoder scheduling is given, and we present a short overview of our quantization scheme. We end this chapter with a detailed overview of the SM Offset decoding approach; the most basic SM approach and the trade-offs inherent in implementing it. Next, we present improved algorithms from the literature, using both fixed and variable offset approaches.

In Chapter 3, we present our approach for optimizing the single-min Offsets used in the Single-Min decoding algorithm. We present the choice of single-min offsets as an optimization problem, where the objective is reduction of Bit Error Rate for a MC simulation, based on our choice of discrete SM offset input. We next present a coordinate descent (CD) process which simplifies the search space by targeting optimization parameters individually. We then present a relaxed-constraint approach where a solution is emulated using an analagous, continuous-valued problem. We explore more efficient methods of optimization, including a new heuristic method called window search algorithm (WSA), which is a method of decomposing these optimization problems into smaller searches while incorporating memory between decoder iterations. Finally, an overview is given for a number of discrete optimization methods which are incorporated into WSA.

Chapter 4 begins with a review of properties of random variables that motivate DE analysis. An in-depth description of DE analysis for MS decoding follows, including a transformation which improves the accuracy of DE analysis using code length. We then present our approach to DE for the single minimum case, including a successful analysis of the first and second decoder iterations. Finally, we present an optimization method that uses DE to calculate SM offsets for each edge by minimizing mean squared error (MSE) against the performance of a standard MS decoder.

In Chapter 5, we present the preliminary results for the relaxed-constraint solution to the SM offset problem, which did not perform as expected. We then show the error performance of our discrete optimization solutions for SM decoders plotted against the performance of an MS decoder for both a regular (10GE) and irregular (5G) code. This notably includes SM offsets obtained using WSA and our DE optimization method.

In Chapter 6, we present our conclusions regarding the SM decoder performance results. This chapter includes a review of the contributions presented herein and a roadmap for future work.

CHAPTER 2 LITERATURE REVIEW

2.1 Channel Model

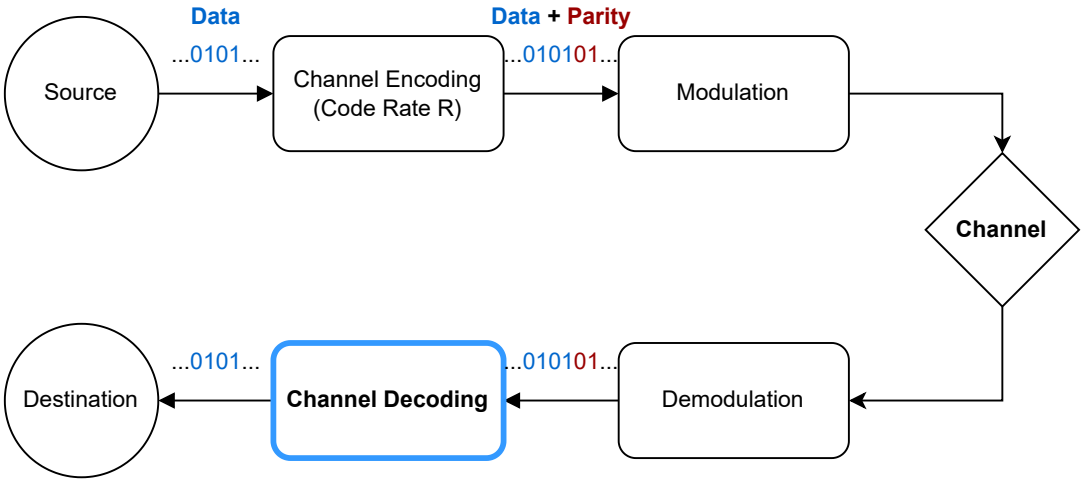

Figure 2.1 Generalized communication channel

Figure 2.1 depicts a communication channel model. This is the scheme by which information travels across a channel. We begin with source encoding, the process by which information is encoded from its original form to a form that may be manipulated by the modules in the transmitter (often binary code). Channel encoding represents a transformation of this source-encoded data which allow parity checks to be performed upon receipt of the data on by the decoder. These parity checks - performed in the highlighted "Channel Decoding" module in Figure 2.1 - allow the receiver to evaluate the quality of data and/or correct transmission errors. Modulation is the process by which the channel encoded data is transformed into a signal suitable for transmission across the specific channel. While modulation may affect the reliability of a communication channel with respect to error rate, channel coding represents the most significant process in preventing transmission errors and creating a robust channel.

The BI-AWGN channel model allows additive noise to be applied according to a normal distribution on real-valued symbols representing the belief for a bit-value in the transmitted code. This is the channel model used in the following research. We assume symbols are transmitted in a causal sequence $i = 0, 1, 2, ...$ and are modulated. We represent the i^{th} transmitted symbol with noise, at channel output, y_i, as a sum of the i^{th} modulated symbol $x_i \in \{-1, 1\}$ with the noise w_i:

$$y_i = x_i + w_i \,, \tag{2.1}$$

where w_i has variance σ^2 and mean 0. In this research, binary phase-shift keying (BPSK) modulation is employed, meaning that we may express x_i in terms of bit values as follows:

$$x_i = 1 - 2b_i \,, \tag{2.2}$$

where b_i represents the i^{th} bit value transmitted and $b_i \in \{0, 1\}$. An advantage of this modulation method is that, upon receipt of BPSK modulated x_i values with added noise, a hard decision can be made for the bit-value y_i by simply evaluating its sign. Therefore, the conditional probability of $b_i = \{0, 1\}$ given the channel output y_i is defined: [22]

$$\Pr(b_i = 1|y_i) = \left(1 + e^{\frac{-2y_i}{\sigma^2}}\right)^{-1}, \tag{2.3}$$

$$\Pr(b_i = 0|y_i) = \left(1 + e^{\frac{2y_i}{\sigma^2}}\right)^{-1}. \tag{2.4}$$

According to the mapping performed by (2.2), a hard decision $\hat{y}_i$ can be taken based on the sign of the channel output:

$$\hat{y}_i = \begin{cases} 1 \,, y_i \leq 0 \,, \\ 0 \,, y_i > 0 \,. \end{cases} \tag{2.5}$$

Decoding performed on hard decision values introduces a loss of information and, in turn, an error performance degradation - typically $2 - 3$ dB. It is preferable to perform decoding on 'soft' inputs, where the initial belief value for the channel output and the message values being passed in the decoder are continuous log-likelihood ratio (LLR) values.

We apply channel output LLR values to a corresponding variable node in the decoder to find our "initial belief" value for each bit. This likelihood value is referred to as L_i, where i is the variable node index, or alternatively the i^{th} transmitted bit. We define our log-likelihood ratio

$$L_i = L(b_i|y_i) = \log\left(\frac{P(b_i = 0|y_i)}{P(b_i = 1|y_i)}\right), \tag{2.6}$$

where y_i is the channel output received by the i^{th} variable node (VN) and b_i is the true value of the corresponding message bit. Since we are operating with a BI-AWGN channel, we can combine Equations (2.4) and (2.6) to form

$$L_i = \frac{2y_i}{\sigma^2}\,. \tag{2.7}$$

2.2 Channel Coding and Decoding

Channel coding refers to a process by which a message, often a binary encoded message, is transformed into a codeword, a version of the message which contains redundant information. After transmission over the channel and upon reception, an inference based on this codeword is made to find the original data that was sent by the source. FEC coding is a channel coding paradigm where the decoding process allows one to recover the original message with some probability. An initial likelihood for each message symbol is determined by the received codeword, and this likelihood is updated based on parity check operations specified by the code. These parity check operations are specified by a Parity Check Matrix (PCM), which uniquely describes an FEC code. A Low Density Parity Check (LDPC) code is an FEC code with a very large, sparse PCM. Non-zero entries of the PCM specify connections between variable nodes (VN) - which represent the current belief for a bit value at a particular iteration - and check nodes (CN) - which represent the parity check operation. For instance, a non-zero entry of the PCM at (j,i) represents a connection between the j^{th} check node and the i^{th} variable node. We are concerned in this research with binary LDPC codes.

The PCM, often represented mathematically as $\boldsymbol{H}$, uniquely defines a generator matrix $\boldsymbol{G}$ that is used for encoding transmissions. The binary message which is encoded is often referred to as the 'data' or 'data word,' which we will represent $\boldsymbol{u}$. The encoded message - the product $\boldsymbol{uG}$ - is referred to as the 'codeword,' and we will represent it here as $\boldsymbol{b}$. The set of all valid codewords is called a 'codebook,' and the number of bit differences between two codewords is called 'distance'. The 'minimum distance,' or 'Hamming distance,' is defined for a code, and is the shortest distance between any two codewords in the codebook. The "code rate"

$R = k/n = ||\boldsymbol{u}||/||\boldsymbol{b}||$ denotes the proportion of data in the overall codeword, where k is the number of rows in the H-matrix and n is the number of columns, or, alternatively, k is the bit-length of the data word $\boldsymbol{u}$, and n is the bit-length of the codeword $\boldsymbol{b}$.

Due to the symmetric behavior of both the channel and the decoder, as well as the linearity of the LDPC code, additive noise functions the same no matter the channel input. Therefore, an all-zero codeword is employed to simplify Monte Carlo simulation. The error performance of an LDPC decoder is independent of the transmitted codeword, so it is equivalent to use the all-zero codeword - which is always present in any linear block code's codebook.

Decoding error performance is calculated using two main metrics: bit-error rate (BER) and block error rate (BLER), also known as frame error rate (FER). A bit error occurs when a bit-value at the output of the decoder does not match the corresponding bit-value at channel input. A block error occurs when the codeword at the decoder output does not match the channel input, or equivalently when at least one bit-error occurs in the transmitted block. We may obtain these metrics through Monte Carlo simulation. A bit error rate (BER) curve depicts the BER performance of a particular decoder over a range of channel normalized signal-to-noise ratio (E_b/N_0) values measured in decibels, graphed with a log-scale for the error rate axis. The BER and BLER behaviors of an LDPC code often follows a specific pattern; error correction performance improves steadily at low E_b/N_0, but, at some E_b/N_0 value, the performance improves very sharply. This region of sharp improvement is called the 'waterfall' region. At a larger E_b/N_0 value, the sharp improvement ceases and the perform stagnates due to message saturation or cycles in the code. This flat region is called the 'error floor' region. These distinct behaviors and their E_b/N_0-value boundaries are important for evaluating the error correction performance of a decoder. Two decoders may be compared at a specific BER or BLER value by measuring the dB 'gap' between curves at the specified error rate, or by measuring the gap in error performance seen at a specified E_b/N_0 value.

2.3 Graph Representation

While an LDPC code may be uniquely represented by a PCM, the code may also be represented as a corresponding bipartite graph known as a Tanner graph. On one side of the graph are variable nodes, often represented with circles, and on the other are check nodes, often represented by boxes. The nodes are connected by a series of edges, which represent bi-directional connections between the check and variable nodes. Nodes connected by these edges are said to be 'neighbors.' A check node will have a number of neighboring variable nodes, and likewise a variable node has a number of neighboring check nodes. The number of edges connected to a check or variable node is denoted as the 'degree' of the node. If

we instead see this from the perspective of the parity-check matrix, the row weight of row j denotes the degree of check node j and the column weight of column i corresponds to the degree of variable node i.

A code for which the check node degree of each check node and variable node degree of each variable node are constant is said to be 'regular,' and a code for which this does not hold is consequently 'irregular.' The check and variable node degree of a regular code are often identified by a convention: a (d_c, d_v)-regular code has CN degree d_c and VN degree d_v.

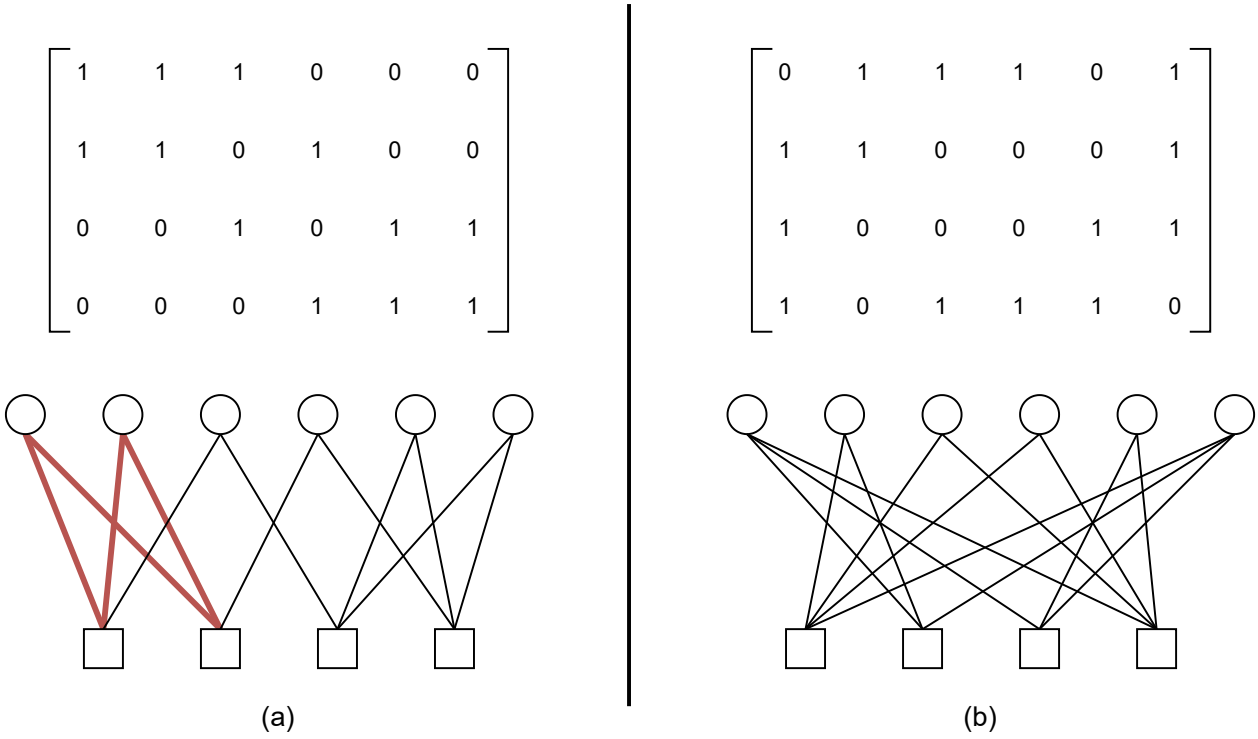

Figure 2.2 Depiction of Tanner graphs and PCMs for a regular and irregular code, with a highlighted cycle

A 'cycle' in the code may more easily be identified using the Tanner graph. Referring to Figure 2.2, side (a) depicts a (2,3) regular code. Side (b) depicts an irregular code with VN degrees $\{2,3\}$ and CN degrees $\{3,4\}$. A cycle has been identified in the regular code, the participating edges shown in red. The 'girth' of a code is measured as the length of the shortest cycle. Cycles are not desirable, as they introduce dependencies, and allow certain errors to be amplified during decoding. However, a cycle-free Tanner graph cannot support good codes, or codes that perform better than those with cycles [23]. This is because the cycle-free constraint for a Tanner graph imposes an upper bound on the minimum distance

of the code - the minimum distance d for any code rate R cannot exceed 2 for a cycle-free code. This is an unsuitable condition for FEC. We often therefore employ codes of girth 6 or 8, as the errors in these longer cycles take more iterations to propagate, while allowing for the implementation of codes with significantly higher minimum distances.

2.4 Quasi-cyclic codes

A quasi-cyclic (QC) code is a parity-check code with a PCM which consists of cells which are represented by cyclically permuted identity matrices. A PCM for a QC code may alternatively be represented as a base-graph (BG). Each entry of the base graph represents an identity matrix with a particular circular shift applied. The dimension of the identity matrix cells is known as the 'lifting size,' z, and the alphabet used in the base graph is determined by this parameter. Entries of the base graph may take integer values within the domain $[-1, z-1]$. Each identity permutation is expressed as an integer on $[0, z-1]$. The entry -1 represents a square zero matrix with dimension z, and the 0 entry represents an identity without a shift applied. Each entry of the BG corresponds to a permutation index, which is determined by the circular shift rule respected. In this case, permutation indices correspond to the number of rows shifted from the bottom of the matrix to the top. Figure 2.3 shows an example of this permutation index rule applied to a matrix $z = 3$. Permutation index 0 corresponds to the identity matrix.

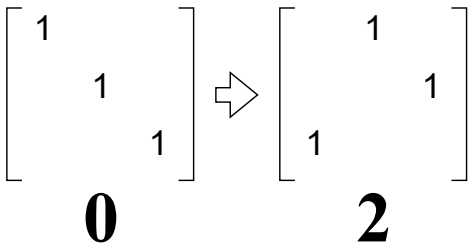

Figure 2.3 Example of a circular shift performed on an identity matrix size $z = 3$, along with the corresponding permutation indices

A BG representation of the PCM simply represents all $z \times z$ non-zero cells of the PCM with their corresponding permutation indices. This representation of the PCM is much smaller, and since encoding and decoding may be performed on QC codes using only circular shifts on the data and codewords respectively, the BG stores all necessary information to describe a QC-LDPC code.

2.5 Decoding

2.5.1 Notation

The following notation will be used to discuss message-passing throughout the remainder of this thesis. Refer to Figure 2.4 for an abstract model depicting message variables. Variables i, j, and ℓ are scalars:

- $\lambda_{j\to i}^{(\ell)}$: Variable-to-Check node message from VN j to CN i at iteration ℓ
- $\gamma_{i\to j}^{(\ell)}$: Check-to-Variable node message from CN i to VN j at iteration ℓ
- $\Lambda_j^{(\ell)}$: Log-likelihood ratio total stored in VN j at iteration ℓ
- $\mathcal{V}_j$: the set of VN indices connected to CN j
- $\mathcal{C}_i$: the set of CN indices connected to VN i

2.5.2 Sum-Product Algorithm

The sum-product algorithm (SPA) is the unaltered belief propagation decoding algorithm, and is therefore the best known algorithm with respect to error correction. [24] However, as we will see below, the check node update operation in this algorithm is too costly and inefficient to use for most applications. Below we detail the flooding scheduled SPA algorithm.

SPA decoding starts by updating variable nodes with channel output information. We set the outgoing VN message $\lambda_{i\to j}^{(0)} = L_i$, as shown in Equation (2.7), for every non-zero entry of the parity check matrix (PCM) $\boldsymbol{H}$, denoted $h_{j,i} = 1$. A check node receives VN messages from its neighbors, and updates outgoing CN messages accordingly. Importantly, a CN message update operation excludes the extrinsic VN message information - that is, if the i^{th} VN is being sent a CN message from the j^{th} check node, the message $\lambda_{i\to j}^{(\ell-1)}$ from the previous decoder iteration is excluded in the calculation of $\gamma_{j\to i}^{(\ell)}$. We can think of this as considering the subset of VN indices $\mathcal{V}_j \setminus \{i\}$. We define $f(.)$ as the generalized check-to-variable (C2V) γ-update rule function, in the case of SPA, the update rule given by: [24]

$$\gamma_{j\to i}^{(\ell)} = f(\boldsymbol{\lambda_{i\to j}^{(\ell-1)}}) = 2\tanh^{-1}\left(\prod_{k\in-\mathcal{V}_j\setminus\{i\}} \tanh\frac{1}{2}\lambda_{k\to j}^{(\ell-1)}\right). \tag{2.8}$$

The initial C2V message is given by $\gamma_{j\to i}^{(0)} = 0$.

Next, we have the variable node and LLR total update operations. These operations are standard in all BP LDPC decoding algorithms. To calculate the next outgoing VN message, a sum of all incoming neighboring CN messages except the one received on the edge being updated is taken, including the initial belief value. For a message from VN i to CN j, the update operation is as follows:

$$\lambda_{i\to j}^{(\ell)} = L_i + \sum_{k\in C_i\setminus j} \gamma_{k\to i}^{(\ell-1)}\,. \tag{2.9}$$

The LLR total is the updated estimation for the codeword at a given decoding iteration. While very similar to the VN update calculation, note that this LLR sum is taken over all VN neighbors without excluding any incoming messages. The LLR total is given by:

$$\Lambda_i^{(\ell)} = L_i + \sum_{k\in C_i} \gamma_{k\to i}^{(\ell-1)}. \tag{2.10}$$

After the variable nodes update their LLR total, the current iteration is over. The check node and variable node update operations alternate until a stopping criterion is met. We have:

$$\gamma_{j\to i}^{(\ell)} = f\left(\boldsymbol{\lambda}_{\bar{e},i\to j}^{(\boldsymbol{\ell-1})}\right), \tag{2.11}$$

$$\boldsymbol{\lambda}_{\bar{e},i\to j}^{(\boldsymbol{\ell-1})} = [\lambda_{i\to k}^{(\ell-1)}]_{\forall k\in\mathcal{V}_j\setminus j}\,, \tag{2.12}$$

with $\boldsymbol{\lambda}_{\bar{e},i\to j}^{(\boldsymbol{\ell-1})}$ corresponding to the vector composed of all λ-messages connected to CN index j, excluding the extrinsic message (edge $i \to j$). Note that the subscript $\bar{e}$ here refers to extrinsic message exclusion.

Stopping Criterion

The current iteration's estimate for the codeword is given

$$\hat{v}_i^{(\ell)} = \begin{cases} 1\,, \text{if } \Lambda_i^{(\ell)} < 0\,, \\ 0\,, \text{else.} \end{cases} \tag{2.13}$$

This is essentially the hard decision outlined in Equation (2.5) at an arbitrary iteration of the decoder, ℓ. If $\hat{\boldsymbol{v}}\boldsymbol{H}^T = 0$, the potential codeword $\hat{\boldsymbol{v}}$ is valid. The product $\hat{\boldsymbol{v}}\boldsymbol{H}^T$ is known as a 'syndrome,' and therefore a 'zero syndrome' corresponds with a valid codeword. For a long

enough code, there is a small probability of receiving a false positive, and receiving a valid codeword is a strong enough stopping criterion. An iteration limit, $\ell_{\max}$, is often imposed, meaning that once a threshold is reached ($\ell = \ell_{\max}$), the potential codeword $\hat{v}$ is used as decoder output. These stopping criteria are standard among BP decoding algorithms.

The LLR total stored by the variable nodes during decoding operates as our 'best guess' at the transmitted bit value at the corresponding codeword bit. The rule given in Equation (2.13) outlines the current iteration's 'decision' for the transmitted codeword; $\hat{v}_j$ is the decoder's estimate at the current iteration for the transmitted codeword. This is a 'hard decision,' as the estimator has chosen the bit values for each transmitted bit, whereas the vector $\Lambda_i^{(\ell)}$ shows the corresponding 'soft decision,' the real-valued LLR totals which correspond to the belief for the bit value at the current iteration. Equation (2.13) may be equivalently represented using the signum function, with bit-decisions performed by evaluating $\mathfrak{s}(\lambda_{i\to j}^{(\ell)} + \gamma_{j\to i}^{(\ell)})$, where $\mathfrak{s}(.)$ represents the signum function.

2.5.3 Min-Sum

The min-sum algorithm adjusts SPA by making a simple assumption about the product operation: that the minimal incoming VN message dominates the product. If this assumption holds, it means that the CN update product operation may be replaced instead by a more simple minimum calculation. Min-Sum is therefore a far less complex algorithm, which only suffers a slight performance degradation [25, 26].

We begin by decomposing each outgoing VN message value into sign and magnitude

$$\lambda_{i\to j}^{(\ell)} = \operatorname{sign}\left(\lambda_{i\to j}^{(\ell)}\right) \times \left|\lambda_{i\to j}^{(\ell)}\right|. \tag{2.14}$$

Now that we have our message values decomposed, we can formulate a new CN update rule using the assumption stated above: the product of message values taken in the SPA CN update is dominated by their minimum value. We define $f(.)$ for min-sum decoding by: [27]

$$f(\boldsymbol{\lambda^{(\ell-1)}}) = \left\lfloor \max\left(\min\ |\boldsymbol{\lambda^{(\ell-1)}}|, 0\right) \times \prod_{\lambda^{(\ell-1)} \in \boldsymbol{\lambda^{(\ell-1)}}} \mathfrak{s}(\lambda^{(\ell-1)}) \right\rfloor. \tag{2.15}$$

This approximate algorithm, while slightly more error-prone, is far less costly to implement due to the product approximation, and, as we will see, may be altered to improve performance using scaling and offset parameters.

Figure 2.4 depicts the min-sum decoder's check node update function from (2.15). Let the message contributed by VN i, $\lambda_{i\to j}^{(\ell-1)}$, during this update be the global minimum LLR value received by CN j, and let the global minimum be unique. For the min-sum decoder to calculate the outgoing CN message to VN i, $\gamma_{j\to i}^{(\ell)}$, the message $\lambda_{i\to j}^{(\ell-1)}$ must be excluded as it is the extrinsic message. This means that the global minimum among the remaining messages, circled in green in Figure 2.4, must be found. It is equivalent to find both the global minimum and next minimal LLR value, and apply this second minimum when the extrinsic message is the unique global minimum.

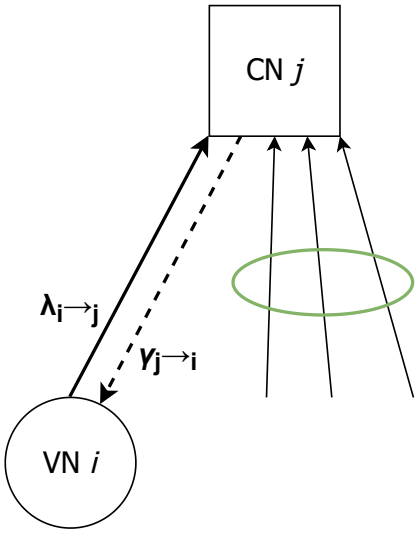

Figure 2.4 Min-Sum decoder check node update as seen from variable node i

2.5.4 Min-Sum with correction

We can improve the error correction performance of the MS algorithm by applying correction factors to the message values in the CN update. For an MS decoder with correction, we introduce a normalization factor $\eta \in \mathbb{R}^{*}+$ and an offset value $\beta \in \mathbb{N}$. An offset min-sum algorithm uses an offset value, β, which can further improve the decoder's error correction performance. The values η and β can be designed for at fixed E_b/N_0 values according to the desired performance benchmarks. Of course, a decoder may utilize both forms of correction. Equation (2.15) can be changed to introduce normalization factor and offset:

$$f(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \eta \times \Big\lfloor \max\Big(\min\ |\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| - \beta, 0\Big) \times \prod_{\lambda^{(\ell-1)} \in \boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}} \mathfrak{s}(\lambda^{(\ell-1)}) \Big\rfloor. \tag{2.16}$$

Note that, for $\eta = 1$ and $\beta = 0$, the formulas in Equations (2.15) and (2.16) will be equivalent.

2.5.5 Message-passing Scheduling

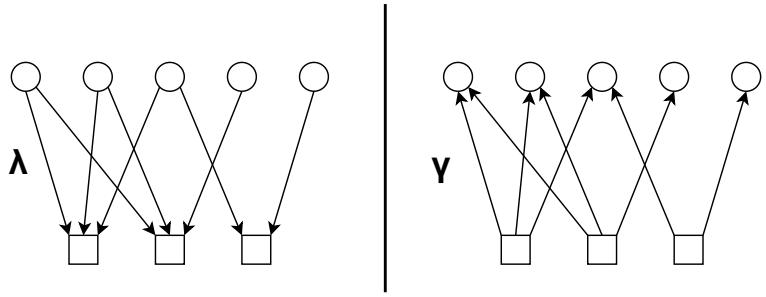

Figure 2.5 Illustration of one iteration of a flooding-scheduled decoder

There are two main scheduling paradigms in BP decoding: flooding (parallel) and layered (serial) scheduling. The algorithms described above all adhere to a flooding schedule as they are presented; this means that all outgoing variable node messages, $\lambda_{i\to j}^{(\ell-1)}$, update at once, followed by all outgoing check node messages, $\gamma_{j\to i}^{(\ell)}$, updating at once. Figure 2.5 depicts a flooding scheduled decoder. The left side of the figure, labeled λ, shows the outgoing VN messages, while the right side depicts the outgoing CN messages, labeled γ.

We can instead perform these update operations adherent to a layered schedule. A layered schedule is divided into sub-iterations, where each sub-iteration represents a single layer updating. There are different ways to layer the schedule of the decoding process - we will use a row-layered schedule as an example.

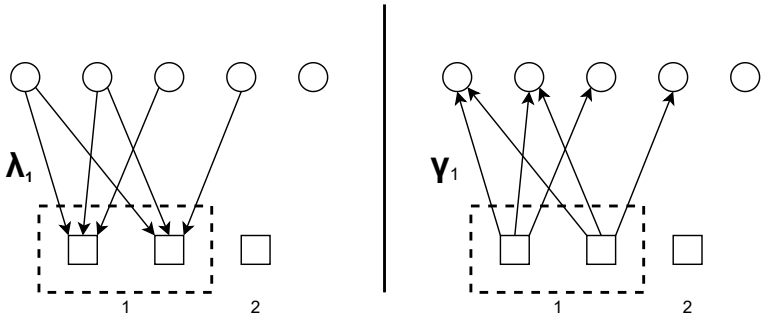

Figure 2.6 Illustration of one sub-iteration of a row-layer-scheduled decoder

Figure 2.6 shows a sub-iteration of a row-layer-scheduled decoder. In a row-layered schedule, a number of row layers are established. For instance, the rows of the PCM associated with a row of the base-graph of a QC-LDPC code, explained in Section 2.4, could be grouped into a row layer. This is represented in Figure 2.6 by the dashed box about the first two check nodes. A sub-iteration begins with all variable nodes neighboring the check nodes in the

subject row layer updating at once, labeled λ_1 in the figure, and each check node in the layer in turn updating its neighbors, labeled γ_1. The subscript is used here to clarify that this is a sub-iteration and does not represent the full layered iteration. The variable nodes contribute incomplete belief values. This process continues for each layer, so that after the final layer has been updated, each outgoing variable node message is completed. The advantage of using a layered schedule is that, for long enough codes, decoding is expected to converge in half the number of iterations as for a flooding schedule [28].

2.6 Quantization

In evaluating LDPC for practical applications, quantization must be considered, as practical decoders quantize LLR message values. A quantized channel LLR message may be given $L \in [-Q, Q]$, where $Q \in \mathbb{N}^*$ represents the maximum value representable by our chosen quantization scheme. All messages and channel outputs are quantized. The maximum value Q is determined by the number of bits used to represent message values. Given the number of bits, n, used in the quantization scheme, we have $Q = 2^{n-1} - 1$.

Real-valued channel outputs and messages passed between nodes are rounded to values representable under this scheme. Using an LLR scaling factor α, the step size of the quantization scheme are scaled to parameterize and improve the precision of the scheme, as the LLR scaling factor is allowed to be a floating point number. We can apply this quantization scheme for our BPSK modulated, AWGN channel by transforming Equation (2.7):

$$L_i = \mathrm{sat}_Q \left\lfloor \frac{2\alpha y_i}{\sigma^2} + \frac{1}{2} \right\rfloor, \tag{2.17}$$

where y_i is the received channel output message received by the i_{th} VN, σ^2 is the noise variance, α is a constant LLR scaling factor, $\lfloor . \rfloor$ is the floor operator and sat_Q is a saturation operator that ensures that $L_i \in [-Q, Q]$. A variable with a tilde - such as $\tilde{\lambda}_{i \to j}^{(\ell)}$ - is a shorthand for a message where saturation has been applied. Any message value below $-Q$ is rounded to $-Q$ and, likewise, any value above Q is rounded to Q:

$$\mathrm{sat}_Q(x) = \begin{cases} Q, & \text{if } x > Q, \\ -Q, & \text{if } x < -Q, \\ x, & \text{otherwise.} \end{cases} \tag{2.18}$$

Parameter α should be chosen such that the least amount of saturation occurs for a given application, as oversaturation leads to reinforcement of noise and an elevated error floor. There is an inherent tradeoff in choosing the scaling factor. A higher LLR scaling factor α allows for more precise calculations due to the smaller step size. However, a higher LLR scaling factor also makes saturation more likely, and will lead to a loss of information. We can see that if we allow α to tend toward infinity in Equation (2.17), the channel output y_i will directly determine the decoder output due to saturation - an operation equivalent to performing a hard decision on the raw channel output. And, if we allow α to tend toward 0, we lose any initial information for message values, as $L_i|_{\alpha=0} = \lfloor \frac{1}{2} \rfloor = 0$.

In addition to channel output, saturation is applied to internal message values and offsets. If we consider a real-valued offset MS decoder with no quantization, following Equation (2.16), we will apply an offset β to the message magnitude value $|\lambda_{i\to j}^{(\ell)}|$. When a quantized case is then considered, it is important to apply LLR scaling not only to $|\lambda_{i\to j}^{(\ell)}|$, but distribute the scaling to β as well. To have an equivalent value for β in the quantized case, we incorporate LLR scaling factor α:

$$\beta_Q = \lfloor \alpha \times \beta \rceil, \tag{2.19}$$

where β_Q represents the equivalent min-sum offset for a quantized case with LLR scaling factor α, and the operator $\lfloor . \rceil$ represents a rounding to the nearest integer. If both β and L_i are quantized using integer values, all decoder operations are performed on integers for all iterations. However, variable-to-check (V2C) message still need to be saturated to avoid overflow.

Equation (2.9) shows the VN update operation that is common to all BP LDPC decoding algorithms. For iteration 1, we take a sum of $d_v - 1$ incoming CN messages with the channel output - an operation which can achieve a maximum value of $Q \times d_v$. Without saturation, performing subsequent iterations' VN update would further increase this possible maximum. Therefore, we saturate VN message values, changing the decomposition shown in Equation (2.14) to reflect this: $\tilde{\lambda}_{i\to j}^{(\ell)} = \text{sign}\,(\tilde{\lambda}_{i\to j}^{(\ell)}) \times |\tilde{\lambda}_{i\to j}^{(\ell)}|$.

VN belief Λ has a max value of $(d_v + 1)Q$, and is quantized on $\lceil \log_2 \left((d_v + 1)Q \right) \rceil$ bits. We can saturate these values on the interval $[-(d_v + 1)Q, (d_v + 1)Q]$.

LLR scaling factors are stored as floating point numbers, as a high precision LLR scaling factor has a high impact in terms of performance.

2.7 Single-Min Decoding

Due to the exclusion of extrinsic LLR information during the check node update, a practical min-sum decoder will find both the global minimum of incoming messages and a second minimum which excludes the global minimum message. To achieve high decoding throughput, a cascaded tree-like structure is implemented. The elementary block in this cascaded structure is shown in Figure 2.7. It finds the first and second minimum for 2 couples of first minimum and second minimum message values previously computed. While clearly less complex than the sum-product algorithm, this 2-min search structure is still the highest-area component in the LDPC decoder.

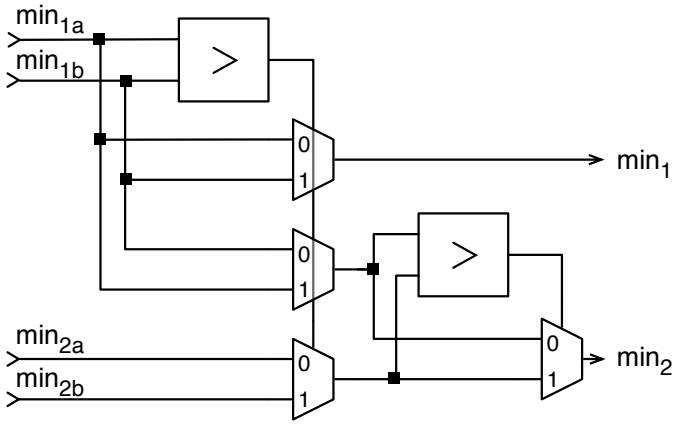

Figure 2.7 Architecture of the 2-min search elementary block [1, Fig. 2b]

This cascaded structure generally corresponds to the critical path of the LDPC decoder if no pipeline registers are added. Pipelining is a method by which a process may be divided into sub-processes, where each sub-process is buffered by a register which stores inputs and outputs at each step. This allows the sub-processes to be completed in parallel, and therefore can greatly increase the throughput and reduce idle time for any particular step. For a decoder architecture processing up to d_c messages in parallel, the tree depth is equal to $D_T = \lceil \log_2 d_c \rceil$, and the number of cascaded comparator blocks is $N_{\text{comp}} = 2 \times D_T$. This means that, for a deeply pipelined architecture, up to N_{comp} pipeline registers need to be instantiated. This further increases the area and the power consumption of the decoder. Additionally, layered-schedule LDPC architectures have trouble supporting high pipeline depths due to potential conflicts during the VN update. This leads to either BER degradation or decoding throughput loss due to inserted idle cycles. When considering hybrid scheduled LDPC architectures, high pipeline depths can lead to higher decoding iterations, which leads to a penalty in the decoding throughput.

Therefore, there is an interest in further simplifying the offset min-sum (OMS) decoding algorithm. One simple approach is to replace the 2-min search tree with a global minimum search and a second minimum emulation; a method known as single-minimum (SM) decoding. By consequence, the min search elementary block in Figure 2.7 is replaced, with only the comparator and the multiplexer outputting $\min_1$ remaining. While the tree depth D_T remains unchanged, the number of cascaded comparators needed for the SM architecture is halved when compared to the typical 2-min search architecture ($N_{\text{comp}} = D_T$). The number of required pipeline registers is likewise reduced, saving area and power consumption. But one question remains: what level of BER performance degradation is incurred by this single-min approach?

The MS γ-update function can be simplified for SM decoding by only computing one minimum magnitude per check node, as proposed in [4]. This is equivalent to violating the extrinsic exclusion rule during the message update:

$$\gamma_{e,j\to i}^{(\ell)} = \mathfrak{s}\big(\lambda_{i\to j}^{(\ell-1)}\big) f\big(\boldsymbol{\lambda}_{\boldsymbol{e,i\to j}}^{\boldsymbol{(\ell-1)}}\big), \tag{2.20}$$

$$\boldsymbol{\lambda}_{\boldsymbol{e,i\to j}}^{\boldsymbol{(\ell-1)}} = \lambda_{i\to k}^{(\ell-1)},\ \forall k \in \mathcal{V}_j\,. \tag{2.21}$$

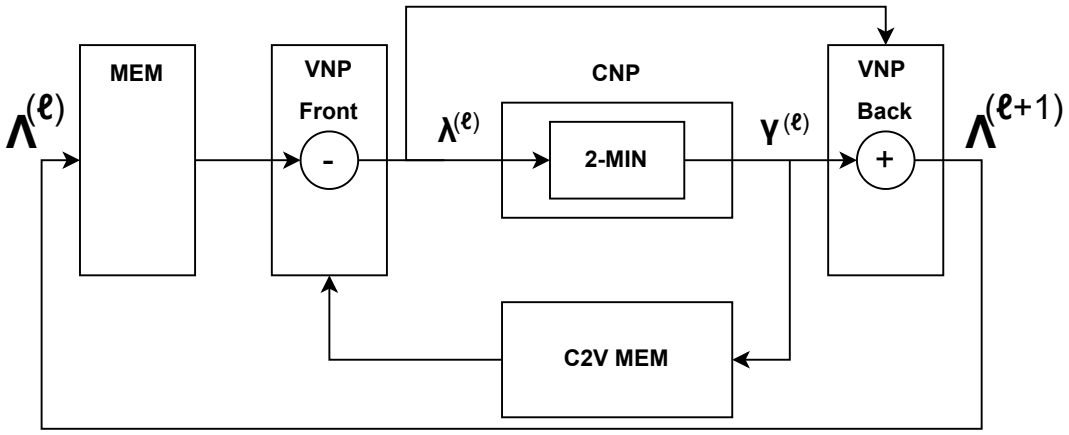

Figure 2.8 General memory framework of min-sum decoder [2, Fig. 1]

The subscript e here refers to the inclusion of extrinsic information in the message update function. As a result, the number of cascaded comparators is divided by 2 in the check node processing unit, reducing its complexity and propagation delay by half. This is an important consideration - and a major motivation for improving single-min architectures. We can consider a general framework for the min-sum decoder as depicted in Figure 2.8.

To reduce the performance gap, [29–31] propose an emulation of the second minimum magnitude by adding a correction factor in the form of an offset that may vary during the decoding procedure. The most general approach is to affect an offset $\omega_{i,j}^{(\ell)}$, referred to as an SM offset, for an edge $i \to j$ and iteration ℓ. The emulation of the second minimum occurs when the minimum V2C message magnitude is extrinsic: $|\lambda_{i\to j}^{(\ell-1)}| < |\gamma_{\bar{e},j\to i}^{(\ell)}|$, $\gamma_{\bar{e},j\to i}^{(\ell)}$ being the message obtained using (2.16) without applying correction factors. Thus, we have: [4, 31, 32]

$$\gamma_{e,j\to i}^{(\ell)} = \mathfrak{s}\left(\gamma_{\bar{e},j\to i}^{(\ell)}\right) \times |\gamma_{e,j\to i}^{(\ell)}|, \tag{2.22}$$

$$|\gamma_{e,j\to i}^{(\ell)}| = \begin{cases} \min\left(\eta\left\lfloor |\lambda_{i\to j}^{(\ell-1)}| + \omega_{i,j}^{(\ell)} \right\rfloor, \mathcal{Q}\right) & \text{if } |\lambda_{i\to j}^{(\ell-1)}| < |\gamma_{\bar{e},j\to i}^{(\ell)}|, \\ \max\left(\eta\left\lfloor |\gamma_{\bar{e},j\to i}^{(\ell)}| - \beta \right\rfloor, 0\right) & \text{otherwise.} \end{cases} \tag{2.23}$$

2.7.1 Existing Methods

Equation (2.22) shows the most general formula for applying a correction factor in a single-min decoder. The case $\omega_{i,j}^{(\ell)} = 0$ corresponds to the single-min decoder output taken without any correction. We see this decoder explored in the literature mainly as a point of comparison, [4]. The performance of this decoder is heavily affected by the loss of the second min calculation, and this decoder tends to saturate and have a high error floor. The correction scheme outlined in [4] is the case $\omega_{SM} = 1$. This correction factor improves somewhat the performance of the single-min decoder, though the BER error floor region is still high. We can alternatively attempt to find an optimal value within some bounds for the value $\omega_{i,j}^{(\ell)}$. This approach was implemented in [29,30,33] with a configurable correction offset, which is shown to outperform the algorithm introduced in [4] and closer approximate the performance of a min-sum decoder. However, to more accurately correct the minima, it will become necessary to vary the single-min offsets in terms of the iteration number or CN degree.

Varying single-min offsets by iteration is a concept explored in [31]. We see that not only do the authors have a fixed SM offset decoder, whose performance is compared to both min-sum and normalized min-sum performances, but the authors include a scheme for varying SM offsets by iteration number. The decoders are layered scheduled, with a maximum iteration count of 30. Several quantization precisions are explored, using a $[q : f]$ paradigm, where q represents the total number of bits used for quantization (including the sign bit), and f represents the number of fractional bits used. A $[7 : 2]$ scheme would therefore correspond to a 7-bit quantized decoder with an LLR scaling factor of 4 according to the our paradigm, shown in Section 2.6. The single-min offsets used in this paper are given as a vector of weights and a vector of thresholds. The thresholds correspond to the iteration where the next weight-vector entry is applied. While this scheme does allow for changing single-min

offsets with respect to decoder iteration, the quantization scheme used limits the performance of the decoder. The LLR scaling quantization scheme we implement allows for more optimally chosen quantization values.

Reference [34] applies these ideas to offset min-sum decoding, with a similar approach to weighting the correction factor as compared to [31]. The work in [34] notably includes offsets varying not only by iteration number, but also by check node degree. The code used is a 5G basegraph index 1 code, and shows the most notable gain in performance of any SM offset optimization. The SM offset optimization in [34] first establishes the following setup, where CN's of the given degrees use the SM offsets correspondent to the group: $G_1 = \{3\}$,$G_2 = \{6, 7, 8, 9, 10\}$ and $G_3 = \{19\}$. CN group setup is formalized in Section 3.1. Based on their group index, SM offsets are then passed to CN's based on a polynomial function varying by iteration number. Notably, the quantization scheme of the SM decoders presented in [34] is unclear. Figure 5.4 shows the results of the highest performance SM offsets from [34], named *EvwsmOMS*, using a coordinate descent (CD) optimized LLR scaling and normalization factor.

CHAPTER 3 Single-Min Offset Optimization

The object of the most relevant existing methods is, as stated in Section 2.7.1, to find an optimal value within some bounds for the value $\omega_{i,j}^{(\ell)}$. These solutions should also be discrete valued - employing the LLR scaling quantization method described in Section 2.6. Little attention has been paid in the literature as to how single-minimum (SM) offsets may be efficiently optimized. We propose a more general formulation of the offset optimization problem, as well as solutions using novel techniques. Notably, the SM offset optimization problem has not received a formal definition in the literature, which we present here. We then present a number of discrete optimization methods applied to this problem.

3.1 Problem formulation

Let there be a matrix of positive integer SM offsets $\mathbf{\Omega}$ of size $n_{\text{edge}} \times \ell_{\max}$, where $n_{\text{edge}} = \sum_j |\mathcal{V}_j|$ is the number of edges in the Tanner graph and $\ell_{\max}$ is the maximum number of iterations supported by the decoder. Each matrix entry corresponds to an SM offset $\omega_{i,j}^{(\ell)}$ applied at a specific decoder iteration $\ell \in [1, \ell_{\max}]$ and edge $j \to i, \forall j, i \in \mathcal{V}_j$. We wish to find the matrix $\mathbf{\Omega}$ such that BER is minimized. Let the $(\eta, \alpha, \mathbf{\Omega})$ tuple represent a particular choice of decoder parameters. Correction factors η, α and offset matrix entries $\omega_{i,j}^{(\ell)}$ have for maximum values $\eta_{\max}, \alpha_{\max}$ and $\omega_{\max}$. The optimal parameters $(\eta^\star, \alpha^\star, \mathbf{\Omega}^\star)$ that minimize the BER performance $B(\eta, \alpha, \mathbf{\Omega}, \zeta)$ of the SM decoder fixed at $E_b/N_o = \zeta$ are given by solving

$$(\eta^\star, \alpha^\star, \mathbf{\Omega}^\star) = \underset{(\eta, \alpha, \mathbf{\Omega})}{\arg\min}\, B(\eta, \alpha, \mathbf{\Omega}, \zeta). \tag{3.1}$$

The BER $B(\eta, \alpha, \mathbf{\Omega}, \zeta)$ can be estimated through MC simulation. Problem (3.1) is infeasible to jointly optimize since it is non-convex and the solution space is large. In addition, MC simulation requires extensive simulation times. Hence, the search space must be reduced. One solution is to constrain the offset matrix $\mathbf{\Omega}$ using one of three approaches: i) similarly to [29, 30], all offsets can be set to a single scalar value ($\omega_{i,j}^{(\ell)} = \omega$), ii) the offset values only vary with the iteration index ($\omega_{i,j}^{(\ell)} = \omega^{(\ell)}$) or iii) with the edge index ($\omega_{i,j}^{(\ell)} = \omega_{i,j}$).

Approach (iii) can be simplified by grouping edges into N_G groups, then assigning the same offset value ω_u to all edges belonging to the u^{th} group for a given iteration ℓ. It is useful in practice to design these groups based on the CN degree $\text{dc}_j = |\mathcal{V}_j|$ of each message, as proposed in [34]. If G_u is the set of CN degrees belonging to group u ($G_u \cap G_{u'} = \emptyset$ if $u \neq u'$), then $\omega_{i,j}^{(\ell)} = \omega_u, \forall i$ if $\text{dc}_j \in G_u$. Note that the approach described in this paragraph can be

combined with (ii), i.e. offset values vary by both edge-group and iteration index. Then, $\boldsymbol{\Omega}$ can be equivalently represented as an $N_G \times \ell_{\max}$ matrix denoted $\boldsymbol{\Omega_G} = [\omega_u^{(\ell)}]$. For the rest of this section, we consider $\boldsymbol{\Omega_G}$ as the offset matrix to be optimized.

It is worth noting that it is not necessary to perform a full search over the maximum iteration count intended for the SM decoder. Instead, the findings gathered from a search over fewer iterations can be extrapolated to design for the intended iteration limit $\ell_{\max}$. For approaches ii) and iii), extrapolation can be performed using linear regression, with saturation at zero if extrapolated offsets take negative values. To design $\boldsymbol{\Omega_G}$, two-dimensional extrapolation can be performed using a modified Akima method [35, 36], derived from cubic interpolation.

3.2 Top-level optimization using coordinate descent

To further simplify the search space for optimizing $(\eta, \alpha, \boldsymbol{\Omega_G})$, we propose optimizing these parameters through CD [37]. CD allows optimization to target each parameter separately; the remaining parameters are fixed while a search is performed to find the best BER-performing value for the targeted parameter, as shown in the equations below

$$\boldsymbol{\Omega}_{\boldsymbol{G}}^{\star} = \arg\min_{\boldsymbol{\Omega_G}} B(\eta^{\star}, \alpha^{\star}, \boldsymbol{\Omega_G}), \tag{3.2}$$

$$\eta^{\star} = \arg\min_{\eta} B(\eta, \alpha^{\star}, \boldsymbol{\Omega}_{\boldsymbol{G}}^{\star}), \tag{3.3}$$

$$\alpha^{\star} = \arg\min_{\alpha} B(\eta^{\star}, \alpha, \boldsymbol{\Omega}_{\boldsymbol{G}}^{\star}). \tag{3.4}$$

The above equations are applied in a loop over I iterations. For (3.3) and (3.4), a single-variable minimizer tool may be used, as these values are scalars with definite bounds. Optimizations stopping criteria depend on methods used, though a comparison limit may be imposed. We use a minimization algorithm based on a golden section search and parabolic interpolation, with tolerance value 10^{-4} [38, 39]. Problem (3.2) is more complex; methods to solve it will be described in further detail in Section 3.4 and Section 4.

An initial solution must be identified for all parameters before performing CD. We observed there is little difference in CD-optimized α and η between MS decoding and fixed offset SM decoding. Therefore, we propose to first find initial values for α and η using CD on (3.3) and (3.4), with BER obtained through MS decoding (variable $\boldsymbol{\Omega_G}$ is excluded). We then move to the SM decoder to set the initial value of $\boldsymbol{\Omega_G}$. For simplicity, we consider that all coefficients of $\boldsymbol{\Omega_G}$ are fixed to ω, such that $\omega_u^{(\ell)} = \omega$. This constraint applies only for this initial step. Then, ω is obtained by performing CD on (3.2)–(3.4). This allows α and η values to be further optimized.

3.3 Relaxed Real-Valued Decoder Method

We have explored a relaxed-constraint search for regular codes: it is a straightforward approach to first solve problems using real-valued decoders and then quantize the results. A real-valued decoder in this case refers to SM decoding performed without quantization; this is, of course, impractical, but allows for the relaxed-constraint solution. This raises the question: how precise can our quantization scheme be while maintaining this superior performance?

3.3.1 Problem Formulation

The real-valued varying offset decoders which utilize the solutions found by the nonlinear optimizations can be referred to as real-valued (RV) decoders, as opposed to decoders using their corresponding quantized solutions which can be referred to as real-value quantized (RVQ) decoders. We can rewrite Equation (3.2) to instead consider the real-valued solution $\boldsymbol{\Omega_{RV}}$:

$$\boldsymbol{\Omega}^{\star}_{\boldsymbol{RV}} = \arg\min_{\boldsymbol{\Omega_{RV}}} B(\eta^{\star}, \alpha^{\star}, \boldsymbol{\Omega_{RV}}). \tag{3.5}$$

Once this solution is rendered by the solver, the real-valued offsets can be quantized according to :

$$\boldsymbol{\Omega_G} = \lceil \alpha' \times \boldsymbol{\Omega}^{\star}_{\boldsymbol{RV}} \rfloor, \tag{3.6}$$

where $\boldsymbol{\Omega}^{\star}_{\boldsymbol{RV}}$ corresponds to the real-valued offsets rendered by the solver, and α' is a scaling factor for quantization that can be different from the LLR scaling factor α. However, for simplicity, we consider $\alpha' = \alpha$. All solutions were initialized using the optimal SM fixed offset solution.

The goal of this method is to quantize the results of our real-valued single-min offset searches to see at what precision - or number of bits used in quantization - is needed to preserve this performance. Quantization will inherently degrade the error-correction performance of an SM decoder, however, the execution time of performing the convexity-solver search, quantization and measuring the performance of the resultant SM decoder is much faster than that of an exhaustive search. Unfortunately, as shown in Section 5.1, the real-value performance could not be recovered at a practical level of precision for a regular code, and thus this method is impractical.

3.3.2 Nonlinear Optimization

Four separate nonlinear optimization algorithms, shown in [40], were employed to solve (3.5): *Trust-Region-Reflective* [41, 42], *Active-Set* [43–45], *SQP* [46] and *Interior-Point* [47–49]. Suppose we have an objective function describing an optimization problem, $m(x)$, subject to constraints $n_i(x) \geq 0, i \in [1, \mu]$. We have:

$$\min_x m(x), \tag{3.7}$$

$$n_i(x) = 0\,,\ 1 \leq i \leq \mu_e\,, \tag{3.8}$$

$$n_i(x) \geq 0\,,\ \mu_e + 1 \leq i \leq \mu\,, \tag{3.9}$$

$$x_{lb} \leq x \leq x_{ub}\,. \tag{3.10}$$

where there exist μ_e equality constraints and $\mu - \mu_e$ inequality constraints, and x is bounded according to $x \in [x_{lb}, x_{ub}]$. Given a constraint $n_i(x)$, the constraint is considered *active* at x if $n_i(x) = 0$, and is considered *inactive* at x if $n_i(x) > 0$. The 'Active set' is made up of those constraints which are active at x [46].

Trust-Region-Reflective

If we consider a simplified function $m'(x)$ which approximates the behavior of $m(x)$ within some neighborhood, or 'trust region,' N about x, we can establish a step s by minimizing about N:

$$\min_s \left\{m'(s), s \in N\right\}, \tag{3.11}$$

If $m(x + s) < m(x)$, the current point is updated and a new trust region is determined. Otherwise, the trust region N is reduced and the step is recomputed [50]. However, in the *Trust-Region-Reflective* method used, a two-dimensional step subspace is calculated, using the linear subspace determined by a first step s_1 in the direction of the gradient, and another either (i) in an approximate direction $H_x \cdot s_2 = -g$, or (ii) a direction of negative curvature $s_2^T \cdot H_x \cdot s_2 < 0$. H_x here refers to the Hessian matrix of m taken at x, and g refers to the gradient [51, 52].

Active-Set and SQP

The Karush-Kuhn-Tucker (KKT) equations describe first-order derivative tests which are both necessary and sufficient for an optimal solution. They are given:

$$\nabla m(x) + \sum_{i=1}^{\mu} \lambda_i \nabla n_i(x) = 0\,, \tag{3.12}$$

$$\lambda_i \nabla n_i(x) = 0,\ 1 \le i \le \mu_e\,, \tag{3.13}$$

$$\lambda_i \ge 0,\ \mu_e + 1 \le i \le \mu\,. \tag{3.14}$$

where λ_i here represents Lagrange multipliers.

Solving the KKT equations motivates many nonlinear optimization algorithms, and these algorithms are referred to as sequential quadratic programming (SQP) algorithms [**Note:** *SQP* is used to refer to the particular algorithm used for simulation, SQP refers to the general formulation of a sequential quadratic programming optimization]. The remaining three methods (*Active-Set*, *SQP*, and *Interior-Point*) are all SQP methods. We wish to iteratively solve a QP subproblem. This problem is obtained, as shown in [43, 45], by first making a quadratic approximation of the Lagrangian function, and then linearizing the nonlinear constraints for a particular solution $x_k \in [x_{lb}, x_{ub}]$:

$$L(x, \lambda) = m(x) + \sum_{i=1}^{\mu} \lambda_i n_i(x), \tag{3.15}$$

$$\min_{d \in \mathbb{R}^2} \frac{1}{2} d^T H_{x_k} d + \nabla m(x_k)^T d\,, \tag{3.16}$$

$$\nabla n_i(x_k)^T d + n_i(x_k) = 0,\ 1 \le i \le \mu_e\,, \tag{3.17}$$

$$\nabla n_i(x_k)^T d + n_i(x_k) \le 0,\ \mu_e + 1 \le i \le \mu\,. \tag{3.18}$$

The solution of the QP problem is used to formulate successive solutions according to step size a_k: $x_{k+1} = x_k + a_k d_k$. Finally, the solution rendered by the QP subproblem d_k is evaluated using a merit function, which in turn determines the step size a_k. The merit function used, implemented in [53, 54], is:

$$\Psi(x) = m(x) + \sum_{i=1}^{\mu_e} r_i \cdot n_i(x) + \sum_{i=\mu_e+1}^{\mu} r_i \cdot \max[0, n_i(x)], \tag{3.19}$$

where r_i is a penalty, in [54] determined by

$$r_i = (r_{k+1})_i = \max\left\{\lambda_i, \frac{(r_k)_i + \lambda_i}{2}\right\}, i \in [1, \mu], \tag{3.20}$$

and initialized as

$$r_i = \frac{||\nabla m(x)||}{||\nabla n_i(x)||}. \tag{3.21}$$

An iteration of a generalized SQP algorithm consists of: (1) Updating the Hessian matrix H_{x_k}, (2) Solving the QP subproblem, (3) Evaluating the merit function.

The *Active-Set* and *SQP* algorithms are similar, with *SQP* having important differences. The step size a_k in *SQP* is constrained by bounds. *SQP* can take a step that fails - where the objective function for the candidate solution is undefined. In this case, *SQP* will attempt a smaller step size, as opposed to simply re-initializing, as *Active-Set* would. Additionally, *SQP* uses more memory-efficient algebraic methods to solve the QP subproblem (Equations (3.16)-(3.18)).

Interior-Point

While the *Interior-Point* algorithm does take 'direct steps,' in which it evaluates the KKT equations to determine the next solution, it may alternatively determine a step using a conjugate gradient (CG), which uses a trust region. If a direct step fails, a CG step is attempted, meaning *Interior-Point*, like *SQP*, can take a step that fails.

A barrier function is an approximation of the original problem (3.7), described in [47–49]. The barrier function is given ($\forall \pi > 0$):

$$\min_{x,\boldsymbol{\sigma}} m_\pi(x, \boldsymbol{\sigma}) = \min_{x,\boldsymbol{\sigma}} m(x) - \pi \sum_i \ln(\sigma_i), i \in [\mu + 1, \mu]. \tag{3.22}$$

subject to the same constraints as (3.7), with the addition of $\sigma \geq 0$. There are as many entries of $\boldsymbol{\sigma}$ as there are inequality constraints. As $\pi \to 0$, the minimum of m_π will approach that of m. This barrier function determines the 'slack' variables $\boldsymbol{\sigma}$, which are in turn used to determine steps in both the direct step and conjugate gradient paradigms. The constraint $\sigma \geq 0$ is used to esure that these steps are taken within a trust region.

3.4 SM offset optimization using windowed search

SM decoding operations involve dependencies that allow the offset at one iteration to change the behavior of later iterations. This can be explained by the fact that messages become correlated over successive iterations due to the extrinsic exclusion principle being violated, as demonstrated later in Section 4.2.1. Therefore, problem (3.2) is not convex, and unlike MS decoding, offsets cannot be optimized on a per-iteration basis. We propose introducing a top-level heuristic that incorporates memory from previous decoder iterations to optimize subsequent ones, which we refer to as WSA.

We denote as $\boldsymbol{\Omega_G}[\ell]$ the offset matrix of the $\ell \leq \ell_{\max}$ first iterations (size $N_G \times \ell$). Defining a window length L, we propose finding $\boldsymbol{\Omega_G} = \boldsymbol{\Omega_G}[\ell_{\max}]$ by successively optimizing $\boldsymbol{\Omega_G}[\ell]$ from $\ell = L$ to $\ell = \ell_{\max}$ with step $p \in [1, L]$. During each step, the L last columns of $\boldsymbol{\Omega_G}[\ell]$, corresponding to a submatrix $\boldsymbol{W}$ of size $N_G \times L$, are optimized through a search. Meanwhile, the remaining coefficients are kept constant during the optimization process. The resulting optimized offset matrix is denoted $\boldsymbol{\Omega}^{\star}_{\boldsymbol{G}}[\ell]$. By noting that $\boldsymbol{\Omega_G}[\ell] = [\boldsymbol{\Omega}^{\star}_{\boldsymbol{G}}[\ell - L], \boldsymbol{W}]$, $\boldsymbol{\Omega}^{\star}_{\boldsymbol{G}}[\ell]$ can be obtained by solving the following optimization problem:

$$\boldsymbol{W}^{\star} = \arg\min_{\boldsymbol{W}} B(\eta^{\star}, \alpha^{\star}, [\boldsymbol{\Omega}^{\star}_{\boldsymbol{G}}[\ell - L], \boldsymbol{W}]), \tag{3.23}$$

and setting $\boldsymbol{\Omega}^{\star}_{\boldsymbol{G}}[\ell] = [\boldsymbol{\Omega}^{\star}_{\boldsymbol{G}}[\ell - L], \boldsymbol{W}^{\star}]$. The size of the window $\boldsymbol{W}$ governs the complexity of this optimization, along with the discrete method chosen. In this research, the search over $\boldsymbol{W}$ is performed using a genetic algorithm, or, when $L \times N_G$ is sufficiently small, through exhaustive search (see Section 3.5). Then, ℓ is incremented by p, and the next offset matrix $\boldsymbol{\Omega_G}[\ell]$ is optimized through the same procedure. This process repeats until $\ell = \ell_{\max}$. Figure 3.1 depicts an iteration of the search for $p = 1$, at an arbitrary iteration ℓ:

Windowed search can alternatively be applied to a complete solution. In this case, the search is applied as before, but iterations beyond the window are initialized to that of an existing solution, and $\ell = \ell_{\max}$ for every iteration of the search.

3.5 Discrete Optimization Methods

There remains a question: what optimization method may we employ to solve the WSA subproblem (3.23)? Additionally, these solutions must be discrete valued. Two of the simplest archetypes for optimizations are exhaustive and random searches.

An optimization by exhaustive search simply tests all possible inputs for the most optimal among them. This method is achievable for regular codes over few enough iterations, and

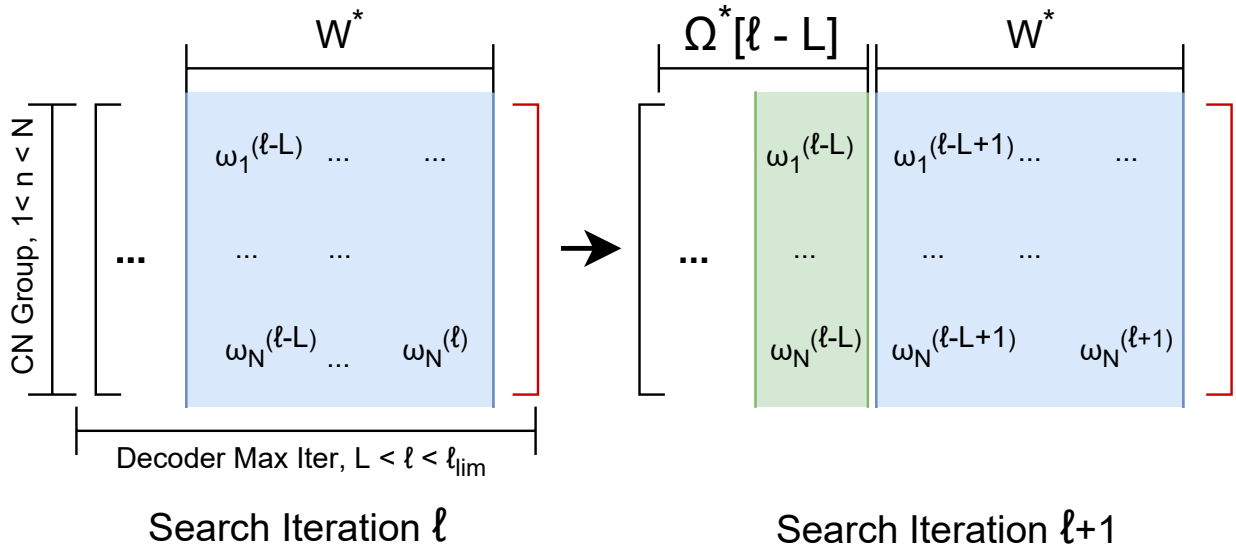

Figure 3.1 Example of an iteration of the generic windowed search

ensures the optimality of the solution. However, if we wish to apply SM integer offsets $\omega_{i,j}^{(\ell)} \in [0, 10]$ over $\ell_{\max}$ iterations, this requires $11^{\ell_{\max}}$ trials of our Monte Carlo simulation, which is simply not feasible due to time constraints. A random search instead instantiates a random solution - in this case, choosing SM offsets for each edge at random within an established range. While this solution is easily the fastest, it will render the optimal solution with very low probability.

A stochastic optimization is an optimization performed using random variables. A practical stochastic optimization method is the genetic algorithm. This algorithm mirrors natural selection by establishing a set of candidate solutions ('individuals'), or a 'population,' measuring the performance of each ('fitness'), choosing the best performing solutions ('elites'), and then combining them with other individuals ('selection'). At each iteration, the population generated is called a 'generation.' In the case of SM offset optimization, this is performed by simply generating SM offset matrices, evaluating their performance through an objective function (a Monte Carlo decoding simulation), and partitioning and recombining the elite solutions. 'Mutations' may also be introduced, by simply randomly applying additive noise to entries of the SM offset matrix [55].

The genetic algorithm begins with a random initial population, for generation $g = 0$, $\boldsymbol{C_0}$, determined by an initial population count $|\boldsymbol{C}|$, with individuals $\boldsymbol{C}_{g_i}$ and bounds $\boldsymbol{C}_{g_{lb}} \leq \boldsymbol{C}_{g_i} \leq \boldsymbol{C}_{g_{ub}}$. The performance of this population is measured, and fitness values ρ are assigned

based on each individuals performance. A number n_e of the most fit individuals are chosen as elites. The formation of subsequent generations ('children' of the current generation) is done according to three rules: (1) Elite children: Elites pass to the subsequent generation automatically, (2) Crossover children: pairs of individuals are chosen for combination into new candidates, (3) Mutation children: a single individual is chosen for mutation. The algorithm may end when a number of stopping criteria are met, including when a maximum number of generations I_{lim} have been performed or the algorithm has stalled for a number of iterations I_{stall}, an overall time limit t_{lim} or a maximum stall time within a generation t_{stall} has been reached, or a fitness limit ρ_{lim} has been attained by a candidate solution.

CHAPTER 4 Application of Density Evolution

The main drawback of MC-based searches outlined in the previous section resides in their long simulation time. In the following chapter, we investigate a faster, analytical approach for solving (3.2) using DE. We present DE analysis for MS decoding. We then present our findings regarding the DE analysis of the first two iterations of the generalized SM decoder. This chapter presents the findings regarding DE from [32] in more detail. The work presented in [32] introduces both the DE analysis for the first two iterations of an SM decoder, as well as the DE-based SM offset optimization method. A review of properties of random variables that are important for deriving DE equations may be found in Appendix A.

4.1 DE for MS Decoding

DE, first proposed for sum-product decoding in [11], is an analysis tool that uses probabilistic properties of belief propagation to predict the behavior of a decoder at each iteration. If we assume that a code is free of dependencies introduced by cycles in the Tanner graph, we can accurately predict the performance of a decoder at each iteration by evolving the probability density of the channel output. The result given by this analysis assumes that the LDPC code length tends to infinity. A review of DE for MS decoders is given in [56]. To simplify presentation, we present the equations for single-edge-type codes, but it is also possible to develop equations for multi-edge-type codes.

Assuming an AWGN channel, the channel output LLR cumulative distribution function (CDF), denoted $\Phi_{\lambda}^{(0)}(k), k \in [-Q, Q]$, is given by

$$\Phi_{\lambda}^{(0)}(k) = \frac{1}{2} + \frac{1}{\sqrt{4\sigma^2}}\mathrm{erf}\left(\left(k + \frac{1}{2}\right)\frac{\sigma^2}{\alpha} - 1\right), \tag{4.1}$$

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt\,, \tag{4.2}$$

where $\mathrm{erf}(x)$ is the Gauss error function. The resulting probability-mass function (PMF) is denoted $P_{\lambda}^{(0)}(k)$ or $\boldsymbol{P}_{\boldsymbol{\lambda}}^{(\mathbf{0})} = [P_{\lambda}^{(0)}(-Q), ..., P_{\lambda}^{(0)}(Q)]$ in vector format.

4.1.1 Variable-to-Check message PMF

The density of V2C messages at iteration ℓ is given:

$$P_{\lambda}^{(\ell)}(k) = \left(\left[\circledast_{d_v-1} \boldsymbol{P}_{\gamma}^{(\boldsymbol{\ell})} \right] \circledast \boldsymbol{P}_{\lambda}^{(\mathbf{0})} \right)(k), \tag{4.3}$$

where $\boldsymbol{P}_{\gamma}^{(\boldsymbol{\ell})} = [P_{\gamma}^{(\ell)}(-Q), ..., P_{\gamma}^{(\ell)}(Q)]$ is the C2V message PMF vector and $\circledast_n$ operator is a n-fold convolution on a vector $\boldsymbol{X}$ with itself. We can see that equation (4.3) requires determining the PMF of a sum of d_v random variables. This formula is presented in equation (A.11) in Section A.

4.1.2 Check-to-Variable message PMF

The PMF of the C2V message corresponds to $P_{\gamma}^{(\ell)}(k) = \Pr\left(s(\gamma^{(\ell)})|\gamma^{(\ell)}| = k\right)$, which can be calculated by separating the cases:

- where $k < 0$, implying that $\mathfrak{s}\left(\gamma^{(\ell)}\right) = -1$ and $|\gamma^{(\ell)}| = -k$,
- where $k = 0$, implying that $\gamma^{(\ell)} = 0$,
- where $k > 0$, implying that $\mathfrak{s}\left(\gamma^{(\ell)}\right) = 1$ and $|\gamma^{(\ell)}| = k$.

This translates to the following equation:

$$P_{\gamma}^{(\ell)}(k) = \begin{cases} \Pr\left(|\gamma^{(\ell)}| = -k \ \cap \mathfrak{s}(\gamma^{(\ell)}) = -1\right) & k < 0\,, \\ \Pr\left(\gamma^{(\ell)} = 0\right) & k = 0\,, \\ \Pr\left(|\gamma^{(\ell)}| = k \ \cap \mathfrak{s}(\gamma^{(\ell)}) = +1\right) & k > 0\,. \end{cases} \tag{4.4}$$

First, we consider the simplest case, when $k = 0$:

$$P_{\gamma}^{(\ell)}(0) = \Pr\left(\min|\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| = 0\right). \tag{4.5}$$

We can instead think of the probability that $k = 0$ as the complement of the probability that $k \neq 0$. The equation related to the PMF of the minimum of a vector is given in Section A, using either equation (A.6) or (A.7), which gives:

$$P_{\gamma}^{(\ell)}(0) = 1 - \left(1 - P_{|\lambda^{(\ell-1)}|}(0)\right)^{d_c-1}. \tag{4.6}$$

Next we consider the case when $k > 0$, denoted $P^{(\ell)}_{\gamma,+}(k)$:

$$P^{(\ell)}_{\gamma,+}(k) = \Pr\Big(\min|\boldsymbol{\lambda^{(\ell-1)}}| = k \ \cap \mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = +1\Big). \tag{4.7}$$

First, let $\boldsymbol{\mathcal{S}_p}$ be a set of $\boldsymbol{\delta} = [\delta_1, ..., \delta_{d_c-1}]$ vectors, $\forall i, \delta_i \in \{-1, +1\}$, such that there are exactly p negative ("-1") elements in each $\boldsymbol{\delta}$ vector:

$$\boldsymbol{\delta} \in \boldsymbol{\mathcal{S}_k} \Leftrightarrow \sum_{i=1}^{d_c-1} \delta_i = d_c - 1 - 2p. \tag{4.8}$$

Let $\boldsymbol{\mathcal{E}}$ be a set of $\boldsymbol{\delta}$ vectors such that there is an even number (including 0) of negative ("-1") elements in each $\boldsymbol{\delta}$ vector:

$$\boldsymbol{\mathcal{E}} = \bigcup_{p \text{ even}} S_p \,. \tag{4.9}$$

Obviously, $\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = +1$ if there are an even number of negative $\lambda^{(\ell-1)}$ values, i.e. $\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) \in \boldsymbol{\mathcal{E}}$. Therefore, by listing all the possible $\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}})$ combinations, (4.7) becomes

$$P^{(\ell)}_{\gamma,+}(k) = \sum_{\forall \boldsymbol{\delta} \in \boldsymbol{\mathcal{E}}} \Pr\Big(\min|\boldsymbol{\lambda^{(\ell-1)}}| = k \ \cap \ \mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = \boldsymbol{\delta}\Big). \tag{4.10}$$

Furthermore, each $\lambda^{(\ell-1)}$ random variables are identically distributed, meaning the position of each $\lambda^{(\ell-1)}$ in $\boldsymbol{\lambda^{(\ell-1)}}$ does not matter. This also applies for their sign. For instance, we have $\Pr\Big(\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}} = [-1,-1,1,1,...,1]\Big) = \Pr\Big(\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}} = [1,..,1,-1,-1]\Big)$, where the two "$-1$" signs are located in the last rows of the vector instead of the beginning. In other words, $\forall(\delta_n, \delta_m) \in \mathcal{S}^2_k, n \neq m$, we have

$$\Pr\Big(\min|\boldsymbol{\lambda^{(\ell-1)}}| = k \ \cap \ \mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = \boldsymbol{\delta_m}\Big) = \Pr\Big(\min|\boldsymbol{\lambda^{(\ell-1)}}| = k \ \cap \ \mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = \boldsymbol{\delta_n}\Big). \tag{4.11}$$

Therefore, all the terms in (4.10) with $\boldsymbol{\delta} \in \boldsymbol{\mathcal{S}_p}$, p even, can be factorized by a common term denoted $\boldsymbol{\delta_p}$. For instance, $\boldsymbol{\delta_p}$ can be defined such that the first p rows are composed of "-1", and the remaining rows composed of "$+1$":

$$\boldsymbol{\delta_p} = [\underbrace{-1,-1,...,-1}_{\text{length } p}, \underbrace{+1,+1,+1,...,+1,+1}_{\text{length } d_c-1-p}]. \tag{4.12}$$

The number of terms factorized is equal to the cardinality (number of elements) of $\mathcal{S}_p$ is given by

$$|\mathcal{S}_p| = \binom{d_c - 1}{p}. \tag{4.13}$$

Equation (4.10) can be rewritten:

$$P^{(\ell)}_{\gamma,+}(k) = \sum_{p \text{ even}} \binom{d_c - 1}{p} \Pr\Big(\min |\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| = k \ \cap \ \mathfrak{s}(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \boldsymbol{\delta_p}\Big). \tag{4.14}$$

Let us now focus on the probability term in the equation. There is no reason to think that the events "$\min |\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| = k$" and "$\mathfrak{s}(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \boldsymbol{\delta_p}$" are independent, so we have to consider the conditional probability:

$$\Pr\Big(\min |\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| = k \ \cap \ \mathfrak{s}(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \boldsymbol{\delta_p}\Big)$$

$$= \Pr\Big(\min |\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| = k \ \Big| \ \mathfrak{s}(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \boldsymbol{\delta_p}\Big) \times \Pr\Big(\mathfrak{s}(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \boldsymbol{\delta_p}\Big). \tag{4.15}$$

We first look at the conditional term. If $\boldsymbol{\delta_p}$ exists, this implies that the first p elements in $\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}$ have negative values, and the $d_c - 1 - p$ following elements have positive values. After taking the magnitude, the first p elements become positive, but with a PMF corresponding to the negative value of $\lambda^{(\ell-1)}$. The last elements keep the PMF related to their positive values.

Let $\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}_p = [-\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}_{p,-}, \boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}_{p,+}]$, with $\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}_{p,-}$ corresponding to the vector composed of the p first $\lambda^{(\ell-1)} < 0$ in $\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}$, and $\boldsymbol{\lambda}^{(\ell-1)}_{p,+}$ corresponding to the vector composed of the $d_c - 1 - p$ last $\lambda^{(\ell-1)} > 0$ in $\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}$. The conditional term in (4.15) can now be rewritten

$$\Pr\Big(\min |\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}| = k \ \Big| \ \mathfrak{s}(\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}) = \boldsymbol{\delta_p}\Big) = \Pr\Big(\min [-\boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}_{p,-}, \boldsymbol{\lambda}^{(\boldsymbol{\ell-1})}_{p,+}] = k\Big), \tag{4.16}$$

$$= \overline{\Phi}^{p}_{\lambda^{(\ell-1)}_-}(k-1)\overline{\Phi}^{d_c-1-p}_{\lambda^{(\ell-1)}_+}(k-1) - \overline{\Phi}^{p}_{\lambda^{(\ell-1)}_-}(k)\overline{\Phi}^{d_c-1-p}_{\lambda^{(\ell-1)}_+}(k). \tag{4.17}$$

$\overline{\Phi}_{\lambda_+^{(\ell-1)}}$ and $\overline{\Phi}_{\lambda_-^{(\ell-1)}}$ correspond to the complementary CDFs of the random variables in $\boldsymbol{\lambda}_{\boldsymbol{p,+}}^{\boldsymbol{(\ell-1)}}$ and $-\boldsymbol{\lambda}_{\boldsymbol{p,-}}^{\boldsymbol{(\ell-1)}}$, respectively, and can be expressed as

$$\overline{\Phi}_{\lambda_+^{(\ell-1)}}(k) = \sum_{l=k+1}^{Q} \Pr\left(\lambda^{(\ell-1)} = l \mid l > 0\right), \tag{4.18}$$

$$= \sum_{l=k+1}^{Q} \frac{P_\lambda^{(\ell-1)}(k)}{\Pr(\lambda^{(\ell-1)} > 0)}, \tag{4.19}$$

$$= \frac{1}{\overline{\Phi}_{\lambda^{(\ell-1)}}(0)} \underbrace{\sum_{l=k+1}^{Q} P_\lambda^{(\ell-1)}(l)}_{A_+(k+1)}, \tag{4.20}$$

and,

$$\overline{\Phi}_{\lambda_-^{(\ell-1)}}(k) = \sum_{l=-Q}^{-(k+1)} \Pr(\lambda^{(\ell-1)} = l \mid l < 0), \tag{4.21}$$

$$= \frac{1}{\Phi_{\lambda^{(\ell-1)}}(-1)} \underbrace{\sum_{l=-Q}^{-(k+1)} P_\lambda^{(\ell-1)}(l)}_{A_-(k+1)}, \tag{4.22}$$

Then, (4.16) becomes

$$\Pr\left(\min|\boldsymbol{\lambda}^{\boldsymbol{(\ell-1)}}| = k \mid \mathfrak{s}(\boldsymbol{\lambda}^{\boldsymbol{(\ell-1)}}) = \boldsymbol{\delta_p}\right) = \frac{A_-^p(k)A_+^{d_c-1-p}(k) - A_-^p(k+1)A_+^{d_c-1-p}(k+1)}{\left(\Phi_{\lambda^{(\ell-1)}}(-1)\right)^p \left(\overline{\Phi}_{\lambda^{(\ell-1)}}(0)\right)^{d_c-1-p}}. \tag{4.23}$$

We calculate the remaining, non-conditional term in (4.15):

$$\Pr\left(\mathfrak{s}(\boldsymbol{\lambda}^{\boldsymbol{(\ell-1)}}) = \boldsymbol{\delta_p}\right) = \Pr\left(\lambda^{(\ell-1)} < 0\right)^p \times \Pr\left(\lambda^{(\ell-1)} > 0\right)^{d_c-1-p}, \tag{4.24}$$

$$= \left(\Phi_{\lambda^{(\ell-1)}}(-1)\right)^p \left(\overline{\Phi}_{\lambda^{(\ell-1)}}(0)\right)^{d_c-1-p}, \tag{4.25}$$

which simplifies with the denominator of (4.23). Combining terms, we have

$$P_{\gamma,+}^{(\ell)}(k) = \Phi_+(k) - \Phi_+(k+1), \tag{4.26}$$

$$\Phi_+(k) = \sum_{p \text{ even}} \binom{d_c - 1}{p} A_-^p(k) A_+^{d_c-1-p}(k). \tag{4.27}$$

A_- and A_+ are respectively the complementary CDF (CCDF) of the negative and positive values of the saturated V2C messages at iteration $\ell - 1$ [56]:

$$A_-(k) = \sum_{x=-Q}^{k} P_\lambda^{(\ell-1)}(x),\ k < 0\,, \tag{4.28}$$

$$A_+(k) = \sum_{x=k}^{Q} P_\lambda^{(\ell-1)}(x),\ k > 0\,. \tag{4.29}$$

The last case to consider is $k < 0$, denoted $P_{\gamma,-}^{(\ell)}(k)$:

$$P_{\gamma,-}^{(\ell)}(k) = \Pr\Big(\min|\boldsymbol{\lambda^{(\ell-1)}}| = -k\ \cap \prod_{\forall i} \mathfrak{s}(\lambda^{(\ell-1)}) = -1\Big). \tag{4.30}$$

The derivation mirrors that of the $P_{\gamma,+}^{(\ell)}(k)$ equation above. This time $\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = -1$, there are an odd number of negative $\lambda^{(\ell-1)}$ values, i.e. $\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) \in \boldsymbol{O}$, where

$$\boldsymbol{O} = \bigcup_{p\ \text{odd}} S_p\,. \tag{4.31}$$

By listing all possible $\mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}})$ combinations and factorizing common terms, (4.30) becomes

$$P_{\gamma,-}^{(\ell)}(k) = \sum_{p\ \text{odd}} \binom{d_c - 1}{p} \Pr\Big(\min|\boldsymbol{\lambda^{(\ell-1)}}| = -k\ \cap\ \mathfrak{s}(\boldsymbol{\lambda^{(\ell-1)}}) = \boldsymbol{\delta_p}\Big). \tag{4.32}$$

Following the same reasoning given for $P_{\gamma,+}^{(\ell)}(k)$ for $k < 0$

$$P_{\gamma,-}^{(\ell)}(k) = \Phi_-(k) - \Phi_-(k-1), \tag{4.33}$$

$$\Phi_-(k) = \sum_{p\ \text{odd}} \binom{d_c - 1}{p} A_-^p(-k) A_+^{d_c-1-p}(-k). \tag{4.34}$$

This translates to the following equation (as in [56]):

$$P_\gamma^{(\ell)}(k) = \begin{cases} \Phi_+(k) - \Phi_+(k+1) & k > 0\,, \\ 1 - \left(1 - P_\lambda^{(\ell-1)}(0)\right)^{d_c-1} & k = 0\,, \\ \Phi_-(k) - \Phi_-(k-1)) & k < 0\,. \end{cases} \tag{4.35}$$

4.1.3 Finite-length Transformation

The final step in our density evolution analysis is to transform the infinite-length result, as this result does not as accurately portray the performance of our code in a real decoder. We perform the transformation shown in [20] to receive a more accurate estimation of the performance of our code for a finite-length code:

$$P_{\gamma,N}^{(\ell)}(p_0) = \int_0^{\frac{1}{2}} P_{\gamma,\infty}^{(\ell)}(p_{ob}) G_{\mathcal{N}}\left(p_{ob}; p_0, \frac{p_0(1-p_0)}{N}\right) dp_{ob}\,. \tag{4.36}$$

Where $G_{\mathcal{N}}(x;\mu,\sigma^2)$ is the probability density function of a normal Gaussian random variable with mean μ and variance σ^2, $P_{\gamma,\infty}^{(\ell)}(p_{ob})$ is the message PDF for an infinite code at iteration ℓ, and $P_{\gamma,N}^{(\ell)}(p_0)$ is the message PDF for a finite code of length N at iteration ℓ. This transformation greatly improves our estimation of the performance of a decoder.

4.2 Proposed SM offset optimization using DE

Instead of directly solving (3.2), we propose to find the offset $\omega_{i,j}^{(\ell)}$ that minimizes the mean square error (MSE) $\epsilon(\omega_{i,j}^{(\ell)})$ between the C2V messages obtained respectively through the single-minimum (SM) and min-sum (MS) message update rules, for a given edge $i \to j$ and iteration ℓ. This translates to the following optimization problem:

$$\arg\min_{\omega_{i,j}^{(\ell)}} \epsilon(\omega_{i,j}^{(\ell)}) = \arg\min_{\omega_{i,j}^{(\ell)}} \mathrm{E}\left|\gamma_{e,j\to i}^{(\ell)} - \gamma_{\bar{e},j\to i}^{(\ell)}\right|^2. \tag{4.37}$$

This method does not guarantee finding the optimal solution of problem (3.2), but its main advantage is that offsets can be independently optimized for each edge, and, more importantly, analytical methods can be applied to solve this problem. Recalling that the SM C2V message update rule can be formulated as in (2.22), the MSE term can be expressed as

$$\epsilon(\omega_{i,j}^{(\ell)}) = \sum_{\substack{(q,v)\\ |q|<|v|}} \left(\omega_{i,j}^{(\ell)} + |q| - |v|\right)^2 \underbrace{P\left(\lambda_{i\to j}^{(\ell-1)} = q \;\cap\; \gamma_{\bar{e},j\to i}^{(\ell)} = v\right)}_{\theta_{i,j}^{(\ell)}(q,v)}, \tag{4.38}$$

with $(q,v) \in [-Q,Q]^2$ and $\theta_{i,j}^{(\ell)}(q,v)$ being the joint probability mass function of the $(\lambda_{i\to j}^{(\ell-1)}, \gamma_{\bar{e},j\to i}^{(\ell)})$ random variables. If these variables are independent, we have $\theta_{i,j}^{(\ell)}(q,v) = P_{\lambda}^{(\ell-1)}(q) \times P_{\gamma_{\bar{e}}}^{(\ell)}(v)$. The message dependencies will be further discussed in the next sub-section.

Problem (4.37) is convex, having solution:

$$\frac{d\epsilon(\bar{\omega}_{i,j}^{(\ell)})}{d\bar{\omega}_{i,j}^{(\ell)}} = 2 \sum_{\substack{(q,v) \\ |q|<|v|}} \Big(\bar{\omega}_{i,j}^{(\ell)} + |q| - |v|\Big)\theta_{i,j}^{(\ell)}(q,v) = 0\,, \tag{4.39}$$

$$\implies \bar{\omega}_{i,j}^{(\ell)} = \frac{\sum_{\substack{(q,v) \\ |q|<|v|}} \Big(|v| - |q|\Big)\theta_{i,j}^{(\ell)}(q,v)}{\sum_{\substack{(q,v) \\ |q|<|v|}} \theta_{i,j}^{(\ell)}(q,v)}, \tag{4.40}$$

where $\bar{\omega}_{i,j}^{(\ell)}$ is the optimal real-valued SM offset, and $\omega_{i,j}^{(\ell)} = \lfloor \bar{\omega}_{i,j}^{(\ell)} + 1/2 \rfloor$.

4.2.1 On the message dependencies

DE equations are derived based on the fact that all messages are independent, a consequence of the extrinsic exclusion rule and the cycle-free assumption. However, when the γ-update rule includes the extrinsic message, the message independence assumption no longer holds. To illustrate this, we study how messages propagate while being exchanged with the first VN in a cycle-free code. For ease of notation, we consider that this VN is connected to the first d_v CNs in the Tanner graph. By noting that, in a SM decoder without offset compensation, $\gamma_{e,j\to i}^{(\ell)} = \mathfrak{s}(\lambda_{i\to j}^{(\ell-1)}) \times f([\lambda_{i\to j}^{(\ell-1)}, \gamma_{\bar{e},j\to i}^{(\ell)}])$, we have

$$\lambda_{1\to i}^{(\ell)} = L_1 + \sum_{\substack{j=1 \\ i\neq j}}^{d_v} \underbrace{\mathfrak{s}(\lambda_{1\to j}^{(\ell-1)}) f\Big([\lambda_{1\to j}^{(\ell-1)}, \gamma_{\bar{e},j\to 1}^{(\ell)}]\Big)}_{\gamma_{e,j\to 1}^{(\ell)}}\,. \tag{4.41}$$

Through recursion, all $\lambda_{1\to j}^{(\ell-1)}$ variables are correlated with $\gamma_{\bar{e},j\to 1}^{(\ell-1)}$ and L_1. Consequently, the $\gamma_{e,j\to 1}^{(\ell)}$ variables are not independent, and (4.3) cannot be applied to derive $P_{\lambda}^{(\ell)}$. Performing DE for the SM decoding algorithm requires conditioning the message PMFs on each message value in the computation tree. Since the number of possibilities grows exponentially with ℓ, the number of message bits and check node degree, this method is computationally expensive.

4.2.2 Optimization for the first iterations

When $\ell \in [1,2]$, (4.35) is valid and can be used to derive $P^{(\ell)}_{\gamma_{\bar{e}}}$. Furthermore, with $\lambda^{(1)}_{i\to j}$ and $\gamma^{(2)}_{\bar{e},i\to j}$ being independent, $\theta_{i,j}$ can be easily calculated and the optimal offsets can be deduced for the first two iterations through (4.39). However, $P^{(1)}_{\lambda}$ cannot be obtained with (4.3) since $\gamma^{(1)}_{e,j\to i}$ is correlated with L_i, as shown in (4.41). Instead, we have

$$P^{(1)}_{\lambda}(k) = \sum_{l=-Q}^{Q} P^{(0)}_{\lambda}(l)\Big(\circledast_{d_v-1} \boldsymbol{P}^{(\mathbf{1})}_{\boldsymbol{\gamma_e|L}}\Big)(k-l), \tag{4.42}$$

where $\boldsymbol{P}^{(\mathbf{1})}_{\boldsymbol{\gamma_e|L}} = [P^{(1)}_{\gamma_e|L}(-Q|l), \dots, P^{(1)}_{\gamma_e|L}(Q|l)]$ is the PMF vector of the SM C2V messages when the extrinsic message value (corresponding to the channel LLR) is l:

$$P^{(1)}_{\gamma_e|L}(k|l) = \begin{cases} P^{(1)}_{\gamma_{\bar{e}}}(k) & |k| \le |l|, \\ \sum_{m=|l|+1}^{Q} P^{(1)}_{\gamma_{\bar{e}}}\big(\mathfrak{s}(k)m\big) & |k| = \min\Big(|l| + \omega^{(\ell)}_{i,j}, Q\Big), \\ 0 & \text{otherwise.} \end{cases} \tag{4.43}$$

Extending DE results to each subsequent iteration becomes exponentially more complex. Therefore, the offset of the remaining iterations $\ell > 2$ are obtained by linearly extrapolating the real-valued offsets deduced by the method described in this section for the first 2 iterations: $\bar{\omega}^{(\ell)}_{i,j} = \left(\bar{\omega}^{(2)}_{i,j} - \bar{\omega}^{(1)}_{i,j}\right) \times (\ell - 1) + \bar{\omega}^{(1)}_{i,j}$.

The DE optimization method outlined here is an analytical approach. The execution time of this method will not depend on the E_b/N_0 analyzed, whereas the Monte Carlo-based approaches have execution times which scale with noise power.

CHAPTER 5 Results and Discussion

This section presents the BER performance results for a number of SM decoders. We start with a presentation of preliminary results gathered for RV and RVQ decoders. We then show the final results for discrete optimized SM offset decoders, for both a regular and irregular LDPC code. For both paradigms, a description of the simulation setup precedes the figures and discussion of BER performance. The solution values for the SM offset vectors may be found in Appendix B.

5.1 Real-Valued Decoding

5.1.1 Simulation Setup

To validate the relaxed-constraint, real-valued optimization methods, MC simulation are performed to measure the error correction performance for a (1723,2048)-regular LDPC code from the 802.3an-2006 10GE standard [14]. Message values for the 10GE decoders are passed on a flooding schedule for a maximum of 25 iterations. The four search algorithms outlined in Section 3.3 are: *SQP, Interior-Point, Active-Set and Trust-Region-Reflective*. The algorithms are implemented using the MATLAB function `fmincon`. The RV SM offsets are bounded by 0 and $\omega_{\max} = 4.0$. The maximum iteration number for all algorithms with the exception of *Interior-Point* is 400, and for *Interior-Point* it is 1000. The step tolerance for all algorithms with the exception of *Interior-Point* is 10^{-6}, and for *Interior-Point* it is 10^{-10}. The optimality tolerance of all algorithms is 10^{-6}.

SM offset solutions for these four algorithms are rendered for normalized signal-to-noise ratio $\zeta = 3.8$ dB, as this E_b/N_0 value falls well within the waterfall region for these decoders. These are then compared to optimized normalized MS and fixed SM offset decoding methods. The best-performing among these solutions is then chosen to evaluate the effect of quantization on real-valued solutions - by quantizing the solution on different numbers of bits and evaluating the performance of the resultant quantized decoding method.

5.1.2 Results

Figure 5.1 depicts the performance of a real-valued MS decoder against that of the solutions rendered by the real-valued search methods, as well as an optimized fixed SM offset decoder. The SM fixed offset decoder has an offset value of 5. Parameters in the legend correspond to (η, ω_{SM}). We can see that the solutions rendered by the four search algorithms perform

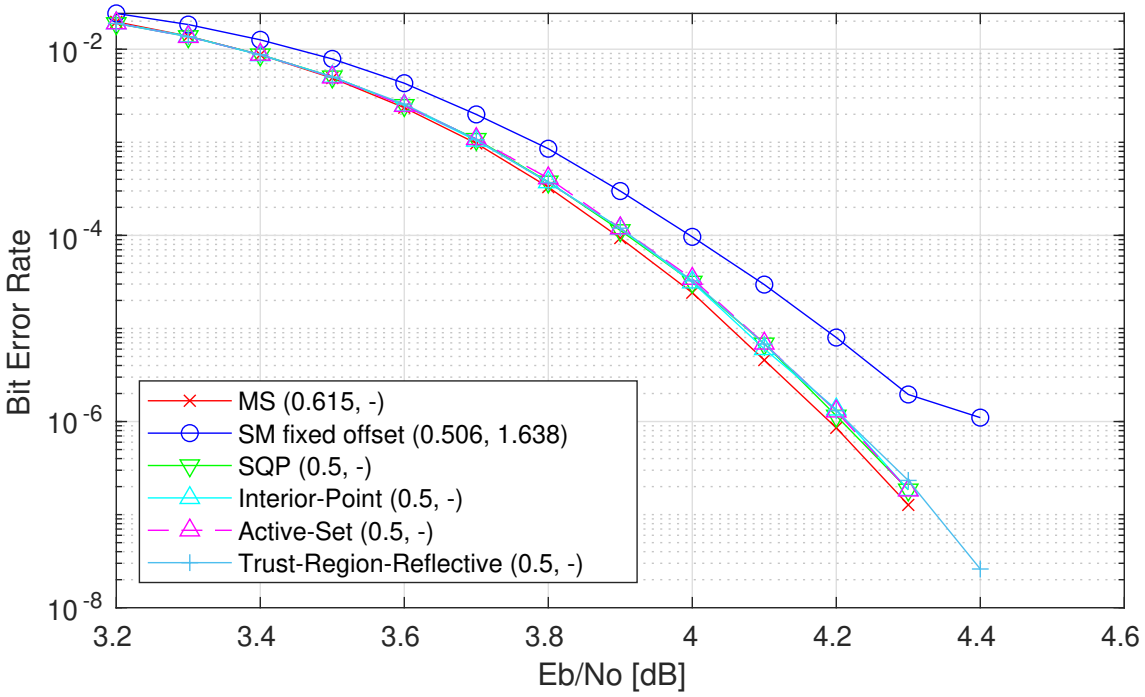

Figure 5.1 Results depicting the performance of real-valued SM decoders against real-valued MS decoders

almost identically to the MS decoder. The best BER result at $\zeta = 3.8$ dB of the real-valued solutions is rendered by the *Active-Set* algorithm. Appendix B shows the results for the real-valued optimizations.

We must now evaluate whether there exists a practical bit-precision for quantization of these decoders which preserves the RV performance. Figure 5.2 depicts the BER performance of the *Active-Set* solution, using 5, 7 and 10-bit quantization schemes. Parameters in the legend correspond to (α, η).

We can see that there is a diminishing return with respect to increasing bit-precision for SM decoders for the 10GE code, and that the level of performance shown to be possible for RV decoding cannot be recovered by the RVQ decoder. However, this solution was outperformed by the discrete SM offset optimization methods described in Sections 3.4 and 3.5, and therefore was not applied to 5G codes. Appendix B shows the results for the RVQ decoders at different bit-precisions.

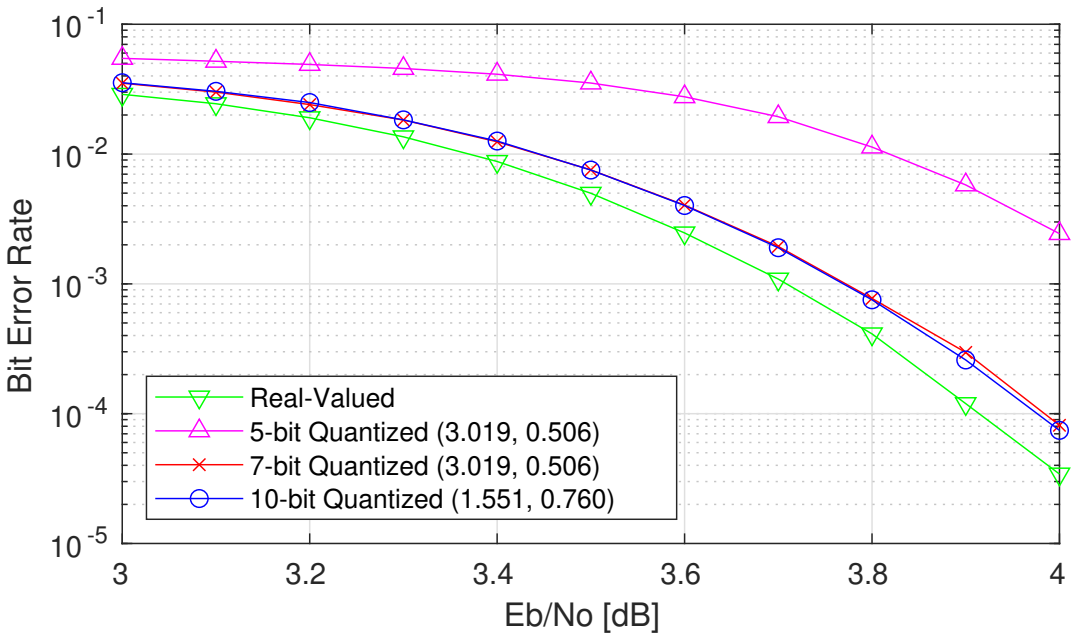

Figure 5.2 *Active-Set* RVQ results for 5-, 7- and 10-bit quantization schemes

5.2 Discrete SM Offset Optimization Methods

5.2.1 Simulation setup

To validate the more practical optimization methods, MC simulations are again performed to measure the error correction performance for two different codes: i) a (1723,2048)-regular LDPC code from the 802.3an-2006 10GE standard [14]; ii) a 5G code with base graph index 1, code length $N = 16128$, lifting size $Z = 384$ and code rate 0.5238 [13]. For all MC searches, at least 1000 frames were measured in error before stopping. Message values for the 10GE decoders are quantized on 7 bits and passed on a flooding schedule for a maximum of 40 iterations. For the 5G decoders, messages are quantized on 6 bits, and passed on a flooding schedule for a maximum of 40 iterations. Different precisions were used for comparison with state of the art methods.

We propose comparing four offset optimization methods for the SM decoder. All methods use CD as described in Section 3.2 to optimize $(\eta, \alpha, \mathbf{\Omega_G})$ parameters. These parameters, and subsequently the SM offset vectors determined by the stochastic searches, are optimized for $\alpha_{\max} = 10$, $\omega_{\max} = 10$ and $\eta \in [0.25, 0.5, 0.75, 1]$ to ease hardware implementation. The differences between the methods resides in how (3.2) is solved. The first method (M1), called *fixed offset*, only optimizes a single offset ω following approach (i) described in Section 3.1. The second method (M2) optimizes $\mathbf{\Omega_G}$ following approach (iii), where (3.2) is solved using

genetic algorithm [57] for the regular 10GE code. For the 5G codes, SM offsets are selected based on the extended variable weight (EVW) method reported in [34], then (η, α) are optimized using CD. This method is here referred to as CD-EVW. The third method (M3) is the proposed WSA performed with $L = 3$, $p = 1$, with (3.23) solved through exhaustive search for the 10GE code and genetic algorithm for the 5G code using the following group setup: $G_1 = \{3\}$, $G_2 = \{6, 7, 8, 9, 10\}$ and $G_3 = \{19\}$. The last method (M4) corresponds to the DE-based optimization proposed in Section 4.

For the 10GE code, problem (3.2) is solved when considering 25 decoding iterations. The remaining iterations use the final SM offset gathered during the optimization procedure. For the 5G code, WSA is applied for 10 decoding iterations, and the modified Akima method was performed to extrapolate SM offsets from 10 to 40. For all instances where the genetic algorithm is used, the parameters are set to a population of 200 individuals, including 10 elites, from which 152 are selected for crossover. The maximum generations I_{lim} is 12000 and $I_{\text{stall}} = 50$. The time limit and maximum stall times are both undefined; $t_{lim} = t_{\text{stall}} = \infty$. The fitness limit $\rho_{lim} = -\infty$. All generation, crossover and mutation functions are restricted to integer solutions.

5.2.2 Results for the 10GE code

Figure 5.3 shows the BER performance for the regular 10GE code when considering the four offset optimization methods for the SM decoder. BER results for the MS decoder are also shown as reference. Triples in the legend correspond to (α, η, ω_{SM}). All parameters are optimized for $\zeta = 3.8$ dB for all codes and methods, indicated by the red vertical line in the figure. This E_b/N_0 value again falls within the waterfall region for these decoders. Appendix B shows the results for (M2)-(M4) for the 10GE code.

We measured the execution times of each method on the same computer (at $\zeta = 3.8$ dB), and obtained the following results: (M1) 14min; (M2) 20h; (M3) 314h; (M4) 0.7s. The DE method (M4) is by far the fastest method. Notably, the compute time is constant for any ζ value, whereas MC approaches require more simulation trials with decreasing noise power. This opens the possibility to re-design the offsets "on-the-fly", when the decoding parameters change.

The results for the stochastic searches show that the difference in performance between optimized varying offset methods (M2) and (M3) is negligible. However, for $\zeta > 4$ dB, the fixed offset method is outperformed by both (M2) and (M3), with a gap of 0.1 dB observed at 10^{-5} BER. We can see also that the difference in performance between methods (M2) and (M3) is very small when compared to the standard MS decoder. Concerning the

DE method (M4), we observe a performance loss compared to other optimization methods. This is likely due to the fact that only SM offsets of the first two iterations are optimized, while we approximate the remaining ones by linear extrapolation. However, this method has significantly lower execution time compared with (M2) and (M3), with more than 1e 6× speedup.

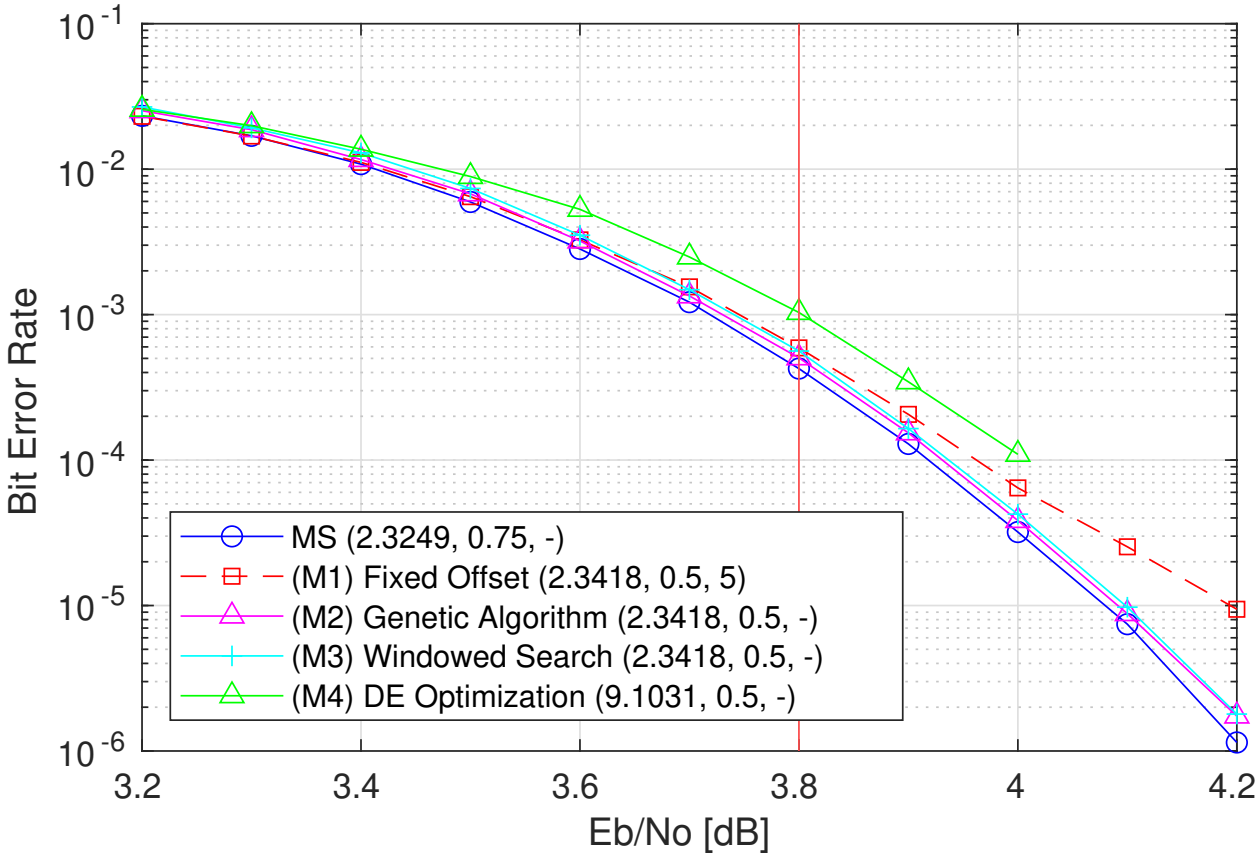

Figure 5.3 Comparison of BER performance for SM decoders against optimized MS for the 802.3an-2006 code

5.2.3 Results for the 5G code

Figure 5.4 depicts the BER of SM decoders for the 5G code; performed using a 5G code BG1 [N=16128, R=0.5238]. For the MS decoder, we have the MS offset $\beta = 1$, applied at each iteration. Parameters are optimized at the following E_b/N_o values: $\zeta = 1.1$ dB for the MS decoder (blue vertical line), $\zeta = 2.8$ dB for the SM fixed offset method (red vertical line) and $\zeta = 2$ dB for the CD-EVW, WSA and DE-based methods (green vertical line). CD-EVW is a CD-optimized version of the SM optimization employed in [34]. Triples in the legend

correspond to (α, η, ω_{SM}). These placements are based on the projected waterfall regions for each decoder, while the fixed offset decoder was observed to perform best when optimized within the floor region. Appendix B shows the results for (M2)-(M4) for the 5G code.

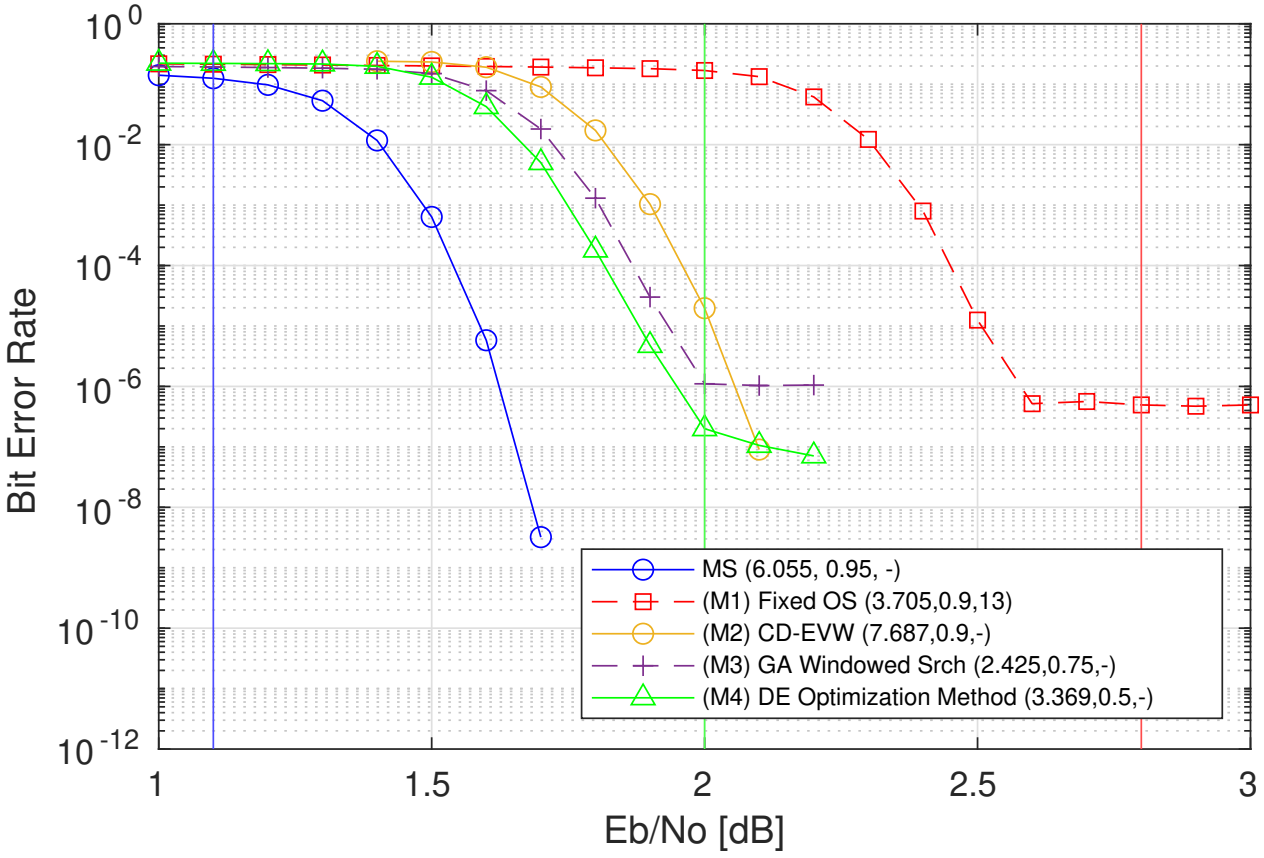

Figure 5.4 Comparison of SM decoding methods against an MS decoder

There is a significant gain in error performance for varying versus fixed offset SM decoding in the 5G code presented. The SM offsets obtained with WSA outperforms the ones obtained using CD-EVW by $0.1 - 0.2$ dB in the Waterfall region. Furthermore, contrary to the 10GE code, the DE analysis method (M4) significantly outperforms both CD-EVW and WSA. This can be explained by the fact that SM offsets are optimized separately for each edge of the base graph. This additional degree of freedom provides gains since the 5G code has several edge types and CN/VN degrees. It should be noted that further performance gains for method (M4) could be achieved by improving the DE analysis to optimize the SM offsets for all iterations, instead of extrapolating them. In addition, a significant advantage in time resources used is observed for the DE optimization method. The WSA method required hundreds of hours of compute time, whereas the DE-based method requires on average 0.6 s,

using the same simulation framework and the same machines. Since DE optimization is both the highest performance and lowest cost algorithm to implement, time complexity analysis was not performed for MC approaches. Consequently, DE-based optimization is the most efficient method for the 5G SM decoder.

CHAPTER 6 CONCLUSION AND RECOMMENDATIONS

6.1 Summary

In this research we have explored the problem of optimizing second-minimum emulation offset values for SM decoders. These decoders require less hardware resources than typical MS decoders, but their performance depends heavily on well-optimized SM offsets. [4] suggests that an SM decoder requires a check node processor roughly 48% the size of a standard MS check node processor, suggesting SM decoders are much more efficient to implement than a standard MS decoder. We first presented a general formulation of the problem and proposed a simplification with the coordinate descent algorithm. We then solved for offset values by proposing two methods: 1) a Monte Carlo-based method referred to as WSA 2) A DE-based analytical method. We evaluate these proposed methods using two different codes (from the 5G NR and 10Gbps Ethernet standards) and compare with existing optimization methods. We additionally attempted to solve a relaxed-constraint version of the problem to infer a solution to the original problem. However, these RVQ decoders cannot match the performance of either the WSA or the DE optimization techniques. We show that, for the 5G code, large BER gains are observed, particularly for the DE method, which also requires less computation time. This method optimizes the SM offsets for only the first two iterations, while the remaining ones are extrapolated. Therefore, we believe that the performance gap between MS and SM decoders can be further reduced. This encourages further investigation to improve DE analysis for SM decoders.

6.2 Future Research

The most important next steps will be:

- Applying the proposed optimization methods for each E_b/N_0 value simulated. This will better validate the behavior of offset values, and should ensure even better performance.
- Extending DE analysis for SM decoders by another iteration, thus allowing for DE optimization with more initial information, and allowing for more robust extrapolation for later iterations. If DE analysis for SM decoders were extended to enough iterations, it would be interesting to explore the feasibility of DE optimization of regular codes.

As stated in Section 4.2.1, DE analysis for SM decoding becomes exponentially more complex as the number of decoding iterations increases, and therefore this second step would likely

be costly to implement using current methods. On the other hand, alternative optimization methods based on Monte-Carlo simulation are also fairly computation intensive. One possibility for future research would be to qualify the tradeoff between the complexity of DE SM offset optimization and the practicality of MC-based approaches.

The searches proposed in this research are done offline at design time, for a single normalized signal-to-noise ratio. The offsets found are stored in a look-up table (LUT), then applied for multiple noise powers in Chapter 5, suggesting that re-designation of parameters for minor changes in E_b/N_0 does not seem to be necessary to achieve practical performance. However, considering a channel subject to, for instance, user mobility, it is possible that noise power could significantly change during normal operation, leading to a performance degradation. The SM decoder performance could be improved by designing SM offsets for multiple target E_b/N_0 values. Practically, this would mean for any searches performed at design time, many LUTs would be required to store the integer offsets for multiple noise power targets. Therefore, we expect a tradeoff between resolution of noise powers simulated - and therefore performance - and hardware complexity. However, as mentioned in Section 5.2.2, the low execution time of the DE-based optimization method opens the possibility for "on-the-fly" design of SM offsets. In this case, a single reconfigurable LUT could be used instead.

REFERENCES

[1] F. Leduc-Primeau, F. R. Kschischang, and W. J. Gross, "Modeling and energy optimization of ldpc decoder circuits with timing violations," *IEEE Transactions on Communications*, vol. 66, no. 3, pp. 932–946, Mar. 2018.

[2] J. Nadal, M. Fiorentino, E. Dupraz, and F. Leduc-Primeau, "A deeply pipelined, highly parallel and flexible ldpc decoder," in *2020 18th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2020, pp. 263–266.

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

[4] A. Darabiha, A. Carusone, and F. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *2006 IEEE International Symposium on Circuits and Systems*, May 2006.

[5] D. Mackay and R. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, Aug. 2002.

[6] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[7] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding ldpc codes," *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270)*, vol. 2, pp. 1036–1036E vol.2, Nov. 2001.

[8] C. Jones, E. Valles, M. Smith, and J. Villasenor, "Approximate-min constraint node updating for LDPC code decoding," in *IEEE Military Communications Conference, 2003. MILCOM 2003.*, vol. 1, Oct. 2003, pp. 157–162 Vol.1.

[9] M. Viens and W. E. Ryan, "A reduced-complexity box-plus decoder for ldpc codes," in *2008 5th International Symposium on Turbo Codes and Related Topics*, Sept. 2008, pp. 151–156.

[10] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[11] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[12] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[13] 3GPP TS 38.212, "3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Multiplexing and channel coding," Aug. 2021.

[14] IEEE, 802.3an, " Standard for Information Technology – LAN/MAN - CSMA/CD Access Method and Physical Layer Specifications - Parameters for 10 Gb/s Operation, Type 10GBASE-T ," pp. 1–181, Sept. 2006.

[15] M. Eroz, F.-W. Sun, and L.-N. Lee, "DVB-S2 low density parity check codes with near shannon limit performance," *International Journal of Satellite Communications and Networking*, vol. 22, no. 3, pp. 269–279, June 2004.

[16] Y. Yuan, Y. Zhao, and B. Zong, "Potential key technologies for 6g mobile communications," *Science China Information Sciences*, no. 183301, May 2020.

[17] K. Zhu and Z. Wu, "Comprehensive study on CC-LDPC, BC-LDPC and polar code," in *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, Apr. 2020, pp. 1–6.

[18] J. Nadal and A. Baghdadi, "FPGA based design and prototyping of efficient 5G QC-LDPC channel decoding," in *2020 International Workshop on Rapid System Prototyping (RSP)*, Sept. 2020.

[19] I. F. Akyildiz, A. Kak, and S. Nie, "6G and beyond: The future of wireless communications systems," *IEEE Access*, vol. 8, July 2020.

[20] F. Leduc-Primeau and W. J. Gross, "Finite-length quasi-synchronous LDPC decoders," in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Sept. 2016, pp. 325–329.

[21] R. Yazdani and M. Ardakani, "Waterfall performance analysis of finite-length LDPC codes on symmetric channels," *IEEE Trans. on Communications*, vol. 57, no. 11, pp. 3183–3187, Nov. 2009.

[22] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[23] T. Etzion, A. Trachtenberg, and A. Vardy, "Which codes have cycle-free tanner graphs?" *Information Theory, IEEE Transactions on*, vol. 45, pp. 2173 – 2181, Sept. 1999.

[24] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[25] A. A. Emran and M. Elsabrouty, "Simplified variable-scaled min sum ldpc decoder for irregular ldpc codes," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, Jan. 2014, pp. 518–523.

[26] H. Hatami, D. G. M. Mitchell, D. J. Costello, and T. Fuja, "A modified min-sum algorithm for quantized ldpc decoders," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2434–2438.

[27] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[28] E. Sharon, S. Litsyn, and J. Goldberger, "Convergence analysis of serial message-passing schedules for ldpc decoding," in *4th International Symposium on Turbo Codes Related Topics; 6th International ITG-Conference on Source and Channel Coding*, Apr. 2006, pp. 1–6.

[29] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Lin, "Flexible LDPC decoder design for multigigabit-per-second applications," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 116–124, Mar. 2009.

[30] F. Yi and P. Wang, "Low complexity decoding algorithm of QC-LDPC code," in *2010 IEEE Asia-Pacific Services Computing Conference*, Dec. 2010, pp. 531–534.

[31] F. Angarita, J. Valls, V. Almenar, and V. Torres, "Reduced-complexity min-sum algorithm for decoding LDPC codes with low error-floor," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 61, no. 7, pp. 2150–2158, Feb. 2014.

[32] D. B. Dermont, J. Nadal, and F. Leduc-Primeau, "Single-minimum ldpc decoding offset optimization methods," in *2022 17th Canadian Workshop on Information Theory (CWIT)*, June 2022, [to appear].

[33] S. Hemati, F. Leduc-Primeau, and W. J. Gross, "A relaxed min-sum LDPC decoder with simplified check nodes," *IEEE Communications Letters*, vol. 20, no. 3, pp. 422–425, Jan. 2016.

[34] V. L. Petrović and D. M. El Mezeni, "Reduced-complexity offset min-sum based layered decoding for 5G LDPC codes," in *2020 28th Telecommunications Forum (TELFOR)*, Nov. 2020, pp. 1–4.

[35] H. Akima, "A new method of interpolation and smooth curve fitting based on local procedures," *J. ACM*, vol. 17, no. 4, p. 589–602, Oct. 1970.

[36] H. Akima, "A method of bivariate interpolation and smooth surface fitting based on local procedures," *Commun. ACM*, vol. 17, no. 1, p. 18–20, Jan. 1974.

[37] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, Mar. 2015.

[38] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, "Computer methods for mathematical computations," *ZAMM - Journal of Applied Mathematics and Mechanics*, vol. 59, no. 2, pp. 141–142, 1979.

[39] R. P. Brent, *Algorithms for Minimization without Derivatives*, 1st ed. Englewood Cliffs, New Jersey: Prentice-Hall, 1973.

[40] MathWorks. (2022) Constrained nonlinear optimization algorithms. [Online]. Available: https://www.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html

[41] T. F. Coleman and Y. Li, "An interior trust region approach for nonlinear minimization subject to bounds," *SIAM J. Optim.*, vol. 6, pp. 418–445, May 1996.

[42] T. Coleman and Y. li, "On the convergence of reflective newton methods for large-scale nonlinear minimization subject to bounds," *Math. Program.*, vol. 67, pp. 189–224, Oct. 1994.

[43] R. Fletcher and M. J. D. Powell, "A Rapidly Convergent Descent Method for Minimization," *The Computer Journal*, vol. 6, no. 2, pp. 163–168, Aug. 1963.

[44] D. Goldfarb, "A family of variable-metric methods derived by variational means," *Mathematics of Computation*, vol. 24, pp. 23–26, Jan. 1970.

[45] P. E. Gill, W. Murray, and M. H. Wright, *Practical optimization*. London: Academic Press, Jan. 1981.

[46] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, July 2006.

[47] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A Trust Region Method Based on Interior Point Techniques for Nonlinear Programming," INRIA, Research Report RR-2896, Nov. 1996, projet PROMATH.

[48] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An interior point algorithm for large-scale nonlinear programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, Sept. 1999.

[49] R. Waltz, J. Morales, J. Nocedal, and D. Orban, "An interior algorithm for nonlinear optimization that combines line search and trust region steps," *Mathematical Programming*, vol. 107, no. 3, pp. 391–408, July 2006.

[50] J. J. Moré and D. C. Sorensen, "Computing a trust region step," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 3, pp. 553–572, Jan. 1983.

[51] M. A. Branch, T. F. Coleman, and Y. Li, "A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems," *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 1–23, Sept. 1999.

[52] R. H. Byrd, R. B. Schnabel, and G. A. Shultz, "Approximate solution of the trust region problem by minimization over two-dimensional subspaces," *Mathematical Programming*, vol. 40, no. 1, pp. 247–263, Jan. 1988.

[53] S. P. Han, "A globally convergent method for nonlinear programming," *Journal of Optimization Theory and Applications*, vol. 22, no. 3, pp. 297–309, June 1977.

[54] M. J. D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical Analysis*, G. A. Watson, Ed., Berlin, Heidelberg, June 1978, pp. 144–157.

[55] MathWorks. (2022) How the genetic algorithm works. [Online]. Available: https://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html

[56] A. Balatsoukas-Stimming and A. Burg, "Density evolution for min-sum decoding of LDPC codes under unreliable message storage," *IEEE Communications Letters*, vol. 18, no. 5, pp. 849–852, Apr. 2014.

[57] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, June 1994.

APPENDIX A PROBABILITY REVIEW

In this section, we review some characteristics of random variables that must be understood to develop the DE analysis tool.

Calculating the probability of the minimum of N random variables

Let $\boldsymbol{X}$ be a vector composed of N discrete-valued random variables, given $\boldsymbol{X} = [X_0, ..., X_{N-1}]$, with X_i being integer-valued, and $X_i \in [0, Q] \forall i$. We consider $Y = \min \boldsymbol{X}$, where Y takes the minimum value from the random variables in the vector $\boldsymbol{X}$. We want to calculate the PMF of Y, i.e. the probability that Y is equal to k, denoted $P_Y(k) \triangleq \Pr(Y = k)$. It is assumed that each of the random variables in X are independent, but can have different distributions, i.e. $P_{X_0}(k) \neq P_{X_1}(k) \neq P_{X_2}(k), ...$ We also denote the CDF $\phi_Y(k) \triangleq \Pr(Y \leq k)$ and $\Phi_{X_i}(k) \triangleq \Pr(X_i \leq k)$. The CMF $\Phi_Y(k) = \Pr(\min \boldsymbol{X} \leq k)$ implies that at least one X_i in $\boldsymbol{X}$ is smaller than k. The probability that at least one X_i is smaller than k is equivalent to one minus the probability that all X_i are greater than k, i.e. $\Phi_Y(k) = 1 - \Pr(X_0 > k \cap X_1 > k \cap ... \cap X_{N-1} > k)$. Since X_i is independent for all i, then the probability that X_i is greater than k is

$$\Pr(X_0 > k \cap X_1 > k \cap ... \cap X_{N-1} > k) = \prod_{i=0}^{N-1} \Pr(X_i > k), \tag{A.1}$$

$$= \prod_{i=0}^{N-1} \Big(1 - \Pr(X_i \leq k)\Big), \tag{A.2}$$

$$= \prod_{i=0}^{N-1} \overline{\Phi}_{X_i}(k). \tag{A.3}$$

Since $\overline{\Phi}_{X_i}(k) = 1 - \Phi_{X_i}(k)$, we have

$$\Phi_Y(k) = 1 - \prod_{i=0}^{N-1} \overline{\Phi}_{X_i}(k), \tag{A.4}$$

$$\overline{\Phi_Y}(k) = \prod_{i=0}^{N-1} \overline{\Phi}_{X_i}(k), \tag{A.5}$$

and

$$P_Y(k) = \begin{cases} \Phi_Y(0) & k = 0\,, \\ \Phi_Y(k) - \Phi_Y(k-1) & k > 0\,. \end{cases} \tag{A.6}$$

The above equation can also be rewritten as follows:

$$P_Y(k) = \begin{cases} 1 - \overline{\Phi}_Y(0) & k = 0\,, \\ \overline{\Phi}_Y(k-1) - \overline{\Phi}_Y(k) & k > 0\,. \end{cases} \tag{A.7}$$

Calculating the probability of the sum of N random variables

Let us consider a vector $\boldsymbol{X}$ composed of N random discrete valued variables $\boldsymbol{X} = [X_1, ..., X_{N-1}]$. It is assumed that each random variable is independent, but can have different PMFs, each denoted $P_{X_i}(k)$. We also have $X_i \in [-Q, Q], \forall i$, where $-Q = \min \boldsymbol{X}$ and $Q = \max \boldsymbol{X}$ are the minimum and maximum representable values for all variables X_i. Therefore, we have $P_{X_i}(k) = 0, \forall k \notin [-Q, Q]$.

We have $Y = \sum_{i=0}^{N-1} X_i$, where Y corresponds to the sum of each variable X_i. We want to calculate the PMF of Y, i.e. the probability that Y equals k, denoted $P_Y(k) \triangleq \Pr(Y = k)$. We first consider the partial sum $Y_n = \sum_{i=0}^{n} X_i = X_n + Y_{n-1}$. The PMF of the partial sum Y_n can be written as follows:

$$P_{Y_n}(k) = \Pr(X_n + Y_{n-1} = k) = \Pr\left(\bigcup_{q=-Q}^{Q} (X_n = q \ \cap \ Y_{n-1} = k - q) \right), \tag{A.8}$$

$$= \sum_{q=-Q}^{Q} \Pr(X_n = q \cap Y_{n-1} = k - q), \tag{A.9}$$

$$= \sum_{n=-Q}^{Q} P_{X_n}(q) P_{Y_{n-1}}(k - q). \tag{A.10}$$

Note that (A.10) corresponds to a convolution between PMFs vector $\boldsymbol{P_{X_n}} = [P_{X_n}(-Q), ..., P_{X_n}(Q)]$ and $\boldsymbol{P_{Y_{n-1}}} = [P_{Y_{n-1}}(-Q), ..., P_{Y_{n-1}}(Q)]$. Denoting $\circledast$ the convolution operator, we then have $\boldsymbol{P_{Y_n}} = \boldsymbol{P_{X_n}} \circledast \boldsymbol{P_{Y_{n-1}}}$.

The PMF $P_Y(k) = P_{Y_{N-1}}(k)$ can be obtained through recurrence by successively calculating $P_{Y_1}(k)$, $P_{Y_2}(k)$, ..., using (A.8). This corresponds of successively performing convolution for each variable X_n.

In vector format, the PMF vector $\boldsymbol{P_Y}$ can be then written as

$$\boldsymbol{P_Y} = \circledast_{n=0}^{N-1} \boldsymbol{P_{X_n}}. \tag{A.11}$$

APPENDIX B SM OFFSET VALUES

For all SM offset vectors excluding DE optimized SM decoders B and B, entries are shown in the boxes beside the iteration number for which they are applied. CN groups are displayed in headings where applicable. For DE optimized SM decoders, histograms depicting the occurence of SM offsets at iterations 1, 10, 20, 30, and 40 are shown.

RV Decoders (25 Flooding Iterations)

Figure 5.1 SQP

1. 1.63758701248424
2. 1.63758700655411
3. 1.63758700877995
4. 1.63758701263943
5. 1.63758701132231
6. 1.63758701096495
7. 1.6375870124679
8. 1.63758700619471
9. 1.63758700378508
10. 1.63758701002152
11. 1.63758701433842
12. 1.63758701572294
13. 1.63758701183078
14. 1.63758701112423
15. 1.63758701473254
16. 1.63758701006236
17. 1.63758700928638
18. 1.63758701475909
19. 1.63758701506131
20. 1.63758701360328
21. 1.63758701417914
22. 1.6375870130601
23. 1.63758700583939
24. 1.63758701342767
25. 1.6375870102053

Figure 5.1 Interior-Point

1. 2.16915594194178
2. 2.19455967008582
3. 2.1891458774703
4. 1.42551357590406
5. 1.45075251874128
6. 2.21704957037426
7. 1.4732881378088
8. 2.22478190421139
9. 2.19617003711467
10. 1.96651992348663
11. 1.44959244931687
12. 2.19792760667628
13. 2.21924533087449
14. 2.21022397407853
15. 1.43833137168987
16. 2.20124445436116
17. 2.2143352627657
18. 2.23743913653796

19. 2.23529367876001
20. 1.43236481330027
21. 1.43217688813485
22. 2.22091345423594
23. 2.21129421991636
24. 2.23792925963835
25. 1.45890103500736

Figure 5.1 Active-Set

1. 1.36864885124676
2. 2.05161557953655
3. 2.04728473930635
4. 2.13896101515846
5. 2.10125902318712
6. 1.88677133157319
7. 2.02384696194694
8. 2.12556725019548
9. 2.13929867773197
10. 1.90428704380838
11. 2.02136355940529
12. 1.30885525335493
13. 2.05296494252675
14. 2.14001348081344
15. 1.99342781365927
16. 2.18967709356206
17. 2.06996970254221
18. 1.24025698553853
19. 1.98026310252799
20. 2.14518808114736
21. 1.99589506045517
22. 2.04565518768703
23. 2.12945450044758
24. 1.90682657030548
25. 1.97834058191798

Figure 5.1 Trust-Region-Reflective

1. 1.72055681963314
2. 2.12266560577333
3. 1.68795069084901
4. 1.59536344246211
5. 1.7008391555497
6. 1.96886017214349
7. 2.15760357628667
8. 2.1046361838147
9. 1.98810505224015
10. 1.94779075108765
11. 1.89304688101835
12. 1.99156594164502
13. 2.50638616748284
14. 1.30338357466903
15. 1.82618653329918
16. 1.57830495051274
17. 1.9146588171397
18. 1.84426016296193
19. 2.02222796077836
20. 1.54022008613437
21. 1.80685916707815
22. 2.09994322356435
23. 1.53423685908377
24. 1.92817678368601
25. 2.31077243920219

RVQ Decoders (Quantized Active-Set Solution, 25 Flooding Iterations)

Figure 5.2 5-bit Quantized

1. 1	6. 1	11. 2	16. 2	21. 2
2. 2	7. 2	12. 1	17. 2	22. 2
3. 2	8. 2	13. 2	18. 1	23. 2
4. 2	9. 2	14. 2	19. 1	24. 1
5. 2	10. 1	15. 2	20. 2	25. 1

Figure 5.2 7-bit Quantized

1. 4	6. 6	11. 6	16. 7	21. 6
2. 6	7. 6	12. 4	17. 6	22. 6
3. 6	8. 6	13. 6	18. 4	23. 6
4. 6	9. 6	14. 6	19. 6	24. 6
5. 6	10. 6	15. 6	20. 6	25. 6

Figure 5.2 10-bit Quantized

1. 17	6. 23	11. 25	16. 27	21. 25
2. 25	7. 25	12. 16	17. 26	22. 25
3. 25	8. 26	13. 25	18. 15	23. 26
4. 27	9. 27	14. 27	19. 25	24. 24
5. 26	10. 24	15. 25	20. 27	25. 25

10GE Stochastic Search Comparison (40 Flooding Iterations)

Figure 5.3 (M2) Genetic Algorithm (Extrapolated to 40)

1. 5	6. 5	11. 7	16. 5	21. 5
2. 6	7. 6	12. 8	17. 7	22. 7
3. 5	8. 7	13. 5	18. 7	23. 5
4. 6	9. 7	14. 6	19. 8	24. 6
5. 7	10. 7	15. 7	20. 6	25. 7

Figure 5.3 (M3) Windowed Search (Extrapolated to 40)

1. 5	6. 6	11. 5	16. 5	21. 5
2. 4	7. 5	12. 10	17. 6	22. 7
3. 5	8. 9	13. 6	18. 7	23. 10
4. 5	9. 8	14. 4	19. 6	24. 4
5. 6	10. 5	15. 5	20. 7	25. 8

Figure 5.3 (M4) DE Optimization

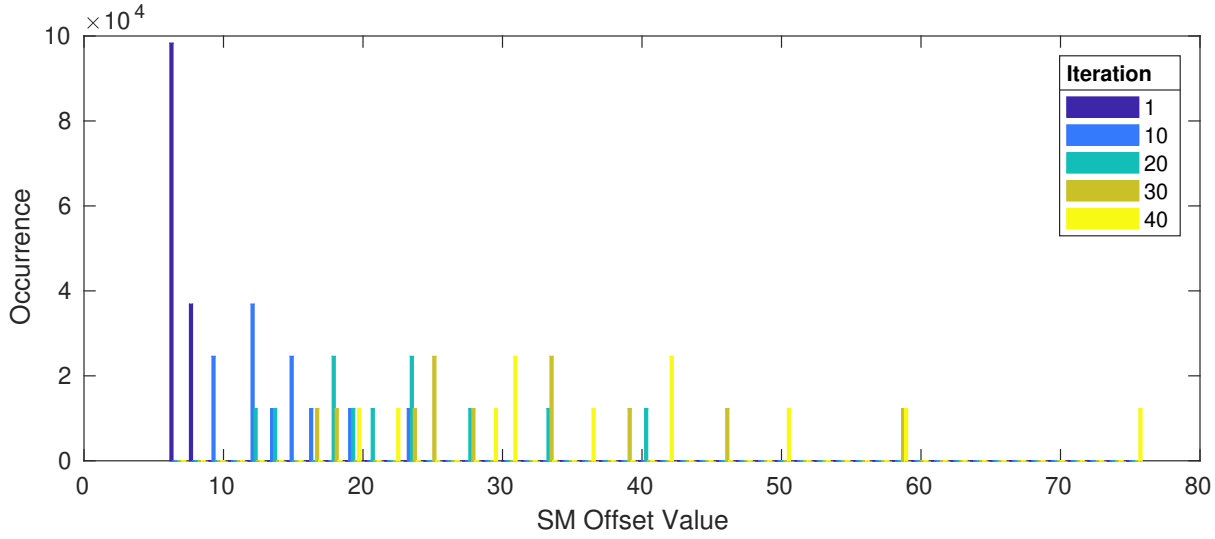

Figure B.1 Histogram showing the occurrence of SM offsets for all DE Optimized edges of the 10GE code

5G Stochastic Search Comparison (40 Flooding Iterations)

Figure 5.4 (M2) CD-EVW

$G_1 = \{3\}$

1. 3	9. 14	17. 25	25. 36	33. 47
2. 4	10. 15	18. 26	26. 37	34. 48
3. 6	11. 17	19. 28	27. 39	35. 50
4. 7	12. 18	20. 29	28. 40	36. 51
5. 9	13. 19	21. 30	29. 41	37. 52
6. 10	14. 21	22. 32	30. 43	38. 54
7. 11	15. 22	23. 33	31. 44	39. 55
8. 13	16. 24	24. 35	32. 45	40. 56

$G_2 = \{6, 7, 8, 9, 10\}$

1. 2	9. 9	17. 17	25. 24	33. 31
2. 3	10. 10	18. 17	26. 25	34. 32
3. 4	11. 11	19. 18	27. 26	35. 33
4. 5	12. 12	20. 19	28. 27	36. 34
5. 5	13. 13	21. 20	29. 28	37. 35
6. 6	14. 14	22. 21	30. 29	38. 36
7. 7	15. 15	23. 22	31. 29	39. 37
8. 8	16. 16	24. 23	32. 30	40. 38

$G_3 = \{19\}$

1. 1	2. 2	3. 3	4. 3	5. 4

6. 5	13. 11	20. 17	27. 23	34. 29
7. 6	14. 12	21. 18	28. 24	35. 30
8. 7	15. 13	22. 19	29. 25	36. 31
9. 8	16. 14	23. 20	30. 26	37. 32
10. 9	17. 15	24. 20	31. 26	38. 32
11. 9	18. 15	25. 21	32. 27	39. 33
12. 10	19. 16	26. 22	33. 28	40. 34

Figure 5.4 (M3) GA Windowed Search (Extrapolated to 40)

$G_1 = \{3\}$

1. 4	4. 8	7. 7	10. 1
2. 6	5. 8	8. 9	
3. 8	6. 8	9. 8	

$G_2 = \{6, 7, 8, 9, 10\}$

1. 5	4. 4	7. 7	10. 10
2. 3	5. 5	8. 8	
3. 4	6. 6	9. 9	

$G_3 = \{19\}$

1. 2	4. 0	7. 3	10. 10
2. 0	5. 1	8. 4	
3. 0	6. 2	9. 4	

Figure 5.4 (M4) DE Optimization Method

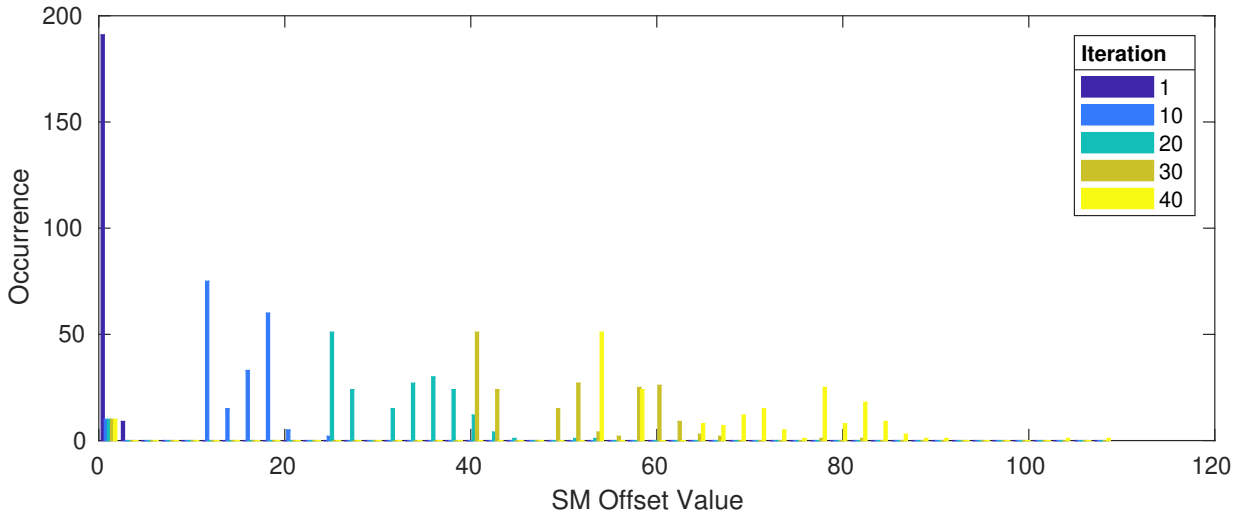

Figure B.2 Histogram showing the occurrence of SM offsets for all DE Optimized edges of the 5G code