Décodage itératif pour les codes LDPC au-delà de la propagation de croyances.

Shiva K. Planjery

Ph.D. THESIS

presented to

University of Cergy-Pontoise
École Doctorale Sciences et Ingénierie

to obtain the title of

Doctor of Science of the University of Cergy-Pontoise
Specialty: **Sciences and Technologies of Information and Communication**

Defended by

Shiva Kumar PLANJERY

Iterative Decoding Beyond Belief Propagation for Low-Density Parity-Check Codes

prepared at
Équipes Traitement de l'Information et Systèmes (ETIS) - UMR 8051
ENSEA - Université de Cergy-Pontoise - CNRS

Defense Date: December 5, 2012

Jury:

President:	Prof.	Jean-Claude Belfiore	Telecom-Paritech
Reviewer:	Prof.	Paul Siegel	University of California, San Diego
Reviewer:	Prof.	Gilles Zémor	University of Bordeaux
Examiner:	Dr.	Lucile Sassatelli	Université Nice Sophia Antipolis
Advisor:	Prof.	David Declercq	ETIS/ENSEA-University of Cergy Pontoise-CNRS
Co-Advisor:	Prof.	Bane Vasić	University of Arizona

Abstract

At the heart of modern coding theory lies the fact that low-density parity-check (LDPC) codes can be efficiently decoded by message-passing algorithms which are traditionally based on the belief propagation (BP) algorithm. The BP algorithm operates on a graphical model of a code known as the Tanner graph, and computes marginals of functions on the graph. While inference using BP is exact only on loop-free graphs (trees), the BP still provides surprisingly close approximations to exact marginals on loopy graphs, and LDPC codes can asymptotically approach Shannon's capacity under BP decoding.

However, on finite-length codes whose corresponding graphs are loopy, BP is sub-optimal and therefore gives rise to the error floor phenomenon. The error floor is an abrupt degradation in the slope of the error-rate performance of the code in the high signal-to-noise regime, where certain harmful structures generically termed as *trapping sets* present in the Tanner graph of the code cause the decoder to fail. Moreover, the effects of finite precision that are introduced during hardware realizations of BP can further contribute to the error floor problem.

In this dissertation, we introduce a new paradigm for finite precision iterative decoding of LDPC codes over the Binary Symmetric channel (BSC). These novel decoders, referred to as *finite alphabet iterative decoders* (FAIDs) to signify that the message values belong to a finite alphabet, are capable of surpassing the BP in the error floor region. The messages propagated by FAIDs are not quantized probabilities or log-likelihoods, and the variable node update functions do not mimic the BP decoder, which is in contrast to traditional quantized BP decoders. Rather, the update functions are simple maps designed to ensure a higher guaranteed error correction capability by using the knowledge of potentially harmful topologies that could be present in a given code. We show that on several column-weight-three codes of practical interest, there exist 3-bit precision FAIDs that can surpass the BP (floating-point) in the error floor without any compromise in decoding latency. Hence, they are able to achieve a superior performance compared to BP with only a fraction of its complexity. We also provide a semi-heuristic-based non-code-specific method for selection of a set of candidate FAIDs, one or several of which are potentially good for any given column-weight-three code.

Additionally in this dissertation, we propose decimation-enhanced FAIDs for LDPC codes, where the technique of decimation is incorporated into the variable node update function of FAIDs. Decimation, which involves fixing certain bits of the code to a particular value during the decoding process, can significantly reduce the number of iterations required to correct a fixed number of errors while maintaining the good performance of a FAID, thereby making such decoders more amenable to analysis. We illustrate this for 3-bit precision FAIDs on column-weight-three codes. We also show how decimation can be used adaptively to further enhance the guaranteed error correction capability of FAIDs that are already good on a given code. The new adaptive decimation scheme proposed has marginally added complexity but can significantly improve the slope of the error floor performance of a particular FAID. On certain high-rate column-weight-three codes of practical interest, we show that adaptive decimation-enhanced FAIDs can achieve a guaranteed error-correction capability that is close to the theoretical limit achieved by maximum-likelihood decoding ($\lfloor (d_{min} - 1)/2 \rfloor$).

Résumé

Les codes Low-Density Parity-Check (LDPC) sont au coeur de la recherche des codes correcteurs d'erreurs en raison de leur excellente performance de décodage en utilisant un algorithme de décodage itératif de type propagation de croyances (Belief Propagation - BP). Cet algorithme utilise la représentation graphique d'un code, dit graphe de Tanner, et calcule les fonctions marginales sur le graphe. Même si l'inférence calculée n'est exacte que sur un graphe acyclique (arbre), l'algorithme BP estime de manière très proche les marginales sur les graphes cycliques, et les codes LDPC peuvent asymptotiquement approcher la capacité de Shannon avec cet algorithme.

Cependant, sur des codes de longueurs finies dont la représentation graphique contient des cycles, l'algorithme BP est sous-optimal et donne lieu à l'apparition du phénomène dit de plancher d'erreur. Le plancher d'erreur se manifeste par la dégradation soudaine de la pente du taux d'erreur dans la zone de fort rapport signal à bruit où les structures néfastes au décodage sont connues en termes de Trapping Sets présents dans le graphe de Tanner du code, entraînant un échec du décodage. De plus, les effets de la quantification introduite par l'implémentation en hardware de l'algorithme BP peuvent amplifier ce problème de plancher d'erreur.

Dans cette thèse nous introduisons un nouveau paradigme pour le décodage itératif à précision finie des codes LDPC sur le canal binaire symétrique. Ces nouveaux décodeurs, appelés décodeurs itératifs à alphabet fini (Finite Alphabet Iterative Decoders - FAID) pour préciser que les messages appartiennent à un alphabet fini, sont capables de surpasser l'algorithme BP dans la région du plancher d'erreur. Les messages échangés par les FAID ne sont pas des probabilités ou vraisemblances quantifiées, et les fonctions de mise à jour des noeuds de variable ne copient en rien le décodage par BP ce qui contraste avec les décodeurs BP quantifiés traditionnels. En effet, les fonctions de mise à jour sont de simples tables de vérité conçues pour assurer une plus grande capacité de correction d'erreur en utilisant la connaissance de topologies potentiellement néfastes au décodage présentes dans un code donné. Nous montrons que sur de multiples codes ayant un poids colonne de trois, il existe des FAID utilisant 3 bits de précision pouvant surpasser l'algorithme BP (implémenté en précision flottante) dans la zone de plancher d'erreur sans aucun compro-

mis dans la latence de décodage. C'est pourquoi les FAID obtiennent des performances supérieures comparées au BP avec seulement une fraction de sa complexité.

Par ailleurs, nous proposons dans cette thèse une décimation améliorée des FAID pour les codes LDPC dans le traitement de la mise à jour des noeuds de variable. La décimation implique de fixer certains bits du code à une valeur particulière pendant le décodage et peut réduire de manière significative le nombre d'itérations requises pour corriger un certain nombre d'erreurs fixé tout en maintenant de bonnes performances d'un FAID, le rendant plus à même d'être analysé. Nous illustrons cette technique pour des FAID utilisant 3 bits de précision codes de poids colonne trois. Nous montrons également comment cette décimation peut être utilisée de manière adaptative pour améliorer les capacités de correction d'erreur des FAID. Le nouveau modèle proposé de décimation adaptative a, certes, une complexité un peu plus élevée, mais améliore significativement la pente du plancher d'erreur pour un FAID donné. Sur certains codes à haut rendement, nous montrons que la décimation adaptative des FAID permet d'atteindre des capacités de correction d'erreur proches de la limite théorique du décodage au sens du maximum de vraisemblance.

Contents

List of Figures

List of Abbreviations

ADFAID	Adaptive Decimation-enhanced Finite Alphabet Iterative Decoder
AWGNC	Additive White Gaussian Noise Channel
BCH	Bose-Chaudhuri-Hocquenghem
BCJR	Bahl Cocke Jelinek Raviv
BP	Belief Propagation
BEC	Binary Erasure Channel
BER	Bit Error Rate
BSC	Binary Symmetric Channel
DFAID	Decimation-enhanced Finite Alphabet Iterative Decoder
FAID	Finite Alphabet Iterative Decoder
FER	Frame Error Rate
LT	Linear Threshold
GF	Galois Field
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood Ratio
LP	Linear Programming
LUT	Look-Up Table
MAP	Maximum A Posteriori
ML	Maximum Likelihood
MP	Message Passing
NCNV	Noisy Critical Number Vector
NLT	Non-Linear Threshold
PEG	Progressive Edge-Growth
RS	Reed-Solomon
SNR	Signal-to-Noise Ratio
TS	Trapping Set
TSO	Trapping Set Ontology.

Chapter 1

Introduction

Ever since its inception in the 1950's, error-correcting codes have played an indispensable role in ensuring high reliability and low power consumption in wireless/wired data transmissions as well as in data storage, and have now become ubiquitous in all modern communication and data storage systems. An error-correction code essentially consists of an encoder and a decoder; information that is to be transmitted is encoded by adding redundancy in a specific manner at the transmitter and decoding is performed at the receiver to correct some or all of the errors contained in the received noisy data. In particular, a specific class of error-correcting codes called *low-density parity-check codes* (LDPC) [1] has not only revolutionized the communications and storage industry, but also sparked a widespread research interest over the past two decades, leading to the so-called field of *modern coding theory*.

Since this dissertation is centrally themed around LDPC codes, we shall begin this chapter by providing a brief overview of the historical background of error-correcting codes especially related to LDPC codes, and provide the necessary preliminaries required for understanding this dissertation. We will then describe the main motivation for this work and the context of the problems we are addressing. Finally, we will highlight the main contributions and provide the outline for the remainder of this dissertation.

1.1 Historical Background

The origin of error-correcting codes dates back to the work of Hamming during his time at Bell Labs in the late 1940's. At that time, Hamming got motivated to develop error-correcting codes when he became increasingly frustrated with relay computers he was working with, which would halt any submitted jobs containing errors. He eventually designed the first type of error-correcting linear block codes, now known as Hamming codes, which could correct a single error, and later published his work in 1950 [2]. This

led to the development of other important early codes such as the triple error-correcting Golay code [3] and the Reed-Muller (RM) codes [4, 5].

Meanwhile, Shannon, who happened to be a colleague of Hamming at Bell Labs, published his landmark paper [6] in 1948 that led to the birth of the field of information theory. The paper essentially laid down the fundamental limits of error-free transmission by introducing the notion of *channel capacity*. Shannon pointed out that every channel has an associated capacity, and he proved using random coding arguments that for transmission rates less than the channel capacity, there exists a code that can achieve arbitrarily low error-rates asymptotically with the length of the code. However, Shannon did not provide any insights into the explicit design of capacity-achieving codes, and thus, the quest for the search of such codes began.

In the ensuing years, much of the research was dedicated towards the design of linear block codes that had good error-correcting capabilities and good minimum distance. Initial code designs were for hard-decision channels and relied heavily on algebraic structure based on finite-field algebra. Codes such as the Bose-Chaudhuri-Hocquenghem (BCH) codes [7,8] and the Reed-Solomon (RS) codes [9] became widely used especially in magnetic recoding applications such as magnetic tape systems. The codes were decoded by an efficient algorithm developed by Berlekamp and Massey [10]. However, algebraic-based codes were still far from achieving the limits of Shannon's capacity in their asymptotic performance, and they lacked the random-like properties originally envisioned by Shannon.

Concurrently, another class of codes called *convolutional* codes were developed by Elias [11], which had a more probabilistic approach to channel coding. Interpreted as discrete-time finite-state systems, these codes had an underlying special structure known as the *trellis* which enabled linear-time encoding and inherently allowed for the capability to use practical soft-decision decoding algorithms. One of the most important decoding algorithms for these codes was developed by Viterbi, famously known as the *Viterbi algorithm* [12], which optimally estimated the most likely sequence of transmitted bits. This proved to be a major attraction for these codes, and they were capable of providing substantial coding gains even though their asymptotic performance was still far from Shannon's capacity. Another algorithm that was developed in the same probabilistic spirit was the BCJR algorithm by Bahl, Cocke, Jelinek, and Raviv [13] which estimated the *a posteriori* probability of each transmitted bit. At the time, the BCJR algorithm was considered to be purely of theoretical interest as it was regarded too complex for practical implementations. However, the BCJR algorithm along with the class of convolutional codes turned out to be the most crucial steps for paving the way towards the development of capacity-achieving codes.

The breakthrough in the search for capacity-achieving codes was finally reached in 1993 by Berrou, Glavieux, and Thitimajshima with their discovery of *turbo codes* [14]. A key feature of these codes was the use of *iterative decoding*, which involved using the BCJR algorithm to iteratively exchange soft information between two convolutional

codes concatenated in parallel. This proved to be pivotal for achieving near-Shannon-limit performance with reasonable decoding complexity. Subsequently, the class of *low-density parity-check* (LDPC) codes, originally invented by Gallager in early 1960's but remained forgotten for nearly thirty years, was rediscovered by MacKay [15] who showed that these codes were also capable of capacity-achieving performance. This led to a renaissance in the field of modern coding theory with LDPC codes becoming one of the most active topics of research.

LDPC codes are essentially linear-block codes whose parity-check matrices have a sparse number of non-zero entries. These codes are conveniently represented as bipartite graphs known as the *Tanner graphs* [16], and the decoding algorithm operates on the Tanner graph of the code. The decoding algorithms used for LDPC codes are based on a central iterative algorithm known as the *belief propagation*. The BP is a message-passing algorithm that involves propagating probabilistic messages along the edges of the graph in order to estimate the *a posteriori* probabilities of the codeword bits thereby lending itself to low complexity implementations. The notion of using graph-based decoding was eventually extended to other codes such as Turbo codes through the significant contributions of Tanner [16] and much later Wiberg [17]. A unifying framework for graph-based decoding is provided in [18] which links algorithms used in many other areas such as artificial intelligence, signal processing, and computer science to the BP algorithm. Under BP decoding, LDPC codes are able to achieve an unprecedentedly good error-rate performance which has made them the overwhelming choice among existing codes for both present and emerging technologies in digital communications and data storage.

Within the past decade, with LDPC codes steadily gaining popularity, there has been an outburst of research related to the design of capacity-achieving LDPC codes as well as in the development and analysis of decoding algorithms. Richardson and Urbanke proposed the key technique of *density evolution* [19] for determining the decoding threshold of a given LDPC code under BP decoding, which is a threshold of noise below which the bit error probability tends to zero asymptotically with the length of the code. Using density evolution, Richardson, Shokrollahi and Urbanke optimized capacity-achieving irregular code ensembles for the best (highest) decoding thresholds that were very close to the Shannon limit. The technique was also subsequently used in the design of reduced-complexity BP decoders by Chen *et al.* [20] and Fossorieri [21]*et al.*, and quantized BP decoders by Lee and Thorpe [22]. Another important asymptotic technique that was proposed for analyzing decoding algorithms simpler than BP (such as bit flipping) was the use of expander arguments and expander codes proposed by Sipser and Speilman [23]. This was later expanded and generalized by Barg and Zémor [24]. Burshtein and Miller [25] applied expander arguments to message-passsing to show that they could correct a linear fraction of errors. For finite-length analysis of codes, a novel approach to decoding called linear programming (LP) decoding was proposed by Feldman *et al.* [26], where the decoding problem is transformed to an LP formulation.

The problem of constructing codes with desirable structural properties and good min-

imum distance while maintaining their capacity-achieving ability also gained significant attention, with many applications facing stringent constraints in terms of storage requirements and implementation complexity. A particularly broad class of codes called *quasi-cyclic* codes generated great appeal to the industry, where the parity-check matrices of these codes consist of blocks of circulants enabling much more efficient encoding and decoding implementations. One of the first works on this was by Tanner *et al.* [27] who proposed a group-theoretic construction method to design structured codes with good minimum distance. Later, other notable works emerged which include the codes proposed by Fossorieri [28], finite geometry codes proposed by Kou *et al.*, [32] the Protograph codes developed by Thorpe [29] and Divsalar *et al.* [30], the algebraic-based quasi-cyclic codes proposed by Lan *et al.* [31], and combinatorially constructed codes by Vasic *et al.* [33]. Another class of codes that recently gained limelight are LDPC convolutional codes [34, 35], which are the convolutional counterparts to the conventional linear block code version of LDPC codes, and were recently shown by Lentmaier *et al.* to be capable of having BP thresholds matching the MAP thresholds [36].

LDPC codes have already found their way into various standards such as the DVB-S2 (Digital Video Broadcasting), IEEE 802.3an (Ethernet), and are also being considered for the IEEE 802.16e (Wimax) and IEEE 802.11n (Wifi) standards. However, in spite of their excellent error-rate performance under BP decoding and their obvious advantages, they still have a major weakness that manifests itself in the form of an *error floor*. The error floor is an abrupt degradation in the performance of the code in the high signal-to-noise (SNR) region. This problem mainly arises due to the sub-optimality of BP decoding on finite-length codes with loopy graphs, especially when codes are designed to be asymptotically close to Shannon's limit. It could prove to be a major disadvantage particularly for applications requiring very low target error-rates such as storage devices. Therefore, the error floor problem has been widely regarded as one of the most important problems in coding theory and has attracted significant research interest over the past few years. Addressing the error floor problem is one of the main goals of this dissertation along with taking into account the effects of finite-precision for decoder realizations.

1.2 LDPC codes: Fundamentals and Notations

In this section, we will provide the necessary preliminaries related to LDPC codes starting from the basics of linear block codes to the general concept of message-passing and graph-based decoding. Along the way, we will introduce the required notations that will be used throughout this dissertation.

1.2.1 Linear block codes

An (N, K) binary linear block code $\mathcal{C}$ [37] is a K-dimensional subspace of $\mathrm{GF}(2)^N$, which is the vector space over the field GF(2) consisting of all possible binary N-tuples. Thus, the code $\mathcal{C}$ contains 2^k N-tuples or *codewords*. Given a message vector of k information bits, this vector is mapped to one of the codewords of length N in the code $\mathcal{C}$. Since $\mathcal{C}$ is a subspace, any linear combination of two codewords in $\mathcal{C}$ results in another codeword in $\mathcal{C}$, an important property of linear block codes. The code $\mathcal{C}$ is said to have a *code rate* of $R = K/N$, which represents the amount of redundancy added by the code.

The support of a codeword $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, denoted by $\mathrm{supp}(\mathbf{x})$, is the set of all positions i such that $x_i \neq 0$. The *hamming weight* (or simply weight) of $\mathbf{x}$ denoted $w(\mathbf{x})$, is the cardinality of $\mathrm{supp}(\mathbf{x})$. The *hamming distance* (or simply distance) between any two codewords is the number of positions for which they differ in value. The minimum distance of a code $\mathcal{C}$, denoted by d_{min} is the smallest possible distance between any pair of codewords in $\mathcal{C}$. A codeword vector that has zero in all its positions is referred to as the *all-zero codeword*. Since a linear block must include the all-zero codeword, the minimum distance is simply the weight of the nonzero codeword that has the smallest weight.

The process of mapping a message vector of k information bits to a codeword of length N in the code $\mathcal{C}$ is known as *encoding*. Encoding is carried out through a $K \times N$ generator matrix $\mathbf{G}$ of the code $\mathcal{C}$, whose rows correspond to the basis vectors of $\mathcal{C}$, i.e., the linearly independent codewords. Given a message vector of k bits, say, $\mathbf{u} = (u_1, u_2, \ldots, u_k)$, a codeword $\mathbf{x} \in \mathcal{C}$ can obtained by performing $\mathbf{x} = \mathbf{uG}$.

A linear block code is also characterized by its parity-check matrix $\mathbf{H}$, which is an $M \times N$ matrix whose rows span the null space of $\mathbf{G}$, i.e., $\mathbf{GH}^T = 0$. Consequently, every codeword $\mathbf{x}$ is orthogonal to the rows of $\mathbf{H}$ so that $\mathbf{xH}^T = 0$. Given a vector $\mathbf{x}' \in \mathrm{GF}(2)^N$, the parity-check matrix can be used to verify whether $\mathbf{x}'$ is a codeword belonging to $\mathcal{C}$ or not. Therefore, each row of $\mathbf{H}$ is referred to as a *parity-check constraint* and there are M such parity-check constraints. The value of M is related to the code parameters by $M \geq (N - K)$, where the equality holds if $\mathbf{H}$ is a full-rank matrix.

Once a given message $\mathbf{u}$ is encoded to a codeword $\mathbf{x}$, it is then transmitted over a noisy channel and is received as $\mathbf{r} = (r_1, r_2, \ldots, r_N)$. *Decoding* is then performed to estimate the transmitted codeword $\mathbf{x}$ based on the received vector $\mathbf{r}$. The optimal decision rule used for determining the estimate $\hat{\mathbf{x}}$ is the *maximum a posteriori* (MAP) decoding rule, which essentially chooses a codeword $x \in \mathcal{C}$ that maximizes the a posteriori probability $\Pr(\mathbf{x}|\mathbf{r})$ (the probability that the transmitted codeword is $\mathbf{x}$ given that $\mathbf{r}$ was received). More precisely,

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

We shall assume that all codewords are equally likely to be transmitted. Under this assumption, the MAP decoding problem reduces to the *maximum likelihood* (ML) decod-

ing problem which determines the codeword that maximizes the $\Pr(\mathbf{r}|\mathbf{x})$. If $\mathbf{r} \in \mathrm{GF}(2)^N$, then the ML decoding problem is equivalent to finding the nearest neighbor of $\mathbf{r}$ in the vector space $\mathrm{GF}(2)^N$ that is a codeword in $\mathcal{C}$. Therefore the code's error-correcting capability is linked to its minimum distance d_{min} as the code is guaranteed to correct $\lfloor (d_{min} - 1)/2 \rfloor$ errors. In general, ML decoding on a linear-block code is NP-hard as it requires a brute-force search of all codewords in the vector space $\mathrm{GF}(2)^N$. Hence, sub-optimal decoding algorithms that enable efficient implementations are used in practice.

1.2.2 Channel assumptions

Let us assume that a codeword $\mathbf{x} = (x_1, x_2, \ldots, x_N) \in \mathcal{C}$ is transmitted over a noisy channel and is received as a vector $\mathbf{r}$. If the channel is memoryless, then the probability $\Pr(\mathbf{r}|\mathbf{x})$ can be expressed as

$$\Pr(\mathbf{r}|\mathbf{x}) = \prod_{i=1}^{N} \Pr(r_i|x_i)$$

This implies that the effect of the channel on every bit in the transmitted codeword is independent from one another. Hence, during decoding, it suffices to decide the value for each x_i independently based on maximizing $\Pr(r_i|x_i)$. This probability is also referred to as a *likelihood*.

For soft-decoding algorithms, the received vector $\mathbf{r}$ is mapped to a vector of probabilities or log-likelihood ratios before serving as input to the decoder. Let $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ denote the vector that is input to a given decoder. We shall refer to the values y_i in vector $\mathbf{y}$ as *channel values*, to signify the fact they are determined based on the values received from the channel. If $\mathbf{y}$ is a vector of log-likelihoods, then the log-likehood ratio (LLR) corresponding to a bit position i is given by

$$y_i = \log\left(\frac{\Pr(r_i|x_i = 0)}{\Pr(r_i|x_i = 1)}\right)$$

Examples of memoryless channels include the Binary Symmetric channel (BSC), the Binary erasure channel (BEC), and the Additive White Gaussian channel (AWGNC). Where as examples of channels with memory include partial-response channels and other channels in magnetic recording that cause inter-symbol intereference (ISI).

For this dissertation, we shall only focus on the BSC. The BSC is a binary-input binary-output memoryless channel and therefore the received vector $\mathbf{r}$ is also binary N-tuple. The channel flips a bit in the transmitted codeword with a *cross-over probability* of α. Fig. 1.1 shows an illustration of the BSC. A transmitted bit 0 is received as 0 with probability $1 - \alpha$ and as a 1 with probability α.

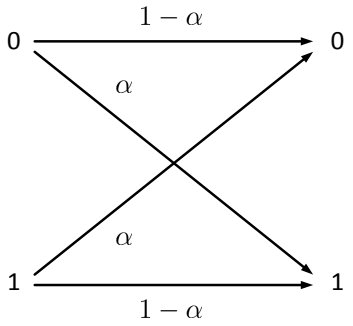


Figure 1.1: Illustration of the Binary Symmetric channel

For the BSC, given a cross-over probability α,the LLR is determined from $\mathbf{r}$ as follows

$$y_i = \begin{cases} \log\left(\frac{1-\alpha}{\alpha}\right) & \text{if } r_i = 0 \\ \log\left(\frac{\alpha}{1-\alpha}\right) & \text{if } r_i = 1 \end{cases}$$

Note that by the definition above, y_i being positive implies that the corresponding codeword bit is more likely to be a zero, and y_i being negative implies the negative corresponding codeword bit is more likely to be a one.

1.2.3 Tanner graph representation

LDPC codes are linear block codes that have sparse parity-check matrices. Any (N, K) linear block code $\mathcal{C}$ can be represented by a bipartite graph G, also known as the Tanner graph, which consists of two sets of nodes: the set of variable nodes $V = \{v_1, \cdots, v_N\}$ and the set of check nodes $C = \{c_1, \cdots, c_M\}$. Given the parity-check matrix $\mathbf{H}$ of a code $\mathcal{C}$, the Tanner graph is obtained by assigning a variable node corresponding to each codeword bit and a check node corresponding to each parity-check constraint, and then connecting variable nodes to certain check nodes based on the parity-check matrix. A edge connection between a particular variable node and a particular check node is made if the codeword bit associated with the variable node participates in the parity-check constraint associated with the check node. This is illustrated with the help of an example.

Example 1.1. *Consider the parity-check matrix of an* $(8, 4)$ *extended Hamming code with* $d_{min} = 4$*, which is given by*

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

In order generate the Tanner graph, first observe that the graph must contain 8 variable nodes corresponding to the 8 columns, and 4 check nodes corresponding to the 4 parity-check constraints. Therefore $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ *and* $C = \{c_1, c_2, c_3, c_4\}$. *By convention,* $\bigcirc$ *is used to depict a variable node, and* $\square$ *is used to depict a check node in the Tanner graph. Now looking at the first column in* $\mathbf{H}$*, there is a 1 in the first row and in the fourth row. Therefore,* v_1 *is connected to check nodes* c_1 *and* c_4*. Similarly, all other variable nodes are connected to the appropriate check nodes. Fig. 1.2 shows the resulting Tanner graph of the code.*

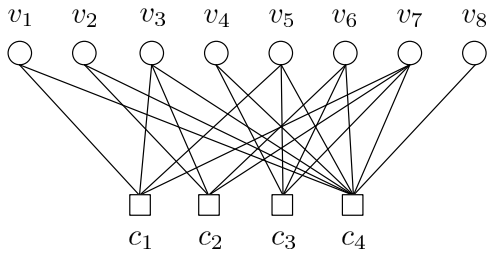


Figure 1.2: Tanner graph of the $(8, 4)$ extended Hamming code

The check nodes (variable nodes respectively) connected to a variable node (check node respectively) are referred to as its *neighbors*. The set of neighbors of a node v_i is denoted as $\mathcal{N}(v_i)$, and the set of neighbors of node c_j is denoted by $\mathcal{N}(c_j)$. The degree of a node is the number of its neighbors. Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ be a vector such that x_i denotes the value of the codeword bit associated with the variable node v_i. We shall refer to this value as the *bit value* of node v_i. $\mathbf{x}$ is a codeword if and only if for each check node, the modulo two sum of the bit values of its neighbors is zero.

An LDPC code is to be *regular*, if in its corresponding Tanner graph G, all variable nodes have the same degree d_v, and all check nodes have the same degree d_c. A code is said to be a column-weight-d_v (or d_v-left-regular) code if all variable nodes have the same degree d_v, which is also referred to as *left degree*. A code is said to be *irregular* if there are variable nodes as well as check nodes that have different degrees. The (8,4) extended Hamming code provided above is an example of an irregular code.

The girth of the Tanner graph g is length of the shortest cycle present in the Tanner graph of the code. A cycle of length g is referred to as a g-cycle. For this dissertation, we will be mostly concerned with column-weight-three LDPC codes. Henceforth, for convenience, Tanner graphs shall be simply referred to as graphs.

1.2.4 Message-passing decoders

Message-passing (MP) decoders are a class of iterative decoders that operate on the graph of the code. The a posteriori probability of a codeword bit associated to each variable

node in the graph is calculated by exchanging messages iteratively between the set of variable nodes and the set of check nodes along the edges of the graph. A major attraction with MP decoders is that all the computations are carried out locally at each node, and therefore an efficient decoder implementation can be realized.

Recall that $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ is the input to the MP decoder. Let $\mathcal{Y}$ denote the alphabet of all possible channel values. Let $\mathcal{M}$ denote the alphabet of all possible values that the messages can assume. Let $m^{(k)}(v_i, c_j)$ denote the message passed by a variable node $v_i \in V$ to its neighboring check node $c_j \in C$ in the k^{th} iteration and $m^{(k)}(c_j, v_i)$ denote the vice versa. Let $m^{(k)}(\mathcal{N}(v_i), v_i)$ denote the set of all incoming messages to variable node v_i and $m^{(k)}(\mathcal{N}(v_i)\backslash c_j, v_i)$ denote the set of all incoming messages to variable node v_i except from check node c_j. Let $m^{(k)}(\mathcal{N}(c_j)\backslash v_i, c_j)$ and $m^{(k)}(\mathcal{N}(c_j)\backslash v_i, c_j)$ be defined similarly.

Any MP decoder requires two update functions: Φ_v used for update at the variable nodes, and Φ_c used for check nodes. Both update functions are symmetric functions on the incoming messages, i.e., they remain unchanged by any permutation of its messages. Thus without abuse of notation, if the arguments of a function are written as a set, we imply that the order of these arguments is insignificant.

We now describe the MP decoding process. Initially, all the messages are set to zero, i.e., $m^{(0)}(\mathcal{N}(v_i), v_i) = 0 \ \ \forall v_i \in V$ and $m^{(0)}(\mathcal{N}(c_j), c_j) = 0 \ \ \forall c_j \in C$. Then for every iteration $k > 0$, messages are propagated in the following manner:

$$\begin{aligned} m^{(k)}(v_i, c_j) &= \Phi_v \left(y_i, m^{(k-1)}(\mathcal{N}(c_j)\backslash v_i, c_j) \right) \\ m^{(k)}(c_j, v_i) &= \Phi_c \left(m^{(k)}(\mathcal{N}(c_j)\backslash v_i, c_j) \right) \end{aligned}$$

An important feature of MP decoding that can be noted from the above is that during the update of messages at a particular node, the outgoing message from the node on an edge is determined as a function of all local messages incoming from its neighbors excluding the message incoming on that particular edge. We refer to such messages as *extrinsic incoming messages*. By doing so, the node is to an extent treating all its incoming messages as independent messages. This is the subtle difference between MP decoding and iterative bit-flipping decoding algorithms. Figs. **??** and 1.4 illustrate the message updating at a variable node and check node respectively, with m_1, m_2, and m_3 denoting the incoming messages to a particular node.

At the end of each iteration, a symmetric *decision function* Ψ, is used to decide the bit value of each node $v_i \in V$ based on its incoming messages. Let $\hat{\mathbf{x}}^{(k)} = (\hat{x}_1^{(k)}, \hat{x}_2^{(k)}, \ldots, \hat{x}_N^{(k)})$ denote the vector of bit values decided at the end of the k^{th} iteration. The bit value $\hat{x}_i^{(k)}$ of

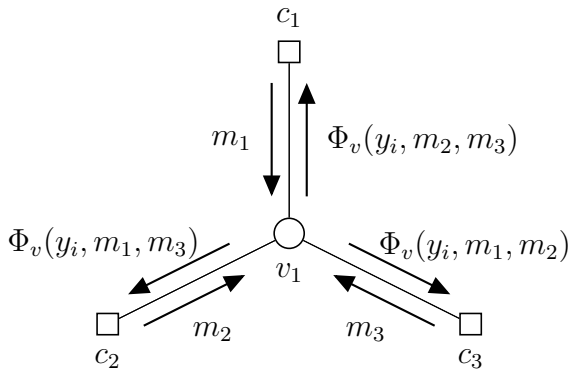


Figure 1.3: Update of the messages at a variable node with $d_v = 3$.

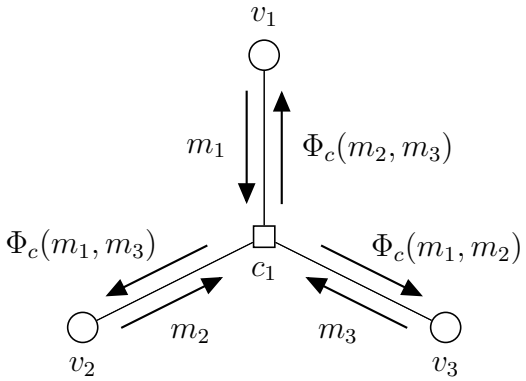


Figure 1.4: Update of messages at a check node with $d_c = 3$

each node v_i is determined by

$$\hat{x}_i^{(k)} = \Psi\left(y_i, m^k(\mathcal{N}(v_i), v_i)\right)$$

The estimate $\hat{\mathbf{x}}^{(k)}$ is then verified to check whether it is a codeword or not. This is easily done by passing the decided bit values along the edges and verifying whether every check node is *satisfied* or not. A check node is satisfied if the modulo 2 sum of all the bit values of its neighbors is zero. If all check nodes are satisfied, then $\hat{\mathbf{x}}^{(k)}$ is a codeword and the decoding is terminated.

In conventional MP decoders, the decision function is either a simple majority rule (if messages are binary), or simply the sign of the sum of its arguments (if messages are log-likelihoods). More precisely, for a node v_i with degree d_v

$$\Psi(y_i, m^k(\mathcal{N}(v_i), v_i)) = \begin{cases} 1 & \text{if } \left(y_i + \sum m^{(k)}(\mathcal{N}(v_i), v_i)\right) > 0 \\ 0 & \text{if } \left(y_i + \sum m^{(k)}(\mathcal{N}(v_i), v_i)\right) < 0 \\ \text{sign}(y_i) & \text{otherwise} \end{cases} \tag{1.1}$$

where the sign function outputs a 0 if the sign of the argument is positive, and a 1 if it is negative.

The algorithm runs until either $\hat{\mathbf{x}}^{(k)}$ is a codeword, or until the maximum number of iterations is reached, whichever occurs first. We say that the decoder has *converged* if $\hat{\mathbf{x}}^{(k)}$ is a codeword for some k, else we say that the decoder has *failed*. A *success* is declared if the decoder converges to the right codeword, and a *mis-correction* is declared if the decoder converges to a wrong codeword.

Note that during our general description of MP decoding, we inherently assumed that the scheduling used is a *flooding* scheduling, which involves updating all variable nodes simultaneously update followed by a simultaneous update of all check nodes. In contrast, a *serial* or layered scheduling could also be carried out for [38]. Discussions regarding scheduling are beyond the scope of this dissertation, and throughout we shall implicitly assume that the flooding scheduling is used for any MP decoder being described.

MP decoders can be broadly classified into two classes: hard decoding algorithms (where the messages assume binary values) such as the Gallager A and the Gallager B algorithms [1], and soft decoding algorithms (where the messages assume real values) which are conventionally based on the BP algorithm such as the sum-product algorithm and the min-sum algorithm [20]. We now discuss the BP algorithm in greater detail.

1.2.5 Belief propagation: Sum-Product and other low-complexity variants

All conventional MP algorithms used for decoding LDPC codes are based on the BP algorithm [39]. The BP has its roots in the broad class of Bayesian inference problems [40], and it is used to compute marginals of functions on a graphical model. Although exact inference in Bayesian belief networks is hard in general and inference using BP is exact only on loop-free graphs (trees), it provides surprisingly close approximations to exact marginals on loopy graphs.

The problem of decoding on the graph of the code is also a Bayesian inference problem, and therefore the BP algorithm is well-suited for this purpose. As a decoding algorithm for LDPC codes, the BP propagates probabilistic messages along the edges of the Tanner graph in an iterative fashion. BP can be implemented n two domains, namely the probabilistic domain where the messages are probabilities, and the LLR domain where messages are LLRs. We shall describe the BP algorithm in the LLR, domain which is more commonly used due to it being less sensitive to numerical precision issues. The BP algorithm is also known as the *Sum-Product* algorithm.

In the LLR domain, the messages and channel values are real-valued LLRs, i.e., $\mathcal{M} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$. The function $\Phi_v : \mathbb{R} \times \mathbb{R}^{d_v-1} \to \mathbb{R}$ is the update function used at a variable node of degree d_v, and$\Phi_c : \mathbb{R}^{d_c-1} \to \mathbb{R}$ is the update rule used at a check node with degree d_c.

Let $m_1, m_2, \ldots, m_{l-1}$ denote the $l-1$ extrinsic incoming messages of a node with degree l, which are used in the calculation of the outgoing message. The update rules for the BP algorithm are given as follows:

$$\Phi_v(y_i, m_1, \ldots, m_{d_v-1}) = y_i + \sum_{j=1}^{d_v-1} m_j \tag{1.2}$$

$$\Phi_c(m_1, \ldots, m_{d_c-1}) = 2\tanh^{-1}\left(\prod_{j=1}^{d_c-1} \tanh\left(\frac{m_j}{2}\right)\right) \tag{1.3}$$

When the magnitudes of the messages reach a large value, the check node update function of the BP algorithm can be approximated to:

$$\Phi_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \operatorname{sgn}(m_j)\right) \min_{j\in\{1,\ldots,d_c-1\}}(|m_j|) \tag{1.4}$$

where sgn denotes the standard signum function. The check node update function defined in Eq. (1.4) along with variable node update function defined in Eq. (1.2) together constitute the min-sum decoder. Thus, the min-sum is an approximation of BP decoding.

Clearly the min-sum decoder is much lower in complexity than the BP algorithm due to its simplified check node operation. However, the performance loss arising from using

min-sum instead of BP is quite large, especially in the low to mid SNR region. Hence, many low-complexity variants have been proposed which attempt to reduce this gap in performance. All of the variants proposed typically deal with simplifying the check node update function while keeping the variable node update function intact, as defined in Eq. (1.2).

One important low-complexity variant of BP that requires mentioning is the *offset min-sum* decoder proposed by Chen *et al.* [20]. For this algorithm, an offset factor γ is introduced into the check node update function. This offset factor serves to reduce the overstimate of the outgoing message produced by a check node using the rule in Eq. (1.4), especially when the magnitudes of the incoming messages are small. Therefore, the offset factor is also often referred to as a *correction factor*. The modified check node update function is as follows:

$$\Phi_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \operatorname{sgn}(m_j) \right) \max \left(\min_{j \in \{1,\ldots,d_c-1\}} (|m_j|) - \gamma, 0 \right) \tag{1.5}$$

The offset factor γ could be fixed or could be varied as a function of the SNR. Also the value for γ can be chosen using the density evolution technique for maximizing the decoding threshold as done in [20], or by a brute-force simulation on a given code that checks different values of γ and selects the one giving the best performance. Note from Eq. (1.5) that the outgoing messages is zero if the magnitude of an incoming message is less than the offset factor γ.

1.3 Error Floor Problem

For any typical finite-length LDPC code, the error-rate performance curve plotted as a function of SNR under iterative decoding, consists of two distinct regions: the waterfall region, and the error floor region. In the waterfall region (which is the low SNR region), the error-rate drops significantly with increase in SNR making the curve look like a waterfall. On the other hand, in the error floor region (which is the high SNR region), the decrease in the error-rate dramatically slows down and the curve tends to flatten out turning into an error floor. Fig. 1.5 illustrates the two regions on a typical FER performance curve of an LDPC code plotted as a function of the cross-over probability α of the BSC.

The error floor problem arises due to the suboptimality of BP decoding on loopy graphs. It can be troublesome for applications requiring very low target error-rates, and especially when high-rate codes are employed, which have relatively dense graphs. Although asymptotic techniques such as density evolution provide an accurate prediction of the FER performance in the early waterfall region, the point at which the error floor starts as well as its slope are greatly dependent on the particular structure of a code, and hence cannot be predicted by density evolution. Therefore, finite-length analysis techniques are

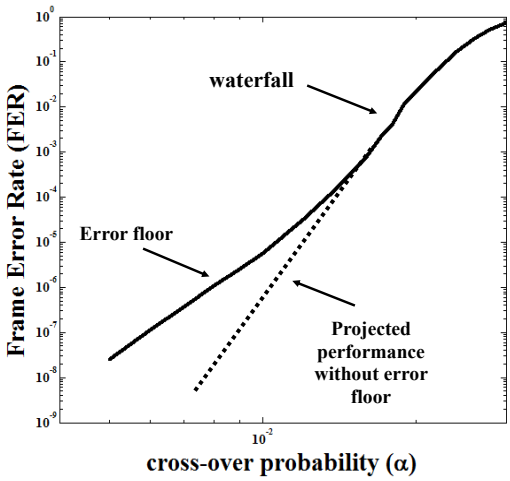


Figure 1.5: Typical performance of BP decoding on an LDPC code over the BSC

required for the study of error floors. Besides, for codes with moderate to large lengths, the ability to predict error floors becomes even more critical as the error floor may be unreachable by Monte-Carlo simulations.

1.3.1 Prior work on error floor estimation

The first crucial observation on error floors was made by MacKay and Postol [41], who found that the minimum distance of an LDPC code did not necessarily play a dominant role in the error floor of a code. In their investigations with a Margulis construction of an LDPC code for the AWGNC, they observed that it showed a high error floor in spite of its good minimum distance, and they attributed it to the presence of *near-codewords* in the graph. Another notion proposed in the spirit of finite-length analysis was the notion of *pseudocodewords* originally introduced by Wiberg [17], and later further developed by Frey *et al.* [42] and Forney *et al.* [43]. Pseudocodewords are essentially outputs of the decoder that are not necessarily codewords and stem from the different possible binary configurations of the computation tree. These works further motivated investigations into finite-length analysis of codes under iterative decoding with attempts to characterize the error floors for different channels.

For the BEC, the error floor could be well-characterized through the work of Di *et al.* [44], who introduced the notion of *stopping sets* which are purely combinatorial objects. Later Orlitsky *et al.* [45] provided asymptotic results on the stopping set distribution for different code ensembles. Unfortunately, such a level of understanding could never be achieved for more general channels such as AWGNC.

However, a huge stride towards the understanding of the error floor problem in a general setting, was made through the pioneering work of Richardson [46], who showed

that the problem is at least partially combinatorial in nature. Richardson introduced the notion of *trapping sets*, which are certain problematic loopy structures present in the graph of the code that cause the decoder to fail for certain low-noise configurations, and are a function of both the code and the decoding algorithm in operation. Using this notion, he proposed a semi-analytical method for estimating the error-rates of BP decoding on the AWGNC in the error floor by identifying the dominant trapping sets. Another approach based on statistical physics was also proposed for the AWGNC by Stepanov *et al.* [47] through the notion of *instantons*, which are certain low-noise configurations that swayed the decoding trajectory away from the correct codeword. Later, Chilappagari *et al.* [48] in line with Richardson's work, presented results on the error floor estimation for the BSC under Gallager B decoding. Another notion called *absorbing sets*, was proposed by Dolecek *et al.* [49], which are a special type of trapping sets that are purely combinatorial and stable under the bit-flipping algorithm, and this notion enabled them to perform a detailed analysis on array-based LDPC codes.

Many other works also subsequently emerged that further developed on the notion of pseudocodewords. Vontobel and Koetter introduced the concept of *graph covers* to explain failures of iterative decoding [51], and showed that pseudocodewords arising from graph covers are identical to pseudocodewords of the linear programming (LP) decoding [26] which constituted the vertices of the *fundamental polytope*. Later, Kelly and Sridhara [52] used graph covers to derive bounds on the minimum pseudocodeword weight in terms of girth and the minimum left-degree of the graph. The pseudocodeword analysis was also extended to the class LDPC-convolutional codes by Smarandache *et al.* [53]. However, pseudocodewords are purely topological in nature and do not take a particular decoding algorithm into account. Therefore, the notion of trapping sets is required for studying failures of a particular iterative decoder.

It is evident from previous discussions that the main utility behind the notion of trapping sets is in enabling the error floor estimation of a given code under a particular decoding algorithm. If the relevant topological structures corresponding to different trapping sets could be identified a priori and enumerated on the graph of a given code without the need for simulation, a reasonable estimate of the error floor could be obtained provided that each trapping set's contribution towards the error floor is known or determined.

Although it was shown by McGregor and Milenkovic [55] that performing an exhaustive search of all trapping sets in the graph of a given code is NP-hard, several good practical approaches have been proposed. Milenkovic *et al.* [56] examined the asymptotic trapping set distributions for both regular and irregular LDPC code ensembles. Cole *et al.* [57] proposed a method to estimate the error floor based on importance sampling, which is similar to Richardson's approach in [46]. Wang *et al.* proposed an efficient algorithm to exhaustively enumerate both trapping sets and stopping sets for codes with relatively short to moderate block lengths ($N \approx 500$). Abu-Surra *et al.* [58] proposed an efficient improved impulse method that could enumerate trapping sets by augmenting the Tanner graph with auxiliary variable nodes, and treating the problem as if it

were searching for low-weight codewords. Although the method does not guarantee enumeration of all trapping sets, it is reasonably reliable and is applicable to any arbitrary graph. A method that uses the *branch-and-bound* approach to enumerate stopping sets in a given code was proposed by Rosnes and Ytrehus [59]. Karimi and Bannihasemi [60] recently proposed an algorithm which could efficiently enumerate the dominant trapping sets present in any arbitrary graph by recursively expanding the cycles present in the graph. More recently, Zhang and Siegel [61] proposed an efficient technique that expanded on the branch-and-bound approach by transforming the bounding step to an LP formulation, and which allowed a complete enumeration of trapping sets up to a certain size with reasonable computation time.

In spite of the fact that all the above works are useful for enumerating trapping sets in a given graph which aid in estimating the error floor, none of them directly address the issue of how to utilize the knowledge of trapping sets for constructing better codes or improving the decoding algorithms. For instance, it is highly non-trivial to determine which particular subgraphs (that correspond to certain trapping sets) should be avoided during the construction of codes in order to improve the error floor performance. Vasić *et al.* [62] proposed the *trapping set onotology*, which is a hierarchy of trapping sets that exploits the topological relations, and can be utilized for code construction or decoder design. We shall use this in our dissertation in the decoder design.

For the remainder of this section, we shall elaborate on some of the relevant notions used to characterize decoder failures, describe the trapping set ontology, and finally highlight the importance of guaranteed error correction which is one of the main themes of this work.

1.3.2 Characterization of failures of iterative decoders

We begin by defining the notion of *trapping sets* and their related terminologies as originally provided by Richardson in [46]. Note that for purpose of exposition, we shall assume that the all-zero codeword was transmitted. This is a valid assumption as the MP decoders considered and the channel (BSC) are symmetric [19]. Recall that $\mathbf{y}$ is the decoder input, and $\hat{\mathbf{x}}^{(k)}$ is the estimate on the codeword bits after k iterations. Let I denote the maximum number of iterations allowed for decoding.

Definition 1.1. *A variable node v_i is said to be eventually correct if there exists a positive integer l such that for all $l \leq k \leq I$, $\hat{x}_i^{(k)} = 0$*

Definition 1.2. *For a decoder input $\mathbf{y}$, a trapping set $\mathbf{T}(\mathbf{y})$ is a non-empty set of variable nodes that are not eventually corrected by the decoder.*

The above definition is quite general as it includes all possible decoder failures such as stable fixed points as well as failures resulting from the oscillatory behavior of the decoder output among different states constituting a basin of attraction. Note that if $\mathbf{T}(\mathbf{y}) = \emptyset$, then the decoding is successful on input $\mathbf{y}$.

A trapping set (TS) $\mathbf{T}(\mathbf{y})$ is said to be an (a, b) TS, if it has a variable nodes, and b odd-degree check nodes in the subgraph induced by $\mathbf{T}(\mathbf{y})$. For convenience, we shall use the notation $\mathcal{T}(a, b)$ to denote the topological structure associated with an (a, b) TS, which is simply a graph consisting of a variable nodes and b odd-degree check nodes. Note that two trapping sets can share the same parameters (a, b) but have different underlying topologies.

If the degree of a check node is at most two in the graph, then the TS is said to be an *elementary* TS. It has been observed by many researchers that the dominant trapping sets in the error floor are typically elementary trapping sets [46, 54]. Therefore, we shall place more focus on elementary trapping sets during our discussions. Fig. 1.6 shows an example of a subgraph corresponding to an elementary $(5, 3)$ TS. Note that ■ is used to denote an odd-degree check node, and will be used throughout this dissertation for any future graphical illustrations.

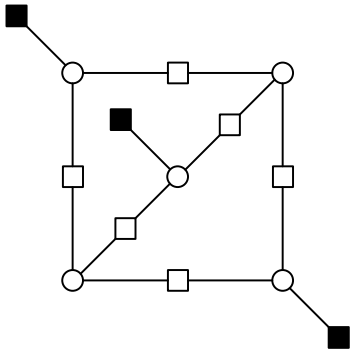


Figure 1.6: Subgraph corresponding to a $(5, 3)$ TS

Note that non-zero codewords (under the all-zero codeword assumption) are $(a, 0)$ trapping sets whose corresponding graphs have only even-degree check nodes. These are precisely the only trapping sets under ML decoding. However, for any sub-optimal iterative decoder, there will be other (a, b) trapping sets.

Stopping sets [44] are a sub-class of trapping sets that were introduced to characterize failures on the BEC. Note that for the BEC, a transmitted bit is either received correctly or is erased. Let S denote a subset of variable nodes and let $\mathcal{N}(S)$ denote the set of neighbors $\{\mathcal{N}(v_i) : \forall v_i \in S\}$. The definition is given below.

Definition 1.3. *A stopping set is a subset of variable nodes S in the graph G, such that every neighbor in $\mathcal{N}(S)$ is connected to subset S at least twice.*

The above definition implies that stopping sets are (a, b) trapping sets whose corresponding graphs do not contain any degree-one check node. For the BEC, iterative decoding fails if all a variable nodes of an (a, b) stopping set is erased, regardless of its neighborhood in the original graph G and the number of iterations allowed. Therefore,

stopping sets can be treated as purely combinatorial objects, and the error floor can be completely characterized for the BEC under iterative decoding. Fig. 1.7 shows an example of an $(8, 2)$ stopping set whose corresponding graph contains two degree-3 check nodes.

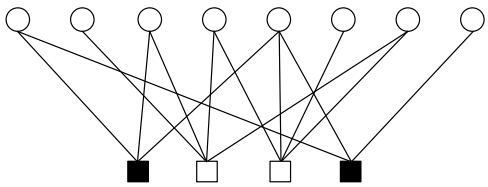


Figure 1.7: Subgraph corresponding to a $(8, 2)$ stopping set

An *absorbing set* [49] is a special type of (a, b) trapping set that is stable under the bit-flipping decoding. For a given subset S of variable nodes, let $\mathcal{E}(S)$ (resp. $\mathcal{O}(S)$) denote the set of neighboring checks of S that are even-degree (resp. odd-degree).

Definition 1.4. *An absorbing set is a subset of variable nodes S in the graph G, such that each variable node in S has strictly greater neighbors in $\mathcal{E}(S)$ than in $\mathcal{O}(S)$. In addition, if all remaining variable nodes in $V \setminus S$ also have strictly greater neighbors in $\mathcal{E}(S)$ than in $\mathcal{O}(S)$, then it is said to be a fully absorbing set.*

Another closely related type of trapping set is a *fixed set* that was introduced by Chilappagari and Vasić [50] to characterize failures of Gallager A/B decoding on the BSC. It is defined as follows.

Definition 1.5. *For a received vector $\mathbf{r}$ from the BSC and decoder input $\mathbf{y}$, a trapping set $\mathbf{T}(\mathbf{y})$ is said to be a fixed set, if $\mathbf{T}(\mathbf{y}) = \mathrm{supp}(\mathbf{y})$.*

The necessary and sufficient conditions for a set of variable nodes to form a fixed set for the Gallager A/B algorithm is provided through the following theorem [50].

Theorem 1.1. *Let $\mathcal{C}$ be an LDPC code with d_v-left-regular graph G. Let $\mathbf{T}$ be a subset of a variable nodes with induced subgraph $\mathcal{T}$. Let the checks in $\mathcal{T}$ be partitioned into two disjoint subsets; $\mathcal{O}$ consisting of checks with odd degree and $\mathcal{E}$ consisting of checks with even degree. Then $\mathcal{T}$ is a fixed set for the Gallager A/B algorithm iff : (a) Every variable node in $\mathcal{I}$ has at least $\lceil \frac{d_v}{2} \rceil$ neighbors in $\mathcal{E}$ and (b) No $\lfloor \frac{d_v}{2} \rfloor$ of $\mathcal{O}$ share a neighbor outside $\mathcal{I}$.*

Note that the definition of absorbing set satisfies the above theorem and therefore is always a fixed set; however the opposite is not necessarily true. For column-weight-three codes though, they are the same. For this dissertation, we shall mainly utilize the notion of trapping sets to characterize decoder failures, although stopping sets will come into play

towards the latter part of the dissertation. The notion of fixed set will used only mainly for discussing the trapping set ontology which will be introduced shortly.

Given an (a, b) TS, it is also important to determine a measure of relative *harmfulness* of the TS which is based on its underlying topological structure. Again for the BSC under the Gallager B decoding, Chilappagari and Vasic [48] introduced the notion of *critical number* as a measure of harmfulness. Given the subgraph corresponding to a TS is $\mathcal{T}(a, b)$, let $\mathbf{y}_{\mathcal{T}}$ denote the vector of channel values received by the variable nodes in $\mathcal{T}(a, b)$. Let $\mathbf{T}(\mathbf{y}_{\mathcal{T}})$ denote the set of nodes that failed to converge to the right values.

Definition 1.6. *The critical number n_c of a trapping set whose subgraph is $\mathcal{T}(a, b)$, is the minimum number of errors introduced in the subgraph $\mathcal{T}(a, b)$ that causes the Gallager-B decoder. More precisely,*

$$n_c = \min_{\mathbf{T}(\mathbf{y}_{\mathcal{T}}) \neq \emptyset} |\mathrm{supp}(\mathbf{y}_{\mathcal{T}})|$$

Note that the dominant trapping sets are the subgraphs with the least critical number. Although this notion is limited to Gallager-B decoding, later on, we will generalize this notion to include other decoding algorithms as well.

Henceforth, for convenience, whenever we refer to a TS, we shall implicitly refer to its underlying topological structure, which is a subgraph induced by the variable nodes belonging to the TS.

1.3.3 Trapping set ontology

The trapping set ontology (TSO) [62] is a database of trapping sets that is organized as a hierarchy based on their topological relations. This topological relationship between trapping sets is specified in the form of a *parent-child* relationship. A trapping set $\mathcal{T}_1$ is said to be a *parent* of a trapping set $\mathcal{T}_2$ if $\mathcal{T}_2$ contains $\mathcal{T}_1$, and $\mathcal{T}_2$ is then considered to be a *child* of $\mathcal{T}_1$. The main objective for using the TSO is to make the identification of relevant trapping sets independent of a given code, as well as to serve as a guide for code construction or decoder design in addition to enabling efficient enumeration techniques.

The development of TSO was primarily motivated by the work of Chilappagari *et al.* in [63], who observed that the trapping sets found for various iterative decoders over different channels are closely related. For instance on a given code, many trapping sets found for the BP decoding over the AWGN were either the same as the trapping sets of Gallager B over BSC, or bigger subgraphs containing them. This implies that there exists a topological interrelation among trapping sets and in a broader sense, a topological interrelation among error patterns that cause decoding failures for various algorithms on different channels. Relying on this fact, the TSO is generated based on the notion of fixed sets for Gallager B, with the purpose of capturing these topological relations in order to provide a hierarchy.

The topological relations can be more conveniently established using an alternate graphical representation of trapping sets, which is based on the incidence structure of lines and points. In this representation, variable nodes correspond to lines and check nodes correspond to points. A point is shaded black if it has odd number of lines passing through it otherwise it is shaded white. An (a, b) trapping set is thus an incidence structure with a lines and b black points. Fig. 1.8 shows an example of a $(5, 3)$ TS in the two graphical representations. In order to avoid confusion, note that the Tanner graph representation always uses □ or ■ to depict check nodes.

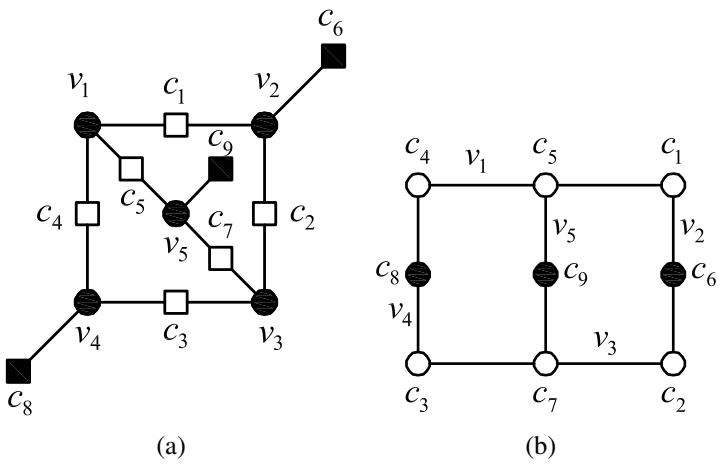


Figure 1.8: Graphical representation of $(5, 3)$ trapping set: (a) Tanner graph representation; (b) Line and point representation.

Since the necessary and sufficient conditions for a fixed set of Gallager B are clearly defined through Theorem 1.1, this notion is used to generate the TSO. Also it is well-known that, in general, trapping sets are subgraphs formed by cycles or union of cycles [46]. Therefore, assuming that any code considered has at least girth g and left degree d_v, the TSO is generated as follows. Starting with a g-cycle, variable nodes of d_v degree are added to the g-cycle in all possible ways thereby expanding the g-cycle to a chosen maximum size, while enforcing the constraint that the resulting subgraph after each addition of a variable node, is elementary and satisfies Theorem 1.1. Once all different expanded subgraphs are generated, a hierarchy is then established based on their parent-child relationship. The procedure is then repeated starting with a $(g + 2)$-cycle, and so on.

In the line and point representation, the method for expanding the subgraphs reduces to adding additional lines to a parent subgraph along with changing the color of the points accordingly such that the resulting subgraph satisfies the constraints. Fig 1.9 illustrates the TSO generated for column-weight-three codes starting from cycles of lengths 6, 8, 10, and 12. Note that although the methodology is restricted by using Theorem 1.1 for

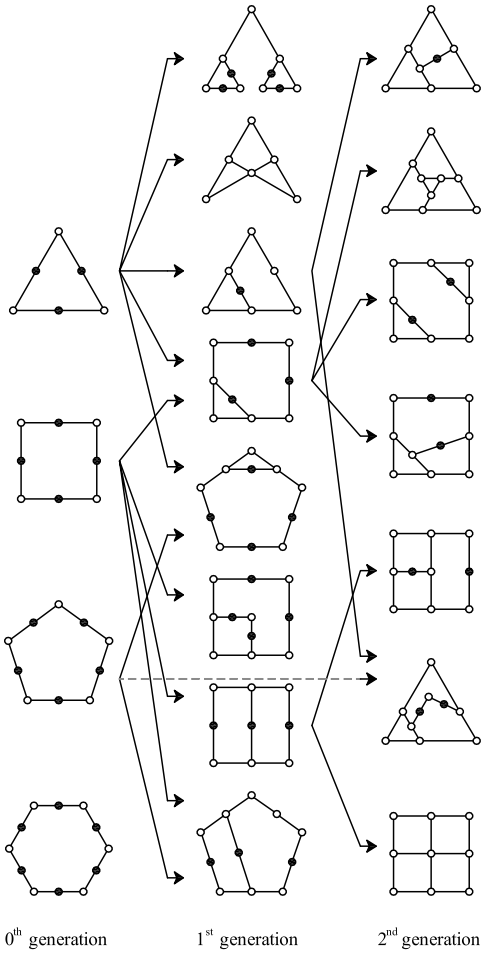


Figure 1.9: Trapping set ontology for column-weight-three codes and Gallager A algorithm.

fixed sets, it can be easily generalized for generating other types of subgraphs, provided that the constraints or desired properties of the subgraph are clearly specified. Moreover, as previously pointed out, since failures of other iterative decoders over channels other than BSC are also topologically linked to failures of Gallager B decoding on the BSC, we regard the subgraphs in the TSO as a good starting point for consideration in decoder design or code construction.

1.3.4 Guaranteed error correction

The guaranteed error correction capability of a code has always been an important parameter especially for applications such as magnetic recording, and optical and solid-state

storage. Recall that under ML decoding, a linear block code with minimum distance d_{min}, can achieve a guaranteed error correction of $\lfloor (d_{min}-1)/2 \rfloor$. However, for an LDPC code under iterative decoding over the BSC, its guaranteed error correction is highly non-trivial to determine since iterative decoding is suboptimal and the guaranteed error correction cannot be simply derived from the d_{min} of the code. An LDPC code $\mathcal{C}$ is said to have to have a guaranteed error correction of t under a particular decoding algorithm if it can correct all error patterns of weight t or less.

In the context of the error floor problem, the guaranteed error correction capability has a direct impact on the error floor performance of a code. This relationship was first established by Ivkovic *et al.* [64] who showed that on the BSC, the slope of the error floor is governed by the guaranteed error correction capability of the code under a particular decoding algorithm. We now describe this relationship as provided in [64].

Let α denote the cross-over probability of the BSC. Let t be the guaranteed error correction capability of an (N, K) LDPC code $\mathcal{C}$, and let n_i be the number of received vectors of weight i, that lead to a frame (codeword) error. Then the frame error rate $\mathrm{FER}(\alpha)$ can be determined as

$$\mathrm{FER}(\alpha) = \sum_{i=t+1}^{N} n_i(\alpha)^i(1-\alpha)^{(N-i)}$$

where $t+1$ is the minimal number of errors required to cause a decoder failure.

On a semilog scale, the FER is given by

$$\begin{aligned}
\log(\mathrm{FER}(\alpha)) &= \log\left(\sum_{i=t+1}^{N} n_i(\alpha)^i(1-\alpha)^{(N-i)}\right) \\
&= \log(n_{t+1}) + (t+1)\log(\alpha) + \log\left((1-\alpha)^{(N-(t+1))}\right) \\
&+ \log\left(1 + \frac{n_{t+2}}{n_{t+1}}\alpha(1-\alpha)^{-1} + \ldots + \frac{n_N}{n_{t+1}}\alpha^{(N-(t+1))}(1-\alpha)^{-(t+1)}\right)
\end{aligned}$$

For a small α, the expression is dominated by the first two terms thereby reducing to

$$\log(\mathrm{FER}(\alpha)) \approx \log(n_{t+1}) + (t+1)\log(\alpha)$$

From the above expression, we can see that on a $\log(\mathrm{FER})$ vs $\log(\alpha)$ scale, the slope of the error floor (which occurs at small α) is governed by the guaranteed error correction capability t.

It is clear from our discussions that the guaranteed error correction capability is important to consider for code/decoder designs on the BSC to improve the error floor performance. For the Gallager B decoding, Chilappagari *et al.* derived conditions linking the guaranteed error correction capability with the girth of the code [65], and also derived ex-

plicit conditions on the graph that guaranteed a correction of three errors [66]. However, for other soft decoding algorithms on the LDPC codes, no such results are known as it is much more difficult to obtain.

Achieving an increased guaranteed error correction capability is one of the aims of the work in this dissertation which is taken into consideration in the decoder design. We will expound more on this in the next chapter.

1.4 Motivation For This Dissertation

The error floor problem of LDPC codes under BP decoding has become a critical issue over the past several years especially in the industrial community with applications now becoming much more power-constrained while also requiring much lower achievable error-rates. It was pointed out in the last section that the error floor phenomenon arises from the sub-optimality of BP on finite-length codes whose graphs have loops, and the achievable performance of BP is far from the optimal performance achieved by ML decoding.

Since BP is an inference algorithm that is exact only for cycle-free graphs, there has been some research efforts to devise algorithms providing better approximates of the marginals than BP, by taking into account the presence of certain loopy structures into the inference problem. Some of the notable works in this direction include the generalized BP algorithms by Yedidia *et al.* [67], loop calculus methods by Chertkov *et al.* [97], and tree-pruning decoding by Lu *et al.* [70]. While all these approaches, which are based in statistical physics, are strongly supported by theoretical justifications, they are too complex to find their way into practical decoder realizations.

There have also been some significant works that address the error floor problem from a code construction perspective. These methods involve avoiding certain harmful loopy structures during code construction rather than optimizing for the d_{min} of the code. One of the first works in this direction was by Tian *et al.* [69], who introduced the notion of extrinsic message degree to selectively avoid cycles during the construction of irregular codes. Ivkovic *et al.* [64] proposed the technique of edge swapping between two graph covers of an LDPC code, and this was later generalized to cyclic liftings of higher order by Asavadi *et al.* [71] for constructing quasi-cyclic codes. More recently, Nguyen *et al.* [72] proposed a construction method based on Latin squares that avoids specific harmful trapping sets identified by the TSO. While all these methods have successfully constructed codes with lowered error floors, their central idea of avoiding loopy structures during construction becomes very difficult to maintain when the code rate is relatively high as graphs become denser. Moreover, the effects of finite precision that get introduced when decoders are realized in hardware, are crucial aspects ignored during code construction which can still deteriorate the error-rate performance.

Therefore, the design of reduced-complexity finite precision iterative decoders is of

prime importance for practical coding systems. A glimpse at the iterative decoders developed so far reveals a wide range of decoders of varying complexity. The simple binary message passing algorithms such as the Gallager A/B algorithms occupy one end of the spectrum, while the BP lies at the other end of the spectrum. The gamut of decoders filling the intermediate space can simply be understood as the implementation of the BP (and variants) at different levels of precision.

Early works on the design of quantized BP decoders include the Gallager-E and other quantized decoders proposed by Richardson and Urbanke [19], and reduced-complexity BP decoders such as the normalized min-sum and offset min-sum decoders proposed by Chen *et al.* [20] and by Fossorier *et al.* [21]. More recent works include the quantized BP decoders proposed by Lee and Thorpe [22], and also by Kurkosi and Yagi [73]. In all of the aforementioned works, the quantization schemes are designed based on optimizing for the best decoding threshold on a given code using the asymptotic technique of density evolution (DE). Another important point to note in the context of this dissertation, is that these decoders were all designed with the primary goal of approaching the performance of the floating-point BP algorithm, as a performance loss is typically associated with quantization of messages. Since asymptotic methods are inapplicable to finite length codes, these decoders designed for the best DE thresholds do not guarantee a good performance on a finite length code especially in the high SNR region.

There have also been several works that have addressed the error problem from a decoding perspective by proposing modifications to the BP decoding algorithm. Some of the notable works include augmented BP by Varnica *et al.* [74], multilple-bases BP by Hehn *et al.* [75], informed dynamic scheduling by Casado *et al.* [76], multi-stage decoding by Wang *et al.* [77], averaged decoding by Laendner and Milenkovic [78], and use of post-processing to lower the error floors by Han and Ryan [79] and also by Zhang *et al.* [80]. While all these schemes certainly provided performance enhancements in the error floor, all of them require either a considerable increase in decoding complexity due to the modifications and post-processing or are restricted to a particular code whose structure is well-known. In addition, they do not take finite precision into account and this can drastically affect the performance gains when implemented in hardware due to possible numerical precision issues.

The fact that message quantization can further contribute to the error floor phenomenon was first observed by Richardson [46], and later Zhang *et al.* [81] also studied the effects of quantization on the error floor behavior of hardware-implemented BP over the AWGNC. More recently, Butler and Siegel [82] showed that the error floor on the AWGNC was greatly affected by the numerical saturations arising from the quantization of the messages, and the error floor performance improved when there was limited or no saturation in the messages.

Therefore, addressing the error floor problem while also taking finite-precision into account in the decoder design is critically important especially with emerging applications in communication and data storage systems now requiring very low error-rates,

faster processing speeds, lower memory usage, and reduced chip area. In this regard, there are some relevant works worth mentioning. Zhao *et al.* in [83] proposed several modifications to the offset min-sum decoder while taking quantization into account. Their proposed quantization schemes enabled their offset-min sum decoder to approach performance of floating-point BP algorithm on the Additive White Gaussian Noise channel (AWGNC) with six bits of quantization with possible improvement in the error floor. Zhang *et al.* in [81] also studied the effects of quantization on the error floors of BP over the AWGNC, and designed schemes that led to substantial error floor reductions when six bits of quantization were used. More recently, Zhang and Siegel [84, 85] proposed a quasi-uniform quantization scheme for various BP-based decoders over the BSC as well as the AWGNC. On the BSC, they were able match the performance of different types of floating-point min-sum decoders with 4 bits of quantization, while they required at least 6 bits of quantization to match the performance of floating-point BP.

As the title suggests, this dissertation is mainly concerned with the development of novel finite precision iterative decoding algorithms, with "beyond belief propagation" implying that the approach we propose is radically different from conventional BP-based approaches. Having established the context of the problem, the main goals of this dissertation can be summarized as: 1) to surpass the error floor performance of conventional floating-point BP decoder at much lower complexity while taking finite precision into account, and 2) to enhance the guaranteed correction ability of a code and reduce the performance gap between iterative decoding and ML decoding in the error floor.

With regards to the first goal, we introduce a novel approach to the design of finite precision iterative decoders on the BSC, which we refer to as *finite alphabet iterative decoders* (FAIDs) to signify the fact that the messages belong to a finite alphabet. Some of the key features that clearly distinguish our approach from all other previously mentioned works on finite precision iterative decoders are: 1) the messages are not quantized values of log-likelihoods or probabilities, and 2) the variable node update functions are simple well-defined maps rather than approximations of the update functions used in BP. The maps for variable node update in FAIDs are designed with the goal of increasing the guaranteed error-correction capability by using the knowledge of potentially harmful subgraphs that could be present in any given code, thereby improving the slope of the error floor on the BSC. Since the variable nodes in the proposed decoders are now equipped to deal with potentially harmful neighborhoods, which is in contrast to BP which treats the loopy Tanner graph as a tree, the proposed decoders are capable of surpassing the floating-point BP in the error floor. In fact, we will show that there are 3-bit precision FAIDs capable of surpassing BP in the error floor.

For this work, we restrict our focus on the design of FAIDs for column-weight-three codes. The main reason for this is that such codes, while enabling extremely simple hardware implementations, are notoriously prone to higher error floors especially at moderate to high code rates compared to codes of higher column-weights. Being able to design good yet simple decoders for these codes not only validates our novel approach but also

further ascertains the importance of addressing the error problem from the viewpoint of improving the iterative decoding rather than from the viewpoint of code construction, as it becomes very difficult to construct such codes to be devoid of certain harmful loopy graphs or with large girth, without compromise on the code rate and length. Moreover, the trapping sets of various iterative decoders are more well-known for column-weight-three codes [62], and can be exploited in the design of FAIDs.

With regards to the second goal, we propose *decimation-enhanced finite alphabet iterative decoders* for LDPC codes on the BSC. The technique of decimation which is incorporated into the message update rule of FAIDs, involves fixing certain codeword bits to a particular value during message-passing. We first propose a simple decimation scheme for a 3-bit precision FAID that allows the decoder to be much more amenable to analysis, significantly reduces the number of iterations required by it to correct a fixed number of errors. We will then show how decimation can be used adaptively to further enhance the guaranteed error correction capability of FAIDs that are already good on a given code, with only marginal increase in complexity. The adaptive decimation scheme is particularly for a 3-bit precision FAIDs which propagate only 3-bit messages. and we show that the guaranteed error correction capability can be significantly enhanced, with possibility of approaching the theoretical limits achieved by ML decoding on certain codes.

1.5 Contributions and Outline

The main contributions of this dissertation along with the outline are provided below.

- In Chapter 2, we introduce a new paradigm for finite precision iterative decoding on the BSC. The proposed novel decoders known as finite alphabet iterative decoders FAIDs are capable of surpassing the floating-point BP in the error floor. An alternate representation in the form of symmetric plane partitions is provided that allows for easy enumeration of the total number of possible FAIDs. We show that there exist 3-bit precision FAIDs for column-weight-three codes that can outperform the floating-point BP algorithm in the error floor without any compromise in decoding latency. Hence, the proposed decoders provide a superior performance with only a fraction of the implementation complexity of BP. Numerical results are provided on several codes with different rates and lengths to validate this fact. We also show that FAIDs designed solely based on optimizing for the best possible asymptotic decoding threshold on a given code are not the best decoders and in some cases can give rise to high error floors. We also provide a semi-heuristic-based selection method that is not code specific but which gives a set of candidate FAIDs that contains potentially good decoders in the error floor for any given column-weight-three code.

- In Chapter 3, we propose decimation-enhanced FAIDs (DFAIDs) on the BSC. The technique of decimation is introduced and incorporated into FAIDs during message-passing in order to make them more amenable to analysis. We illustrate this with a simple decimation scheme for a 3-bit precision FAID and show how it can be analyzed in terms of guaranteed error-correction capability. We also show how decimation can be used adaptively to further enhance the guaranteed error correction capability of a FAID that is already known to surpass BP. We provide insights into the design of the decimation scheme and show how failures of DFAIDs are linked to stopping sets. Using this approach, we provide the adaptive decimation schemes with 3-bit precision FAIDs for two column-weight-three codes, and show that not only can the guaranteed error correction be significantly increased but also possibly approach the limits achieved by ML decoding.

The work in this dissertation has led to the following publications.

Journals

1. **S. K. Planjery**, D. Declercq, B. Vasić, and L. Danjean, "Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders," *IET Electronics Letters*, vol. 46, no. 16, Aug. 2011.

2. **S. K. Planjery**, D. Declercq, L. Danjean, and B. Vasić,"Finite alphabet iterative decoders, Part I: Decoding beyond belief propagation on the BSC," *submitted to IEEE Trans. Comm.*, Jul. 2012.

3. D. Declercq, B. Vasić, **S. K. Planjery**, and E. Li, "Finite alphabet iterative decoders, Part II: Improving the guaranteed error correction of LDPC codes via iterative decoder diversity," *submitted to IEEE Trans. Comm.*, Jul. 2012.

4. **S. K. Planjery**, B. Vasić, and D. Declercq, "Decimation-enhanced finite alphabet iterative decoders," *under preparation for submission*

Conferences

1. **S. K. Planjery**, S. K. Chilappagari, B. Vasić, D. Declercq, and L. Danjean,"Iterative decoding beyond belief propagation," *Proc. of IEEE Inf Theory and App. Workshop (ITA'10)*, San Diego, CA, Jan. 2010 (invited).

2. **S. K. Planjery**, D. Declercq, S. K. Chilappagari, B. Vasić, "Multilevel decoders surpassing belief propagation over the binary symmetric channel" *Proc. of IEEE Int. Symp. Inf. theory (ISIT'10)*, Austin, TX, Jun. 2010.

3. D. Declercq, L. Danjean, **S. K. Planjery**, B. Vasić, and E. Li, "Finite alphabet iterative decoding (FAID) of the (155,64,20) Tanner code," in *Proc. IEEE Int. Symp. Turbo codes*, Brest, France, Sept. 2010 (invited).

4. **S. K. Planjery**, B. Vasić, D. Declercq, and M. Marcellin, "Low-complexity finite-precision decoders for LDPC codes," *Proc. Int. conf. telemetering (ITC'10)*, San Diego, CA, Oct. 2010.

5. **S. K. Planjery**, B. Vasić, and D. Declercq, "Decimation-enhanced finite alphabet iterative decoders for LDPC codes on the BSC," *Proc. of IEEE Int. Symp. Inf. theory (ISIT'11)*, St. Petersburg, Russia, Aug. 2011.

6. L. Danjean, D. Declercq, **S. K. Planjery**, and B. Vasić, "On the selection of finite alphabet iterative decoders for LDPC codes on the BSC," *Proc. of IEEE Int. Inf. Theory Workshop (ITW'11)*, Paraty, Brazil, Oct. 2011

7. **S. K. Planjery**, B. Vasić, and D. Declercq, "Enhancing the error correction of finite alphabet iterative decoders via adaptive decimation," *in Proc. IEEE Int. Symp. Inform. Theory (ISIT'12)*, Boston, MA, Jul. 2012.

8. D. Declercq, B. Vasić, **S. K. Planjery**, and E. Li, "Finite alphabet iterative decoders approaching maximum likelihood decoding on the Binary Symmetric channel," *Proc. of IEEE Inf. theory and app. workshop (ITA'12)*, San Diego, CA, Feb. 2012 (invited).

French National Conferences

1. L. Danjean, D. Declercq, **S. K. Planjery** and B. Vasić, "Trapping sets bruités pour la sélection des décodeurs multi-niveaux des codes LDPC", *Proc. of GRETSI 2011*, Bordeaux, France, Oct. 2011

2. L. Danjean, D. Declercq, **S. K. Planjery**, and B. Vasić, "Décodeurs multi-niveaux pour le décodage quasi-optimal des codes LDPC," . *Proc. MajecSTIC2010*, Bordeaux, France, Oct. 2010.

Patents

S. K. Planjery, S. K. Chilappagari, B. Vasic, and D. Declercq, "Low-complexity finite precision decoders and apparatus for LDPC codes," US Patent Application 12/900584, filed in Oct. 2010.

Chapter 2

Decoding Beyond Belief Propagation

PREVIOUSLY, we pointed out that BP decoding is optimal only on codes whose graphs are cycle-free. The BP algorithm and all its variants operate on the loopy graphs of finite-length LDPC codes, with the underlying principle that the graph is assumed to be a tree, and the topological neighborhood of a node is completely ignored when performing local computations for message update at the node at every iteration. Although this principle enables efficient implementations and BP still performs exceptionally well on loopy graphs in the low SNR region, the performance in the high SNR region is affected, where the error floor occurs. Depending on the particular neighborhood of a node, this could greatly influence the decoding trajectory, and cause BP to fail for certain low-weight error patterns leading to a poor guaranteed error correction capability and an error floor with a low slope. This is especially true for high-rate codes whose graphs are relatively dense.

In this chapter, we provide a new approach to finite precision iterative decoding of LDPC codes for the BSC, where the propagated messages belong to a finite alphabet. In contrast to the underlying principle of BP-based decoders (quantized), the proposed *finite alphabet iterative decoders* (FAIDs) do not propagate probabilistic (quantized) messages, and the variable node update functions do not mimic BP. Instead, the variable node update functions used are simple maps that are inherently designed to deal with potentially harmful neighborhoods while still maintaining the local nature of message updating. Hence, these decoders are simpler and different from conventional quantized decoders. Moreover, these finite precision decoders remarkably have the capability to surpass even the floating-point precision BP in the error floor. Using our approach, we will show that there exist 3-bit precision FAIDs that outperform the floating-point precision BP in the error floor without any compromise on latency.

We start by providing a simple motivating example to give some insights into the rationale behind our approach. We will then formally introduce the general framework of FAIDs highlighting their key differences compared to existing finite precision decoders. Subsequently, we will describe a methodology for selection of particularly good FAIDs

for column-weight-three codes, and provide 3-bit precision FAIDs that outperform the floating-point BP in the error floor on a variety of codes with different rates and lengths.

2.1 A Motivating Example: 2-bit Decoder

In this section, we discuss the motivation for our approach using a simple 2-bit message-passing decoder. The example we use involves analyzing the message-passing on a 3-error pattern associated with a six-cycle for which the Gallger B decoder fails to correct. By adding an extra bit in the message, and appropriately choosing the update function, the error pattern can be corrected.

The purpose of this example is to illustrate how the update function can be chosen to correct certain error patterns associated with a harmful subgraph, which will later form the basic essence for the design of FAIDs capable of surpassing BP in the error floor. Note that for this example of six-cycle, the 2-bit decoder employed is a FAID with message alphabet $\mathcal{M} = \{00, 01, 10, 11\}$.

Consider decoding on a column-weight-three code with a graph G containing a six-cycle over the BSC. For the message passing in this example, $\mathbf{y} = \mathbf{r}$ which implies that $\mathcal{Y} = \{0, 1\}$. Recall that V denotes the set of variable nodes and C denote the set of check nodes in G. Under all-zero codeword transmission, assume that the BSC introduced three errors such that the subgraph induced by the three variable nodes in error is a six cycle as shown in Fig. 2.1. Let $V_1 = \{v_1, v_2, v_3\}$ denote the set of variable nodes in the six cycle and let $C_1 = \{c_1, c_2, c_3\}$ and $C_2 = \{c_4, c_5, c_6\}$ denote the set of degree-two check nodes and degree-one check nodes in the subgraph respectively. Assume that no two check nodes in C_2 are connected to a common variable node in $V \backslash V_1$. Note that for figures throughout, we shall use ●/○ to represent a variable node that is initially incorrect/correct, and a ■/□ to represent an odd/even degree check node.

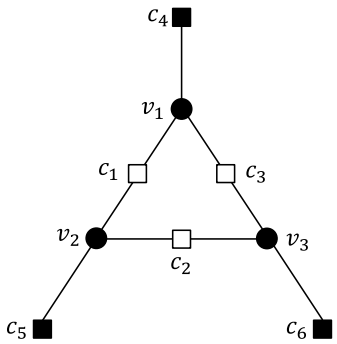
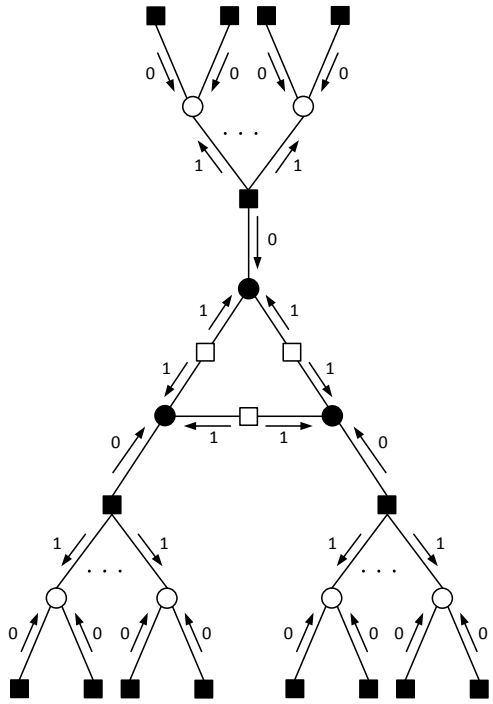
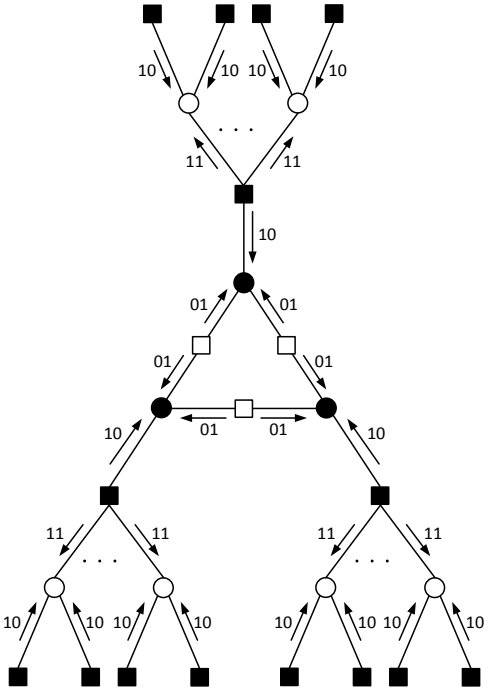


Figure 2.1: A six-cycle present in the Tanner graph, where the three variable nodes are initially in error.



(a) The messages as defined by Gallager B algorithm. The decoder is blind to the presence of a six-cycle, and cannot correct the above error configuration.



(b) Two-bit messages indicating the involvement of a variable node in a six-cycle.

Figure 2.2: Comparison of message-passing between Gallger B decoder and 2-bit decoder on a six-cycle.

2.1.1 Gallager B decoder fails

The outgoing message at the variable node in the Gallager B algorithm is calculated as the majority of the incoming two messages and the received value. The outgoing message at the check node is simply the XOR of all the incoming extrinsic messages. Fig. 2.2(a) illustrates how messages are passed under Gallager B decoding. Note that for Fig. 2.2, the check node degree is arbitrary but only significant connections are shown. At the end of every iteration, all variable node neighbors of C_2 receive two message that coincide with their received value and one message that disagrees with the received value. Since, the neighborhood of C_2 consists of both correct and corrupt variable nodes, the algorithm cannot distinguish between the two based solely on the incoming messages. The variable nodes initially in error remain in error and all the other variable nodes remain uncorrupted. One realizes that this situation primarily arises due to the circulation of incorrect messages in the six cycle, but unfortunately the decoder cannot detect this scenario.

2.1.2 2-bit decoder succeeds

Now consider adding one more bit to represent the messages in the decoder. Let the second bit (right-most bit) be used to denote the estimated bit value of the variable node, and the first bit (left-most bit) to denoting a strength (or confidence) of the estimated bit value with 0 indicating a *weak* and 1 indicating a *strong* bit value. In the first iteration, assume that nodes v_1, v_2, and v_3 send '01' (weak one), and all nodes in $V \backslash V_1$ send '00' (weak zero) as their outgoing messages. At the check node, the first bit of the outgoing message is the AND of all the first bits of the incoming messages. The second bit is simply the XOR of all the second bits in the incoming messages. In the second iteration at the variable node, the first bit of the outgoing message is set to 1 if the second bit of the two incoming messages agree with the received value; else, it is set to 0. The second bit is computed by taking the majority of the second bits of the incoming messages along with the received value (as in Gallager B). It can be seen that at the end of the second iteration, the nodes in V_1 receive two messages as '01' (weak 1) and one message as '10' (strong zero) (Fig. 2.2(b)). By receiving such messages after two iterations, the node in error now can clearly infer that it is possibly in a troublesome neighborhood.

Let $\mathcal{N}(U)$ denote the set of the neighbors of all nodes in a set U. Additionally, let $V_2 = \mathcal{N}(C_2) \backslash V_1$ and $C_3 = \mathcal{N}(V_2) \backslash C_2$ and assume no two checks in C_3 and C_1 share a common variable node in $V \backslash (V_1 \cup V_2)$. Under such a scenario, if the variable node update function is chosen so that $\Phi_v(1, 10, 01) = 00$, and $\Phi_v(1, 00, 10) = 10$, then decoder is guaranteed to correct the 3-error pattern (with a suitable decision rule). The complete specification for Φ_v of the 2-bit decoder is given below in the form of a Boolean map, where m_o denotes the outgoing message.

m_1	m_2	r_i	m_o
00	00	0	10
00	00	1	00
00	10	0	10
00	10	1	10
00	01	0	00
00	01	1	01
00	11	0	01
00	11	1	01
10	10	0	10
10	10	1	10

m_1	m_2	r_i	m_o
10	01	0	00
10	01	1	00
10	11	0	00
10	11	1	01
01	01	0	01
01	01	1	11
01	11	0	11
01	11	1	11
11	11	0	11
11	11	1	11

Table 2.1: Boolean map used at the variable node for the 2-bit decoder

We just illustrated how one can utilize the knowledge of potentially harmful subgraphs to design an update function for the 2-bit decoder. More examples related to this discus-

sion can be found in our paper [86]. Although this was a simple example, the underlying philosophy used, which was to choose an update function that increases the correction ability of error patterns on harmful subgraphs, will later form the basis of our approach for the design of FAIDs with larger alphabets that are capable of surpassing BP in the error floor.

At this point in the context of 2-bit decoders, it is worth mentioning the work of Sassatelli *et al.* [87], which can be regarded as a precursor to the present work. They proposed two-bit message-passing decoders for column-weight-four codes on the BSC that provided performance improvements compared to Gallager B decoding, and they also derived conditions to guarantee correction of 3 errors for such codes. In contrast, our approach involves designing FAIDs with alphabet sizes larger than four, that have the ability to surpass the BP in the error floor. Note that although messages were represented as binary vectors in the above example, for the next section we shall simply represent the messages as levels belonging to a finite alphabet.

2.2 Finite Alphabet Iterative Decoders

We now introduce the general framework for a new class of finite precision iterative decoders referred to as multilevel FAIDs [88–90] for LDPC codes. For this class, an N_s-level FAID denoted by $\mathscr{F}$ is defined as a 4-tuple given by $\mathscr{F} = (\mathcal{M}, \mathcal{Y}, \Phi_v, \Phi_c)$. The finite alphabet $\mathcal{M}$ defined as $\mathcal{M} = \{-L_s, \ldots, -L_1, 0, L_1, \ldots, L_s\}$, where $L_i \in \mathbb{R}^+$ and $L_i > L_j$ for any $i > j$, consists of $N_s = 2s + 1$ levels for which the message values are confined to. The sign of a message $x \in \mathcal{M}$ can be interpreted as the estimate of the bit associated with the variable node for which x is being passed to or from (positive for zero and negative for one), and the magnitude $|x|$ as a measure of how reliable this value is.

The set $\mathcal{Y}$ which denotes the set of possible channel values is defined for the case of BSC as $\mathcal{Y} = \{\pm \mathrm{C}\}$, and the channel value $y_i \in \mathcal{Y}$ corresponding to node v_i is determined by $y_i = (-1)^{r_i}\mathrm{C}$, i.e., we use the mapping $0 \to \mathrm{C}$ and $1 \to -\mathrm{C}$. The value of C can be considered as a parameter used to control how much of an effect the channel value should have on the output of Φ_v.

Let $m_1, \cdots, m_{l-1}$ denote the extrinsic incoming messages to a node with degree l.

2.2.1 Definitions of the update functions Φ_v and Φ_c

The update function $\Phi_c : \mathcal{M}^{d_c-1} \to \mathcal{M}$ used at a check node with degree d_c is defined exactly as in Eq. (1.4) for min-sum decoder, which is provided below for convenience

$$\Phi_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \operatorname{sgn}(m_j) \right) \min_{j \in \{1,\ldots,d_c-1\}} (|m_j|).$$

Note that since this is the same function used in the min-sum decoder, the novelty in the proposed FAIDs lies in the definition of the variable node update function Φ_v. Consequently, a particular Φ_v also defines a particular FAID, as Φ_c is the same in all FAIDs considered.

The update function $\Phi_v : \mathcal{Y} \times \mathcal{M}^{d_v-1} \to \mathcal{M}$ used at a variable node with degree d_v, can be described either as a closed-form function or simply as a map that is a look-up table (LUT). The closed form definition is useful for understanding FAIDs in the context of the existing framework of quantized BP decoders. In closed-form, Φ_v is defined as

$$\Phi_v(m_1, m_2, \cdots, m_{d_v-1}, y_i) = Q\left(\sum_{j=1}^{d_v-1} m_j + \omega_i \cdot y_i\right) \tag{2.1}$$

where the function $Q(.)$ is defined as follows based on a threshold set $\mathscr{T} = \{T_i : 1 \leq i \leq s+1\}$ such that $T_i \in \mathbb{R}^+$ and $T_i > T_j$ if $i > j$, and $T_{s+1} = \infty$.

$$Q(x) = \begin{cases} \text{sgn}(x)L_i, & \text{if } T_i \leq |x| < T_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

The channel weight ω_i assigned to the channel value while determining the output in Eq. (2.1), is computed using a symmetric function $\Omega : \mathcal{M}^{d_v-1} \to \{0, 1\}$, which is a function of the extrinsic incoming messages to a particular node.

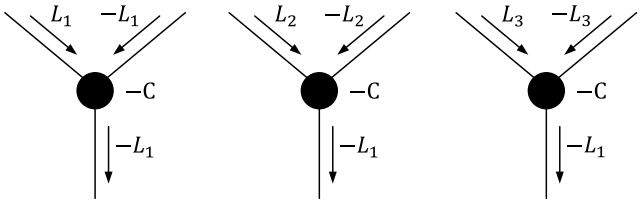
Based on this definition, the function Φ_v can be classified as two types: a linear-threshold (LT) function and a non-linear-threshold (NLT) function. If $\Omega = 1$ (or constant), i.e., if the value of ω_i is always 1 (or constant) for all possible inputs of Ω, then Φ_v is an LT function and a FAID with such a Φ_v is classified as an LT FAID. Else, Φ_v is an NLT function and a FAID with such a Φ_v is classified as an NLT FAID.

Note that if Φ_v is an LT function, this means that it simply takes a linear combination of its arguments and then applies the thresholding function Q to determine its final output. An important characteristic to note for an LT function is that Φ_v will always output the same value for any two distinct ssets of incoming messages as long as their respective sums along with the channel value are the same. For instance, for a node with $d_v = 3$, $\Phi_v(-\text{C}, m_1, m_2) = \Phi_v(-\text{C}, m_3, m_4)$ when $m_1 + m_2 = m_3 + m_4$. This is also a typical property present in all existing quantized message-passing decoders such as quantized BP and min-sum.

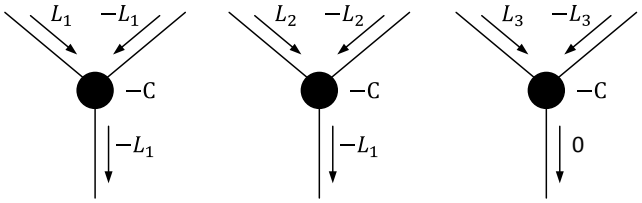
On the other hand, for an NLT FAID, due to the definition of Ω, Φ_v can output different outgoing messages even for distinct sets of incoming messages that lead to the same sum (upon which the function Q is applied to). In other words, Φ_v takes a non-linear combination of its arguments before applying the function Q on the result.

Fig. 2.3 illustrates an example on a variable node with $d_v = 3$ that highlights this difference for three different instances of message passing. The sets of incoming messages considered for the three instances are $\{-L_1, L_1\}$, $\{-L_2, L_2\}$, and $\{-L_3, L_3\}$ with

the channel value being $-\mathrm{C}$. Note that for all three instances, the sum of the incoming messages along with the channel value are the same and equal to $-\mathrm{C}$. Therefore, the output of the Φ_v for an LT function must be the same in all three instances. In Fig. 2.3(a), the output of Φ_v is shown to be $-L_1$ for all three instances as $Q(-\mathrm{C}) = -L_1$ assuming $T_1 = -\mathrm{C}$. However, for an NLT function, the outputs of Φ_v can be different. In Fig. 2.3(b), $\Phi_v(-\mathrm{C}, L_1, -L_1) = -L_1$ and $\Phi_v(-\mathrm{C}, L_2, -L_2) = -L_1$, but $\Phi_v(-\mathrm{C}, L_3, -L_3) = 0$.



(a) Φ_v as an LT function: Assuming $T_1 = -\mathrm{C}$, $Q(-\mathrm{C}) = -L_1$, and therefore the outputs for the three instances must be the same and equal to $-L_1$.



(b) Φ_v as an NLT function: The output of Φ_v is L_1 for two instances, and 0 for the third instance

Figure 2.3: Comparison of possible outputs of Φ_v between LT and NLT functions for three different instances of message-passing.

From the above example, it is evident that NLT FAIDs are different from all existing quantized message-passing decoders based on BP, while LT FAIDs can be regarded as a subclass of quantized min-sum decoders. However, even in the case of LT FAIDs, note their subtle difference in the fact that the messages are not quantized probabilities (or log-likelihoods) unlike quantized min-sum decoders.

One of the important properties that the function Φ_v satisfies is the property of it symmetry given below.

Definition 2.1 (Symmetry property)**.** *If the update function Φ_v of a FAID satisfies the following condition*

$$\Phi_v(y_i, m_1, ..., m_{d_v-1}) = -\Phi_v(-y_i, -m_1, ..., -m_{d_v-1}) \tag{2.2}$$

then the FAID is a symmetric decoder.

Based on the closed-form definition, Φ_v can be completely described either by assigning real values to the elements of $\mathcal{M}$, $\mathscr{T}$ and $\mathcal{Y}$, and defining Ω, or by providing a set of constraints which the assigned values can take so that it describes a particular unique map. As examples, we provide the closed-form description of Φ_v for a particularly good 5-level NLT FAID and a 7-level LT FAID defined for column-weight-three codes.

Example 2.1 (5-level NLT FAID)**.** *The constraints on the values assigned to elements of $\mathcal{M}$ and $\mathscr{T}$ that describe this map are: $\mathrm{C} = L_1$, $L_2 = 3L_1$, $T_1 = L_1$, $T_2 = L_2$, and the channel weight function Ω used to compute ω_i is given by*

$$\begin{aligned} \omega_i &= \Omega(m_1, m_2) \\ &= 1 - \Big(\mathrm{sign}(m_1) \oplus \mathrm{sign}(m_2)\Big) \cdot \delta(|m_1| + |m_2| - 2L_2) \end{aligned}$$

where $\mathrm{sign}(x) = 1$ if $x < 0$, and $\mathrm{sign}(x) = 0$ otherwise.

Example 2.2 (7-level LT FAID)**.** *The constraints on the values assigned to elements of $\mathcal{M}$ and $\mathscr{T}$ that describe this map are: $L_1 < \mathrm{C} < 2L_1$, $L_2 = 2L_1$, $L_3 = 2L_2 + \mathrm{C}$, and $T_1 = L_1$, $T_2 = L_2$, and $T_3 = L_3 - \mathrm{C}$, where $\Omega = 1$ since it is an LT function.*

Note that the decision rule used in FAIDs is the same as in BP and min-sum which was given in Eq. (1.1).

2.2.2 Describing the maps of Φ_v as arrays

Let us alternatively define $\mathcal{M}$ to be $\mathcal{M} = \{M_1, M_2, \cdots, M_{N_s}\}$ where $M_1 = -L_s$, $M_2 = -L_{s-1}, \cdots$, $M_s = -L_1$, $M_{s+1} = 0$, $M_{s+2} = L_2, \cdots$, $M_{N_s} = L_s$. Then, Φ_v can be defined using d_{v-1}-dimensional arrays or look-up tables (LUTs) rather than as closed-form functions, which enables simple implementations and also may be more convenient for decoder selection. We now restrict our discussion to column-weight-three codes for the remainder of this chapter.

For column-weight-three codes, the map specifying Φ_v is a simple two-dimensional array defined by $[l_{i,j}]_{1 \le i \le N_s, 1 \le j \le N_s}$, where $l_{i,j} \in \mathcal{M}$, such that $\Phi_v(-\mathrm{C}, M_i, M_j) = l_{i,j}$ for any $M_i, M_j \in \mathcal{M}$. The values for $\Phi_v(\mathrm{C}, M_i, M_j)$ can be deduced from the symmetry of Φ_v.

Table 2.2 shows an example of a Φ_v defined as an array for a 7-level FAID when $y_i = -\mathrm{C}$.

It is easy to see that if the diagonal entries in the array $[l_{i,j}]_{1 \le i \le N_s, 1 \le j \le N_s}$ are different, then Φ_v must be an NLT function (based on the discussion in the previous subsection). However, Φ_v can still be a NLT function even if all the diagonal entries in the array are the same due to the remaining entries in the array as highlighted in the following lemma.

Lemma 2.1. *If $\Phi_v(-\mathrm{C}, -L_1, L_2) = -L_1$, $\Phi_v(-\mathrm{C}, 0, L_1) = 0$, $\Phi_v(-\mathrm{C}, 0, -L_2) = -L_3$, and $\Phi_v(-\mathrm{C}, -L_1, -L_1) = -L_2$, then Φ_v can not be expressed as a linear-threshold function, and hence it is a non-linear-threshold function.*

Table 2.2: LUT for Φ_v of a 7-level FAID with $y_i = -\mathrm{C}$

m_1/m_2	$-L_3$	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$	$+L_3$
$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_1$
$-L_2$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	$+L_1$
$-L_1$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	$-L_1$	$-L_1$	$+L_1$
0	$-L_3$	$-L_3$	$-L_2$	$-L_1$	0	0	$+L_1$
$+L_1$	$-L_3$	$-L_2$	$-L_1$	0	0	$+L_1$	$+L_2$
$+L_2$	$-L_3$	$-L_1$	$-L_1$	0	$+L_1$	$+L_1$	$+L_3$
$+L_3$	$-L_1$	$+L_1$	$+L_1$	$+L_1$	$+L_2$	$+L_3$	$+L_3$

Proof: Assume that there exists a LT representation for such a Φ_v which is defined by assigning real values to the elements of the message alphabet $\mathcal{M}$, the threshold set $\mathscr{T}$, and the set $\mathcal{Y}$. Since $\Phi_v(-\mathrm{C}, -L_1, L_2) = -L_1$, the following inequality must be satisfied

$$L_1 - L_2 + \mathrm{C} \geq T_1. \tag{2.3}$$

Also since $\Phi_v(-\mathrm{C}, 0, L_1) = 0$, we have

$$|\mathrm{C} - L_1| < T_1 \tag{2.4}$$

From Eq. (2.3) and Eq. (2.4), we get

$$L_1 - L_2 + \mathrm{C} > |\mathrm{C} - L_1| \tag{2.5}$$

If $\mathrm{C} > L_1$, then $2L_1 > L_2$. If $L_1 \geq \mathrm{C}$, then

$$L_1 - L_2 + \mathrm{C} > L_1 - \mathrm{C} \Rightarrow 2\mathrm{C} > L_2 \Rightarrow 2L_1 > L_2 \tag{2.6}$$

But since $\Phi_v(-\mathrm{C}, 0, -L_2) = -L_3$ and $\Phi_v(-\mathrm{C}, -L_1, -L_1) = -L_2$, we have

$$L_2 + \mathrm{C} > 2L_1 + \mathrm{C} \Rightarrow L_2 > 2L_1 \tag{2.7}$$

Clearly Eq. (2.7) contradicts Eq. (2.6), and therefore such a Φ_v is not an LT function. Hence, it is an NLT function ■

Consequently, the map defined by Table 2.2 is an NLT function. Fig. 2.4 shows the error-rate performances of the 5-level NLT and 7-level LT FAIDs defined in the two examples, and the 7-level NLT FAID defined by the Table 2.2, along with the floating-point BP and min-sum decoders on the well-known $R = 0.4129$ $(155, 64)$ Tanner code [27] which has girth-8 and $d_{min} = 20$. The maximum number of iterations allowed was 100 for all decoders. It is evident from the plot that all three FAIDs (which require only three bits of precision) significantly outperform the floating-point BP and min-sum decoders, with 7-level NLT FAID performing the best.

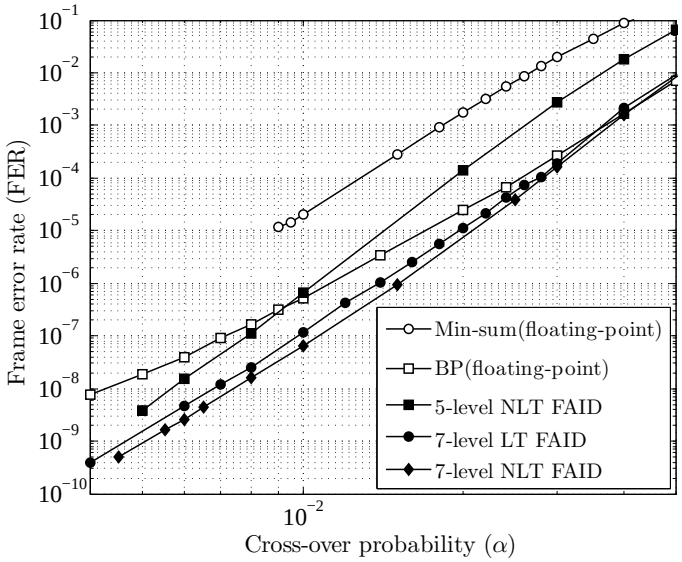


Figure 2.4: Performance comparisons between the floating-point decoders: BP and min-sum , and the 3-bit precision decoders: 5-level NLT, 7-level LT, and 7-level NLT FAIDs on the $(155, 64)$ Tanner code.

Note that a particular choice of $[l_{i,j}]_{1\leq i\leq N_s, 1\leq j\leq N_s}$ gives rise to a particular Φ_v. Therefore, it can be deduced from this representation that the total number of symmetric maps possible for Φ_v is $N_s^{\binom{N_s}{2}+N_s}$, which is a prohibitively large number for decoder selection. In order to restrict the number of FAIDs considered, we disregard the maps that do not exhibit a "typical" behavior observed in soft message passing decoders which is: for a given y_i at a node v_i, the value of the outgoing message is always non-decreasing with the increase in the values of the incoming messages. For instance, all FAIDs with maps Φ_v such that $\Phi_v(-\mathrm{C}, L_1, L_2) = L_2$ $\Phi_v(-\mathrm{C}, L_2, L_2) = L_1$ are forbidden, since $\Phi_v(-\mathrm{C}, L_2, L_2)$ should not be less than $\Phi_v(-\mathrm{C}, L_1, L_2)$. We incorporate this property into Φ_v by enforcing a constraint called *lexicographic ordering*.

Definition 2.2 (Lexicographic ordering)**.** *A FAID is said to be lexicographically ordered if* Φ_v *satisfies the following property.*

$$\Phi_v(-\mathrm{C}, m_1, \ldots, m_{d_v-1}) \geq \Phi_v(-\mathrm{C}, m'_1, \ldots, m'_{d_v-1}) \tag{2.8}$$

$\forall m_i \geq m'_i$ *where* $i \in \{1, \ldots, d_v - 1\}$

The ordering significantly reduces the number of FAIDs considered but still allows many good FAIDs to be included for possible selection.

Definition 2.3 (Class-A N_s-level FAID)**.** *A class-A* N_s*-level FAID is a symmetric lexico-*

graphically ordered FAID for column-weight-three codes.

Note that the three FAIDs described previously satisfy the lexicographic ordering property. Additional numerical results are also provided to demonstrate how a particular FAID (in this case the 7-level LT FAID) that is good on one code is capable of surpassing BP on several other codes as well. Fig. 2.5 and Fig. 2.6 show the performance comparisons between BP, min-sum, and the 7-level LT FAID on a $R = 0.75$ $(768, 576)$ quasi-cyclic code which has girth-8 and $d_{min} = 12$, and a $R = 0.82$ $(4095, 3358)$ random MacKay code which has girth-6, respectively.

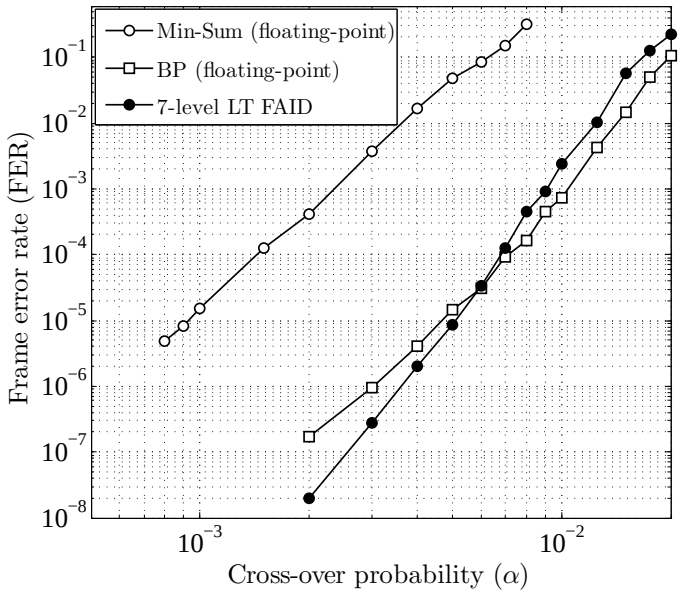


Figure 2.5: Performance comparisons between the floating-point decoders: BP and min-sum , and the 3-bit precision 7-level LT FAID on a $R = 0.75$ $(768, 576)$ quasi-cyclic code.

Clearly, the 7-level LT FAID surpasses BP and min-sum on both codes in the error floor. Although all the numerical results above show that the 7-level LT FAID (which was mainly provided to serve as an example) is particularly good, we have generally found that the best FAIDs for a given code are NLT FAIDs especially on codes that have larger code length and higher code rates. We also found that NLT FAIDs have a greater ability to surpass BP on a wider range of codes compared to LT FAIDs. We will subsequently explain the methodology used to identify good FAIDs which involves choosing the appropriate 2D array (or LUT) that defines Φ_v. But before we delve into this, a natural question that arises at this point is the number of class-A FAIDs that exist. This can be easily enumerated by establishing a connection between class-A FAIDs and symmetric plane partitions.

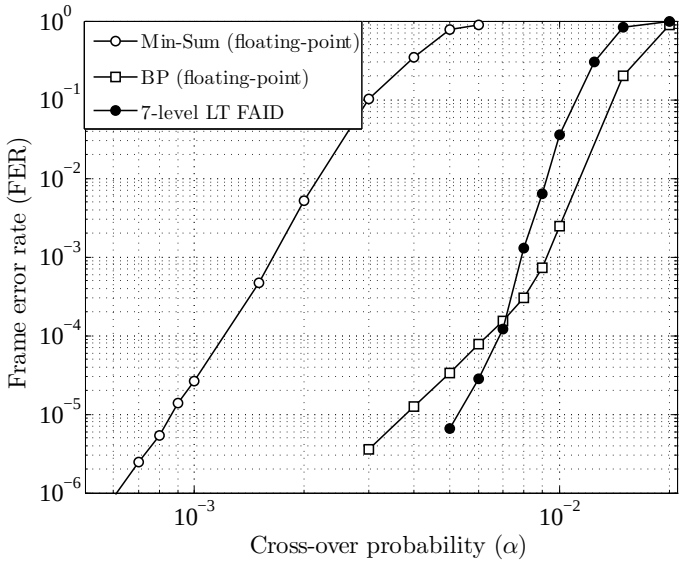


Figure 2.6: Performance comparisons between the floating-point decoders: BP and min-sum , and the 3-bit precision 7-level LT FAID on a $R = 0.82$ $(4095, 3358)$ MacKay code.

2.2.3 Symmetric plane partition representation of Φ_v

A symmetric plane partition π is an array of nonnegative integers $(\pi_{i,j})_{i\geq 1, j\geq 1}$ such that $\pi_{i,j} \geq \pi_{i+1,j}$, $\pi_{i,j} \geq \pi_{i,j+1}$ $\forall i, j \geq 1$, and $\pi_{i,j} = \pi_{j,i}$. If $\pi_{i,j} = 0 \; \forall \; i > r$ and $\forall \; j > s$, and if $\pi_{i,j} \leq t \; \forall i, j$, then the plane partition is said to be *contained* in a box with side lengths (r, s, t). The value $\pi_{i,j}$ is represented as a box of height $\pi_{i,j}$ positioned at (i, j) coordinate on a horizontal plane.

Due to the imposition of the lexicographic ordering and symmetry of Φ_v, there exists a bijection between the array $[l_{i,j}]_{1\leq i\leq N_s, 1\leq j\leq N_s}$ and a symmetric plane partition contained in a $(N_s \times N_s \times N_s - 1)$ box, where each $\pi_{i,j}$ is determined based on $l_{i,j}$. Fig. 2.7 shows the visualization of a plane partition corresponding to Φ_v of the 7-level FAID defined in Table 2.2.

Kuperberg in [91] gave an elegant formula for the enumeration of symmetric plane partitions contained in a box of dimensions (r, r, t). We can directly utilize this formula in order to enumerate the number of class-A N_s-level FAIDs.

Theorem 2.1 (Number of Class-A N_s-level FAID)**.** *The total number $K_A(N_s)$ of symmetric lexicographically ordered N_s-level FAIDs is given by*

$$K_A(N_s) = \frac{H_2(3N_s)H_1(N_s)H_2(N_s-1)}{H_2(2N_s+1)H_1(2N_s-1)}$$

where $H_k(n) = (n-k)!\,(n-2k)!\,(n-3k)!\ldots$ is the staggered hyperfactorial function.

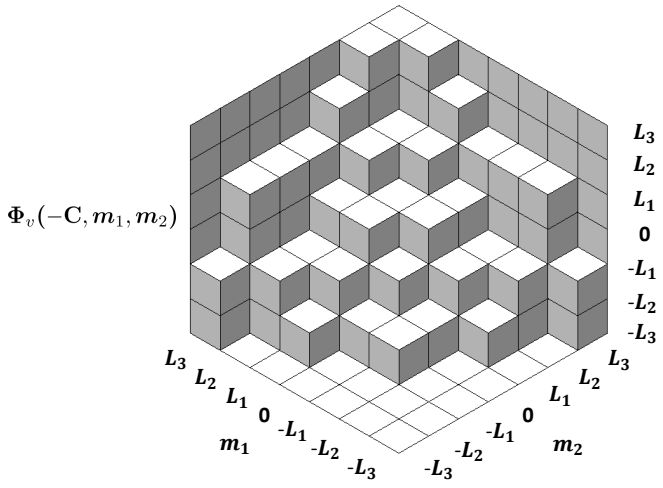


Figure 2.7: A visualization of the plane partition as stacked boxes for the 7-level FAID whose Φ_v is described in Table 2.2.

Proof. The proof of the theorem follows from the bijection between the map Φ_v of a class-A FAID and a symmetric boxed plane partition.

The total number of class-A FAIDs for $N_s = 5$ and $N_s = 7$ levels are 28,314 and 530,803,988 respectively. Henceforth, we shall only consider class-A FAIDs (and simply refer to them as FAIDs).

2.2.4 Isolation assumption and critical number

Earlier in Section 2.1, we illustrated through the example as to how the knowledge of a potentially harmful subgraph could be used to design variable node update functions. However, note that the neighborhood of the subgraph was ignored when we analyzed the message passing. In other words, we carried out the message passing on the subgraph in an isolated manner. We refer to this assumption on the subgraph that was inherently used in the example as the *isolation assumption*.

The isolation assumption is a pivotal notion that enables us to analyze the message passing on potentially harmful subgraphs (or trapping sets) for different FAIDs. It is a specific condition on the neighborhood of the subgraph such that the messages flowing into the subgraph from its neighborhood are not in any way influenced by the messages flowing out of the subgraph for certain number of iterations. It enables us to to extend the important concept of critical number, which was a measure of harmfulness originally introduced for Gallager B (refer to Section 1.3.2 in Chapter 1, to message-passing decoders other than Gallager B (such as the FAIDs).

Before we formally define the concept of isolation assumption, we first require the notion of computation tree [42] and introduce some notations. Note that during the analysis

of any decoders, the all-zero codeword assumption is used.

Definition 2.4. *A computation tree corresponding to a message-passing decoder of the Tanner graph G is a tree that is constructed by choosing an arbitrary variable node in G as its root and then recursively adding edges and leaf nodes to the tree that correspond to the messages passed in the decoder up to a certain number of iterations. For each vertex that is added to the tree, the corresponding node update function in G is also copied.*

Let G be the graph of a code $\mathcal{C}$ with variable node set V and check node set C. Let H be the induced subgraph of a trapping set (a, b) contained in G with variable node set $P \subseteq V$ and check node set $W \subseteq C$. Let $\mathcal{N}(u)$ denote the set of neighbors of a node u. Let $\mathscr{T}_i^k(G)$ be the computation tree of graph G corresponding to a decoder $\mathscr{F}$ enumerated for k iterations with variable node $v_i \in V$ as its root. Let $W' \subseteq W$ denote the set of degree-one check nodes in the subgraph H. Let $P' \subseteq P$ denote the set of variable nodes in H where each variable node has at least one neighbor in W'. During decoding on G, for a node $v_i \in P'$, let μ_l denote the message that v_i receives from its neighboring degree-one check node in H in the l^{th} iteration.

Definition 2.5. *A vertex $w \in \mathscr{T}_i^k(G)$ is said to be a descendant of a vertex $u \in \mathscr{T}_i^k(G)$ if there exists a path starting from vertex w to the root v_i that traverses through vertex u. The set of all descendants of the vertex u in $\mathscr{T}_i^k(G)$ is denoted as $\mathcal{D}(u)$. For a given vertex set U, $\mathcal{D}(U)$ (with some abuse of notation) denotes the set of descendants of all $u \in U$.*

Definition 2.6. *$\mathscr{T}_i^k(H)$ is called the computation tree of the subgraph H enumerated for k iterations for the decoder $\mathscr{F}$, if $\forall\ c_j \in W'$, μ_l is given for all $l \leq k$, and if the root node $v_i \in P$ requires only the messages computed by the nodes in H and μ_l to compute its binary hard-decision value.*

We now provide the definition for isolation assumption as follows.

Definition 2.7 (Isolation assumption)**.** *Let H be a subgraph of G induced by $P \subseteq V$ with check node set $W \subseteq C$. The computation tree $\mathscr{T}_i^k(G)$ with the root $v_i \in P$ is said to be isolated if and only if for any node $u \notin P \cup W$ in $\mathscr{T}_i^k(G)$, u does not have any descendant belonging to $P \cup W$. If $\mathscr{T}_i^k(G)$ is isolated $\forall v_i \in P$, then the subgraph H is said to satisfy the isolation assumption in G for k iterations.*

Essentially the isolation assumption of a subgraph validates the number of iterations for which message passing can be carried out on a subgraph in an isolated manner ignoring its neighborhood, which is done by treating it as if it were an arbitrary Tanner graph (of some code), provided that the messages passed from the degree-one check nodes of H in each iteration are known or determined a priori.

Note that the isolation assumption is a purely topological condition on the neighborhood of H, as it is dependent only on the particular graph G containing H and not on the decoder being used. Also note that the isolation assumption can still be satisfied even

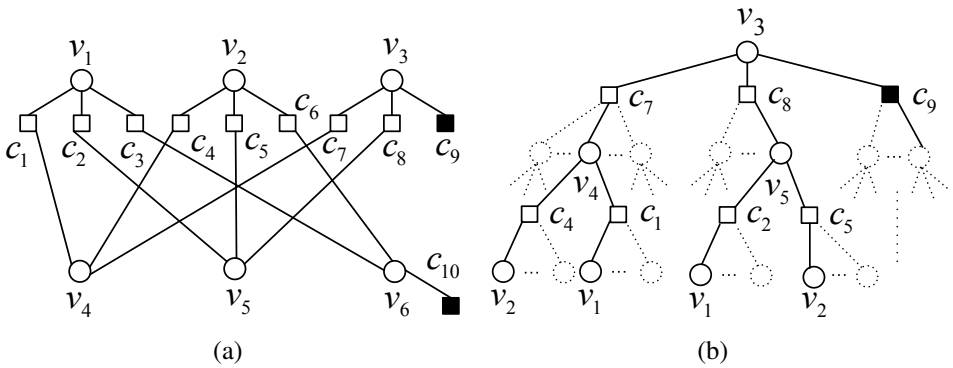


Figure 2.8: Subgraph H corresponding to a $\mathcal{T}(6,2)$ trapping set contained in G: (a) Tanner graph of H; (b) computational tree $\mathcal{T}_3^2(G)$

when there are nodes in H that appear multiple times in $\mathscr{T}_i^k(G)$. Whereas Gallager's independence assumption [1] will be violated if any node in H is repeated in $\mathscr{T}_i^k(G)$. Hence, isolation assumption is a weaker condition than independence. For clarity, we illustrate with an example shown in Fig. 2.8.

Example 2.3. *Let us assume that the graph G of code $\mathcal{C}$ contains a subgraph H corresponding to a $\mathcal{T}(6,2)$ trapping set. Fig. 2.8 shows the subgraph H, and the computation tree $\mathcal{T}_3^2(G)$ of graph G with v_3 as its root enumerated for two iterations. The ■ denotes a odd-degree check node present in H. The solid lines represent connections within subgraph H and the dotted lines represent connections from the rest of the graph G outside the subgraph H. The isolation assumption is satisfied for two iterations if none of the descendants of a node u which is outside H and represented by a dotted circle in $\mathcal{T}_3^2(G)$ are nodes in H. But the independence assumption does not hold for two iterations.*

In order to enable the message passing of a particular decoder on a subgraph H under isolation assumption, we need to know the messages passed from the degree-one check nodes of H in each iteration. Assuming that the decoder $\mathscr{F}$ is a FAID, this can be computed without knowledge of its neighborhood through the following theorem (under all-zero codeword assumption).

Theorem 2.2 (Isolation theorem)**.** *Let H be a subgraph of a 3-left-regular graph G induced by $P \subseteq V$ with check node set $W \subseteq C$. Let $W' \subseteq W$ denote the set of degree-one check nodes in H and let $P' \subseteq P$ denote the set of variable nodes in H for which each has at least one neighbor in W'. Consider decoding on G with FAID $\mathscr{F}$. If $\mathbf{r}$ is received from the BSC such that* $\mathrm{supp}(\mathbf{r}) \subseteq P$*, and if H satisfies the isolation assumption in G for k iterations, then for each $c_j \in W'$, the message from c_j to its neighbor in H in the l^{th} iteration denoted by μ_l, is the output of $\Phi_v(\mathrm{C}, \mu_{l-1}, \mu_{l-1})\ \forall l \leq k$, with $\mu_0 = 0$.*

Proof: Firstly, since $\text{supp}(\mathbf{r}) \subseteq P$ and all messages are initialized to zero, all nodes $v_i \in V \backslash P$ are initially correct, and every variable node initially sends $\Phi_v(y_i, 0, 0)$.

Now consider a computation tree $\mathscr{T}_s^l(G)$ where $l \leq k$ with $v_s \in P'$ as its root. Due to the isolation assumption of H in G satisfied for k iterations, for any $c_j \in \mathcal{N}(v_s) \cap W'$, $\mathcal{D}(c_j) \cap (P \cup W) = \emptyset$. Therefore all nodes $v_i \in \mathcal{D}(c_j)$ are initially correct. Let q denote the level of depth involving variable nodes in $\mathscr{T}_s^l(G)$ such that level $q = 0$ denotes the base and level $q = l$ denotes the root of the tree. Let $\mathcal{D}^{(q)}(c_j)$ denote the variable node descendants of $c_j \in \mathcal{N}(v_s) \cap W'$ at level q of the tree $\mathscr{T}_s^l(G)$. At $q = 1$, due to the nature of Φ_c for FAID $\mathscr{F}$ along with the fact that every correct node initially sends $\Phi_v(\mathrm{C}, 0, 0)$, any $v_i \in \mathcal{D}^{(1)}(c_j)$ receives the message $\mu_1 = \Phi_v(\mathrm{C}, 0, 0)$, from its leaf check nodes. Again, since $\mathcal{D}(c_j) \cap (P \cup W) = \emptyset$, at $q = 2$, any $v_i \in \mathcal{D}^{(2)}(c_j)$ receives $\mu_2 = \Phi_v(\mathrm{C}, \mu_1, \mu_1)$. By induction, this can be generalized to any level q, where any node $v_i \in \mathcal{D}^{(q)}(c_j)$, receives $\mu_q = \Phi_v(\mathrm{C}, \mu_{q-1}, \mu_{q-1})$ from its leaf check nodes. Therefore the root v_s receives $\mu_l = \Phi_v(\mathrm{C}, \mu_{l-1}, \mu_{l-1})$. ■

Although the theorem is stated specifically for column-weight-three codes, it can be generalized to left-regular codes of higher column-weight as well. Also the isolation theorem can be restated for the min-sum decoder through the following corollary.

Corollary 2.1. *Consider the min-sum decoder for column-weight-three codes with $\mathcal{Y} = \{\pm 1\}$. If subgraph H contained in G satisfies the isolation assumption for k iterations, and if all variable nodes outside H are initially correct, then μ_l of the degree-one check node for the min-sum decoder is $2\mu_{l-1} + 1$.*

Corollary 2.2. *If H is a subgraph contained in G such that it satisfies the isolation assumption for k iterations, and if all variable nodes outside H are initially correct, then the computation tree $\mathscr{T}_i^k(G)$ with $v_i \in P$ is equivalent to $\mathscr{T}_i^k(H)$, provided μ_l for each degree-one check node in H is computed using the isolation theorem.*

The above corollary implies that we can analyze error patterns on isolated potentially harmful subgraphs and this can be useful for deriving good FAIDs. With the help of isolation assumption and theoem, we can now extend the concept to critical number to FAIDs through the definition below.

Definition 2.8. *The critical number of a FAID denoted by $\mathscr{F}$ on a subgraph H is the smallest number of errors introduced in H for which $\mathscr{F}$ fails on H under the isolation assumption.*

Given the subgraph of a potential trapping set, the critical number can be determined for a given FAID by introducing errors into nodes of the subgraph, and decoding under isolation assumption. The critical number of $\mathscr{F}$ on H is said to be ∞ if $\mathscr{F}$ corrects all possible error patterns on H. A FAID could then be chosen based on maximizing the critical number on the trapping set. Another important parameter to be considered is the number of iterations required for the decoder to successfully converge. Since the isolation assumption will not hold for more than few iterations in a actual graph G containing the

subgraph, it is important that the FAID is not only able to have the highest critical number but also achieves this with the fewest iterations.

For example, consider the subgraph corresponding to a $\mathcal{T}(9,5)$ TS which contains a $\mathcal{T}(6,2)$ TS shown in Fig. 2.9. This subgraph is present in the graph of the $R = 0.75$ $(768, 576)$ quasi-cyclic code. The subgraph contains three degree-one check nodes and two degree-3 check nodes for a total of five odd-degree check nodes. The three nodes initially in error are v_1, v_2, and v_3.

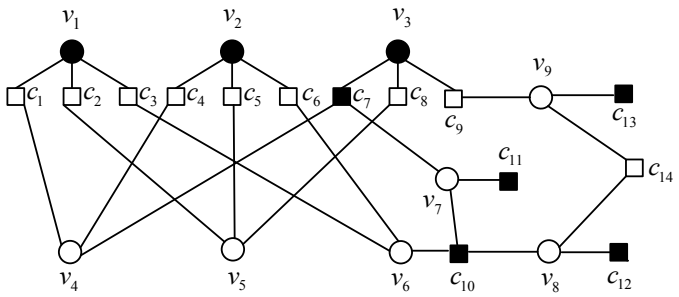


Figure 2.9: Subgraph corresponding to a $\mathcal{T}(9,5)$

Under isolation assumption, the 7-level LT FAID (defined in Ex. 2.2) takes three iterations to successfully converge on the subgraph whereas the floating-point min-sum decoder takes four iterations to converge. Now when decoding was carried out with the same error pattern on the graph G instead of on the subgraph, the 7-level LT FAID still successfully corrected the 3-error pattern in three iterations, but the min-sum decoder failed on the 3-error pattern. This is because the isolation assumption of the subgraph was violated by the fourth iteration in G, and the influence of the neighborhood began to affect the message passing causing the min-sum decoder to fail. Therefore, number of iterations required for convergence is also important to consider.

In principle, one could consider a database of subgraphs which are potential trapping sets that are generated either through analytical or empirical evaluations of traditional decoders such as BP and min-sum on several different codes, and then select a FAID based on its critical numbers and number of iterations required for convergence on all these trapping sets. For instance, in [92], we utilized the subgraphs corresponding to the minimum-weight (weight-20) codewords of the $(155, 64)$ Tanner code as the trapping sets, and then selected the best FAIDs based on their respective critical numbers on these trapping sets.

A major drawback with designing the FAIDs solely based on critical number is that if trapping sets of smaller size are considered, the critical number may not reflect the true error-correction capability of the FAID on a code containing the trapping set as the isolation assumption typically does not hold in an actual code for more than few iterations. Therefore, unless a very large database of trapping sets is considered or unless trapping sets with large sizes are considered such that isolation assumption holds for many more

iterations (as done in [92] which considered the weight-20 codewords), the strategy will remain ineffective. In the next section which addresses the methodology for selection, we will introduce a more refined notion of the critical number of the trapping set that to an extent takes into account the influence of its neighborhood.

2.3 Selection of Finite Alphabet Iterative Decoders

In the previous section, we enumerated the number of all possible class-A N_s-level FAIDs using symmetric plane partitions, and it is evident from the large numbers that identifying particularly good FAIDs is highly non-trivial. In this section, we shall describe a general approach that can be used to identify a subset of candidate N_s-level FAIDs, one or several of which are potentially good for any given column-weight-three code. Our main aim behind this approach is to restrict the choice of FAIDs to a possibly small subset containing good candidates. Given a particular code, it would then be feasible to identify the best performing FAID from this small subset by using brute-force simulation or emulation or some other technique on the given code. Moreover, since the actual performance of a FAID on a given code depends on its structure, the goal of identifying several candidate FAIDs is more realistic than identifying a single particularly good FAID and allows for devising a selection method that is not code-specific. Another important objective of our approach is to ensure that any candidate FAID belonging to this subset is capable of surpassing BP in the error floor not just on a single code but on several codes.

The approach we use relies on the knowledge of potentially harmful subgraphs that could be trapping sets for traditional iterative decoders when present in a given code. The candidate FAIDs are chosen by analyzing their behavior on each of these subgraphs with errors introduced in them. We will first introduce some important notions that form the basis of our approach, and then subsequently present a methodology based on these notions to identify particularly good FAIDs for column-weight-three codes. Note that for the methodology, we only consider (a, b) elementary trapping sets of relatively small size for which $b > 0$.

2.3.1 Noisy trapping sets and noisy critical numbers

Let us consider a subgraph with topology $\mathcal{T}$ that has been identified as a harmful subgraph that could be a potential (a, b) trapping set on a given code. We introduce the notion of *initialization vector* which will allow us to partially capture the influence of its arbitrary (unknown) neighborhood during the analysis of a FAID on the $\mathcal{T}$.

Definition 2.9. *An initialization vector on a trapping set $\mathcal{T}(a, b)$ is defined as a vector $\mathbf{\Theta} = (\theta_1, ..., \theta_b)$ where $\theta_i \in \mathcal{M}$, such that during the message passing of a FAID on $\mathcal{T}$, the message passed by the i^{th} degree-one check node in any iteration is θ_i. The TS $\mathcal{T}(a, b)$ is said to be initialized by such a vector and is referred to as a noisy trapping set.*

In other words, a FAID is still analyzed under isolation assumption of the TS $\mathcal{T}(a, b)$ but with subtle difference that the initialization vector specifies the messages passed by the degree-one check nodes instead of the isolation theorem. A FAID can now be analyzed by introducing errors into the variable nodes of the TS $\mathcal{T}(a, b)$ and passing messages iteratively on the edges of $\mathcal{T}$ under a given initialization vector. Note that we consider only static initialization vectors, i.e., the initialization vector $\mathbf{\Theta}$ is not iteration-dependent.

As an example, Fig. 2.10 depicts how a FAID is analyzed for a three-error pattern on a $\mathcal{T}(6, 2)$ initialized by a vector $\mathbf{\Theta} = (\theta_1, \theta_2)$. A ● denotes a variable node initially wrong (v_1, v_2, and v_4) and a ○ denotes a node initially correct (v_3, v_5, and v_6). A □ denotes a degree-two check node and a ■ denotes a degree-one check node. Initially all the messages passed by all nodes except the degree-one check nodes are set to zero. Then the messages are iteratively updated using the maps Φ_v and Φ_c in the usual manner by treating the topology $\mathcal{T}$ as if it were an arbitrary graph (under isolation assumption) but with the exception that a degree-one check node sends θ_1 (or θ_2) to its neighbors in all iterations of the message passing. The message update on a single edge from a variable node is shown in the figure for each of the nodes v_1, v_2, v_3 , and v_5 (v_4 and v_6 are similar to v_2 and v_3 respectively). Note that the messages $m_1, m_2, \ldots, m_6$ denote the extrinsic incoming messages to these nodes.

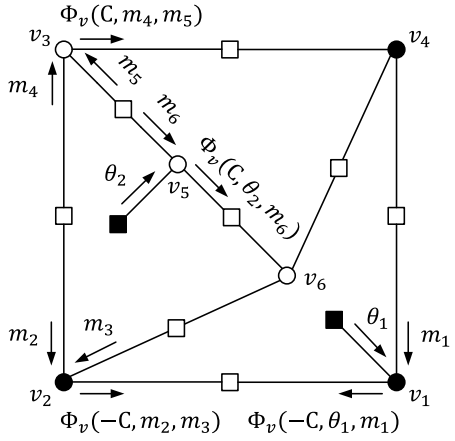


Figure 2.10: An example of a noisy $\mathcal{T}(6, 2)$ initialized by a vector $\mathbf{\Theta}$.

Let N_I denote the maximum number of iterations allowed for message passing on TS $\mathcal{T}(a, b)$ during the analysis of a FAID. We examine whether an error pattern is corrected by the FAID within N_I iterations under a given initialization vector on the TS $\mathcal{T}(a, b)$.

Our main intuition for defining such a notion is as follows. Let us consider a code whose graph G contains a subgraph H that is isomorphic to the topology $\mathcal{T}(a, b)$. Assume that a particular FAID is being used for decoding an error pattern where some (or all) of the variable nodes in H are initially in error and the nodes outside H are initially correct. During each iteration of decoding, different possible messages belonging to $\mathcal{M}$ will be

passed into the nodes of H from outside of H depending on its neighborhood.

The initialization vector can be considered as a possible snapshot of the messages entering H through its check nodes in some arbitrary iteration, and different initializations represent the different possible influences that the neighborhood of H can have. Therefore, analyzing the FAID under different initializations on a given $\mathcal{T}$ can provide a reasonably good indication of its error correction capability on a code whose graph G contains H.

Although the initialization vector should ideally be iteration-dependent and include all messages passed to all check nodes of $\mathcal{T}(a,b)$ from outside of $\mathcal{T}(a,b)$, this would make analyzing a FAID on $\mathcal{T}(a,b)$ computationally intractable. Therefore we only include constant values that are passed to degree-one check nodes into the initialization vector.

We now define the notion of *noisy critical number* which is an extension of the notion of critical number defined in the previous section, and which will be used as the main parameter for the selection of FAIDs.

Definition 2.10. *The noisy critical number of a FAID under a given initialization vector* $\mathbf{\Theta}$ *on a trapping set* $\mathcal{T}(a,b)$ *is the smallest number of errors introduced in* $\mathcal{T}(a,b)$ *for which the FAID fails on* $\mathcal{T}(a,b)$.

By determining the noisy critical number under every possible initialization vector $\mathbf{\Theta} \in \mathcal{M}^b$ on the TS $\mathcal{T}(a,b)$, a vector of noisy critical numbers, which we refer to as *noisy critical number vector* (NCNV), can be obtained for a particular FAID. Let N_Θ denote the number of all possible initialization vectors, i.e., $N_\Theta = |\mathcal{M}^b|$. The noisy critical number vector of a FAID denoted by $\mathscr{F}$ on a given TS $\mathcal{T}(a,b)$ is given by

$$\mathscr{N}_{\mathscr{F}}(\mathcal{T}(a,b), N_I) = (\zeta_1, \zeta_2, \ldots, \zeta_{N_\Theta})$$

where ζ_i is the noisy critical number determined under a initialization vector $\mathbf{\Theta_i} \in \mathcal{M}^b$ on TS $\mathcal{T}(a,b)$ with N_I being the maximum number of allowed decoding iterations. The NCNV can now be used to identify which candidate FAIDs are potentially good.

2.3.2 Choice of trapping sets for decoder selection

Since our approach for identifying good FAIDs relies on determining the NCNVs of FAIDs on different trapping sets, the first step in the decoder selection is to carefully select the harmful topologies that should be considered for the analysis. The selected trapping sets should be topologies that are known to exist in practical high-rate codes with dense graphs and are regarded as relatively harmful for existing iterative decoders. Also the trapping sets used in the analysis should have notable differences in their topological structures, so that the candidate FAIDs identified from the analysis are more likely to be good on several codes rather than just on a single code.

We use the TSO [62] to determine which harmful topologies to consider. The TSO was introduced earlier in Chapter 1 as a hierarchy of trapping sets based on their topological relations in the form of a parent-child relation. For the decoder selection, the trapping sets are chosen such that they do not have many common parents, and that most of the parents (graphs of smaller size) in the TSO are considered. For simplicity, we ensure that all the trapping sets selected have the same value of b, so that the NCNVs determined from different trapping sets all have the same dimension.

2.3.3 Decoder domination

Having selected the harmful topologies, the next step in the decoder selection is to determine and be able to compare the NCNVs of different FAIDs on all the selected trapping sets. We introduce the notion of *decoder domination* in order to make the comparison between different FAIDs based on their NCNVs.

Let the set of chosen trapping sets for the analysis of FAIDs be denoted by $\Lambda = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{N_\Lambda}\}$ with cardinality N_Λ. Let $\mathcal{F} = \{\mathrm{D}_1, \ldots, \mathrm{D}_{N_\mathcal{F}}\}$ denote the set of class-A N_s-level FAIDs considered for possible decoder selection with cardinality $N_\mathcal{F}$. Let $\mathscr{N}_{\mathrm{D}_k}(\mathcal{T}_j, N_I)$ denote the NCNV of a FAID $\mathrm{D}_k \in \mathcal{F}$ determined on a TS $\mathcal{T}_j \in \Lambda$, and let $\mathscr{N}^{(i)}_{\mathrm{D}_k}(\mathcal{T}_j, N_I)$ denote the i^{th} component of the NCNV, i.e., $\mathscr{N}^{(i)}_{\mathrm{D}_k}(\mathcal{T}_j, N_I) = \zeta_i$, where ζ_i is the noisy critical number of FAID D_k under the initialization vector $\mathbf{\Theta_i}$.

A FAID D_k is said to dominate a FAID D_l for a given initialization vector $\mathbf{\Theta_i}$, if

$$\mathscr{N}^{(i)}_{\mathrm{D}_k}(\mathcal{T}_j, N_I) \geq \mathscr{N}^{(i)}_{\mathrm{D}_l}(\mathcal{T}_j, N_I) \;\; \forall j \in \{1, 2, \ldots, N_\Lambda\}$$

In other words, D_k dominates D_l under a given initialization vector $\mathbf{\Theta_i}$ if the noisy critical numbers of D_k are greater than or equal to the noisy critical number of D_l on all the trapping sets belonging to Λ.

The number of initialization vectors under which D_k dominates D_l is denoted by $\tilde{n}(\mathrm{D}_k, \mathrm{D}_l)$ and is given by

$$\tilde{n}(\mathrm{D}_k, \mathrm{D}_l) = \sum_{i=1}^{N_\Theta} \prod_{j=1}^{N_\Lambda} \mathbb{1}\left(\mathscr{N}^{(i)}_{\mathrm{D}_k}(\mathcal{T}_j, N_I) \geq \mathscr{N}^{(i)}_{\mathrm{D}_l}(\mathcal{T}_j, N_I)\right)$$

where $\mathbb{1}$ denotes the indicator function that outputs a one when the condition in its argument is true and zero otherwise.

If $\tilde{n}(\mathrm{D}_k, \mathrm{D}_l) \geq \tilde{n}(\mathrm{D}_l, \mathrm{D}_k)$, then D_k is said to dominate D_l with *domination strength* $\tilde{n}(\mathrm{D}_k, \mathrm{D}_l) - \tilde{n}(\mathrm{D}_l, \mathrm{D}_k)$. For simplicity we shall use the symbol $\rhd$ to denote domination, i.e., $(\mathrm{D}_k \rhd \mathrm{D}_i) = 1$ implies that D_k dominates D_i.

2.3.4 Methodology for selection: a general approach

For a given value of N_s, a methodology for identifying good N_s-level FAIDs can now be devised based on the notions of decoder domination and the NCNVs. We remind the reader that the main goal of our approach is to be able to identify a small subset of candidate N_s-level FAIDs, where each candidate FAID is potentially good on several codes. Let this small subset of selected candidate FAIDs be denoted by $\mathcal{F}_c$. Ideally, if a candidate FAID could be selected solely based on how it dominates all the other FAIDs in $\mathcal{F}$, then one could possibly obtain an ordering of the FAIDs in $\mathcal{F}$ in terms of their dominance and conclude as to which ones are more likely to be good on a particular code containing one or more of the trapping sets in Λ. Unfortunately, we have found that such an ordering does not exist since there can be many FAIDs that dominate a particularly good FAID (known a priori to be good) and yet perform poorly on certain codes.

Therefore, without going into the details, we shall describe a general approach for selection that utilizes pre-determined small sets of good FAIDs and bad FAIDs denoted by $\mathcal{F}_g$ and $\mathcal{F}_b$ respectively. The set $\mathcal{F}_g$ consists of N_s-level FAIDs that are known a priori to have good error floor performance on several codes of different rates and possibly containing different trapping sets. The set $\mathcal{F}_b$ consists of N_s-level FAIDs that were found to perform well on one particular code but perform poorly on other codes. We regard FAIDs in $\mathcal{F}_b$ to be bad since our goal is to identify FAIDs that are capable of surpassing BP on several codes.

We then evaluate whether a particular FAID $\mathrm{D}_k \in \mathcal{F}$ dominates or is dominated by the FAIDs in the sets $\mathcal{F}_g$ and $\mathcal{F}_b$. By using the sets $\mathcal{F}_g$ and $\mathcal{F}_b$ to compare with, we are inherently trying to select FAIDs whose NCNVs have characteristics similar to the NCNVs of FAIDs in $\mathcal{F}_g$ but dissimilar to the NCNVs of the FAIDs in $\mathcal{F}_b$. Therefore, we define a cost function denoted by $\mathscr{C}_{\tilde{n}}$ that is based on domination strengths and whose value determines whether the FAID D_k should be accepted for inclusion into $\mathcal{F}_c$. We have observed that it is crucial for a candidate FAID to dominate most (or all) FAIDs in $\mathcal{F}_g$ and also not be dominated by most (or all) FAIDs in $\mathcal{F}_b$, in order for it to be considered potentially good. This aspect is reflected in the cost function $\mathscr{C}_{\tilde{n}}$ which is defined below.

$$\begin{aligned}\mathscr{C}_{\tilde{n}}(\mathrm{D}_k) = &\sum_{i:\ \mathrm{D}_i\in\mathcal{F}_g,(\mathrm{D}_k\rhd\mathrm{D}_i)=1} \Big(\tilde{n}(\mathrm{D}_k,\mathrm{D}_i)-\tilde{n}(\mathrm{D}_i,\mathrm{D}_k)\Big)\\ &+\sum_{j:\ \mathrm{D}_j\in\mathcal{F}_b,(\mathrm{D}_k\rhd\mathrm{D}_j)=1} \Big(\tilde{n}(\mathrm{D}_k,\mathrm{D}_j)-\tilde{n}(\mathrm{D}_j,\mathrm{D}_k)\Big)\\ &-\sum_{i:\ \mathrm{D}_i\in\mathcal{F}_g,(\mathrm{D}_i\rhd\mathrm{D}_k)=1} \Big(\tilde{n}(\mathrm{D}_i,\mathrm{D}_k)-\tilde{n}(\mathrm{D}_k,\mathrm{D}_i)\Big)\\ &-\sum_{j:\ \mathrm{D}_j\in\mathcal{F}_b,(\mathrm{D}_j\rhd\mathrm{D}_k)=1} \Big(\tilde{n}(\mathrm{D}_j,\mathrm{D}_k)-\tilde{n}(\mathrm{D}_k,\mathrm{D}_j)\Big)\end{aligned} \tag{2.9}$$

The value of the cost function $\mathscr{C}_{\tilde{n}}$ is compared to a threshold τ. If $\mathscr{C}_{\tilde{n}}(\mathrm{D}_k) \geq \tau$,

then the FAID D_k is selected as a candidate to be included in $\mathcal{F}_c$, else it is rejected. The cardinality of $\mathcal{F}_c$ depends on τ since a smaller τ accepts more FAIDs and a larger τ accepts less FAIDs. The choice of N_I also plays a role and should generally be chosen to be small (5 to 10 iterations).

Note that the approach we have just presented is slightly different from the one proposed in [94]. In [94], the selection algorithm assumes it has no a priori knowledge on the sets $\mathcal{F}_g$ and $\mathcal{F}_b$, and then tries to progressively build the sets before using them to identify good candidate FAIDs. Instead, by utilizing pre-determined sets of $\mathcal{F}_g$ and $\mathcal{F}_b$ in our approach, we have found that the selection procedure is greatly improved and we were able to obtain much better sets of candidate FAIDs $\mathcal{F}_c$ (in terms of their error floor performance). Note however that the approach of [94] is still applicable to the selection method presented in this Section as it could still be used as an initial step for determining the sets $\mathcal{F}_g$ and $\mathcal{F}_b$.

Using our methodology, we were able to derive a set of good candidate 7-level FAIDs (which are 3-bit precision decoders) for column-weight-three codes. On a variety of codes of different rates and lengths, particularly good 7-level FAIDs chosen from $\mathcal{F}_c$ all outperformed the BP (floating-point) in the error floor. Moreover, the loss in the waterfall compared to BP was found to be very reasonable. The numerical results to support this statement are provided in the next section. Another interesting remark related to our selection procedure that we have found is that, although the density evolution (DE) was not used as part of the selection method to ensure good threshold values, the candidate FAIDs that we obtained in set $\mathcal{F}_c$ were all found to have fairly good DE thresholds.

2.4 Numerical results

Earlier in Section 2.2, we demonstrated the capability of 3-bit precision FAIDs to outperform BP in the error floor on the well-known $(155, 64)$ Tanner code, a high-rate girth-8 quasi-cyclic code, and a girth-6 random MacKay code. In this section, we provide additional numerical results on girth-8 column-weight-three codes of higher practical interest in order to validate our approach for decoder selection and illustrate the efficacy of FAIDs derived from the selection. The three codes used for the simulations were chosen to cover a broad variety of LDPC codes in terms of rate, length, and structure. They are:

1. an $R = 0.751$ $(2388, 1793)$ structured code based on latin squares,
2. an $R = 0.5$ $(504, 252)$ code,
3. an $R = 0.833$ $(5184, 4322)$ quasi-cyclic code.

The $(2388, 1793)$ structured code was constructed using the method of Nguyen *et al.* [72] which is based on Latin squares and avoids certain harmful trapping sets in the Tanner graph of the code. The $R = 0.5$ $(504, 252)$ code with girth-8 was designed using

the progressive edge-growth (PEG) method of [93] but with the additional constraint that it contains no $(5, 3)$ TS (see Fig. 1.6 for the topology). The $(5184, 4322)$ quasi-cyclic code is an example of a high-rate girth-8 quasi-cyclic code that was designed for having a minimum distance of 12.

Figures 2.11, 2.12, and 2.13, show the frame error-rate (FER) performance comparisons versus the cross-over probability α of the BSC between the particularly good 7-level (3-bit precision) FAIDs we identified and the BP (floating-point). The 7-level FAID used on the $(2388, 1793)$ code and the $(5184, 4322)$ quasi-cyclic code is defined by Table 2.2 which was already previously established as an NLT FAID. The 7-level FAID that was used on the $(504, 252)$ PEG-based code is defined by Table 2.3, which is also an NLT FAID due to the diagonal entries in Table 2.3. Both 7-level NLT FAIDs were candidate FAIDs identified using the approach we described in the previous section. All decoders were run for a maximum of 100 iterations.

Table 2.3: LUT for Φ_v of a 7-level FAID with $y_i = -\mathrm{C}$

$\boldsymbol{m_1/m_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\boldsymbol{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	0
$\boldsymbol{-L_2}$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	$-L_1$	L_1
$\boldsymbol{-L_1}$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	$-L_1$	0	L_2
0	$-L_3$	$-L_2$	$-L_2$	$-L_1$	0	L_1	L_2
$\boldsymbol{L_1}$	$-L_3$	$-L_2$	$-L_1$	0	0	L_1	L_2
$\boldsymbol{L_2}$	$-L_3$	$-L_1$	0	L_1	L_1	L_2	L_3
$\boldsymbol{L_3}$	0	L_1	L_2	L_2	L_2	L_3	L_3

In all three codes, the 3-bit precision 7-level FAIDs begin to surpass the BP in the error floor at an FER$\simeq 10^{-5}$. Specifically, for the $(2388, 1793)$ structured code, it is worth mentioning that this code was optimized for the best known BP performance in terms of error floor for the given code parameters by ensuring that the graph of the code is devoid of certain harmful trapping sets during its construction. We see that even for such a code designed for better error floor performance, the 7-level NLT FAID is able to surpass BP. This validates our argument we made in Chapter 1 that efforts to address the error floor from a code construction viewpoint has its limitations especially when involving higher code rates. Furthermore, the result also shows that FAIDs are capable of surpassing BP not just on codes exhibiting high error floors due to poor constructions but even on codes optimized for improved error floor performance.

Also notice the difference in the better slopes of the error floor for the 7-level FAIDs which can be attributed to their enhanced guaranteed error correction capability. For instance, both the 5-level and 7-level FAIDs used on the $(155, 64)$ Tanner code in Fig. 2.4 guarantee a correction of 5 errors, whereas BP fails to correct several 5-error patterns.

Also included in Fig. 2.11 are results provided to illustrate the ineffectiveness of designing FAIDs solely based on their asymptotic thresholds evaluated from density evo-

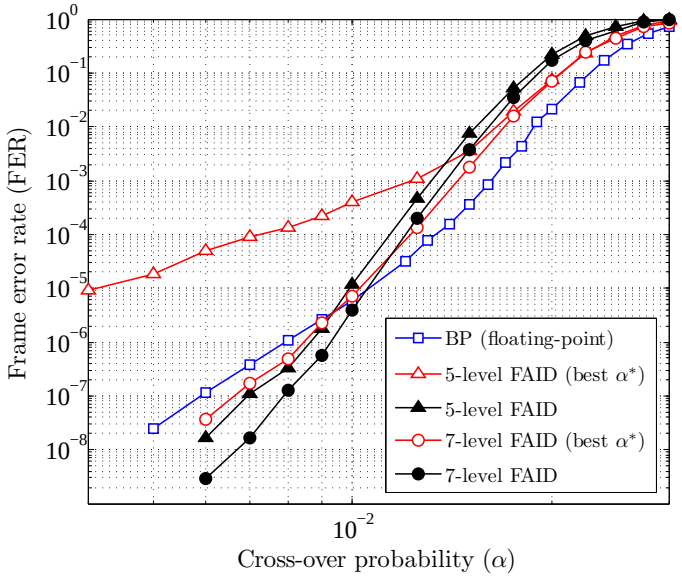


Figure 2.11: Performance comparisons between the BP (floating-point), the 5-level, and the 7-level FAIDs on the $(2388, 1793)$ structured code. The best 7-level NLT FAID used is defined by Table 2.2.

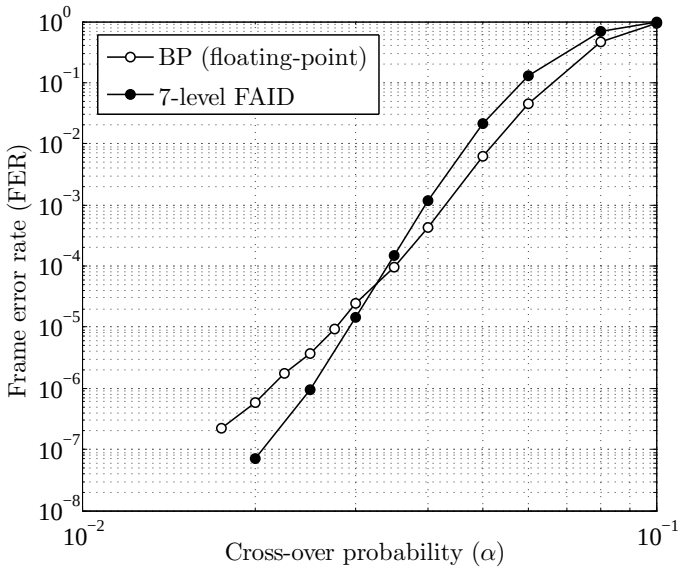


Figure 2.12: Performance comparisons between the BP (floating-point) and the 7-level NLT FAID defined by Table 2.3 on the $(502, 252)$ PEG-based code.

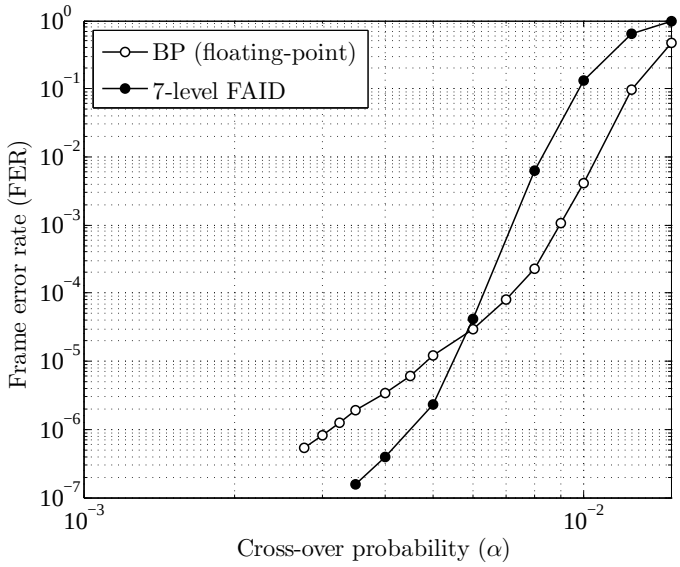


Figure 2.13: Performance comparisons between the BP (floating-point) and the 7-level NLT FAID defined by Table 2.2 on the $(5184, 4322)$ quasi-cyclic code.

lution [19]. For the $R = 0.75$ $(2388, 1793)$ structured code, we identified the 5-level and 7-level FAIDs with the best thresholds α^* for the degree profile of the code. The 5-level FAID has a threshold of $\alpha^* = 0.025134$, and the 7-level FAID has a threshold of $\alpha^* = 0.025244$. Additionally, the result of another 5-level FAID that surpasses BP is also included in Fig. 2.11, which is defined by the following constraints on the values of $\mathcal{M}$, $\mathscr{T}$, and the function Ω: $L_2 = L_1 + \mathrm{C}$, $\mathrm{C} = 1.5L_1$, $T_1 = L_1$, $T_2 = L_2$, and Ω given by

$$\Omega(m_1, m_2) = 1 - \Big(\mathrm{sign}(m_1) \oplus \mathrm{sign}(m_2)\Big) \cdot \delta(|m_1| + |m_2| - 2L_2).$$

It is evident from the results that FAIDs with best thresholds are not necessarily the best performing FAIDs, and in fact they can even have high error floors as exhibited by the 5-level FAID. This verifies the inapplicability of using the decoding threshold as a possible design parameter for identifying FAIDs with good error floor performance. More evidence for this argument is provided in [94].

We finally remark that although numeral results in this section were provided for only three codes, the good 7-level FAIDs identified using our approach outperformed BP on several other tested codes as well. Therefore, the methodology is applicable to any column-weight-three code and provides to an extent "universally" good FAIDs, as they are all capable of surpassing BP on not just few but several column-weight-three codes

2.5 Conclusions

In this Chapter 2, we introduced a new paradigm for finite precision iterative decoding of LDPC codes on the BSC. Referred to as FAIDs, the newly proposed decoders use node update maps that are much simpler than BP yet capable of surpassing the floating-point BP with only three bits of precision. We described the general framework of FAIDs with focus on column-weight-three codes and provided examples of particularly good 3-bit precision FAIDs that can have a higher guaranteed error correction capability than the floating-point BP and min-sum. We also provided a general methodology to identify a set of “universally" good FAIDs, one or several of which are potentially good for any given column-weight-three code. Our methodology is thus not code-specific but rather utilizes the knowledge of harmful topologies that could be potentially trapping sets for a given code under iterative decoding. Using our methodology, we were able to identify FAIDs that outperform the floating-point BP in the error floor on several column-weight-three codes. The supporting numerical results show that it is possible to achieve a much superior error-rate performance in the error floor at a much lower complexity and lower memory requirements than BP by using FAIDs.

Chapter 3

Decimation-Enhanced Decoding: Analysis and Improved Error Correction

In the previous chapter, we proposed a novel approach to finite precision iterative decoding and illustrated how FAIDs can be designed to have a much superior error-rate performance than BP in the error floor without compromise in decoding latency. However, despite their superior performance and improved guaranteed error correction compared to BP, the proposed FAIDs are still far from achieving the guaranteed error correction ability of ML decoding. Moreover, the ability to analyze FAIDs in order to derive provable statements in terms of guaranteed error correction still remains a challenge. The main focus of this chapter is to address the above two issues. We approach these problems by introducing the technique of *decimation*, which involves fixing certain bits of the codeword to a particular value, and incorporating this into the variable node update function of FAIDs in different novel ways, thus constituting two new classes of decoders, namely, *decimation-enhanced FAIDs* and *adaptive decimation-enhanced FAIDs* for LDPC codes on the BSC.

We first propose decimation mainly as a tool that enables the analysis of FAIDs. We provide a simple decimation scheme for a particularly good 3-bit precision FAID (surpassing BP) that reduces the number of iterations required to correct a fixed number of errors while maintaining the good performance of the original FAID. With the help of this scheme, we provide insights into how the decoders can be analyzed to derive provable results related to their guaranteed error correction capability. We then propose a more sophisticated scheme that involves using decimation in an adaptive manner to further enhance the guaranteed error correction of a particularly good FAID in an effort to reduce the gap between FAIDs and ML decoding. Specifically focusing on 3-bit precision FAIDs for column-weight-three codes, we will show that on certain high-rate codes, the adaptive decimation scheme proposed enables the decoders to achieve a guaranteed error correction ability that is close to ML decoding with only a marginal increase in complexity. We

will also provide some analysis on these decoders which suggests that their failures are linked to stopping sets.

3.1 Background

Finite-length analysis of message-passing (MP) algorithms for LDPC codes has always been an important problem that received significant attention from the research community. One of the early pioneering works on analyzing MP algorithms for a fixed number of iterations was done by Wiberg [17] who introduced the notion of computation trees (which was used in the previous chapter). For finite-length analysis of codes on the BEC, significant work was done by Di *et al.* [44] who introduced the notion of stopping sets. Subsequently, Vontobel and Koetter [51] introduced the concept of graph cover decoding for finite-length analysis of MP decoders. Kelly and Sridhara [52] proposed a pseudocodeword analysis for LDPC codes and derived bounds on the minimum pseudocodeword-weight of a given code. Burshtein and Miller used expander arguments which allows for infinite iterations in the analysis of MP algorithms [25]. More recently, Chilappagari *et al.* [65] provided bounds on the guaranteed error correction capability of Gallager A/B decoding in relation to the girth on column-weight-three codes.

In spite of the aforementioned techniques proposed for finite-length analysis, the problem of analyzing soft MP algorithms such as FAIDs for a fixed number of iterations and deriving bounds on their error correction capability still proves to be a difficult problem. This is because the dynamics of MP becomes too complex beyond a certain number of iterations as there is exponential growth in the number of nodes with the number of iterations in the computation trees of the codes. In the context of FAIDs, even though they are designed to surpass BP in the error floor, the convergence of a FAID for an error pattern in a trapping set is heavily influenced by the neighborhood of the trapping set in a non-trivial manner which complicates the analysis.

Furthermore, on the BSC, the importance of guaranteed error correction capability was already established in Chapter 1 as it governs the slope of the error floor in the performance of the decoder [64]. Therefore, it would be desirable to have an MP decoder that is able to correct a fixed number of errors within the fewest possible iterations, and for which we will be able to provide performance guarantees in terms of guaranteed correction capability. Even from a practical standpoint, this would be an attractive feature with many present-day applications requiring much higher decoding speeds as well as a high guaranteed error correction capability. In order to achieve this objective with FAIDs and make them more amenable to analysis, we propose decimation-enhanced FAIDs for LDPC codes on the BSC.

Decimation is a method based in statistical physics that was developed for solving constraint satisfaction problems. It essentially involves guessing the values of certain variables and fixing them to these values while continuing to estimate the remaining vari-

ables. One of the notable works related to decimation is the work of Montanari *et al.* [95] who proposed and analyzed a BP-guided randomized decimation procedure that estimates the variables in the *k*-SAT problem. In the context of decoding LDPC codes, a similar notion was proposed for LP decoding by Dimakis and Wainwright [96] in the form of facet guessing, wherein the algorithm first runs LP decoding, and if a fractional pseudocodeword is the decoded output, the algorithm then guesses facets of the polytope in either a randomized or exhaustive manner to eliminate the fractional pseudocodeword and reruns LP on those facets. Chertkov proposed a bit-guessing algorithm for LP decoding that is a simplified version of the facet-guessing algorithm in order to reduce error floors of LDPC codes [97].

In contrast to the aforementioned works, we utilize decimation in a novel way by incorporating it into the decoding of LDPC codes so that certain variable nodes are decimated after a few iterations of message passing, and a variable node is decimated based on its incoming messages at the end of some iteration. In this manner, decimation primarily serves as a guide to help the decoder to converge faster on low-weight error patterns. Our main insight in the role of decimation is not necessarily in correcting more errors, but in ensuring that more variable nodes in the graph that are initially correct are shielded from the erroneous messages emanating from the nodes initially in error by decimating those correct variable nodes.

We now discuss decimation with goal of correcting a fixed number of errors in fewest iterations, and illustrating how it can help with the analysis of decoders. Later we will show how decimation can also be used to further improve the guaranteed error correction achievable by FAIDs.

3.2 Decimation As a Tool For Analysis

3.2.1 A motivating example

We illustrate the benefit of decimation through the following example. Note once again that we shall use the all-zero codeword assumption for the analysis of decoders.

Let $\mathcal{N}(u)$ denote the set of neighbors of a node u in the graph G and let $\mathcal{N}(U)$ denote the set of neighbors of all $u \in U$. Let $m^{(k)}(v_i, c_j)$ denote the message being passed from a variable node v_i to the check node c_j, in the k^{th} iteration, and let $m^{(k)}(c_j, v_i)$ be defined similarly. Let $m^{(k)}(v_i, \mathcal{N}(v_i))$ denote the set of outgoing messages from v_i to all its neighbors in the k^{th} iteration, and let $m^{(k)}(c_j, \mathcal{N}(c_j))$ be defined similarly. Recall that $\hat{x}_i^{(k)}$ denotes the bit value of a variable node $v_i \in V$ decided by the decoder at the end of the k^{th} iteration.

Consider a particular 4-error pattern on a Tanner graph G, such that the induced subgraph H of the four nodes in error is an 8-cycle as shown in Fig. 3.1. In the figure, ● represents a variable node initially in error, and ○ represents an initially correct node that

is in the neighborhood outside H. The ■ and □ denote the degree one and degree two check nodes in the induced subgraph respectively. Let V'=$\{v_1, v_2, v_3, v_4\}$ denote the set of nodes initially in error in H. Let C^1=$\{c_1, c_3, c_5, c_7\}$ denote the set of degree one checks and C^2=$\{c_2, c_4, c_6, c_8\}$ denote the set of degree two checks in H.

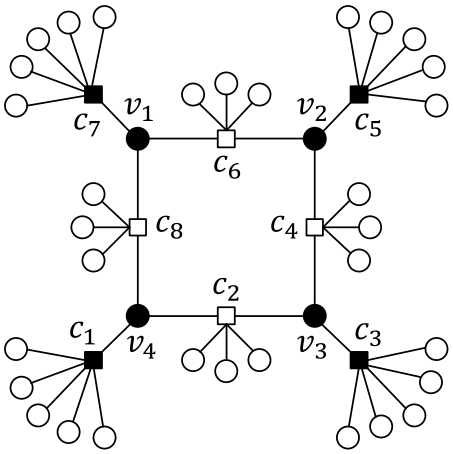


Figure 3.1: Subgraph induced by the 4-error pattern which forms an 8-cycle

We shall now examine the behavior of MP on this particular error configuration from the context of N_s-level FAIDs without any assumptions on its neighborhood. Messages with a positive sign will be referred to as *good* messages, and messages with a negative sign will be referred to as *bad* messages. Also a message is referred to as *strongly* good (bad) or *weakly* good (bad) based on the magnitude of the message. For instance, a weakly good or bad message refers to $\pm L_1$, and a strongly good or bad message refers to $\pm L_i$ where $L_i > L_1$.

Assuming that $\Phi_v(\mathrm{C}, 0, 0) = L_1$, in the first iteration, for all $v_i \in V'$, $m^{(1)}(v_i, \mathcal{N}(v_i))$ will consist of weakly bad messages, and for all $v_j \in \mathcal{N}(C^1 \cup C^2) \backslash V'$, $m^{(1)}(v_j, \mathcal{N}(v_j) \cap (C^1 \cup C^2))$ entering into the subgraph will consist of weakly good messages. In the second iteration, for all $v_i \in V'$, $m^{(2)}(v_i, \mathcal{N}(v_i) \cap C^2)$ will consist of either weakly good or weakly bad messages depending on the Φ_v, but $m^{(2)}(v_i, \mathcal{N}(v_i) \cap C^1)$ which consists of messages sent to check nodes in C^1, will be strongly bad assuming $\Phi_v(-\mathrm{C}, -L_1, -L_1) = -L_2$. As a result, variable nodes $v_i \in \mathcal{N}(C^1) \backslash V'$ will start receiving a strongly bad message on at least one of its edges. Now if the decoder does not converge within the next few iterations, then these bad messages become more strongly bad with circulation of bad messages within H, and this can subsequently spread further to other nodes in the graph outside H depending how dense the neighborhood is. Eventually too many nodes get corrupted by the bad messages being propagated in the graph causing the decoder to fail.

There are some important observations to note from the above discussion. Firstly, at the k^{th} iteration, there may have been many variable nodes $v_i \notin \mathcal{N}(C^1 \cup C^2)$ whose

decided bit value $\hat{x}_i^{(k')}$ converged to the right value in some $k' < k$ iteration, but eventually these nodes became corrupted in the k^{th} iteration due to the bad messages flowing out of the subgraph. Secondly, if certain variable nodes initially correct in the neighborhood of H, are isolated from the bad messages possibly through decimation, then the decoder is more likely to converge. This is the case especially when there are relatively fewer errors introduced by the channel that are localized to a particular part of the graph (such as the 4-cycle), and this typically occurs in the high SNR (error floor) region. This is precisely where decimation becomes beneficial and important.

From the example, we can see that the role of decimation is inherently linked to the concept of isolation assumption which was defined in the previous chapter. Recall that the isolation assumption is a condition on the neighborhood of a subgraph contained in the graph such that the messages passed within the subgraph as well as messages entering into it from outside are not influenced by the messages being passed in its neighborhood and vice versa for certain number of iterations. In this context, the role of decimation can be understood as trying to isolate the subgraph induced by the nodes in error from the rest of the graph, so that the rest of the graph converges quickly and in turn helps to correct the nodes initially in error.

It is important to note once again that the emphasis with regards to using decimation is on low-weight error patterns typically associated with harmful trapping sets and being able to correct these patterns in the fewest possible iterations. An error pattern is considered to be low-weight if its weight is less than or equal to $\lfloor (d_{min} - 1)/2 \rfloor$.

3.2.2 Decimation-enhanced FAIDs

We now formally define the concept of decimation, introduce the required notations, and describe the novel way in which it is incorporated into the message passing of FAIDs. Note that the definitions are general enough to be applicable to any column-weight-d_v code, but we will eventually restrict our discussion to only column-weight-three codes.

Definition 3.1. *A variable node v_i is said to be decimated at the end of l^{th} iteration if $\hat{x}_i^{(k)}$ is set to $\hat{x}_i^*$ $\forall k > l$. Then $m^{(k)}(v_i, \mathcal{N}(v_i)) = \{(-1)^{\hat{x}_i^*} L_s\}$, $\forall k \geq l$ irrespective of its incoming messages $m^{(k)}(\mathcal{N}(v_i), v_i)$, i.e., v_i will always send the strongest possible messages.*

Note that if a node v_i is decimated at the end of l^{th} iteration, then this is equivalent to effectively deleting all its descendants $\mathcal{D}(v_i)$ in the computation tree $\mathscr{T}_i^k(G)$ $\forall k > l$ since the node always sends $(-1)^{b_i^*} L_s$ to its parent.

A *decimation rule* $\beta : \mathcal{Y} \times \mathcal{M}^{d_v} \rightarrow \{-1, 0, 1\}$ is a function used at the end of some l^{th} iteration by the decoder to decide whether a variable node should be decimated and what value it should be decimated to based on the incoming messages and the channel value in the l^{th} iteration. Let γ_i denote the output of a decimation rule applied to a node v_i. If

$\gamma_i = 0$, then the node is not decimated. If $\gamma_i = 1$, then the node is decimated with $\hat{x}_i^* = 0$, and if $\gamma_i = -1$, then the node is decimated with $\hat{x}_i^* = 1$.

There are two key aspects to note regarding the application of a decimation rule that add to the novelty of how we incorporate decimation into FAIDs.

1) The decimation rule is applied after messages are passed iteratively for some l iterations.

2) After each instance of applying the decimation rule, all messages are cleared to zero (which is practically restarting the decoder except that the decimated nodes remain decimated).

The first aspect implies that the decoder itself decides which nodes to decimate after passing messages for some number of iterations, which is in contrast to existing approaches that use decimation. The second aspect aids in preventing the growth of bad messages in the graph as well as in simplifying the analysis. More on the rationale behind this will be discussed later. We shall refer to each instance of applying a decimation rule on all variable nodes as a *decimation round*.

For now we only consider the use of a single decimation rule β which may be used in different iterations. We will later generalize to using muliple decimation rules when we discuss adaptive decimation. We now formally introduce the class of decimation-enhanced N_s-level FAIDs for the BSC.

A decimation-enhanced N_s-level FAID (DFAID) denoted by $\mathscr{F}^D$ is defined as a 4-tuple given by $\mathscr{F}^D = (\mathcal{M}, \mathcal{Y}, \Phi_v^D, \Phi_c)$, where $\mathcal{M} = \{-L_s, \ldots, -L_1, 0, L_1, \ldots, L_s\}$, $\mathcal{Y} = \{\pm \mathrm{C}\}$, and Φ_c are the same as defined for a N_s-level FAID. The map $\Phi_v^D : \mathcal{Y} \times \mathcal{M}^{d_v-1} \times \{-1, 0, 1\} \to \mathcal{M}$ is similar to Φ_v of the FAID $\mathscr{F}$ with the minor difference that it uses the output of β in some l^{th} iteration as an additional argument in the function. Again let $m_1, \cdots, m_{d_v-1}$ denote the extrinsic incoming messages to a node $v_i \in V$ with degree d_v. Then Φ_v^D is defined as

$$\Phi_v^D(y_i, m_1, m_2, \ldots, m_{d_v-1}, \gamma_i) = \begin{cases} \Phi_v(y_i, m_1, m_2, \ldots, m_{d_v-1}), & \gamma_i = 0 \\ \gamma_i L_s, & \gamma_i = \pm 1 \end{cases}$$

In our proposed framework of DFAIDs, a decimation rule β must satisfy certain important properties. For ease of exposition, the properties are specified below for a degree-3 variable node.

1. $\beta(\mathrm{C}, m_1, m_2, m_3) = -\beta(-\mathrm{C}, -m_1, -m_2, -m_3)$
   $\forall m_1, m_2, m_3 \in \mathcal{M}$

2. $\beta(\mathrm{C}, m_1, m_2, m_3) \neq -1$ and $\beta(-\mathrm{C}, m_1, m_2, m_3) \neq 1$ $\forall m_1, m_2, m_3 \in \mathcal{M}$

3. Given $m_1, m_2, m_3 \in \mathcal{M}$, if $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$, then $\beta(\mathrm{C}, m_1', m_2', m_3') = 1$ $\forall m_1', m_2', m_3' \in \mathcal{M}$ such that $m_1' \geq m_1$, $m_2' \geq m_2$, and $m_3' \geq m_3$.

Property 1 just enforces symmetry on the function β.

Property 2 implies that a node v_i can be decimated to zero only if $y_i = \mathrm{C}$ and to one only if $y_i = -\mathrm{C}$, meaning a node can be decimated only to its received value r_i from BSC. An important consequence of this is that, a node initially correct will never be decimated to a wrong value and a node initially wrong will never be decimated to the correct value. This means that, by Property 2, decimation will not by itself correct nodes initially in error, but rather acts as a reinforcer to the nodes initially correct by preventing them from being corrupted when they are decimated. The rationale behind this is that since we are focusing on low-weight error pattens, the number of nodes initially in error are significantly small compared nodes initially correct, so we rely on the nodes initially correct to drive the decoder to converge. Moreover, this simplifies the analysis of the decoders. It is evident then that a necessary condition for successful decoding is that no node initially in error is decimated.

Property 3 ensures that there is monotonicity in the inputs with relation to the output of the decimation rule, and this is required due to the nature of decimation. For instance, if a degree-3 variable node initially correct is decimated when its incoming messages are L_1, L_1, and L_2, then it must be decimated when it receives L_2, L_2, and L_2.

For convenience, we shall refer to variable nodes initially in error in G as *error nodes* and variable nodes initially correct in G as *correct nodes* throughout this chapter. We now propose a simple decimation scheme on column-weight-three codes with the underlying FAID being a 7-level FAID.

3.2.3 Proposed scheme for 7-level FAID on column-weight-three codes

Let N_d denote the number of decimation rounds carried out by the decoder with a given decimation rule β. The decimation scheme essentially specifies exactly which iteration the decimation rule is applied, and the number of such decimation rounds. For the scheme, we only consider 7-level FAIDs whose variable node update maps Φ_v satisfy $\Phi_v(\mathrm{C}, 0, 0) = L_1$, $\Phi_v(\mathrm{C}, L_1, L_1) = L_2$, and $\Phi_v(\mathrm{C}, L_2, L_2) = L_3$. This is because, all the particularly good 7-level FAIDs were found to have this property.

The scheme we propose involves performing the first round of decimation at the end of the third iteration, then restarting the decoder and allowing one iteration of message passing after which the next decimation round is performed, and this is repeated until a total of N_d decimation rounds have been completed. Note that once a node is decimated in a particular decimation round, it remains decimated for the remainder of the entire decoding, and the decimation rule is applied only on the non-decimated nodes. The skeleton of the proposed scheme for a 7-level FAID on a column-weight-three code is given in Algorithm 3.2.3.

There are two main reasons for performing the first decimation round at the end of the third iteration (which are specific to the 7-level FAID). Firstly, since the bit value of a node remains fixed once the node is decimated, we want to ensure that nodes being decimated

Algorithm 1 Decimation-enhanced FAID algorithm

1) Initialize $\gamma_i = 0 \; \forall v_i \in V$.

2) Run the decoder for three iterations using update maps Φ_v and Φ_c defined for the 7-level FAID.

3) Perform decimation using the rule β for every $v_i \in V$, which constitutes the first decimation round.

4) Restart the decoder by resetting all the messages to zero and pass messages for one iteration. This implies that a decimated node v_i will send $\gamma_i L_3$ and a non-decimated nodes v_j will send $\Phi_v(y_j, 0, 0)$.

5) Repeat step 3) only for nodes $v_i \in V$ whose $\gamma_i = 0$, followed by 4) until N_d decimation rounds have been completed.

6) Run the decoder for the remainder of iterations using maps Φ_v^D and Φ_c.

have reliably converged, which is partially determined by the strength of messages entering the node. Therefore, message passing must be done for enough iterations to allow the possibility of a message with the highest possible strength being passed in the graph. For the 7-level FAIDs considered, since $\Phi_v(\mathrm{C}, 0, 0) = L_1$, and $\Phi_v(\mathrm{C}, L_1, L_1) = L_2$, at least three iterations are required for an L_3 or $-L_3$ to be passed by a variable node. Therefore we need to allow at least three iterations of message passing before performing the decimation.

Secondly, as illustrated in the example discussed previously, it would be desirable to decimate the correct nodes as much as possible before they get influenced by the bad messages emanating from the error nodes. Therefore, the decoder must not carry out too many iterations of message passing before performing the first decimation. Based on the two reasons, the choice of allowing three iterations for message passing seems to be an appropriate choice. Moreover, decimating nodes after only three iterations makes the algorithm much more amenable to analysis. For instance, it becomes possible now to analyze under what conditions of the graph will an error node get decimated. We shall in fact derive such conditions for the previous example of the 4-cycle.

With regards to restarting the decoder after each decimation round, the rationale behind this is to allow the decimated correct nodes to drive the decoder convergence in the right direction by preventing the growth of bad messages in the graph (as all messages are reset to zero), assuming that no error node is decimated in the first decimation round. Furthermore, by performing subsequent decimation rounds at the end of just one iteration after restarting the decoder, it leads to the following important lemma.

Lemma 3.1. *Consider decoding on G with DFAID $\mathscr{F}^D$ where the first decimation round is performed at the end of third iteration, and subsequent decimation rounds are performed after resetting all messages to zero and passing messages for one iteration. Let β be such that $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$. Given an error pattern on G, if no error node gets decimated in the first decimation round, then no error node will get decimated in the subsequent decimation rounds.*

Proof: Since no error node is decimated in the first decimation round, an error node will send $\Phi_v(-\mathrm{C}, 0, 0) = -L_1$ in the first iteration after resetting all the messages to zero. In the worst-case scenario for the error node, each of its neighboring checks can be connected to an additional error node, in which case it receives a $-L_1$ from all three of its neighbors at the end of one iteration. But since $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$ implying that $\beta(-\mathrm{C}, -L_1, -L_1, -L_1) = 0$ by Property 1, an error node will not get decimated. This holds for any subsequent decimation round. ■

As a result of Lemma 3.1, if β is defined so that $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$, then we need to only ensure that an error node is not decimated at the end of third iteration, which simplifies analysis. Moreover, any number of decimation rounds can be used without being concerned about an error node being decimated. We now provide some insights into the design of the decimation rule β.

3.2.4 Design of decimation rule β

It is clear from the previous discussion that the design of β is critical for ensuring that none of the error nodes are decimated on low-weight error patterns while more correct nodes are decimated. Also due to Lemma 3.1, we only consider decimation rules that satisfy $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$.

In order to define the decimation rule β, we define a set Ξ that consists of all unordered triples $(m_1, m_2, m_3) \in \mathcal{M}^3$ such that $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$. Note that for any unordered triple $(m_1, m_2, m_3) \in \Xi$, $\beta(-\mathrm{C}, -m_1, -m_2, -m_3) = -1$ by property 1, so Ξ is sufficient to completely specify β.

The design of the rule β can now be considered as a procedure of selecting the unordered triples to be included in the set Ξ. This depends not only on the particular graph G of a code but also on the particular underlying FAID being used. Given an underlying 7-level FAID, we would like to do the selection with particular focus on correcting small number of errors typically associated with trapping sets in the error floor region. Referring back to the example of the 8-cycle shown in Fig. 3.1, a good decimation rule would be one where γ_j for most or all nodes $v_j \in \mathcal{N}(C^1 \cup C^2) \backslash V'$ is 1 and γ_i for nodes $v_i \in V'$ is 0 at the end of all decimation rounds. Let us assume we are designing a decimation rule β for a particularly good 7-level FAID identified in the previous chapter, whose Φ_v is defined by Table 3.1. Although we do not describe a rigorous method to design the rule β, we highlight three main points that are important to consider for its design.

Table 3.1: Φ_v of a 7-level FAID for a node v_i with $d_v = 3$ and $y_i = +\mathrm{C}$

$\boldsymbol{m_1/m_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\mathbf{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	$-L_1$	$-L_1$	L_1
$\boldsymbol{-L_2}$	$-L_3$	$-L_1$	$-L_1$	0	L_1	L_1	L_3
$\boldsymbol{-L_1}$	$-L_2$	$-L_1$	0	0	L_1	L_2	L_3
$\mathbf{0}$	$-L_1$	0	0	L_1	L_2	L_3	L_3
$\boldsymbol{L_1}$	$-L_1$	L_1	L_1	L_2	L_2	L_3	L_3
$\boldsymbol{L_2}$	$-L_1$	L_1	L_2	L_3	L_3	L_3	L_3
$\boldsymbol{L_3}$	L_1	L_3	L_3	L_3	L_3	L_3	L_3

Table 3.2: Set Ξ consisting of all message triples such that $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$

m_1	m_2	m_3
L_3	L_3	L_3
L_3	L_3	L_2
L_3	L_3	L_1
L_3	L_3	0
L_3	L_3	$-L_1$

m_1	m_2	m_3
L_3	L_2	L_2
L_3	L_2	L_1
L_3	L_2	0
L_3	L_2	$-L_1$
L_3	L_1	L_1

m_1	m_2	m_3
L_3	L_1	0
L_3	L_1	$-L_1$
L_3	0	0
L_2	L_2	L_2
L_2	L_2	L_1

Firstly, when considering whether a particular unordered triple (m_1, m_2, m_3) should be included in Ξ or not, the number of positive messages and negative messages in the triple as well as their magnitudes play a role in the selection. For instance, (L_3, L_1, L_1) may be a more plausible candidate than $(L_3, L_2, -L_2)$, because the former has three positive messages, and also the latter has a high negative message in $-L_2$.

Secondly, the inherent nature of the particular Φ_v used in the FAID must be taken into consideration during selection. For this, we need to look at what outgoing messages a variable node would send when a particular triple is the set of incoming messages, and then decide if this is good for selection. For instance, given the 7-level FAID defined by Table 3.1, the triple (L_2, L_2, L_1) might be a more plausible candidate than $(L_3, L_2, -L_2)$ since the outgoing messages for the former would be all L_3, but the outgoing messages for the latter would be L_3, L_3, and L_1.

Thirdly, if the harmful subgraphs present in graph G of a given code are known, the isolation assumption can be a useful tool to identify which triples should be avoided for inclusion in Ξ. If our goal is to correct certain number of errors , say t, the MP with the 7-level FAID would be analyzed for a given t-error pattern on the subgraph under isolation assumption. If a particular triple matches with the messages received by an error node in the subgraph at the end of third iteration, this triple must be not be included in Ξ.

Using the approach outlined in the above three points, we provide a decimation rule designed, with the underlying 7-level FAID defined by Table 3.1, for the well-known $(155, 64)$ Tanner code. The rule was designed with the goal of reducing the number of

iterations required to guarantee a correction of 5 errors on the code. Table 3.2 shows all unordered triples included in the set Ξ.

Using the 7-level DFAID comprising of the decimation rule β defined by Ξ in Table 3.2 and the 7-level FAID defined by Table 3.1, we will now illustrate how decimation aids in analysis.

3.2.5 Analysis

Given a particular error pattern in a graph G, we can analyze the conditions on the graph under which an error node is decimated at the end of third iteration. We can then place conditions on the graph such that the node in error is never decimated. To illustrate this, we revert back to the example of the 4-error pattern shown in Fig. 3.1 whose induced subgraph forms an 8-cycle, and provide such conditions in the following theorem. Note that the proof involves using Tables 3.1 and 3.2 and also the same notations previously defined in the example.

Theorem 3.1. *Consider the 4-error pattern contained in a graph G for which the subgraph induced by the nodes initially in error forms an 8-cycle. Also consider the 7-level DFAID for decoding where Φ_v and the β are defined in Tables 3.1 and 3.2 respectively. If the graph G has girth-8, and no three check nodes of the 8-cycle share a common variable node, then the nodes initially in error will not get decimated in any decimation round.*

Proof: Firstly note that by virtue of Φ_v of the 7-level FAID (Table 3.1), the highest magnitude of a message that any node $v_i \in V$ can send is L_1 in the first iteration and L_2 in the second iteration. Since a node $v_j \in \mathcal{N}(C^1 \cup C^2) \backslash V'$ can be connected to atmost two checks in subgraph, the node v_j in the worst case recieves two $-L_1$ messages from checks in $C^1 \cup C^2$ and L_1 from outside at the end of first iteration. Node $v_i \in V'$ will also receive two $-L_1$ messages from check nodes in C^2 and L_1 from $c_k \in C^1 \cap \mathcal{N}(v_i)$. At the end of the second iteration, the node $v_i \in V'$ will once again receive two $-L_1$ from checks in C^2, and L_1 from $c_k \in C^1$. This means that node v_i will receive two $-L_1$ messages once gain from checks in C^2 at the end of third iteration. In order for it to be decimated, from Table 3.2, it must receive $-L_3$ from $c_k \in C^1 \cap \mathcal{N}(v_i)$. This means that the node v_j in the worst case has to receive at least one $-L_3$ at the end of the second iteration, but this is not possible by virtue of Φ_v in the second iteration. Hence, a node initially in error can not get decimated at the end of third iteration and using Lemma 2, will never get decimated. ■

Note that the above condition is easily satisfied in most practical codes. This implies that on most practical codes, 4 errors on an 8-cycle will not be decimated.

In a similar manner, we can derive such conditions on the graph for other types of error configurations. In fact the first step in proving the achievable guaranteed error correction of t errors by a DFAID is to prove that for all possible t-error configurations, no error node gets decimated (under certain conditions of the graph G). The next step would be

to analyze under what conditions a node initially correct is decimated. For example, we may be able to derive conditions on the neighbors of the 4-error configuration such that they get decimated. The final step is then to link the analytical results of decimation with decoder convergence.

We will soon provide numerical results showing that the 7-level DFAID maintains the good performance of the 7-level FAID but can significantly reduce the number of iterations required to guarantee a correction of t errors.

An interesting question that arises when comparing the performance of a 7-level FAID and a 7-level DFAID is that, whether the DFAID is guaranteed to correct a t-error pattern by the 7-level FAID, if none of the error nodes are decimated. In other words, assuming that FAID corrects an error pattern in I iterations, if DFAID decimates only correct nodes for a given error pattern, can it only help the decoder to converge since it uses the same map Φ_v after N_d decimation rounds?

Intuitively, one may expect decimating only correct nodes for a given error pattern to always improve the convergence of the decoder, since these nodes will then continuously send strong correct messages for the entire decoding process. However, this is not the case due to the nature of Φ_c. This can be explained with a simple example of analyzing MP in some k^{th} iteration.

Consider an error pattern that FAID corrects in I iterations. Assume that during a particular iteration, say k^{th} under MP with a FAID, a check node with degree d_c receives as its incoming messages, one weakly good message (say L_1) from a correct node, and strongly bad messages (say -L_3) from its remaining neighbors. The outgoing message passed by the check node is then a weakly bad message by virtue of Φ_c. However, if that correct node is decimated, this could lead to a strongly bad message being passed which could potentially lead to decoder failure.

We remark though that such instances are more likely to occur when the decimation rule is not appropriately designed or when an insufficient number of decimation rounds have been used especially on error patterns that needed large number of iterations by FAID for convergence. Therefore the design of the decimation rule as well as choice of the number of decimation rounds is critical for ensuring that there is no loss in performance compared to FAID.

3.2.6 Numerical results and discussion

Numerical results are presented on the well-known $(155, 64)$ Tanner code in order to evaluate the performance of the DFAID. The frame-error rate curves for BP, 7-level FAID, and 7-level DFAID are shown in Fig. 3.2. For the results, the 7-level DFAID with β and Φ_v defined by Tables 3.1 and 3.2 used $N_d = 4$ decimation rounds. All decoders used a maximum of 100 iterations. Note that the DFAID was designed primarily to correct a fixed number of errors (in this case 5 errors) in fewer iterations compared to 7-level FAID. On the Tanner code, with N_d = 1, the 7-level DFAID corrects all 5 errors within 10

iterations whereas the 7-level FAID requires 15 iterations. At the same time, we see that DFAID performs just as good as the 7-level FAID which surpasses BP in the error floor.

So far our main aim was to provide a simple decimation scheme with FAIDs that allows us to analyze their behaviour while maintaining their good performance. From the preliminary analysis, we see that the role of decimation is important not just in improving the decoder performance or reducing the decoder speed, but more so in terms of increasing the feasibility to obtain provable statements on the performance of MP decoders such as FAID that are known to be empirically good.

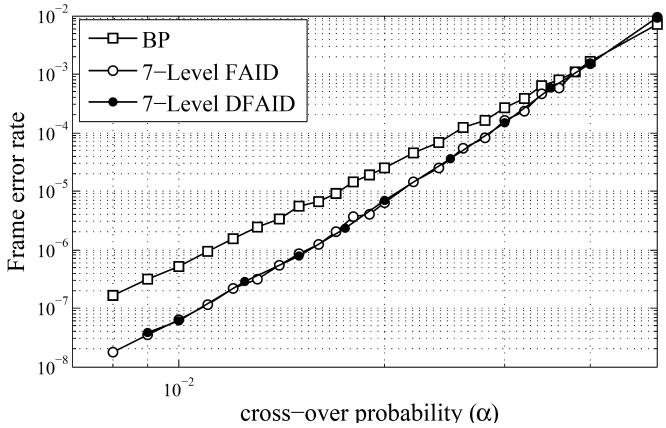


Figure 3.2: Frame error rate performance comparison of Belief Propagation (BP), Finite Alphabet Iterative Decoder (FAID), and Decimation-enhanced FAID (DFAID) on the $(155, 64)$ Tanner code

We now turn our attention to the goal of further improving the guaranteed error correction of FAIDs with an attempt to approach the guaranteed error correction of the optimal ML decoding. In order to achieve this, we still utilize decimation but in a more sophisticated way than what was presented so far.

3.3 Enhancing the Guaranteed Error Correction of FAIDs via Adaptive Decimation

In this section, we show how decimation can be used adaptively to further increase the guaranteed error correction capability of FAIDs. It was previously established that FAIDs while being able to surpass BP, are still far from achieving the guaranteed error correction of ML decoding. For instance, the 7-level FAIDs used in the previous section is able to guarantee a correction of 5 errors on the $(155, 64)$ Tanner code with $d_{min} = 20$, which is still a significant gap compared to the maximum achievable guaranteed error correction

of $\lfloor (d_{min} - 1)/2 \rfloor = 9$. In [**?**], we discussed how one can improve the guaranteed error correction by using several FAIDs sequentially or in parallel. In contrast to that, we propose to use adaptive decimation to improve the guaranteed error correction, which may have lower complexity and lower memory requirements. The adaptive scheme we propose using decimation has only marginally increased complexity compared to FAID , but can significantly improve the error-rate performance compared to the FAIDs. We once again specifically focus on decoders that propagate only 3-bit messages, i.e., 7-level FAIDs, and column-weight three codes since these enable simple implementations and thus have high practical value.

3.3.1 Adaptive decimation-enhanced FAIDs

We shall utilize all the definitions and notations related to decimation introduced earlier in the previous section. We now formally define the class of adaptive decimation-enhanced multilevel FAIDs (ADFAIDs) as follows. A decoder belonging to such a class denoted by $\mathscr{F}^A$ is defined as $\mathscr{F}^A = (\mathcal{M}, \mathcal{Y}, \Phi_v^D, \Phi_v^d, \Phi_v^r, \mathcal{B}, \Phi_c)$, where the sets $\mathcal{M}$ and $\mathcal{Y}$, and the map Φ_c are same as the ones defined for a multilevel FAID. The map $\Phi_v^D : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \times \{-1, 0, 1\} \to \mathcal{M}$ is the update function used at the variable node. It requires the output of a decimation rule β as an one of its arguments and also uses the maps $\Phi_v^d : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \to \mathcal{M}$ and $\Phi_v^r : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \to \mathcal{M}$ to compute its output. Again, we restrict ourselves to the case of column-weight-three codes.

$$\Phi_v^D(y_i, m_1, m_2, \gamma_i) = \begin{cases} \Phi_v^d(y_i, m_1, m_2), & \text{if } \gamma_i = 0, p \leq N_d \\ \Phi_v^r(y_i, m_1, m_2), & \text{if } \gamma_i = 0, p > N_d \\ \gamma_i L_s, & \text{if } \gamma_i = \pm 1 \end{cases}$$

where p denotes the p^{th} decimation round completed by the decoder. The maps Φ_v^d and Φ_v^r are defined as either LT or NLT functions or as LUTs similar to Φ_v of a FAID $\mathscr{F}$.

The new class of decoders proposed in this section use two different maps, Φ_v^d and Φ_v^r, for updating the messages on non-decimated variable nodes, as oppposed to DFAIDs which uses a single map Φ_v). Φ_v^d is the map used to update messages specifically during the decimation procedure, whereas Φ_v^r is the map used to decode the remaining non-decimated nodes after the decimation procedure is completed. Also note that for the case of $|\mathcal{M}| = 7$, we restrict the definition of Φ_v^d to satisfy $\Phi_v^d(\mathrm{C}, 0, 0) = L_1$, $\Phi_v^d(\mathrm{C}, L_1, L_1) = L_2$, and $\Phi_v^d(\mathrm{C}, L_2, L_2) = L_3$, just as we did for DFAIDs in the previous section. Φ_v^r is also defined similarly.

In the definition of ADFAIDs, let N_d denote the number of decimation rounds carried out by the decoder with a given decimation rule β beyond which no more variable nodes are decimated. Note that this is in contrast to DFAIDs, where the value of N_d was required to be chosen. In the case of ADFAIDs, decimation rounds are continued until no more nodes in the graph G can get decimated. The stopping criterion for this is provided in the

following proposition.

Proposition 3.1. *Given a decimation rule β, if the number of decimated nodes after the p^{th} decimation round is the same as the number of decimated nodes after the $(p-1)^{th}$ decimation round, then no additional nodes will get decimated in the subsequent decimation rounds.*

This leads to the notion of *residual graph* defined below.

Definition 3.2. *The residual graph G' is the induced subgraph of the set of variable nodes in G that are not decimated after N_d decimation rounds.*

The set $\mathcal{B}$ is the set of decimation rules used for adaptive decimation, and note that for any $\beta \in \mathcal{B}$, β must still satisfy the three properties provided in the previous section which are related to symmetry, monotonicity in the inputs in relation to the output, and the property that a node can be decimated only to its received value.

Again we use the set Ξ to uniquely define a decimation rule β. Note that a β is considered to be a *conservative* decimation rule if $|\Xi|$ is small and an *aggressive* rule if $|\Xi|$ is large. Also note that the DFAIDs defined in the previous section can be regarded as a special case of the newly proposed ADFAIDs, with $\Phi_v^d = \Phi_v^r = \Phi_v$, and $\mathcal{B} = \{\beta\}$. In other words, while only a single non-adaptive decimation rule and a single map is used for updating messages in the DFAIDs, the ADFAIDs use multiple decimation rules and two distinct maps for updating messages. Once again note that we shall refer to variable nodes that are initially in error in G as *error* nodes and variable nodes that are initially correct as *correct* nodes.

3.3.2 Motivation for adaptive decimation

Given an error pattern of relatively low weight ($\leq \lfloor \frac{d_{min}-1}{2} \rfloor$), the primary role of decimation is to isolate the subgraph associated with the error pattern from the rest of the graph by decimating as many correct nodes outside this subgraph as possible. Now if a given error pattern is such that the error nodes are relatively clustered with many interconnections between them through their neighboring check nodes, then a more conservative β would have to be used by the decoder to ensure that none of the error nodes are decimated. However, if the error pattern is such that the error nodes are more spread out, then it may be desirable to use a more aggressive β as there will be many correct nodes in the neighborhood of the error nodes that can be decimated without decimating the error nodes, and, in turn, possibly help the decoder to converge. This is our main motivation for the use of adaptive decimation in the newly proposed decoders, and we will eventually show that adaptive decimation can help achieve an increase in the guaranteed error correction capability of the code.

3.3.3 Proposed scheme

We now describe a particular adaptive decimation scheme used by the decoder $\mathscr{F}^A$ in order to enhance the guaranteed error correction capability. In the proposed scheme, the set $\mathcal{B}$ consists of two decimation rules, namely $\mathcal{B} = \{\beta^{(1)}, \beta^{(2)}\}$, where $\Xi^{(1)}$ and $\Xi^{(2)}$ are the sets of unordered triples that completely specify the rules $\beta^{(1)}$ and $\beta^{(2)}$ respectively. The rule $\beta^{(1)}$ is used only once at the end of the third iteration, and then from that point, $\beta^{(2)}$ is used after every two iterations ($l = 2$). The use of adaptive decimation is carried out only through $\beta^{(2)}$ as follows.

We define a sequence of decimation rules $\{\beta^{(2)[j]}\}_j$ from $\beta^{(2)}$ by considering ordered subsets of $\Xi^{(2)}$ with increasing size. Let N_β be the number of rules in the sequence $\{\beta^{(2)[j]}\}_j$ and let $\Xi^{(2)[j]}$ denote the set that specifies the rule $\beta^{(2)[j]}$. Then $\Xi^{(2)[j]}$ is defined for each $\beta^{(2)[j]}$ in a way such that $\Xi^{(2)[j]} \subset \Xi^{(2)[j+1]}\ \forall i \in \{1, \ldots, N_\beta - 1\}$ with $\Xi^{(2)[N_\beta]} = \Xi^{(2)}$. This implies that the sequence of rules are such that $\beta^{(2)[j+1]}$ is less conservative than $\beta^{(2)[j]}$, with $\beta^{(2)[1]}$ being the most conservative and $\beta^{(2)[N_\beta]} = \beta^{(2)}$ being least conservative (or most aggressive). Note that each subset $\Xi^{(2)[j]}$ must be chosen in a manner that ensures that its corresponding rule $\beta^{(2)[j]}$ satisfies the properties of β mentioned previously.

For a given error pattern, the decoder starts the decimation procedure by passing messages using the map Φ_v^d and applying the decimation rule $\beta^{(1)}$ at the end of the third iteration after which the messages are reset to zero. Then the most conservative rule in the sequence $\{\beta^{(2)[j]}\}_j$, which is $\beta^{(2)[1]}$, is used after every two iterations (followed by resetting the messages) until no more nodes can be decimated. The map Φ_v^r then is used to decode the remaining non-decimated nodes. If the decoder still does not converge, then the whole decoding process is repeated by using a more aggressive rule $\beta^{(2)[2]}$ in place of $\beta^{(2)[1]}$. This decoding process continues until the decoder converges or until all rules in the sequence $\{\beta^{(2)[j]}\}_j$ have been used. Let N_b denote the number of decimated bits at the end of a decimation round. The decoding scheme can be summarized as follows. Note that this scheme is devised particularly for the case of $|\mathcal{M}| = 7$.

Note that the only criterion used by the decoder to decide when to use a more aggressive rule $\beta^{(2)[j]}$ on a given error pattern is whether the decoding has failed.

The rationale for using a distinct decimation rule $\beta^{(1)}$ specifically for the end of third iteration is based on a similar reasoning for FAIDs. Since it is the very first decimation that is being carried out, by virtue of Φ_v, we need at least three iterations to allow a variable node to pass $\pm L_3$ to be passed. Moreover, dynamics of MP involved with subsequent decimations is different, since we are allowing two iterations of MP before every subsequent decimation round. Therefore the decimation at the third iteration requires a specific decimation rule $\beta^{(1)}$

Also note the aspect of ADFAIDs using two iterations of MP before each decimation, in contrast to DFAIDs which uses only one iteration. The main disadvantage with allowing only one iteration of MP before performing each decimation round, is that the only possible messages that are passed (in the case of 7-level FAID) is $\pm L_3$ and $\pm L_1$. Due to

Algorithm 2 Adaptive decimation-enhanced FAID algorithm

1) Set $j = 1$. Note that Φ_c will always be used to update messages at the check node.

2) Initialize $\gamma_i = 0 \; \forall v_i \in V$.

3) Start the decimation procedure by passing messages for three iterations using Φ_v^d. If the decoder converges within those three iterations, STOP.

4) Apply decimation rule $\beta^{(1)}$ for every $v_i \in V$. Then reset all messages to zero and set $q = 0$.

5) Pass messages for two iterations using Φ_v^d for update at the non-decimated nodes. If the decoder converges within those two iterations, STOP.

6) Apply decimation rule $\beta^{(2)[j]}$ only on nodes v_i for which $\gamma_i = 0$. Then reset all messages to zero. If $N_b > q$, $q = N_b$ and go back to step 5, else go to step 7.

7) Pass messages using Φ_v^r on the nodes v_i for which $\gamma_i = 0$.

8) If decoder converges or has reached maximum allowed iterations, STOP. Else $j = j + 1$.

9) If $j > N_\beta$ STOP. Else go to step 2.

this, there will still be a large number of correct nodes in the graph that are not decimated which could lead to decoder failure for a slightly higher-weight error pattern. By increasing it two iterations for ADFAIDs, it is still small enough to help prevent the growth of wrong message strengths but sufficient to allow all levels in $\mathcal{M}$ to be passed, thereby increasing the chance for more nodes to get decimated. We found this to be important for enhancing the guaranteed error correction. On the other hand, the nice property of Lemma 3.1 will not hold for ADFAIDs.

3.3.4 Choice of Φ_v^r and Φ_v^d

For the proposed ADFAIDs, the map Φ_v^r is simply chosen to be the Φ_v of a particular FAID already known to be good on a given code, and for which we want to improve the guaranteed error correction capability. For the numerical results, Φ_v^r is chosen to be the Φ_v of a 7-level FAID defined by Table 2.2 which was identified as particularly good FAID in the previous chapter.

The choice of Φ_v^d on the other hand is non-trivial. It is designed based on analyzing messages that are passed within dense subgraphs that could potentially be trapping sets for a given FAID when errors are introduced in them under the isolation assumption. The rule is chosen under the premise that the growth of message strengths within the subgraph should be slow since many correct nodes in the subgraph would most likely be connected to error nodes, and multiple error nodes may be interconnected to each other in the subgraph (if the number of errors introduced is comparable to the size of the subgraph).

Explicit design methods for Φ_v^d are not discussed here, but we provide a particular Φ_v^d that was designed based on the above philosophy and used for the numerical results. It is an LT function (see Chapter 2, Section 2.2.1). Therefore we described by assigning values to elements in $\mathcal{M}$, $\mathscr{T}$, and $\mathcal{Y}$. The map is defined with the following assignments; $L_1 = 1.1$, $L_2 = 2.3$, $L_3 = 6.6$, $T_1 = 0.8$, $T_2 = 2.8$, $T_3 = 4$, $\mathrm{C} = 1.5$. This was found to be a good rule for decimation.

3.3.5 Analysis

We now some provide some preliminary analysis with ADFAID on a graph G that gives insight into the design of the decimation rules. For the analysis, we assume that the all-zero codeword is transmitted which is valid since the decoders considered are symmetric.

Lemma 3.2. *A node v_i can receive a $\pm L_3$ from its neighbor c_j in the first or second iteration after resetting the messages, only if all nodes in $\mathcal{N}(c_j) \backslash v_i$ have been decimated.*

Proof: By virtue of Φ_v^d, any non-decimated node can only send $\pm L_1$ in the first iteration, and $\pm L_2$ in the second iteration. Therefore, a node has to be decimated in order

to send $\pm L_3$ within the first two iterations. For the neighbor c_j to send a $\pm L_3$, we require $m(\mathcal{N}(c_j)\backslash v_i, v_i) = \{\pm L_3\}$. This is only possible if all nodes in $\mathcal{N}(c_j)\backslash v_i$ have been decimated. ■

Lemma 3.3. *If $\beta^{(2)[j]}(\mathrm{C}, L_3, -L_2, -L_2) = 1$ and if $\forall c_k \in \mathcal{N}(v_i)$ all error nodes in $\mathcal{N}(c_k)\backslash v_i$ are non-decimated, then a correct node v_i will be decimated if it receives an L_3 within the two iterations before a decimation round.*

Proof: Let c_k, c_m, and c_j be the three neighboring check nodes of node v_i. If node v_i receives L_3 from say c_j within the two iterations before a decimation round, then by Lemma 3.2, all nodes in $\mathcal{N}(c_j)\backslash v_i$ are decimated. Therefore, only c_k, and c_m can have error nodes as its neighbors. But if none of their neighboring error nodes are decimated, then in the worst case, v_i receives $-L_2$ from both c_m and c_r. Since $\beta^{(2)[j]}(\mathrm{C}, L_3, -L_2, -L_2) = 1$, the correct node v_i is decimated. ■

Due to Lemma 3.3, we ensure that $\beta^{(2)[j]}$ is defined such that $\beta^{(2)[j]}(\mathrm{C}, L_3, -L_2, -L_2) = 1$ for any j. Also note how resetting messages at the end of each decimation round can help with decimating more correct nodes due to the above Lemma. This leads to following important theorem.

Theorem 3.2. *If $\beta^{(2)[j]}(\mathrm{C}, L_3, -L_2, -L_2) = 1$ and no error node is decimated, then any correct node in the residual graph G' is connected to check nodes that have at least degree-two.*

Proof: Assume that a correct node v_i in residual graph G' is connected to a degree-one check node c_j. This implies that all nodes in $\mathcal{N}(c_j)\backslash v_i$ are decimated. By Lemma 3.2, this implies that node v_i would have received an L_3 from c_j. Also since no error node is decimated and $\beta^{(2)[j]}(\mathrm{C}, L_3, -L_2, -L_2) = 1$, then by Lemma 3.3, this node v_i would be decimated in a decimation round, in which case v_i does not belong to residual graph G', which is a contradiction. ■

The following corollary follows from the Theorem, which turns out to be crucial for the design of the sequence of rules $\beta^{(2)[j]}$.

Corollary 3.1. *If Theorem 3.2 holds and no error node in the residual graph G' is connected to a degree-one check node, then G' is a stopping set.*

We remark that if an error node in the residual graph G' is connected to a degree-one check node, it would receive L_3 in every iteration for the remainder of the decoding (again assuming no error nodes are decimated), and this will most likely lead to a decoder convergence esepcially for low-weight error patterns which we are focussing on. Therefore, if no error node is decimated, the decoder is more likely to fail when the residual graph G' is a stopping set.

The above remark is an important observation since we can now design the rules $\beta^{(1)}$ and the sequence $\{\beta^{(2)[j]}\}_j$ based on analyzing error patterns whose errors are entirely

contained in the minimal stopping sets of a given code. For instance, if our goal is to correct up to t-errors, then we consider all error patterns up to a weight t in the stopping sets in order to design $\beta^{(1)}$ and $\{\beta^{(2)[j]}\}_j$.

If FAID $\mathscr{F}$ with $\Phi_v = \Phi_v^r$ has a critical number of t+1 on a stopping set whose induced subgraph is H, then $\mathscr{F}^A$ is guaranteed to correct up to t errors introduced in H on the code if the residual graph is H. In other words, Φ_v^r is more likely to correct all error patterns up to weight-t on a particular code whose support lies in the stopping set present in the code, if it has a critical number of $t+1$ on the stopping set.

There are several more interesting observations related to the analysis that we didn't cover in this dissertation. For instance, we believe the there is a link between the size of the residual graph in relation to the size of the minimal stopping set, and decoder convergence. The guaranteed error correction capability may also be related to size of the minimal stopping set. One thing that is clear from the analysis though is that the failures of ADFAIDs are linked to stopping sets, and the decimation rules must be designed based on analyzing error configurations associated with them.

3.3.6 Discussion on design of decimation rules $\beta^{(1)}$ and $\beta^{(2)}$

The design of $\beta^{(1)}$ involves selecting the triples that should be included in $\Xi^{(1)}$, which depends on the number of errors we are trying to correct and the type of harmful subgraphs present in G. $\beta^{(1)}$ should be chosen to be conservative enough so that no error nodes are decimated. On the other hand, the design of $\beta^{(2)}$ not only involves selecting the triples that should be included in $\Xi^{(2)}$, but also determining a specific ordering on the triples that will be included in subsets $\Xi^{(2)[j]}$ which determine the sequence of rules $\{\beta^{(2)[j]}\}_j$ used starting from the least conservative rule, and this is dependent on the structure of the code. Both rules can be designed by analyzing them on errors introduced in stopping sets of the code. Fig. 3.3 shows an example of a $(12, 2)$ stopping set that is present on a $(732, 577)$ structured code that has a $d_{min} = 12$. This was one of the subgraphs used to derive the rules.

In order to specify the set $\Xi^{(1)}$, we just specify the message triples with the weakest values. For specifying $\Xi^{(2)}$ in a concise way, we shall introduce some notations. Let $\Xi^{(2)}$ be divided into two disjoint subsets, i.e., $\Xi^{(2)} = \Lambda \cup \Gamma$, where Λ is a subset that contains all triples $(L_3, m_2, m_3) \in \mathcal{M}^3$ such that $m_2, m_3 \geq -L_2$. Based on the analysis described previously, any $\Xi^{(2)[j]}$ defined should always have Λ as its subset, regardless of the code. The subset Γ, which is dependent on the code, is an ordered set whose ordering determines the subsets used to specify the sequence of rules $\{\beta^{(2)[j]}\}_j$.

3.3.7 Numerical Results and Discussion

Numerical results are provided in Fig. 3.4 and Fig. 3.5 for two codes: the well-known $(155, 64)$ Tanner code and a structured rate 0.753 $(732, 551)$ code constructed based on

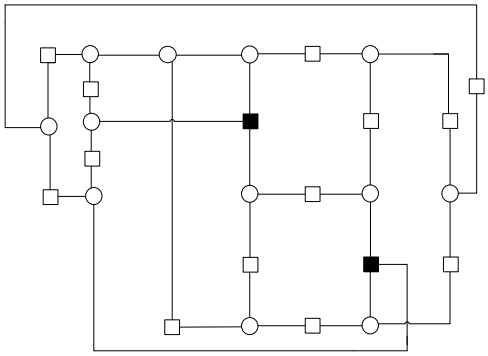


Figure 3.3: $(12, 2)$ Stopping set present in graph of the $(732, 551)$ $d_{min} = 12$ code

latin squares [72] with $d_{min} = 12$.

For the Tanner code, the set $\Xi^{(1)}$ contains all triples $(m_1, m_2, m_3) \in \mathcal{M}^3$ such that $(m_1, m_2, m_3) \geq (L_3, 0, 0)$ and $(m_1, m_2, m_3) \geq (L_2, L_2, L_1)$ (comparison is componentwise). For the high-rate structured code, $\Xi^{(1)}$ contains all triples such that $(m_1, m_2, m_3) \geq (L_3, L_1, -L_3)$, $(m_1, m_2, m_3) \geq (L_3, -L_1, -L_1)$, and $(m_1, m_2, m_3) \geq (L_2, L_1, L_1)$. $|\Xi^{(1)}| = 12$ for the Tanner code and $|\Xi^{(1)}| = 24$ for the $(732, 551)$ code.

The Γ sets in $\Xi^{(2)} = \Lambda \cup \Gamma$ for the Tanner code and high-rate structured code are shown in Table 3.3. The cardinalities of the subsets of $\Xi^{(2)}$ used by each of the two codes are $\{|\Xi^{(2)[j]}|\}_j = \{24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35\}$ and $\{|\Xi^{(2)[j]}|\}_j = \{23, 25, 26, 27, 29\}$ respectively. The maximum number of iterations allowed for BP and 7-level FAID, and for Φ_v^r of the 7-level ADFAID, was 100.

The significant improvement in the slope of the error floor by using the 7-level ADFAID is evident. For the Tanner code, it was verified that all 6-error patterns are corrected by the 7-level ADFAID while the 7-level FAID corrects all 5-errors and BP fails on 5-errors. For the high-rate structured code, no failed 5-error patterns were found in the region of simulation shown in Fig. 3.5, which is significant since the code has $d_{min} = 12$. This shows that for certain high-rate codes whose graphs are relatively dense and for which it becomes difficult to ensure high d_{min} in the code, the FAIDs with adaptive decimation can possibly come close to achieving the guaranteed error correction of maximum likelihood decoding. Note that the 7-level ADFAIDs are still 3-bit message passing decoders which have reasonable complexity, and that is still lower than BP.

3.4 Conclusions

In this chapter, we introduced the technique of decimation and incorporated it into the framework of FAIDs for two main purposes: 1) to use as a tool to make FAIDs more amenable to analysis, and 2) to enhance the guaranteed error correction of codes from

(a) Subset Γ of $\Xi^{(2)}$ designed for $(155, 64)$ Tanner code

m_1	m_2	m_3
L_2	L_2	L_2
L_2	L_2	L_1
L_2	L_2	0
L_2	L_1	L_1
L_2	L_1	0
L_2	L_2	$-L_1$
L_2	L_1	$-L_1$
L_2	0	0

(b) Subset Γ of $\Xi^{(2)}$ designed for $(732, 551)$ code

m_1	m_2	m_3
L_2	L_2	L_2
L_2	L_2	L_1
L_2	L_1	L_1
L_2	L_2	0
L_2	L_2	$-L_1$
L_2	L_1	0
L_1	L_1	L_1
L_2	0	0
L_2	L_1	$-L_1$
L_1	L_1	0
L_2	L_2	$-L_2$
L_2	0	$-L_1$
L_1	L_1	$-L_1$
L_1	0	0

Table 3.3: Definition of Subset Γ of $\Xi^{(2)}$ for both codes. Note that the ordering of the triples matters since the subsets are chosen with increasing cardinality based on the ordering.

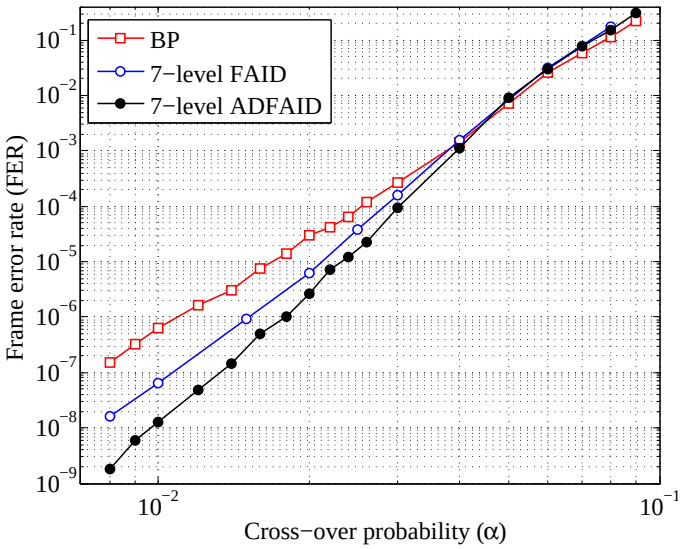


Figure 3.4: FER performance comparison on the $(155, 64)$ Tanner code

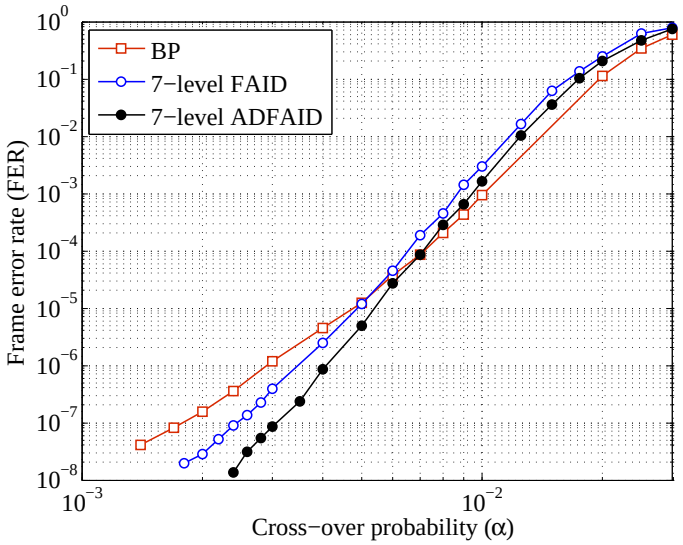


Figure 3.5: FER comparison on the $(732, 551)$ structured code with $d_{min} = 12$

what is achievable by FAIDs. In contrast to many existing approaches related to decimation, we incorporated it by allowing the decoder to decide which nodes to decimate after passing messages for some iterations. We proposed decimation-enhanced FAIDs for column-weight-three codes and illustrated that decimation could be used to reduce the number of iterations required to correct a fixed number of errors while maintaining the good performance of the original FAID. We also showed how decimation can aid in the analysis of the decoders. By proposing a simple decimation scheme which involves performing the decimation at the end of third iteration using some rule, and restarting the decoder, one can easily analyze whether a node initially in error would get decimated or not. We derived conditions on the graph such that errors introduced in a 8-cycle present in a graph G are not decimated by the proposed DFAIDs.

Although the analysis was provided only for a particular error configuration involving the 8-cycle, this can easily be extended to other types of error configuration. One major motivation for using decimation as a tool for analysis was that although FAIDs were shown to be capable of surpassing BP and empirically known to have good performance, analyzing FAIDs in terms of guaranteed error correction remains a challenge. For instance, we know that the particularly good 7-level FAID which surpasses BP on the $(155, 64)$ Tanner code achieves a guaranteed error correction of 5 errors. The question is can we actually prove that 7-level FAID guarantees correction of 5 errors under certain conditions of the graph. We believe that decimation has certainly brought us a step closer to this goal. For instance, although not included in this dissertation, we have begun to analyze all possible 5-error configurations and found that the conditions on the graph in order to ensure that none of the errors get decimated, were simple conditions. Future

work in this regard involves linking the analytical results on decimation to guaranteed error correction capability.

We also proposed adaptive decimation-enhanced FAIDS for LDPC codes. By using decimation in an adaptive way, and choosing the appropriate decimation rules, the guaranteed error correction of FAIDs can be simply enhanced. We provided some preliminary analysis linking failures of ADFAIDs to stopping sets and therefore used stopping sets to design the decimation rules. We designed ADFAIDs for two test codes and was able to significantly increase the guaranteed error correction compared to FAIDs, which also improved the slopes in their error-rate performance. Future work includes developing a more rigorous and systematic method to design decimation rules as well as to gain a deeper understanding in the relationship between stopping sets and decoder convergence on residual graphs.

Finally we remark that although this entire chapter focused on column-weight-three codes, the underlying principles and philosophy related to decimation can still be used in decoder designs for codes with higher column-weights and also for FAIDs that use higher levels.

Conclusions and Perspectives

With many next generation communication and data storage systems now requiring very low error-rates, faster processing speeds, lower memory usage, and reduced chip area, the need for low-complexity decoding algorithms that take finite precision into account while still achieving the required target error-rates on LDPC codes, has now become crucially important to address. Most of the existing approaches on decoding rely on modifying or taking quantized versions of the BP algorithm with the goal of minimizing the loss compared to the theoretical un-quantized performance of the BP (or floating-point BP). This dissertation is primarily concerned with the development of novel iterative decoding algorithms that take finite precision into account and yet are able to surpass the floating-point BP.

In Chapter 2, we introduced a novel approach to finite precision iterative decoding for LDPC codes over the BSC. The proposed class of decoders which we referred to as FAIDs, propagate messages that are not quantized probabilities but are simply levels belonging to an arbitrary alphabet. In addition, the variable node update functions do not mimic BP but are rather designed to handle potentially harmful neighborhoods in the graph so that a better guaranteed error correction can be achieved. We illustrated how FAIDs differed from existing quantized decoders, and provided 3-bit precision FAIDs that are capable of surpassing the floating-point BP in the error floor. Moreover, these 3-bit precision FAIDs have the capability to surpass BP not just on a single code but several codes. We introduced the important notion of isolation assumption which allows analyzing MP of different decoders on isolated subgraphs that are potential trapping sets. The numerical results presented show that FAIDs can surpass BP without any compromise in decoding latency and hence are able to achieve a superior performance at a much lower complexity. Regarding the design of FAIDs, we showed that using density evolution to optimize for the best thresholds did not lead to good FAIDs. We instead provided a semi-heuristic-based method that is not code-specific but relies on the knowledge of potentially harmful subgraphs, in order to identify a set of candidate FAIDs, one or several of which are potentially good for any column-weight-three code. Due to their superior performance as well as low complexity, FAIDs could prove to be attractive in a wide

range of communication systems.

Although the entire framework of the proposed FAIDs in this dissertation focused on column-weight-three codes, many of the underlying ideas related to design of FAIDs are still applicable for codes with higher column-weight. The design of FAIDs for column-weight-four codes as well as allowing non-binary alphabets for the channel values are important to consider especially for magnetic recording applications, and are scope for future work.

In Chapter 3, our main goals were to further enhance the guaranteeed error correction with an attempt to approach the limits established by ML decoding as well as to make FAIDs more amenable to analysis. In order to achieve these goals, we proposed the technique of decimation both as a tool for analysis as well as for improving guaranteed error correction. Decimation is a technique based in statistical physics which involves fixing certain bits value to a specific value. In contrast to existing approaches related to decimation, we incorporated it into the MP of FAIDs thereby allowing the FAIDs to decide which nodes to decimate based on the messages they passed for a certain number of iterations. Using the $(155, 64)$ Tanner code as an example, we first provided a simple decimation scheme that reduced the number of iterations required to guarantee a correction of 5 errors on the code. But we will also illustrated how analytical results could be obtained with the help of the decimation scheme. As an example, we derived conditions on the graph such that errors introduced in the 8-cycle do not get decimated. The main future work in this context, would be to extend the analysis for all possible 5-error configurations, and link the results on decimation to guaranteed error correction capability.

We also proposed ADFAIDs for column-weight-three codes, and illustrated how they could increase the guaranteed error correction compared to FAIDs. The numerical results on the two test codes validated their efficacy, and in fact on one of the test codes which had $d_{min} = 12$, no failed 5-errors were detected in the region of simulation for which both the original 7-level FAID and BP failed on 5-errors. Besides, on both codes, the slope of the error floors was significantly increased. We also provided preliminary analysis that suggested that their failures are linked to the stopping sets of small size present in the code. This observation was crucial for the design of the decimation rules. Future work includes developing a more rigorous and systematic method to design decimation rules as well as to gain a deeper understanding in the relationship between stopping sets and decoder convergence on residual graphs. Also as part of future work, it would be interesting to investigate if decimation can be applied to column-weight-four codes and see if it can give the same benefits in terms of analysis as well as improving the guaranteed error correction of the code.

Bibliography

[1] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[2] R. W. Hamming, "Error detecting and error correcting codes," *Bell Sys. Tech. Journal*, vol. 29, pp. 147–160, April, 1950.

[3] M. J. E. Golay, "Notes on digital coding," *Proc. IEEE*, 37, pp. 657, 1949.

[4] D. E. Muller, "Application of boolean algebra to switching circuit design and to error detection," *IRE Trans. on Electronic Computers*, vol. 3, pp. 6–12, 1954.

[5] Irving S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Trans. of the IRE Professional Group on Inform. Theory*, vol 4, pp. 38–49, 1954.

[6] C.E. Shannon, "A Mathematical Theory of Communication," *Bell Sys. Tech. Journal*, vol. 27, pp. 379–423 and pp. 623–656, Jul. and Oct. 1948.

[7] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, Sep. 1959.

[8] R. C. Bose, D. K. Ray-Chaudhuri, "On a class of Error Correcting Binary Group Codes," *Information and Control*, vol. 3, pp. 68–79, Mar. 1960.

[9] I. S. Reed, G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Indust. Appl. Math.*, vol. 8, pp. 300–304, Jun. 1960.

[10] E. R. Berlekamp, "Factoring Polynomials Over Finite Fields," *Bell Sys. Tech. Journal*, vol. 46, pp. 1853–1859, 1967.

[11] P. Elias, "Coding for noisy channels," *IRE Conv. Rep.*, Pt. 4, pp. 37–47, 1955.

[12] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 4, pp. 260-269, Apr. 1967.

[13] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. 20, no. 3, pp. 284–287, Mar. 1974.

[14] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," *Proc. Int. Conf. on Commun.*, pp. 1064–1070, May 1993.

[15] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," *Proc. 5th IMA Conf. Cryptography and Coding*, no. 1025, pp. 100–111, Dec. 1995.

[16] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[17] N. Wiberg, " Codes and decoding on general graphs," *PhD thesis*, Linköping University, Sweden, 1996.

[18] F. R. Kschischang, B. J. Frey and H.-A. Loeliger, "Factor graphs and the sum-Product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, pp. 498–519, Feb. 2001.

[19] T. Richardson and R. Urbanke, "The capacity of LDPC codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[20] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

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

[22] J. Lee and J. Thorpe, "Memory-efficient decoding of LDPC codes," *Proc. Int. Symp. Inform. Theory*, Adelaide, Australia, pp. 459–463, Sep. 2005.

[23] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.

[24] A. Barg, and G. Zémor, "Error exponents of expander codes," *IEEE Trans. Inf. Theory*, vol. 48, no 6, pp. 1725–1729, 2002.

[25] D. Burshtein and G. Miller, "Expander graph arguments for message passing algorithms," *IEEE Trans. Inf. Theory*, vol. 47, pp. 782–790, Feb. 2001.

[26] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to Decode Binary linear codes ," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.

[27] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," *Proc. ISTA*, Ambleside, England, 2001

[28] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, pp. 1788–1793, 2004.

[29] J. Thorpe, "Low density parity-check codes constructed from protographs," *IPN Progress Report*, 42-154, Aug. 15, 2003.

[30] D. Divsalar, S. Dolinar, C.R. Jones, K. Andrews, "Capacity-approaching protograph codes," *IEEE J. Select. Areas Commun.*, vol. 27, no. 6, Aug. 2009.

[31] L. Lan, L. Zeng, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "Construction of quasi-cyclic LDPC codes for AWGN and Binary Erasure channels: A finite field approach," *IEEE Trans. Inf. Theory*, vol. 53, no. 7, pp. 2429–2458, Jul. 2007.

[32] Y. Kou, S. Lin and M. C. Fossorier, "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results," *IEEE Trans. on Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[33] B. Vasic, O. Milenkovic, "Combinatorial construction of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 50, pp. 1156–1176, Jun. 2004.

[34] R. M. Tanner, "Convolutional codes from quasi-cyclic codes: A link between the theories of block and convolutional codes," *Univ. Calif. Santa Cruz, Tech. Rep.*, 1987.

[35] A. Feltström and K. S. Zigangirov, "Periodic time-varying convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 5 pp. 2181–2190, Sep. 1999.

[36] M. Lentmaier, A. Sridharan, D. J. Costello, and K. S. Zigangirov, "Iterative Decoding Threshold Analysis for LDPC Convolutional Codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, 5274–5289, Oct. 2010.

[37] S. Lin and D. J. Costello, "*Error Control Coding: Fundamentals and Applications*," second edition, Prentice Hall: Englewood Cliffs, NJ, 2005.

[38] E. Sharon, S. Litsyn and J. Goldberger, "Efficient serial message-passing schedule for LDPC decoding," *IEEE Trans. on Inf. Theory*, pp. 4076-4091, Nov. 2007.

[39] J. Pearl, *Probablisitic Reasoning in Intelligent Systems*, San Francisco, CA: Kaufmann, 1988.

[40] B. J. Frey, *Graphical models for machine learning and digital communication*, Cambridge, MA, USA: MIT Press, 1998.

[41] D. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes,", *Electronic Notes in Theoretical Computer Science*, vol. 74, 2003.

[42] B. J. Frey, R. Koetter, and A. Vardy, "Signal-space characterization of iterative decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 766–781, Feb. 2001.

[43] G. D. Forney, Jr., R. Koetter, F. R. Kschischang, and A. Reznick, "On the effective weights of pseudocodewords for codes defined on graphs with cycles," *Codes, systems and graphical models*, pp. 101–112, Springer, 2001.

[44] C. Di, D. Proietti. I. E. Teletar, T. J. Richardson, and R. L. Urbanke, "Finite length analysis of low-density parity-check codes on the Binary Erasure channels," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1576–1579, Jun. 2002.

[45] A. Orlitsky, K. Viswanathan, and J. Zhang, "Stopping sets distribution of LDPC code ensemble," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 929-953, Mar. 2005.

[46] T. Richardson, "Error floors of LDPC codes," *Proc. 41st Annual Allerton Conf. on Commun., Control and Computing*, 2003.

[47] M.G. Stepanov, V. Chernyak, M. Chertkov, B. Vasić, "Diagnosis of weakness in error correction codes: a physics approach to error floor analysis," *Phys. Rev. Lett.*, vol. 95, no. 22, Nov. 2005.

[48] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasić, "Error floors of LDPC codes on the binary symmetric channels," *Proc. IEEE Int. Conf. on Commun.*, Istanbul, Turkey, pp. 1089–1094, Jun. 2006.

[49] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets for Array-Based LDPC Codes," *IEEE Trans. Inf. Theory*, , pp. 6261–6268, Jun., 2007.

[50] S. K. Chilappagari and B. Vasić, "Error-correction capability of column-weight-three LDPC codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 5, pp. 2055–2061, May 2009.

[51] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC code," *Corr arXiv:cs/0512078v1*, Dec. 2005.

[52]] C. A. Kelley and D. Sridhara, "Pseudocodewords of Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 53, no. 11, pp. 4013–4038, Nov. 2007.

[53] R. Smarandache, A. Pusane, P. Vontobel, and D.J. Costello, "Pseudocodeword performance analysis for LDPC convolutional codes," *IEEE Trans. Inform. Theory*, vol. 55, no. 6, pp. 2577–2598, Jun. 2009.

[54] S. Laendner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes,"*Proc. IEEE Int. Conf. on Wireless Networks, Commun. and Mobile Computing*, Hawaii, USA, pp. 630-635, Jun. 13-16, 2005.

[55] A. McGregor and O. Milenkovic, "On the hardness of approximating stopping and trapping sets in LDPC codes," *IEEE Trans. Inform.Theory*, vol. 56, no. 4, pp. 1640–1650, Apr. 2010.

[56] O. Milenkovic, E. Soljanin, and P. Whiting, "Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles," *IEEE Trans. Inf. Theory*, vol. 53, no. 1, pp. 39-55, Jan. 2007.

[57] C. Cole, S. Wilson, E. Hall, and T. Giallorenzi, "A general method for finding low error rates of LDPC codes," [online] http://arxiv.org/abs/cs/0605051, May 2006.

[58] S. Abu-Surra, D. Declercq, D. Divsalar, and W. Ryan, "Trapping set enumerators for specific LDPC codes," *Proc. Inform. Theory and Applications Workshop*, San Diego, CA, pp. 1–5, Feb. 2010.

[59] E. Rosnes and O. Ytrehus, "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices," *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4167–4178, Sept. 2009.

[60] M. Karimi and A. H. Banihashemi, "Efficient algorithm for finding dominant trapping sets of LDPC codes," *IEEE Trans. Inf. Theory*, to appear, 2012.

[61] X. Zhang and P. H. Siegel, "Efficient algorithms to find all small error-prone substructures in LDPC Codes," *Proc. IEEE Globecom.*, Houston, TX, pp. 1–6, Dec. 2011.

[62] B. Vasić, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," *Proc. Allerton Conf. on Commun., Control and Computing*, pp. 1–7, Sep. 2009.

[63] S. K. Chilappagari, M. Chertkov, M. G. Stepanov, and B. Vasić, "Instanton-based techniques for analysis and reduction of error floors of LDPC codes," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 855–865, Aug. 2009.

[64] M. Ivkovic, S. K. Chilappagari, B. Vasić, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug. 2008.

[65] S. K. Chilappagari, D. V. Nguyen, B. Vasić, and M. Marcellin, "Error-correction capability of column-weight-three LDPC codes under the Gallager A Algorithm: Part II," *IEEE Trans. Inf. Theory*, vol. 56, no. 6, pp. 2626–2639, May 2010.

[66] S. K. Chilappagari, A. R. Krishan and B. Vasić, "LDPC codes which can correct three errors under iterative decoding,"*IEEE Inform. Theory Workshop*, Porto, Portugal, pp. 406–410, May 2008.

[67] J. S. Yedidia, W. T. Freeman, and Y.Weiss, "Constructing free energy approximations and generalized belief propagation algorithms," *IEEE Trans. Inform. Theory*, vol. 51, pp. 2282–2312, Jul. 2005.

[68] M. Chertkov and V. Y. Chernyak, "Loop calculus helps to improve belief propagation and linear programming decodings of low-density-parity-check codes," in *Proc. 44th Annual Allerton Conf. on Commun., Control and Computing*, Monticello, IL, Sep. 2006.

[69] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242-1248, Aug. 2004.

[70] Y. Lu, C. Measson, and A. Montanari, "TP decoding," in *Proc. 45th Annual Allerton Conf. on Commun., Control and Computing*, Monticello, IL, USA, Sept. 2007.

[71] R. Asvadi, A. H. Banihashemi and M. Ahmadian-Attari, "Lowering the error floor of LDPC codes using cyclic liftings," *IEEE Trans. Inf. Theory*, vol. 57, no. 4, pp. 2213–2224, Apr. 2011.

[72] D. V. Nguyen, S. K. Chilappagari, M. W. Marcellin, and B. Vasic, "On the construction of structured LDPC Codes free of small trapping sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr. 2012.

[73] B. Kurkoski and H. Yagi, "Quantization of binary-input Discrete Memoryless Channels with applications to LDPC decoding," [Online]. Available: http://arxiv.org/abs/1107.5637

[74] N. Varnica, M. Fossorier, and A. Kavcic, "Augmented belief propagation decoding of low-density parity check codes," *IEEE Trans. Commun.*, vol. 55, no. 7, pp. 1308–1317, Jul. 2007.

[75] T. Hehn, J. B. Huber, S. Laendner, and O. Milenkovic, "Multiple-bases belief-propagation for decoding of short block codes," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT '07)*, Nice, France, pp. 311–315, Jun. 2007.

[76] A. I. Vila Casado, M. Griot, and R. D. Wesel,"LDPC decoders with informed dynamic scheduling," *IEEE Trans. Commun.*, vol. 58, no. 12, pp. 3470–3479, Dec. 2010.

[77] Y. Wang, J. S. Yedidia, and S. C. Draper, "Multi-stage decoding of LDPC codes," *Proc. IEEE Int. Symp. Inform. Theory*, pp. 2151–2155, Jul. 2009.

[78] S. Laendner, and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes ," *Proc. Int. Conf. Wireless Networks, Commun., and Mobile Commun.*, Maui, HI, pp. 630–635, Jun. 2005.

[79] Y. Han, and W. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1663–1673, Jun. 2009.

[80] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, and M. Wainwright, "Lowering LDPC error floors by postprocessing," *Proc. IEEE Global Telecommun. Conf.*, pp. 1–6, Dec. 2008.

[81] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, and M. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. Commun.*, vol. 57, no. 11, pp. 3258–3268, Nov. 2009.

[82] B. Butler, and P. H. Siegerl, "Error floor approximation for LDPC Codes in the AWGN channel," *submitted to IEEE Trans. Inf. Theory*. [online]. Available: http://arxiv.org/pdf/1202.2826.pdf

[83] F. Zarkeshvari, A. H. Banihashemi, "On implementation of min-sum and its modifications for decoding LDPC codes," *IEEE Trans. Commun. lett.*, vol. 53, no. 4, pp. 549–554, Apr. 2005.

[84] X. Zhang and P. H. Siegel, "Quantized min-sum decoders with low error floor for LDPC Codes," *Proc. IEEE Int. Symp. Inform. Theory*, Boston, MA, pp.2871–2875, Jul. 2012

[85] X. Zhang and P. H. Siegel, "Will the real error floor please stand up?," *IEEE Int. Conf. Signal Processing and Commun.*, Bangalore, India, pp. 1–5, Jul. 2012.

[86] S. K. Planjery, S. K Chilappagari, B. Vasić, D. Declercq, and L. Danjean, "Iterative decoding beyond belief propagation," *Proc. Inform. Theory and App. Workshop*, San Diego, CA, Jan. 2010.

[87] L. Sassatelli, D. Declercq, S. K. Chilappagari, and B. Vasić, "Two-bit message passing decoders for LDPC codes over the Binary Symmetric channel,"*IEEE Int. Symp. Inform. Theory*, Seoul, Korea, pp. 2156–2160, June 2009.

[88] S. K. Planjery, D. Declercq, L. Danjean, B. Vasic, "Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders," *Electron. Lett.* vol. 47, no. 16, pp. 919–921, Aug. 2011.

[89] S. K. Planjery, D. Declercq, S. K. Chilappagari, and B. Vasic, "Multilevel decoders surpassing belief propagation on the binary symmetric channel," *Proc. Int. Symp. Inform. Theory*, Austin, TX, pp. 769–773, Jul. 2010.

[90] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders for LDPC codes surpassing floating-point iterative decoders," *IET Electron. Lett.*, vol. 47, no. 16, Aug. 2011.

[91] G. Kuperberg, "Symmetries of plane partitions and the permanent-determinant method," *J. Comb. Theory A*, 68, pp. 115–151, 1994.

[92] D. Declercq, L. Danjean, E. Li, S. K. Planjery, and B. Vasic, "Finite alphabet iterative decoding (FAID) of the (155,64,20) Tanner code," *Proc. Int. Symp. Turbo Codes Iter. Inform. Process.*, Brest, France, pp. 11–15, Sep. 2010.

[93] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[94] L. Danjean, D. Declercq, S. K. Planjery, and B. Vasic, "On the selection of finite alphabet iterative decoders for LDPC codes on the BSC," *Proc. IEEE Inform. Theory Workshop*, Paraty, Brazil, pp. 345–349, Oct. 2011.

[95] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian, "Solving constraint satisfaction problems through belief propagation-guided decimation," in *Proc. Allerton Conf. on Commun.*, Monticello, IL, 2007.

[96] A. G. Dimakis, A. A. Gohari, M. J. Wainwright, "Guessing Facets: Polytope Structure and Improved LP Decoder," *IEEE Trans. Inf. Theory*, vol. 55, no. 8, pp. 3479–3487, Aug. 2009.

[97] M. Chertkov, "Reducing the error floor," *Proc. IEEE Inform. Theory Workshop*, pp. 230–235, Sep. 2007.

[98] S. K. Planjery, B. Vasić,D. Declercq, "Decimation-enhanced finite alphabet iterative decoders for LDPC codes on the BSC," *Proc. Int. Symp. Inform. Theory*, St. Petersburg, Russia, pp. 2383–2387, Jul. 2011.

[99] D. Declercq, B. Vasić, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders approaching maximum likelihood performance on the Binary Symmetric channel," *Proc. IEEE Inform. Theory and App. Workshop*, San Diego, CA, pp. 23–32, Feb. 2012.