3-2018

# Design and Analysis of Graph-based Codes Using Algebraic Lifts and Decoding Networks

Allison Beemer
*University of Nebraska - Lincoln*, allison.beemer@huskers.unl.edu

# DESIGN AND ANALYSIS OF GRAPH-BASED CODES USING ALGEBRAIC LIFTS AND DECODING NETWORKS

by

Allison Beemer

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Mathematics

Under the Supervision of Professor Christine A. Kelley

Lincoln, Nebraska

March, 2018

# DESIGN AND ANALYSIS OF GRAPH-BASED CODES USING ALGEBRAIC LIFTS AND DECODING NETWORKS

Allison Beemer, Ph.D.

University of Nebraska, 2018

Adviser: Christine A. Kelley

Error-correcting codes seek to address the problem of transmitting information efficiently and reliably across noisy channels. Among the most competitive codes developed in the last 70 years are low-density parity-check (LDPC) codes, a class of codes whose structure may be represented by sparse bipartite graphs. In addition to having the potential to be capacity-approaching, LDPC codes offer the significant practical advantage of low-complexity graph-based decoding algorithms. Graphical substructures called trapping sets, absorbing sets, and stopping sets characterize failure of these algorithms at high signal-to-noise ratios.

This dissertation focuses on code design for and analysis of iterative graph-based message-passing decoders. The main contributions of this work include the following: the unification of spatially-coupled LDPC (SC-LDPC) code constructions under a single algebraic graph lift framework and the analysis of SC-LDPC code construction techniques from the perspective of removing harmful trapping and absorbing sets; analysis of the stopping and absorbing set parameters of hypergraph codes and finite geometry LDPC (FG-LDPC) codes; the introduction of multidimensional decoding networks that encode the behavior of hard-decision message-passing decoders; and the presentation of a novel Iteration Search Algorithm, a list decoder designed to improve the performance of hard-decision decoders.

## ACKNOWLEDGMENTS

First and foremost, thank you to my adviser, Christine Kelley, for her unwavering support and good humor. Her advice has been an invaluable resource, and I truly cannot thank her enough for her dedication to my work and mathematical upbringing.

Thank you to my readers, Jamie Radcliffe and Judy Walker, and to Mark Brittenham, Myra Cohen, and Khalid Sayood for showing interest in my work and generously contributing their time to be on my supervisory committee.

The Whitman College math department has my gratitude for challenging me and encouraging me to continue on in mathematics. Deserving of particular thanks are: Barry Balof, for convincing me to be a math major, Russ Gordon, for an analysis course that got me through a qualifying exam, David Guichard, for teaching me the value of productive struggle, and Pat Keef, for his enthusiastic approach to teaching mathematics.

Endless thanks are due to everyone who has made Nebraska feel like home. Among these (numerous) people are the brilliant officewomen of Avery 232, who have added a whole bunch of brightness to my life. Thank you also to family and friends who have supported me from near and far. I appreciate you all so much.

Finally, thank you to Eric Canton, for an unrelenting belief in my abilities and ambitions, for knowing how to make me laugh at the unlikeliest of times, and for brewing me a lot of coffee.

# Table of Contents

# Chapter 1

# Introduction

Mathematical coding theory addresses the problem of transmitting information efficiently and reliably across noisy channels; that is, finding methods of encoding information with the minimum redundancy required to protect against noise during transmission. In 1948, Claude Shannon pioneered the field by proving that for every channel across which one might wish to send information, there exist methods of encoding and decoding that are both arbitrarily efficient and reliable [1]. In particular, every channel has a capacity, and there exist codes and decoders for every rate below capacity such the probability of error after decoding is as small as desired. Since this revelation, coding theorists have worked to find code ensembles and decoders satisfying these conditions.

Among the most competitive codes developed in the last 70 years are low-density parity-check (LDPC) codes, a class of codes whose structure may be represented by sparse bipartite graphs, or, equivalently, sparse parity-check matrices [2,3]. LDPC codes were first introduced by Gallager in his 1962 Ph.D. dissertation [2], and gained renewed interest in the coding community with the advent of turbo codes in 1993 [4]. Graph-based codes may now be found in a variety of communications applications, ranging from video streaming to data storage.

Among their strengths, LDPC codes offer the significant practical advantage of low-complexity graph-based decoding algorithms. This dissertation focuses on code design for iterative graph-based message-passing decoders. Graphical substructures called *trapping sets* characterize failure of message-passing decoding algorithms for channels with low error probability – that is, high *signal-to-noise ratios* (SNR) – creating what is called the *error floor region* of bit error rate (BER) curves. We examine trapping sets – as well as the related substructures of *absorbing sets* and *stopping sets* – and methods for their removal with reference to a variety of code families, including spatially-coupled LDPC, finite geometry LDPC, and hypergraph codes. Finally, we introduce a multidimensional network framework for the general analysis of iterative decoders.

Regular LDPC codes were shown to be asymptotically good in Gallager's original work [2]; in [5], irregular LDPC codes were optimized to be capacity-achieving. More recently, LDPC codes have been used to construct a class of codes called *spatially-coupled LDPC* (SC-LDPC) codes, which have been shown to exhibit *threshold saturation*, resulting in capacity-approaching, asymptotically good codes [6–8]. We show that the eponymous coupling step of the construction process for these codes may be exploited to remove harmful trapping sets in the resulting code. Furthermore, we show that the variety of existing construction methods for SC-LDPC codes may be unified using the language of algebraic graph lifts, allowing for a more systematic removal of harmful absorbing sets as well as simplified analysis of construction techniques.

The inherent structure of finite geometry LDPC (FG-LDPC) codes [9] and codes constructed from hypergraphs [10] allows us to determine harmful stopping and absorbing set parameters for these classes of codes. We show that FG-LDPC codes constructed from the line-point incidence matrices of finite Euclidean and projective planes have smallest absorbing sets whose parameters meet the lower bounds shown in [11], and that these smallest absorbing sets are *elementary*. We then present new results on the number of erasures correctable in regular hypergraph codes, and give a proof of the existence of infinite sequences of regular hypergraphs with expansion-like properties by introducing a construction for *hypergraph lifts*.

Finally, we present a *multidimensional decoding network* framework that encodes the behavior of any hard-decision message-passing decoder for a code with a given graph representation. Notably, the trapping sets of a code may be easily seen using the decoding network of the code and chosen decoder. Decoding networks with a particular transitivity property allow for a simplified application of results; we show that every code with dimension at most two less than its block length has a transitive representation under the Gallager A decoding algorithm. We also introduce the *decoding diameter*, *aperiodic length*, and *period* of a decoding network as parameters containing essential information about the interaction of the code and decoder. Employing these ideas, we then present an Iteration Search Algorithm, a type of list decoder designed to improve the performance of hard-decision decoders for which decoding network parameters are known.

The dissertation is organized as follows. Chapter 2 introduces the background necessary for material in later chapters. In Chapter 3, we unify the construction of SC-LDPC codes within a general algebraic graph lift framework, and present strategies for removing trapping and absorbing sets during the construction process. Parts of Chapter 3 appear in joint work with Kelley [12] and with Habib, Kelley, and Kliewer [13]. Chapter 4 focuses on bounds on the sizes of stopping and absorbing sets in FG-LDPC codes and hypergraph codes. This work appears in joint work with Haymaker and Kelley [14] and Mayer and Kelley [15]. A multidimensional network framework for trapping set analysis is introduced and analyzed in Chapter 5, and a novel list decoder for improved performance of hard-decision decoders is also introduced. Work in Chapter 5 also appears in [16] and [17]. Chapter 6 concludes the dissertation.

# Chapter 2

# Preliminaries

Suppose we have some information we wish to transmit, represented by vectors of length $k$ over the field with $q$ elements: that is, by vectors in $\mathbb{F}_q^k$ where $q$ is a prime power. Based on the *channel* over which these vectors are transmitted, some amount of error will be introduced. For example, a wireless signal may have noise introduced by the hardware in its transmitter or receiver, and information encoded in a CD may experience errors when the CD is scratched. Errors may take many forms, including erasures, bit switches, or the addition of real-valued noise from a normal distribution. These channel types may be modeled mathematically by the *q-ary erasure channel (QEC)*, the *q-ary symmetric channel (QSC)*, and the *additive white Gaussian noise (AWGN) channel*, respectively. This dissertation will focus on *binary* codes, for which the size of the field is $q = 2$. Figures 2.1 and 2.2 show representations of the binary erasure channel (BEC), in which binary bits are erased with some probability, and the binary symmetric channel (BSC), in which binary bits are flipped with some probability.

In order to protect from channel noise, redundancy is introduced into our information vectors by embedding them within $\mathbb{F}_2^n$, where $n > k$: an $[n, k]$ *binary linear block code* $C$ is a linear subspace of $\mathbb{F}_2^n$ of dimension $k$, and thus can be viewed as the kernel of a (non-unique) *parity-check matrix*, generally denoted by $H$. We call $n$ the *block length* of $C$, and $k$ its *dimension*. The *rate* of $C$ is given by $k/n$, and is a measure of the code's efficiency.

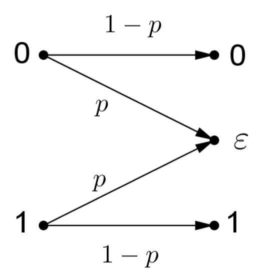


**Figure 2.1:** A representation of the binary erasure channel (BEC), over which binary bits are erased with probability $p$, and transmitted reliably with probability $1-p$.

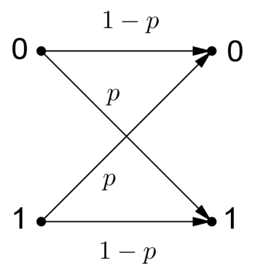


**Figure 2.2:** A representation of the binary symmetric channel (BSC), over which binary bits are flipped with probability $p$, and transmitted reliably with probability $1-p$.

We may also consider $C$ as the row space of a *generator matrix* $G$. The *dual code* of the code $C$, denoted $C^{\perp}$, is the subspace of $\mathbb{F}_2^n$ formed by the vectors whose inner product with all elements of $C$ is equal to zero. Notice that the rows of a parity-check matrix of $C$ are elements of $C^{\perp}$.

Classical coding theory addresses the problem of finding subspaces of low codimension (i.e. high rate) that also have high *minimum distance* – the minimum *Hamming weight* of a nonzero vector in the subspace, measured as the minimum number of nonzero coordinates. Since a linear code is a subspace of $\mathbb{F}_2^n$, the minimum Hamming weight is equal to the minimum Hamming distance between two vectors in the space. We denote the minimum distance of a code $C$ by $d_{\min}(C)$. Intuitively, if codewords are more "spread apart" in the space, we have a good chance of decoding a received word to the correct codeword, as long as the channel did not take it too far from the originally transmitted word. More formally, if we receive the word $\mathbf{y}$ from the channel, decoding to the closest codeword $\hat{\mathbf{c}}$ in terms of Hamming distance will yield the correct codeword as long as there are fewer than $\lfloor d_{\min}(C)/2 \rfloor$ errors in the received word. This is called *Nearest Neighbor* decoding. In the case that the channel is the QSC and all codewords are equally likely, this is equivalent to

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}\in C} P(\mathbf{y}\ \text{ received} \mid \mathbf{c}\ \text{ sent}),$$

called *Maximum Likelihood (ML)* decoding.

From a choice of binary parity-check matrix $H$, a binary linear code may be represented graphically by viewing $H$ as the (simplified) adjacency matrix of a bipartite graph: one vertex set consists of *variable nodes* corresponding to the columns of $H$, and the other vertex set consists of *constraint* or *check nodes* corresponding to the rows of $H$. There is an edge between a variable node and a check node if and only if there is a 1 in the corresponding entry of $H$. This graph representation was introduced by Tanner in 1981 as a means to recursively construct codes with low-complexity encoders and decoders [3], and is now called a *Tanner graph* of the code. A vector in $\mathbb{F}_2^n$ is a codeword if and only if when the coordinates are input to the correct variable nodes of the graph, the sum of the neighbors of each check node is $0 \in \mathbb{F}_2$ [3]. For codes over $\mathbb{F}_q$, the edges of the Tanner graph are labeled with the corresponding nonzero matrix entry from $\mathbb{F}_q$, and a vector in $\mathbb{F}_q^n$ is a codeword if and only if when the coordinates are input to the correct variable nodes of the graph, the *weighted* sum of the neighbors of each check node is $0 \in \mathbb{F}_q$. Notice that in either case, this is equivalent to the vector belonging to the kernel of $H$. A small example illustrating this relationship is shown in Figure 2.3.

## 2.1 Low-density parity-check codes

*Low-density parity-check (LDPC)* codes are a class of highly competitive linear codes introduced by Gallager in 1962 whose parity-check matrices have a low density of 1's; equivalently, an LDPC code's Tanner graph representation is sparse with respect to the number of edges [2]. Gallager's work focused on $(j, k)$-regular LDPC codes, in which the number

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

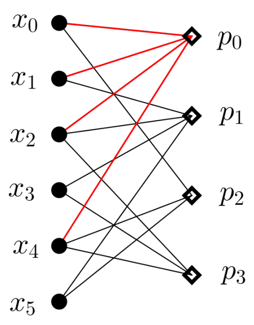


**Figure 2.3:** A parity-check matrix, $H$, of a code $C$ and its corresponding Tanner graph representation. Variable nodes are denoted by $\bullet$, and check nodes are denoted by $\diamond$. The highlighted 1's in $H$ correspond to the highlighted edges of the graph.

of nonzero entries in each column or row of the parity-check matrix is held constant at $j$ or $k$, respectively. He showed that, with high probability, randomly chosen $(j,k)$-regular LDPC codes approach the Gilbert-Varshamov bound (see e.g. [18, p.86]) for large $j$ and $k$, meaning these ensembles are asymptotically good in the sense that their minimum distance grows with block length for a fixed nonzero rate. However, Gallager also showed that $(j,k)$-regular codes fall short of channel capacity. *Irregular* LDPC code ensembles, which have varying column and row weights, were introduced as a modification of Gallager's work [19]. The row and column weights of these codes follow a choice of *degree distribution*. Degree distributions were optimized in [5], resulting in the first known construction of capacity-achieving codes since Shannon originally proved their existence in 1948 [1]. However, this result relies on very large block length. An important open question is how we may design explicit finite-length, capacity-achieving, asymptotically good codes.

Tanner used his graph representation to expand the notion of an LDPC code to *generalized* low-density parity-check (GLDPC) codes, which assign smaller "subcodes" to each of the constraint nodes, rather than simple parity checks. It is important to note that the term subcode, while standard in the literature, is something of a misnomer: these are not literally subcodes of the overall code. Instead, each subcode has block length equal to the degree of its constraint node, and the constraint node is considered satisfied if, for some fixed ordering of the incident graph edges, the adjacent variable nodes form a codeword of the subcode. Tanner showed that lower bounds for the rate and minimum distance of the overall code improve as the same parameters improve in the chosen subcode:

**Theorem 2.1.1.** *[3] Let $R$ be the rate of a linear code constructed from an $(m, n)$-regular bipartite graph. If a fixed linear subcode with parameters $(n, k)$ and rate $r = k/n$ is associated with each of the constraint nodes, then*

$$R \geq 1 - (1 - r)m.$$

Moreover, he gave a bound on the minimum distance of the overall code which depends on the minimum distance of the subcode as well as the girth of the Tanner graph:

**Theorem 2.1.2** (The Tree Bound on Minimum Distance)**.** *[3] Let $d$ be the minimum Hamming distance of the single subcode acting at all constraint nodes, $D$ the minimum Hamming distance of the resulting code, $m$ the degree of the variable nodes, and $g$ the girth of the Tanner graph. Then*

$$D \geq d\left(\frac{[(d-1)(m-1)]^{(g-2)/4} - 1}{(d-1)(m-1) - 1}\right) + \frac{d}{m}[(d-1)(m-1)]^{(g-2)/4} \quad \textit{for } g/2 \textit{ odd, and}$$

$$D \geq d\left(\frac{[(d-1)(m-1)]^{g/4} - 1}{(d-1)(m-1) - 1}\right) \quad \textit{for } g/2 \textit{ even.}$$

Recall that the Tanner graph of an LDPC code is not unique, but depends on our choice of parity-check matrix. Theorem 2.1.2 suggests that choosing a representation with high girth will give us a tighter lower bound on the minimum distance of the code, and was the first result to indicate that decoding is improved with large girth. Since then, research has shown that girth is not the only graph parameter that will affect the performance of the decoder. In particular, how cycles are arranged in the graph has a heavy influence, and other graph properties affect decoding as well. It is then apparent that the choice of parity-check matrix, and thus Tanner graph, for a code will have an effect on the performance of iterative decoding algorithms, which we will discuss further in Section 2.3.

The capacity-approaching performance of turbo codes and associated graph-based decoders introduced by Berrou, Glavieux, and Thitimajshima in 1993 [4] sparked renewed interest in the long-dormant[1] area of LDPC codes and graph-based codes generally, giving rise to the work of Sipser and Spielman [21]. In 1996, Sipser and Spielman presented a family of explicit asymptotically good codes using *expander graphs*, which may be characterized by a large spectral gap in the adjacency matrix of a regular graph, or by an expansion factor guaranteeing a lower bound on the size of the neighborhood of subsets of variable nodes. Their results showed that codes from graphs with good expansion have improved error correction capabilities [21].

### 2.1.1 Protograph LDPC codes

One common method of constructing LDPC codes involves designing a small *protograph* block code and then algebraically *lifting* the corresponding Tanner graph to arrive at a code with longer block length, while preserving some desirable Tanner graph properties [22,23]. A *graph lift* is a finite topological covering space of a graph; we may view a *degree $J$ lift* of a protograph as a graph consisting of clouds of $J$ copies of each of the vertices of the protograph, with $J$ edges forming a matching between each pair of clouds which arise from adjacent vertices in the protograph. To specify the placement of these edges, we may assign permutations from $S_J$, the symmetric group on $J$ elements, to the edges of the protograph.

[1]with the exceptions of Tanner's work and the work of Zyablov and Pinsker in 1975 [20].

More formally,

**Definition 2.1.3.** *Let $G$ be a graph with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subseteq V \times V$. A* degree $J$ lift *of $G$ is a graph $\hat{G}$ with vertex set $\hat{V} = \{v_{1_1}, \ldots, v_{1_J}, \ldots, v_{n_1}, \ldots, v_{n_J}\}$ of size $nJ$ and for each $e \in E$, if $e = v_iv_j$ in $G$, then there are $J$ edges from $\{v_{i_1}, \ldots, v_{i_J}\}$ to $\{v_{j_1}, \ldots, v_{j_J}\}$ in $\hat{G}$ which form a vertex matching. To algebraically obtain a specific lift $\hat{G}$, permutations may be assigned to each of the edges in $G$ so that if $e = v_iv_j$ is assigned a permutation $\tau \in S_J$, the corresponding edges in $\hat{G}$ are $v_{i_k}v_{j_{\tau(k)}}$ for $1 \leq k \leq J$. The edge $e$ is considered as directed for the purpose of lifting.*

Recall that cycle notation for an element in $S_J$ is an ordering of the elements of $[J] = \{1, 2, \ldots, J\}$ in a list partitioned by parentheses which is read as follows: each element is mapped to the element on its right, and an element before a closing parenthesis is mapped to the first element within that parenthesis. Each set of parentheses denotes a *cycle*. The *cycle structure* of a permutation $\pi \in S_J$ is a vector $(c_1, \ldots, c_J)$ where, for $i \in [J]$, $c_i$ denotes the number of $i$-cycles in the cycle notation of $\pi$. We will also equate a permutation in $S_J$ with its corresponding $J \times J$ permutation matrix, where entry $(i, j)$ is equal to 1 if $j \mapsto i$ in the permutation, and 0 otherwise.

**Example 2.1.4.** *The permutation $\pi = (1\ 2)(3\ 4\ 5)$ is the permutation which maps 1 to 2, 2 to 1, 3 to 4, 4 to 5, and 5 to 3. The cycle structure of $\pi$ is $(0, 1, 1, 0, 0)$ since $\pi$ has one 2-cycle and one 3-cycle. The permutation $\pi$ may be represented by the permutation matrix*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

We may also view the process of lifting a protograph as replacing the nonzero entries of its adjacency matrix with $J \times J$ permutation matrices, and the zero entries with the $J \times J$ all-zeros matrix. In practice, permutation edge assignments are often made randomly. However, the following theorem demonstrates how graph lifting may be leveraged to improve the girth of a graph, while retaining properties such as degree distribution. Let the *net permutation* of a path in a protograph be the product of its (oriented) permutation edge labels.

**Theorem 2.1.5.** *[24] If $C$ is a cycle of length $k$ with net permutation $\pi$ in the graph $G$, and $\hat{G}$ is a degree $J$ lift of $G$, then the edges corresponding to $C$ in $\hat{G}$ will form $c_1 + \cdots + c_J$ components with exactly $c_i$ cycles of length $ki$ each, where $(c_1, \ldots, c_J)$ is the cycle structure of $\pi$.*

Permutation edge assignments to the protograph, and the corresponding graph lift properties, are studied further in [24].

## 2.2 Iterative decoders

Due to their sparsity, LDPC codes are amenable to low-complexity graph-based message-passing decoding algorithms, which drastically reduce decoding complexity from that of ML decoding for practical codes of large block length. Each such algorithm processes information locally in some way at variable nodes, and then sends a corresponding message to adjacent constraint nodes. The constraint nodes, in turn, process their received information, and send messages back to variable nodes. In each case, a node does not use information received along a given edge to form the message to be sent back along that edge. These processes occur in discrete time, and a single iteration is considered a complete round of variable-to-check and check-to-variable messages.

LDPC codes were first introduced alongside what are known as the Gallager A/B decoding algorithms [2]. Both algorithms operate for transmission over the BSC. Variable to check ($\mu_{v\to c}$) and check to variable ($\mu_{c\to v}$) messages for the Gallager A algorithm are calculated as follows:

$$\mu_{v\to c}(r, m_1, \dots, m_{d(v)-1}) = \begin{cases} r+1 \text{ if } m_1 = \cdots = m_{d(v)-1} = r+1 \\ r \text{ else,} \end{cases} \tag{2.1}$$

where $r$ is the received value at variable node $v$, $d(v)$ is the degree of $v$, and $m_1, \dots, m_{d(v)-1}$ are the most recent messages received from the check nodes which are not those to which the current message is being calculated. Notice, then, that the $m_i$'s will be a different set of messages depending on the check node to which the current message is being sent. In the other direction,

$$\mu_{c\to v}(m_1, \dots, m_{d(c)-1}) = \sum_{i=1}^{d(c)-1} m_i \,, \tag{2.2}$$

where here $m_1, \dots, m_{d(c)-1}$ are messages received from variable nodes, and will differ depending on to which variable node the current message is being sent. Note that calculations are performed in $\mathbb{F}_2$. Gallager B relaxes the unanimity condition of $\mu_{v\to c}$. Throughout the examples in this work which use Gallager A decoding, we let the final decoder output at a variable node $v$ be given by the received value at $v$, unless incoming messages $\mu_{c\to v}$ for $c$ adjacent to $v$ unanimously disagree with this received value.

A variety of graph-based decoding algorithms have been introduced since Gallager's original work, including other hard message-passing algorithms [3, 21], and implementations of belief propagation (BP) decoding. The sum-product algorithm (SPA), a method of BP decoding [25, 26], differs from Gallager A/B in that it is not a bit-flipping algorithm. Instead, it sends and processes soft messages that represent the probability of a node assuming a certain value.

## 2.3 Trapping, stopping, and absorbing sets

Unfortunately, despite their advantage in terms of complexity, iterative message-passing decoders can get caught in error patterns, even when there are relatively few errors in received words – that is, even when transmission is at high signal-to-noise ratios (SNR). This behavior is reflected in so-called *error floors* of bit error rate (BER) curves. Such failures have been shown to be caused by substructures in a code's graph representation; among these problematic structures are *stopping sets*, *absorbing sets*, *pseudocodewords*, and *trapping sets* [27–30]. The presence of trapping sets is highly dependent on the choice of graph representation, channel, and decoder. Absorbing sets and stopping sets are combinatorially-defined, and are special cases of the more general trapping sets; nevertheless, it has been shown that their presence affects decoder performance over many channels and under a variety of decoders.

Let $C$ be a binary code of length $n$ with associated Tanner graph $G$, to be decoded with some chosen hard- or soft-decision decoder. Following the notation and definitions in [31], suppose that the codeword $\mathbf{x}$ is transmitted, and $\mathbf{y}$ is received. Let $\mathbf{y}^\ell$ be the output after $\ell$ iterations of the decoder are run on $G$. A node $y_i$, with $1 \leq i \leq n$, is said to be *eventually correct* if there exists $L \in \mathbb{Z}_{\geq 0}$ such that $y_i^\ell = x_i$ for all $\ell \geq L$.

**Definition 2.3.1.** *Let $\mathcal{T}(\mathbf{y})$ denote the set of variable nodes that are not eventually correct for a received word $\mathbf{y}$, and let $G[\mathcal{T}]$ denote the subgraph induced by $\mathcal{T}(\mathbf{y})$ and its neighbors in the graph $G$. If $G[\mathcal{T}]$ has $a$ variable nodes and $b$ odd-degree check nodes, $\mathcal{T}(\mathbf{y})$ is said to be an* $(a, b)$-trapping set. *In this case, the set of variable nodes in error in the received word $\mathbf{y}$ is called an* inducing set *for $\mathcal{T}(\mathbf{y})$.*

Examples of possible trapping set structures are given in Figure 2.4.

**Definition 2.3.2.** *The* critical number *of a trapping set $\mathcal{T}$, denoted $m(\mathcal{T})$, is the minimum number of variable nodes in an inducing set of $\mathcal{T}$.*

It is important to note that the critical number may arise from an inducing set not fully contained in $\mathcal{T}$.

Due to their complexity, trapping sets are typically analyzed under hard-decision decoders, such as Gallager A/B [2, 32], although interesting work has also been done on trapping set analysis for soft-decision decoders on the AWGN channel [33–35]. Simulation results suggest that trapping sets with respect to hard-decision decoders also affect the error floor performance of soft-decision decoders [31, 36]. Several specific instances of trapping sets under certain hard-decision decoders are defined next.

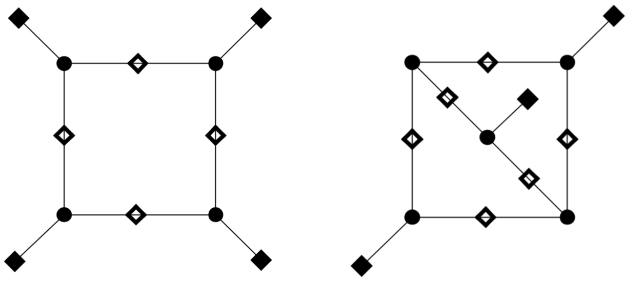


**Figure 2.4:** Examples of a $(4, 4)$-trapping set on the left, and a $(5, 3)$-trapping set on the right. Both examples are from [36]. Variable nodes are denoted by $\bullet$, even degree check nodes (in the induced subgraph shown) are denoted by $\diamond$, and odd degree check nodes by $\blacklozenge$.

**Definition 2.3.3.** *[27,37,38] A* stopping set *in the Tanner graph representing a generalized LDPC code is a subset $S$ of variable nodes such that each neighbor of $S$ has at least $d_{\min}(C)$ neighbors in $S$, where $C$ is the subcode represented by each constraint node.*

Because the minimum distance of a simple parity-check code is equal to 2, this definition reduces in the LDPC code case to a subset $S$ of variable nodes such that each neighbor of $S$ is connected to $S$ at least twice. A stopping set is a trapping set when information is sent across an erasure channel and decoded using a *peeling decoder*, in which erasures are iteratively peeled away. Intuitively, when the values of variable nodes in a stopping set are erased, the decoder will be unable to determine their correct values due to a lack of information, and the decoding process will halt.

Once a node is corrected with this decoder, it remains statically correct, so any inducing set of a stopping set must contain that stopping set. This implies that the critical number of a stopping set of size $a$ is equal to $a$. A stopping set is said to be *minimum* in a Tanner graph if there is no stopping set of smaller size in the graph. The size of a minimum stopping set is denoted $s_{\min}(H)$, where $H$ is the corresponding parity-check matrix; it is desirable to design codes for which $s_{\min}(H)$ is large, though it should be noted that $s_{\min}(H)$ is bounded above by the minimum distance of the code, since the support of any codeword forms a stopping set.

For other channels, the following structures were introduced as a way to analyze error floor behavior.

**Definition 2.3.4.** *[39] An* $(a, b)$-absorbing set *is a subset $D$ of variable nodes in a code's Tanner graph such that $|D| = a$, $|O(D)| = b$, and each variable node in $D$ has strictly fewer neighbors in $O(D)$ than in $N(D) \setminus O(D)$, where $N(D)$ is the set of check nodes adjacent to variable nodes in $D$ and $O(D)$ is the subset of check nodes of odd degree in the subgraph induced on $D \cup N(D)$. An* $(a, b)$-fully absorbing set *is an $(a, b)$-absorbing set with the additional property that each variable node not in $D$ has strictly fewer neighbors in $O(D)$ than in $F \setminus O(D)$, where $F$ is the set of all check nodes.*

Absorbing sets arise as trapping sets under bit-flipping decoders: in particular, fully absorbing sets are trapping sets under the Simple Parallel Decoding Algorithm of [21]. An *elementary absorbing set* is an absorbing set such that all adjacent check nodes have degree 1 or 2 in the induced subgraph. The trapping set examples in Figure 2.4 are also elementary absorbing sets. Though absorbing sets are not truly trapping sets for other decoders, their presence has been shown to have a significant effect on error floor performance [40, 41].

# Chapter 3

# Spatially-coupled LDPC Codes

Spatially-coupled LDPC (SC-LDPC) codes form an important class of LDPC codes. First introduced by Felström and Zigangirov in [6], SC-LDPC ensembles have been shown to be asymptotically good, and capacity-approaching under low-complexity decoding [7]. Furthermore, there exist ensembles which are good for an entire class of channels, rather than being channel-dependent [8]. One method of constructing SC-LDPC codes is by applying an algebraic graph lift to a *spatially-coupled protograph* (SC-protograph). To form the SC-protograph, a base Tanner graph is copied and *coupled*. There are many ways to couple the edges from one copy of the base graph to the other copies; this process of coupling is generally termed *edge-spreading*. The repetition of structure imposed in this process allows the resulting SC-LDPC code to be a terminated LDPC convolutional code if the permutations applied to lift the resulting SC-protograph are cyclic permutations [42, 43]. Furthermore, this repeated structure allows for the implementation of a *windowed decoder*, a decoder that operates on only a small portion of the code's graph at a time [44, 45]. In general, terminated SC-LDPC codes are desirable for practical applications due to their performance and structure [39, 42, 46].

Despite their many advantages, SC-LDPC codes may still exhibit error floors in their BER curves. However, the edge-spreading step of the construction process of SC-LDPC codes may be leveraged to alleviate the influence of harmful trapping sets in the code's

Tanner graph representation. A significant amount of work has been done to this end [12, 13, 39, 46–49]. In this chapter, we present several known methods of constructing SC-LDPC codes, and show how these methods may be implemented with trapping set removal in mind. We then establish an algebraic construction approach for SC-LDPC codes that unifies disparate work under a single umbrella. Finally, we show how this approach can prove highly advantageous in the quest to eliminate harmful substructures in a code's graph representation. Work in subsection 3.2 appears in [12], and work in subsections 3.3 and 3.4 appears in [13][1].

## 3.1 Construction & decoding of SC-LDPC codes

To construct an SC-protograph, $L$ copies of a base graph, such as the one shown in Figure 3.1, are coupled. The coupling process may be thought of as first replicating the base graph at positions $0, \ldots, L-1$, and then "edge-spreading" the edges to connect the variables nodes at position $i$ to check nodes in positions $i, \ldots, i+m$ so that the degrees of the variable nodes in the base graph are preserved. The number of copies of the base graph, $L$, is referred to as the *coupling length*, and the number of future copies of the base graph that an edge may spread to, $m$, is called the *memory* or *coupling width*. The way in which edges are spread from the variable nodes in Position 0 will be applied at all future positions $1, \ldots, L-1$. In the case of a *terminated* SC-protograph, terminating check nodes are introduced at the end of the SC-protograph as necessary to *terminate* the SC-protograph. An example of an SC-protograph obtained by coupling the base graph in Figure 3.1 is given in Figure 3.2. Allowing edges to instead loop back around to the first few positions of check nodes results in a *tailbiting* SC-protograph, in which both variable and check node degrees are preserved in all positions.

[1]© 2016, 2017 IEEE

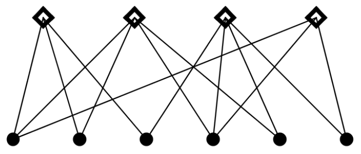


**Figure 3.1:** Base Tanner graph to be coupled to form an SC-protograph. Variable nodes are denoted by $\bullet$, and check nodes are denoted by $\diamond$. © 2016, 2017 IEEE

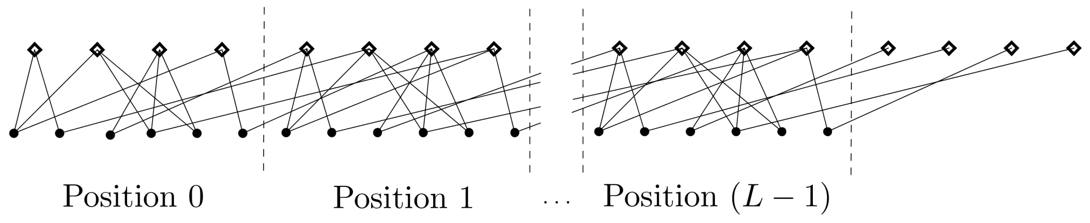


**Figure 3.2:** Terminated SC-protograph resulting from randomly edge-spreading $L$ copies of the Tanner graph in Figure 3.1 with memory $m = 1$, and applying the same map at each position. © 2016, 2017 IEEE

This edge-spreading process may also be viewed in terms of the parity-check matrix, $H$, of the base graph. Edge-spreading is equivalent to splitting $H$ into a sum of $m + 1$ matrices of the same dimension, so that $H = H_0 + H_1 + \cdots + H_m$, and then arranging them into $L$ block columns as in Matrix (3.1) to form the parity-check matrix of a terminated SC-protograph. The tailbiting code corresponding to this terminated code has parity-check matrix as in Matrix (3.2), so that every check node has degree equal to its corresponding vertex in the base graph.

$$\begin{pmatrix} H_0 & & & \\ H_1 & H_0 & & \\ \vdots & \vdots & \ddots & \\ H_m & & & \\ & H_m & & \\ & & & H_0 \\ & & \ddots & \vdots \\ & & & H_m \end{pmatrix} \tag{3.1}$$

$$\begin{pmatrix} H_0 & & & H_m \cdots & H_1 \\ H_1 & \ddots & & \ddots & \vdots \\ \vdots & & & & H_m \\ H_m & & & \ddots & \\ & \ddots & & & \\ & & H_m & \cdots & H_0 \end{pmatrix} \tag{3.2}$$

Edge-spreading may be done in a variety of ways. Two common methods are [8]: (i) For each variable node $v$ in Position 0, if $v$ has $j$ neighbors $c_1, \dots, c_j$ in the base graph, randomly choose for each $\ell = 1, \dots, j$, a copy of $c_\ell$ from the Positions $0, \dots, m$, and (ii) if every variable node in Position 0 has $j$ neighbors in the base graph, randomly choose $j$ of the Positions $0, \dots, m$ to spread edges to, then, for each of the $j$ neighbors $c_1, \dots, c_j$ of a variable node $v$, randomly choose a check neighbor from the copies of $c_\ell$ ($\ell = 1, \dots, j$) such that $v$ has exactly one neighbor in each of the chosen $j$ positions, and exactly one of each check node neighbor type. Note that method (ii) is a particular instance of (i).

Finally, regardless of the edge-spreading technique, a *terminal lift* may then be applied to the SC-protograph, yielding the SC-LDPC code.

### 3.1.1 Array-based SC-LDPC codes

Array-based LDPC codes are a class of structured LDPC codes which have been used to construct SC-LDPC codes [39, 50]. An array-based LDPC block code with parameters $\gamma, p \in \mathbb{Z}_{>0}$, where $p$ is prime, is defined by the $p\gamma \times p^2$ parity-check matrix

$$H(\gamma, p) = \begin{pmatrix} I & I & I & \cdots & I \\ I & \sigma & \sigma^2 & \cdots & \sigma^{p-1} \\ I & \sigma^2 & \sigma^4 & \cdots & \sigma^{(p-1)\cdot 2} \\ \vdots & \vdots & \vdots & & \vdots \\ I & \sigma^{\gamma-1} & \sigma^{2(\gamma-1)} & \cdots & \sigma^{(p-1)(\gamma-1)} \end{pmatrix},$$

where $I$ denotes the $p \times p$ identity matrix, and $\sigma$ is the $p \times p$ permutation matrix given by left-shifting the columns of the $p \times p$ identity matrix by one position, as shown below.

$$\sigma = \begin{pmatrix} 0 & & & 0 & 0 & 1 \\ 1 & 0 & & & \ddots & 0 \\ 0 & 1 & \ddots & & & 0 \\ & \ddots & \ddots & \ddots & & \\ & & & 0 & 1 & 0 \end{pmatrix}.$$

We may use the Tanner graph of an array-based LDPC block code as the base graph when constructing an SC-LDPC code. One way of defining the edge-spreading in this case is via a *cutting vector*, denoted by $\xi = [\xi_0, \xi_1, \ldots, \xi_{\gamma-1}]$ where $0 \leq \xi_0 < \xi_1 < \ldots < \xi_{\gamma-1} \leq p$, which defines how the parity-check matrix of the base graph will be split into a sum of two matrices [6,39]. In particular, the first $\xi_i$ block columns of the $(i+1)$st block row of $H(\gamma, p)$ are copied into the corresponding positions in a $p\gamma \times p^2$ matrix $H_0$, and the remaining $p - \xi_i$ are copied into a matrix $H_1$. The remaining entries of each matrix are set to zero. Note that this technique automatically gives a memory of $m = 1$: indeed, $H_0 + H_1 = H(\gamma, p)$.

For example, if $p = \gamma = 3$, and $\xi = [0, 2, 3]$, then

$$H(3,3) = \begin{pmatrix} I & I & I \\ I & \sigma & \sigma^2 \\ I & \sigma^2 & \sigma \end{pmatrix},$$

$$H_0 = \begin{pmatrix} 0 & 0 & 0 \\ I & \sigma & 0 \\ I & \sigma^2 & \sigma \end{pmatrix}, \quad \text{and} \quad H_1 = \begin{pmatrix} I & I & I \\ 0 & 0 & \sigma^2 \\ 0 & 0 & 0 \end{pmatrix}.$$

The parity-check matrix of the corresponding SC-protograph with coupling length $L$ is given by $H(\gamma, p, L, \xi)$, shown in Equation 3.3, which has $L$ block columns. In this construction method, a terminal lift is not typically applied.

$$H(\gamma, p, L, \xi) = \begin{pmatrix} H_0 & & & & \\ H_1 & H_0 & & & \\ & H_1 & \ddots & & \\ & & \ddots & H_0 & \\ & & & H_1 & H_0 \\ & & & & H_1 \end{pmatrix}. \tag{3.3}$$

The cutting vector approach has been expanded in [47] and [48] to allow for higher memory and more freedom in the edge-spreading structure, though blocks of edges remain spread as single units.

The high structure of these array-based SC-LDPC (AB-SC-LDPC) codes has allowed for the analysis of multiple parameters, including minimum distance [51], and the presence of absorbing sets [13, 39, 46–48].

### 3.1.2 Windowed decoding

While SC-LDPC codes may be decoded via belief propagation performed on the code's entire graph, we may instead take advantage of their structure in order to decode smaller portions of the code in parallel, using a *windowed decoder*. This is ideal for applications such as streaming, where information packets should be received and decoded in order, rather than being recovered all at once. Note that decoding is performed on the terminal lift of the SC-protograph, corresponding to the SC-LDPC code, if such a lift is performed. However, as discussed in Chapter 2, lifting preserves much of the structure of the SC-protograph. In particular, each node is replaced with a "cloud" of nodes in the lift, but variable and constraint nodes may still be partitioned into positions. From the way in which the SC-protograph is constructed, a single position of variable nodes may be adjacent to at most $m + 1$ positions of constraint nodes, where $m$ is the memory of the code. Therefore, variable nodes that are far enough apart in the SC-protograph, and thus in the resulting SC-LDPC code, will not be involved in the same check equations. Exploiting this idea, Felström and Zigangirov proposed a pipeline decoder in [6], and windowed decoders for LDPC convolutional codes were analyzed in numerous subsequent papers [44, 45, 52–54]. Papaleo et al. and Iyengar et al. [44, 45] showed that the structure of SC-LDPC codes allows for the use of windowed decoders.

A windowed decoder runs on a small *window* of nodes, sliding from left to right along the received bits as it decodes. The window is defined to be $W$ consecutive positions of constraint nodes and all of their adjacent variable nodes [52]. We will assume, as in [44] and [52], that the *window length* $W$ satisfies $m + 1 \leq W \leq m + L$, where $L$ is the code's coupling length. While decoding on a given portion of the graph, the decoder runs until a target error probability in the first position of variable nodes in the window, called the *targeted symbols*, has been reached, or until a fixed number of iterations have been completed [45, 52]. Then, the window slides over by one position of constraint nodes, and repeats this process. We call the set of edges contained in a window at a given decoding instant the *window configuration* of the windowed decoder. Note that, in a terminated SC-LDPC code, the end window configurations will differ from the typical window configuration. If the code is terminated with coupling length $L$, then the process will terminate after $L$ windows have been decoded.

We may also view the windowed decoder in terms of the parity-check matrix of the SC-LDPC code. The window will cover $W$ block rows, and $W + m$ block columns. Figure 3.3 shows a decoding window in the context of the parity-check matrix of an SC-LDPC code.

As mentioned above, windowed decoding allows for multiple decoders to run in parallel. Furthermore, windowed decoding may be implemented for decreased complexity and decoding latency with low performance degradation [7]. The structure of SC-LDPC codes make them amenable to decoding with a windowed decoder, and thus to applications for which a windowed decoder is well-suited.

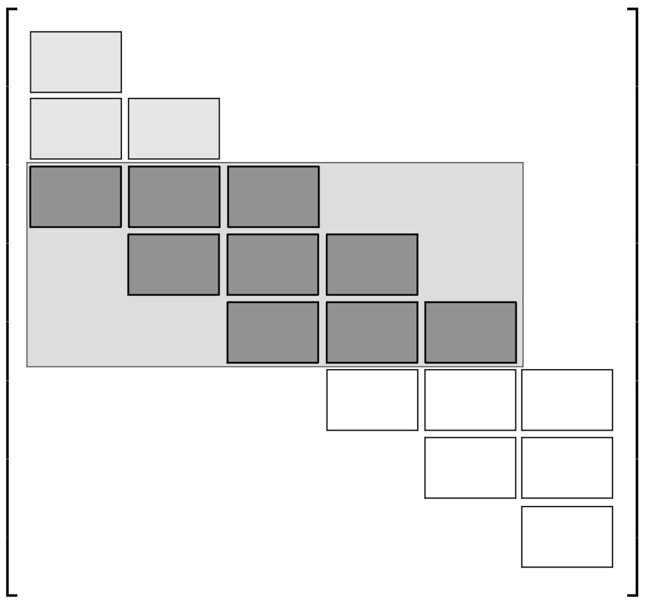


**Figure 3.3:** A window of the windowed decoder with $W = 3$, operating on the parity-check matrix of an SC-LDPC code with $m = 2$, and $L = 6$. The first block column consists of some already-decoded code symbols in Position 0 of variable nodes, while the blocks on the lower right are variable nodes not contained in the window, which have not yet been processed. On its next iteration, the window will slide down and right by one block, and will process an identical window configuration.

## 3.2 Trapping set removal algorithm

In this section, we present an algorithm for general edge-spreading that is designed to eliminate certain trapping set subgraphs in the resulting SC-protograph. We assume our base Tanner graph to be the Tanner graph of a block code, with no multi-edges and a reasonable block length. Our algorithm assumes that a *priority list* of trapping sets to remove is provided at the start. Methods such as that presented in [55] may be used to identify trapping sets in the base graph, which is a graph of reasonably small block length. The priority list, which we will call $P$, contains trapping sets from the base graph listed individually; that is, if two sets of variable nodes induce trapping set subgraphs that are isomorphic (and therefore, also of the same type), each will be listed in $P$ as separate entries. In Section 3.2.1, we describe how the ordering of this list may be obtained using the trapping set poset structure; the list may also be obtained using methods from the trapping set ontology [36]. We adopt some terminology from the algorithm in [56] that is used to remove trapping sets in protograph LDPC codes.

Let $P = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_k\}$ be a list of trapping sets to avoid among variable nodes within each position of the SC-protograph, in order of priority. Let $m$ be an upper bound on the desired memory of our SC-protograph, and let $E_{\mathcal{T}_i}$ denote the set of edges in $G[\mathcal{T}_i]$ and $E_{i,c}$ the set of edges in $G[\mathcal{T}_i]$ incident to check node $c$. Let $e_c$ denote the check node incident to an edge $e$ in the Tanner graph, and recall that $d_H(v)$ denotes the degree of vertex $v$ in the graph $H$. We present Algorithm 1, below, to remove trapping sets from the priority via edge-spreading.

The central idea of Algorithm 1 is to break apart the most harmful trapping set subgraphs appearing in the base graph by spreading their edges to later positions in the SC-protograph. By "freezing" edges which, within a trapping set subgraph on the priority list, share a check node with a spread edge, we ensure that we do not simply reconstruct the trapping set subgraph with check nodes in later positions. Because trapping set subgraphs contain cycles [55], choosing to spread an edge which is contained in a cycle, when possible, will have a greater chance of eliminating more trapping set subgraphs at once, and improving the girth of the SC-protograph. To avoid low-degree check nodes (e.g. degree 0 and 1) at either end of the SC-protograph, it may be necessary to adjust connections.

**Lemma 3.2.1.** *The trapping sets in the set $S$ at the termination of Algorithm 1 do not occur*

**Algorithm 1** Trapping Set Removal Algorithm

```
Input: $L$ copies of the base Tanner graph $G$; $P = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \dots, \mathcal{T}_k\}$
 1: $SpreadEdges \leftarrow \emptyset$
 2: $FrozenEdges \leftarrow \emptyset$
 3: $S \leftarrow \emptyset$
 4: for $i = 1$ to $k$ do
 5:     if $E_{\mathcal{T}_i} \cap SpreadEdges = \emptyset$ then
 6:         if $\exists e \in E_{\mathcal{T}_i} \setminus FrozenEdges$ such that $d_{G[\mathcal{T}_i]}(e_c) > 1$ then
 7:             Spread $e$ randomly to $e_c$ in Position $1, 2, \dots,$ or $m$.
 8:             $SpreadEdges \leftarrow SpreadEdges \cup e$
 9:             $FrozenEdges \leftarrow FrozenEdges \cup E_{i,c}$
10:             $S \leftarrow S \cup \mathcal{T}_i$
11:     if $E_{\mathcal{T}_i} \cap SpreadEdges \neq \emptyset$ then
12:         Choose $c \in G[\mathcal{T}_i]$ incident to an edge in $E_{\mathcal{T}_i} \cap SpreadEdges$
13:         $FrozenEdges \leftarrow FrozenEdges \cup E_{i,c}$
14:         if $E_c \setminus SpreadEdges \neq \emptyset$ then
15:             $S \leftarrow S \cup \mathcal{T}_i$
Output: $S$
16: Randomly spread edges in Position 0 not in $FrozenEdges$ to a copy of its incident
    check node in Position $0, 1, \dots$ or $m$.
17: Repeatedly apply this edge-spreading at Positions $1, 2, \dots, L-1$, adding terminating
    check nodes as necessary.
Output: Terminated SC-protograph of coupling length $L$ and memory at most $m$.
```

*within a single position in the resulting SC-protograph. That is, if $\mathcal{T} \in S$, then copies of the variable nodes in $\mathcal{T}$ do not induce a subgraph isomorphic to $G[\mathcal{T}]$ in any single position of the SC-protograph.*

*Proof.* Suppose that, after running Algorithm 1, the trapping set $\mathcal{T} = \{v_1, \dots, v_a\} \in S$. Let $v_{j,i}$ denote the copy in Position $j$ of the SC-protograph output by Algorithm 1 of variable node $v_i$ in the base graph. Because $\mathcal{T} \in S$, there exists some check node $c$ in $G[\mathcal{T}]$ of degree at least 2 which had at least one incident edge spread, and at least one incident edge frozen in Position 0. Thus, for all $j \in \{0, \dots, L-1\}$, the subgraph of the SC-protograph induced by the variable nodes $\{v_{j,1}, \dots, v_{j,a}\}$ contains at least two copies of check node $c$. Furthermore, this subgraph contains at least one copy of all other check nodes in $G[\mathcal{T}]$. In particular, the subgraph induced by $\{v_{j,1}, \ldots, v_{j,a}\}$ has strictly more check nodes than $G[\mathcal{T}]$, so the subgraphs cannot be isomorphic. $\square$

**Remark 3.2.2.** *Depending on the number of trapping sets in $P$ it is possible that not all trapping sets in the priority list will end up in $S$; however, by ranking them in order of harmfulness (see Section 3.2.1), the most significant trapping sets will.*

### 3.2.1 Ranking trapping sets

Recall that Algorithm 1 relies on a priority list that ranks the trapping sets to be removed in the base Tanner graph, and cannot be iterated indefinitely, as eventually the set $FrozenEdges$ will dominate the base graph. It is therefore important to construct the priority list judiciously. A ranking of the relative harmfulness of these trapping sets should take into account critical number and the number of small inducing sets, as well as the topological relations between trapping sets [31, 36]. Thus, one option is to order all of the trapping sets of the base graph $G$ by critical number. We note that, when possible, ties may be broken by the number of inducing sets of size equal to the critical number. That is, if $\mathcal{T}_1$ and $\mathcal{T}_2$ both have critical number $m(\mathcal{T})$, but $\mathcal{T}_1$ has more inducing sets of size $m(\mathcal{T})$ than $\mathcal{T}_2$, $\mathcal{T}_1$ may be deemed more harmful than $\mathcal{T}_2$. Constructing the priority list in this way forces Algorithm 1 to prioritize eliminating trapping sets from most harmful to least harmful. Such a ranking method was employed in [31] and [56].

However, a ranking based solely on critical number does not take the topological relations between trapping sets into account. A trapping set $\mathcal{T}_1$ is a *parent* of the trapping set $\mathcal{T}_2$ if $G[\mathcal{T}_1]$ is a subgraph of $G[\mathcal{T}_2]$. In this case, $\mathcal{T}_2$ is a *child* of $\mathcal{T}_1$, and in general $\mathcal{T}_2$ is more harmful than $\mathcal{T}_1$ [36]. Under the parent/child relationship, where $\mathcal{T}_1 \leq \mathcal{T}_2$ if and only if $\mathcal{T}_1$ is a child of $\mathcal{T}_2$, the set of trapping sets in a graph $G$ forms a partially ordered set, or poset. Let $P(G)$ denote this poset. Graphically, parents are subgraphs of their children, so this poset has a Galois correspondence with a subposet of the poset of induced subgraphs of $G$, partially ordered by inclusion.

Note that if a priority list arranged solely by critical number is used in Algorithm 1, then edges of several children of a parent trapping set may be frozen before arriving at that parent in the list. Doing so will freeze more edges early on, inhibiting progress through the list. Since avoiding a parent subgraph avoids its children, more trapping sets may be removed by simply reordering them. We now make this method precise.

Observe that eliminating a trapping set subgraph $G[\mathcal{T}_i]$ in $G$ eliminates the *down-set* generated by $\mathcal{T}_i$,

$$D[\mathcal{T}_i] = \{\mathcal{T}_j \mid \mathcal{T}_j \leq \mathcal{T}_i\},$$

in $P(G)$. Following the notation of [57], we denote the poset of down-sets of $P(G)$ by $J(P(G))$. Note that $J(P(G))$ is a graded lattice, where the *rank* of an element of $J(P(G))$ is given by the size of the corresponding down-set in $P(G)$. If we wish to eliminate a set $S$ of trapping sets in $G$ deemed the most harmful, there is a unique minimal join of the down-sets of the elements of $S$, given by $D[S]$. Eliminating the maximal elements of this join down-set will eliminate the entire down-set, as will eliminating maximal elements of any down-set containing this join. Thus, we can label the elements of $J(P(G))$ according to how many maximal elements they contain (notice this is not necessarily order-preserving). To determine which parents to remove in order to remove a set $S$ of trapping sets from $G$, we will look for the maximal element with minimal label in the up-set of $D[S]$, denoted $U(D[S])$, where $D[S]$ is an element of $J(P(G))$.

Consider the poset of trapping sets in Figure 3.4. Ordering by critical number gives the priority list $\{\mathcal{T}_8, \mathcal{T}_9, \mathcal{T}_{10}, \mathcal{T}_4, \mathcal{T}_5, \mathcal{T}_6, \mathcal{T}_7, \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$. However, if we are especially concerned with eliminating trapping sets with critical number 3 or smaller, it would be more efficient if we simply eliminated $\mathcal{T}_5$ followed by $\mathcal{T}_6$, or, even better, just $\mathcal{T}_2$ (notice that all trapping sets of critical number at most 3 are contained in both $D[\mathcal{T}_5, \mathcal{T}_6]$ and $D[\mathcal{T}_2]$). Similarly, to eliminate trapping sets with critical number at most 4, we could prioritize eliminating $\mathcal{T}_1$ and then $\mathcal{T}_2$ (since all trapping sets of critical number at most 4 are contained in $D[\mathcal{T}_1, \mathcal{T}_2]$).

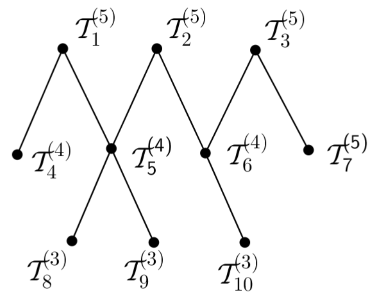


**Figure 3.4:** Example of a poset of trapping sets of a given base Tanner graph, where $\mathcal{T}_i^{(j)}$ denotes the $i$th trapping set, with critical number $j$. © 2016 IEEE

Summarily, priority lists may be formed in many ways including by: (a) critical number, (b) taking the maximal elements in the poset, or (c) taking the maximal elements of a down-set containing the join of the down-sets.

### 3.2.2 Simulation results

Figure 3.5 shows the performance of SC-LDPC codes of block length 32000 bits and code rate 0.45 on the binary symmetric channel with Gallager A decoding. A base protograph with 8 check nodes and 16 variable nodes was coupled using several different edge-spreading algorithms with coupling length $L = 20$. Random protograph 1 and 2 were random edge-spreading methods, whereas Algorithm I protograph 1 and 2 were based on the proposed algorithm with priority lists ordered by eliminating parents first and then remaining trapping sets by critical number, and purely by critical number, respectively. The protographs were each lifted by a lifting factor $J = 100$ to yield SC-LDPC codes. The permutations for lifting the SC-protograph were chosen from the group of shifted identity matrices and were chosen randomly without any optimization. Figure 3.5 shows Algorithm 1 with priority list ordered by eliminating parents first significantly outperforming the other edge-spreading methods with more than two orders of improvement in the error floor region.

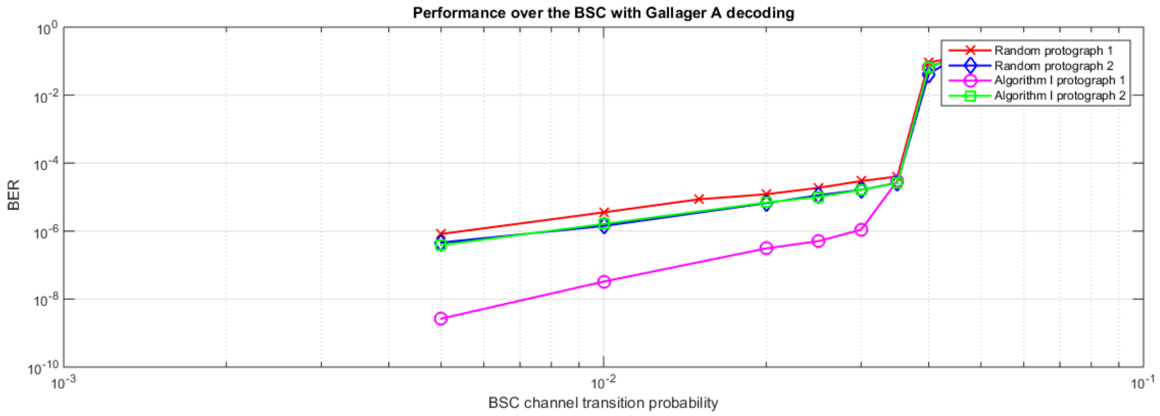


**Figure 3.5:** A comparison of the performance of randomly-constructed protograph SC-LDPC codes to the performance of protograph SC-LDPC codes constructed using Algorithm 1. © 2016 IEEE

### 3.2.3 Trapping sets & the windowed decoder

The relationship between trapping sets in a window of the windowed decoder and in the SC-LDPC graph is given next.

**Lemma 3.2.3.** *If a subset $\mathcal{T}$ of variable nodes within a window has all of its neighbors in the window and is a trapping set with respect to the windowed decoder on the SC-LDPC graph, then $\mathcal{T}$ is a trapping set of the same type with respect to the standard decoder.*

*Proof.* If all the neighbors of $\mathcal{T}$ lie in the window, then the subgraph of the code's Tanner graph induced by $\mathcal{T} \cup N(\mathcal{T})$ in the window is isomorphic to the subgraph induced by $\mathcal{T} \cup N(\mathcal{T})$ in the entire Tanner graph. Thus, an inducing set for $\mathcal{T}$ in the windowed decoder will also be an inducing set for $\mathcal{T}$ with respect to the standard decoder. □

Due to the constraints on window size, Lemma 3.2.3 implies that any trapping set with respect to the windowed decoder contained entirely in the same position as the first position of constraint nodes of the window is also a trapping set in the SC-LDPC graph with standard decoding. In the other direction, if the variable nodes in a trapping set with respect to the standard decoder have all of their neighbors in a window, then they still may form a trapping set with respect to the windowed decoder, depending on the structure of the inducing sets. However, there are variable nodes from previous positions and future positions of the SC-protograph which do not have all of their constraint node neighbors in a given window, yielding the following:

**Lemma 3.2.4.** *If a subset $\mathcal{T}$ of variable nodes within a window does not have all of its neighbors in the window and is a trapping set with respect to the standard decoder on the SC-LDPC graph, then $\mathcal{T}$ may not be a trapping set of the same type with respect to the windowed decoder.*

*Proof.* In this case, the subgraph induced by $\mathcal{T} \cup N(\mathcal{T})$ in the entire graph has been broken in the window, and so $\mathcal{T}$ may no longer yield a trapping set of the same type within the window. □

This allows for the possibility of alleviating the harmful effects of a trapping set simply by utilizing a windowed decoder on the SC-LDPC code.

## 3.3 Algebraic graph lift framework for SC-LDPC code construction

Constructing SC-LDPC codes using Algorithm 1 results in improved error floor performance, as does executing the edge-spreading process using optimized cutting vectors or generalized cutting vectors [39, 46–48]. However, we observe that these approaches are disparate and somewhat ad hoc. In this section, we place SC-LDPC code construction –

that is, the edge-spreading process as well as the terminal lift of the resulting SC-protograph – in terms of algebraic graph lifts. Phrasing the construction process in this way not only unifies previous approaches, but will allow us to remove harmful substructures by leveraging previous results on graph lifting.

To construct a terminated SC-protograph, we may first construct a tailbiting protograph, and then break this graph, copying the constraint nodes at which the graph is broken. We claim that a tailbiting SC-protograph may be viewed as a degree $L$ lift of the base graph – where $L$ denotes the coupling length – by considering the $L$ copies of a node type in the SC-protograph to be the lift of the corresponding node in the base graph. That is, copy $i$ of variable node $v$ from the base graph will appear in position $i$ of the SC-protograph. While a terminated SC-protograph is not, then, strictly a graph lift of the base graph, the set of terminated SC-protographs is in one-to-one correspondence with the set of tailbiting SC-protographs, and so each can be associated with a lift of the base graph.

Recall that once an edge-spreading assignment is made for variable nodes in a single position, that same edge-spreading is repeated at all future positions. This translates to the following:

**Lemma 3.3.1.** *To construct a tailbiting SC-protograph with coupling length $L$ and memory $m$ from a base graph via a graph lift, the possible permutation edge assignments to the base graph from the permutation group $S_L$ are the permutations corresponding to $\tau_L^k$, for $0 \leq k \leq m$, where $\tau_L$ is the $L \times L$ identity matrix left-shifted by one position, and at least one assignment corresponds to $\tau_L^m$. We denote this set of permutations by $A_{L,m}$.*

*Proof.* Since there will be $L$ total positions of nodes in the SC-protograph, it is clear that permutation assignments should come from $S_L$.

If $e_{v \to c}$ denotes a directed variable-to-check edge in the base graph with permutation edge assignment $\pi \in S_L$, the $i$th copy of $v$ in the lift, which we denote by $v_i$, is adjacent to the $\pi(i)^{\text{th}}$ copy of check node $c$ in the lift, denoted $c_{\pi(i)}$. With memory $m$, the gap between $v_i$ and its adjacent check node $c_{\pi(i)}$ must be at most $m$ positions. Furthermore, since the assignments at one position of the SC-protograph are repeated at every position, for any $j \in [L]$, the gap between $j$ and $\pi(j)$ must be the same as that between $i$ and $\pi(i)$ (the position numbers of $v_i$ and $c_{\pi(i)}$).

Thus, any possible permutation should be such that the gap between each value in $[L]$ and its image under the permutation is a fixed constant less than or equal to $m$. Since $\tau_L$ corresponds to the permutation $(1\ 2\ \cdots\ L)$, powers of $\tau_L$ give exactly this set of permutations. At least one permutation assignment should be $\tau_L^m$ to ensure that the memory is, in fact, equal to $m$. □

We should be careful to ensure that the resulting SC-protograph is connected; for instance, assigning the identity permutation to every edge would result in $L$ disconnected copies of the base graph, and so should be avoided.

**Example 3.3.2.** *Suppose $L = 6$ and $m = 3$. Then, $A_{6,3} = \{\tau_6^0, \ldots, \tau_6^3\}$, and all permutation edge assignments to our base graph should be from this set.*

In general,

**Lemma 3.3.3.** *The size of $A_{L,m} \subseteq S_L$ is equal to $m + 1$, and for $0 \le k \le m$, the permutation corresponding to the permutation matrix $\tau_L^k$, has order $L/gcd(k, L)$. Furthermore, the set $A_{L,(L-1)}$ forms a subgroup of $S_L$ for any choice of L.*

*Proof.* The size of the set $A_{L,m}$ and the subgroup structure of $A_{L,(L-1)}$ are straightforward to show. The order may be obtained by considering the cycle structure of $\tau_L^k$. □

Given a fixed memory, we may spread edges by simply assigning allowed permutations to edges in the base graph uniformly at random. This is equivalent to method (i) of edge-spreading, as described in Section 3.1. Method (ii) is more restrictive: it stipulates that for a given variable node, each possible permutation assignment is used at most once on its incident edges.

This framework may be applied to a variety of existing methods for coupling, with additional restrictions on possible permutation assignments in each case. In Section 3.3.2, we will discuss how it may be used to describe the cutting vector, as well as the generalized cutting vectors of [47] and [48].

To arrive at the standard matrix structure of the terminated SC-protograph as given in Matrix (3.1) – and hence the correct ordering of bits in a codeword –, one must rearrange the rows and columns of the matrix resulting from this lift: each $L \times L$ block that has replaced an entry in the base parity-check matrix corresponds to edges of a single type (i.e. between a single type of variable node and check node) in the SC-protograph. To arrive at the ordering of variable and check nodes in Matrix (3.2), we should place the first variable node of type 1 with the first variable node of type 2, etc., and similarly with check nodes. In other words, if the columns (resp., rows) of the parity-check matrix are given by

$$(1,1),\ (1,2),\ \ldots,\ (1,L),\ (2,1),\ \ldots,\ (2,L),\ \ldots,\ (V,1),\ \ldots,\ (V,L),$$

then we should reorder by second entry, then first, as

$$(1,1),\ (2,1),\ \ldots,\ (V,1),\ (1,2),\ \ldots,\ (V,2),\ \ldots,\ (1,L),\ \ldots,\ (V,L).$$

That is, ordering is done primarily by a vertex's index within an $L \times L$ block (ranging from 1 to $L$), and secondarily by the index of that $L \times L$ block (ranging from 1 to $V$, where $V$ is the number of columns of the base matrix); this is the reverse lexicographic order. See Example 3.3.4 for an example of this reordering process.

**Example 3.3.4.** *We give a small example of using algebraic graph lifts to construct an*

*SC-protograph. Let $I_2$ be the $2 \times 2$ identity matrix, and let $\sigma$ be the $2 \times 2$ identity matrix left-shifted by one column. Let the parity-check matrix of our base graph be given by*

$$H = \begin{pmatrix} I_2 & I_2 \\ I_2 & \sigma \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right).$$

*To construct an SC-protograph with coupling length $L = 3$ and memory $m = 1$, we may assign permutations $\tau_3^0$ or $\tau_3^1$ to the edges of the base graph and then algebraically lift according to this assignment. In matrix form, this amounts to replacing the nonzero entries of $H$ with $\tau_3^0$ or $\tau_3^1$, and the zero entries with the $3 \times 3$ all-zeros matrix. That is, for example,*

$$\left(\begin{array}{cc|cc} \tau_3^1 & 0 & \tau_3^0 & 0 \\ 0 & \tau_3^1 & 0 & \tau_3^0 \\ \hline \tau_3^0 & 0 & 0 & \tau_3^1 \\ 0 & \tau_3^0 & \tau_3^1 & 0 \end{array}\right) = \left(\begin{array}{ccc|ccc||ccc|ccc} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline\hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right)$$

*Next, we reorder the rows and columns. Let $(i, j)$ indicate the row or column in block $1 \leq i \leq 4$ with index $1 \leq j \leq 3$ within that block. The current order of rows and columns* *is, then,*

$$(1,1),\ (1,2),\ (1,3),\ldots,\ (4,1),\ (4,2),\ (4,3)$$

*We reorder both rows and columns as:*

$$(1,1),\ (2,1),\ (3,1),\ (4,1),\ldots,\ (1,3),\ (2,3),\ (3,3),\ (4,3),$$

*so that*

$$H_{SC,\ tailbiting} = \left(\begin{array}{cccc|cccc|cccc}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{array}\right) = \left(\begin{array}{cc|cc|cc}
0 & I_2 & 0 & 0 & I_2 & 0 \\
I_2 & 0 & 0 & 0 & 0 & \sigma \\
\hline
I_2 & 0 & 0 & I_2 & 0 & 0 \\
0 & \sigma & I_2 & 0 & 0 & 0 \\
\hline
0 & 0 & I_2 & 0 & 0 & I_2 \\
0 & 0 & 0 & \sigma & I_2 & 0
\end{array}\right).$$

*Notice that this is equal to*

$$H_{SC,\ tailbiting} = \begin{pmatrix} H_0 & 0 & H_1 \\ H_1 & H_0 & 0 \\ 0 & H_1 & H_0 \end{pmatrix}$$

*where*

$$H = H_0 + H_1 = \begin{pmatrix} 0 & I_2 \\ I_2 & 0 \end{pmatrix} + \begin{pmatrix} I_2 & 0 \\ 0 & \sigma \end{pmatrix}$$

*In other words, the power of the permutation $\tau_3$ that was assigned to an entry in the base matrix determines which $H_i$ block the entry will belong to in the SC-protograph's matrix, or how far over the corresponding edge is spread in the Tanner graph. This is the parity-check matrix of a tailbiting SC-protograph. To make a terminating SC-protograph, we move the top right block down as:*

$$H_{SC,\ terminating} = \begin{pmatrix} H_0 & 0 & 0 \\ H_1 & H_0 & 0 \\ 0 & H_1 & H_0 \\ 0 & 0 & H_1 \end{pmatrix}.$$

Rearranging rows and columns does not change the structure of the associated graph (e.g. the minimum distance of the underlying code, or the number of absorbing sets therein), but places bits in the correct order, highlights the repeated structure, and allows us to break the tailbiting portion of the code and yield a parity-check matrix in the form of Matrix (3.1). In particular, this is useful for the implementation of a windowed decoder.

### 3.3.1 Combining edge-spreading and the terminal lift

Edge-spreading and the terminal lift may be combined into a single, higher-degree lift. In other words, the entire construction process of first replacing each nonzero entry of the base matrix with an $L \times L$ circulant matrix of the form $\tau_L^k$, and then replacing each nonzero entry of the parity-check matrix of the SC-protograph with an unrestricted $J \times J$ permutation matrix $\lambda$ to perform the terminal lift, may be accomplished in a single step by assigning permutations from $S_{JL}$ to edges in the base graph. The block length of the resulting code will be $V \cdot L \cdot J$, where $V$ is the number of variable nodes in the base graph.

Making a single combined assignment per edge of the base graph, and thus per edge of the same type in the SC-protograph, is useful for two reasons: (1) breaking absorbing sets in the base graph will break absorbing sets in the terminally-lifted Tanner graph, and (2) the structure of the code is repeated, reducing storage and implementation complexity, particularly for windowed decoding.

Recall that the *Kronecker product* of two matrices $M$ and $N$ is denoted $M \otimes N$ and is given by replacing each entry $m_{ij}$ of $M$ with the matrix $m_{ij} \cdot N$.

**Theorem 3.3.5.** *To construct a tailbiting SC-LDPC code with coupling length L, memory m, and terminal lift of degree J from a base graph via a single graph lift, the possible permutation edge assignments from the permutation group $S_{JL}$ are those whose corresponding matrices of the form $\tau_L^k \otimes \lambda$ where $\tau_L$ is the $L \times L$ identity matrix left-shifted by one position, $0 \leq k \leq m$, and $\lambda$ is any $J \times J$ permutation matrix. We denote this set of permutations by $B_{L,m,J}$.*

*Proof.* The proof is clear from Lemma 3.3.1 and the above discussion. □

Notice that for $J = 1$, $B_{L,m,J} = A_{L,m}$. To give the parity-check matrix of the SC-LDPC code the structure of Matrix (3.1), we must again rearrange rows and columns after this lift is performed, and then break the tailbiting code to form a terminated SC-LDPC code. In this case, however, rows and columns are rearranged as blocks, so that $J \times J$ blocks corresponding to choices of $\lambda$ remain intact.

**Theorem 3.3.6.** *The set $B_{L,(L-1),J}$ has size $(m+1) \cdot J!$, and the element $\tau_L^k \otimes \lambda$ has order*

$$\frac{L \cdot o(\lambda) \cdot gcd(k, L, o(\lambda))}{gcd(k, L) \cdot gcd\,(L, o(\lambda))}$$

*where $o(\lambda)$ indicates the order of the permutation $\lambda$. Furthermore, $B_{L,(L-1),J}$ forms a subgroup of $S_{JL}$ for any choice of $J$ and $L$.*

*Proof.* The proof follows from Lemma 3.3.3 and properties of the Kronecker product.

□

We now discuss the case where we restrict the permutation $\lambda$ to be a cyclic shift of the $J \times J$ identity matrix.

**Corollary 3.3.7.** *The permutation given by $\tau_L^k \otimes \tau_J^\ell$ has order*

$$\frac{JL \cdot gcd(k, J, L) \cdot gcd(\ell, J, L)}{gcd(\ell, J) \cdot gcd(k, L) \cdot gcd(J, L) \cdot gcd(k, \ell, J, L)}.$$

*Proof.* This follows directly from Lemma 3.3.3 and Theorem 3.3.6. □

If we use permutations of this type to lift a base matrix, we may say more about the structure of the parity-check matrix of the resulting SC-LDPC code.

**Lemma 3.3.8.** *If a base parity-check matrix is lifted to form an SC-LDPC code using permutation matrices of the form $\tau_L^k \otimes \tau_J^\ell$ in $B_{L,m,J}$, for $J \geq 2$, then the resulting parity-check matrix is quasi-cyclic, independently of whether block rows and columns are reordered.*

*Proof.* The resulting matrix will be comprised of blocks of the form $\tau_J^\ell$. □

This structure is a consequence of the terminal degree $J \geq 2$ lift. Note that when $J = 1$ (i.e. the SC-protograph is not lifted), the parity-check matrix is not necessarily quasi-cyclic post-reordering, but will be if the base matrix is array-based and permutations are assigned constantly on blocks.

### 3.3.2 Comparison of construction methods

Of the existing methods for SC-LDPC code construction, the framework presented in Theorem 3.3.5 is the most general. In particular, traditional cutting vectors and the generalized cutting vector constructions of [47] and [48] form a proper subset of this approach.

Given a fixed array-based base graph, let the set of SC-LDPC codes formed with all possible edge-spreadings and terminal lifts as described in Theorem 3.3.5 be given by $A$, the set of codes formed using a traditional cutting vector (without a terminal lift) be given by $C$, the set of codes formed using a generalized cutting vector (also without a terminal lift) be given by $C_g$, the set of codes for which there is no terminal lift ($J = 1$ in Theorem 3.3.5) be given by $E$, and the set of codes formed by restricting $\lambda$ of Theorem 3.3.5 to be of the form $\tau_J^\ell$ (as in Lemma 3.3.8) be given by $Q$. Then we have the following nested set inclusions:

**Proposition 3.3.9.** *With $C$, $C_g$, $E$, $Q$, and $A$ defined as above,*

$$C \subsetneq C_g \subsetneq E \subsetneq Q \subsetneq A.$$

*Proof.* For an SC-LDPC code constructed using a traditional cutting vector in $C$, the memory is equal to 1, and so Lemma 3.3.1 stipulates that the two possible permutation assignments to edges of the base graph are the identity and $\tau_L$. However, there is additional structure: if the cutting vector is $\xi = (\xi_0, \xi_1, \ldots)$, then the first consecutive (blocks of) $\xi_0$ variable nodes have the identity assigned to all of their edges, while the next consecutive

$\xi_1$ (blocks) have the permutation $\tau_L$ assigned to their first edge[2], and the identity assigned to all later edges, the next $\xi_2$ (blocks of) variable nodes have the permutation $\tau_L$ assigned to their first two edges, and the identity to all later edges, etc.

In the generalized cutting vector approach of [47] and [48], which are edge-spreading methods applied specifically to array-based codes, blocks in the base matrix have constant assignments, but assignments to those blocks are not as restricted as in the traditional approach.

Relaxing the restriction of constant assignments per block of an array-based code and allowing multiple permutation assignments per block is enough to show $C_g \subsetneq E$. The final two inclusions are clear from Theorem 3.3.5 and Lemma 3.3.8.

$\square$

## 3.4 Removing absorbing sets

It has been shown that for protograph-based LDPC codes, substructures such as trapping or stopping sets may be removed and girth may be improved with certain permutation assignments in the lifting process [23, 56]. Consequently, we may remove absorbing sets by choosing suitable permutation assignments when constructing an SC-LDPC code via Theorem 3.3.5.

We consider as an example base graphs which are array-based column-weight 3 codes of the form

$$H(3,p) = \begin{pmatrix} I & I & I & \cdots & I \\ I & \sigma & \sigma^2 & \cdots & \sigma^{p-1} \\ I & \sigma^2 & \sigma^4 & \cdots & \sigma^{2(p-1)} \end{pmatrix}, \tag{3.4}$$

[2]The ordering on the edges incident to a variable node is induced by the ordering of the corresponding parity-check matrix.

where $\sigma$ is the $p \times p$ identity matrix left-shifted by 1. Figure 3.6 shows (3,3)- and (4,2)-absorbing sets, which have been shown to be the most harmful to error floor performance in such codes [58]. Notice that there is a 6-cycle in each of these absorbing sets (in fact, $(3,3)$-absorbing sets are in one-to-one correspondence with 6-cycles in this case). To remove them algebraically by lifting we can assign permutations to the edges of the cycle that increase the cycle lengths corresponding to those edges. This may be done using Theorem 2.1.5.

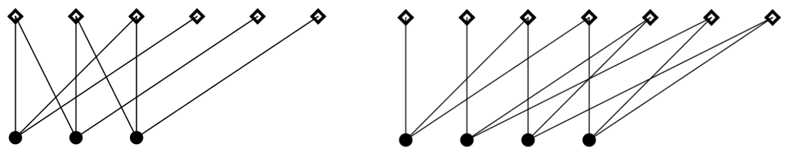


**Figure 3.6:** A $(3,3)$-absorbing set and a $(4,2)$-absorbing set in a column-weight 3, array-based code. Variable nodes are denoted by $\bullet$, and check nodes are denoted by $\diamond$. © 2017 IEEE

In the case of SC-LDPC codes, the edge permutation assignments are limited to those detailed in Theorem 3.3.5. However, even if we restrict ourselves to $m = 1$ (or 2) and no terminal lift, assigning the permutation $\tau_L$ (and $\tau_L^2$) to a strategic subset of the edges of a 6-cycle will break the 6-cycle in the lift, and hence will break the corresponding $(3,3)$-absorbing set. Thinking back to the poset structure of trapping sets, notice that since the $(3,3)$-absorbing set is a subgraph of the $(4,2)$-absorbing set in Figure 3.6, the latter are also removed.

Using Theorem 2.1.5, we may express the probability that a given $(3,3)$-absorbing set from our base graph is eliminated in the SC-protograph, given that our edge permutation assignments are random.

**Theorem 3.4.1.** *Suppose an SC-protograph is constructed via a graph-lift from an array-based, column-weight 3 base graph. If the coupling length of the SC-protograph is L, the memory is $m < \frac{L}{3}$, and permutation edge labels are assigned uniformly at random to the* *edges of the base graph, the probability that a given* $(3,3)$*-absorbing set is broken in the coupling process is given by*

$$1 - \frac{11m^4 + 44m^3 + 71m^2 + 54m + 20}{20(m+1)^5}.$$

*Proof.* In the case of an array-based, column-weight 3 base matrix, eliminating a $(3,3)$-absorbing set is equivalent to assigning permutations to its 6-cycle such that the net permutation is not equal to the identity permutation. Indeed, no other product of allowed permutations of the form $\tau^k$ has any fixed points, and so by Theorem 2.1.5, there will be no 6-cycles in the lift corresponding to the absorbing set's 6-cycle in the base graph for (allowed) non-identity net permutations. Let the assignments to the edges of a 6-cycle be $\tau_L^{k_1}, \ldots, \tau_L^{k_6}$, where all edges are directed from variable to check node. Without loss of generality, the net permutation on the cycle is given by $\tau_L$ raised to the power of

$$\sum_{i=1}^{6} (-1)^{i+1} k_i.$$

This permutation is the identity if and only if $\sum_{i=1}^{6}(-1)^{i+1}k_i \equiv 0 \pmod{L}$. By assumption, each $k_i$ is bounded between 0 and $m$, inclusive, and $3m < L$. Then, $\left|\sum_{i=1}^{6}(-1)^{i+1}k_i\right| \leq 3m$, and so the net permutation can only be the identity if $\sum_{i=1}^{6}(-1)^{i+1}k_i = 0$, or, equivalently, $k_1 + k_3 + k_5 = k_2 + k_4 + k_6$. The probability that this occurs is the sum over all possible summation values of the probability that both sums have that value:

$$= \sum_{a=0}^{3m} Pr(k_1 + k_3 + k_5 = a) Pr(k_2 + k_4 + k_6 = a) \quad (3.5)$$

$$= \sum_{a=0}^{3m} Pr(k_1 + k_3 + k_5 = a)^2 \quad (3.6)$$

For a given value of $a$, $Pr(k_1 + k_3 + k_5 = a)$ is equal to the number of triples $k_1, k_3, k_5$ whose sum is $a$, divided by the total number of possible triples, $(m+1)^3$. Notice that the number of such triples is equal to the coefficient of $x^a$ in the generating function

$$\begin{aligned}\left(\sum_{i=0}^{m} x^i\right)^3 &= \left(\frac{x^{m+1}-1}{x-1}\right)^3 \\ &= \left(\sum_{i=0}^{3}\binom{3}{i}x^{i(m+1)}(-1)^{3-i}\right)\left(-\sum_{j=0}^{\infty}\binom{3+j-1}{j}x^j\right).\end{aligned}$$

That is, the coefficient of $x^{i(m+1)+j}$ in the above product when $i(m+1) + j = a$. Using this observation, Equation (3.6) reduces to:

$$\frac{11m^4 + 44m^3 + 71m^2 + 54m + 20}{20(m+1)^5}.$$

□

**Remark 3.4.2.** *Notice that the result depends only on the prescribed memory of the SC-protograph, provided that the coupling length is large enough. The stipulation that the coupling length be at least three times the memory is nearly a given for practical applications, in which the memory is kept small, and the coupling length large.*

**Corollary 3.4.3.** *Suppose an SC-protograph code is constructed via a graph-lift from an array-based, column-weight 3 base graph. If the coupling length of the SC-protograph is L, the memory is $m < \frac{L}{3}$, and permutation edge labels are assigned uniformly at random to the edges of the base graph, the probability that a given $(3,3)$-absorbing set is broken in the coupling process approaches $1$ as m goes to infinity.*

*Proof.* The result follows by taking the limit of the probability in Theorem 3.4.1. □

Of course, we may do better than random assignments by cleverly choosing permutation edge assignments in a more structured way. Using the algebraic lift method of constructing SC-LDPC codes, in [13] we used my co-author's low-complexity algorithm to count the number of absorbing sets in SC-LDPC codes constructed with algorithmically-chosen permutation assignments. For array-based, column-weight 3 base graphs, our structured graph-lift approach to eliminating $(3,3)$-absorbing sets is able to outperform the edge-spreading accomplished by the optimized cutting vector, and also outperforms the results of [47] and [48]. In this case, permutation assignments were made by block, meaning that nonzero entries of each copy of $\sigma^t$ in the parity-check matrix were assigned the same lifting permutation. Furthermore, memory was restricted to $m = 1$ or $m = 2$. Absorbing set counts for three different code constructions, as well as for the constructions of [47] and [48], are shown in Table 3.1 for various coupling lengths [13]. These three code constructions are described below. Recall that $H(\gamma, p)$ denotes the $p\gamma \times p^2$ array-based parity-check matrix, as described in Section 3.1.1.

**Code 1:** Obtained by coupling $H(3,17)$ using the optimal cutting vector of [58] (recall that for cutting vectors, $m = 1$).

**Code 2:** Obtained by an optimized lifting of $H(3,17)$ with $m = 1$.

**Code 3:** Obtained by an optimized lifting of $H(3,17)$ with $m = 2$.

| **L** | **Code 1** | **Code 2** | **Code 3** | **[47] for m=2** | **[48] for m=2** |
|---|---|---|---|---|---|
| 10 | 19108 | 5644 | 442 | 646 | n/a |
| 20 | 39508 | 11764 | 952 | 1326 | n/a |
| 30 | 59908 | 17884 | 1462 | 2006 | 4335 |
| 40 | 80308 | 24004 | 1972 | 2686 | n/a |
| 50 | 100710 | 30124 | 2482 | 3366 | n/a |

**Table 3.1:** The number of $(3,3)$-absorbing sets in Codes 1-3, in addition to the results presented in [47,48]. It should be noted that [48] only provides counts for the case $L = 30$. © 2017 IEEE

| Window Length | Code 1 | Code 2 | Code 3 |
|---|---|---|---|
| 2 | 1700 | 51 | n/a |
| 3 | 3740 | 544 | n/a |
| 4 | 5780 | 1156 | 0 |
| 5 | 7820 | 1768 | 85 |

**Table 3.2:** The number of $(3, 3)$-absorbing sets in Codes 1-3 for varying window lengths. © 2017 IEEE

Observe that permutation assignments aimed at breaking 6-cycles in $H(3, 17)$ (see Codes 2 and 3 of Table 3.1) give a marked improvement on comparable constructions for both the $m = 1$ and $m = 2$ cases. Further generalization to multiple assignments per block and to higher memory is expected to yield even more improvement.

As suggested in Section 3.2, a windowed decoder works even further to our advantage: in [13], we also show that with certain parameters, we are able to eliminate all $(3, 3)$-absorbing sets as seen by the windowed decoder (see Code 3 with window length 4 in Table 3.2). In particular, we employ a slight variation on the window described in Section 3.1.2, as described in [13]. We sum the observed $(3, 3)$-absorbing sets within one window over all possible positions of the windowed decoder. These results are shown in Table 3.2.

# Chapter 4

# Bounds on Stopping and Absorbing Set Sizes

For specific classes of codes, we can determine the parameters of the most harmful absorbing or stopping sets using known structural properties. Knowledge of these parameters not only helps us to understand the performance of these classes under iterative decoding, but gives us a starting point for trapping and absorbing set removal. In this chapter, we look at two classes of codes – codes based on finite geometries and codes constructed using hypergraphs – and examine the structure of harmful substructures in their Tanner graphs.

## 4.1 Finite geometry LDPC codes

Codes constructed from finite geometries, called finite geometry LDPC (FG-LDPC) codes, were first introduced by Kou, Lin, and Fossorier, who gave families of cyclic and quasi-cyclic LDPC codes with parity-check matrices determined by the incidence structure of finite Euclidean and projective geometries [9, 59]. Since then, a wide variety of FG-LDPC codes have been introduced and analyzed; creative constructions of codes using other finite incidence structures such as generalized quadrangles and Latin squares have also been studied extensively [60–62]. In particular, the structure inherent in these constructions allow for ease of determination of parameters of the resulting codes. In this section, we use the structure of FG-LDPC codes to determine the parameters of the most harmful absorbing sets present in such codes. The results of this section were developed in collaboration with Haymaker and Kelley [14]; a result will occasionally be stated without proof when the proof is primarily the work of these coauthors.

We begin by giving the basic properties of finite Euclidean and projective geometries. Recall the definitions of affine and projective spaces (see e.g. [63]):

**Definition 4.1.1.** *A* linear space *is a collection of points and lines such that any line has at least two points, and a pair of points share exactly one line. A* hyperplane *of a linear space is a maximal proper subspace. A* projective plane *is a linear space of dimension 2 in which any two lines meet, and there exists a set of four points no three of which are collinear. A* projective space *is a linear space in which any two-dimensional subspace is a projective plane. An* affine space *is a projective space with one hyperplane removed.*

The set of points formed by $m$-tuples with entries from the finite field $\mathbb{F}_q$ forms an affine space, called a *finite Euclidean geometry*. For the case in which $m = 2$, lines of the Euclidean geometry are sets of points $(x, y) \in \mathbb{F}_q^2$ satisfying either $y = mx + b$ or $x = a$ for some $m, b, a \in \mathbb{F}_q$. Formally,

**Definition 4.1.2.** *The $m$-*dimensional finite Euclidean geometry *$EG_0(m, q)$ is a linear space satisfying the following: it has $q^m$ points and $\frac{q^{m-1}(q^m-1)}{q-1}$ lines. Each line contains $q$ points, and each point is on $\frac{q^m-1}{q-1}$ lines. Any two points have exactly one line in common and any two lines either have one point in common or are parallel (i.e., have no points in common).*

It is common to define a code using a modified version of $\mathrm{EG}_0(m, q)$, in which the origin point is removed and every line containing the origin is also deleted. Defining a code from this modified geometry results in a cyclic or quasi-cyclic code, which, as previously mentioned, have advantages for practical implementation. By convention, the notation $\mathrm{EG}(m, q)$ is used to refer to the Euclidean geometry with the origin removed [59, 64]. We use $\mathrm{EG}_0(m, q)$ to distinguish the case when the origin and all lines containing it are included.

Next, we recall the parameters of projective geometries:

**Definition 4.1.3.** *The* $m$-dimensional finite projective geometry $\mathrm{PG}(m, q)$ *is a linear space satisfying the following: it has* $\frac{q^{m+1}-1}{q-1}$ *points and* $\frac{(q^m+\cdots+q+1)(q^{m-1}+\cdots+q+1)}{(q+1)}$ *lines. Each line contains* $q + 1$ *points, and each point is on* $\frac{q^m-1}{q-1}$ *lines. Any two points have exactly one line in common and each pair of lines has exactly one point in common.*

An important subclass of finite geometries are the finite Euclidean and projective planes: that is, finite Euclidean and projective geometries for which $m = 2$. The parameters for $\mathrm{EG}_0(2, q)$, $\mathrm{EG}(2, q)$, and $\mathrm{PG}(2, q)$ and are organized in Table 4.1 for ease of reference.

| | $\mathrm{EG}_0(2,q)$ | $\mathrm{EG}(2,q)$ | $\mathrm{PG}(2,q)$ |
|---|---|---|---|
| Number of points | $q^2$ | $q^2-1$ | $q^2+q+1$ |
| Number of lines | $q(q+1)$ | $q^2-1$ | $q^2+q+1$ |
| Number of points on each line | $q$ | $q$ | $q+1$ |
| Number of lines that intersect at a point | $q+1$ | $q$ | $q+1$ |

**Table 4.1:** Parameters of finite Euclidean and projective planes.

A $\mu$*-flat* is a $\mu$-dimensional subspace of a finite geometry. We may define an LDPC code with parity-check matrix equal to the incidence matrix of $\mu_1$-flats and $\mu_2$-flats, where $0 \leq \mu_1 < \mu_2 \leq m$, of an $m$-dimensional finite geometry. In this section, we will consider binary codes defined as the null space of an incidence matrix of the points and lines in a finite geometry, as in [59]: that is, we will focus on the case in which $\mu_1 = 0$ and $\mu_2 = 1$.

Let $H_{\mathrm{EG}}(m, q)$ denote a (binary) parity-check matrix which is the incidence matrix of points and lines in $\mathrm{EG}(m, q)$, and let $C_{\mathrm{EG}}(m, q)$ denote the corresponding code. Similarly, $H_{\mathrm{PG}}(m, q)$ and $C_{\mathrm{PG}}(m, q)$ are defined for codes from finite projective geometries. Points of the finite geometry FG correspond to columns in $H_{\mathrm{FG}}(m, q)$ or variable nodes in the corresponding Tanner graph, and lines in the geometry correspond to rows in the parity-check matrix or check nodes in the Tanner graph.

It has been shown that binary EG- and PG-LDPC codes perform well under the sum-product and other iterative decoding algorithms [59], but these codes may still experience error floors in their BER curves. Graphical substructures affecting the error floor performance of these codes have been studied extensively: general trapping sets of FG-LDPC codes were studied in [65, 66], stopping sets and pseudocodewords were examined in [61, 67, 68], and absorbing sets of LDPC codes from finite planes over fields of characteristic 2 were analyzed in [64].

The *smallest* $(a, b)$-absorbing sets of the Tanner graph of a code are absorbing sets with the minimum possible $a$ value, and the corresponding smallest $b$ for that given $a$; these absorbing sets are regarded as the most problematic for iterative graph-based decoders. In the case of FG-LDPC codes, high code structure allows for explicit characterization of these undesirable small absorbing sets, as well as their enumeration. Although an absorbing set is formally a set of variable nodes in the Tanner graph corresponding to a parity-check matrix $H$, we will refer in this section to absorbing sets of the matrix $H$ for brevity.

Using an argument similar to that used in the proof of Tanner's tree bound (Theorem 2.1.2, [3]), Dolecek obtains the following bounds on the parameters of a smallest absorbing set in a Tanner graph whose variable nodes all have degree $r$. We will call such a Tanner graph $r$-left regular.

**Lemma 4.1.4.** *[11] Let $t = \lceil \frac{r+1}{2} \rceil$, where $r$ is the variable node degree. If $g$, the girth of the graph, is at least $6$, the smallest $(a^*, b^*)$-absorbing sets have parameters bounded as follows:*

$$a^* \geq \begin{cases} 1 + \sum_{i=0}^{\ell} t(t-1)^i & \text{for } g \equiv 2 \pmod 4 \\ 1 + \sum_{i=0}^{\ell-1} t(t-1)^i + (t-1)^{\ell} & \text{for } g \equiv 2 \pmod 4, \end{cases}$$

*where $\ell = \left\lfloor \frac{g}{4} \right\rfloor - 1$, and*

$$b^* \geq a^* \cdot \left\lfloor \frac{r-1}{2} \right\rfloor.$$

The Tanner graph of a PG$(m, q)$ or an EG$(m, q)$-LDPC code with $m \geq 2$ has girth 6 [59], and so Lemma 4.1.4 gives lower bounds of $a^* \geq 1 + t$ and $b^* \geq a^* \cdot \lfloor \frac{r-1}{2} \rfloor$ for the FG-LDPC codes we will consider, where $r$ is the number of lines containing a given point. For $H_{\mathrm{EG}_0}(m, q)$ and $H_{\mathrm{PG}}(m, q)$, which have the same variable node degrees,

**Corollary 4.1.5.** *The parameters $(a^*, b^*)$ of the smallest absorbing sets in the Tanner graphs of $H_{EG_0}(m, q)$ and $H_{PG}(m, q)$ for $m \geq 2$ are*

$$a^* \geq \left\lceil \frac{q^{m-1} + q^{m-2} + \cdots + q + 2}{2} \right\rceil + 1, \quad \textit{and} \quad b^* \geq a^* \cdot \left\lfloor \frac{q^{m-1} + q^{m-2} + \cdots + q}{2} \right\rfloor.$$

*Proof.* This follows directly from Lemma 4.1.4 using the girth and variable node degrees of the PG-LDPC codes and EG$_0$-LDPC codes. □

For the case in which $m = 2$, these bounds reduce to:

$$a^* \geq \left\lceil \frac{q+2}{2} \right\rceil + 1, \quad \text{and} \quad b^* \geq a^* \cdot \left\lfloor \frac{q}{2} \right\rfloor.$$

Liu et al. used geometric substructures called *k-arcs* to find the parameters of absorbing sets of FG-LDPC codes for $q = 2^s$ [64].

**Definition 4.1.6.** *A* $(k, d)$-arc *in a finite affine or projective plane is a set of k points such that any d of the points are collinear, and any collection of $d + 1$ are not collinear. Often, $(k, 2)$-arcs are simply referred to as* $k$-arcs.

**Definition 4.1.7.** *A* $k$-cap *in a finite affine or projective geometry is a set of k points, no three of which are collinear.*

In finite planes, $k$-caps coincide with $k$-arcs, but in higher dimensions the notions are distinct (see [69] for the complete definition of an arc in higher dimensions). Finding bounds on the size of a maximal cap in finite geometries is an ongoing area of research [69–71].

### 4.1.1 Smallest absorbing sets for $\mathrm{EG}_0$ and PG using $k$-caps

In this section, we show that for FG-LDPC codes $C_{\mathrm{EG}_0}(2, q)$, $C_{\mathrm{EG}_0}(3, q)$, and $C_{\mathrm{PG}}(3, q)$, equality is met in Lemma 4.1.4. The general approach in [64] for finding the parameters $(a^*, b^*)$ of the smallest absorbing sets in $C_{\mathrm{EG}}(2, 2^s)$ and $C_{\mathrm{PG}}(2, 2^s)$ is to first find a $k$-arc, where $k$ is the minimal possible value for $a^*$ given by Lemma 4.1.4: $k = 2^{s-1} + 2$. We use a similar approach for the case of $\mathrm{EG}_0$ and PG families of dimensions $m = 2$ and $m = 3$; notice that we do not restrict our $q$ values to be powers of 2.

**Theorem 4.1.8.** *The parameters $(a^*, b^*)$ of the smallest absorbing sets for $H_{\mathrm{EG}_0}(2, q)$ satisfy $a^* = \left\lceil \frac{q+2}{2} \right\rceil + 1$, and $b^* = a^* \cdot \left\lfloor \frac{q}{2} \right\rfloor$.*

*Proof.* Recall that the variable node degree of $H_{\mathrm{EG}_0}(2, q)$ is $q + 1$, and let $t = \left\lceil \frac{q+2}{2} \right\rceil$. That is, for any variable node in an absorbing set of $H_{\mathrm{EG}_0}(2, q)$, $t$ is the minimum number of its neighbors that have even degree in the subgraph induced by the absorbing set and its neighbors. From Lemma 4.1.4, we know that $a^*$ is bounded below by $t + 1$. Thus, it is sufficient to show that there exists a set of $t + 1$ variable nodes that forms an absorbing set in $H_{\mathrm{EG}_0}(2, q)$.

We construct such an absorbing set $A$ by choosing $t + 1$ variable nodes that lie in a $(q + 1)$-cap; the existence of such a cap is shown in [70]. From the structure of $\mathrm{EG}_0(2, q)$, each variable node in $A$ shares exactly one check node neighbor with each other variable node in $A$. Since $A$ forms a $(t + 1)$-cap, every check node has degree at most 2. Thus, every variable node in $A$ has exactly $t$ neighbors of degree 2 in the subgraph induced by $A$ and its neighbors, and $(q+1)-t < t$ neighbors of degree 1. We conclude that $A$ forms an absorbing set with parameters

$$a^* = t + 1 = \left\lceil \frac{q+2}{2} \right\rceil + 1, \text{ and}$$

$$b^* = a^* \cdot ((q+1) - t) = a^* \cdot \left( (q+1) - \left\lceil \frac{q+2}{2} \right\rceil \right) = a^* \cdot \left\lfloor \frac{q}{2} \right\rfloor.$$

□

The proof of the following theorem was completed primarily by my coauthor of [14], and thus has been omitted here. However, it follows a similar argument to the proof of Theorem 4.1.8.

**Theorem 4.1.9.** *The parameters* $(a^*, b^*)$ *of the smallest absorbing sets for* $H_{EG_0}(3, q)$ *and* $H_{PG}(3, q)$ *satisfy* $a^* = \left\lceil \frac{1}{2} \cdot \left(q^2 + q + 2\right) \right\rceil + 1$*, and* $b^* = a^* \cdot \left\lfloor \frac{1}{2} \cdot \left(q^2 + q\right) \right\rfloor$.

### 4.1.2 Absorbing set parameters and the tree bound

In this section, we give other classes of codes based on finite geometries that have minimum absorbing set parameters matching the bounds in Lemma 4.1.4. Equality is shown for codes from projective planes using a tree-based argument, where the Tanner graph of the code is enumerated as a tree for as many layers as the girth permits (this may result in a partial graph). We further show that the bound of Lemma 4.1.4 holds for $H_{\mathrm{EG}}(2, q)$ when $q$ is odd, building on the characteristic 2 results of [64], and that the smallest absorbing sets of $H_{\mathrm{EG}_0}(2, q)$ and $H_{\mathrm{PG}}(2, q)$ are elementary. Finally, any code whose minimum distance satisfies the tree bound of Theorem 2.1.2 [3] has minimum absorbing set parameters that meet the bound in Lemma 4.1.4. The proof of this result was primarily done by my coauthor; its proof appears in [14].

Recall that in [64], the authors use $k$-arcs to show that equality in Lemma 4.1.4 is met for $H_{\mathrm{PG}}(2, q)$ and $H_{\mathrm{EG}}(2, q)$ when $q = 2^s$. It is also true that the the bounds are met for $H_{\mathrm{PG}}(2, q)$

when $q$ is odd; the proof of the following result utilizes the tree-like representation and labeling of the Tanner graph of $C_{\mathrm{PG}}(2,q)$ from [61].

**Theorem 4.1.10.** *The parameters $(a^*, b^*)$ of the smallest absorbing sets for $H_{PG}(2,q)$ satisfy $a^* = \left\lceil \frac{q+2}{2} \right\rceil + 1$, and $b^* = a^* \cdot \left\lfloor \frac{q}{2} \right\rfloor$.*

*Proof.* Notice that the variable and check node degrees of the graph corresponding to this code are both equal to $q+1$. Enumerate the graph as tree in the following way: Layer 0 consists of a single root variable node which is chosen from the set of the graph's variable nodes. The next layer, Layer 1, consists of the $q+1$ check node neighbors of the root. Label these check nodes $x_c$, $0_c$, $1_c$, $\alpha_c$, ..., $\alpha_c^{q-2}$, where $\alpha$ is a primitive element of $GF(q)$, so that in particular $\alpha^{q-1} = 1$.

Layer 2 contains $q(q+1)$ variable nodes. Because the girth of $H_{\mathrm{PG}}(2,q)$ is equal to 6, these variable nodes are all distinct, and thus comprise all remaining variable nodes in the graph. For each $i \in \{x, 0, 1, \alpha, \ldots, \alpha^{q-2}\}$, denote the $q$ variable nodes in Layer 2 descending from check node $i_c$ in Layer 1 by

$$(i,0)_v, (i,1)_v, (i,\alpha)_v, \ldots, (i,\alpha^{q-2})_v.$$

Layer 3 contains the remaining $q^2$ check nodes. For each $i \in \{0, 1, \alpha, \ldots, \alpha^{q-2}\}$ label the $q$ check nodes in the $i$th group of $q$ check nodes as

$$(i,0)_c,\ (i,1)_c,\ (i,\alpha)_c,\ \ldots,\ (i,\alpha^{q-2})_c.$$

The nodes from Layer 2 and Layer 3 are connected (using mutually orthogonal Latin squares (MOLS)) as follows [61]:

- For $j \in \{0, 1, \alpha, \ldots, \alpha^{q-2}\}$, variable node $(x, j)_v$ in Layer 2 connects to the following

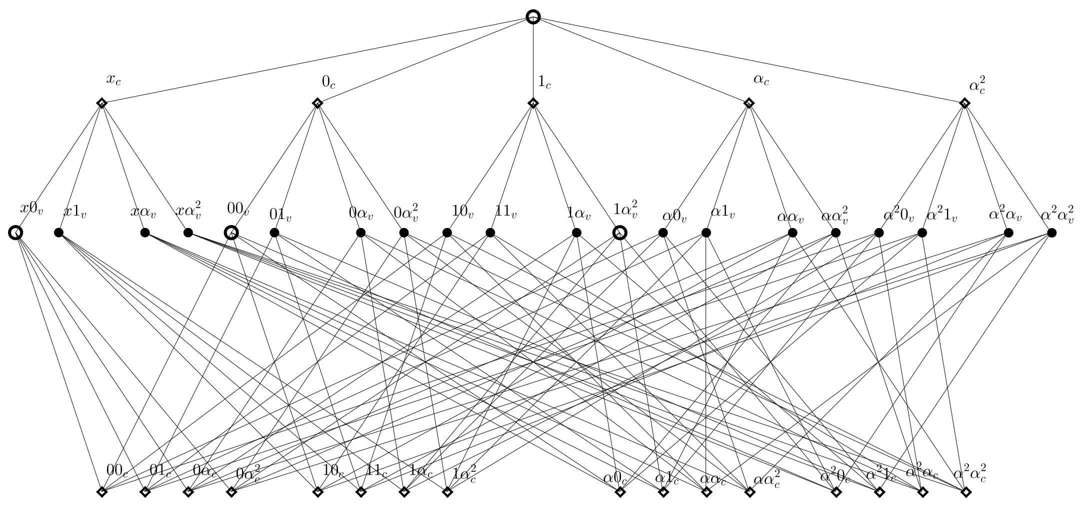


**Figure 4.1:** A tree diagram of a Tanner graph for $C_{\mathrm{PG}}(2,4)$ [61].

$q$ check nodes in Layer 3:

$$(j,0)_c,\ (j,1)_c,\ (j,\alpha)_c,\ \ldots,\ (j,\alpha^{q-2})_c.$$

- For $i, j \in \{0,1,\alpha,\ldots,\alpha^{q-2}\}$, variable node $(i,j)_v$ in Layer 2 connects to the $q$ check nodes in Layer 3:

$$(0, j+i\cdot 0)_c,\ (1, j+i\cdot 1)_c,\ (\alpha, j+i\cdot\alpha)_c,\ \ldots,\ (\alpha^{q-2}, j+i\cdot\alpha^{q-2})_c.$$

We claim that, together with our root node, the following set of variable nodes forms an absorbing set in $H_{\mathrm{PG}}(2,q)$:

$$S = \{(x,0)_v,\ (0,0)_v,\ (1,\alpha^{q-2})_v,\ (\alpha,\alpha^{q-3})_v,\ \ldots,(\alpha^{t-3},\alpha^{q+1-t})_v\}.$$

To prove our claim, we show that every variable node in $S$ has at least $t$ neighbors of even degree in the subgraph induced on the root, $S$, and their check node neighbors. Indeed, it is clear that the root has at least $t$ such neighbors. Based on the connections described above, among the check nodes involving active variable node $(x, 0)_v$, we have the following: $(0, 0)_c$ has active neighbors $(x, 0)_v$ and $(0, 0)_v$, and no others in the set; $(0, \alpha^{q-2})_c$ has active neighbors $(x, 0)_v$ and $(1, \alpha^{q-2})_v$; $(0, \alpha^{q-3})_c$ has active neighbors $(x, 0)_v$ and $(\alpha, \alpha^{q-3})_v$; and so on. Finally, $(0, \alpha^{q+1-t})_c$ has active neighbors $(x, 0)_v$ and $(\alpha^{t-3}, \alpha^{q+1-t})_v$.

Thus, at least $t - 1$ of $(x, 0)_v$'s neighbors in Layer 3 have degree 2 in the subgraph induced on $S$ and its neighbors. Similarly, for each element of $S$, we may demonstrate $t-1$ check node neighbors in Layer 3 that have degree 2 in the subgraph induced on $S$ and its neighbors (see Theorem 4.3 in [61]). Thus, each variable node in $S$ has at least $t$ even-degree neighbors in the induced subgraph (including the degree 2 check node neighbor in Layer 1 that it shares with the root), and the set $S$ together with the root forms an absorbing set.

Because each check node counted above was counted twice in this process, we have found a total of $|S|(t - 1)/2 = t(t - 1)/2 = \binom{t}{2}$ check nodes in Layer 3 that have degree 2 in the induced subgraph. Because each pair of points in PG$(2, q)$ intersects in exactly one line, this shows that there are no check nodes of higher degree in the induced subgraph. Thus, this minimal absorbing also realizes the lower bound on $b^*$ from Lemma 4.1.4: each variable node in the absorbing set has exactly $q + 1 - t$ check node neighbors of odd degree (in particular, of degree 1) in the induced subgraph, giving

$$b^* = a^* \cdot ((q + 1) - t) = a^* \cdot \left\lfloor \frac{q}{2} \right\rfloor .$$

□

The smallest absorbing sets of $H_{\text{PG}}(2, q)$ are not necessarily fully absorbing, as is seen in the next example:

**Example 4.1.11.** *The* $(4,8)$*-absorbing set from the Tanner graph pictured in Figure 4.1 is not a fully absorbing set. To see this, consider the variable node outside of the absorbing set labeled* $\alpha^2 1_v$*, which is adjacent to the three check nodes* ($01_c$, $\alpha 0_c$, *and* $1\alpha_c$) *that are odd in the subgraph induced by the absorbing set. Therefore this absorbing set is not a fully absorbing set.*

However, they are elementary:

**Proposition 4.1.12.** *Every smallest absorbing set in* $H_{EG_0}(2,q)$ *or* $H_{PG}(2,q)$ *is elementary.*

*Proof.* As shown in Theorems 4.1.8 and 4.1.10, every smallest absorbing set $A$ has $t+1$ variable nodes, where $t$ is calculated based on the variable node degree $r$ as $\left\lceil \frac{r+1}{2} \right\rceil$. Each variable node in $A$ has exactly one check node neighbor in common with each of the other variable nodes in $A$. Thus, each variable node has at most $t$ neighbors of degree greater than 1 in the induced subgraph. Every variable node in $A$ has exactly $t$ neighbors of degree greater than 1 if and only if every pair of variable nodes in $A$ has a distinct check node neighbor. In other words, $A$ has exactly $t$ neighbors of degree greater than 1 if and only if the points of the geometry corresponding to the variable nodes in $A$ form a $(t+1)$-cap. Since each variable node in $A$ must have at least $t$ even degree neighbors in the induced subgraph in order to form an absorbing set, we see that this is indeed the case, and $A$ is thus an elementary absorbing set. □

Next, we show that equality in Lemma 4.1.4 is met for $H_{\mathrm{EG}}(2,q)$ when $q$ is odd.

**Theorem 4.1.13.** *When q is odd, the parameters* $(a^*, b^*)$ *of the smallest absorbing sets for* $H_{EG}(2,q)$ *satisfy* $a^* = \left\lceil \frac{q+1}{2} \right\rceil + 1$*, and* $b^* = a^* \cdot \left\lfloor \frac{q-1}{2} \right\rfloor$.

*Proof.* We begin by constructing EG$(2,q)$ from PG$(2,q)$, as in [61]. Starting with the graph for PG$(2,q)$ as described in the proof of Theorem 4.1.10, delete the root node in Layer 0 and its check node neighbors, which form Layer 1. Now, each variable node in Layer 2 has degree $q$, and the degree of each check node in Layer 3 remains $q+1$. Next, delete a check node in Layer 3 that is not adjacent to any of the variable nodes of the smallest absorbing set constructed in the proof of Theorem 4.1.10. Such a check node exists, because the neighbors in Layer 3 of this set do not constitute all of Layer 3.

Because the girth of the graph is 6, this process will remove exactly one variable node in each of the clouds in Layer 2. This reduces the degrees of each of the remaining check nodes by 1. The resulting graph is $q$-regular with $q^2 - 1$ vertices of each type. This graph corresponds to $\mathrm{EG}(2, q)$ [61].

Consider the $\left\lceil \frac{q+2}{2} \right\rceil + 1$ variable nodes in the absorbing set constructed in the proof of Theorem 4.1.10. Exactly one of these variable nodes was deleted in the process described above: the root of the graph. We claim that the remaining $\left\lceil \frac{q+2}{2} \right\rceil$ form an absorbing set. Each has $q$ check node neighbors, and $\left\lceil \frac{q+2}{2} \right\rceil - 1$ even-degree neighbors. Since $q$ is odd,

$$\left\lceil \frac{q+2}{2} \right\rceil - 1 = \frac{q+3}{2} - 1 = \frac{q+1}{2} > \frac{q}{2},$$

and so each of these variable nodes still has more even-degree than odd-degree neighbors. Thus, they form an absorbing set of size

$$a^* = \left\lceil \frac{q+2}{2} \right\rceil = \frac{q+3}{2} = \frac{q+1}{2} + 1 = \left\lceil \frac{q+1}{2} \right\rceil + 1, \quad \text{with}$$

$$b^* = a^* \cdot \left( q - \frac{q+1}{2} \right) = a^* \cdot \left( \frac{q-1}{2} \right) = a^* \cdot \left\lfloor \frac{q-1}{2} \right\rfloor$$

variable nodes of odd degree. These parameters meet the bounds of Lemma 4.1.4.

□

**Theorem 4.1.14.** *Let $C$ be a block length $n$ linear code defined by an $r$-left regular Tanner graph with girth $g$. If the minimum distance of $C$ equals the tree bound, the parameters*

*$(a^*, b^*)$ of the smallest absorbing sets satisfy*

$$a^* = \begin{cases} 1 + t + t(t-1) + t(t-1)^2 + \cdots + t(t-1)^{\frac{g-6}{4}} & \frac{g}{2} \text{ odd} \\ 1 + t + t(t-1) + t(t-1)^2 + \cdots + t(t-1)^{\frac{g-8}{4}} + (t-1)^{\frac{g-4}{4}} & \frac{g}{2} \text{ even} \end{cases}$$

*where* $t = \left\lceil \frac{r+1}{2} \right\rceil$, *and* $b^* = a^* \cdot \left\lfloor \frac{r-1}{2} \right\rfloor$.

Theorem 4.1.14 shows that any code with a Tanner graph whose minimum distance equals the tree bound of Theorem 2.1.2 [3] also has a smallest absorbing set whose parameters meet the bounds in Lemma 4.1.4. Thus, more code families can be shown to have this characteristic. In particular, in Table IV of [61], several examples of such codes are given, including the codes based on finite generalized quadrangles over $\mathbb{F}_{2^s}$.

## 4.2 Hypergraph codes

Codes from regular hypergraphs with expansion-like properties were introduced and analyzed in [10] and [72]. As was shown for expander codes that have underlying expander graph representations, the authors show that better expansion (referred to as $\epsilon$-homogeneity in the hypergraph case) implies improved minimum distance and error correction. The past decade has seen an increased interest in coding for distributed storage systems (DSS) due to the increasing amounts of data that need to be stored and accessed across many servers. A primary focus in this area is the design of codes with locality properties, where error correction of small sets of symbols may be performed efficiently without having to access all symbols or all information from accessed symbols, and where data may be protected by multiple repair groups. In this section, we consider codes based on regular hypergraphs, and present bounds on their error-correction capabilities in the context of distributed storage, specifically the minimum stopping set size and cooperative locality of the codes. In addition, we present a definition of hypergraph lifts, which allow for the construction of protograph hypergraph codes.

A *hypergraph* $\mathcal{H}$ is defined by a set of vertices, $V$, and a set of edges, $E$, where $E$ is a set of subsets of $V$. A hypergraph is said to be *$t$-uniform* if every edge contains exactly $t$ vertices, and is *$t$-partite* if the vertex set $V$ can be partitioned into $t$ sets $V_1, \ldots, V_t$ such that no edge contains more than one vertex from any part. Notice that a 2-uniform hypergraph is simply a graph. We will use the notation $\mathcal{H} = (V_1, V_2, \ldots, V_t; E)$ to denote a $t$-uniform $t$-partite hypergraph. In this case, each edge must contain exactly one vertex from each part. Finally, a hypergraph is *$\Delta$-regular* if every vertex belongs to $\Delta$ edges. In this section, all hypergraphs considered have no parallel edges: no two distinct edges are comprised of exactly the same set of vertices.

The class of $t$-uniform $t$-partite $\Delta$-regular hypergraphs were used in [10] and [72] to design codes, letting the edges of the hypergraph represent the code bits (or, more generally, symbols), and the vertices of the hypergraph represent constraints. Specifically, when there are $n$ vertices in each part, the block length of the corresponding *hypergraph code* is $n\Delta$ and the number of constraint vertices is $nt$. As in the GLDPC codes of [3], each constraint node represents a linear block length $\Delta$ "subcode," and is satisfied when the assignment on the edges incident to that constraint node form a codeword of the subcode.

Recall that Sipser and Spielman showed that the guaranteed error correction capabilites of a code may be improved when the underlying graph is a good expander [21]. Loosely speaking, a graph is a good expander if small sets of vertices have large sets of neighbors; graphs with small second largest eigenvalue in absolute value (or, equivalently, a large *spectral gap*) have this property [73]. In particular, a finite, connected, $d$-regular graph with second largest eigenvalue $\lambda$ is a *Ramanujan graph* if $|\lambda| \leq 2\sqrt{d-1}$.

Let $G$ be a $d$-regular bipartite graph with $n$ vertices in each part, and let $\lambda$ be the second largest eigenvalue (in absolute value) of the adjacency matrix of $G$. Then for subsets $A_1$ and

$A_2$ of the left and right vertices, respectively, the number of edges in the subgraph induced by $A_1 \cup A_2$ is at most $\frac{|E(A_1,A_2)|}{nd} \leq \alpha_1\alpha_2 + \frac{\lambda}{d}\sqrt{\alpha_1\alpha_2}$, where $|A_i| = \alpha_i n$ (see, e.g. [74]). Bilu and Hoory use an analogous version of this property to introduce a notion of hypergraph expansion:

**Definition 4.2.1.** *[10] Let $\mathcal{H} = (V_1, V_2, \ldots, V_t; E)$ be a t-uniform t-partite $\Delta$-regular hypergraph with n vertices in each part. Then $\mathcal{H}$ is $\epsilon$-*homogeneous *if for every choice of $A_1, A_2, \ldots, A_t$ with $A_i \subseteq V_i$ and $|A_i| = \alpha_i n$,*

$$\frac{|E(A_1, A_2, \ldots, A_t)|}{n\Delta} \leq \prod_{i=1}^{t} \alpha_i + \epsilon\sqrt{\alpha_{\sigma(1)}\alpha_{\sigma(2)}},$$

*where $\sigma \in S_t$ is a permutation on $[t]$ such that $\alpha_{\sigma(i)} \leq \alpha_{\sigma(i+1)}$ for each $i \in [t-1]$, and $E(A_1, \ldots, A_t)$ denotes the set of edges which intersect all of the $A_i$'s.*

Then, hypergraphs with $\epsilon$-homogeneity for small $\epsilon$ are in some sense good expanders.

Let $[N, K, D]$ denote a binary linear code with block length $N$, dimension $K$, and minimum distance $D$. The following bounds on the rate and minimum distance of a code $\mathcal{Z}$ from an $\epsilon$-homogeneous $t$-uniform $t$-partite $\Delta$-regular hypergraph with $n$ vertices in each part and a $[\Delta, \Delta R, \Delta\delta]$ subcode $C$ at each constraint node are given in [10]:

$$\begin{aligned} \text{rate}(\mathcal{Z}) &\geq tR - (t-1) \\ d_{\min}(\mathcal{Z}) &\geq n\Delta\left(\delta^{\frac{t}{t-1}} - c(\epsilon, \delta, t)\right) \end{aligned}$$

where $c(\epsilon, \delta, t) \to 0$ as $\epsilon \to 0$. Note that $n\Delta$ is the block length of $\mathcal{Z}$.

The *locality* of a code measures how many code symbols must be used to recover an erased code symbol. While there are a variety of locality notions relevant to coding for distributed storage, we will focus on $(r, \ell)$-cooperative locality and $(r, \tau)$-availability [75, 76].

**Definition 4.2.2.** *A code $C$ has* $(r, \ell)$-cooperative locality *if for any $\mathbf{y} \in C$, any set of $\ell$ symbols in $\mathbf{y}$ are functions of at most $r$ other symbols. Furthermore, $C$ has* $(r, \tau)$-availability *if any symbol in $\mathbf{y}$ can be recovered by using any of $\tau$ disjoint sets of symbols each of size at most $r$.*

### 4.2.1 Bounds on regular hypergraph codes

In this section, we examine the erasure correction and cooperative locality of regular hypergraph codes. We first define stopping sets for regular hypergraph codes, and give a lower bound on the minimum stopping set size.

**Definition 4.2.3.** *Let $\mathcal{Z}$ be a code on a hypergraph $\mathcal{H} = (V_1, \dots, V_t; E)$, with edges representing code symbols and vertices representing the constraints of a subcode $C$. Then a* stopping set *$S$ is a subset of the edges of $\mathcal{H}$ such that every vertex contained in an element of $S$ is contained in at least $d_{\min}(C)$ elements of $S$.*

Though the size of a minimum stopping set depends on both the hypergraph representation and the choice of subcode, we denote this size by $s_{\min}(\mathcal{H})$, and assume that the subcode is clear from context.

**Theorem 4.2.4.** *Let $\mathcal{H}$ be a $t$-uniform $t$-partite $\Delta$-regular hypergraph. If the vertices of $\mathcal{H}$ represent constraints of a subcode $C$ with minimum distance $d_{\min}(C)$ and block length $\Delta$, then the size of the minimum stopping set, $s_{\min}(\mathcal{H})$, is bounded by*

$$s_{\min}(\mathcal{H}) \geq d_{\min}(C)^{t/(t-1)}.$$

*Proof.* Let $\mathcal{H}$ be as above, and let $S$ be a minimum stopping set. Each edge in $S$ contains exactly one constraint node from each of the $t$ parts of $\mathcal{H}$, so each part of $\mathcal{H}$ has exactly $|S| = s_{\min}(\mathcal{H})$ incident edges belonging to $S$. Each constraint node contained in an edge in $S$ must be contained in at least $d_{\min}(C)$ edges in $S$. By the pigeonhole principle, the number of vertices in any part of $\mathcal{H}$ that are contained in some edge in $S$ is bounded above by $s_{\min}(\mathcal{H})/d_{\min}(C)$. Indeed, were there more than $s_{\min}(\mathcal{H})/d_{\min}(C)$ vertices incident to $S$ in a single part, some vertex must have fewer than $s_{\min}(\mathcal{H})/(s_{\min}(\mathcal{H})/d_{\min}(C)) = d_{\min}(C)$ incident edges from $S$, a contradiction. Now consider the maximum size of $S$: this amounts to counting the number of edges possible, given that each edge is incident to exactly one vertex of (at most) $s_{\min}(\mathcal{H})/d_{\min}(C)$ vertices in each of the $t$ parts of $\mathcal{H}$. That is, there are at most $(s_{\min}(\mathcal{H})/d_{\min}(C))^t$ edges in $S$. Thus,

$$\left(\frac{s_{\min}(\mathcal{H})}{d_{\min}(C)}\right)^t \geq s_{\min}(\mathcal{H}) \Rightarrow s_{\min}(\mathcal{H}) \geq d_{\min}(C)^{t/(t-1)}.$$

□

The bound of Theorem 4.2.4 is tight. For example, when $\mathcal{H}$ is a complete 3-uniform 3-partite hypergraph with at least two vertices in each part and constraint code $C$ such that $d_{\min}(C) = 4$, it is easy to show that $s_{\min}(\mathcal{H}) = 8$.

Since the errors of particular relevance to DSS are erasures (such as a server going down), we can use the stopping set bound to characterize how many errors can be corrected. Theorem 4.2.4 guarantees that we may correct any $d_{\min}(C)^{t/(t-1)} - 1$ erasures using iterative decoding. If $C$ is a code with locality $r_1$, at most $(s_{\min}(\mathcal{H})/d_{\min}(C)) \cdot r_1 \cdot t$ other codeword symbols are involved in the repair of the erasures in the decoding process. This yields the following:

**Corollary 4.2.5.** *If the subcodes $C$ of the regular hypergraph code $\mathcal{Z}$ have $r_1$ locality, then*

*$\mathcal{Z}$ has $(r,\ell)$-cooperative locality where*

$$r = r_1 t s_{\min}(\mathcal{H})/d_{\min}(C)$$

$$s_{\min}(\mathcal{H}) - 1 \geq \ell \geq d_{\min}(C)^{t/(t-1)} - 1.$$

Observe that if the subcode $C$ has $(r,\tau)$-availability, then the hypergraph code $\mathcal{Z}$ has at least $(r,\tau)$-availability. We now extend the result to codes on hypergraphs with known $\epsilon$-homogeneity.

**Theorem 4.2.6.** *Let $\mathcal{H} = (V_1, V_2, \ldots, V_t; E)$ be a $t$-uniform $t$-partite $\Delta$-regular $\epsilon$-homogeneous hypergraph where there are $n$ vertices in each of the $t$ parts. If the subcodes $C$ have minimum distance $d_{\min}(C)$,*

$$s_{\min}(\mathcal{H}) \geq \left(\left(1 - \frac{\epsilon\Delta}{d_{\min}(C)}\right)\frac{n^{t-1}d_{\min}(C)^t}{\Delta}\right)^{1/(t-1)}.$$

*For $\epsilon < \frac{d_{\min}(C)(n^{t-1}-\Delta)}{\Delta n^{t-1}}$, this gives an improvement on the bound in Theorem 4.2.4.*

*Proof.* Let $S$ be a minimum stopping set. By Theorem 4.2.4, $s_{\min}(\mathcal{H}) \geq d_{\min}(C)^{t/(t-1)}$. Now, let $A_i \subseteq V_i$ be the set of vertices in $V_i$, for $i \in [t]$, contained in an edge in $S$. By $\epsilon$-homogeneity,

$$s_{\min}(\mathcal{H}) = |S| \leq |E(A_1, \ldots, A_t)| \leq n\Delta\left(\prod_{i=1}^{t} \alpha_i + \epsilon\sqrt{\alpha_{\sigma(1)}\alpha_{\sigma(2)}}\right).$$

Since $|A_i| \leq s_{\min}(\mathcal{H})/d_{\min}(C)$ for all $i$, $\alpha_i \leq s_{\min}(\mathcal{H})/nd_{\min}(C)$. Thus, the above inequality simplifies to obtain the result:

$$s_{\min}(\mathcal{H}) \leq n\Delta\left(\left(\frac{s_{\min}(\mathcal{H})}{nd_{\min}(C)}\right)^t + \epsilon\frac{s_{\min}(\mathcal{H})}{nd_{\min}(C)}\right).$$

Observe that we have shown in general that

$$s_{\min}(\mathcal{H}) \geq \left(\left(1 - \frac{\epsilon\Delta}{d_{\min}(C)}\right)\frac{n^{t-1}}{\Delta}\right)^{1/(t-1)} d_{\min}(C)^{t/(t-1)}.$$

Then, if $\left(\left(1 - \frac{\epsilon\Delta}{d_{\min}(C)}\right)\frac{n^{t-1}}{\Delta}\right)^{1/(t-1)} > 1$, this gives a better lower bound for $s_{\min}(\mathcal{H})$ than that found in Theorem 4.2.4. Simplifying, we have our condition on $\epsilon$. □

**Corollary 4.2.7.** *Using iterative decoding on a code $\mathcal{Z}$ based on a t-uniform t-partite $\Delta$-regular $\epsilon$-homogeneous hypergraph with vertices representing constraints of a subcode C, up to*

$$\left(\left(1 - \frac{\epsilon\Delta}{d_{\min}(C)}\right)\frac{n^{t-1}d_{\min}(C)^t}{\Delta}\right)^{1/(t-1)} - 1$$

*erasures may be corrected.*

In other words, if $\delta$ is the relative minimum distance of $C$, and $N$ is the total number of edges in the hypergraph (that is, the block length of the code $\mathcal{Z}$), we may correct up to a $\delta(\delta - \epsilon)^{1/(t-1)} - \frac{1}{N}$ fraction of erasures.

**Remark 4.2.8.** *We may correct up to a $\delta^{t/(t-1)} - \frac{1}{N} - c(\epsilon, \delta, t)$ fraction of erasures, where $c(\epsilon, \delta, t) \to 0$ as $\epsilon \to 0$. It can be shown that the bound in Corollary 4.2.7 improves the error correction capability of*

$$\binom{t-1}{t/2}^{-2/t}\left(\frac{\delta}{2}\right)^{(t+2)/t} - c'(\epsilon, \delta, t)$$

*in [10] for any $0 < \delta < 1$ and $t \geq 2$ (i.e. for all relevant cases) and large block length. Note that $c'(\epsilon, \delta, t) \neq c(\epsilon, \delta, t)$, but that both vanish as $\epsilon \to 0$. It is important to note that we are focusing solely on erasures, while [10] gives a decoding algorithm and correction capabilities for more general errors.*

### 4.2.2 Algebraic lifts of hypergraphs

In this section, we present a lifting process for hypergraphs that will allow us to create sequences of hypergraphs with larger block length but some preserved properties. In order to be a topological covering graph, we would like our definition to satisfy the following:

**Definition 4.2.9.** *Let $\mathcal{H}$ and $\tilde{\mathcal{H}}$ be hypergraphs. $\tilde{\mathcal{H}}$ is a* degree $J$ lift *or* cover *of $\mathcal{H}$ if there exists $\phi : V(\tilde{\mathcal{H}}) \to V(\mathcal{H})$ with the following properties:*

1. *the fiber of each $v \in V(\mathcal{H})$ has exactly $J$ elements;*

2. *for each $u \in V(\tilde{\mathcal{H}})$, the restriction of $\phi$ to $N_{\tilde{\mathcal{H}}}(u)$ is a bijection onto $N_{\mathcal{H}}(\phi(u))$.*

*In this case, $\phi$ is called a* covering map.

Let $S_J^p$ denote a $p$-tuple of elements from $S_J$. Then, we may construct a topological lift of a hypergraph as follows:

**Definition 4.2.10.** *Let $\mathcal{H}$ be a hypergraph, and let $v_1, \ldots, v_n$ be some ordering of the vertices of $\mathcal{H}$. Label each edge $E$ of $\mathcal{H}$ with an element of $S_J^{t-1}$, where $t$ is the number of vertices in $E$, and $J \geq 2$ is a fixed integer. Then the* degree $J$ lift*, $\tilde{\mathcal{H}}$, of $\mathcal{H}$ corresponding to the above vertex ordering and edge labeling, is the hypergraph with vertex set*

$$\bigcup_{i=1}^{n} \{v_{i,1}, \ldots, v_{i,J}\},$$

*and $J$ edges, $E_{j,1}, \ldots, E_{j,J}$, for each edge $E_j$ of $\mathcal{H}$. If $E_j$ contains vertices $v_{i_1}, \ldots, v_{i_t}$, where $i_1 < i_2 < \cdots < i_t$, and has label $(\sigma_1, \ldots, \sigma_{t-1}) \in S_J^{t-1}$, then*

$$E_{j,k} = \{v_{i_1,k}, v_{i_2,\sigma_1(k)} \ldots, v_{i_t,\sigma_{t-1}\sigma_t \cdots \sigma_1(k)}\}.$$

**Example 4.2.11.** *Consider the hypergraph with five vertices ($v_1, v_2, v_3$, $v_4$, and $v_5$, in order) and three edges – two of size three and one of size two – as shown in Figure 4.2. Notice that*

*$E_1$, one of the three-vertex edges, is labeled $(\sigma_1, \sigma_2)$. Let $\sigma_1, \sigma_2 \in S_3$ be the permutations $(1\ 2\ 3)$ and $(1\ 3)$, respectively. The two-vertex edge, $E_2$, is labeled $(\sigma_3)$. Let $\sigma_3 \in S_3$ be the permutation $(2\ 3)$. Lastly, $E_3$ is labeled $(\sigma_4, \sigma_5)$. Let $\sigma_4 = (1\ 3\ 2)$ and let $\sigma_5$ be the identity permutation in $S_3$. In the corresponding degree 3 lift of this hypergraph, there are 15 vertices and 9 edges: vertices $v_{i,1}, v_{i,2}$ and $v_{i,3}$ that cover vertex $v_i$ in the base, and three edges in the lift that cover each edge of the base. For example, edge $E_1$ in the base graph is covered by*

$$E_{1,1} = (v_{1,1}, v_{2,\sigma_1(1)}, v_{5,\sigma_2\sigma_1(1)}) = (v_{1,1}, v_{2,2}, v_{5,2}),$$

$$E_{1,2} = (v_{1,2}, v_{2,\sigma_1(2)}, v_{5,\sigma_2\sigma_1(2)}) = (v_{1,2}, v_{2,3}, v_{5,1}),$$

$$E_{1,3} = (v_{1,3}, v_{2,\sigma_1(3)}, v_{5,\sigma_2\sigma_1(3)}) = (v_{1,3}, v_{2,1}, v_{5,3}).$$

*The resulting degree 3 hypergraph lift is shown in Figure 4.3.*

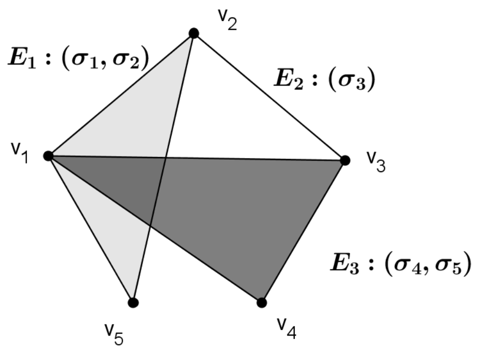


**Figure 4.2:** Base hypergraph with edge labels for lifting.

In [10], the authors present a method of constructing $t$-uniform $t$-partite $\Delta$-regular hypergraphs from regular Ramanujan graphs. Roughly, this is accomplished by letting paths of length $t - 1$ in a Ramanujan graph $G$ correspond to edges in a hypergraph. More specifically, if the regular Ramanujan graph $G$ is bipartite, half of the $t$ parts of the hypergraph

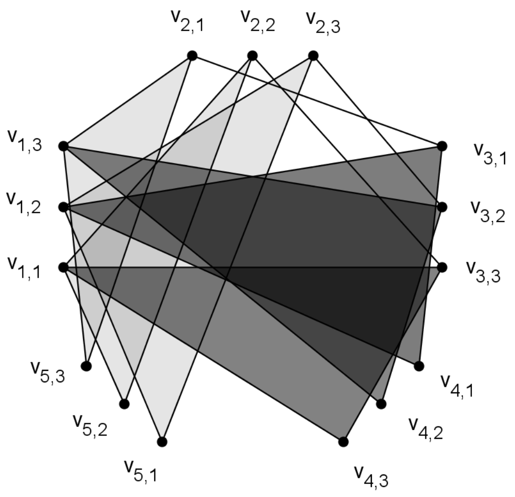


**Figure 4.3:** Hypergraph from a degree 3 lift of the base hypergraph.

are copies of the vertices in one part of $G$, and half are copies of the vertices in the other part. An edge in the hypergraph contains copies of the $t$ (not necessarily distinct) vertices in a walk of length $t-1$ in $G$. With this construction, the authors are able to show that the resulting hypergraph is $2(t-1)\lambda$-homogeneous, where $\lambda$ is the second largest eigenvalue (in absolute value) of the normalized adjacency matrix of $G$.

In [77], the existence of a sequence of $d$-regular bipartite Ramanujan graphs is shown for all $d \geq 3$ by proving that every $d$-regular graph has a 2-lift in which all of the new eigenvalues of the lifted graph are at most $2\sqrt{d-1}$, and recalling that bipartite graphs have eigenvalues symmetric about zero.

Using hypergraph lifts and the results of [10], we can show that there is a corresponding sequence of $\left(4(t-1)\sqrt{d-1}\right)$-homogeneous hypergraphs that can be viewed as resulting from 2-lifts of a hypergraph:

Let $G_1, G_2, \ldots$ be a sequence of $d$-regular bipartite Ramanujan graphs. Fix an even integer $t \geq 2$, and let $\mathcal{H}_i = \varphi(G_i)$ be the hypergraph constructed from $G_i$ as in [10]. We claim that there exists a hypergraph lift of degree 2 of $\mathcal{H}_i$ which gives $\mathcal{H}_{i+1} = \varphi(G_{i+1})$ for each $i \geq 1$. That is, there exists an ordering of vertices and an edge labeling of $\mathcal{H}_i$ such that the corresponding 2-lift is the hypergraph $\mathcal{H}_{i+1}$. This will give an infinite sequence of $\left(4(t-1)\sqrt{d-1}\right)$-homogeneous hypergraphs.

Without loss of generality, assign an ordering to the $t$ parts of $\mathcal{H}_i$. Each edge $\{v_1, \ldots, v_t\}$ in $\mathcal{H}_i$ corresponds to a path of length $t-1$ in $G_i$. For ease of notation, let $v_j \in V(G_i)$ denote the preimage of the vertex $v_j \in V(\mathcal{H}_i)$. Notice that the $v_j$'s may not be distinct in $G_i$. To the edge $\{v_1, \ldots, v_t\}$ in $\mathcal{H}_i$, assign the $(t-1)$-tuple $(\sigma_1, \ldots, \sigma_{t-1})$, where $\sigma_1$ is the permutation assignment to the edge $v_1 v_2$ in $G_i$ for the lift to $G_{i+1}$, $\sigma_2$ the permutation assignment to the edge $v_2 v_3$ in $G_i$, and so on. Since the lift from $G_i$ to $G_{i+1}$ is a 2-lift, the direction in which an edge is traveled does not matter. We claim that the corresponding lift of $\mathcal{H}_i$, which we will call $\tilde{\mathcal{H}}_i$ for now, is in fact $\mathcal{H}_{i+1}$.

Indeed, it is readily apparent that the number of vertices in $\tilde{\mathcal{H}}_i$ and $\mathcal{H}_{i+1}$ are equal: if $G_i$ has $n$ vertices, then $\mathcal{H}_i$ has $\frac{t}{2} \cdot n$ vertices, and $\tilde{\mathcal{H}}_i$, as a degree 2 lift, has $t \cdot n$ vertices. Similarly, $G_{i+1}$ has $2n$ vertices, so $\mathcal{H}_{i+1}$ has $\frac{t}{2} \cdot 2n = t \cdot n$ vertices.

Furthermore, the edge sets of the two hypergraphs are equal. Suppose first that $E$ is an edge in $\tilde{\mathcal{H}}_i$. Then $E$ has the form $\{v_{i_1,j}, v_{i_2,\sigma_1(j)}, \ldots, v_{i_t,\sigma_{t-1}\cdots\sigma_1(j)}\}$, where $j \in \{1,2\}$, $\sigma_k \in S_2$ for all $1 \leq k \leq t-1$, and $(v_{i_1}, \ldots, v_{i_t})$ is a walk in $G_i$. By the assignment of edge labels in $\mathcal{H}_i$, $(v_{i_1,j}, v_{i_2,\sigma_1(j)}, \ldots, v_{i_1,\sigma_t\cdots\sigma_1(j)})$ is a walk in $G_{i+1}$ for $j \in \{1,2\}$, and so $E$ is also an edge in $\mathcal{H}_{i+1}$.

On the other hand, suppose $E = \{w_1, \ldots, w_t\}$ is an edge in $\mathcal{H}_{i+1}$. Then $(w_1, \ldots, w_t)$ is a walk in $G_{i+1}$, and so must have the form $(v_{i_1,j}, v_{i_2,\sigma_1(j)}, \ldots, v_{i_1,\sigma_t\cdots\sigma_{t-1}(j)})$ for some $v_{i_k}$'s forming a walk in $G_i$ and $j \in \{1,2\}$, where the $\sigma$'s are permutation assignments to the edges in the corresponding walk $(v_{i_1}, v_{i_2}, \ldots, v_{i_t})$ in the base graph $G_i$. Then, $\{v_{i_1}, v_{i_2}, \ldots, v_{i_t}\}$ is an edge in $\mathcal{H}_i$, and so $\{v_{i_1,j}, v_{i_2,\sigma_1(j)}, \ldots, v_{i_t,\sigma_{t-1}\cdots\sigma_1(j)}\}$ is an edge in $\tilde{\mathcal{H}}_i$, by our definition of the lift. We conclude that $\mathcal{H}_{i+1} = \tilde{\mathcal{H}}_i$.

The existence of infinite families of $(c,d)$-biregular bipartite Ramanujan graphs for $c, d \geq 3$ was also shown in [77] using 2-lifts of graphs. We conjecture that we may construct

$$\begin{tikzcd} G_{i+1} & \mathcal{H}_{i+1} \\ G_i \arrow[u, ""{name=A}] & \mathcal{H}_i \arrow[u, dashed, ""'{name=B}] \arrow[from=A, to=B, mapsto, "\varphi"] \end{tikzcd}$$

**Figure 4.4:** Given a 2-lift from $G_i$ to $G_{i+1}$, $\varphi$ induces a 2-lift from $\mathcal{H}_i$ to $\mathcal{H}_{i+1}$, letting $\varphi$ denote the construction of [10].

biregular hypergraphs from biregular bipartite graphs in a way that will allow us to state the $\epsilon$-homogeneity of the hypergraph based on the eigenvalues of the bipartite graph. If this can be shown, then we can construct sequences of biregular 2-lifted $\epsilon$-homogeneous graphs in an analogous way.

# Chapter 5

# Multidimensional Decoding Networks

In this chapter, we present a multidimensional network model for analyzing hard-decision message-passing decoders. The structure of this network depends not only on the code, but also the choice of Tanner graph representation and decoder. Thus, our model takes into account all parameters determining the presence of harmful trapping sets. We show how decoding networks may be used to identify trapping sets, and therefore analyze decoder behavior of LDPC codes. This analysis is simplified for networks with a transitivity property, and so we discuss the connection between transitive networks and redundancy in their corresponding parity-check matrices. As applications, we relate the decoding networks of product and half-product codes and codes arising from a $(\mathbf{u} \mid \mathbf{v})$ construction to those of their underlying component codes, and examine the connection between the decoding networks of a protograph and its lift. Finally, we show how decoding networks can provide insight into the optimal number of iterations to run on a given code (and representation) with a chosen decoder. Taking advantage of this connection, we present an algorithm designed to improve the performance of hard-decision message-passing decoders.

## 5.1 Multidimensional network framework

Suppose we decode a code $C$ using a hard-decision decoder, and consider the labeled directed graph (digraph) for a fixed $\ell \in \mathbb{Z}_{>0}$, denoted $\mathcal{D}_\ell$, with vertex and edge sets $V = \{\mathbf{x} : \mathbf{x} \in S\}$ and $E = \{(\mathbf{x}_i, \mathbf{x}_j, \ell) : \mathbf{x}_i^\ell = \mathbf{x}_j\}$, respectively, where $S$ is the set of possible received words, $(\mathbf{x}_i, \mathbf{x}_j, \ell)$ denotes an edge from $\mathbf{x}_i$ to $\mathbf{x}_j$ with edge label $\ell$, and $\mathbf{x}_i^\ell$ is the output of the decoder after $\ell$ iterations with input $\mathbf{x}_i$. Note that we allow loops, which are edges of the form $(\mathbf{x}_i, \mathbf{x}_i, \ell)$. For simplicity, we refer to the label of a vertex – that is, its corresponding word in $S$ – interchangeably with the vertex itself. There will be a potentially distinct digraph on this same vertex set for each choice of $\ell \in \mathbb{Z}_{>0}$. We call the union of these digraphs for all $\ell \in \mathbb{Z}_{>0}$ the *(multidimensional) decoding network* corresponding to the code $C$ and the specific choice of decoder, as we may consider the digraph which incorporates the information for all $\ell$ as a *multidimensional network.*

**Definition 5.1.1.** *[78] A* multidimensional network *is an edge-labeled directed graph $\mathcal{D} = (V, E, D)$, where $V$ is a set of vertices, $D$ a set of edge labels, called* dimensions*, and $E$ is a set of triples $(u, v, d)$ where $u, v \in V$ and $d \in D$. We say that an edge (or vertex)* belongs to *a given dimension $d$ if it is labeled $d$ (or is incident to an edge labeled $d$).*

In this framework, the decoding network is a multidimensional network with $D = \mathbb{Z}_{>0}$, and each edge labeled with the number of decoder iterations, $\ell$, to which it corresponds. Notice that, in any given dimension (i.e. number of iterations), every vertex has outdegree equal to one.

**Example 5.1.2.** *As a small example, consider the* $0^{th}$ *order binary Reed-Muller code $\mathscr{R}(0, 2)$ of length $2^2$, defined by the parity-check matrix*

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

*transmitted over the BSC and decoded with the Gallager A algorithm. Notice that* $\mathscr{R}(0,2)$ *is the binary repetition code of length 4. Figure 5.1 shows the first two dimensions of the corresponding decoding network.*

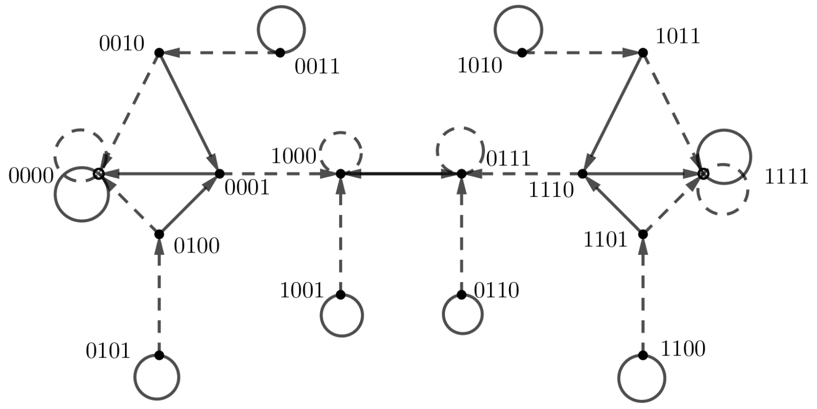


**Figure 5.1:** Dimensions 1 and 2 of the decoding network of $\mathscr{R}(0,2)$, with the edges belonging to $\mathcal{D}_1$ shown in solid and those of $\mathcal{D}_2$ dashed.

An important type of decoding network is given by codes for which running $i$ iterations of the decoder on a received word, and using that output as input to a subsequent $j$ decoding iterations, is equivalent to running $i + j$ iterations on the originally-received word.

**Definition 5.1.3.** *We say that a decoding network is* transitive *if* $(v_1, v_2, \ell) \in E$ *if and only if for every choice of* $1 \leq k \leq \ell - 1$*, there exists* $v_k \in V$ *such that* $(v_1, v_k, k), (v_k, v_2, \ell - k) \in E$*. We say a decoder is transitive for a code C and a choice of representation of C if its resulting decoding network is transitive.*

Let $\mathcal{D}_\ell$ denote the digraph corresponding to the $\ell$th dimension of the decoding network $\mathcal{D}$, and let $A(\mathcal{D}_\ell)$ denote the adjacency matrix of the digraph $\mathcal{D}_\ell$. Observe that a decoding network $\mathcal{D}$ is transitive if and only if $A(\mathcal{D}_\ell) = (A(\mathcal{D}_1))^\ell$ for all $\ell \geq 1$, as the product $(A(\mathcal{D}_1))^\ell$ gives directed paths of length $\ell$ in dimension 1 of $\mathcal{D}$.

**Example 5.1.4.** *Consider a simple binary parity-check code of length $n$, with parity-check matrix given by the $1 \times n$ all-ones matrix. The Tanner graph representation of such a code is a single check node with $n$ variable node leaves. Thus, if codewords are sent over the BSC and decoded with the Gallager A algorithm, the message the solitary check node sends to each of its adjacent variable nodes is either (a) the node's channel value, if a codeword is received, or (b) the opposite of its originally-received value, otherwise. Each variable node will always send back its channel value. For any number of iterations, codewords will decode to codewords, and any received non-codeword $\mathbf{y}$ will be decoded to $\mathbf{y} + \mathbf{1}$ (mod $2$). If $n$ is odd, every received word will decode to a codeword, and the network will be transitive. If $n$ is even, $\mathbf{y} + \mathbf{1}$ will not be a codeword, and the network will not be transitive. The cases $n = 3$ and $n = 4$ are shown in Figure 5.2; in both networks, edges belonging to higher dimensions are suppressed, as all dimensions are identical.*

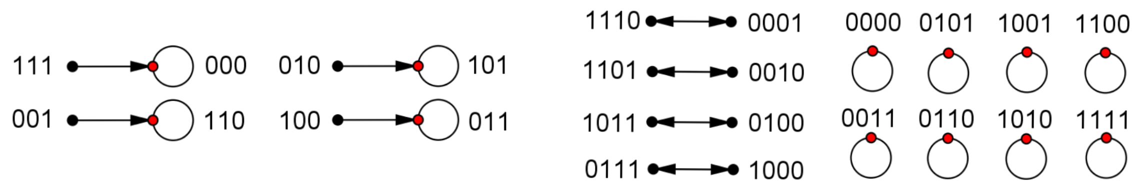
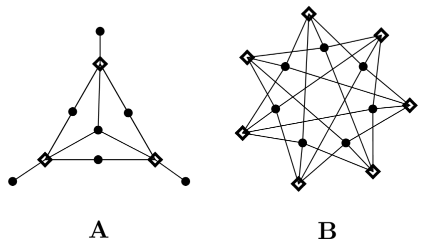


**Figure 5.2:** The decoding networks of parity-check codes of lengths 3 (left) and 4 (right).

**Example 5.1.5.** *Consider the binary Hamming code of length $7 = 2^3 - 1$, denoted $\mathscr{H}_3$. This code's canonical $3 \times 7$ parity-check matrix has columns consisting of all nonzero binary words of length $3$. The corresponding Tanner graph may be seen in representation A of Figure 5.3. However, $\mathscr{H}_3$ may also be defined by the parity-check matrix whose Tanner graph is representation B in Figure 5.3. Under Gallager A decoding, representation A does not yield a transitive decoding network. However, if representation B is decoded via*

$$H_A = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$



$$H_B = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 5.3:** Two distinct Tanner graph representations of $\mathscr{H}_3$, along with their parity-check matrices. Variable nodes are denoted by $\bullet$, and check nodes are denoted by $\diamond$.

*Gallager A, the resulting decoding network is transitive: every word decodes under a single iteration to a codeword, and decodes identically for any higher number of iterations.*

If a decoder is transitive for all representations of all codes $C$, we say that it is *universally transitive*. Any decoder which ignores channel values at each subsequent iteration will be universally transitive; some examples of this are given below.

**Example 5.1.6.** *If codewords from a code $C$ are sent over the BEC, and words are decoded using a peeling decoder which iteratively corrects erasures, then the corresponding decoding network of $C$ is universally transitive. Indeed, corrections at each iteration are performed regardless of previous iterations. Similarly, iterative bit-flipping decoders over the BSC are universally transitive.*

### 5.1.1 Trapping set characterization

Within the decoding network framework, trapping sets of a code transmitted over the BSC may be determined by looking at the supports of the words corresponding to vertices in the decoding network that have nonzero indegree in an infinite number of dimensions. That is,

**Theorem 5.1.7.** *For each vertex $\mathbf{x} \in V = \mathbb{F}_2^n$ in a decoding network $\mathcal{D} = (V, E, D)$ for a code $C$, let $M_{\mathbf{x}}$ be the set of vertices $\mathbf{y} \in V$ for which there is an edge $(\mathbf{x}, \mathbf{y}, \ell) \in E$ for* *infinitely many choices of $\ell$. Then the set of variable nodes corresponding to*

$$\bigcup_{\mathbf{y}\in M_{\mathbf{x}}} \mathrm{supp}(\mathbf{y}) \ ,$$

*denoted $\mathcal{T}(\mathbf{x})$, is a trapping set with an inducing set given by the variable nodes corresponding to* $\mathrm{supp}(\mathbf{x})$*. Furthermore, the set of trapping sets of the code $C$ is*

$$\{\mathcal{T}(\mathbf{x}) : \mathbf{x} \in \mathbb{F}_2^n\} \ ,$$

*and, given a trapping set $\mathcal{T}$, its set of inducing sets is given by the variable nodes corresponding to sets of the form*

$$\{\mathrm{supp}(\mathbf{x}) : \mathcal{T}(\mathbf{x}) = \mathcal{T}\} \ ,$$

*and its critical number is*

$$m(\mathcal{T}) = \min\{ \, |\mathrm{supp}(\mathbf{x})| : \mathcal{T}(\mathbf{x}) = \mathcal{T}\} \ .$$

*Proof.* Assuming that the all-zero codeword was sent over the BSC, any decoding errors will be given by 1's. If, during the process of decoding the received word $\mathbf{x}$, there is some word $\mathbf{y}$ such that an edge from $\mathbf{x}$ to $\mathbf{y}$ occurs in an infinite number of network dimensions, the support of $\mathbf{y}$ gives variable node positions which are not eventually correct. By definition, these variable nodes belong to a trapping set induced by the variable nodes of the support of $\mathbf{x}$. However, these may not be the only variable nodes that are not eventually correct given the received word $\mathbf{x}$. Taking the union of the supports of all such $\mathbf{y}$ gives us our expression for $\mathcal{T}(\mathbf{x})$, the trapping set induced by $\mathbf{x}$. Repeating this for each possible received word, we find all trapping sets of the code. Note that each trapping set may be induced by multiple received words. □

**Example 5.1.8.** *Recalling the decoding network of $\mathscr{R}(0,2)$ in Figure 5.1, consider the received word* 0011*. As seen in the first two dimensions,* 0011 *is decoded as* 0011 *in 1 iteration of Gallager A, and as* 0010 *in 2 iterations of Gallager A. In three iterations,* 0011 *will decode as* 1110*. For any higher number of iterations, the decoder's output will be one of these three words. That is, the vertices* 0011*,* 0010*, and* 1110 *form the set $M_{0011}$ defined in Theorem 5.1.7. Since the union of the supports of these words is all four bits, the set $\{v_1, v_2, v_3, v_4\}$ forms a $(4,2)$-trapping set induced by $\{v_3, v_4\}$.*

**Example 5.1.9.** *Let $C$ be the binary repetition code of length* 3*, with the parity-check matrix*

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

*The associated Tanner graph is a path of length* 4 *with variable nodes $v_1, v_2$, and $v_3$. Let $\mathcal{D}$ be the (non-transitive) decoding network of $C$ under Gallager A, shown in Figure 5.4. Assuming* **0** *was transmitted, the received word* 011*, for example, decodes in one iteration to the codeword* 111*, but decodes for any number of iterations greater than* 1 *to the (non-code)word* 110*. Thus, $\{v_1, v_2\}$ is a (2,1)-trapping set. Note that the support of* 011*, which induces this trapping set, is not contained in the trapping set. Similarly, $\{v_1, v_2, v_3\}$ is a trapping set corresponding to the codeword 111, with inducing sets $\{v_1, v_3\}$ and $\{v_1, v_2, v_3\}$. Other trapping sets of the code include $\{v_2, v_3\}$ (induced by $\{v_1, v_2\}$), $\{v_1\}$ (induced by $\{v_3\}$), and $\{v_3\}$ (induced by $\{v_1\}$).*

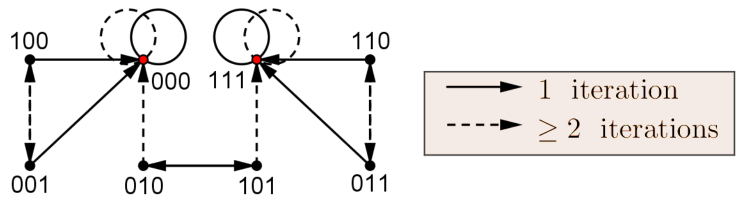


**Figure 5.4:** The decoding network of a binary repetition code of length 3.

In the case of transitive decoding networks, trapping sets may be identified by looking only at dimension 1, as follows:

**Corollary 5.1.10.** *If the decoding network, $\mathcal{D}$, of a code $C$ is transitive, then the trapping sets are given by*

*(1) the sets of variable nodes corresponding to supports of vertices with loops in $\mathcal{D}_1$, and*

*(2) the sets of variable nodes corresponding to unions of the supports of vertices forming directed cycles in $\mathcal{D}_1$.*

*Furthermore, inducing sets of trapping sets in a transitive decoding network are given by the variable nodes corresponding to the support of any vertex which has a directed path to either a (nonzero) vertex with a loop, or to a directed cycle, regardless of where that path enters the cycle.*

*Proof.* In a transitive decoding network, the edges in dimension $\ell$ correspond to directed paths of length $\ell$ in $\mathcal{D}_1$. Thus, in order for a word to appear as the output of the decoder in an infinite number of dimensions, it must be the terminating vertex of infinitely many directed paths (of distinct lengths) from the received word. Because the decoding network for a code is finite, and the outdegree of every vertex in $\mathcal{D}_1$ is equal to 1, this can only occur if there is a loop at that vertex, or if it belongs to a directed cycle. The result follows from Theorem 5.1.7. $\square$

In the adjacency matrix of dimension 1 of a decoding network, nonzero diagonal entries indicate loops. There are numerous algorithms for finding directed cycles in a digraph, such as Depth-First Search (DFS) [79]. We further note that several works have addressed the computational aspects of finding and/or removing trapping sets [12, 36, 80, 81].

## 5.2 Representations yielding transitivity

Due to the effect of the choice of representation on a decoding network's structure, it is natural to ask which representations, if any, ensure that a code's decoding network is transitive under a fixed decoder. Recall from Example 5.1.5 that the canonical representation of the Hamming code $\mathscr{H}_3$ does not yield a transitive decoding network under Gallager A, while the decoding network arising from the representation given by the parity-check matrix including all nonzero codewords of the dual code, $\mathscr{H}_3^{\perp}$, is transitive. In fact,

**Theorem 5.2.1.** *Every $[n,k]$ binary linear code with $n-k > 2$ has a parity-check matrix representation whose corresponding decoding network is transitive under Gallager A decoding.*

In particular, adding exactly the nonzero codewords of the dual to the parity-check matrix of such a code (a representation which we will refer to as the *complete representation*) will result in a transitive decoding network under Gallager A decoding: using symmetries of the complete representation, we may show that after a single iteration of the decoder, the received word either decodes to a codeword and continues decoding to that codeword for any number of iterations, or it will decode to itself under any number of iterations. We defer the full proof of Theorem 5.2.1 until after the proofs of three necessary lemmas.

In the following three lemmas, suppose we have an $[n,k]$ binary linear code $C$ with $n-k > 2$. Let $H$ be the parity-check matrix of $C$ whose rows are exactly the nonzero codewords of $C^{\perp}$. Let $G$ be the corresponding Tanner graph representation of $C$.

**Lemma 5.2.2.** *There are $2^{n-k}-1$ constraint nodes in $G$, every variable node has degree $2^{n-k-1}$, and every set of $i$ variable nodes for $i \leq n-k$ has $2^{n-k-i}$ common neighbors.*

*Proof.* The first statement is clear from the construction of $H$ and the fact that $\dim(C^{\perp}) = n-k$. The second statement follows from the fact that exactly half of the codewords of a binary linear code have a 1 in a given coordinate.

We note that the final statement is equivalent to the following: for any choice of $i \leq n-k$ columns in $H$, there are exactly $2^{n-k-i}$ rows in $H$ with a 1 in all $i$ chosen columns.

Choose a subset $I \subseteq [n]$ with $|I| = i$, corresponding to a set of $i$ columns of $H$, or coordinates of words in $C^\perp$. Without loss of generality, we may assume that $I$ corresponds to $\{1, 2, \ldots, i\}$, the first $i$ coordinates or columns. Note that the set of all codewords in $C^\perp$ with 0's in the first $i$ coordinates form a subspace of $C^\perp$; call this subspace $C_I^\perp$. Furthermore, we claim that the dimension of this subspace is $n - k - i$.

Indeed, if $i = 1$, this is equivalent to saying that half of the codewords of a binary linear code have a 0 in the first coordinate, which we know to be true. Suppose the statement holds true for $i = m < n - k$. That is, the set of codewords in $C^\perp$ with a 0 in the first $m$ coordinates is a subcode of dimension $n - k - m$. Half of these codewords have a 0 in the $(m + 1)$st coordinate, giving a subspace of $C^\perp$ of dimension $n - k - (m + 1)$. We conclude that the statement holds for all $i \leq n - k$.

Next, observe that $C^\perp/C_I^\perp$ has dimension $n - k - (n - k - i) = i$. Each of the $2^i$ cosets of $C_I^\perp$ corresponds to a choice of a subset of $I$, seen by mapping a representative of the coset to the subset of its first $i$ coordinates which have an entry of 1 (it is straightforward to see that any choice of representative will map to the same subset). Furthermore, no two distinct cosets will map to the same subset of $I$. Since there are $2^i$ subsets of $I$, this map gives a bijection. Then, it must be the case that some coset corresponds to the entire set $I$. In other words, the set of rows in $H$ with 1's in every column corresponding to an element of $I$ forms a coset of $C_I^\perp$. We conclude that there are $|C_I^\perp| = 2^{n-k-i}$ rows in $H$ with 1's in all of the columns corresponding to elements of $I$, and we have proven our claim. □

**Lemma 5.2.3.** *Let $C$ be decoded via Gallager A or B decoding. During the first iteration, there are either zero or exactly $2^{n-k-1}$ unsatisfied constraint nodes in $G$.*

*Proof.* Let $\mathbf{y} \in \mathbb{F}_2^n$ be the input to the decoder. Observe that the number of unsatisfied constraint nodes in $G$ during the first decoding iteration is exactly the number of codewords in $C^\perp$ whose inner product with $\mathbf{y}$ is equal to 1. We claim that there are exactly $2^{n-k-1}$ such codewords in the dual.

If $\langle \mathbf{c}, \mathbf{y} \rangle = 0$ for all $\mathbf{c} \in C^\perp$, then $\mathbf{y}$ is a codeword in $C$, and no constraint nodes are unsatisfied. Otherwise, there exists $\tilde{\mathbf{c}} \in C^\perp$ such that $\langle \tilde{\mathbf{c}}, \mathbf{y} \rangle = 1$. Consider the map

$$\begin{aligned} \varphi : C^\perp &\to C^\perp \\ \mathbf{c} &\mapsto \mathbf{c} + \tilde{\mathbf{c}} \end{aligned}$$

This map is clearly bijective. Notice that $\langle \varphi(\mathbf{c}), \mathbf{y} \rangle = \langle \mathbf{c}, \mathbf{y} \rangle + 1$ for all $\mathbf{c}$. In particular, each word in the dual whose inner product with $\mathbf{y}$ is equal to 1 maps to a word whose inner product with $\mathbf{y}$ is equal to 0, and vice versa. Since $\varphi$ is bijective, this implies that exactly half of the elements of $C^\perp$ have an inner product with $\mathbf{y}$ equal to 1. That is, there are

$$|C^\perp|/2 = 2^{n-k}/2 = 2^{n-k-1}$$

nonzero codewords in the dual whose inner product with $\mathbf{y}$ is equal to 1. In other words, there are $2^{n-k-1}$ unsatisfied constraints in the first iteration of the decoder on $G$. □

**Lemma 5.2.4.** *Under Gallager A decoding, no variable node $v$ in $G$ ever has $d_G(v) - 1$ incoming messages disagreeing with the channel value at that variable node.*

*Proof.* Suppose the above does not hold, and let the end of iteration $i$ be the first time a variable node receives $d_G(v) - 1$ messages disagreeing with the channel value.

Let $v$ be a variable node receiving such a set of messages. Since $n - k > 2$, Lemma 5.2.2 implies that the degree of each variable node in $G$ is even, and in particular this implies that $v$ receives an odd number of each possible message value (0 or 1) at the end of iteration $i$.

That is, the sum of the messages coming into $v$ at the end of iteration $i$ must be 1. Let $G_v$ be the subgraph of $G$ induced by the vertices $N(v) \cup N(N(v))$. From the Gallager A check to variable message rule, the sum of the messages into $v$ at the end of iteration $i$ is also equal to

$$\sum_{v_j \neq v} d_{G_v}(v_j) \cdot m_{v_j}$$

where $m_{v_j}$ is the message from $v_j$ to all its check nodes during the $i$th iteration (the same message is sent along each edge here, since iteration $i$ is the first time a variable node receives all-but-1 unanimous channel-contradicting messages, which is the only case in which outgoing variable to check messages disagree with one another under Gallager A). Thus,

$$\sum_{v_j \neq v} d_{G_v}(v_j) \cdot m_{v_j} = 1.$$

However, by Lemma 5.2.2, since $n - k > 2$, each vertex in $V \setminus \{v\}$ has even degree in $G_v$ (in particular, degree $2^{n-k-2}$), a contradiction to the above summation. We conclude that it cannot be the case that a variable node ever receives $d_G(v) - 1$ of one message and one of the other. □

We now prove Theorem 5.2.1 by proving that the complete representation of such a code will result in a transitive decoding network under Gallager A.

*Proof of Theorem 5.2.1.* Let $C$ be an $[n, k]$ binary linear code with $n - k > 2$. Let $H$ be the parity-check matrix of $C$ whose rows are exactly the nonzero codewords of $C^{\perp}$. Let $G$ be the corresponding Tanner graph representation of $C$.

Let $\mathbf{y} \in \mathbb{F}_2^n$ be the input to the decoder. If there are zero unsatisfied constraint nodes during the first iteration of Gallager A, $\mathbf{y}$ is a codeword, and will decode as $\mathbf{y}$ for any number of iterations. Otherwise, there are $2^{n-k-1}$ unsatisfied constraint nodes in $G$ by Lemma 5.2.3.

In this second case, since two variable nodes cannot have exactly the same set of neighbors by Lemma 5.2.2, at most one variable node in $G$ has all of its neighbors unsatisfied at the end of the first iteration of the decoder, and so at most one variable node flips its value after one decoder iteration.

If a single variable node flips, then the unsatisfied constraints in the first iteration of the decoder were exactly its neighbors. Then, all of its adjacent constraints are satisfied on the next (and all future) iteration(s), as it will send out its flipped value along all incident edges. In other words, all constraint nodes are satisfied on the next iteration. We conclude that flipping that single variable node value resulted in a codeword. That is, $\mathbf{y}$ will decode as this codeword after any number of iterations, and the decoding network is transitive.

If no variable node flips on the first iteration, then, because no variable node receives more than $d_G(v) - 2$ messages conflicting with its channel value by Lemma 5.2.4, all messages sent out from variable nodes on the second (or any higher number of) iteration(s) will agree with their channel values. That is, $\mathbf{y}$ will decode as itself after any number of iterations, and the decoding network is, again, transitive. This completes our proof.

□

While the complete representation establishes the existence of a representation yielding a transitive network for any code with $n - k > 2$, this level of redundancy is not necessarily required, and in fact may create an excess of trapping sets in the code. In Example 5.1.5, adding row seven of representation B to the canonical representation A gives a transitive network, as does adding any three additional rows to the canonical representation. However, any other combination of row additions does not yield a transitive network. Thus, it is interesting to determine the minimum level of redundancy needed to yield a transitive decoding network for a code with a fixed choice of decoder.

### 5.2.1 Applications

In the remainder of this section, we apply the decoding network model to product and half-product codes, which have been subject to renewed interest [82–84], as well as to a $(\mathbf{u} \mid \mathbf{v})$ code construction and codes constructed from protographs. By phrasing the decoding networks of these classes of codes in terms of the decoding networks of their component codes, we can identify trapping sets in the larger codes with fewer up-front computations. Recall,

**Definition 5.2.5.** *Let $C_1$ and $C_2$ be binary linear codes of lengths $n$ and $m$, respectively. Then the* product code *$C_1 \times C_2$ is the set of $m \times n$ binary arrays such that each row forms a codeword in $C_1$, and each column forms a codeword in $C_2$.*

Consider a product code, $C_1 \times C_2$, with a decoder that operates by running iterations of a specified decoder on each component code in an alternating fashion. At each iteration, channel information is dispensed with and decoding is performed based solely on the current estimate.

**Theorem 5.2.6.** *Let $C_1$ and $C_2$ be codes of lengths $n$ and $m$, respectively, with decoding networks $\mathcal{D}^1$ and $\mathcal{D}^2$ with respect to some specified decoders. Let $A_1$ be the adjacency matrix of the directed graph product $(\mathcal{D}^1_1)^m$, and let $A_2$ be the adjacency matrix of $(\mathcal{D}^2_1)^n$. Then, the adjacency matrix of dimension $\ell$ of the transitive decoding network, $\mathcal{D}$, of the product code $C_1 \times C_2$ is given by $(A_1 A_2)^\ell$.*

*Proof.* Consider the product code $C_1 \times C_2$, where $C_1$ has length $n$, and $C_2$ has length $m$. Let $\mathcal{D}^1$ (resp. $\mathcal{D}^2$) be the decoding network of $C_1$ (resp., $C_2$) under a hard-decision message-passing decoding algorithm. Since $C_1 \times C_2 \subseteq \mathbb{F}_2^m \times \mathbb{F}_2^n$, the size of the vertex set of the product code decoding network, $\mathcal{D}$, will be $|\mathbb{F}_2^m \times \mathbb{F}_2^n| = 2^{mn}$. We may view each word associated with a vertex as the product of $m$ words of length $n$, or, equivalently, $n$ words of length $m$.

For the first half of a single iteration, the rows of a word in $\mathbb{F}_2^m \times \mathbb{F}_2^n$ are decoded. Thus, for a half iteration, $(\mathbf{v}_1, \ldots, \mathbf{v}_m) \to (\mathbf{w}_1, \ldots, \mathbf{w}_m)$ if and only if $\mathbf{v}_i \to \mathbf{w}_i$ in $\mathcal{D}^1$ for each $1 \leq i \leq m$, where $\mathbf{v}_i$ and $\mathbf{w}_i$ correspond to words in $\mathbb{F}_2^n$. In particular, the decoding network for the first half iteration is the directed product graph $(\mathcal{D}_1^1)^m$; let $A_1$ be its adjacency matrix. On the next half of the first iteration, the columns of the word are decoded. By similar reasoning, the graph for the second half of the decoding process is given by the directed product graph $(\mathcal{D}_1^2)^n$, with adjacency matrix denoted $A_2$.

To construct the decoding network for a full iteration, we first identify the vertices of $(\mathcal{D}_1^1)^m$ and $(\mathcal{D}_1^2)^n$ as follows:

$$(\mathbf{v}_1, \ldots, \mathbf{v}_m) = ((v_{11}, v_{12}, \ldots, v_{1n}), \ldots, (v_{m1}, v_{m2}, \ldots, v_{mn}))$$

in the vertex set of $(\mathcal{D}_1^1)^m$ is identified with the vertex

$$(\mathbf{v}'_1, \ldots, \mathbf{v}'_n) = ((v_{11}, v_{21}, \ldots, v_{m1}), \ldots, (v_{1n}, v_{2n}, \ldots, v_{mn}))$$

in the vertex set of $(\mathcal{D}_1^2)^n$. Once the vertices have been identified so that the rows and columns of $A_1$ and $A_2$ correspond, the adjacency matrix for $\mathcal{D}_1$ is equal to $A_1A_2$, which gives directed paths of length two with the first edge in the path corresponding to the first half of an iteration, and the second edge corresponding to the second half of an iteration.

Following from the transitivity of the decoder, the adjacency matrix for dimension $\ell$ in the decoding network, $\mathcal{D}$, of $C_1 \times C_2$ is thus equal to $(A_1A_2)^\ell$.

$\square$

Next, we consider half-product codes, a class of codes constructed from product codes that offer improved performance in both the waterfall and error floor regions [83, 84].

**Definition 5.2.7.** *[83] Let $C$ be a binary linear code of length $n$, and let*

$$\tilde{C}_H = \{X - X^T : X \in C \times C\}.$$

*The* half-product code *with component code $C$, denoted $C_H$, is obtained from $\tilde{C}_H$ by setting the symbols below the diagonal of each element of $\tilde{C}_H$ equal to zero.*

A decoder runs by iteratively decoding at each of the $n$ constraints corresponding to "folded" codewords, as in [83, 84]. Again, channel information is dispensed with at each subsequent iteration. For each $i \in [n]$, let $A_i$ be the adjacency matrix of the digraph on the vertex set of the decoding network of the half-product code, $\mathcal{D}^H$, which gives the behavior of a single decoder iteration run on the $i$th constraint code. While decoding is performed on the $i$th constraint, all $(n-1)(n-2)/2$ symbols not participating in constraint $i$ are fixed. Let $\mathcal{D}^i$ be the decoding network associated with the $i$th constraint code. Then, $A_i$ is the adjacency matrix of a disjoint union of $2^{(n-1)(n-2)/2}$ copies of $\mathcal{D}^i$, corresponding to all the ways non-participating symbols may be fixed. Permute the rows and columns of the $A_i$'s so that they all correspond to a single ordering of the vertices in $\mathcal{D}^H$, and let $S_n$ denote the symmetric group on $n$ elements.

**Theorem 5.2.8.** *The product $(A_{\sigma(1)} \cdots A_{\sigma(n)})^\ell$, where $\sigma \in S_n$, gives the adjacency matrix of $\mathcal{D}^H_\ell$, dimension $\ell$ of the decoding network of the half-product code $C_H$, where the component constraints are decoded in the order determined by $\sigma$.*

*Proof.* The proof is similar to that of Theorem 5.2.6.

□

Another construction yielding a transitive decoding network is as follows.

**Theorem 5.2.9.** *Let $C_1$ and $C_2$ be binary linear codes with parity-check matrices $H_1$ and $H_2$, respectively. Suppose their decoding networks are transitive with respect to a fixed decoder and the above representations. Then, the decoding network of the code $C = \{(\mathbf{u} \mid \mathbf{v}) : \mathbf{u} \in C_1, \mathbf{v} \in C_2\}$ with parity-check matrix $H = H_1 \oplus H_2$ is transitive with respect to the same decoder.*

*Proof.* Notice that the Tanner graph of the code $C$ is the disjoint union of the Tanner graphs of $C_1$ and $C_2$; iterative decoding is therefore performed independently in parallel on the two components of $C$. That is, if $\mathcal{D}^1$ and $\mathcal{D}^2$ are the decoding networks of $C_1$ and $C_2$, respectively, and $\mathcal{D}$ is the decoding network of $C$, then the edge $((\mathbf{u}_1 \mid \mathbf{v}_1), (\mathbf{u}_2 \mid \mathbf{v}_2), \ell) \in E(\mathcal{D})$ if and only if $(\mathbf{u}_1, \mathbf{u}_2, \ell) \in E(\mathcal{D}^1)$ and $(\mathbf{v}_1, \mathbf{v}_2, \ell) \in E(\mathcal{D}^2)$. It follows that if both $\mathcal{D}^1$ and $\mathcal{D}^2$ are transitive, so is $\mathcal{D}$.

□

Observe that we may apply Corollary 5.1.10 to transitive decoding networks resulting from Theorems 5.2.6 and 5.2.8, and 5.2.9. Finally, we consider protograph codes.

**Theorem 5.2.10.** *Let $C$ be a binary linear code with Tanner graph $G$ and decoding network $\mathcal{D}$ with respect to a fixed decoder. Viewing $G$ as a protograph, let $\hat{C}$ be the code corresponding to a degree $J$ lift of $G$, denoted $\hat{G}$, and (with an abuse of notation), let $\hat{\mathcal{D}}$ be the decoding network of $\hat{C}$ with respect to the same decoder. Then, there exists a subgraph of $\hat{\mathcal{D}}$ that is isomorphic to $\mathcal{D}$. In particular, if $\mathcal{D}$ is not transitive, then $\hat{\mathcal{D}}$ is not transitive. However, transitivity of $\mathcal{D}$ does not necessarily imply transitivity of $\hat{\mathcal{D}}$.*

*Proof.* Let $C$ be a binary linear code of length $n$ with Tanner graph $G$ and decoding network $\mathcal{D}$ with respect to a fixed decoder. Let $\hat{C}$ be the code corresponding to a degree $J$ lift of $G$, denoted $\hat{G}$, and let $\hat{\mathcal{D}}$ be the decoding network of $\hat{C}$ with respect to the same decoder. Let $\hat{v}_{i,1}, \ldots \hat{v}_{i,J}$ denote the $J$ variable nodes in $\hat{G}$ that correspond to variable node $v_i$ in $G$, and let $\hat{c}_{j,1}, \ldots \hat{c}_{j,J}$ denote the $J$ check nodes in $\hat{G}$ that correspond to check node $c_j$ in $G$. We say that a variable node in $\hat{G}$ is of type $i$ if it belongs to $\{\hat{v}_{i,1}, \ldots \hat{v}_{i,J}\}$, and similarly for check nodes.

For each $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$, let $\hat{\mathbf{a}}$ denote the word in $\mathbb{F}_2^{J \cdot n}$ such that each of the $J$ coordinates of type $i$ in the lift is equal to $a_i$.

We claim that $\mathbf{a}$ decodes to $\mathbf{b}$ in $\ell$ iterations of the decoder run on $G$ if and only if $\hat{\mathbf{a}}$ decodes to $\hat{\mathbf{b}}$ in $\ell$ iterations of the decoder run on $\hat{G}$. To show this, we make the following claim, whose proof is deferred until the end of the theorem's proof:

**Lemma 5.2.11.** *If* $\mathbf{a}$ *is the set of inputs to the decoder on G, and* $\hat{\mathbf{a}}$ *is the set of inputs to the decoder on* $\hat{G}$*, then the (ordered) set of messages received by a variable node of type i in* $\hat{G}$ *after* $\ell$ *decoding iterations is identical to the set of messages received by* $v_i$ *in G after* $\ell$ *decoding iterations.*

Because the channel value $a_i$ is equal to the channel value for every coordinate of type $i$ in $\hat{\mathbf{a}}$, Lemma 5.2.11 says that $a_i$ will decode in the same way as all coordinates of type $i$ in $\hat{\mathbf{a}}$. Thus, $\mathbf{a}$ decodes to $\mathbf{b}$ in $\ell$ iterations if and only if $\hat{\mathbf{a}}$ decodes to $\hat{\mathbf{b}}$ in $\ell$ iterations. In terms of the decoding network, this means that $(\mathbf{a}, \mathbf{b}, l)$ is an edge in $\mathcal{D}$ if and only if $(\hat{\mathbf{a}}, \hat{\mathbf{b}}, l)$ is an edge in $\hat{\mathcal{D}}$.

We conclude that the map

$$\begin{aligned} \varphi : V(\mathcal{D}) &\rightarrow V(\hat{\mathcal{D}}) \\ \mathbf{x} &\mapsto \hat{\mathbf{x}} \end{aligned}$$

gives an isomorphism between $\mathcal{D}$ and the network induced by the vertices in $\mathrm{Im}(\varphi)$ in $\hat{\mathcal{D}}$. It follows easily that if $\mathcal{D}$ is not transitive, then its isomorphic image in $\hat{\mathcal{D}}$ is not transitive, and therefore $\hat{\mathcal{D}}$ itself is not transitive, concluding our proof.

We now return to prove our earlier claim.

*Proof of Lemma 5.2.11:* We will prove the lemma by induction on $\ell$, the number of decoding iterations.

Because all variable nodes of type $k$ in $\hat{G}$ have the same channel value as $v_k$ in $G$, by the definition of $\hat{\mathbf{a}}$, the initial message from a variable node of type $k$ to a check node of type $m$ is equal to the initial message from $v_k$ to $c_m$ in $G$. Because this holds for all types $k$ and $m$, the ordered set of messages received by a check node of type $j$ in $\hat{G}$ is equal to the ordered set of messages received by $c_j$ in $G$. Thus, the message from a check node of type $j$ to a variable node of type $i$ in $\hat{G}$ will be calculated in the same way as the message from $c_j$ to $v_i$ will be calculated in $G$. We conclude that the ordered set of messages received by a variable node of type $i$ in $\hat{G}$ after a single decoding iteration is identical to the set of messages received by $v_i$ in $G$ after a single decoding iteration. Thus, the claim holds for a single iteration of the decoder. Suppose that it holds for $\ell - 1$ iterations.

Since each neighbor of a check node of type $j$ in $\hat{G}$ received the same set of messages as its counterpart in $G$ after $\ell - 1$ iterations by assumption, the set of messages into a check node of type $j$ in $\hat{G}$ during the $\ell$th iteration is identical to the set of messages into $c_j$ in $G$ during the $\ell$th iteration. Thus, a check node of type $j$ sends out the same set of messages as $c_j$ during the $\ell$th iteration: that is, it sends its neighbor of type $i$ the same message that is sent from $c_j$ to $v_i$. As this holds for all $j$, the set of messages received by a variable node of type $i$ in $\hat{G}$ after $\ell$ decoding iterations is identical to the set of messages received by $v_i$ in $G$ after $\ell$ decoding iterations, proving the claim. $\square$

## 5.3 Periodic decoders and the Iteration Search Algorithm

In Example 5.1.9, if the all-zero codeword was sent, more received words are decoded correctly if only a single iteration of the decoder is run, rather than multiple iterations. In this section, we introduce the Iteration Search Algorithm (ISA), which optimizes the implementation of hard-decision decoding algorithms based on this observation. To this end, we introduce parameters we call the *decoding diameter*, the *aperiodic length*, and the *decoding period*.

**Definition 5.3.1.** *Let $\mathcal{D}$ be the decoding network of a code $C$ under a fixed decoder. For $\mathbf{y} \in V(\mathcal{D})$, let $L_{\mathbf{y}}$ be the minimum nonnegative integer such that for all $\ell \geq L_{\mathbf{y}}$, $\mathbf{y}^{\ell}$, the output of the decoder after $\ell$ iterations, appears an infinite number of times in the sequence $\{\mathbf{y}^k\}_{k=L_{\mathbf{y}}}^{\infty}$. Then, the* decoding diameter *of $\mathcal{D}$ is given by $\Delta(\mathcal{D}) = \max_{\mathbf{y}\in V(\mathcal{D})} L_{\mathbf{y}}$.*

After running a number of iterations equal to the decoding diameter of a decoding network, all errors will be contained in trapping sets of the code. Once this occurs, if the iterative decoder is allowed to run for more iterations, we may show that there exists an integer $p$ such that the iterative decoder output $\mathbf{y}^{p+i} = \mathbf{y}^i$ for any received word $\mathbf{y}$ and $i \geq k$, where $k$ is some value greater than or equal to $\Delta(\mathcal{D})$. We call the smallest $k$ for which this holds the *aperiodic length* and denote it by $m$; that is, for iterations 1 through $m-1$, the decoder's behavior is aperiodic.

**Definition 5.3.2.** *The* period $p_{\mathbf{y}}$ *for a received word* $\mathbf{y}$ *is the smallest positive integer such that the decoder output at iteration $p+i$ is equal to the output at iteration $i$ for all $i \geq m$. If no such integer exists, we say that the period is $\infty$. We denote by $p_{\max}$ the longest period over all possible received words. The* decoding period *is the least common multiple of the periods for all received words. We say that a decoding network is* periodic *if it has decoding period less than $\infty$.*

**Proposition 5.3.3.** *The decoding network of a finite block length code is periodic under any hard-decision decoder.*

*Proof.* The Tanner graph of a code of finite block length is finite, so the number of potential message states is finite. Thus, for each received word, the check node to variable node message state will eventually repeat, creating periodic outputs of the decoder. Since the set of possible received words is finite by assumption, the decoding period is finite by definition. □

**Example 5.3.4.** *Returning to Example 5.1.2, the decoding diameter of* $\mathscr{R}(0,2)$ *is equal to* 4*. After* 4 *iterations, for all* 16 *possible received words, the decoder's behavior is periodic. Thus, in this case, the aperiodic length is equal to the decoding diameter, though we note that this may not always be the case. In this example,* 4 *received words have decoding period* 7*,* 4 *have decoding period* 3*, and the remaining* 8 *have decoding period* 1*. Thus, the decoding network is periodic with decoding period* 21*, and* $p_{\max} = 7$.

**Example 5.3.5.** *Consider a Gallager-like finite geometry LDPC code, C, constructed from the Euclidean geometry* $EG(2,3)$ *as in Chapter 4, for transmission over the BSC, and assume decoding is performed via the Gallager A algorithm. We examine decoding parameters for two different parity-check matrix representations of C. First, let*

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

*There are* 16 *codewords in* $C$*; under Gallager A decoding, this representation has decoding diameter* 2*, though it does not admit a transitive decoding network. More specifically, after* 2 *iterations of the decoder, every word has decoded to the word it will statically decode to in any higher number of iterations. Thus, the aperiodic length is also* 2*, and both* $p_{\max}$ *and the decoding period are equal to* 1.

*Now, consider the representation* $H'$ *obtained by adding the element* 110001110 *of the dual code to the parity-check matrix* $H$ *as a seventh row. With representation* $H'$*, the decoding diameter of the code under Gallager A remains equal to* 2*, but the decoding period drastically increases. While most received words decode to the same word for any number of iterations greater than* 1*, a total of* 64 *received words move through a periodic cycle of outputs with period* 18.

*For example, in the case of the received word* 010000000*, the decoder cycles though outputs shown in Table 5.1.*

| Iteration | Output | Iteration | Output | Iteration | Output |
|---|---|---|---|---|---|
| 1 | 000000000 | 7 | 010000000 | 13 | 110010000 |
| 2 | 000001100 | 8 | 010000000 | 14 | 010000001 |
| 3 | 000100001 | 9 | 010000000 | 15 | 000000001 |
| 4 | 010001100 | 10 | 011000011 | 16 | 010001100 |
| 5 | 110110001 | 11 | 010100000 | 17 | 000000000 |
| 6 | 110010000 | 12 | 110011100 | 18 | 010000000 |

**Table 5.1:** Outputs of the Gallager A algorithm run on the received word 010000000 for 1 through 18 iterations. This pattern of outputs repeats for higher numbers of iterations.

*Thus, assuming the all-zero codeword was transmitted, each variable node is in error at some point during this cycle of outputs, and so the set of all variable nodes forms a trapping set induced by the second variable node, as stated in Theorem 5.1.7. The only codewords among these possible outputs from the decoder are* 000000000 *and* 011000011*. Given that* 010000000 *was received,* 000000000 *is the most likely codeword to have been sent. This output appears after running* 1 *iteration or after running* 17 *iterations, as shown* *in Figure 5.1. The other codeword output,* 011000011*, is output after* 10 *iterations.*

*In fact, this pattern holds for each of the* 64 *words with decoding periodicity* 18*. Thus, running either* 1 *or* 17 *iterations is optimal for these received words. Since all other received words are decoded statically for* 2 *iterations and higher, we conclude that it is optimal to run* 17 *iterations of the decoder. In this case, a codeword will be output for every received word.*

In Example 5.3.5, we were able to completely analyze the code's decoder outputs with parity-check matrix $H'$, and determine the optimal number of iterations to run for any received word. However, predetermining the decoder behavior for every received word is costly, and we may not be able to build a code's decoding network, and thus analyze behavior, from smaller component codes as was done in Section 5.2.1. However, if the aperiodic length and the value of $p_{\max}$ for a code are known, we can use a hard-decision decoder as a low-complexity tool to give a small list of possible codeword outputs. In the case of Example 5.3.5, this would allow us to simply choose between the outputs 000000000 and 011000011 for received word 010000000 using Maximum Likelihood decoding.

We now present an algorithm which realizes this idea. Let $m$ be the aperiodic length of a code with a particular representation under a given hard-decision decoder.

**Algorithm 2** Iteration Search Algorithm (ISA)

```
Input: Received word $\mathbf{y}$
1: $k \leftarrow 1$.
2: Run $m - 1$ iterations of the iterative decoder.
3: for $i = 0$ to $p_{\max} - 1$ do
4:     Run iteration $m + i$ of the iterative decoder.
5:     if $\mathbf{y}^{m+i}$ is a codeword then
6:         $\hat{\mathbf{x}}_k \leftarrow \mathbf{y}^{m+i}$
7:         $k \leftarrow k + 1$
8: if $k > 1$ then
9:     $\hat{\mathbf{x}} = \text{argmax}_{\hat{\mathbf{x}}_k} Pr(\mathbf{y} \text{ received } \mid \hat{\mathbf{x}}_k \text{ sent})$
Output: $\hat{\mathbf{x}}$ if it exists, ERROR else
```

Notice that if no codeword is output during the period of the decoder, then the Iteration Search Algorithm throws an error.

**Example 5.3.6.** *Continuing to build on Example 5.1.2, suppose we receive the word* $\mathbf{y} = 1000$*, and decode using the ISA. We first run* $m - 1 = 3$ *iterations of the decoder. On iterations* $m = 4$ *through* $m + p_{\max} - 1 = 10$*, we check to see if the output is a codeword. If it is, we record its value. In this case, iteration* $6$ *outputs the codeword* $0000$*, and iteration* $10$ *outputs the codeword* $1111$*. Iterations* $4, 5,$ *and* $7-9$ *output non-codewords. Of the two codeword outputs,* $0000$ *maximizes the probability that* $\mathbf{y}$ *was received given that it was sent, and so* $0000$ *is the output of the decoder. In fact, this is the ML codeword. However, if Gallager A were simply run until a codeword was output, the first codeword output would be* $1111$ *(on iteration* $3$*), which is not the ML codeword.*

*Suppose instead that we received* $\mathbf{y} = 0001$*. In this case, iterations* $7$ *and* $8$ *both output* $0000$*, and this is the only codeword output appearing the ISA's list. Thus, the ISA outputs the* $0000$*, which is, again, the ML codeword.*

*In this example,* $10$ *of the* $16$ *possible received words will output the ML codeword with the ISA. The other* $6$ *received words would not decode to a codeword in any number of iterations.*

**Remark 5.3.7.** *The ISA will, in general, output the ML codeword more often than the standard implementation of Gallager A. The exception is if the iterative decoder produces an ML codeword before the aperiodic length and, in addition, the ML codeword does not appear in the list produced by the ISA.*

To guarantee that the ISA always outputs the ML codeword if the ML codeword appears as an output in any dimension of the decoding network, we present a modification in Algorithm 3 which also records outputs for iterations before the aperiodic length. In ISA2, a list of codewords is not maintained; rather, only the codeword with the lowest "cost" is retained at each step of the algorithm. However, we still require ISA2 to run for all $m + p_{\max} - 1$ iterations since this range includes all possible codeword outputs and it remains possible that the ML codeword appears only in later iterations.

**Algorithm 3** Iteration Search Algorithm 2 (ISA2)

```
Input: Received word $\mathbf{y}$
 1: $k \leftarrow 1$, cost = $\infty$, $\hat{\mathbf{x}} = \mathbf{0}$.
 2: for $i = 1$ to $m - 1$ do
 3:     if $\mathbf{y}^i$ is a codeword then
 4:         newcost = $Pr(\mathbf{y}$ received $\mid \mathbf{y}^i$ sent$)$
 5:         if newcost < cost then
 6:             cost $\leftarrow$ newcost
 7:             $\hat{\mathbf{x}} \leftarrow \mathbf{y}^i$
 8: for $i = 0$ to $p_{\max} - 1$ do
 9:     Run iteration $m + i$ of the iterative decoder.
10:     if $\mathbf{y}^{m+i}$ is a codeword then
11:         newcost = $Pr(\mathbf{y}$ received $\mid \mathbf{y}^{m+i}$ sent$)$
12:         if newcost < cost then
13:             cost$\leftarrow$ newcost
14:             $\hat{\mathbf{x}} \leftarrow \mathbf{y}^{m+i}$
Output: $\hat{\mathbf{x}}$
```

Algorithms 2 and 3 rely on knowledge of the aperiodic length and the value of $p_{\max}$; however, even if we do not know the exact values of these parameters, we may put an upper bound on their sum (and each of them individually):

**Theorem 5.3.8.** *If $\chi$ is the number of possible variable-to-check message patterns for a given hard-decision decoder, then*

$$m + p_{\max} \leq 2 \cdot \chi.$$

*Proof.* Each of $m$ and $p_{\max}$ is bounded above by the number of variable-to-check message patterns, since as soon as a pattern repeats, the decoder is periodic. □

For the Gallager A and B algorithms, we may put bounds on the number of possible variable-to-check message patterns as follows.

**Theorem 5.3.9.** *For Gallager B operating on a particular choice of representation of a code, the number of possible variable-to-check message patterns is bounded above by*

$$\left(\binom{\Delta_V}{\lceil t \cdot (\Delta_V - 1)\rceil} + 2\right)^n,$$

*where n is the block length of the code, t is the portion of other check-to-variable messages that must disagree with the channel value in order for an edge to send back a message disagreeing with the channel value, and $\Delta_V$ is the maximum variable node degree in the Tanner graph corresponding to that representation.*

*Proof.* In the Gallager B algorithm, there are only three possible types of variable-to-check message patterns leaving a single variable node $v$. Either all outgoing messages agree with the channel value, all disagree with the channel value, or exactly $\lceil t \cdot (d(v) - 1)\rceil$ messages agree with the channel value, where $d(v)$ is the degree of variable node $v$. This is a consequence of the variable-to-check message rule of the algorithm. There is one way each that the first two types can occur, and $\binom{d(v)}{\lceil t\cdot(d(v)-1)\rceil}$ patterns of the final type.

Then, the total number of possible variable-to-check message patterns is upper bounded as follows:

$$\begin{aligned}\prod_{v\in V}\left(\binom{d(v)}{\lceil t \cdot (d(v) - 1)\rceil} + 2\right) &\leq \prod_{v\in V}\left(\binom{\Delta_V}{\lceil t \cdot (\Delta_V - 1)\rceil} + 2\right)\\ &= \left(\binom{\Delta_V}{\lceil t \cdot (\Delta_V - 1)\rceil} + 2\right)^n,\end{aligned}$$

where $V$ is the set of all variable nodes. $\square$

**Corollary 5.3.10.** *For Gallager A operating on a particular choice of representation of a code, the number of possible variable-to-check message patterns is bounded above by $(\Delta_V + 2)^n$, where $n$ is the block length of the code and $\Delta_V$ is the maximum variable node degree in the Tanner graph corresponding to that representation.*

*Proof.* This follows from Theorem 5.3.9 by letting $t = 1$. □

**Corollary 5.3.11.** *For Gallager A decoding,*

$$m + p_{\max} \leq 2(\Delta_V + 2)^n,$$

*where $n$ is the block length of the code and $\Delta_V$ is the maximum variable node degree.*

*Proof.* This follows from Theorem 5.3.8 and Corollary 5.3.10. □

If more is known about the Tanner graph's structure, then these bounds may be improved. An ongoing line of work is providing improved bounds on the aperiodic length and maximum period length for particular classes of codes.

A modified version of the ISA, which we call the Trapping set Search Algorithm (TSA), can be used to identify the trapping sets of the code. To find the trapping set induced by a received word $\mathbf{y}$, we may run the TSA, given in Algorithm 4. The output trapping set is an $(a, b)$-trapping set with $a$ equal to the size of the output set, and $b$ determined using the parity-check matrix of the code. In order to find all trapping sets of the code, the TSA may be run on a set of $2^{n-k}$ possible received words which are not translations of one another by codewords. The resulting trapping sets and their translations by codewords will give the set of all trapping sets.

**Example 5.3.12.** *The $r^{th}$* order binary Reed-Muller code $\mathscr{R}(r, m)$ *may be defined as the set of all binary vectors of length $2^m$ that arise as the output vectors of polynomial Boolean functions of degree at most $r$. Consider the Reed-Muller code $\mathscr{R}(1, 3)$, which has dimension*

**Algorithm 4** Trapping set Search Algorithm (TSA)

```
Input: Received word $\mathbf{y}$
1: Run $m - 1$ iterations of the iterative decoder.
2: for $i = 0$ to $p_{\max} - 1$ do
3:     Run iteration $m + i$ of the iterative decoder.
4:     $\hat{\mathbf{x}}_i \leftarrow \mathbf{y}^{m+i}$
5: $S = \bigcup_{0 \le i \le p_{\max}-1} \mathrm{supp}(\hat{\mathbf{x}}_i)$
Output: $\{v_i\}_{i \in S}$
```

4 *and block length* $2^3$*, for transmission over the BSC. We may define the code via the parity-check matrix*

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

*If decoded using Gallager A, the aperiodic length of the decoding network is* $m = 10$*, and the decoding period and* $p_{\max}$ *are both equal to* 2*. If decoding is halted as soon as a codeword has been reached, as is standard, a received word is decoded as an ML codeword* 43.75% *of the time. With this representation, the ISA has the same success rate.*

*However, suppose we use a different representation,* $H'$*, of the same code, obtained by adding the row* 01011010 *to* $H$*. With this representation, the aperiodic length is* $m = 5$*, and* $p_{\max} = 20$*. With the ISA,* 75% *of received words will decode to an ML codeword, as compared with* 62.5% *that will decode to an ML codeword with the standard Gallager A implementation. In particular, translations of received words* 00000111 *and* 00001000 *by codewords will decode to a non-ML codeword under the traditional implementation of Gallager A, but decode to an ML codeword with the ISA. The* 25% *of received words which do not decode to an ML codeword output an error with the ISA, and are run for the maximum number of iterations with standard Gallager A, never reaching a codeword output in* *either case. For example, the decoder outputs for the received word* 00000110 *eventually oscillate between* 00000111 *and* 11110111*, both non-codewords. For this received word, the TSA would output the set* $\{v_1, v_2, v_3, v_4, v_6, v_7, v_8\}$*, the trapping set induced by* $\{v_6, v_7\}$*.*

In the case of a transitive decoding network, the ISA behaves according to the following proposition.

**Proposition 5.3.13.** *If the decoding network, $\mathcal{D}$, of a code is transitive, then given a received word* $\mathbf{y}$*, the ISA decodes* $\mathbf{y}$ *as follows: if the directed path in $\mathcal{D}_1$ beginning at* $\mathbf{y}$ *ends in a vertex with a loop, then* $\mathbf{y}$ *decodes to that word. If the directed path from* $\mathbf{y}$ *ends in a directed cycle, the decoder returns an error.*

In particular, when a code has a transitive decoding network, once the chosen decoder outputs a codeword, it will output that codeword for any higher number of iterations. Thus, we may simplify the ISA in this case.

**Corollary 5.3.14.** *If the decoding network, $\mathcal{D}$, of a code is transitive, then the ISA may be simplified to only check the $m^{th}$ iteration. If the $m^{th}$ iteration outputs a codeword, this is the output of the ISA. Otherwise, an error is output.*

*Proof.* If the $m^{th}$ iteration outputs a codeword, all higher numbers of iterations will output this same codeword, and so it will be the output of the ISA. Otherwise, no higher number of iterations will output a codeword; if so, $m$ was not the aperiodic length. In this case, the ISA will output an error. □

Recall that in the case of transitive decoding networks, trapping sets are in one-to-one correspondence with directed cycles and vertices with loops in dimension 1 of the network. Thus, the TSA will identify the loops and directed cycles of dimension 1 of the decoding network.

# Chapter 6

# Conclusions

In this work, we examined the behavior of iterative low-complexity graph-based decoding algorithms in several contexts. We first looked at spatially-coupled LDPC codes, showing that their construction process may be standardized using algebraic graph lifts, and that this framework may be leveraged to remove harmful trapping and absorbing sets in the resulting code. We then examined the stopping and absorbing sets of hypergraph codes and finite geometry LDPC codes, respectively, and gave a construction of algebraic hypergraph lifts. Finally, we introduced a multidimensional decoding network framework to encode the behavior of any hard-decision message-passing decoder for a code with a given graph representation. We examined several parameters of decoding networks, and presented an Iteration Search Algorithm to improve the performance of hard-decision decoders based on these parameters.

Avenues for future work include defining explicit permutation assignments to minimize the number of harmful absorbing sets in an SC-LDPC code, determining graph or parity-check matrix conditions that capture decoding network parameters, and extensions of the applications of decoding networks to larger classes of codes and decoders.

# Bibliography

[1] C. Shannon, “A mathematical theory of communication,” *Bell Sys. Tech. J.*, vol. 27, pp. 379–423.

[2] R. Gallager, “Low-density parity-check codes,” *IRE Trans. on Info. Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[3] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. on Info. Theory*, vol. 27, no. 5, pp. 533–547, Sep 1981.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo-codes,” in *Proc. IEEE Int’l Conf. on Communications (ICC)*, vol. 2, May 1993, pp. 1064–1070 vol.2.

[5] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Trans. on Info. Theory*, vol. 47, no. 2, pp. 619–637, Feb 2001.

[6] A. Jimenez Felstrom and K. Zigangirov, “Time-varying periodic convolutional codes with low-density parity-check matrix,” *IEEE Trans. on Info. Theory*, vol. 45, no. 6, pp. 2181–2191, Sep 1999.

[7] D. J. Costello, L. Dolecek, T. E. Fuja, J. Kliewer, D. G. M. Mitchell, and R. Smarandache, “Spatially coupled sparse codes on graphs: theory and practice,” *IEEE Comm. Magazine*, vol. 52, no. 7, pp. 168–176, July 2014.

[8] S. Kudekar, T. Richardson, and R. L. Urbanke, “Spatially coupled ensembles universally achieve capacity under belief propagation,” *IEEE Trans. on Info. Theory*, vol. 59, no. 12, pp. 7761–7813, Dec 2013.

[9] Y. Kou, S. Lin, and M. Fossorier, “Construction of low density parity check codes: a geometric approach,” in *Proc. of the 2nd Int’l Symp. on Turbo Codes and Related Topics*, 2000, pp. 137–140.

[10] Y. Bilu and S. Hoory, “On codes from hypergraphs,” *European Journal of Combinatorics*, vol. 25, no. 3, pp. 339 – 354, 2004.

[11] L. Dolecek, “On absorbing sets of structured sparse graph codes,” in *Information Theory and Applications Workshop (ITA)*, Jan 2010, pp. 1–5.

[12] A. Beemer and C. A. Kelley, “Avoiding trapping sets in SC-LDPC codes under windowed decoding,” in *Proc. IEEE Int’l Symp. on Info. Theory and its Applications (ISITA)*, Oct 2016, pp. 206–210.

[13] A. Beemer, S. Habib, C. A. Kelley, and J. Kliewer, “A generalized algebraic approach to optimizing SC-LDPC codes,” in *Proc. of the 55th Annual Allerton Conf. on Communication, Control, and Computing*, Oct 2017, pp. 672–679.

[14] A. Beemer, K. Haymaker, and C. A. Kelley, “Absorbing sets of codes from finite geometries,” Submitted 2017.

[15] A. Beemer, C. Mayer, and C. A. Kelley, “Erasure correction and locality of hypergraph codes,” in *International Castle Meeting on Coding Theory and Applications*. Springer, 2017, pp. 21–29.

[16] A. Beemer and C. A. Kelley, "Multidimensional decoding networks for trapping set analysis," in *International Castle Meeting on Coding Theory and Applications*. Springer, 2017, pp. 11–20.

[17] ——, "Iterative decoder analysis using multidimensional decoding networks," Submitted 2017.

[18] W. C. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[19] M. G. Luby, M. A. Shokrolloahi, M. Mizenmacher, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation," in *Proc. IEEE Int'l Symp. on Info. Theory (ISIT)*, Aug 1998, pp. 117–.

[20] V. V. Zyablov and M. S. Pinsker, "Estimation of the error-correction complexity for Gallager low-density codes," *Problems of Info. Transmission*, vol. 11, no. 1, pp. 18–28, 1975.

[21] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. on Info. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov 1996.

[22] J. Thorpe, "Analysis and design of protograph based LDPC codes and ensembles," Ph.D. dissertation, California Institute of Technology, 2005.

[23] C. A. Kelley, "On codes designed via algebraic lifts of graphs," *Proc. of the 46th Annual Allerton Conf. on Communication, Control, and Computing*, pp. 1254–1261, 2008.

[24] J. L. Gross and T. W. Tucker, *Topological Graph Theory*. Dover, 1987.

[25] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. on Info. Theory*, vol. 45, no. 2, pp. 399–431, Mar 1999.

[26] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. on Info. Theory*, vol. 47, no. 2, pp. 498–519, Feb 2001.

[27] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, “Finite-length analysis of low-density parity-check codes on the binary erasure channel,” *IEEE Trans. on Info. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun 2002.

[28] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, “Analysis of absorbing sets for array-based ldpc codes,” in *IEEE Int. Conf. on Communications*, June 2007, pp. 6261–6268.

[29] R. Koetter and P. Vontobel, “Graph-covers and iterative decoding of finite length codes,” in *Proc. 3rd Int. Symp. on Turbo Codes*, 2003.

[30] T. Richardson, “Error-floors of LDPC codes,” in *Proc. of the 41st Allerton Conf. on Communication, Control and Computing*, 2003, pp. 1426–1435.

[31] 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. on Info. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr 2012.

[32] S. Sankaranarayanan, S. K. Chilappagari, R. Radhakrishnan, and B. Vasic, “Failures of the Gallager B decoder: analysis and applications,” in *Proc. IEEE Int’l Conf. on Comm.*, 2006.

[33] J. Sun, “Studies on graph-based coding systems,” Ph.D. dissertation, The Ohio State University, 2004.

[34] H. Park, J. S. No, B. Shin, and H. Chung, “On the combinatorial substructures of LDPC codes causing error floors in the AWGN channel,” in *Proc. Int’l Conf. on ICT Convergence (ICTC)*, Oct 2012, pp. 420–425.

[35] B. K. Butler and P. H. Siegel, “Error floor approximation for LDPC codes in the AWGN channel,” *IEEE Trans. on Info. Theory*, vol. 60, no. 12, pp. 7416–7441, Dec 2014.

[36] B. Vasic, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, “Trapping set ontology,” in *Proc. of the 47th Allerton Conf. on Communication, Control, and Computing*, Sept 2009, pp. 1–7.

[37] C. A. Kelley and D. Sridhara, “Pseudocodewords of Tanner graphs,” *IEEE Trans. on Info. Theory*, vol. 53, no. 11, pp. 4013–4038, Nov 2007.

[38] N. Miladinovic and M. P. C. Fossorier, “Generalized LDPC codes and generalized stopping sets,” *IEEE Trans. on Comm.*, vol. 56, no. 2, pp. 201–212, February 2008.

[39] D. Mitchell, L. Dolecek, and D. Costello, “Absorbing set characterization of array-based spatially coupled LDPC codes,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, June 2014, pp. 886–890.

[40] H. Hatami, D. G. M. Mitchell, D. J. Costello, and T. Fuja, “Performance bounds for quantized LDPC decoders based on absorbing sets,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, July 2016, pp. 2539–2543.

[41] ——, “Lower bounds for quantized LDPC min-sum decoders based on absorbing sets,” in *Proc. of the 55th Annual Allerton Conf. on Communication, Control, and Computing*, Oct 2017, pp. 694–699.

[42] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, “LDPC block and convolutional codes based on circulant matrices,” *IEEE Trans. on Info. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec 2004.

[43] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, “Deriving good LDPC convolutional codes from LDPC block codes,” *IEEE Trans. on Info. Theory*, vol. 57, no. 2, pp. 835–857, Feb 2011.

[44] M. Papaleo, A. R. Iyengar, P. H. Siegel, J. K. Wolf, and G. E. Corazza, “Windowed erasure decoding of LDPC convolutional codes,” in *IEEE Info. Theory Workshop (ITW), Cairo*, Jan 2010, pp. 1–5.

[45] A. R. Iyengar, P. H. Siegel, R. L. Urbanke, and J. K. Wolf, “Windowed decoding of spatially coupled codes,” *IEEE Trans. on Info. Theory*, vol. 59, no. 4, pp. 2277–2292, Apr 2013.

[46] B. Amiri, A. Reisizadeh, J. Kliewer, and L. Dolecek, “Optimized array-based spatially-coupled LDPC codes: An absorbing set approach,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, June 2015, pp. 51–55.

[47] D. G. M. Mitchell and E. Rosnes, “Edge spreading design of high rate array-based SC-LDPC codes,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, June 2017, pp. 2940–2944.

[48] H. Esfahanizadeh, A. Hareedy, and L. Dolecek, “A novel combinatorial framework to construct spatially-coupled codes: Minimum overlap partitioning,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, June 2017, pp. 1693–1697.

[49] L. Chen, S. Mo, D. J. Costello, D. G. M. Mitchell, and R. Smarandache, “A protograph-based design of quasi-cyclic spatially coupled LDPC codes,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, June 2017, pp. 1683–1687.

[50] J. Fan, “Array-codes as low-density parity-check codes,” in *Second Int’l Symp. on Turbo Codes*, Sept. 2000, pp. 543–546.

[51] E. Rosnes, “On the minimum distance of array-based spatially-coupled low-density parity-check codes,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, June 2015, pp. 884–888.

[52] A. Iyengar, M. Papaleo, P. Siegel, J. Wolf, A. Vanelli-Coralli, and G. Corazza, “Windowed decoding of protograph-based LDPC convolutional codes over erasure channels,” *IEEE Trans. on Info. Theory*, vol. 58, no. 4, pp. 2303–2320, Apr 2012.

[53] M. Lentmaier, M. Prenda, and G. Fettweis, “Efficient message passing scheduling for terminated LDPC convolutional codes,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, July 2011, pp. 1826–1830.

[54] A. E. Pusane, A. J. Feltstrom, A. Sridharan, M. Lentmaier, K. S. Zigangirov, and D. J. Costello, “Implementation aspects of LDPC convolutional codes,” *IEEE Trans. on Communications*, vol. 56, no. 7, pp. 1060–1069, July 2008.

[55] M. Karimi and A. H. Banihashemi, “An efficient algorithm for finding dominant trapping sets of irregular LDPC codes,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, July 2011, pp. 1091–1095.

[56] M. Ivkovic, S. K. Chilappagari, and B. Vasic, “Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers,” *IEEE Trans. on Info. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug 2008.

[57] D. B. West, “Combinatorial mathematics,” Fall 2014 edition.

[58] B. Amiri, A. Reisizadehmobarakeh, H. Esfahanizadeh, J. Kliewer, and L. Dolecek, “Optimized design of finite-length separable circulant-based spatially-coupled codes: An absorbing set-based analysis,” *IEEE Trans. on Communications*, vol. 64, no. 10, pp. 4029–4043, Oct 2016.

[59] Y. Kou, S. Lin, and M. P. C. Fossorier, “Low-density parity-check codes based on finite geometries: a rediscovery and new results,” *IEEE Trans. on Info. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov 2001.

[60] S. J. Johnson and S. R. Weller, “Codes for iterative decoding from partial geometries,” *IEEE Trans. on Comm.*, vol. 52, no. 2, pp. 236–243, Feb 2004.

[61] C. A. Kelley, D. Sridhara, and J. Rosenthal, “Tree-based construction of LDPC codes having good pseudocodeword weights,” *IEEE Trans. on Info. Theory*, vol. 53, no. 4, pp. 1460–1478, April 2007.

[62] R. M. Tanner, “Explicit concentrators from generalized N-gons,” *SIAM Journal on Algebraic Discrete Methods*, vol. 5, no. 3, pp. 287–293, 1984.

[63] L. M. Batten, *Combinatorics of finite geometries*. Cambridge University Press, 1997.

[64] H. Liu, Y. Li, L. Ma, and J. Chen, “On the smallest absorbing sets of LDPC codes from finite planes,” *IEEE Trans. on Info. Theory*, vol. 58, no. 6, pp. 4014–4020, June 2012.

[65] S. Landner and O. Milenkovic, “Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes,” in *Int’l Conf. on Wireless Networks, Comm. and Mobile Computing*, vol. 1, 2005, pp. 630–635.

[66] Q. Diao, Y. Y. Tai, S. Lin, and K. Abdel-Ghaffar, “Trapping set structure of LDPC codes on finite geometries,” in *Information Theory and Applications Workshop (ITA)*, Feb 2013, pp. 1–8.

[67] S.-T. Xia and F.-W. Fu, “On the stopping distance of finite geometry LDPC codes,” *IEEE Comm. Letters*, vol. 10, no. 5, pp. 381–383, May 2006.

[68] R. Smarandache and P. O. Vontobel, “Pseudo-codeword analysis of Tanner graphs from projective and Euclidean planes,” *IEEE Trans. on Info. Theory*, vol. 53, no. 7, pp. 2376–2393, July 2007.

[69] J. W. Hirschfeld and L. Storme, “The packing problem in statistics, coding theory and finite projective spaces: update 2001,” in *Finite geometries*. Springer, 2001, pp. 201–246.

[70] J. Bierbrauer and Y. Edel, “Large caps in projective Galois spaces,” *Current Research Topics in Galois Geometry (Eds. M. De Beule and L. Storme), Nova Science Publishers*, pp. 85–102, 2011.

[71] S. Ball, *Finite geometry and combinatorial applications*. Cambridge University Press, 2015, vol. 82.

[72] A. Barg and G. Zemor, “Codes on hypergraphs,” in *Proc. IEEE Int’l Symp. on Info. Theory (ISIT)*, July 2008, pp. 156–160.

[73] N. Alon, “Eigenvalues and expanders,” *Combinatorica*, vol. 6, no. 2, pp. 83–96, Jun 1986.

[74] N. Alon and J. H. Spencer, *The probabilistic method*. John Wiley & Sons, 2004.

[75] A. S. Rawat, A. Mazumdar, and S. Vishwanath, “Cooperative local repair in distributed storage,” *EURASIP Journal on Advances in Signal Processing*, vol. 2015, no. 1, p. 107, Dec 2015.

[76] A. S. Rawat, D. S. Papailiopoulos, A. G. Dimakis, and S. Vishwanath, “Locality and availability in distributed storage,” *IEEE Trans. on Info. Theory*, vol. 62, no. 8, pp. 4481–4493, Aug 2016.

[77] A. Marcus, D. A. Spielman, and N. Srivastava, “Interlacing families i: Bipartite ramanujan graphs of all degrees,” in *Proc. IEEE Annual Symp. on Foundations of Computer Science (FOCS)*. IEEE, 2013, pp. 529–537.

[78] M. Berlingerio, M. Coscia, F. Giannotti, A. Monreale, and D. Pedreschi, “Foundations of multidimensional network analysis,” in *Int’l Conf. on Adv. in Social Networks Analysis and Mining*, Jul 2011, pp. 485–489.

[79] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms, 3rd ed.*, 3rd ed. MIT Press, Jul. 2009.

[80] M. Karimi and A. H. Banihashemi, “Efficient algorithm for finding dominant trapping sets of LDPC codes,” *IEEE Trans. on Info. Theory*, vol. 58, no. 11, pp. 6942–6958, Nov 2012.

[81] Y. Hashemi and A. H. Banihashemi, “Characterization of elementary trapping sets in irregular LDPC codes and the corresponding efficient exhaustive search algorithms,” *IEEE Trans. on Info. Theory*, 2018.

[82] J. Justesen, “Performance of product codes and related structures with iterated decoding,” *IEEE Trans. on Comm.*, vol. 59, no. 2, pp. 407–415, February 2011.

[83] T. Mittelholzer, T. Parnell, N. Papandreou, and H. Pozidis, “Improving the error-floor performance of binary half-product codes,” in *Proc. IEEE Int’l Symp. on Info. Theory and its Applications (ISITA)*, Oct 2016.

[84] H. D. Pfister, S. K. Emmadi, and K. Narayanan, “Symmetric product codes,” in *Info. Theory and App. Wkshp (ITA)*, Feb 2015, pp. 282–290.