University of Alberta

Controlling the Error Floors of the Low-Density Parity-Check Codes

by

Shuai Zhang

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Communications

Department of Electrical and Computer Engineering

Spring 2013
Edmonton, Alberta

To my parents
and grandparents

Abstract

The goal of error control coding is to encode information in such a way, that in the event that errors occur during transmission over a noisy communication channel or during storage in an unreliable memory, the receiver can correct the errors and recover the original transmitted information. Low-density parity-check (LDPC) codes are a kind of high performance linear block code, which are already used in many recent communication systems. Information rates guaranteed by these codes approach the theoretical Shannon capacity limit. In addition, LDPC codes are now widely used in practice and included in many communication standards. Therefore study of these codes is very important and practical approaches to their design and decoding techniques are of great interest.

There exist very good decoding algorithms for LDPC codes with respect to different channel models. However, LDPC codes suffer from the infamous error floor problem at high signal-to-noise ratios. This problem is attributed to the trapping sets by Richardson. It is shown that the dominant trapping sets of regular LDPC codes, so called absorption sets, undergo a two-phase dynamic behavior in the iterative message-passing decoding algorithm. We present a linear dynamic model for the iteration behavior of these sets. It can be seen that they undergo an initial geometric growth phase which stabilizes in a final bit-flipping behavior where the algorithm reaches a fixed point. Our analysis is shown to lead to very accurate numerical calculations of the error floor bit error rates down to error rates that are inaccessible by simulation.

The topologies of the dominant absorption sets of two example codes, the IEEE 802.3an $[2048, 1723]$ regular $(6, 32)$ LDPC code and the Tanner $[155, 64, 20]$ regular $(3, 5)$ LDPC code, are identified and tabulated by using topological features in combination with search algorithms, respectively.

To make our analysis more complete, we provide more solid evidence showing that trapping sets are equivalent to absorption sets. In other words, we argue that absorption sets characterize all failure mechanisms. Some insights from our linear analysis can be borrowed to approach this problem.

Another insight that this formula provides to us is the means to reduce the error floor, which is another important part of what we hoped to achieve by developing the formula in the first place. By allowing the log-likelihood ratios (LLRs) utilized by the message-passing decoding to grow bigger in precision length, the absorption sets as a trouble-causing structure will be successfully corrected. Therefore, the absorption sets, generically born with the code design, will no longer threaten the error correcting performance as long as the messages have enough room and time to grow. This translates to greater LLR clipping thresholds and more iterations, both of which are preset at the decoder. However, the actual settings are dependent on how much the subgraph of the dominant absorption set resembles that of a non-zero minimum-weight codeword, therefore the settings are empirical. In general more likeness in their topologies, the more effort it costs to correct the absorption set.

Acknowledgements

I owe sincere thanks to my research supervisor, Dr. Christian Schlegel, who provided continuous support and made me believe in myself through the entire process of the doctorate program and dissertation writing. His deep understanding of the coding theory from both the academic and the industrial points of view has improved my knowledge significantly. The many discussions with Dr. Schlegel truly inspired and benefited me in completing my work. I would also like to express my sincere gratitude to my co-supervisor, Dr. Aiden Bruen from the University of Calgary. It was he who originally recruited me into this program. There is no doubt that this dissertation would not have been possible without their guidance, understanding, patience, and generosity.

I would like to thank the other members of my examining committee, Dr. Bruce Cockburn, Dr. Ivan Fair, Dr. Masoud Ardakani, and certainly last but not least, Dr. Lance Pérez from the University of Nebraska, USA. I thank them all for reviewing my thesis thoroughly and professionally. Their insightful feedback and expertise helped broaden my perspective and improve my thesis during the last step of this program.

I would also like to thank my fellow colleagues and friends for their help and friendship throughout the whole program. They are Dr. Dmitry Trukhachev, Dr. Deyasini Majumdar, Dr. Sumeeth Nagaraj, Dr. Saeed Fouladi Fard, Dr. Lukasz Krzymien, Dr. Marcel Jar, Dr. Gongpu Wang, Dr. Rongfei Fan, Dr. Larry Hua, Majid Ghanbarinejad, Russell Dodd, Michiko Maruyama, Rezwana Huq, Navid Rezaei, Sheehan Khan, Ke Li, Jie Gao, Bo Hu, Li Xu, and Michelle Xia. The list could go on. I would like to thank the staff of the ECE Department, especially, Ms. Pinder Bains and Ms. Kathleen Forbes, for their assistance in a timely and friendly manner. I would also like to express my gratitude to the families of the Sieberts, the Wienses, the Sealeys, the Fehrs, and the Cornfields for their love and encouragement.

Finally, my special thanks go to my parents and grandparents for their unconditional love and support.

Table of Contents

List of Tables

List of Figures

List of Symbols

Chapter 1

Introduction

Nowadays, the number of applications which involve information transmission is rapidly increasing, as well as the requirements for quality of service. Ad-hoc networks, satellite communications, single and multi-antenna wireless communications, Internet packet transmission are just some of the large variety of examples. Each transmission link is exposed to noise corrupting the data sent from source to destination. Therefore the fundamental component of information transfer over any communication link is *error control coding*. The code introduces redundant information to the transmitted data in order to correct the errors and recover the original transmitted message at the receiver. Hence understanding the performance of an error control code is critical for designing a communications system.

If we intend to communicate over a distance, the messages originally sent very often become distorted, weakened, or lost. For example, if Bob received a message from Alice saying that "I found a *rlog* in the riverbed today", this would not make very much sense to him, apparently. Requesting Alice to resend the message is out of the question because she is actually currently on a mission on Mars. Hence Bob has to guess the word based on what he received plus the context. The candidates "*frog*", "*clog*" and "*blog*" are all very *close* to the received message. However only the first two would make the sentence meaningful in context. Bob decides that there is more chance that Alice is trying to tell him that she found a *frog*, instead of a *clog*, in the riverbed on Mars. That decision is probability-based. In other words, it is also quite possible that the original message could be "I found a *rock* in the riverbed today". All depends on the level of distortion the message experienced.

In this communications context, the role played by Alice is usually called the transmitter, which makes Bob the receiver. The words of the original message from the transmitter belong to a special technical vocabulary, termed *the codewords*, which is known to both the transmitter and receiver. For example, the English words are valid codewords in the code we call *language*. Technically, of course, the codewords are more typically sequences of binary digits that are processed by computers.

If there is a word in the received message that does not match a codeword in the vocabulary, we say an error has occurred during the transmission or an error has been detected at the receiver. So Bob detected the error "rlog" in his received message. The receiver will try the best to recover the original message or correct the error by mapping the erroneous word to the *closest* codeword, considering operational constraints. This process is referred to as operating a *decoding algorithm*. "Frog", "clog" and "blog" are all valid codewords that differ from the erroneous word "rlog" in one letter. Therefore they, along with potentially others, are considered the closest codewords to the erroneous word. Unfortunately, assum-

ing letter errors occur with equal likelihood, there is an equal chance that any of them could be the original transmitted codeword. Even taking into account the context that "rlog" was found in a riverbed, Bob was still not able to limit the candidates to a single one. Given that it would be impractical, time- and cost-wise, to ask Alice to sent the message from the Mars again, Bob has to make a decision and is taking the risk of making a *decoding error*.

In this example, the *distance* between two codewords is defined as the number of letters where the two codewords differ. And the decoding rule is to *decode to the valid codeword with the smallest distance from the received word*. There may be multiple choices, in which case, the decoder will pick one randomly. To minimize the probability of decoding errors, the code designer will pick the codewords carefully such that they are spread far apart in terms of letter differences. With high probability now, the received word will fall in the "neighborhood" of a unique codeword, which is the nearest codeword to the error.

In addition, the nature of the media, primarily the *channel* in which the message is transmitted, can be studied and simulated by employing mathematical models. Taking advantage of the knowledge of the channel, the decoding algorithm can be adapted accordingly. In other words, learning how the channel affects transmissions will help a decoder improve its efficiency and accuracy.

Some of the key components of a communication scheme are plotted in Figure 1.1. The message $\mathbf{y}$ is distorted by the noise when passing through a channel. Thus a decoder

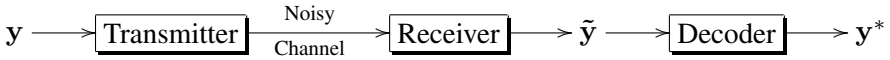


Figure 1.1: An illustration of a communication scheme.

is applied on $\tilde{\mathbf{y}}$ to correct the distortion and get what was originally transmitted. In other words, the decoder output, $\mathbf{y}^*$, is supposed to be the same as $\mathbf{y}$, ideally. However, the performance of decoding algorithms depend on both the code structure and the channel characteristics. The most common communications channel models are the binary symmetric channel (BSC), the binary erasure channel (BEC), and the additive white Gaussian noise (AGWN) channel. In this thesis, the noisy channel in Figure 1.1 is modeled only by binary-input memoryless channels, which are characterized by the conditional probabilities:

$$\Pr_{\text{noise}}[\text{output}|\text{input}]. \tag{1.1}$$

That is the formulation that allowed Shannon to show the channel capacity mathematically, [66]. The channels will be explained more in the next chapter where the decoding module is introduced. Before that, the basics of the code are reviewed in the rest of this chapter.

1.1 Binary Linear Block Codes

Prior to transmission, a message will be mapped to a sequence that consists of zeros and ones, i.e., a digitized binary sequence. This binary sequence will be partitioned into equal-length blocks and fed into an encoder block by block. The output of the encoder will be binary codewords with a one-to-one correspondence to the input blocks. The "distance" between two codewords is a way to characterize the resemblance or difference of them.

In information theory, the *Hamming distance*, or *distance* for simplicity, between two strings of equal length is measured as the number of positions where their corresponding

digits differ. The *Hamming weight* or *weight* of a string is the Hamming distance between the string and the all-zero string, or, equivalently, the number of non-zero entries of the string.

Example 1.1. The distance between $\mathbf{y}_1 = [10010101]$ and $\mathbf{y}_2 = [11010101]$ is one since they differ at the second position. The weights of $\mathbf{y}_1$ and $\mathbf{y}_2$ are 4 and 5, respectively.

Note that the distance between $\mathbf{y}_1$ and $\mathbf{y}_2$ is equal to the weight of their "sum", $\mathbf{y}_1 \oplus \mathbf{y}_2$, where the operation "$\oplus$" represents bit-wise addition modulo two, also known as the logical operation *exclusive disjunction* or *exclusive or*, often denoted by XOR or EOR.

$$\begin{aligned} \mathrm{wt}\left(\mathbf{y}_1 \oplus \mathbf{y}_2\right) &= \mathrm{wt}\left([1 \oplus 1 \quad 0 \oplus 1 \quad 0 \oplus 0 \quad 1 \oplus 1 \quad 0 \oplus 0 \quad 1 \oplus 1 \quad 0 \oplus 0 \quad 1 \oplus 1]\right) && (1.2)\\ &= \mathrm{wt}\left([01000000]\right) && (1.3)\\ &= 1 && (1.4)\\ &= \mathrm{dist}\left(\mathbf{y}_1, \mathbf{y}_2\right), && (1.5) \end{aligned}$$

where wt and dist are meant for "weight" and "distance", respectively.

Definition 1.1. The minimum distance of a code $\mathcal{C}$ is defined as

$$d_{\min} = \min\left\{ \mathrm{dist}\left(\mathbf{c}_i, \mathbf{c}_j\right) \middle| \, \mathbf{c}_i \in \mathcal{C}, \mathbf{c}_j \in \mathcal{C} \text{ and } \mathbf{c}_i \neq \mathbf{c}_j \right\}. \tag{1.6}$$

However, the following theorem is more practical for finding the $d_{\min}$ of binary linear codes.

Theorem 1.1. *The minimum distance equals the minimum weight of non-zero codewords.*

Proof. Let $\mathbf{c}_i$ and $\mathbf{c}_j$ be any two different codewords in code $\mathcal{C}$. Let $\mathbf{c}_k = \mathbf{c}_i \oplus \mathbf{c}_j$. Hence,

$$\mathrm{dist}\left(\mathbf{c}_i, \mathbf{c}_j\right) = \mathrm{wt}\left(\mathbf{c}_i \oplus \mathbf{c}_j\right) = \mathrm{wt}\left(\mathbf{c}_k\right). \tag{1.7}$$

Note that $\mathbf{c}_k$ is also a codeword in $\mathcal{C}$ due to its linearity. In addition, $\mathbf{c}_k \neq \mathbf{0}$, otherwise we must have $\mathbf{c}_i = \mathbf{c}_j$. Therefore, (1.6) becomes

$$d_{\min} = \min\left\{ \mathrm{wt}\left(\mathbf{c}_k\right) \middle| \, \mathbf{c}_k \in \mathcal{C} \text{ and } \mathbf{c}_k \neq \mathbf{0} \right\}. \tag{1.8}$$

□

Another commonly used consequence of the linearity is that $\mathbf{0}$ is always a codeword of any binary linear code $\mathcal{C}$, in that the addition to a codeword itself must be a codeword, as well, and

$$\mathbf{c}_i \oplus \mathbf{c}_i = \mathbf{0}. \tag{1.9}$$

The channels through which the codewords are sent are often simulated by binary symmetric or additive white Gaussian noise (AWGN) models. The former is one of the simplest channels to analyze, whereas the latter is a close model to reality. In the binary symmetric channel, it is assumed that a given bit will be "flipped" with a small probability.

To decode, the received sequence of bits will be checked against the code structure, which can be represented as a bipartite graph, introduced by Tanner [74], a small example of which is illustrated in the example below.

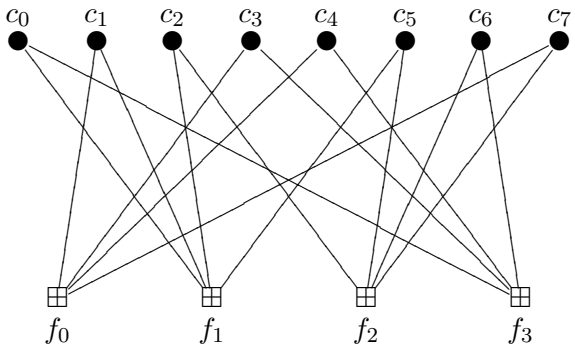


Figure 1.2: The Tanner graph of a linear block code of length 8.

Example 1.2. Figure 1.2 depicts the Tanner graph of a block code of length eight.

Each bit in a codeword here denoted by $c_i = \{0, 1\}, i = 0, 1, \ldots, 7$, corresponds to a solid node in Figure 1.2. Such nodes are called the *variable* or *bit* nodes of the Tanner graph. In order to be a codeword, the addition of all bits incident to each box $f_i, i = 0, 1, 2, 3$, called the *check* nodes, has to be 0 modulo 2, or zero under binary addition:

$$\begin{cases} c_1 \oplus c_3 \oplus c_4 \oplus c_7 = 0 & \longleftarrow f_0 \\ c_0 \oplus c_1 \oplus c_2 \oplus c_5 = 0 & \longleftarrow f_1 \\ c_2 \oplus c_5 \oplus c_6 \oplus c_7 = 0 & \longleftarrow f_2 \\ c_0 \oplus c_3 \oplus c_4 \oplus c_6 = 0 & \longleftarrow f_3 \end{cases} . \tag{1.10}$$

In other words, Figure 1.2 is merely a graphical representation of the conventional *parity-check* matrix expressions of linear block codes:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}_{4\times 8} . \tag{1.11}$$

Note that there is no connection between the variable nodes and the check nodes, hence the graph is bipartite.

For example, the received message

$$\tilde{\mathbf{y}}_1 = [1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1] \tag{1.12}$$

enables the equality

$$\mathbf{H}\tilde{\mathbf{y}}_1^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}} \tag{1.13}$$

in the binary field, which is equivalent to saying that $\tilde{\mathbf{y}}_1$ satisfies all the equality constraints in (1.10) and hence is a codeword and no decoding will be carried out.[1] If, instead,

$$\tilde{\mathbf{y}}_2 = [\tilde{y}_0 \quad \tilde{y}_1 \quad \tilde{y}_2 \quad \tilde{y}_3 \quad \tilde{y}_4 \quad \tilde{y}_5 \quad \tilde{y}_6 \quad \tilde{y}_7] \tag{1.14}$$

$$= [1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1] \tag{1.15}$$

was received, then (1.10) is not fully satisfied or

$$\mathbf{H}\tilde{\mathbf{y}}_2^{\mathrm{T}} \neq \mathbf{0}^{\mathrm{T}}, \tag{1.16}$$

[1]Note that the right hand of (1.13) is an all-0 vector.

and the receiver will start decoding in an iterative manner. To illustrate this process, the *bit-flipping* decoding algorithm [30, 31] is assumed at the decoder:

① First Step: $c \to f$. All variable nodes c_i send their received i-th bit to their neighboring check nodes f_j (two in our example).

For example, c_0 sends $\tilde{y}_0 = 1$ to f_1 and f_3, c_1 sends $\tilde{y}_1 = 1$ to f_0 and f_1, c_2 sends $\tilde{y}_2 = 0$ to f_1 and f_2, and so on.

② Step 2: $f \to c$. Every check node f_j calculates a response to each input c_i. The responding message contains the bit that f_j believes to be the correct one for this node c_i assuming that all other nodes connected to f_j are correct.

For example, f_0 receives $[c_1 \quad c_3 \quad c_4 \quad c_7] = [\tilde{y}_1 \quad \tilde{y}_3 \quad \tilde{y}_4 \quad \tilde{y}_7] = [1 \quad 1 \quad 0 \quad 1]$ which does not fulfill (1.10). As a consequence, f_0 will send 0 $(= c_3 \oplus c_4 \oplus c_7)$ back to c_1, 0 $(= c_1 \oplus c_4 \oplus c_7)$ to c_3, 1 $(= c_1 \oplus c_3 \oplus c_7)$ to c_4, and 0 $(= c_1 \oplus c_3 \oplus c_4)$ to c_7, respectively.

Note that f_2 receives $[c_2 \quad c_5 \quad c_6 \quad c_7] = [\tilde{y}_2 \quad \tilde{y}_5 \quad \tilde{y}_6 \quad \tilde{y}_7] = [0 \quad 1 \quad 0 \quad 1]$ which satisfies the third equality in (1.10). When all equalities are fulfilled at the same time, the decoding algorithm will terminate and output the values at the nodes c_i, which will be a valid codeword, although not necessarily the originally transmitted one.

③ Step 3: $c \to f$. The variable node uses the additional inputs received from its incident f_j's to decide if it needs to modify its originally received bit. One simple way to do this is via a *majority vote*. Then the nodes c_i send the "corrected" bits to the incident check nodes f_j.

In our example, $c_0 = 1$ and the variable node representing c_0 receives "suggestions" 0 and 1 from f_1 and f_3, respectively. Thus c_0 will stay as 1 by majority vote. On the other hand, $c_1 = 1$ and the variable node representing c_1 receives 0 from both f_0 and f_1. Thus c_1 will flip to 0.

④ Go to ② and repeat Steps 2 and 3.

Given enough iterations, this decoding algorithm should be able to return a codeword, although not necessarily the originally transmitted codeword since it depends on how *far off* the received message is from the original one.

To satisfy the curiosity of the readers on how this decoding illustration ends, the algorithm will terminate after two iterations and output $\mathbf{y}_2^* = [1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$ which is a one-digit-corrected codeword from the received $\tilde{\mathbf{y}}_2 = [1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$.

This concludes our demonstration of the process in this example.

As shown above, the messages passing along the edges in the Tanner graph can be as straightforward as binary digits, $\{0, 1\}$, or more generally, a probability which stands for the amount of "belief" that the bit is a "0" or a "1".

Let $\tilde{y}_i$ denote the received bit for the originally sent y_i. Then the following probabilities can be estimated using the channel characteristics:

$$\begin{aligned} \Pr_{\text{noise}}[\tilde{y}_i | y_i = 0] &= \text{The likelihood of receiving } \tilde{y}_i \text{ given } y_i = 0 \text{ was transmitted;} \\ \Pr_{\text{noise}}[\tilde{y}_i | y_i = 1] &= \text{The likelihood of receiving } \tilde{y}_i \text{ given } y_i = 1 \text{ was transmitted.} \end{aligned} \tag{1.17}$$

Thus, assuming that all codewords have equal probability,

$$y_i^* = \begin{cases} 0, \text{ if } \Pr_{\text{noise}}[\tilde{y}_i|y_i = 0] > \Pr_{\text{noise}}[\tilde{y}_i|y_i = 1] \text{ or, equivalently, } \frac{\Pr_{\text{noise}}[\tilde{y}_i|y_i=0]}{\Pr_{\text{noise}}[\tilde{y}_i|y_i=1]} > 1; \\ 1, \text{ otherwise.} \end{cases} \tag{1.18}$$

For more numerically stable computation, the likelihood ratio that appeared in (1.18) is often replaced by

$$\gamma_i = \ln\left(\frac{\Pr_{\text{noise}}[\tilde{y}_i|y_i = 0]}{\Pr_{\text{noise}}[\tilde{y}_i|y_i = 1]}\right), \tag{1.19}$$

also called the *log-likelihood ratio* (LLR), where $\ln(\cdot)$ denotes the natural logarithm, the logarithm to the base e, an irrational and transcendental constant approximately equal to 2.718281828. Then the magnitude comparison of probabilities in (1.18) translates into distinguishing the polarity of LLRs. The sign of the LLR determines whether the transmitted bit is more likely to be a "0" or a "1":

$$y_i^* = \begin{cases} 0, & \text{if } \gamma_i > 0; \\ 1, & \text{otherwise.} \end{cases} \tag{1.20}$$

These LLRs play an important role in the iterative decoding algorithms.

At each iteration, the variable nodes will aggregate the probabilities, taking the form of LLRs that they receive and then send the accumulated LLRs back to start a new iteration. Depends on the way of aggregating the LLRs, this process can be named *sum-product*, *min-sum*, *min-sum with correction factor*, etc. Compared to bit flipping, this type of decoding algorithms is called *message passing* or *belief propagation* [30, 31, 55]. To calculate the probabilities defined in (1.17), statistical knowledge of the channel has to be employed. As a consequence, message passing yields a better decoding performance than simple bit flipping.

The tradeoffs are that the belief messages flowing within the Tanner graph become more complex which induces a floating-point calculation at the decoder. From an operational point of view, floating-point calculation is impractical and had better be replaced or approximated by, for example, integers. The real axis can be partitioned into equal-size bins with each bin assigned an integer in an ascending order. The LLR falling into the range of a bin will be represented by its bin value, so that the magnitude and the sign of the LLR will be preserved to some extent. One can tell that the smaller the width of the bin, the more accurate the approximation. Also, this process can be easily translated to the binary operations that most digital devices use.

Furthermore, either this floating-point accumulated LLR or the corresponding integer representation cannot be allowed to grow indefinitely, due to implementation constraints, such as limited memory on a decoder. The LLRs will have to be *clipped* at some point, to comply with such limits. The cutoff point is usually called the *clipping threshold* of the message passing in the decoder.

In addition, also because of the computational restrictions, the algorithm has to be terminated after a number of iterations, sometimes, before a codeword is found. In this case, an incorrect decoded word with a *decoding error* will be returned by the decoder. The error rate is a critical factor to evaluate the decoding performance with respect to a decoding algorithm of a certain code on a particular channel.

1.2 Low-Density Parity-Check Codes

A low-density parity-check (LDPC) code is associated with a *sparse* parity-check matrix, denoted by $\mathbf{H}_{m\times n}$, an example of which is shown in (1.11). Every column of $\mathbf{H}$ represents a code bit, and every row constitutes a parity-check equation. The code length is n, but the number of information bits $k \geq n-m$, since $\mathbf{H}$ is not necessarily full rank. If every column and every row of $\mathbf{H}$ have the same number of non-zero elements, then we have a "regular" LDPC code, otherwise the code is irregular. A regular LDPC code can also be represented by a two-tuple (d_v, d_c), where d_v and d_c are the Hamming weights of each column and each row, respectively, together with an interleaver. The parity-check matrix (1.11) used in Example 1.2 represents a regular $(2, 4)$ LDPC code, which can also be readily seen from its corresponding Tanner graph, shown in Figure 1.2. Figure 1.3 shows the parity-check matrix of a length-155 $(3, 5)$ regular LDPC code, where non-zero elements are plotted as solid dots and zeros are left blank, a commonly used display practice when the matrix dimensions are large. We can easily see that there are three dots in each column and five in each row, with the aid of the thin line blue separation segments.

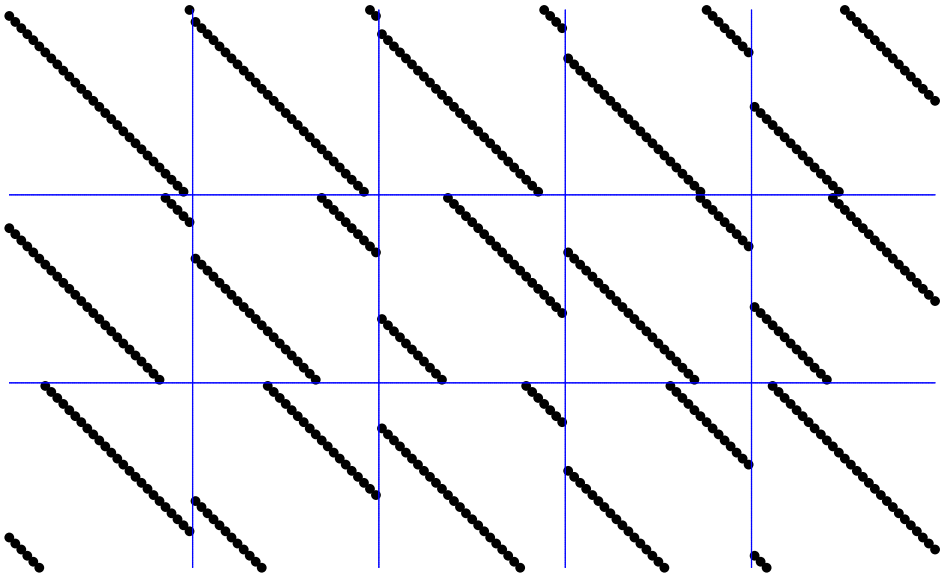


Figure 1.3: A parity-check matrix $\mathbf{H}$ with dimension 93×155.

Check nodes will correspond to the rows of the parity-check matrix, and variable nodes will correspond to the columns of the matrix. Then every non-zero entry in $\mathbf{H}$ indicates a connection between these two disjoint sets, as shown in Figure 1.2.

LDPC codes are a class of linear block codes, which provide a near-optimal performance. Their name stems from the characteristic of their sparse parity-check matrix. They were first introduced by Gallager in his PhD thesis in 1960 [30]. However, the true strength of these codes was not recognized in the beginning, due in part to a lack of sufficient computing power for their implementation at the time. Recently, much work has focussed on LDPC code construction and analysis [51, 69, 72, 47, 44, 64, 57, 6, 40, 41, 42, 2]. Some

large irregular LDPC codes have been shown to perform within 0.04 dB of the Shannon capacity [14].

In the literature, LDPC codes usually refer to block codes. However, there exists a variation of the LDPC block codes (LDPC-BCs), named LDPC convolutional codes (LDPC-CCs) [45]. The time-varying LDPC-CCs also have capacity-approaching performance. A recent decoder comparison between LDPC-BCs and LDPC-CCs conducted in [8] shows that for a fixed number of iterations, an LDPC-CC decoder achieves a better throughput/silicon area efficiency and latency than an LDPC-BC decoder, in terms of comparable BER performance, only when the LDPC-CC decoder parallelism, including both the iteration dimension and the node dimension, exceeds a certain threshold.

Today, LDPC-like codes are used and included in many current digital communications standards, such as W-CDMA, UMTS (Universal Mobile Telecommunications System), DVB (Digital Video Broadcasting), and IEEE 802.3an.

Following the convention in coding theory, LDPC block codes can also be denoted by $[n, k, d_{\min}]$-$\mathcal{C}$, where n is the codeword length, k is the code dimension, and $d_{\min}$ is the minimum Hamming distance among all codewords of $\mathcal{C}$. The minimum distance is often used to evaluate the correcting capability of the code.

There are still many open problems in LDPC coding, such as the reduction of the error floor, the avoidance of trapping sets, the design of codes and iterative decoding methods for high-speed hardware implementation. The design of a decoding algorithm is usually the most difficult part in the design of an error-correcting coding system, especially for those methods that operate close to the theoretical capacity limits.

Just as speaking louder may let you be understood better, increasing the power of a signal before transmission can help the signal overcome the channel attenuation and noise. Hence the received message will be less distorted and has a better chance of being decoded successfully. This is exactly what we observe in the error rate curves — the number of errors drops rapidly with increasing signal power. However, a further increase in power leads to a surprising effect in that the error curves tend to flatten out, regardless of the code size. This observation has troubled researchers and spawned much activity in finding an explanation and solution. The technical term for this phenomenon is called the problem of the *error floor* [58].

In the literature, it is now widely accepted that these error floors are caused by *trapping sets* [58], from particular subgraphs in the Tanner graph of the LDPC code. The iterative decoding algorithm can get locked up in these subgraphs and cause the decoder to produce one or more output errors. The iterative message-passing algorithm cannot overcome such weaknesses and gets trapped in these subgraphs, causing decoded patterns which are easily identifiable as erroneous since they are recognizable as not being valid codewords, but difficult to overcome or correct. These trapping sets are dependent on the code, the channel used, and also on the details of the operation of the decoding algorithm.

An example of a trapping set is depicted in Figure 1.4. The solid red nodes represent the trapping set members. All their neighboring parity-check nodes are shown as green (lower) and blue (upper) boxes, whereas the rest of the code bits and their connected check nodes are hidden. The set shown is the dominant trapping set [85] of a length-155 regular $(3, 5)$ LDPC code, constructed by Tanner [75, 76]. This trapping set has only eight members, while the minimum distance among the codewords of this Tanner code is 20. The decoder is much more likely to get trapped in this set than to produce a genuine codeword error.

The trapping set behaves similarly to a codeword in that when all but the set bits are

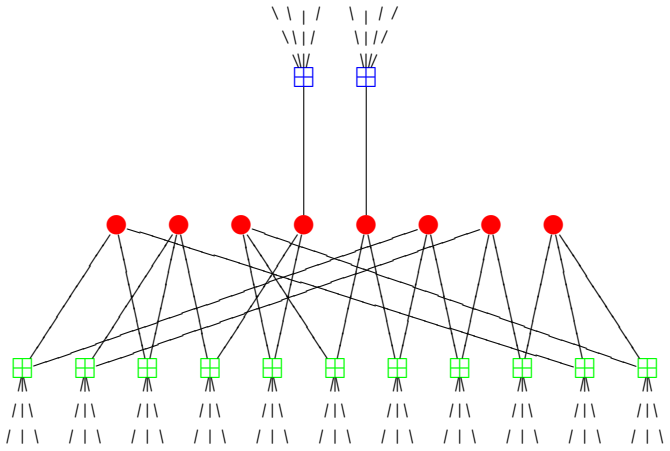


Figure 1.4: The subgraph induced by a trapping set of size-8 of a Tanner LDPC code.

"0", the green check nodes plus all the hidden ones are satisfied. Only the two blue check nodes, representing a very small fraction of the number of check equalities in (1.10), are not fulfilled. As a consequence, the satisfied green check nodes will "tell" the nodes in the set to remain in the "1" state, that is in error, even while the unsatisfied blue checks suggest otherwise, but lose to a majority vote taken at the nodes. The set will remain in error and the decoder will stay stuck. Since the size of the smallest trapping sets are usually smaller than the minimum distance between codewords [58], they tend to dominate the code performance rather than the codewords. Further, the smaller the size of a trapping set, the more impact the set has on the error floor [85, 64, 63, 62].

By enumerating the "smallest" trapping sets of a given code along with a linearized dynamics analysis, we are able to estimate the contribution to the decoding degradation from the trapping sets in the low-error region where conventional numerical methods are impractical due to excessive run times, which would take years even in simple cases [85, 64, 63, 62].

Figure 1.5 [64] shows that the error floor of the IEEE 802.3an LDPC code starts at an SNR of around 5 dB and is dominated by length-8 trapping sets. The x-axis represents the *signal-to-noise ratio*, i.e., the ratio of signal power to the background noise power, which is determined by the channel. The y-axis denotes the ratio of erroneous bits to all the bits transmitted. Note that the BER does not refer just to the information bits. The red curves are our analytical estimation based on the dynamics of these dominant trapping sets, whose subgraph is shown in Figure 1.6.

More importantly, during the process of developing our analysis, we found that the code, the channel, the decoding algorithm and, last but not least, the decoder settings play important roles in the error behavior of the decoder in the error floor regime [85].

The topology of these trapping sets is largely determined by the code structure, as can be seen from Figure 1.4, and is quite independent of the code size. Therefore, simply increasing the size of a code has only a small impact on its error floor behavior. With growing code length, finding and enumerating these trapping sets becomes a formidable combinatorial problem, making code performance analysis in the error floor regime an extremely difficult problem. Finding a generic solution to disable the entire family of the trapping sets appears to be an attractive research avenue. In our work we therefore concentrate on

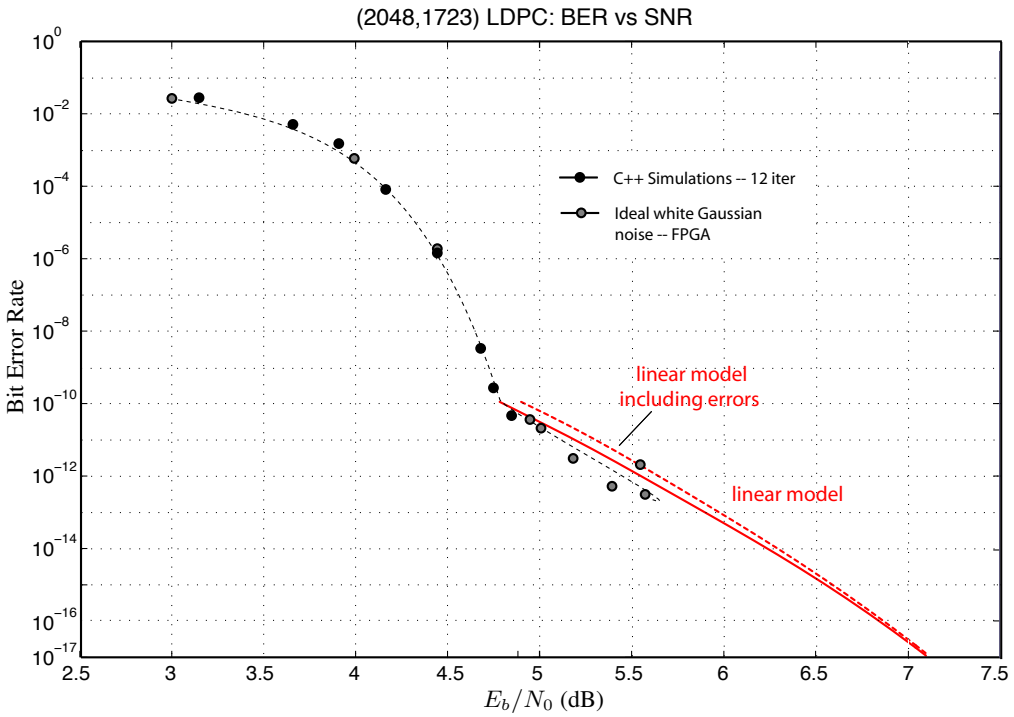


Figure 1.5: The error rate of the IEEE 802.3an LDPC code.

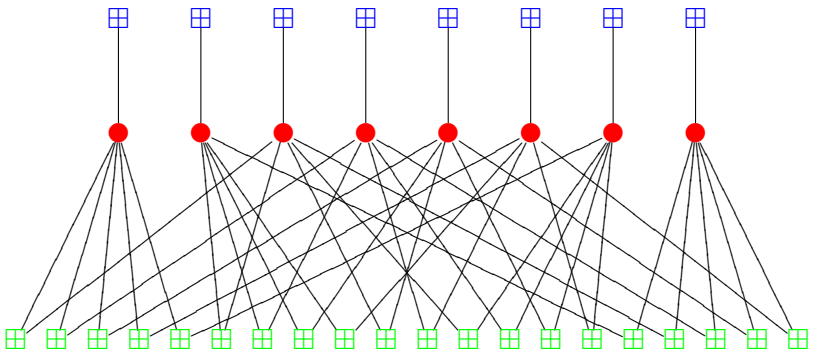


Figure 1.6: The topology of the dominant trapping set of the IEEE 802.3an LDPC code.

modifying the decoder, targeting the entire ensemble of trapping sets rather than any particular subset. Such a solution would allow us concentrate on designing codes with large distance, without worrying about the existence of trapping sets [77, 71]. Once a decoder is able to handle sub-graphs such as the ones shown in Figure 1.4 or Figure 1.6, without getting trapped and causing an error floor, there will be no need to carry out an exhaustive trapping set search [64, 24].

In fact every channel has its own characteristics and the decoding algorithm applied to it should be customized accordingly. This causes the trapping sets for each channel to be slightly different. Thus, targeting specific trapping sets for a specific channel and modifying the associated decoding algorithm is required to address the performance problem for different channels. Not surprisingly, this can also be addressed at a cost of an increase in the decoder's complexity by adding extra functionalitiy to the decoding algorithm. Good examples of post-processing targeted to eliminate the effect of trapping sets are given in [89, 46, 85].

1.3 The Influence of the Decoding Algorithm

Consider the trapping set example shown in Figure 1.4 together with the bit-flipping decoding algorithm presented earlier. It is easy to see that no matter how many iterations we run, the decoder will remain locked on the trapping set and will repeatedly cause the corresponding bit errors. Hence simply increasing the maximum iteration number at the decoder has no effect on correcting the trapping sets, but rather costs more running time since the decoder will never converge to an output from a trapping set until the decoder reaches its maximum allowed iterations and is forced to stop, which is one of the decoding stopping criteria introduced in the first section of this chapter.

The situation is much the same when we turn to decoding algorithms for more popular channels, which exchange probabilistic messages that represent the amount of likelihood of a given bit's value. Again, increasing the number of iterations alone will often have little or no success in correcting trapping sets.

In this context we began to investigate a key parameter in the decoder — the clipping threshold, which prevents the accumulated log-likelihood ratio values within the decoder from growing too large and causing numerical overflows. There is no rigid guideline for specifying how this parameter should be set. It is usually chosen as a compromise between computational effort and performance. The values chosen are typically found via extensive simulations. However, as far as the error floor is concerned, extensive simulations would be extremely time and resource-intensive.

However, in our experiments and analysis, we found that the dominance of the trapping sets in determining the error floor is highly related to this clipping threshold [85]. By allowing the messages to grow larger, the number of trapping sets at the output of the decoder drops significantly. As a result, the error rate is improved. In fact, we can theoretically show that, if the accumulated LLRs are allowed to grow unbounded in length, the "correct" probabilistic information from the blue unsatisfied checks will eventually grow big enough to break the trapping set, therefore correcting the erroneous bits in the trapping set. In other words, unlike codewords, these trapping sets should not pose a fundamental limitation to the decoding algorithm.

Besides helping the decoder conquer the trapping set failure patterns and consequently improve the error rate, setting the LLR clipping threshold high has some other conse-

quences. An apparent problem will be an increased demand on decoder memory and computational complexity, which may lead to design challenges and expenses. Another issue is the role played by the number of iterations when the decoder will no longer lock up after a finite number of iterations. To answer this question, we remind the reader of why iterative decoding was conceived in the first place: given enough iterations, the decoder should return a codeword from any given input signal. Of course, in reality, a maximum number of iterations is commonly used to strike a balance between run time and error rate. Without the obstruction from trapping sets, however, the error rates will be highly dependent on this number of iterations, and other stopping criteria may be preferable.

1.4 Importance Sampling

The error floor happens at high SNR values, and commonly the error rates are too low to be simulated using conventional methods. Hence, *importance sampling* (IS) targeting the dominant trapping sets becomes an alternative for estimating the error rate.

Importance sampling is a method to increase the number of significant events in a low event-rate testing environments such as bit error rate testing or simulation [82, 58, 36, 64, 86]. The basic principle of importance sampling is based on Monte-Carlo sampling. Specifically, in our case we wish to evaluate the probability that noise carries an original signal x_0 into the decision region of another signal, causing an error. If N_s noise samples are selected according to the channel's noise distribution, then an error estimate can be obtained as

$$\tilde{P}_{\mathrm{e}} = \frac{1}{N_s} \sum_{i=1}^{N_s} w(y_i). \tag{1.21}$$

$\tilde{P}_{\mathrm{e}}$ is an *unbiased estimate* of the true error probability

$$P_{\mathrm{e}} = \int_{\mathcal{D}} \Pr(y|x_0)\, dy, \tag{1.22}$$

where the weighting index is

$$w(y) = \begin{cases} 1, & \text{if } y \in \mathcal{D}; \\ 0, & \text{otherwise.} \end{cases} \tag{1.23}$$

$\mathcal{D}$ is the signal space area where the decoder fails to produce the correct output x_0, and $\Pr(y|x_0)$ is the conditional probability density function of the received signal y given the transmitted signal x_0, which is related to the noise distribution, for example, the Gaussian noise density in our case.

Unfortunately, for low values of P_{e}, one has to generate in the order of $10/P_{\mathrm{e}}$ samples to obtain statistically relevant numbers of error events. For instance, the error floor of the IEEE 802.3an LDPC code appears below a bit error rate of 10^{-10}, as shown in Figure 1.5, which requires 10^{11} to 10^{12} samples to be simulated and tested.

One way of increasing the number of error events, or positive counts, is to distort the noise distribution to cause more errors. This is typically done by shifting the mean of the noise towards a convenient boundary of $\mathcal{D}$ (mean-shift importance sampling) [82, 58].

Since trapping sets are examined as the primary causes of the significant events in the error floor region, we add such a mean shift, or *bias*, to the members belonging to both x_0 and a dominant trapping set. Having identified and enumerated the dominant trapping

sets of an LDPC code, we technically need to do this for each trapping set separately, but symmetries can often be exploited in reducing this task. Gaussian-distributed noise is added to the biased codeword x_0. As a result, the biased received y will have an increased chance of causing an error. Consequently the sample size N_s can be significantly reduced, which translates into what is called "the gain" of importance sampling.

Taking into account the bias value and the dominant trapping set selected in the operation, the estimation formula (1.21) along with the weighting factor (1.23) must be adjusted accordingly, leading to a weight term $w(y) \ll 1$. The combined effect of measuring more significant events and ascribing them lower weight will produce the same error rate measure in (1.21) if the shifting is done correctly. In Chapter 3, Figure 3.12 shows the results of importance sampling for the IEEE 802.3an code compared to regular (no-bias) Monte-Carlo simulations which are also presented in Figure 1.5. As a word of warning: it is common for importance sampling results to become "biased" and cause erroneous error measurements, and therefore extreme caution is needed.

1.5 Two Example LDPC Codes

The IEEE 802.3an length-$2,048$ LDPC code has been widely used as a "poster boy" example code in error floor research [88, 89, 64]. One reason is because its error curve clearly demonstrates a turning point from waterfall to error floor, as shown in Figure 1.5. However, it is almost impossible to simulate bit error rates of 10^{-10} and below.

On the other hand, the length-155 Tanner code used in our examination does not create major difficulties in simulating its error floor, which enables us to plot error floor statistics in the following chapters to understand the dominance the trapping sets and verify analytical approaches. However, a short code like this has a very "weak" error floor which is not easily visually identifiable from the waterfall region.

Both codes have an error floor caused by trapping sets. By studying their dominant trapping sets, their error floors can be estimated using a linearized analysis. The accuracy of this analysis is confirmed by importance sampling. We identified the LLR range at the decoder as a major trigger of trapping sets. These errors will all go away when the LLR clipping value is set high enough, and given a sufficient number of iterations.

1.6 Organization of this Thesis

We started by introducing the characteristics of some classical decoding algorithms of LDPC codes on different channel models in Chapter 2. The decoders most often used for LDPC codes are based on message-passing (MP), where messages are iteratively sent across a bipartite graph of the code. While the performance of message-passing is excellent in practice, analyzing its behavior is often difficult particularly due to cycles in the code's graph.

Because of inherent structural weakness of LDPC codes, with increasing SNR, the error probability does not fall as rapidly as expected, after some point. At small SNR, the error probability decreases rapidly with the SNR and the curve forms the so-called *waterfall* region. The decrease slows down at moderate values turning into the *error floor* asymptotic at very large SNR. Under MP decoding, the underlying reason for error floors is the existence of *trapping sets* [58]. In Chapter 3 we provide a linear analysis based on the minimal trapping sets of LDPC codes to accurately predict the error floor. This is useful especially when

the code has a low error probability at high SNR and software or even hardware simulations are no longer practical. Two LDPC codes are showcased in this chapter. One code is widely recognized as a good-length error correcting code and utilized in one of the IEEE standards. The other has fewer code bits that enable the software simulation and error pattern statistics in the high SNR region. The enumeration process of the absorption sets of the two codes is systematic but a bit tedious, thus it is presented in two appendix chapters, respectively.

Once we understood that how the primary decoding failures are contributing the error floor, Chapter 4 describes an effort to eliminate such a contribution. As we carefully examined our estimation formula that was derived in Chapter 3 and accurately approximates the error floors, we find out that code construction, channel properties, the decoding algorithm and decoder settings are all responsible for the formation of the absorption sets as dominant decoding error patterns. By adjusting the decoder settings, LLR clipping threshold and maximum iteration number to be exact, which are configured to reduce the computational complexity of the decoding algorithm, we show that the error floors of the two LDPC codes showcased in Chapter 3 can be lowered very effectively.

Lastly, the Linear programming (LP) decoding technique, an alternative decoding method proposed recently, is believed to be able to achieve better performance than message-passing decoding since it is closely related to the optimal maximum-likelihood decoding. However the connection between message-passing and linear programming decoding techniques is still being studied [11]. Regarding LP decoding, *pseudocodewords* are the main reason that cause decoding failure. A recent numerical result [13] implies that there is some connection between pseudocodewords and trapping sets, which is examined at the end of Chapter 3. The explicit link between those two is as yet unknown. Proposed work is described in Chapter 5.

Chapter 2

LDPC Decoding Algorithms

An LDPC code is defined by a parity-check matrix:

$$\mathbf{H}_{m\times n} = [h_{ji}], \text{ where } h_{ji} \in \{0,1\}, j = 1,2,\ldots,m \text{ and } i = 1,2,\ldots,n, \tag{2.1}$$

where n is the codeword length and m is the number of parity-check equations.

If every row and every column of $\mathbf{H}$ has the same number of 1's, respectively, it is called a *regular* LDPC code, otherwise the code is *irregular*. Let d_v denote the number of 1's in each column and d_c denote the number of 1's in each row of a regular LDPC code. They are called *variable node degree* and *check node degree* of the LDPC code, respectively. A regular LDPC code can also be denoted as a (d_v, d_c) LDPC code.

In 1981, Tanner [74] introduced a bipartite graphical representation for LDPC codes, now called a *Tanner graph*. The nodes of the graph are separated into two distinct sets, named variable nodes set and check nodes set, and edges only connect nodes of two different types. The Tanner graph consists of m check nodes, each corresponding to one parity-check equation, and n variable nodes, corresponding to the codeword bits. There is an edge connecting check node j and variable node i, whenever $h_{ji} = 1$.

Note that message-passing decoding is based on the Tanner graph, whereas linear programming decoding is not. Let us see how they work one by one.

2.1 Iterative Decoding

Although *maximum-likelihood* decoding is optimal, it is too complex to implement for codes of useful length. On the other hand, iterative decoding performs extremely well. The idea of iterative decoding is that each check node combines all the information sent to it and returns a likelihood/suggestion back to the variable nodes connected to it. The variable nodes can either make a decision of the incoming messages, for example, a majority vote, or add them up and send them back out to the check nodes for another iteration cycle. The process will stop when all check nodes are satisfied, i.e., a valid codeword is encountered, or the decoder reaches a maximum number of iterations allowed.

Decoding is monotonic in that the more iterations the decoder runs, the better the performance. But monotonicity is true only for cycle-free codes. When the codes have cycles, the messages passing along the edges become dependent after a few iterations, which degrades the final decoding performance. Therefore, constructing a code with large girth became one option to improve performance. However, we pursue another direction which is not linked

to designing the code. We instead modify the decoding algorithm to achieve a better error rate.

The algorithm used to decode LDPC codes was discovered independently several times and as a matter of fact comes under different names. The most common ones are the *message-passing* algorithm, or the *belief-propagation* algorithm.

Let $I = \{i : i = 1, 2, \ldots, n\}$ and $J = \{j : j = 1, 2, \ldots, m\}$ be the set of variable nodes and the set of check nodes, respectively. For each $j \in J$, define $V_j = \{i : i = 1, 2, \ldots, n \text{ and } h_{ji} = 1\}$ as the set of neighbor variable nodes of j, i.e., all variable nodes connected to j. And for each $i \in I$, define $C_i = \{j : j = 1, 2, \ldots, m \text{ and } h_{ji} = 1\}$ as the set of neighbor check nodes of i, i.e., all check nodes connected to i.

We start with the hard-decision bit-flipping decoding algorithms prior to considering the more complex soft-decision message-passing decoding.

2.1.1 LDPC Decoding Algorithm on BEC

The binary erasure channel is, in some sense, error-free. When the receiver gets a bit, it is 100% certain that the bit is correct. The only confusion arises when the bit is erased. If $\tilde{y_i} = \varepsilon$ then the received symbol i has been erased and the variable node i is said to be *unknown*. The decoding algorithm will try to determine whether i should be 0 or 1 based on the knowledge of other variable nodes [50]. An illustration of the BEC is shown in Figure 2.1, where X and Y denote the sets of input and output symbols, respectively.

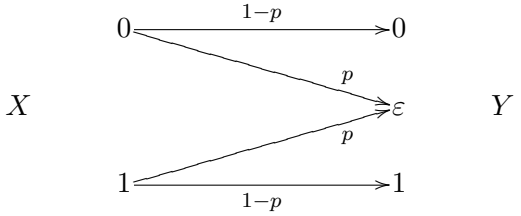


Figure 2.1: A binary erasure channel with probability of erasure p.

Step 1. Initialize

$$d_i = r_i = 1 - 2\tilde{y_i} = \begin{cases} 1, & \text{if } \tilde{y_i} = 0; \\ 0, & \text{if } \tilde{y_i} = \varepsilon; \\ -1, & \text{if } \tilde{y_i} = 1, \end{cases} \tag{2.2}$$

for each variable node $i \in I$.

Step 2. All variable nodes send

$$\mu_{i \to j} = d_i \tag{2.3}$$

to each check node $j \in C_i$.

Step 3. Check nodes connected to variable node i send

$$\beta_{j \to i} = \prod_{l \in V_j \setminus \{i\}} \mu_{l \to j} \tag{2.4}$$

to variable node i. That is, if all incoming messages are different from 0, the check node sends back to i the value that makes the check consistent, otherwise it sends back a 0 for "unknown".

Step 4. At the variable node i, if i is unknown and at least one $\beta_{j\to i} \neq 0$, then set $d_i = \beta_{j\to i}$ and declare i to be *known*. Note that, under BEC, the values of all non-zero $\beta_{j\to i}$'s will be identical.

Step 5. Stop when all variable nodes are known or after a fixed number of iterations have been executed. Otherwise go back to Step 2.

2.1.2 Gallager's LDPC Decoding Algorithm A for BSC

The binary symmetric channel is a frequently used model for a noisy transmission medium, where the binary input bits are flipped with a small probability, called *crossover probability*. An illustration of the BSC is shown in Figure 2.2. The BEC, like the BSC, is popular in information theory and coding theory for its simplicity.

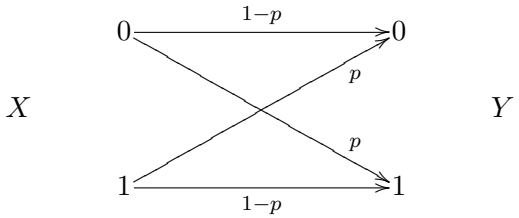


Figure 2.2: A binary symmetric channel with crossover probability p.

Step 1. Initialize

$$d_i = \tilde{y}_i \tag{2.5}$$

for each variable node $i \in I$.

Step 2. All variable nodes send

$$\mu_{i\to j} = d_i \tag{2.6}$$

to each check node $j \in C_i$.

Step 3. Check nodes connected to variable node i send

$$\beta_{j\to i} = \prod_{l\in V_j\setminus\{i\}} \mu_{l\to j} \tag{2.7}$$

to variable node i. That is, the check node sends back to i the value that would make the parity check satisfied.

Step 4. At the variable node i, if $\left\lceil \frac{d_v}{2} \right\rceil$ or more of the incoming messages $\beta_{j\to i}$ disagree with d_i, change the value of variable node i to its opposite value, i.e., $d_i = d_i \oplus 1$.

Step 5. Stop when no more variables are changing, or after a fixed number of iterations have been executed. Otherwise go back to Step 2.

Note that, in Step 4, a *majority vote* is taken at the variable node to make it satisfy the majority check nodes.

2.1.3 Message-Passing Algorithm on AWGN

The additive white Gaussian noise (AWGN) channel is the most important model of practical channels, especially for wireless communications. The output of an AWGN channel model is the sum of the input signal and a Gaussian (or normal) distributed noise.

$$\tilde{y_i} = y_i + n_i, \tag{2.8}$$

where the noise samples are independently and normally distributed:

$$n_i \sim \mathcal{N}\left(0, \sigma^2\right). \tag{2.9}$$

It approximates a "real" environment better than the BEC and the BSC do.

Now we get to the message-passing decoding of LDPC codes on AWGN channels:

Step 1. Initialize

$$\lambda_i = \frac{2\tilde{y_i}}{\sigma^2} \tag{2.10}$$

for each variable node $i \in I$.

Step 2. All variable nodes send

$$\mu_{i\to j} = \lambda_i \tag{2.11}$$

to each check node $j \in C_i$.

Step 3. Check nodes connected to variable node i send

$$\beta_{j\to i} = 2\tanh^{-1}\left(\prod_{l\in V_j\setminus\{i\}} \tanh\left(\frac{\mu_{l\to j}}{2}\right)\right) \tag{2.12}$$

to variable node i.

Step 4. Variable nodes connected to check nodes j send

$$\mu_{i\to j} = \sum_{l\in C_i\setminus\{j\}} \beta_{l\to j} \tag{2.13}$$

to j.

Step 5. When a fixed number of iterations have been completed or the estimated codeword $\mathbf{y}^*$ satisfies $\mathbf{H}\mathbf{y}^{*\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$, output $\mathbf{y}^*$ as the decoded codeword and stop. Otherwise go back to Step 3.

All values in this algorithm are within the log-domain, so-called LLRs. Let us show how those LLRs are derived.

Similar to what we did in Section 2.1.1, let us map y_i from $\{0, 1\}$ to $\{1, -1\}$:

$$y_i = 1 - 2y_i = \begin{cases} 1, & \text{if } y_i = 0; \\ -1, & \text{if } y_i = 1. \end{cases} \qquad i = 1, 2, \ldots, n. \tag{2.14}$$

Note that

$$\Pr_{\text{noise}}[\tilde{y_i}|y_i] \quad = \quad \Pr_{\text{noise}}[y_i + n_i|y_i] \tag{2.15}$$

$$= \Pr_{\text{noise}}[n_i] \tag{2.16}$$
$$= \Pr_{\text{noise}}[\tilde{y_i} - y_i] \tag{2.17}$$
$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\tilde{y_i} - y_i)^2}{2\sigma^2}\right). \tag{2.18}$$

Then this LLR

$$\ln\left(\frac{\Pr_{\text{noise}}[\tilde{y_i}|y_i = 1]}{\Pr_{\text{noise}}[\tilde{y_i}|y_i = -1]}\right) = \ln\left(\frac{\exp\left(-\frac{(\tilde{y_i}-1)^2}{2\sigma^2}\right)}{\exp\left(-\frac{(\tilde{y_i}+1)^2}{2\sigma^2}\right)}\right) \tag{2.19}$$
$$= -\frac{(\tilde{y_i} - 1)^2}{2\sigma^2} + \frac{(\tilde{y_i} + 1)^2}{2\sigma^2} \tag{2.20}$$
$$= \frac{2\tilde{y_i}}{\sigma^2} \tag{2.21}$$

as given in (2.10).

To show how to get (2.12), we shall go back to the probability domain and define

$$r_{ji}(b) = \Pr\left[\text{check node } j \text{ is satisfied}|y_i = b \text{ and all messages } \mu_{l\to j} \text{ where } l \in V_j \backslash \{i\}\right], \tag{2.22}$$

where $b \in \{-1, 1\}$. Therefore

$$\beta_{j\to i} = \ln\left(\frac{r_{ji}(1)}{r_{ji}(-1)}\right). \tag{2.23}$$

In the same way, $\mu_{i\to j}$ in (2.13) can be written as

$$\mu_{i\to j} = \ln\left(\frac{q_{ij}(1)}{q_{ij}(-1)}\right). \tag{2.24}$$

To connect $r_{ji}(b)$ and $q_{ij}(b)$, Gallager has showed [30]:

Lemma 2.1. *Consider a sequence of M independent binary digits a_i for which $\Pr[a_i = 1] = p_i$. Then*

$$\Pr\left[\{a_i\}_{i=1}^{M} \textit{ contains an even number of } 1\textit{'s}\right] = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{M}(1 - 2p_i). \tag{2.25}$$

Proof. By induction on M,

$M = 1$: $P_1 = 1 - p_1$;

$M = 2$: $P_2 = (1 - p_1)(1 - p_2) + p_1p_2$;

$M = 3$: $P_3 = (1 - p_1)(1 - p_2)(1 - p_3) + p_1p_2(1 - p_3) + p_1(1 - p_2)p_3 + (1 - p_1)p_2p_3$;

$M = k$: Assume $P_k = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{k}(1 - 2p_i)$;

$M = k + 1$: $P_{k+1} = (1 - p_{k+1})P_k + p_{k+1}(1 - P_k) = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{k+1}(1 - 2p_i)$.

□

In addition, with the correspondence between p_i and $q_{ij}(-1)$, we have

$$r_{ji}(1) = \frac{1}{2} + \frac{1}{2} \prod_{l \in V_j \backslash \{i\}} (1 - 2q_{lj}(-1)) \,. \tag{2.26}$$

In other words, when $y_i = 1$ (representing 0), the variable nodes $\{l \in V_j \backslash \{i\}\}$ must contain an even number of -1's in order to satisfy the check node j. By noting that

$$\begin{array}{l} r_{ji}(1) = 1 - r_{ji}(-1) \\ q_{ij}(1) = 1 - q_{ij}(-1) \end{array} , \tag{2.27}$$

(2.26) becomes:

$$r_{ji}(1) = \frac{1}{2} + \frac{1}{2} \prod_{l \in V_j \backslash \{i\}} (q_{ij}(1) + q_{ij}(-1) - 2q_{lj}(-1)) \tag{2.28}$$

$$\Rightarrow \quad 2r_{ji}(1) = 1 + \prod_{l \in V_j \backslash \{i\}} (q_{ij}(1) - q_{lj}(-1)) \tag{2.29}$$

$$\Rightarrow \quad 2r_{ji}(1) = r_{ji}(1) + r_{ji}(-1) + \prod_{l \in V_j \backslash \{i\}} (q_{ij}(1) - q_{lj}(-1)) \tag{2.30}$$

$$\Rightarrow \quad r_{ji}(1) - r_{ji}(-1) = \prod_{l \in V_j \backslash \{i\}} (q_{lj}(1) - q_{lj}(-1)) \,. \tag{2.31}$$

Due to the fact that

$$\tanh\left(\frac{1}{2}\ln\left(\frac{a}{b}\right)\right) = \frac{\left(\frac{a}{b}\right)^{\frac{1}{2}} - \left(\frac{a}{b}\right)^{-\frac{1}{2}}}{\left(\frac{a}{b}\right)^{\frac{1}{2}} + \left(\frac{a}{b}\right)^{-\frac{1}{2}}} \tag{2.32}$$

$$= \frac{a-b}{a+b}, \tag{2.33}$$

we can substitute (2.31) by

$$r_{ji}(1) - r_{ji}(-1) = (r_{ji}(1) + r_{ji}(-1)) \tanh\left(\frac{1}{2}\ln\left(\frac{r_{ji}(1)}{r_{ji}(-1)}\right)\right) \tag{2.34}$$

$$= \tanh\left(\frac{1}{2}\beta_{j\to i}\right) \tag{2.35}$$

$$q_{ij}(1) - q_{ij}(-1) = (q_{ij}(1) + q_{ij}(-1)) \tanh\left(\frac{1}{2}\ln\left(\frac{q_{ij}(1)}{q_{ij}(-1)}\right)\right) \tag{2.36}$$

$$= \tanh\left(\frac{1}{2}\mu_{i\to j}\right) \tag{2.37}$$

to get (2.12), finally.

The $\tanh$ rule is preferred in analysis due to its conceptual simplicity. However, in order to avoid the multiplication in the implication, (2.12) can be equivalently expressed as a sign-adjusted sum [4]:

$$\beta_{j\to i} = 2\tanh^{-1}\left(\prod_{l \in V_j \backslash \{i\}} \tanh\left(\frac{\mu_{l\to j}}{2}\right)\right)$$

$$= \left(\prod_{l \in V_j \setminus \{i\}} \operatorname{sign}(\mu_{l \to j}) \right) \cdot f \left(\sum_{l \in V_j \setminus \{i\}} f\left(|\mu_{l \to j}| \right) \right), \tag{2.38}$$

where

$$\operatorname{sign}(x) = \begin{cases} 1, & \text{if } x > 0; \\ 0, & \text{if } x = 0; \\ -1, & \text{if } x < 0, \end{cases} \tag{2.39}$$

and

$$\begin{aligned} f(x) &= -\ln\left(\tanh\left(\frac{x}{2}\right)\right) && (2.40) \\ &= \ln\left(\frac{e^x + 1}{e^x - 1}\right) && (2.41) \\ &= \ln\left(1 + \frac{2}{e^x - 1}\right), && (2.42) \end{aligned}$$

where none of the incoming LLR $\mu_{l \to j}$ can be zero.

Proof. First we have

$$\tanh\left(\frac{y}{2}\right) = \prod_{i=1}^{n} \tanh\left(\frac{x_i}{2}\right). \tag{2.43}$$

Let $y = \operatorname{sign}(y) \cdot |y|$ and $x_i = \operatorname{sign}(x_i) \cdot |x_i|$. Since $\tanh(x)$ is an odd function on its domain, we obtain

$$\begin{aligned} \operatorname{sign}(y) &= \prod_{i=1}^{n} \operatorname{sign}(x_i) && (2.44) \\ \tanh\left(\frac{|y|}{2}\right) &= \prod_{i=1}^{n} \tanh\left(\frac{|x_i|}{2}\right). && (2.45) \end{aligned}$$

Taking $-\ln(\cdot)$ of both sides of (2.45) yields:

$$f\left(|y|\right) = \sum_{i=1}^{n} f\left(|x_i|\right), \tag{2.46}$$

where $f(x)$ is given by (2.40). Due to the fact that

$$\begin{aligned} f\left(f(x)\right) &= \ln\left(1 + \frac{2}{e^{\ln\left(1 + \frac{2}{e^x - 1}\right)} - 1}\right) && (2.47) \\ &= \ln\left(1 + \frac{2}{\left(1 + \frac{2}{e^x - 1}\right) - 1}\right) && (2.48) \\ &= \ln\left(1 + (e^x - 1)\right) && (2.49) \\ &= x && (2.50) \end{aligned}$$

for all $x > 0$, we can apply $f(\cdot)$ to both sides of (2.46), yielding:

$$|y| = f\left(\sum_{i=1}^{n} f\left(|x_i|\right)\right), \tag{2.51}$$

which, after combining with (2.44), gives (2.38). □

Furthermore, to avoid the complex $\tanh(\cdot)$ calculation, we first note that the message with the minimum magnitude will dominate the $\tanh(\cdot)$ value. Some approximations have therefore been proposed, such as *min-sum* decoding algorithm (and min-sum with correction factors) [81] [39] [61], in which the messages sent to variable nodes from check nodes are approximated by:

$$\beta_{j\to i} = \min_{l\in V_j\backslash\{i\}} (|\mu_{l\to j}|) \prod_{l\in V_j\backslash\{i\}} \text{sign}(\mu_{l\to j}). \tag{2.52}$$

The message-passing algorithm does really well on AWGN channels and has numerous practical applications. It is the algorithm that is almost exclusively used in practical decoders.

2.2 Linear Programming Decoding

To motivate the *linear programming decoding* of LDPC codes, we will start with the optimal decoding method, i.e., *maximum-likelihood* (ML) decoding. We then show that it results in an equivalent linear programming (LP) problem. Therefore LP solvers can be applied to the decoding of LDPC codes.

One disadvantage of LP decoding is that it may output real-valued solutions, while only integer solutions are meaningful. But restricting all LP variables to be integers does not help at all in the sense of complexity. A common strategy to mitigate this effect is to remove the integer restriction. This approach is called *linear programming relaxation*. After that, we will see how LP decoding works.

2.2.1 Maximum-Likelihood Decoding

Assume that all codewords have equal probability, then

Definition 2.1. ([43]) Maximum-likelihood decoding aims to find the codeword $\mathbf{y}^*$ that maximizes the likelihood of what was received from the channel $\tilde{\mathbf{y}}$, given a codeword $\mathbf{y}$ was transmitted:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in\mathcal{C}} \Pr_{\text{noise}}[\tilde{\mathbf{y}} \text{ received}|\mathbf{y} \text{ transmitted}]. \tag{2.53}$$

ML decoding is optimal but NP-hard [52]. Therefore, we hope to find an alternative sub-optimal but still efficient decoding method to approximate optimal ML decoding. The following theorem motivated the idea of applying linear programming to decoding.

Theorem 2.2. *([25]) For any binary-input memoryless channel, ML decoding is equivalent to finding an optimal solution that minimizes a linear function of the codeword.*

Proof.

$$\begin{aligned}
\mathbf{y}^* &= \arg\max_{\mathbf{y}\in\mathcal{C}} \Pr_{\text{noise}}[\tilde{\mathbf{y}} \text{ received}|\mathbf{y} \text{ transmitted}] && (2.54)\\
&= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\prod_{i=1}^{n} \Pr_{\text{noise}}[\tilde{y}_i|y_i]\right) && (2.55)\\
&= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\ln\left(\prod_{i=1}^{n} \Pr_{\text{noise}}[\tilde{y}_i|y_i]\right)\right) && (2.56)
\end{aligned}$$

$$= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i=1}^{n} \ln \left(\Pr_{\text{noise}} [\tilde{y}_i | y_i] \right) \right) \tag{2.57}$$

$$= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i=1}^{n} \ln \left(\Pr_{\text{noise}} [\tilde{y}_i | y_i] \right) - \sum_{i=1}^{n} \ln \left(\Pr_{\text{noise}} [\tilde{y}_i | 0] \right) \right) \tag{2.58}$$

$$= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i=1}^{n} \left(\ln \left(\Pr_{\text{noise}} [\tilde{y}_i | y_i] \right) - \ln \left(\Pr_{\text{noise}} [\tilde{y}_i | 0] \right) \right) \right) \tag{2.59}$$

$$= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i:y_i=1} \left(\ln \left(\Pr_{\text{noise}} [\tilde{y}_i | y_i] \right) - \ln \left(\Pr_{\text{noise}} [\tilde{y}_i | 0] \right) \right) \right) \tag{2.60}$$

$$= \arg\max_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i:y_i=1} \ln \left(\frac{\Pr_{\text{noise}}[\tilde{y}_i | y_i = 1]}{\Pr_{\text{noise}}[\tilde{y}_i | y_i = 0]} \right) \right) \tag{2.61}$$

$$= \arg\min_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i:y_i=1} \ln \left(\frac{\Pr_{\text{noise}}[\tilde{y}_i | y_i = 0]}{\Pr_{\text{noise}}[\tilde{y}_i | y_i = 1]} \right) \right) \tag{2.62}$$

$$= \arg\min_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i:y_i=1} \gamma_i \right) \tag{2.63}$$

$$= \arg\min_{\mathbf{y}\in\mathcal{C}} \left(\sum_{i=1}^{n} \gamma_i y_i \right). \tag{2.64}$$

(2.55) is due to the memoryless property of the channel. Next (2.56) follows from the fact that $\ln(\cdot)$ is a strictly increasing function. Subtracting a constant term $\sum_{i=1}^{n} \ln \left(\Pr_{\text{noise}} [\tilde{y}_i | 0] \right)$ from (2.57) does not compromise the argument. Canceling the identical terms in (2.59) gives us the summation in (2.60). Flipping the ratio in (2.61) results in a minimizing argument (2.62). By (1.19), the summation in (2.62) becomes a summation of γ_i over all i where $y_i = 1$. This summation can be extended over all $i \in [1, n]$ in (2.64), since y_i's are either 0 or 1. □

2.2.2 Linear Programming

In mathematics, linear programming problems involve the optimization of a linear *objective function*, subject to linear equality and inequality *constraint functions* ([65]). These constraint functions define the domain of the objective function. Regarding LP, this domain is also called a *polytope*, denoted by P. Every LP problem has an associated polytope and the goal is to find a point in the polytope where the objective function achieves its extreme value. Further, there always exists a solution to an LP problem and the solution is always obtained on a *vertex* of its polytope ([65]). So the set of the vertices, denoted by $V(P)$, is the feasible solution space.

In (2.64), we have a linear objective function. The question then becomes how to adapt the constraint $\mathbf{y} \in \mathcal{C}$ into linear constraints. One straightforward way is to restrict all variables to be integers, then we obtain an *integer linear programming* (ILP) problem. Unfortunately, ILP is NP-hard [32] and does not have straightforward efficient solution methods.

Even if we require all variables to be either 0 or 1, which is called *binary integer programming*, a special case of ILP, it is still NP-hard. Therefore, the next feasible approach seems to be simply removing the integer restrictions, for example, by allowing the variables to belong to the real interval $[0, 1]$ instead of just 0 or 1, and then solving the resulting LP. This technique is called *linear programming relaxation*.

- ILP

$$\begin{aligned} &\text{Objective function:} && \min \sum_{i=1}^{n} \gamma_i y_i \\ &\text{Constraints:} && \mathbf{y} \in \mathcal{C} \end{aligned} \tag{2.65}$$

- LP relaxation

$$\begin{aligned} &\text{Objective function:} && \min \sum_{i=1}^{n} \gamma_i y_i \\ &\text{Constraints (polytope } P\text{):} && 0 \leq y_i \leq 1 \\ & && \mathbf{y} \in V(P) \end{aligned} \tag{2.66}$$

In ILP, the solution space is exactly the set of codewords. In LP relaxation, the solution space is extended to real values. In other words, we enlarge the original domain to make the problem solvable. As a consequence, the set of vertices (solutions) is enlarged, as well. We may have both *integral* and *fractional* vertices. Even worse, some of the integral vertices may not be valid codewords. In the following section, we will see, however, that every integral vertex of the LP decoding polytope still corresponds to a codeword. But the LP decoding polytope also contains fractional vertices, which are potential outputs of the decoder, so we call *all* vertices of the LP decoding polytope as *pseudocodewords*.

2.2.3 Linear Programming Decoding for LDPC Codes

After we explore the connection between LP and decoding, let us study the performance of LP decoding for LDPC codes, by looking at their polytope.

A codeword $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ must satisfy

$$\mathbf{H}\mathbf{y}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}. \tag{2.67}$$

In other words, we can solve this system of binary linear equations (2.67) to find a codeword $\mathbf{y}$. There are m equations in this system and we can solve them one by one. If we call the solutions to one equation as *local solutions*, then the solutions to the entire system (2.67), named *global solutions*, will be the intersection of all local solutions.

Consider one equation from (2.67) now. There are d_c binary variables involved. If an even number of these d_c variables equal 1 and the rest of them are equal to 0, then this is a solution to this equation. Note that there are n variables, and the ones not involved in this equation could be either 0 or 1. Then mathematically, for each parity-check equation j we can enumerate the local solutions as follows.

Let E_j be the set of all subsets of V_j with even cardinality:

$$E_j = \{S \subseteq V_j : |S| \text{ even}\},\ \forall j \in J. \tag{2.68}$$

V_j contains the d_c variables in the j-th equation of (2.67). Then for each S in E_j:

$$y_i = \begin{cases} 1, & \text{if } i \in S; \\ 0, & \text{if } i \in V_j \backslash S. \end{cases} \qquad \forall i \in V_j \tag{2.69}$$

constitutes a local solution. So (2.69) will generate $|E_j|$ local solutions.

In LP terminology, each local solution defines a *local polytope*. Likewise, the global solution defines the *global polytope*, which will be the intersection of all local polytopes. We start by building the local polytope [25]:

1. Let $w_{j,S}$ indicate that the local solution to check node j uses configuration $S \in E_j$:

$$w_{j,S} = \begin{cases} 1, & \text{use the configuration } S; \\ 0, & \text{not use the configuration } S. \end{cases} \tag{2.70}$$

   In addition, we need to keep in mind that we are eventually trying to enumerate all global solutions. One potential global solution will be the intersection of *one* S from each j, i.e., each check node is satisfied. For a given global solution, only *one* particular S can be in effect at node i. Hence by (2.70):

$$\sum_{S \in E_j} w_{j,S} = 1. \tag{2.71}$$

2. Let $\mathbf{f} = [f_1, f_2, \ldots, f_n]$ be the relaxed version of $\mathbf{y}$:

$$0 \le f_i \le 1, \quad i = 1, 2, \ldots, n. \tag{2.72}$$

3. Obviously, $w_{j,S}$ and f_i are dependent of each other. The value of f_i at each variable node i must be consistent with the point in the local polytope defined by $\mathbf{w} = [w_{j,S_1}, w_{j,S_2}, \ldots]$, where $S_k \in E_j$ for check node j. So

$$f_i = \sum_{S \in E_j, S \ni i} w_{j,S}, \quad \forall i \in V_j. \tag{2.73}$$

   To interpret this, if $f_i = 1$, then one and only one of $S \subseteq E_j$ containing i must be in effect, where "in effect" means $w_{j,S} = 1$. If $f_i = 0$, then none of $S \subseteq E_j$ containing i should be in effect.

To avoid a *mixed integer linear programming* problem, we relax the indicator variables (2.70) as well:

$$0 \le w_{j,S} \le 1, \quad \forall S \in E_j. \tag{2.74}$$

By combining (2.72), (2.74), (2.71) and (2.73) altogether as our constraint functions, we have the local LP polytope:

$$Q_j = \left\{ (\mathbf{f}, \mathbf{w}) \;\middle|\; \begin{array}{ll} 0 \le f_i \le 1, & \forall i \in I; \\ 0 \le w_{j,S} \le 1, & \forall S \in E_j; \\ \displaystyle\sum_{S \in E_j} w_{j,S} = 1; & \\ f_i = \displaystyle\sum_{S \in E_j, S \ni i} w_{j,S}, & \forall i \in V_j. \end{array} \right\}, \quad j = 1, 2, \ldots, m. \tag{2.75}$$

The global polytope is then

$$Q = \bigcap_{j \in J} Q_j. \tag{2.76}$$

Finally, our LP problem for LDPC codes becomes

$$\min \sum_{i=1}^{n} \gamma_i f_i \quad \text{s.t.} \quad (\mathbf{f}, \mathbf{w}) \in Q. \tag{2.77}$$

We should point out that in the literature, two fast general-purpose LP optimization algorithms are widely used. One is called the *Simplex* method, practical but in the worst-case is greater than polynomial-time [70]; the other is named *Ellipsoid*, provably polynomial-time but slow in practice [34]. Ongoing research focusses on customizing the general-purpose algorithms to the decoding problem by making more use of the code structure.

Now let us examine the properties of this polytope Q.

2.2.3.1 ML certificate

Firstly, it can be shown that all integer vertices of Q are valid codewords:

$$Q \bigcap \{0,1\}^n = \mathcal{C}. \tag{2.78}$$

So if we obtain an integer solution from an LP decoder based on Q, this solution must be a valid codeword from $\mathcal{C}$. Given this is true, the ML decoder will output the same codeword. This is called the *ML certificate*. This property is one of the touted advantages of LP decoding [25].

2.2.3.2 Fractional Distance

In linear codes, the minimum Hamming distance among all codewords d is used to characterize the performance of the code. In parallel, there is a similar concept in LP decoding.

Definition 2.2. ([25]) Define the fractional distance of Q as the minimum (pseudocodewords) distance between any integral vertex, i.e., codeword, and any other vertex:

$$d_{\text{frac}} = \min_{\substack{\mathbf{y} \in (V(Q) \bigcap \{0,1\}^n) \\ \mathbf{f} \in V(Q) \\ \mathbf{f} \neq \mathbf{y}}} \sum_{i=1}^{n} |y_i - f_i|. \tag{2.79}$$

Obviously, $\mathbf{f}$ can be a codeword. Then the absolute values in (2.79) will be equivalent to the Hamming distance which is used in calculating the minimum code distance d. Therefore d_{frac} is upper bounded by d.

Recall that an $[n, k, d]$ linear code can correct up to $\lceil \frac{d}{2} \rceil - 1$ errors [43]. Regarding LP decoding we have

Theorem 2.3. *([25]) The LP decoder using Q will be successful if at most*

$$\left\lceil \frac{d_{\text{frac}}}{2} \right\rceil - 1 \tag{2.80}$$

bits are flipped by a binary symmetric channel.

2.2.3.3 $\mathcal{C}$-Symmetry

We say that the LP decoder succeeds if the transmitted codeword is the *unique* optimal solution to the LP. If there are more than one LP optima, the LP decoder fails.

$$\Pr[\text{error}|\mathbf{y} \text{ transmitted}] = \Pr\left[\exists \mathbf{f} \in Q, \mathbf{f} \neq \mathbf{y} \left| \sum_{i=1}^{n} \gamma_i f_i \leq \sum_{i=1}^{n} \gamma_i y_i \right.\right]. \tag{2.81}$$

In coding theory, the all-zeros codeword $\mathbf{0}$ is widely applied as reference to evaluate decoding failure probability, which is based on the fact that the error probability is independent of which codeword is transmitted and $\mathbf{0}$ is always a codeword. In addition, d is equal to the minimum Hamming weight of non-zero codewords. Both are due to code linearity. As a matter of fact, in LP decoding, the (absolute) difference of an integral vertex and any other vertex is still a vertex. This is interesting in that, firstly, (2.79) can be computed as the minimum weight of non-zero vertices:

$$d_{\text{frac}} = \min_{\mathbf{f} \in (V(Q) \setminus \{\mathbf{0}\})} \sum_{i=1}^{n} f_i. \tag{2.82}$$

Note that (2.82) is an LP problem, so it can be solved by an LP solver, say based on *Simplex*. An improved procedure computing d_{frac} was provided by Feldman in [25]. It says that one should first pick up a non-zero vertex of Q. There will be some constraints defining Q that are not met with equality by this vertex. Then for each of those constraints do the following. Derive a new polytope Q' from Q by making the constraint into an equality constraint. Then minimize the summation of (2.82) over Q' by an LP solver. The minimum value obtained over all Q''s is d_{frac}. It works due to the fact that a vertex of a D-dimension polytope is uniquely determined by D linearly independent constraints, which are met with equality by this vertex. The initial vertex must meet D linearly independent constraints. By forcing one other constraint to be an equality, we will get a $D-1$ dimensional polytope Q' and the vertices of Q' will be checked to see if the optimal value is achieved. After all Q' are checked, it is guaranteed that all vertices of the original Q have been examined. Thus we obtain the solution to (2.82).

Second, as Q "looks" exactly the same from any other codeword in $\mathcal{C}$,

Theorem 2.4. *([25]) For any LP decoder using Q under a binary-input memoryless symmetric channel, the probability that the LP decoder fails is independent of the transmitted codeword.*

So we may assume that $\mathbf{0}$ is transmitted under LP decoding scheme. By Theorem 2.4 and (2.81), we have

Corollary 2.5. *([25]) The LP decoder based on the polytope Q will fail iff $\exists \mathbf{f} \in Q$ with a value of its objective function less than or equal to zero, where $\mathbf{f} \neq \mathbf{0}$.*

$$\begin{aligned} \Pr[\textit{error}|\mathbf{y} \textit{ transmitted}] &= \Pr[\textit{error}|\mathbf{0} \textit{ transmitted}] & (2.83) \\ &= \Pr\left[\exists \mathbf{f} \in Q, \mathbf{f} \neq \mathbf{0} \left| \sum_{i=1}^{n} \gamma_i f_i \leq 0 \right.\right]. & (2.84) \end{aligned}$$

As remarked earlier, the LP polytope is enlarged from the ML polytope through relaxation. The greater the relaxation, the poorer the decoding results. So we want the polytope to be as small as possible, i.e., to make $V(Q)$ as small as possible, to reduce the number of fractional vertices which can cause the decoding failure. However, on the other hand, to shrink the polytope by adding more linear constraints, the complexity will grow. We must balance the trade-offs between the computation complexity and the decoding performance.

2.2.4 Comparison with Message-Passing Decoding

The sum-product algorithm performs very well in practice [17]. However, it does not always converge. In addition, it does not have the ML certificate property. But it should be noted that in practice, for large block length LDPC codes, when the sum-product decoder outputs a codeword, it is extremely rare for it not to be the ML codeword [25]. Also, we can make use of *LP duality* to give message-passing algorithms the ML certificate [25].

Comparing the computational complexity of LP decoding and message-passing decoding is still very much an open issue, both in theory and practice. Theoretically, the LP decoder is far less efficient than the message-passing decoders, most of which run in linear time (for a fixed number of iterations). Intuitively, this is because that LP decoding has to "pay" computationally for the ML certificate property.

In a numerical experiment run in [25], it was noted that the performance of the LP decoder for LDPC codes lies in between the performance of min-sum and sum-product. It is interesting that when the BSC crossover probability becomes very small, LP even outperforms sum-product. But still, there is a big gap between them and optimal ML decoding.

Chapter 3

Error Floor Estimation

The error floor in modern graph-based error control codes such as low-density parity-check codes is caused by inherent structural weaknesses in the code's interconnect network. The iterative message-passing algorithm cannot overcome these weaknesses and gets trapped in error patterns which are easily identifiable as erroneous (in LDPC codes), and are thus not valid codewords, but difficult to overcome or correct. These weaknesses were termed *trapping sets* by Richardson in [58], a summary definition for the patterns on which the message-passing algorithm fails for Gaussian channels. These trapping sets are dependent on the code, the channel used, and to a lesser degree also on the details of the decoding algorithm.

The message-passing decoding is based on the assumption that the Tanner graph is a *tree*, i.e., there is a unique path connecting any two distinct variable nodes. Then the messages passing along the edges are independent of each other. However, the actual code graph can never be a tree. The highly structured parity-check matrix will introduce all kinds of *cycles* to the graph, which compromise the independence of the passed messages. After a few iterations, whose number depends on the girth of the Tanner graph and the number of cycles, the probabilities will become dependent on each other. And that causes the decoding failure, eventually. This also explains that why trapping sets are cycles or unions of cycles.

Therefore, making the girth as large as possible is one of the primary objectives when constructing LDPC codes. The value of girth can be as small as 4. Four 1's at the four corners of any sub-matrix in the parity-check matrix corresponds to one 4-cycle. This can be avoided by carefully constructing the $\mathbf{H}$ matrix. To avoid 6-cycles seems to be much more complicated. However, some 4-cycle free LDPC codes provide pretty good performance already [21].

Prior work on identifying the weaknesses of LDPC codes on erasure channels led to the definition of *stopping sets* in [16]. Stopping sets, being the weaknesses of LDPC codes on erasure channels, also play a role on Gaussian channels, but are not typically the dominant error mechanism. In [88] the authors define *absorption sets*, which are the subgraphs of the code graph on which the Gallager bit-flipping decoding algorithms fail for binary symmetric channels. The authors observed that these absorption sets also show up as the dominant trapping sets in certain structured LDPC codes. In [89] they devise *post-processing methods* to reduce the effects of these absorption sets and lower the error floor of the codes in question.

In this chapter we present a linear algebraic approach to the dynamic behavior of absorption sets. We show that these sets follow a geometric growth phase during early iter-

ations where messages inside the absorption set grow towards a largest eigenvector which characterizes the absorption set. The seemingly erratic behavior of the messages at early iterations is due to the decreasing influence of lesser eigenvectors. We define the *gain* of an absorption set and show how it affects the influence of the extrinsic messages that flow into the absorption set at each iteration from the remainder of the code network. The importance of set extrinsic information was already informally observed in [83], who reported a lowering of the error floor with increased extrinsic connectivity. We use our analysis to produce accurate error formulas for the error floor BER/FER and support these results with importance sampling simulations targeting the absorption sets.

As illustrations we carefully identify and classify absorption sets of the regular $[2048, 1723]$ LDPC code recently designed in [21], which is used in the IEEE 802.3an standard, and a length-155 Tanner code. Topological features of dominant absorption sets are identified and a search algorithm is presented which finds the leading dominant sets.

3.1 Error Patterns of LDPC Codes on BEC

Stopping sets completely determine the performance of graph-based decoding of LDPC codes on erasure channels, i.e., on channels where the transmitted binary symbols are either received correctly, or are erased, as shown in Figure 2.1. A complete statistical treatment of stopping sets was given in [16]. Aptly named, a stopping set is a subset of uncorrected variable nodes where the decoder stops, i.e., makes no further correction progress. It is simply defined as:

Definition 3.1. A stopping set $\mathcal{S}$ is a set of variable nodes, all of whose neighboring check nodes are connected to the set $\mathcal{S}$ at least twice.

Figure 3.1 shows the Tanner graph of a length-20 regular $(3, 4)$ LDPC code. A size-6 stopping set is plotted as black nodes. The check nodes that are incident to the set nodes are colored blue. It is quite straightforward to see that if erasure decoding is performed following Gallager's decoding algorithm [61] the variable values in the stopping set cannot be reconstructed. The variable nodes will keep receiving all 0's from their neighbor check nodes and stay "unknown", as from Step 3 in Section 2.1.1. Valid codewords are trivially stopping sets, but the set of stopping sets is larger than the set of valid codewords.

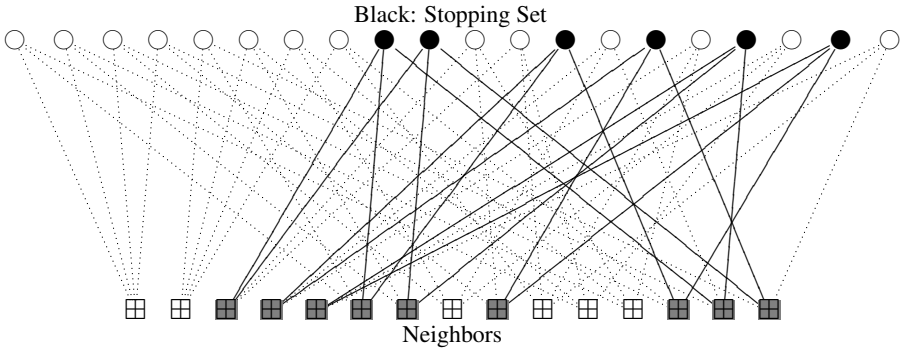


Figure 3.1: Example of a stopping set.

3.2 Bit-Flipping on BSC

An absorption set is an extension of the notion of a stopping set to the binary symmetric channels [88] [89], and is defined as:

Definition 3.2. An absorption set $\mathcal{A}$ is a set of variable nodes, such that the majority of each variable node's neighbors are connected to the set $\mathcal{A}$ an even number of times.

Figure 3.2 shows the same LDPC code as shown in Figure 3.1. However, one of its length-4 absorption sets is highlighted. The set's neighboring check nodes are in blue and red, representing satisfied and unsatisfied checks, respectively, as used in Figures 1.4 and 1.6. It can be verified that Gallager-type bit-flipping decoding will not be able to correct an absorption set, since a majority of messages impinging on each variable node will retain the erroneous sign for each iteration, and the messages carrying correct information will lose the majority vote at Step 4 of Section 2.1.2. Consequently, the algorithm locks up. It is believed that the smaller the weight of the absorption sets, the more severe impact on the error curve. Thus by knowing the structure and the multiplicity of the first less weight absorption sets, the error floors of LDPC codes over BSC can be well predicted [12].

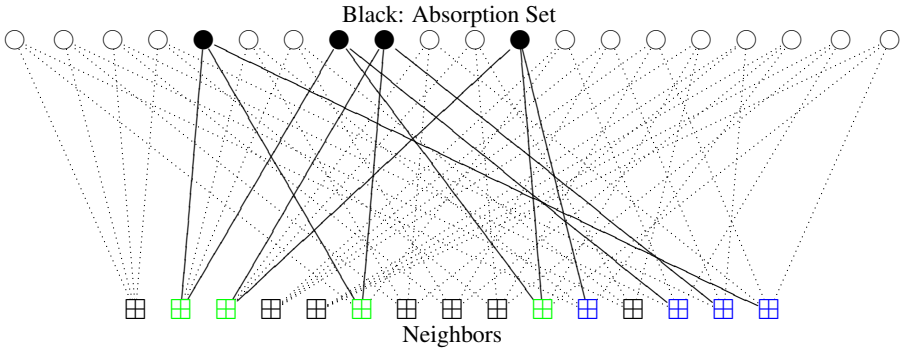


Figure 3.2: Example of an absorption set.

3.3 Message-Passing on AWGN

The Gaussian channel is different from the binary symmetric and binary erasure channels and causes a more complicated error behavior on LDPC codes. Richardson [58] first seriously explored the error floor of LDPC codes on Gaussian channels and defined trapping sets as the failure mechanism. Noting that typically very few trapping sets dominate the error floor region, he proposed a semi-analytical method which amounts to a variant of importance sampling to numerically predict the error floor from the knowledge of a code's trapping sets.

While finding trapping sets remained a largely open problem, [88] observed that in certain structured LDPC codes the dominant trapping sets are absorption sets, i.e., the failure mechanism of the code on binary symmetric channels. In [89], algorithmic modifications were proposed to "eliminate" the error floor caused by these absorption sets.

Table 3.1 lists the specific terms of trapping sets used according to each channel model.

Due to its popularity and extensive exposure, we will start with the $[2048, 1723]$ regular LDPC code [21] used in the IEEE 802.3an standard. This code has been extensively ana-

Table 3.1: The dominant failure patterns of iterative message-passing decoding.

Channel	Decoding Algorithm	Failure Pattern	
BEC	Hard Decision	Stopping Set $\mathcal{S}$ (Definition 3.1)	Trapping Set
BSC	Hard Decision	Absorption Set $\mathcal{A}$ (Definition 3.2)	
AWGN	Soft Decision		

lyzed. It has a low error floor that appears at $E_b/N_0 \approx 5$ dB at a BER of 10^{-12}, that is too low to be efficiently explored using conventional simulations[1].

Despite the fact that this code has a visually identifiable error floor in the high SNR region, it turns out that its code length in the $2k$ range does not simplify analysis of the code or the absorption sets behavior or even help prove the accurateness of our estimation formula in the error floor region.

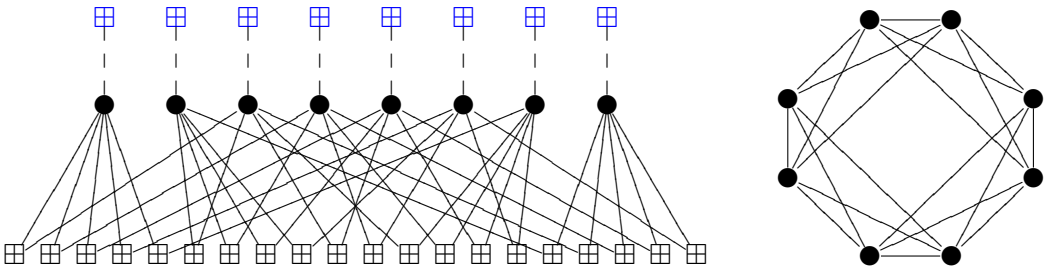
Therefore, another regular LDPC code with length 155 introduced by Tanner caught our attention during the course of our search for a smaller code to facilitate the error floor analysis. This code possesses a similar symmetry in its parity-check matrix, and hence cause the symmetric-looking Tanner graphs of the absorption sets, for which we already have a valid enumeration technique. The compromise is that there is no readily-seen error floor in its error rate curve. We had to catch all error events at each given SNR value during the simulation process, and then sort them, and finally calculate the contribution of absorption sets to the error rate, in order to determine where the error floor starts. One benefit of this approach is that it is confirmed once again that there exist absorption sets that dominate the decoding failures in the error floor region. Another surprise from this *tedious* error event collecting and sorting approach is that it provides us a means to evaluate our efforts to lower or even eliminate the error floors. We have run several experiments to counter-act the absorbing phenomenon, based on the insights from our formula. One of them gives us the best result in terms of error rate.

In the rest of this chapter, we will analyze the dynamics of both the popular IEEE 802.3an LDPC code and the more simulation-friendly length-155 Tanner code. The analysis will include identifying absorption sets and applying our error floor estimation to both codes. Then a solution to eliminate the appearance of the absorption sets as decoding failures will be proposed in the following chapter.

3.4 Decoding Failures of the IEEE 802.3an LDPC Code in its Error Floor Region

Figure 3.3 shows the structure of the *dominant* absorption set of this code (see also [89, Figure 2]). There are $14,272$ such sets in this $[2048, 1723]$ code. They dominate the error floor since they are the *minimal* absorption sets of this code (for definitions of minimal and dominant, see Definition 3.4).

[1]Even an FPGA-based simulation running at 100Mb/s requires about a week for a single data point.



(a) Check nodes shown. (b) Check nodes hidden.

Figure 3.3: A dominant absorption set of the regular $(6, 32)$ IEEE 802.3an code (not all check node connections are shown).

3.4.1 Finding Dominant Absorption Sets

The $[2048, 1723]$ rate 0.8413 regular LDPC code [21] considered here has a highly structured parity-check matrix:

$$\mathbf{H}_{m\times n} = \begin{bmatrix} \boldsymbol{\sigma}_{1,1} & \boldsymbol{\sigma}_{1,2} & \boldsymbol{\sigma}_{1,3} & \cdots & \boldsymbol{\sigma}_{1,32} \\ \boldsymbol{\sigma}_{2,1} & \boldsymbol{\sigma}_{2,2} & \boldsymbol{\sigma}_{2,3} & \cdots & \boldsymbol{\sigma}_{2,32} \\ \boldsymbol{\sigma}_{3,1} & \boldsymbol{\sigma}_{3,2} & \boldsymbol{\sigma}_{3,3} & \cdots & \boldsymbol{\sigma}_{3,32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\sigma}_{6,1} & \boldsymbol{\sigma}_{6,2} & \boldsymbol{\sigma}_{6,3} & \cdots & \boldsymbol{\sigma}_{6,32} \end{bmatrix}_{384\times 2048}, \tag{3.1}$$

where each $\boldsymbol{\sigma}_{i,j}$ is a carefully selected 64×64 permutation matrix. The detailed construction steps can be found in Appendix A.1.

We point out that the appearance of this matrix is different in the IEEE 802.3 standard and in [21]. However the two matrices are equivalent as one is simply the permuted version of the other. Therefore the performance is identical, however the topologies of absorption sets might be different. Figure 3.4 shows the non-zero elements of the matrix as solid circles, whereas the zeros are left blank. We found that the matrix that came directly from [21] possesses more symmetric properties, which is reflected in the absorption sets enumeration, for instance, the statistics shown in Figure 3.7 that will be presented later.

Definition 3.3. Let a two-tuple (a, b) denote an absorption set, where a is the size of the set (number of variable nodes) and b is the extrinsic message degree (EMD), i.e., the cardinality of the set of the neighboring check nodes that are connected to the set an odd number of times (the unsatisfied check nodes).

Example 3.1. For example, Figures 3.2 and 3.3 show a $(4, 4)$ and an $(8, 8)$ absorption sets, respectively.

Table 3.2 shows the first few absorption sets of the code [21]. Note that the ratio b/a is the average EMD for an (a, b) absorption set. The smaller this ratio, the smaller the portion of "correct" extrinsic messages that "help" recover the transmitted bits. This motivates the following definitions.

Definition 3.4. (i) An (a, b) absorption set is called minimal if no (a', b') absorption set exists with $a' < a$ and $b'/a' \leq b/a$, i.e., fewer variable nodes and smaller average EMD.

Table 3.2: The first few absorption sets of the IEEE 802.3an LDPC code.

a	b	**Existence**	**Multiplicity**	**Gain** ($\mu_{\max}$)
< 5		No		
5	10	No		
6	6	No		
	8			
	10			
	12			
7	0	No		
	2			
	4			
	6			
	8			
	10			
	12	Yes	$65,472$	3.29
	14		$14,720$	3
8	0	No		
	2			
	4			
	6			
	8	Yes	$14,272$	4
	10	No		
	12	Yes	$44,416$	3.5
	14		$88,896$	3.25
	16		$661,824$	3
9	0	No		
	2			
	4			
	6			
	8			
	10			
	12	Yes	?	3.67
	14			3.44
	16			3.22
	18			3
10	0	No		
	2			
	4			
	6			
	8	?		
	10	Yes	> 192	4
	12		?	3.8
	14			3.6
	16			3.4
	18			3.2
	20			3

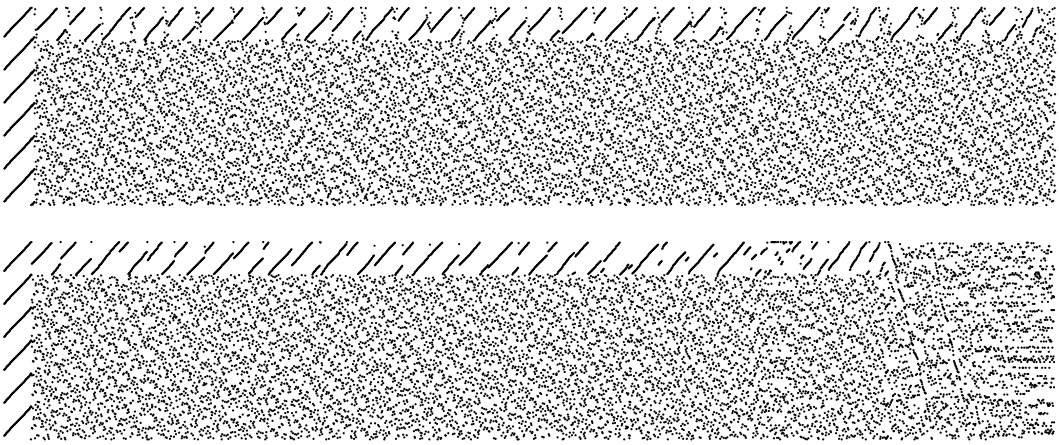


Figure 3.4: Square pattern plots of the binary parity check matrix: Top: output of [21]; Bottom: used in the IEEE 802.3an standard.

(ii) A minimal (a, b) absorption set is called dominant if no (a, b') absorption set exists with $b' < b$, i.e., smaller EMD.

The smaller the absorption set, the more severe the effect on the error floor. Thus our target is to find the dominant absorption sets in terms of a, b and b/a. Since the variable node degree $d_v = 6$, $a \geq 5$ by the definition of absorption sets and using the fact that the code is 4-cycle free. In addition, $b \in [0, 2a]$ and must be even. The coefficient $5 - b/a$ is the *gain* of the absorption set, which determines how fast the extrinsic information enters the set — see later. Also note that an $(a, 0)$ absorption set corresponds to a weight-a codeword in $\mathcal{C}$. Thus the existence of $(a, 0)$ absorption sets implies the existence of weight-a codewords, and vice versa.

Let us introduce additional notation needed to prove the existence of absorption sets.

Definition 3.5. (i) For any variable node v in an absorption set, let $\mathrm{Deg}(v)$ denote the number of neighboring check nodes of v that are connected to the set an even number of times. $\mathrm{Deg}(v)$ is the degree of vertex v in the topology graph with check nodes hidden. (ii) Let an unordered integer array $[\mathrm{Deg}(v_i) : i = 1, 2, \ldots, a]$ denote a class of (a, b) absorption sets, where $\mathrm{Deg}(v_i) \in \left\{ \left\lfloor \frac{d_v}{2} \right\rfloor + 1, \left\lfloor \frac{d_v}{2} \right\rfloor + 2, \ldots, d_v \right\}$ and $\sum_{i=1}^{a} \mathrm{Deg}(v_i) = ad_v - b$.

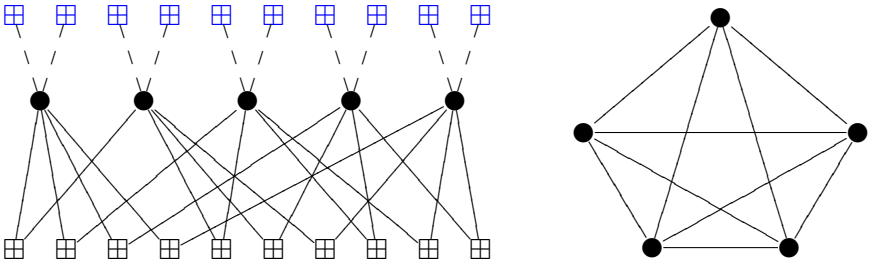
Example 3.2. Obviously, all five nodes of an $(5, 10)$ absorption set have $\mathrm{Deg}(v) = 4$ as can be seen in Figure 3.5. Thus, there exists only one class of the $(5, 10)$ absorption sets that can be denoted as $[4, 4, 4, 4, 4]$, which satisfies $\sum_{i=1}^{a} \mathrm{Deg}(v_i) = 6a - b = 20$.

Note that there are a total of $\sum_{i=1}^{a} \mathrm{Deg}(v_i) = 6a - b$ edges that need to be paired up in one graph. Therefore the sum has to be even. It follows that b has to be an even number, as seen in Table 3.2.

Theorem 3.1. *(i) There are no size-*5 *absorption sets. (ii) There are no size-*6 *absorption sets. (iii) There are no* $(7, b)$ *absorption sets with* $b < 12$*.* $(7, 12)$ *and* $(7, 14)$ *absorption*

sets do exist. (iv) For $b < 8$, there exist no $(8, b)$ absorption sets.[2] *For $b = 8$, there exists no $(8, 8)$ absorption set that contains a degree-6 variable node. (v) The number of $(8, 8)$ absorption sets is $14,272$ and they all have the topology of Figure 3.6(e).*

Proof. (i) Clearly b can only equal 10 when $a = 5$ and there is only one possible connecting topology shown in Figure 3.5. The parity-check matrix $\mathbf{H}$ of [21] is searched by Algorithm 1, observing the constraints imposed by this topology, combined with some of the properties listed in [23, 24, 22] for array-based LDPC codes.



(a) Check nodes shown. (b) Check nodes hidden.

Figure 3.5: The only possible topology of $(5, 10)$ absorption sets.

(ii–iv) When the sets are growing bigger, it is getting more difficult to find absorption sets, even by making use of Algorithm 1-type methods, due to their extremely low appearance and the combinatorial complexity. Hence we take advantage of Definition 3.5 to classify the absorption sets first. For each pair (a, b), there may be several classes of absorption sets, and each class may exhibit several topologies. What we are trying to do is to reduce one unknown absorption set to a smaller absorption set whose non-existence is known by eliminating nodes from the original set. We can then argue that there is only a limited number of topologies that have to be searched algorithmically. See Appendix A.2.3 or [63] [64] for details.

(v) By (iv), the only possible class of $(8, 8)$ absorption set would have connectivity $[5, 5, 5, 5, 5, 5, 5, 5]$. It can be shown that graphically, there exist five possible topologies for this connectivity as shown in Figure 3.6 [63] [64]. By searching all those topologies against the $\mathbf{H}$ matrix, the proof is completed. □

It is interesting to note that because of the block structure of the $\mathbf{H}$ matrix, certain groups of variable nodes do share the same multiplicity, as listed in Table 3.3, although the average multiplicity of each variable node appeared in such sets is $14272 \times 8 \div 2048 = 55.75$, as shown in Figure 3.7(a), as well.

3.4.2 Less Dominant Absorption Sets

There definitely exist larger and less dominant absorption sets. Examples are listed in [64]. With the size of the absorption sets growing, it is getting extremely difficult to enumerate

[2] As a corollary, since there is no $(8, 0)$ absorption set, the minimum distance bound of this LDPC code [21] is strengthened to $d_{\min} \geq 10$. Therefore, there are no $(9, 0)$ absorption sets since a $(9, 0)$ absorption set is a length-9 codeword.

Input: Parity-check matrix $\mathbf{H}_{384\times 2048}$ and Figure 3.5.
Output: Absorption sets.
foreach *variable node* $v \in \{0, 1, 2, \ldots, 2047\}$ **do**
    Pick 4 out of 6 neighboring check nodes of v denoted c_0, c_1, c_2 and c_3, respectively;
    foreach *one out of* 31 *neighboring variable nodes other than* v *of* c_0*, denoted as* v_1 **do**
        foreach *one out of* 31 *neighboring variable nodes other than* v *of* c_1*, denoted as* v_2 **do**
            if v_2 *and* v_1 *are not connected* **then**
                re-pick v_2;
            else
                foreach *one out of* 31 *neighboring variable nodes other than* v *of* c_2*, denoted as* v_3 **do**
                    if v_3 *and* v_1 *or* v_3 *and* v_2 *are not connected* **then**
                        re-pick v_3;
                    else
                        foreach *one out of* 31 *neighboring variable nodes other than* v *of* c_3*, denoted as* v_4 **do**
                            if v_4 *and* v_1 *or* v_4 *and* v_2 *or* v_4 *and* v_3 *are not connected* **then**
                                re-pick v_4;
                            else
                                follow Definition 3.3 to determine if this candidate set is an absorption set;

Algorithm 1: Algorithm for finding $(5, 10)$ absorption sets.

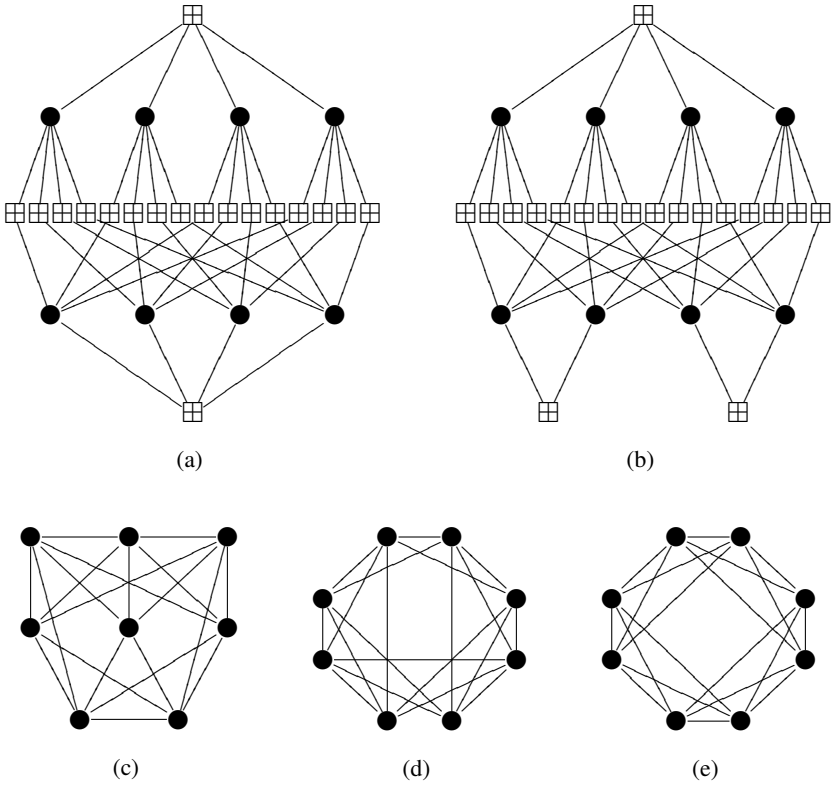
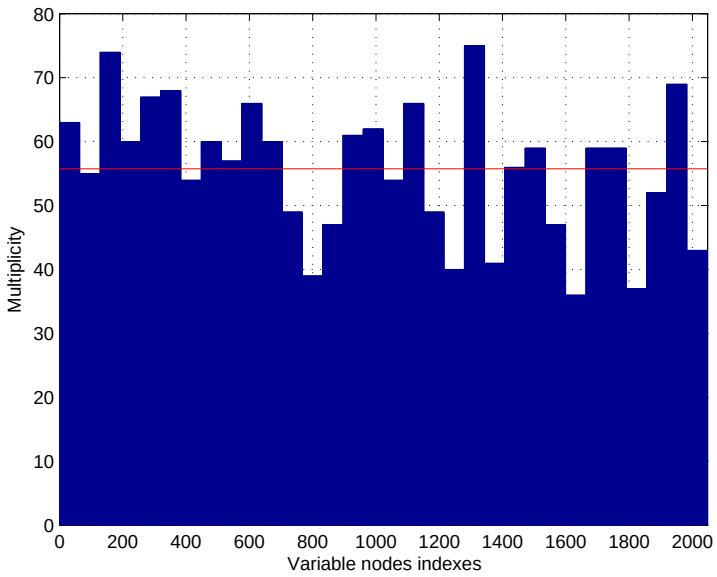


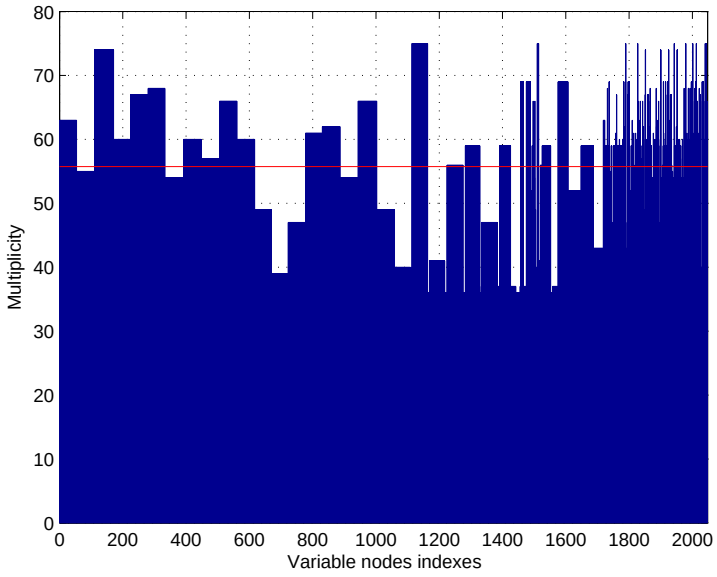
Figure 3.6: Possible topologies of $(8, 8)$ absorption sets.

Table 3.3: The multiplicity of each variable node in $(8, 8)$ absorption sets.

Group	**Variable Nodes**	**Multiplicity**	**Group**	**Variable Nodes**	**Multiplicity**
1	0—63	63	17	1024—1087	54
2	64—127	55	18	1088—1151	66
3	128—191	74	19	1152—1215	49
4	192—255	60	20	1216—1279	40
5	256—319	67	21	1280—1343	75
6	320—383	68	22	1344—1407	41
7	384—447	54	23	1408—1471	56
8	448—511	60	24	1472—1535	59
9	512—575	57	25	1536—1599	47
10	576—639	66	26	1600—1663	36
11	640—703	60	27	1664—1727	59
12	704—767	49	28	1728—1791	59
13	768—831	39	29	1792—1855	37
14	832—895	47	30	1856—1919	52
15	896—959	61	31	1920—1983	69
16	960—1023	62	32	1984—2047	43



(a) Permuted $\mathbf{H}$.



(b) Original $\mathbf{H}$.

Figure 3.7: Histograms of the multiplicity of 2,048 variable nodes in $(8, 8)$ absorption sets, where the horizontal thin red line marks the average multiplicity 55.75.

them all. However, compared to the smaller sets, their contribution to the error curve can be ignored.

The enumerating process of absorption sets gets exhaustive and lengthy with increasing set size, due to the exponentially increasing combinations of the nodes and connectivities. Hence the detailed and systematic absorption sets search is moved to Appendix A.2.

3.5 Error Floor Estimation of the IEEE 802.3an LDPC Code

After we confirmed that the absorption sets are responsible for the error floor, we studied their dynamic in the decoding process and used this knowledge to build a formula to approximate the error rate caused by absorption sets. We will keep using the IEEE 802.3an LDPC Code in our illustration, followed by a numerical analysis.

3.5.1 Dynamic Analysis of Absorption Sets

We now present a linearized analysis to gain insight into the behavior of absorption sets starting with the leading $(8, 8)$ absorption set, shown in Figure 3.3. First we note that the variable nodes perform simple addition, as shown by (2.13). Furthermore, the check nodes basically choose the minimum of the incoming signals. If we make the reasonable assumption that the absorption set converges more slowly than the remaining nodes in the code, and due to the fact that each (satisfied) check node is connected exactly to two absorption set variables, the minimum absolute-value signal into the participating check nodes will come from one of the absorption set variables. If this is true, the check nodes simply *exchange* the signals on the connections to the absorption set variable nodes. We will refine this approximation below.

Additionally, each absorption set variable node is singly connected to a "lone" floating extrinsic parity check node, all of whose other connections go to other, set-external variable nodes. The messages through these eight extrinsic check nodes are the extrinsic messages into the absorption set, and are of crucial importance since the error messages introduced by the set nodes will be circling around within the local network of absorption sets. Algorithmically, they play exactly the same role as the intrinsic channel values which are fed into the variable nodes by virtue of the summation function executed at the variable nodes.

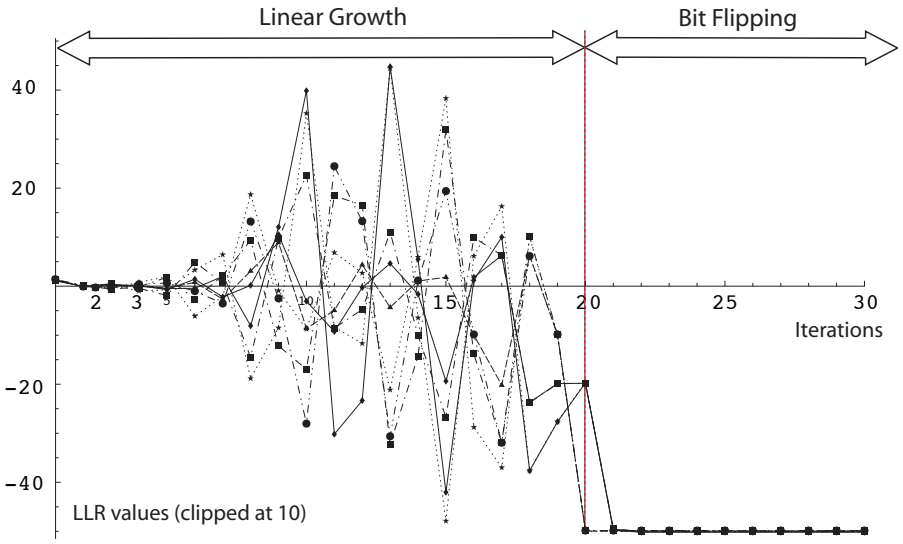
Figure 3.8 shows two examples of the dynamic behavior of the absorption set variables close to its decision threshold boundary. The seemingly erratic behavior resolves after a number of iterations when all variables follow highly correlated trajectories. This observation is the basis for the following analysis.

Denote the outgoing solid edge values from the variable nodes by x_i, i.e., $x_1, x_2, \ldots, x_5$ leave variable node $v = 0$, $x_6, x_7, \ldots, x_{10}$ variable node $v = 1$, etc. Collect the x_i in the length-40 column vector $\mathbf{x}$, which is the vector of outgoing variable edge values in the absorption set. Likewise, and analogously, let $\mathbf{y}$ be the incoming edge values to the variable nodes, such that y_j corresponds to the reverse-direction message. Now, at iteration $i = 0$

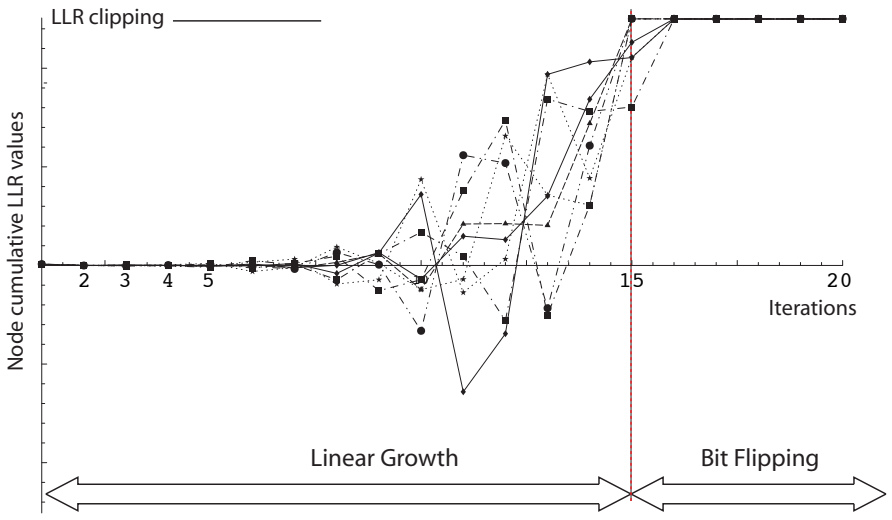
$$\mathbf{x}_0 = \boldsymbol{\lambda}, \tag{3.2}$$

where the initial input is the vector of channel intrinsics

$$\boldsymbol{\lambda} = \Big[\underbrace{\lambda_1, \ldots, \lambda_1}_{\text{eight entries}}, \underbrace{\lambda_2, \ldots, \lambda_2}_{\text{eight entries}}, \ldots\ldots, \underbrace{\lambda_8, \ldots, \lambda_8}_{\text{eight entries}}\Big]^{\mathrm{T}} \tag{3.3}$$



(a) This frame results in an $(8, 8)$ absorption set error event.



(b) This frame is corrected.

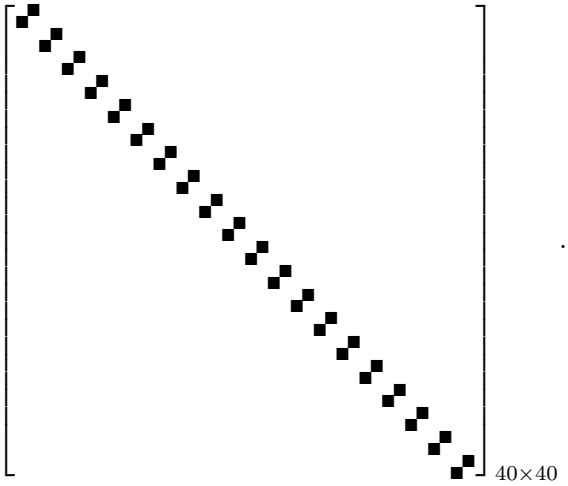
Figure 3.8: The accumulated LLRs at an $(8, 8)$ absorption set nodes of the IEEE 802.3an LDPC code verses iterations, assuming that the all-zero codeword is transmitted.

duplicated onto the outgoing messages. It undergoes the following operation at the check node:

$$\mathbf{y}_0 = \mathbf{C}\mathbf{x}_0 = \mathbf{C}\boldsymbol{\lambda}, \tag{3.4}$$

where $\mathbf{C}$ is a permutation matrix that exchanges the absorption set signals as discussed above.

$$\mathbf{C} = \tag{3.5}$$

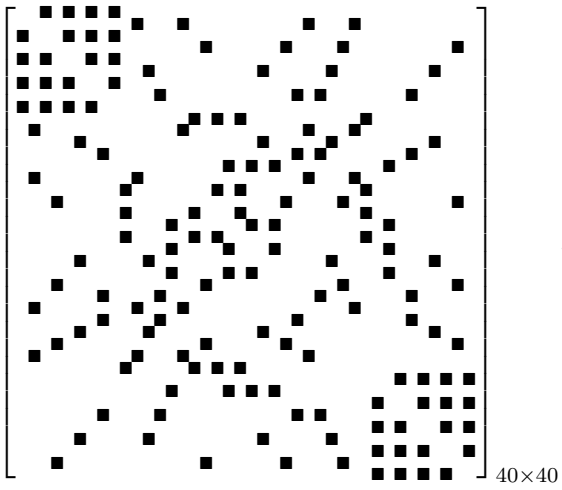


At iteration $i = 1$ we obtain

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{V}\mathbf{y}_0 + \boldsymbol{\lambda} + \boldsymbol{\lambda}_1^{(\mathrm{ex})} && (3.6)\\ &= \mathbf{V}\mathbf{C}\boldsymbol{\lambda} + \boldsymbol{\lambda} + \boldsymbol{\lambda}_1^{(\mathrm{ex})}, && (3.7) \end{aligned}$$

where $\mathbf{V}$ is the variable node function matrix, i.e., each output is the sum of the other four inputs from the check nodes plus the intrinsic input.

$$\mathbf{V} = \tag{3.8}$$



The extrinsic inputs from the remainder of the code graph are contained in $\boldsymbol{\lambda}_1^{(\mathrm{ex})}$. Following the linear model, at iteration $i = j$ extrinsic signals are injected into the absorption set as

$$\boldsymbol{\lambda}_j^{(\mathrm{ex})} = \Big[\underbrace{\lambda_{j1}^{(\mathrm{ex})}, \ldots, \lambda_{j1}^{(\mathrm{ex})}}_{\text{eight entries}}, \underbrace{\lambda_{j2}^{(\mathrm{ex})}, \ldots, \lambda_{j2}^{(\mathrm{ex})}}_{\text{eight entries}}, \ldots\ldots, \underbrace{\lambda_{j8}^{(\mathrm{ex})}, \ldots, \lambda_{j8}^{(\mathrm{ex})}}_{\text{eight entries}}\Big]^{\mathrm{T}} \tag{3.9}$$

via the extrinsic check nodes. By induction on i:

$$\begin{aligned}
\mathbf{x}_2 &= \mathbf{V}\mathbf{y}_1 + \boldsymbol{\lambda} + \boldsymbol{\lambda}_2^{(\mathrm{ex})} && (3.10)\\
&= \mathbf{V}\mathbf{C}\mathbf{x}_1 + \boldsymbol{\lambda} + \boldsymbol{\lambda}_2^{(\mathrm{ex})} && (3.11)\\
&= (\mathbf{V}\mathbf{C})^2\boldsymbol{\lambda} + \mathbf{V}\mathbf{C}\boldsymbol{\lambda} + \boldsymbol{\lambda} + \mathbf{V}\mathbf{C}\boldsymbol{\lambda}_1^{(\mathrm{ex})} + \boldsymbol{\lambda}_2^{(\mathrm{ex})} && (3.12)\\
&= \sum_{i=0}^{2}(\mathbf{V}\mathbf{C})^i\boldsymbol{\lambda} + \sum_{i=1}^{2}(\mathbf{V}\mathbf{C})^{2-i}\boldsymbol{\lambda}_i^{(\mathrm{ex})}, && (3.13)\\
\mathbf{x}_3 &= \mathbf{V}\mathbf{y}_2 + \boldsymbol{\lambda} + \boldsymbol{\lambda}_3^{(\mathrm{ex})} && (3.14)\\
&= \mathbf{V}\mathbf{C}\mathbf{x}_2 + \boldsymbol{\lambda} + \boldsymbol{\lambda}_3^{(\mathrm{ex})} && (3.15)\\
&= \sum_{i=0}^{2}(\mathbf{V}\mathbf{C})^{i+1}\boldsymbol{\lambda} + \sum_{i=1}^{2}(\mathbf{V}\mathbf{C})^{3-i}\boldsymbol{\lambda}_i^{(\mathrm{ex})} + \boldsymbol{\lambda} + \boldsymbol{\lambda}_3^{(\mathrm{ex})} && (3.16)\\
&= \sum_{i=-1}^{2}(\mathbf{V}\mathbf{C})^{i+1}\boldsymbol{\lambda} + \sum_{i=1}^{3}(\mathbf{V}\mathbf{C})^{3-i}\boldsymbol{\lambda}_i^{(\mathrm{ex})} && (3.17)\\
&= \sum_{i=0}^{3}(\mathbf{V}\mathbf{C})^i\boldsymbol{\lambda} + \sum_{i=1}^{3}(\mathbf{V}\mathbf{C})^{3-i}\boldsymbol{\lambda}_i^{(\mathrm{ex})}, && (3.18)
\end{aligned}$$

we obtain at iteration $i = I$

$$\begin{aligned}
\mathbf{x}_I &= \sum_{i=0}^{I}(\mathbf{V}\mathbf{C})^i\boldsymbol{\lambda} + \sum_{i=1}^{I}(\mathbf{V}\mathbf{C})^{I-i}\boldsymbol{\lambda}_i^{(\mathrm{ex})} && (3.19)\\
&= \sum_{i=0}^{I}\left((\mathbf{V}\mathbf{C})^i\boldsymbol{\lambda} + (\mathbf{V}\mathbf{C})^{I-i}\boldsymbol{\lambda}_i^{(\mathrm{ex})}\right), && (3.20)
\end{aligned}$$

where $(\mathbf{V}\mathbf{C})^0 = \mathbf{I}$ and $\boldsymbol{\lambda}_0^{(\mathrm{ex})} = \mathbf{0}$. Applying the spectral theorem we obtain

$$(\mathbf{V}\mathbf{C})^i\boldsymbol{\lambda} \to \mu_{\max}^i\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}, \tag{3.21}$$

where $\mathbf{v}_{\max}$ is the unit-length eigenvector of the maximal eigenvalue $\mu_{\max}$ of the matrix $\mathbf{V}\mathbf{C}$.

The absorption set in question falls in error if

$$\begin{aligned}
\beta &= \mathbf{x}_I^{\mathrm{T}} \cdot \mathbf{1} && (3.22)\\
&\approx \left(\sum_{i=0}^{I}\left(\mu_{\max}^i\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}} + \mu_{\max}^{I-i}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right)\cdot\mathbf{1} && (3.23)\\
&= \mu_{\max}^I\left(\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{I-i}}\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}} + \frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right)\cdot\mathbf{1} && (3.24)\\
&= \mu_{\max}^I\left(\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{I-i}}\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right.\\
&\quad\left.+\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right)\cdot\mathbf{1} && (3.25)
\end{aligned}$$

$$
\begin{aligned}
&= \mu_{\max}^{I}\left(\sum_{k=0}^{I}\left(\frac{1}{\mu_{\max}^{k}}\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right.\\
&\quad\left.+\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_{i}^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right)\cdot\mathbf{1} && (3.26)\\
&= \mu_{\max}^{I}\left(\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right.\\
&\quad\left.+\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_{i}^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right)\cdot\mathbf{1} && (3.27)\\
&= \mu_{\max}^{I}\left(\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\left(\boldsymbol{\lambda}+\boldsymbol{\lambda}_{i}^{(\mathrm{ex})}\right)^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\right)\cdot\mathbf{1} && (3.28)\\
&\leq 0. && (3.29)
\end{aligned}
$$

With the Gaussian assumptions and the fact that β is a linear combination of independent Gaussian variables, the probability of (3.29) happening can be calculated as

$$
\begin{aligned}
P_{\mathrm{AS}} &= \Pr\left(\beta\leq 0\right) && (3.30)\\
&= \frac{1}{2}\left(1+\operatorname{erf}\left(\frac{-m_{\beta}}{\sqrt{2\sigma_{\beta}^{2}}}\right)\right) && (3.31)\\
&= \frac{1}{2}\left(1-\operatorname{erf}\left(\frac{m_{\beta}}{\sqrt{2\sigma_{\beta}^{2}}}\right)\right) && (3.32)\\
&= Q\left(\frac{m_{\beta}}{\sqrt{\sigma_{\beta}^{2}}}\right). && (3.33)
\end{aligned}
$$

To find m_{β} and σ_{β}^{2}, let us drop the factor $\mu_{\max}^{I}$ and rewrite (3.28) into two parts

$$
\begin{aligned}
\beta &= \sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1}+\sum_{i=0}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_{i}^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1} && (3.34)\\
&= \underbrace{\left(\left(\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{i}}}_{\triangleq\beta_{1}}+\underbrace{\sum_{i=1}^{I}\left(\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_{i}^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1}}_{\triangleq\beta_{2}}. && (3.35)
\end{aligned}
$$

So $m_{\beta}=m_{\beta_1}+m_{\beta_2}$ and $\sigma_{\beta}^{2}=\sigma_{\beta_1}^{2}+\sigma_{\beta_2}^{2}$.

Finding m_{β_1} is trivial as

$$
\begin{aligned}
m_{\beta_1} &= \left(m_{\lambda}\mathbf{1}^{\mathrm{T}}\mathbf{v}_{\max}\mathbf{v}_{\max}^{\mathrm{T}}\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{i}} && (3.36)\\
&= m_{\lambda}\left(\sum_{j=1}^{ad_v-b}v_j\right)^{2}\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{i}}. && (3.37)
\end{aligned}
$$

Even though finding m_{β_2}, $\sigma^2_{\beta_1}$ and $\sigma^2_{\beta_2}$ cannot be generalized, the error probability does have a general form:

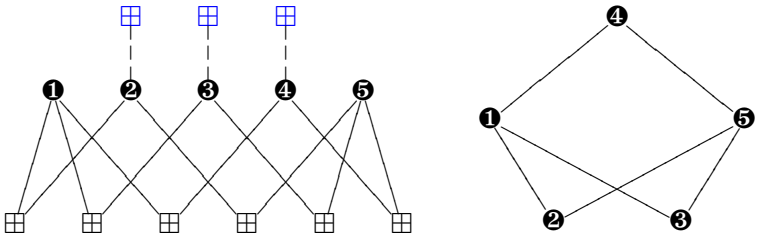
$$P_{\text{AS}} = \Pr(\beta \leq 0) \tag{3.38}$$

$$= Q\left(\frac{Am_\lambda \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} + B\sum_{i=1}^{I} \frac{m_{\lambda^{(\text{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2Cm_\lambda\left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i}\right)^2 + 2D\sum_{i=1}^{I} \frac{m_{\lambda^{(\text{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}\right) \tag{3.39}$$

$$= Q\left(\frac{Am_\lambda \frac{\mu_{\max}^{I+1}-1}{\mu_{\max}^{I+1}-\mu_{\max}} + B\sum_{i=1}^{I} \frac{m_{\lambda^{(\text{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2Cm_\lambda\left(\frac{\mu_{\max}^{I+1}-1}{\mu_{\max}^{I+1}-\mu_{\max}}\right)^2 + 2D\sum_{i=1}^{I} \frac{m_{\lambda^{(\text{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}\right), \tag{3.40}$$

where the coefficients $A = \sum_{j=1}^{ad_v-b} v_j, B, C$ and D characterize the significance of λ and $\lambda^{(\text{ex})}$ to the error floor. Note that, without loss of generality, $\mathbf{v}_{\max}$ is chosen such that $A \geq 0$, so it can be canceled out of the square root in the denominator inside the Q function, which can be seen in the example below.

Example 3.3. Consider the subgraph induced by a $(5, 3)$ absorption set of the $[155, 64, 14]$ regular $(3, 5)$ Tanner code, as shown in Figure 3.9.



(a) Check nodes shown. (b) Check nodes hidden.

Figure 3.9: The only possible topology of a $(5, 3)$ absorption set.

Label the variable nodes as 1 to 5 in Figure 3.9(a), from left to right. And label the solid edges emanating from the variable nodes as $1, 2, \ldots, 12$ from left to right. Then we have the channel intrinsics

$$\boldsymbol{\lambda} = \big[\underbrace{\lambda_1, \lambda_1, \lambda_1}_{\text{node ❶}}, \underbrace{\lambda_2, \lambda_2}_{\text{node ❷}}, \underbrace{\lambda_3, \lambda_3}_{\text{node ❸}}, \underbrace{\lambda_4, \lambda_4}_{\text{node ❹}}, \underbrace{\lambda_5, \lambda_5, \lambda_5}_{\text{node ❺}}\big]^{\text{T}} \tag{3.41}$$

and extrinsics

$$\boldsymbol{\lambda}_i^{(\text{ex})} = \Big[\underbrace{0, 0, 0}_{\text{node ❶}}, \underbrace{\lambda_{i2}^{(\text{ex})}, \lambda_{i2}^{(\text{ex})}}_{\text{node ❷}}, \underbrace{\lambda_{i3}^{(\text{ex})}, \lambda_{i3}^{(\text{ex})}}_{\text{node ❸}}, \underbrace{\lambda_{i4}^{(\text{ex})}, \lambda_{i4}^{(\text{ex})}}_{\text{node ❹}}, \underbrace{0, 0, 0}_{\text{node ❺}}\Big]^{\text{T}}, \tag{3.42}$$

associated with the matrices

$$\mathbf{V} = \begin{bmatrix} 0 & 1 & 1 & & & & & & & & & \\ 1 & 0 & 1 & & & & & & & & & \\ 1 & 1 & 0 & & & & & & & & & \\ & & & 0 & 1 & & & & & & & \\ & & & 1 & 0 & & & & & & & \\ & & & & & 0 & 1 & & & & & \\ & & & & & 1 & 0 & & & & & \\ & & & & & & & 0 & 1 & & & \\ & & & & & & & 1 & 0 & & & \\ & & & & & & & & & 0 & 1 & 1 \\ & & & & & & & & & 1 & 0 & 1 \\ & & & & & & & & & 1 & 1 & 0 \end{bmatrix}_{12\times 12}, \tag{3.43}$$

$$\mathbf{C} = \begin{bmatrix} & & & 1 & & & & & & & & \\ & & & & & 1 & & & & & & \\ & & & & & & & 1 & & & & \\ 1 & & & & & & & & & & & \\ & & & & & & & & & 1 & & \\ & 1 & & & & & & & & & & \\ & & & & & & & & & & 1 & \\ & & 1 & & & & & & & & & \\ & & & & & & & & & & & 1 \\ & & & & 1 & & & & & & & \\ & & & & & & 1 & & & & & \\ & & & & & & & & 1 & & & \end{bmatrix}_{12\times 12}. \tag{3.44}$$

Then we will find $\mu_{\max}$ and the corresponding unit-length eigenvector $\mathbf{v}_{\max}$ of $\mathbf{VC}$. Let $\mathbf{v}_{\max} = [v_1, v_2, \ldots, v_{12}]^{\mathrm{T}}$. Then

$$\beta_1 = \left(\left(\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{i}} \tag{3.45}$$

$$= \left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\left(\mathbf{v}_{\max}^{\mathrm{T}}\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{i}} \tag{3.46}$$

$$= \left(\left(v_1+v_2+v_3\right)\lambda_1 + \left(v_4+v_5\right)\lambda_2 + \left(v_6+v_7\right)\lambda_3 + \left(v_8+v_9\right)\lambda_4 + \left(v_{10}+v_{11}+v_{12}\right)\lambda_5\right)\cdot\left(\sum_{j=1}^{12}v_j\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{i}} \tag{3.47}$$

$$\beta_2 = \sum_{i=1}^{I}\frac{1}{\mu_{\max}^{i}}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\left(\mathbf{v}_{\max}^{\mathrm{T}}\mathbf{1}\right) \tag{3.48}$$

$$= \sum_{i=1}^{I}\frac{1}{\mu_{\max}^{i}}\left(\left(v_4+v_5\right)\lambda_{i2}^{(\mathrm{ex})} + \left(v_6+v_7\right)\lambda_{i3}^{(\mathrm{ex})} + \left(v_8+v_9\right)\lambda_{i4}^{(\mathrm{ex})}\right)\left(\sum_{j=1}^{12}v_j\right). \tag{3.49}$$

Therefore

$$m_{\beta_1} = m_\lambda \left(\sum_{j=1}^{12} v_j \right)^2 \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \tag{3.50}$$

$$\begin{aligned} \sigma_{\beta_1}^2 &= \sigma_\lambda^2 \Big((v_1+v_2+v_3)^2 + (v_4+v_5)^2 + (v_6+v_7)^2 + (v_8+v_9)^2 \\ &\quad + (v_{10}+v_{11}+v_{12})^2 \Big) \cdot \left(\sum_{j=1}^{12} v_j \right)^2 \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 \end{aligned} \tag{3.51}$$

$$\begin{aligned} &= 2m_\lambda \Big((v_1+v_2+v_3)^2 + (v_4+v_5)^2 + (v_6+v_7)^2 + (v_8+v_9)^2 \\ &\quad + (v_{10}+v_{11}+v_{12})^2 \Big) \cdot \left(\sum_{j=1}^{12} v_j \right)^2 \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 \end{aligned} \tag{3.52}$$

$$m_{\beta_2} = \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i} (v_4+v_5+v_6+v_7+v_8+v_9) \left(\sum_{j=1}^{12} v_j \right) \tag{3.53}$$

$$\sigma_{\beta_2}^2 = \sum_{i=1}^{I} \frac{\sigma_{\lambda_i^{(\mathrm{ex})}}^2}{\mu_{\max}^{2i}} \left((v_4+v_5)^2 + (v_6+v_7)^2 + (v_8+v_9)^2 \right) \left(\sum_{j=1}^{12} v_j \right)^2 \tag{3.54}$$

$$= 2 \left((v_4+v_5)^2 + (v_6+v_7)^2 + (v_8+v_9)^2 \right) \left(\sum_{j=1}^{12} v_j \right)^2 \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}. \tag{3.55}$$

Now, let

$$A = \sum_{j=1}^{12} v_j \tag{3.56}$$

$$B = v_4+v_5+v_6+v_7+v_8+v_9 \tag{3.57}$$

$$C = (v_1+v_2+v_3)^2 + (v_4+v_5)^2 + (v_6+v_7)^2 + (v_8+v_9)^2 + (v_{10}+v_{11}+v_{12})^2 \tag{3.58}$$

$$D = (v_4+v_5)^2 + (v_6+v_7)^2 + (v_8+v_9)^2 \tag{3.59}$$

Then we have

$$m_{\beta_1} = A^2 m_\lambda \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \tag{3.60}$$

$$\sigma_{\beta_1}^2 = 2A^2 C m_\lambda \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 \tag{3.61}$$

$$m_{\beta_2} = AB \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i} \tag{3.62}$$

$$\sigma_{\beta_2}^2 = 2A^2 D \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}. \tag{3.63}$$

So, we obtain

$$P_{\mathrm{AS}} = \Pr(\beta \le 0) \tag{3.64}$$

$$= Q\left(\frac{m_\beta}{\sqrt{\sigma_\beta^2}}\right) \tag{3.65}$$

$$= Q\left(\frac{m_{\beta_1}+m_{\beta_2}}{\sqrt{\sigma_{\beta_1}^2+\sigma_{\beta_2}^2}}\right) \tag{3.66}$$

$$= Q\left(\frac{A^2 m_\lambda \sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+AB\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2A^2C\left(\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}\right)^2 m_\lambda+2A^2D\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}\right) \tag{3.67}$$

$$= Q\left(\frac{A m_\lambda \sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+B\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2C\left(\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}\right)^2 m_\lambda+2D\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}\right), \tag{3.68}$$

which completes this example.

For a special case (a, b), where b equals a multiple of a and all b unsatisfied check nodes are evenly distributed among all a variable nodes, we can derive a general expression of the factors A, B, C and D as follows.

Both $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}_i^{(\mathrm{ex})}$ have $ad_v - b$ non-zero members. Every row of $\mathbf{V}$ has exactly $d_v - b/a - 1$ non-zero elements. So does each row of $\mathbf{VC}$, since right-multiplying a permutation matrix $\mathbf{C}$ to $\mathbf{V}$ simply permutes the columns of $\mathbf{V}$. Therefore, by the Perron-Frobenius theorem [56, 29], $\mu_{\max} = d_v - b/a - 1$ and the corresponding unit-length eigenvector will be $\mathbf{v}_{\max} = \frac{1}{\sqrt{ad_v-b}}[1, 1, \ldots, 1]^{\mathrm{T}}$. Hence,

$$\beta = \left(\left(\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+\sum_{i=1}^{I}\left(\frac{1}{\mu_{\max}^i}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{v}_{\max}\right)\mathbf{1}_{\max}^{\mathrm{T}}\right)\cdot\mathbf{1} \tag{3.69}$$

$$= \frac{1}{ad_v-b}\left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{1}\right)\left(\mathbf{1}^{\mathrm{T}}\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+\frac{1}{ad_v-b}\sum_{i=1}^{I}\frac{1}{\mu_{\max}^i}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{1}\right)\left(\mathbf{1}^{\mathrm{T}}\mathbf{1}\right) \tag{3.70}$$

$$= \left(\boldsymbol{\lambda}^{\mathrm{T}}\mathbf{1}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+\sum_{i=1}^{I}\frac{1}{\mu_{\max}^i}\left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}}\mathbf{1}\right) \tag{3.71}$$

$$= \left(\left(d_v-\frac{b}{a}\right)\sum_{j=1}^{a}\lambda_j\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+\sum_{i=1}^{I}\frac{1}{\mu_{\max}^i}\left(\left(d_v-\frac{b}{a}\right)\sum_{j=1}^{a}\sum_{k=1}^{b/a}\lambda_{ij,k}^{(\mathrm{ex})}\right) \tag{3.72}$$

$$= \left(d_v-\frac{b}{a}\right)\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}\sum_{j=1}^{a}\lambda_j+\left(d_v-\frac{b}{a}\right)\sum_{i=1}^{I}\frac{1}{\mu_{\max}^i}\sum_{j=1}^{a}\sum_{k=1}^{b/a}\lambda_{ij,k}^{(\mathrm{ex})}. \tag{3.73}$$

Then

$$m_\beta = a\left(d_v-\frac{b}{a}\right)m_\lambda\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}+b\left(d_v-\frac{b}{a}\right)\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i} \tag{3.74}$$

$$\sigma_\beta^2 = a\sigma_\lambda^2 \left(\left(d_v - \frac{b}{a} \right) \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 + b \left(d_v - \frac{b}{a} \right)^2 \sum_{i=1}^{I} \frac{\sigma_{\lambda_i^{(\mathrm{ex})}}^2}{\mu_{\max}^{2i}} \tag{3.75}$$

$$= 2am_\lambda \left(d_v - \frac{b}{a} \right)^2 \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 + 2b \left(d_v - \frac{b}{a} \right)^2 \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}. \tag{3.76}$$

So

$$P_{\mathrm{AS}} = \Pr(\beta \leq 0) \tag{3.77}$$

$$= Q\left(\frac{m_\beta}{\sqrt{\sigma_\beta^2}} \right) \tag{3.78}$$

$$= Q\left(\frac{am_\lambda \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} + b \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2a \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 m_\lambda + 2b \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}}} \right). \tag{3.79}$$

The following lemma holds:

Lemma 3.2. *The largest eigenvalue of* $\mathbf{VC}$ *for the* $(8,8)$ *set is* $\mu_{\max} = \mathrm{Deg}(v) - 1 = d_v - 2 = 4$*, and its associated eigenvector is* $\mathbf{v}_{\max} = [1, 1, \ldots, 1]^{\mathrm{T}}$.

Proof. First write $\mathbf{VC} = 4\mathbf{M}$. By inspection $\mathbf{M}$ is a probability matrix, i.e., the sum of each row equals unity. $\mathbf{M}$ is irreducible due to the facts that $\mathbf{V}$ denotes a strongly connected graph and $\mathbf{C}$ is a permutation matrix. As a special case of the Perron-Frobenius theorem it is known that the largest eigenvalue of a probability matrix is 1, therefore the largest eigenvalue of $\mathbf{VC}$ equals 4. By inspection $\mathbf{VC}[1, 1, \ldots, 1]^{\mathrm{T}} = 4[1, 1, \ldots, 1]^{\mathrm{T}}$. □

In the case of the $(8,8)$ absorption set, $\mathbf{v}_{\max} = \frac{1}{\sqrt{40}}[1, 1, \ldots, 1]^{\mathrm{T}}$ and

$$\beta = \left(\sum_{i=0}^{I} \left(\mu_{\max}^i \left(\frac{5}{\sqrt{40}} \sum_{j=1}^{8} \lambda_j \right) \mathbf{v}_{\max}^{\mathrm{T}} + \mu_{\max}^{I-i} \left(\frac{5}{\sqrt{40}} \sum_{j=1}^{8} \lambda_{ij}^{(\mathrm{ex})} \right) \mathbf{v}_{\max}^{\mathrm{T}} \right) \right) \cdot \mathbf{1} \tag{3.80}$$

$$= \left(\sum_{i=0}^{I} \left(\mu_{\max}^i \left(\frac{5}{40} \sum_{j=1}^{8} \lambda_j \right) \cdot \mathbf{1}^{\mathrm{T}} + \mu_{\max}^{I-i} \left(\frac{5}{40} \sum_{j=1}^{8} \lambda_{ij}^{(\mathrm{ex})} \right) \cdot \mathbf{1}^{\mathrm{T}} \right) \right) \cdot \mathbf{1} \tag{3.81}$$

$$= \sum_{i=0}^{I} \left(\mu_{\max}^i \left(5 \sum_{j=1}^{8} \lambda_j \right) + \mu_{\max}^{I-i} \left(5 \sum_{j=1}^{8} \lambda_{ij}^{(\mathrm{ex})} \right) \right) \tag{3.82}$$

$$= 5 \sum_{j=1}^{8} \sum_{i=0}^{I} \left(\mu_{\max}^i \lambda_j + \mu_{\max}^{I-i} \lambda_{ij}^{(\mathrm{ex})} \right) \tag{3.83}$$

$$= 5\mu_{\max}^I \sum_{j=1}^{8} \sum_{i=0}^{I} \left(\frac{\lambda_j}{\mu_{\max}^{I-i}} + \frac{\lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^i} \right) \tag{3.84}$$

$$= 5\mu_{\max}^I \sum_{j=1}^{8} \left(\sum_{i=0}^{I} \frac{\lambda_j}{\mu_{\max}^{I-i}} + \sum_{i=0}^{I} \frac{\lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^i} \right) \tag{3.85}$$

$$= 5\mu_{\max}^{I} \sum_{j=1}^{8} \left(\sum_{k=0}^{I} \frac{\lambda_j}{\mu_{\max}^{k}} + \sum_{i=0}^{I} \frac{\lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^{i}} \right) \tag{3.86}$$

$$= 5\mu_{\max}^{I} \sum_{j=1}^{8} \left(\sum_{i=0}^{I} \frac{\lambda_j}{\mu_{\max}^{i}} + \sum_{i=0}^{I} \frac{\lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^{i}} \right) \tag{3.87}$$

$$= 5\mu_{\max}^{I} \sum_{j=1}^{8} \sum_{i=0}^{I} \frac{\lambda_j + \lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^{i}} \tag{3.88}$$

$$\leq 0. \tag{3.89}$$

The eigenvalue $\mu_{\max} = d_v - 2$ is the *gain* of the absorption set and it is determined by the variable node degree.

Exact knowledge of $\lambda_{ji}^{(\mathrm{ex})}$ is not available to the analysis since these values depend on the received signals. However, assuming that the code structure extrinsic to the absorption set operates "regularly", we may substitute average values for the $\lambda_{ji}^{(\mathrm{ex})}$. Note that λ_i is Gaussian distributed from the channel, and that we may assume that $\lambda_{ji}^{(\mathrm{ex})}$ is also Gaussian distributed as is customary in density evolution analysis [15] [61]. Furthermore, like λ_i, we assume that $\lambda_{ji}^{(\mathrm{ex})}$ has a *consistent* Gaussian distribution with $\sigma^2 = 2m$, where m is the mean. We therefore only need the mean of $\lambda_{ji}^{(\mathrm{ex})}$, which we can calculate from a Gaussian density evolution calculation[3], i.e.,

$$m_{\lambda^{(\mathrm{ex})}}^{(i)} = \phi^{-1}\left(1 - \left[1 - \phi\left(m_\lambda + (d_v - 1)m_{\lambda^{(\mathrm{ex})}}^{(i-1)}\right)\right]^{d_c - 1}\right), \tag{3.90}$$

where $m_\lambda = 2E_s/\sigma^2$ is the mean of λ_i, $m_{\lambda^{(\mathrm{ex})}}^{(i)}$ is the mean of the extrinsic signal $\lambda_{ji}^{(\mathrm{ex})}$, and ϕ is the check node mean transfer function [61].

With the Gaussian assumptions and the fact that β is a linear combination of independent Gaussian variables, the probability of (3.89) happening can be calculated as

$$P_{\mathrm{AS}} = \Pr\left(\beta \leq 0\right) \tag{3.91}$$

$$= \Pr\left(\sum_{j=1}^{8} \sum_{i=0}^{I} \frac{\lambda_j + \lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^{i}} \leq 0\right) \tag{3.92}$$

$$= \Pr\left(\sum_{j=1}^{8} \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^{i}}\right) \lambda_j + \sum_{j=1}^{8} \sum_{i=0}^{I} \frac{\lambda_{ij}^{(\mathrm{ex})}}{\mu_{\max}^{i}} \leq 0\right) \tag{3.93}$$

$$= Q\left(\frac{m_\beta}{\sqrt{\sigma_\beta^2}}\right) \tag{3.94}$$

$$= Q\left(\frac{8\sum_{i=0}^{I} \frac{m_\lambda}{\mu_{\max}^{i}} + 8\sum_{i=0}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{i}}}{\sqrt{8\left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^{i}}\right)^2 \sigma_\lambda^2 + 8\sum_{i=0}^{I} \left(\frac{1}{\mu_{\max}^{i}}\right)^2 \sigma_{\lambda_{(i)}^{(\mathrm{ex})}}^2}}\right) \tag{3.95}$$

[3]For details and definitions, see [61, Chapter 11].

$$= Q\left(\frac{8\sum_{i=0}^{I}\frac{m_\lambda + m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{16\left(\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}\right)^2 m_\lambda + 16\sum_{i=0}^{I}\frac{1}{\mu_{\max}^{2i}} m_{\lambda^{(\mathrm{ex})}}^{(i)}}}\right) \tag{3.96}$$

$$= Q\left(\frac{2m_\lambda\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i} + 2\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{\left(\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}\right)^2 m_\lambda + \sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}\right), \tag{3.97}$$

which is a special case of (3.79) when plugging in $a = b = 8$.

The probability P_{AS} needs to be multiplied with the multiplicity factor of $14,272$ in order to obtain a union bound. In order to compute a BER estimate, we further multiply this number by $8/1723$, since there are eight errors that occur in a frame of $1,723$ bit errors due to this absorption set.

Figure 3.11 shows P_{AS} for the first most dominant absorption sets. Also shown are general tendencies of P_{AS} as a function of a and b:

$$P_{\mathrm{AS}} = Q\left(f(a,b)\right), \tag{3.98}$$

where $f(a,b)$ is defined by (3.99).

$$f(a,b) = \frac{am_\lambda\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i} + b\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2a\left(\sum_{i=0}^{I}\frac{1}{\mu_{\max}^i}\right)^2 m_\lambda + 2b\sum_{i=1}^{I}\frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}. \tag{3.99}$$

The number of non-zero elements in each row of matrix $\mathbf{V}$ for an (a,b) absorption set is determined by $\mathrm{Deg}(v_i),\ i = 1,2,\ldots,a$. From the equality in Definition 3.5 (ii), the average of all $\mathrm{Deg}(v_i)$ is

$$\overline{\mathrm{Deg}}(v) = d_v - \frac{b}{a}. \tag{3.100}$$

Thus by the Perron-Frobenius theorem again, the gain of the set $\mu_{\max}$ of $\mathbf{VC}$ can be closely approximated by:

$$\mu_{\max} \approx \overline{\mathrm{Deg}}(v) - 1 = d_v - \frac{b}{a} - 1 = 5 - \frac{b}{a}. \tag{3.101}$$

This was used in (3.99) and Figure 3.11 to plot the curves.

It can be seen that the $(8,8)$ absorption set is the most dominant, which is consistent with numerical observations. Additionally, some sets, like the majority of $(7,12)$ sets, are "contained" in larger sets, that is, such $(7,12)$ absorption sets are not stable under bit flipping and will evolve into $(8,8)$ sets, of which they are subgraphs. There are $179,648$ $(7,12)$ absorption sets. They all have the topology shown in Figure 3.10, which can be obtained from Figure 3.6(e) by removing one node. However, the $(8,8)$ sets generate $14272 \times 8 = 114176$ $(7,12)$ absorption sets (no duplicates). Hence there are $179648 - 114176 = 65472$ $(7,12)$ sets that are not contained in the $(8,8)$ ones.

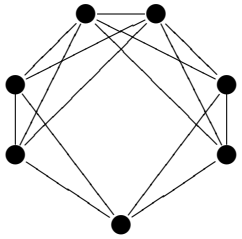


Figure 3.10: Topology of the $(7, 12)$ absorptions sets.

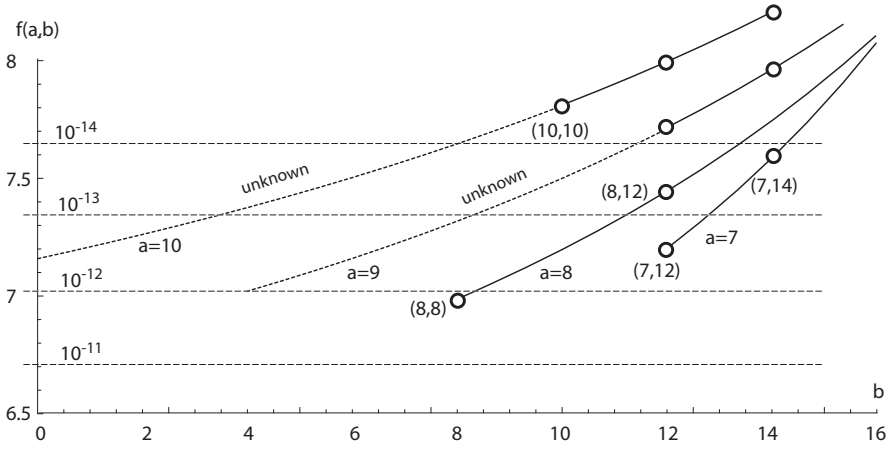


Figure 3.11: Error probability of the dominant absorption sets at $E_b/N_0 = 5$ dB and approximation functions based on a and b. (Curves are drawn only for possible or existing parameter combinations.)

To have an idea how dominant the $\mu_{\max}$ is, we calculated all forty eigenvalues μ_j of $\mathbf{VC}$, the product of (3.8) and (3.5), in (3.103).

$$\det(\mathbf{VC}) = \prod_{j=1}^{40} \mu_j \tag{3.102}$$

$$\begin{aligned} &= 4 \\ &\quad \times \left(\frac{1}{2} + \frac{\sqrt{15}}{2}i\right)^2 \left(\frac{1}{2} - \frac{\sqrt{15}}{2}i\right)^2 \left(-\frac{1}{2} + \frac{\sqrt{15}}{2}i\right)^4 \left(-\frac{1}{2} - \frac{\sqrt{15}}{2}i\right)^4 \\ &\quad \times \left(-\frac{3}{2} + \frac{\sqrt{7}}{2}i\right)\left(-\frac{3}{2} - \frac{\sqrt{7}}{2}i\right) \\ &\quad \times 1^{13}(-1)^{12}, \end{aligned} \tag{3.103}$$

with magnitudes 4, 2 and 1, respectively.

3.5.2 Numerical Verification

Figure 3.12 shows the analytical error floor calculation using (3.148) and the multiplicity of $14,272$. Lesser absorption sets have an impact more than an order of magnitude lower, and so they are not considered. The figure also shows hardware simulations using an FPGA platform, as well as importance sampled simulations using the same absorption sets as bias targets. Regular mean-shift importance sampling was utilized and each of the absorption sets containing a specific variable node was biased separately. As evidenced by the figure, our linearized analysis provides an accurate picture of the error floor behavior of this code and illustrates the dominance of the $(8, 8)$ absorption sets.

3.6 Error Floor of the Tanner $[155, 64, 14]$ Regular $(3, 5)$ LDPC Code

In [75] and [76], Tanner introduced a class of regular LDPC codes composed of blocks of permutation matrices, called circulant matrices. The parity-check matrix $\mathbf{H}$ of these codes has a $d_v \times d_c$ array of circulant permutation matrices $\mathbf{I}_{i,j}$, where each $\mathbf{I}_{i,j}$ denotes a $p \times p$ identity matrix with its rows shifted cyclically to the left.

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{0,0} & \mathbf{I}_{0,1} & \cdots & \mathbf{I}_{0,d_c-1} \\ \mathbf{I}_{1,0} & \mathbf{I}_{1,1} & \cdots & \mathbf{I}_{1,d_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_{d_v-1,0} & \mathbf{I}_{d_v-1,1} & \cdots & \mathbf{I}_{d_v-1,d_c-1} \end{bmatrix}_{d_v p \times d_c p} \tag{3.104}$$

Immediately, we can tell that each row of $\mathbf{H}$ has exactly d_c non-zero elements and each column d_v, making the code a regular (d_v, d_c) LDPC code. Its length is $d_c p$ and the number of check nodes is $d_v p$. Its "design" code rate is $1 - d_v/d_c$, however the actual rate R will be slightly higher in that within every row block, all p rows add to an all-one vector and at least $d_v - 1$ rows are linearly dependent.

The dimension of the base identity matrix p was originally designed to be prime to eliminate 4-cycles. In [76], p was extended to non-primes, where the girth, the shortest

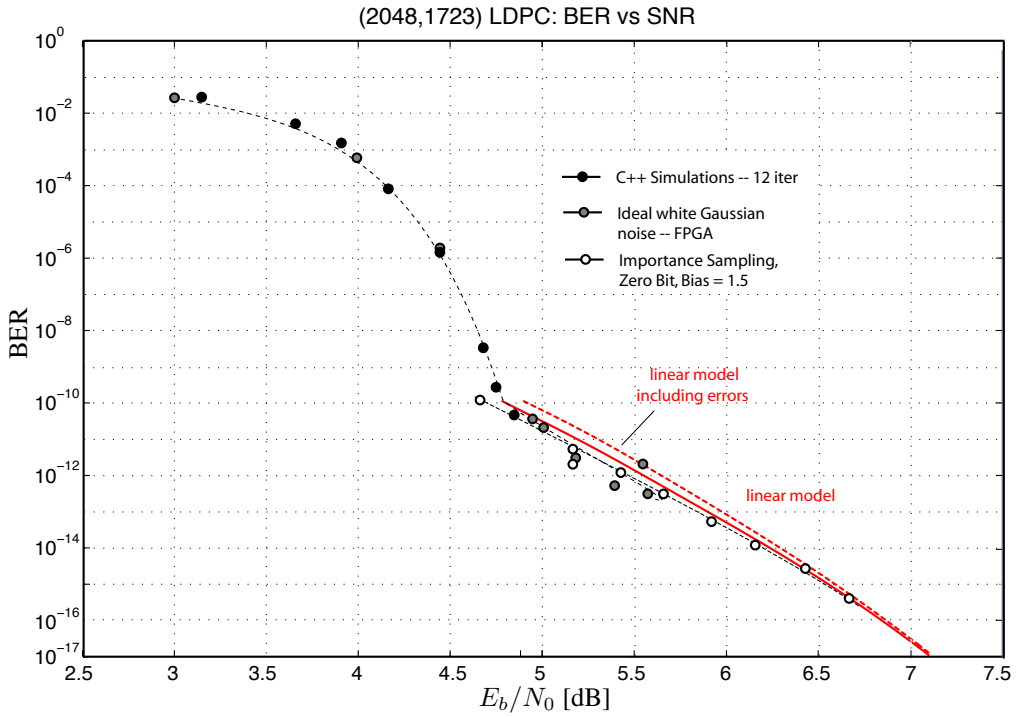


Figure 3.12: IS simulations, FPGA hardware simulations, and analytical error floor analysis for the $[2048, 1723]$ regular $(6, 32)$ LDPC code.

cycle in the Tanner graph and denoted by g, can be as short as 4. It can be shown that g is upper bounded by 12, no matter how big n is taken to be [75, 27]. The minimum distance of a Tanner code is bounded by $d_{\min} \leq (d_v + 1)!$ [53]. When d_v is small, this upper bound can be met by carefully selecting the parameters [53, 76]. Tanner code comes with relatively large girth and/or minimum distance, and provide good error floor performance. However, inherent structured flaws cause the existence of absorption sets.

Tanner presented a $[155, 64, 20]$ regular $(3, 5)$ LDPC code in the "Recent Results" session of 2010 IEEE International Symposium on Information Theory (ISIT 2010). Its block structure parity-check matrix is given by

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_4 & \mathbf{I}_8 & \mathbf{I}_{16} \\ \mathbf{I}_5 & \mathbf{I}_{10} & \mathbf{I}_{20} & \mathbf{I}_9 & \mathbf{I}_{18} \\ \mathbf{I}_{25} & \mathbf{I}_{19} & \mathbf{I}_7 & \mathbf{I}_{14} & \mathbf{I}_{28} \end{bmatrix}_{93 \times 155}, \tag{3.105}$$

where each $\mathbf{I}_x$ is derived by shifting the rows of a 31×31 identity matrix cyclically to the left by x positions.

Example 3.4. Let $\mathbf{I}$ denote a 5×5 identity matrix.

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.106}$$

Then $\mathbf{I}_1$ is obtained by shifting the rows of $\mathbf{I}$ cyclically to the left by one position:

$$\mathbf{I}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.107}$$

And so on and so forth,

$$\mathbf{I}_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.108}$$

$$\mathbf{I}_3 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.109}$$

$$\mathbf{I}_4 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.110}$$

$$\mathbf{I}_5 = \mathbf{I}_0 = \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.111}$$

Its binary structure is shown in Figure 1.3. This Tanner code has rate $R \approx 0.4129$ and is equipped with a large girth $g = 8$ and $d_{\min} = 20$. We will use this code as an example in this section.

The detailed generation of Tanner codes is explained in Appendix B.

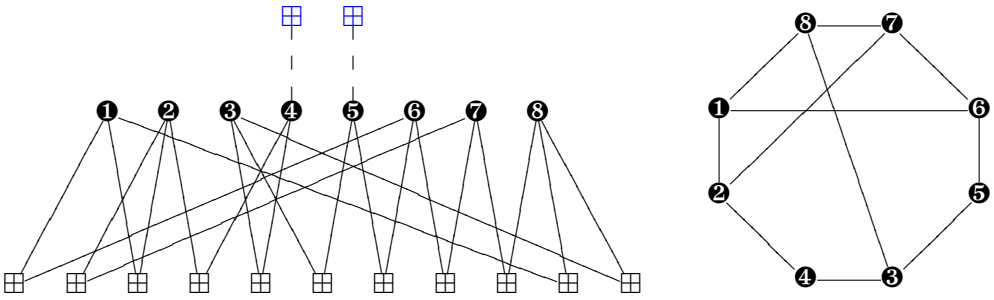
3.6.1 Absorption Set Identification

Following the enumerating methods in [63, 64], we list the first few absorption sets of this Tanner $[155, 64, 20]$, $(3, 5)$ LDPC code in Table 3.4. Figure 3.13 shows a subgraph induced by the $(8, 2)$ absorption set, which we believe to be the dominant absorption set of this code.

We observe that, from Figure 3.13(b), the $(8, 2)$ absorption set consists of cycles of lengths $8, 10, 12, 14$ and 16. In addition, it is also a hybrid of lower weight absorption sets. As a matter of fact, all $(4, 4)$, $(5, 3)$, $(6, 4)$ and $(7, 3)$ sets are contained in $(8, 2)$ sets. And 50% of $(5, 5)$, 4.1% of $(6, 6)$, 8.6% of $(7, 5)$ sets are contained in $(8, 2)$ sets, respectively. We claim that this $(8, 2)$ set is the dominant absorption/trapping set of this Tanner code on the Gaussian channel under iterative message-passing decoding. Actually, Figure 3.13(a) looks very much like a codeword in that if all those eight variable nodes are erroneous and the rest are correct, then all but two check nodes remain unsatisfied.

Table 3.4: First few absorption sets of the $[155, 64, 20]$ regular $(3, 5)$ Tanner code.

a	b	**Existence**	**Multiplicity**	**Gain** ($\mu_{\max}$)
< 4		No		
4	4	Yes	465	1
5	1	No		
	3	Yes	155	
	5		3,720	
6	2	No		
	4	Yes	930	
	6		22,630	1
7	1	No		
	3	Yes	930	
	5		16,275	
	7		140,430	1
8	2		465	1.7870
	4		5,115	
	6		196,540	
	8		823,515	1



(a) Check nodes shown.

(b) Check nodes hidden.

Figure 3.13: The topology of the $(8, 2)$ absorption set of the $[155, 64, 20]$ regular $(3, 5)$ Tanner code.

3.6.2 Linear Algebraic Estimation of the Error Rate

Now let us apply the error formulation to this Tanner code using the $(8, 2)$ absorption set structure.

The variable nodes in Figure 3.13 are labelled as 1 to 8. We also label the solid edges emanating from the variable nodes of Figure 3.13(a) as $1, 2, \ldots, 22$ from left to right. Denote the outgoing values from the variable nodes to the satisfied check nodes along these edges by x_i, i.e., x_1, x_2, x_3 leave variable node ❶, x_4, x_5, x_6 variable node ❷, etc. Collect the x_i in the length-22 column vector $\mathbf{x}$, which is the vector of outgoing variable edge values in the absorption set. Likewise, and analogously, let $\mathbf{y}$ be the incoming edge values to the variable nodes, such that y_j corresponds to the reverse-direction message on edge j, $j = 1, 2, \ldots, 22$. Now, at iteration $i = 0$,

$$\mathbf{x}_0 = \boldsymbol{\lambda}, \tag{3.112}$$

where the channel intrinsics vector $\boldsymbol{\lambda}$ is defined in (3.113).

$$\boldsymbol{\lambda} = \big[\underbrace{\lambda_1, \lambda_1, \lambda_1}_{\text{node ❶}}, \underbrace{\lambda_2, \lambda_2, \lambda_2}_{\text{node ❷}}, \underbrace{\lambda_3, \lambda_3, \lambda_3}_{\text{node ❸}}, \underbrace{\lambda_4, \lambda_4}_{\text{node ❹}}, \underbrace{\lambda_5, \lambda_5}_{\text{node ❺}}, \underbrace{\lambda_6, \lambda_6, \lambda_6}_{\text{node ❻}}, \underbrace{\lambda_7, \lambda_7, \lambda_7}_{\text{node ❼}}, \underbrace{\lambda_8, \lambda_8, \lambda_8}_{\text{node ❽}}\big]^{\mathrm{T}}. \tag{3.113}$$

It undergoes the following operation at the check node:

$$\mathbf{y}_0 = \mathbf{C}\mathbf{x}_0 = \mathbf{C}\boldsymbol{\lambda}, \tag{3.114}$$

where $\mathbf{C}$ is a permutation matrix that reflects the incoming messages back to the absorption set and is defined in (3.117). By induction, we obtain at iteration $i = I$:

$$\mathbf{x}_I = \sum_{i=0}^{I} \left((\mathbf{VC})^i \boldsymbol{\lambda} + (\mathbf{VC})^{I-i} \boldsymbol{\lambda}_i^{(\mathrm{ex})} \right), \tag{3.115}$$

where $\boldsymbol{\lambda}_i^{(\mathrm{ex})}$ is the extrinsics vector and defined in (3.116),

$$\boldsymbol{\lambda}_i^{(\mathrm{ex})} = \Big[\underbrace{0, 0, 0}_{\text{node ❶}}, \underbrace{0, 0, 0}_{\text{node ❷}}, \underbrace{0, 0, 0}_{\text{node ❸}}, \underbrace{\lambda_{i4}^{(\mathrm{ex})}, \lambda_{i4}^{(\mathrm{ex})}}_{\text{node ❹}}, \underbrace{\lambda_{i5}^{(\mathrm{ex})}, \lambda_{i5}^{(\mathrm{ex})}}_{\text{node ❺}}, \underbrace{0, 0, 0}_{\text{node ❻}}, \underbrace{0, 0, 0}_{\text{node ❼}}, \underbrace{0, 0, 0}_{\text{node ❽}}\Big]^{\mathrm{T}}, \tag{3.116}$$

and the variable node addition matrix $\mathbf{V}$ is defined in (3.118). Note that $(\mathbf{VC})^0 = \mathbf{I}$ and $\boldsymbol{\lambda}_0^{(\mathrm{ex})} = \mathbf{0}$.

$$\mathbf{C} = \begin{bmatrix}
 & & & & & & & & & & 1 & & & & & & & & & & \\
 & & 1 & & & & & & & & & & & & & & & & & 1 & \\
 & & & & & & & & & & & & & & & & 1 & & & & \\
 & 1 & & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & 1 & & & & & & & & & & & & \\
 & & & & & & & & & 1 & & & & & & & & & & & \\
 & & & & & & & & & & 1 & & & & & & & & & & 1 \\
 & & & & & 1 & & & & & & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & & & & & & \\
 & & & & & & & 1 & & & & & & & & & & & & & \\
 & & & & & & & & & & & & 1 & & & & & & & & \\
1 & & & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & 1 & & & & & & & & & & & \\
 & & & & & & & & & & & & & & 1 & & & & & & \\
 & & & 1 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & 1 & & & & & & & \\
 & & & & & & & & & & & & & & & & & 1 & & & \\
 & & & & & & & & & & & & & & & & 1 & & & & \\
 & & 1 & & & & & & & & & & & & & & & & & & \\
 & & & & & & 1 & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & & & & 1
\end{bmatrix}_{22\times 22} \tag{3.117}$$

$$\mathbf{V} = \begin{bmatrix}
0 & 1 & 1 & & & & & & & & & & & & & & & & & & & \\
1 & 0 & 1 & & & & & & & & & & & & & & & & & & & \\
1 & 1 & 0 & & & & & & & & & & & & & & & & & & & \\
 & & & 0 & 1 & 1 & & & & & & & & & & & & & & & & \\
 & & & 1 & 0 & 1 & & & & & & & & & & & & & & & & \\
 & & & 1 & 1 & 0 & & & & & & & & & & & & & & & & \\
 & & & & & & 0 & 1 & 1 & & & & & & & & & & & & & \\
 & & & & & & 1 & 0 & 1 & & & & & & & & & & & & & \\
 & & & & & & 1 & 1 & 0 & & & & & & & & & & & & & \\
 & & & & & & & & & 0 & 1 & & & & & & & & & & \\
 & & & & & & & & & 1 & 0 & & & & & & & & & & \\
 & & & & & & & & & & & 0 & 1 & 1 & & & & & & & \\
 & & & & & & & & & & & 1 & 0 & 1 & & & & & & & \\
 & & & & & & & & & & & 1 & 1 & 0 & & & & & & & \\
 & & & & & & & & & & & & & & 0 & 1 & 1 & & & & \\
 & & & & & & & & & & & & & & 1 & 0 & 1 & & & & \\
 & & & & & & & & & & & & & & 1 & 1 & 0 & & & & \\
 & & & & & & & & & & & & & & & & & 0 & 1 & 1 & \\
 & & & & & & & & & & & & & & & & & 1 & 0 & 1 & \\
 & & & & & & & & & & & & & & & & & 1 & 1 & 0 &
\end{bmatrix}_{22\times 22} \tag{3.118}$$

Now we calculate the maximum eigenvalue $\mu_{\max}$ of $\mathbf{VC}$ and its corresponding unit-length eigenvector $\mathbf{v}_{\max}$.

We list all eigenvalues and their norms of $\mathbf{VC}$, the product of (3.118) and (3.117), in

(3.119).

$$\boldsymbol{\mu} = \begin{bmatrix} 1.7870 \\ 1.4142i \\ -1.4142i \\ -1.2733 + 0.4949i \\ -1.2733 - 0.4949i \\ 0.2565 + 1.3239i \\ 0.2565 - 1.3239i \\ 0.5734 + 1.1433i \\ 0.5734 - 1.1433i \\ -0.9501 + 0.8428i \\ -0.9501 - 0.8428i \\ 0.4551 + 1.0987i \\ 0.4551 - 1.0987i \\ -0.4551 + 1.0987i \\ -0.4551 - 1.0987i \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}_{22\times 1}, \qquad |\boldsymbol{\mu}| = \begin{bmatrix} 1.7870 \\ 1.4142 \\ 1.4142 \\ 1.3660 \\ 1.3660 \\ 1.3485 \\ 1.3485 \\ 1.2790 \\ 1.2790 \\ 1.2700 \\ 1.2700 \\ 1.1892 \\ 1.1892 \\ 1.1892 \\ 1.1892 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}_{22\times 1}. \tag{3.119}$$

Then the maximum eigenvalue is denoted as $\mu_{\max}$ with corresponding eigenvector $\mathbf{v}_{\max}$.

$$\mu_{\max} \approx 1.7870 \tag{3.120}$$

$$\mathbf{v}_{\max} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \\ v_{13} \\ v_{14} \\ v_{15} \\ v_{16} \\ v_{17} \\ v_{18} \\ v_{19} \\ v_{20} \\ v_{21} \\ v_{22} \end{bmatrix} \approx \begin{bmatrix} 0.2369 \\ 0.2369 \\ 0.2273 \\ 0.2031 \\ 0.2031 \\ 0.2651 \\ 0.2254 \\ 0.2254 \\ 0.1660 \\ 0.1261 \\ 0.1483 \\ 0.1483 \\ 0.1261 \\ 0.2031 \\ 0.2651 \\ 0.2031 \\ 0.2369 \\ 0.2369 \\ 0.2273 \\ 0.2201 \\ 0.2201 \\ 0.2544 \end{bmatrix}. \tag{3.121}$$

Therefore, applying the same argument used in (3.29) [63, 64], the $(8, 2)$ absorption set falls in error if

$$\beta = \mathbf{x}_I^{\mathrm{T}} \cdot \mathbf{1} \leq 0. \tag{3.122}$$

Applying the spectral theorem

$$(\mathbf{VC})^i \boldsymbol{\lambda} \to \mu_{\max}^i \left(\boldsymbol{\lambda}^{\mathrm{T}} \mathbf{v}_{\max}\right) \mathbf{v}_{\max}, \tag{3.123}$$

we can approximate β using (3.124). The validity of this approximation will be revisited in Section 3.7.3. For simplicity, once again, we separate this expression into two parts as shown in (3.125).

$$\beta \approx \mu_{\max}^I \Big[\underbrace{\left(\left(\left(\boldsymbol{\lambda}^{\mathrm{T}} \mathbf{v}_{\max}\right) \mathbf{v}_{\max}^{\mathrm{T}}\right) \cdot \mathbf{1}\right) \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i}}_{\triangleq \beta_1} + \underbrace{\sum_{i=1}^{I} \left(\frac{1}{\mu_{\max}^i} \left(\boldsymbol{\lambda}_i^{(\mathrm{ex})^{\mathrm{T}}} \mathbf{v}_{\max} \right) \mathbf{v}_{\max}^{\mathrm{T}} \right) \cdot \mathbf{1}}_{\triangleq \beta_2} \Big] \tag{3.124}$$

$$= \mu_{\max}^I \left(\beta_1 + \beta_2\right), \tag{3.125}$$

where β_1 and β_1 can be expanded as

$$\begin{aligned} \beta_1 = \; & \left((v_1 + v_2 + v_3)\lambda_1 + (v_4 + v_5 + v_6)\lambda_2 + (v_7 + v_8 + v_9)\lambda_3 \right. \\ & + (v_{10} + v_{11})\lambda_4 + (v_{12} + v_{13})\lambda_5 + (v_{14} + v_{15} + v_{16})\lambda_6 \\ & \left. + (v_{17} + v_{18} + v_{19})\lambda_7 + (v_{20} + v_{21} + v_{22})\lambda_8 \right) \left(\sum_{j=1}^{22} v_j \right) \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right) \end{aligned} \tag{3.126}$$

$$\beta_2 = \sum_{i=1}^{I} \frac{1}{\mu_{\max}^i} \left((v_{10} + v_{11})\, \lambda_{i4}^{(\mathrm{ex})} + (v_{12} + v_{13})\, \lambda_{i5}^{(\mathrm{ex})} \right) \left(\sum_{j=1}^{22} v_j \right). \tag{3.127}$$

Then, we obtain the means and variances of β_1 and β_2, individually, as shown in (3.128)–(3.131).

$$m_{\beta_1} = m_\lambda \left(\sum_{j=1}^{22} v_j \right)^2 \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right) \tag{3.128}$$

$$m_{\beta_2} = (v_{10} + v_{11} + v_{12} + v_{13}) \left(\sum_{j=1}^{22} v_j \right) \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i} \tag{3.129}$$

$$\begin{aligned} \sigma_{\beta_1}^2 = {} & 2m_\lambda \Big((v_1 + v_2 + v_3)^2 + (v_4 + v_5 + v_6)^2 + (v_7 + v_8 + v_9)^2 \\ & + (v_{10} + v_{11})^2 + (v_{12} + v_{13})^2 + (v_{14} + v_{15} + v_{16})^2 \\ & + (v_{17} + v_{18} + v_{19})^2 + (v_{20} + v_{21} + v_{22})^2 \Big) \left(\sum_{j=1}^{22} v_j \right)^2 \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 \end{aligned} \tag{3.130}$$

$$\sigma_{\beta_2}^2 = 2 \left((v_{10} + v_{11})^2 + (v_{12} + v_{13})^2 \right) \left(\sum_{j=1}^{22} v_j \right)^2 \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}. \tag{3.131}$$

Let

$$A = \sum_{j=1}^{22} v_j = 4.6052 \tag{3.132}$$

$$B = v_{10} + v_{11} + v_{12} + v_{13} = 0.5489 \tag{3.133}$$

$$\begin{aligned} C = {} & (v_1 + v_2 + v_3)^2 + (v_4 + v_5 + v_6)^2 + (v_7 + v_8 + v_9)^2 + (v_{10} + v_{11})^2 \\ & + (v_{12} + v_{13})^2 + (v_{14} + v_{15} + v_{16})^2 + (v_{17} + v_{18} + v_{19})^2 + (v_{20} + v_{21} + v_{22})^2 \\ = {} & 2.8981 \end{aligned} \tag{3.134}$$

$$D = (v_{10} + v_{11})^2 + (v_{12} + v_{13})^2 = 0.1507. \tag{3.135}$$

Then (3.128)–(3.131) simplify to

$$m_{\beta_1} = A^2 \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} m_\lambda \tag{3.136}$$

$$\sigma_{\beta_1}^2 = 2A^2 C \left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} \right)^2 m_\lambda \tag{3.137}$$

$$m_{\beta_2} = AB \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^i} \tag{3.138}$$

$$\sigma_{\beta_2}^2 = 2A^2 D \sum_{i=1}^{I} \frac{m_{\lambda^{(\mathrm{ex})}}^{(i)}}{\mu_{\max}^{2i}}. \tag{3.139}$$

So, the probability of this absorption set falling in error is given by:

$$P_{\text{AS}} = \Pr\left(\beta \leq 0\right) \tag{3.140}$$

$$= Q\left(\frac{m_\beta}{\sqrt{\sigma_\beta^2}}\right) \tag{3.141}$$

$$= Q\left(\frac{m_{\beta_1} + m_{\beta_2}}{\sqrt{\sigma_{\beta_1}^2 + \sigma_{\beta_2}^2}}\right) \tag{3.142}$$

$$= Q\left(\frac{Am_\lambda \sum_{i=0}^{I} \frac{1}{\mu_{\max}^i} + B\sum_{i=1}^{I} \frac{m_{\lambda^{(\text{ex})}}^{(i)}}{\mu_{\max}^i}}{\sqrt{2Cm_\lambda\left(\sum_{i=0}^{I} \frac{1}{\mu_{\max}^i}\right)^2 + 2D\sum_{i=1}^{I} \frac{m_{\lambda^{(\text{ex})}}^{(i)}}{\mu_{\max}^{2i}}}}\right). \tag{3.143}$$

The factors A, B, C and D are determined by the absorption set topology. The knowledge of (a, b) and d_v is concealed in them. For this Tanner code, B or D are much smaller than A or C, compared to the case of the dominant absorption set of the IEEE 802.3an LDPC code as shown in (3.97) or, the more general form, (3.79). This implies that the critical extrinsic information has very low impact on the error rate of the absorption set. Therefore, it would be more difficult to improve this error rate than that of the IEEE 802.3an code.

3.7 Error Probability Formula Refinement

Besides the relative simplicity of (3.143), one of the important insights we gain is that $\mu_{\max}$ and $\mathbf{v}_{\max}$ play a crucial role in the decoding failure mechanism. Absorption sets with large $\mu_{\max}$ tend to have more impact on the code performance. However, we do notice that there are a few places where the error formula can be improved. In this section, we revisit its derivation to obtain a more accurate error probability estimation formula.

3.7.1 External Variable Nodes

A couple of refinements can be added to this analysis, taking into account the activity of the variable nodes that are incident to an absorption set.

The exchange of extrinsics through the matrix $\mathbf{C}$ is an approximation in two ways: (i) as long as the remaining $d_c - 2$ inputs to the "satisfied" check node are relatively small, the entries of $\mathbf{C}$ are strictly less than unity, and, (ii) in case one of the extrinsic incoming check node messages has the wrong polarity, the returned signal to the absorption set switches polarity.

Case (i) is approached as follows. Using a Taylor series approximation we show that the quintessential check node operation

$$\tanh^{-1}\left(\tanh(x)\prod_{i=1}^{d_c-2}\tanh(x_i)\right) = \prod_{i=1}^{d_c-2}\tanh(x_i)\,x + \mathcal{O}\left[x^3\right], \tag{3.144}$$

where $\prod_{i=1}^{d_c-2} \tanh(x_i)$ can be interpreted as a *check node gain*. If we use for x_i the mean $m^{(i)}_{\mu^{(\text{ex})}}$ of the signals $\mu^{(\text{ex})}$ from the variable to the check nodes, an average gain can be computed as

$$g_i = \mathbb{E}\left[\prod_{i=1}^{d_c-2} \tanh\left(\frac{m^{(i)}_{\mu^{(\text{ex})}}}{2}\right)\right] \tag{3.145}$$

$$= \mathbb{E}\left[\tanh\left(\frac{m^{(i)}_{\mu^{(\text{ex})}}}{2}\right)\right]^{d_c-2} \tag{3.146}$$

$$= \left(1-\phi\left(m^{(i)}_{\mu^{(\text{ex})}}\right)\right)^{d_c-2}, \tag{3.147}$$

where the last equality results from the definition of the density evolution function $\phi(\cdot)$. With this result the probability in (3.97) is modified to (3.148), using the formula to estimate the error rate of the IEEE 802.3an LDPC code as an example.

$$P_{\text{AS}} = Q\left(\frac{2m_\lambda + 2\sum_{j=1}^{I}\left(\frac{m^{(j)}_{\lambda^{(\text{ex})}} + m_\lambda}{\mu^j_{\max}}\prod_{l=1}^{j}\frac{1}{g_l}\right)}{\sqrt{\left(1+\sum_{j=1}^{I}\prod_{l=1}^{j}\frac{1}{g_l\mu_{\max}}\right)^2 m_\lambda + \sum_{j=1}^{I} m^{(j)}_{\lambda^{(\text{ex})}}\left(\prod_{l=1}^{j}\frac{1}{g_l\mu_{\max}}\right)^2}}\right). \tag{3.148}$$

In the case of general (a, b) sets we need to work with (3.29) instead, and compute $\mu_{\max}$ and $\mathbf{v}_{\max}$ numerically using the set topology.

Case (ii) can be handled by the linear analysis as well in the following way. If an external variable to the absorption set has an incorrect sign, this reverses the polarity of the signal returned to the absorption set from that particular check node. During the first iteration, these extrinsic signals are basically the received channel LLRs from the connected variable nodes. The probability that these are in error is given by the raw bit error rate

$$P_{\text{e}} = Q\left(\sqrt{2\frac{E_s}{N_0}}\right). \tag{3.149}$$

There are $d_c - 2 = 30$ external inputs impinging on each check node of the absorption set, therefore the probability that a returned signal experiences a polarity reversal is given by

$$P_p = \sum_{k=1}^{15}\binom{30}{2k-1} P_e^{2k-1}\left(1-P_e\right)^{31-2k}. \tag{3.150}$$

The model in (3.148) can now be expanded by injecting a correction value into the absorption set node whenever an external value is in error. We assume that if a polarity reversal occurs, the minimum value of the check node is likely to be close to zero, therefore the injected correction value needs to cancel the absent feedback signal and is set to $-\lambda_{\text{ex},i}/\mu_{\max}$.

If k check nodes are in error, then $2k$ correction values are injected, one for each message going back to the absorption set. The injected correction values will alter the mean value of the decision variable to

$$\text{mean} \to 8\left(m_\lambda\left(1-\frac{k}{4\mu_{\max}}\right)+\sum_{j=1}^{I}\left(\frac{m_{\lambda^{(\text{ex})}}^{(j)}}{\mu_{\max}^{j}}\prod_{l=1}^{j}\frac{1}{g_l}\right)\right) \tag{3.151}$$

and the variance is adjusted accordingly, where care needs to be taken how the correction values accumulate. We have used an upper bound on the variance.

Note that these modifications only include check node polarity reversal at the first iteration, but an extension to subsequent iterations is straight-forward if perhaps messy. Furthermore, as seen in Figure 3.12 (dashed curves), the addition of this mechanism has only a minor effect on the results.

3.7.2 P_{AS} Reinforcement

We interpreted the probability of (3.122) or (3.29) happening to constitute the probability that an absorption set falls in error in [63, 64]. This condition can be strengthened because at the last iteration I, all elements of $\mathbf{x}_I$, defined in (3.115), are negative by virtue of (3.120) and (3.121). So the failure probability is more accurately given as

$$P_{\text{AS}} = \Pr(\mathbf{x}_I \leq \mathbf{0}). \tag{3.152}$$

In addition, all elements of $\mathbf{x}_I$ are a linear combination of $\lambda_1, \lambda_2, \ldots, \lambda_8$ and $\lambda_{i4}^{(\text{ex})}, \lambda_{i5}^{(\text{ex})}$, where $i = 1, 2, \ldots, I$, the elements of (3.113) and (3.116) in the case of the $[155, 64, 20]$ Tanner code, without loss of generality. In other words, they are linearly dependent. Therefore, (3.152) is equivalent to saying

$$\begin{aligned} P_{\text{AS}} &= \Pr\left(\max_{j=1}^{ad_v-b}(x_{I,j}) \leq 0\right) && (3.153)\\ &= \Pr\left(\max_{j=1}^{22}(x_{I,j}) \leq 0\right). && (3.154) \end{aligned}$$

Analogously, we rewrite (3.115) into vectors $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$.

$$\begin{aligned} \boldsymbol{\beta}_1 &= \boldsymbol{\lambda}+\sum_{i=1}^{I}(\mathbf{VC})^i\boldsymbol{\lambda} && (3.155)\\ &\approx \boldsymbol{\lambda}+\left[\left(\sum_{i=1}^{I}\mu_{\max}^i\right)\left(\boldsymbol{\lambda}^{\text{T}}\mathbf{v}_{\max}\right)\right]\mathbf{v}_{\max} && (3.156)\\ \boldsymbol{\beta}_2 &= \boldsymbol{\lambda}_I^{(\text{ex})}+\sum_{i=1}^{I-1}(\mathbf{VC})^i\boldsymbol{\lambda}_{I-i}^{(\text{ex})} && (3.157)\\ &\approx \boldsymbol{\lambda}_I^{(\text{ex})}+\sum_{i=1}^{I-1}\mu_{\max}^i\left(\boldsymbol{\lambda}_{I-i}^{(\text{ex})\text{T}}\mathbf{v}_{\max}\right)\mathbf{v}_{\max} && (3.158) \end{aligned}$$

If we take the mean of every entry of $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$, individually, we obtain

$$\mathbf{m}_{\boldsymbol{\beta}_1} = m_\lambda\mathbf{1}+m_\lambda A\left(\sum_{i=1}^{I}\mu_{\max}^i\right)\mathbf{v}_{\max} \tag{3.159}$$

$$\mathbf{m}_{\boldsymbol{\beta}_2} = m^{(I)}_{\boldsymbol{\lambda}^{(\mathrm{ex})}} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + B \sum_{i=1}^{I-1} \mu^i_{\max} m^{(I-i)}_{\boldsymbol{\lambda}^{(\mathrm{ex})}} \mathbf{v}_{\max}. \tag{3.160}$$

It can be seen that the maximum entry of $\mathbf{v}_{\max}$, shown in (3.121), v_6 or v_{15}, is dominating the elements of $\mathbf{m}_{\boldsymbol{\beta}_1} + \mathbf{m}_{\boldsymbol{\beta}_2}$. Thus (3.154) becomes

$$P_{\mathrm{AS}} = \Pr\left(x_{I,6} \leq 0\right). \tag{3.161}$$

Our refined error probability formula can be derived accordingly, using the same technique developed in Section 3.6.2. It will retain the form of (3.143), but is more accurate.

We point out that this modification will not change the result that we had obtained previously for the IEEE 802.3an LDPC code [63, 64]. Due to the more symmetric appearance of the $(8, 8)$ absorption set of the IEEE 802.3an code and (3.3) and (3.9), its $\mathbf{m}_{\boldsymbol{\beta}_1}$ remains the format shown in (3.159), whereas

$$\mathbf{m}_{\boldsymbol{\beta}_2} = m^{(I)}_{\boldsymbol{\lambda}^{(\mathrm{ex})}} \mathbf{1} + B \sum_{i=1}^{I-1} \mu^i_{\max} m^{(I-i)}_{\boldsymbol{\lambda}^{(\mathrm{ex})}} \mathbf{v}_{\max}. \tag{3.162}$$

Besides, the $\mathbf{v}_{\max}$ of the dominant absorption set of the IEEE 802.3an code is an all-one vector. Therefore, we have

$$\Pr\left(\mathbf{x}_I^{\mathrm{T}} \cdot \mathbf{1} \leq 0\right) = \Pr\left(\max_j^{40}(x_{I,j}) \leq 0\right), \tag{3.163}$$

in that case.

3.7.3 Spectral Approximation

In (3.23), (3.124), (3.156) and (3.158), we use the approximation

$$(\mathbf{VC})^i \boldsymbol{\lambda} \approx \mu^i_{\max} \left(\boldsymbol{\lambda}^{\mathrm{T}} \mathbf{v}_{\max}\right) \mathbf{v}_{\max}, \quad i = 1, 2, \ldots, I. \tag{3.164}$$

This is not very accurate when i is small. The results from the early iterations are of great importance. So we will drop this approximation and use the matrix formulation (3.115) when plotting the error rate estimates.

This refined formula predicting the error floor of the Tanner $[155, 64, 20], (3, 5)$ code is plotted in Figure 4.12. The blue circles represents Monte Carlo simulation on a standard sum-product decoder and the solid black curve depicts the error formula utilizing the same decoder configuration. In addition, the error rate of a min-sum decoding is presented in Figure 3.14.

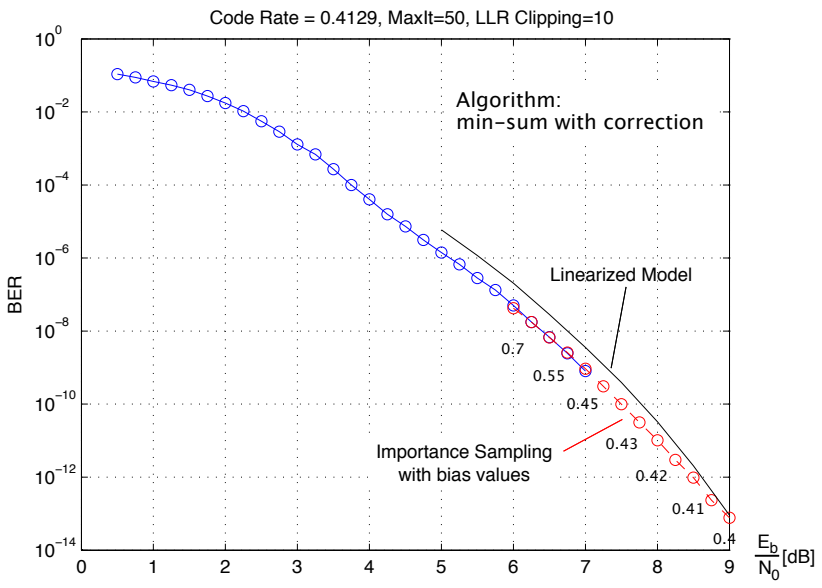


Figure 3.14: The numerical and analytical results of the performance of Tanner $[155, 64, 20]$, $(3, 5)$ LDPC code, assuming min-sum decoding and BPSK modulation on AWGN channel and maximum iteration=50.

3.8 Some Other LDPC Codes

During our search for a *good* code with a small length yet and an easy-to-reach error floor, we have found a few other interesting Tanner codes. For example, the error floor of a length-755 regular LDPC code is completely dominated by weight-$d_{\min}$ codewords. For two other codes with lengths between 100 and 200, it is possible to enumerate their codeword spectrums, whereas the error rates drop too fast with growing SNR to simulate the error floors due to their relatively large minimum distances among the codewords. Interested readers are directed to Appendix B for more information.

3.9 Error Patterns of Linear Programming Decoding

Analogous to absorption sets which are the reason for the error floor of LDPC codes under message-passing decoding, *pseudocodewords* cause the error floor under LP decoding. We first present the studies of [9, 13, 11] on LP decoding on the binary symmetric channel. [13] suggests that one class of the absorption sets of the Tanner $[155, 64, 20]$ regular $(3, 5)$ LDPC code is responsible for the error floor of the LP decoding on BSC. However, when we try to connect the failure patterns of LP decoding with the ones of message-passing decoding, we find out that such connection is unclear as an error pattern of the former is not necessarily an absorption set of the latter.

3.9.1 Linear Programming Decoding on BSC

In this section we will introduce a concept called the *instanton*, which is a binary vector that leads to a non-zero pseudocodeword in the LP decoder [9, 13, 11].

Let us represent the non-zero positions of a vector by its *support*:

Definition 3.6. The support of a vector $\mathbf{r} = [r_1, r_2, \ldots, r_n]$ is the set of its non-zero positions:

$$\mathrm{supp}(\mathbf{r}) = \{i | r_i \neq 0, i = 1, 2, \ldots, n.\}. \tag{3.165}$$

For a BSC, using the crossover probability shown in Figure 2.2, the log-likelihood ratio γ_i in (1.19) can be scaled as:

$$\gamma_i = \begin{cases} \ln\left(\frac{1-p}{p}\right) & \triangleq 1, \quad \text{if } \tilde{y}_i = 0; \\ \ln\left(\frac{p}{1-p}\right) = -\ln\left(\frac{1-p}{p}\right) & \triangleq -1, \quad \text{if } \tilde{y}_i = 1. \end{cases} \qquad i = 1, 2, \ldots, n. \tag{3.166}$$

Then the objective function in (2.77) becomes

$$\sum_{i=1}^{n} \gamma_i f_i = \sum_{i \notin \mathrm{supp}(\tilde{\mathbf{y}})} f_i - \sum_{i \in \mathrm{supp}(\tilde{\mathbf{y}})} f_i \tag{3.167}$$

$$\triangleq C(\tilde{\mathbf{y}}, \mathbf{f}). \tag{3.168}$$

Therefore, If $\tilde{\mathbf{y}}$ is the input, then the LP decoder converges to the pseudocodeword which minimizes $C(\tilde{\mathbf{y}}, \mathbf{f})$:

$$\mathrm{Decode}_{\mathrm{LP}}(\tilde{\mathbf{y}}) = \arg \min_{\mathbf{f} \in V(Q)} C(\tilde{\mathbf{y}}, \mathbf{f}). \tag{3.169}$$

Right away, we observe that

$$C(\tilde{\mathbf{y}}, \mathbf{0}) = 0. \tag{3.170}$$

So if there exists an $\mathbf{f}$ with $C(\tilde{\mathbf{y}}, \mathbf{f}) \leq 0$, then $\tilde{\mathbf{y}}$ will not converge to $\mathbf{0}$ under LP decoding, assuming that $\mathbf{0}$ was transmitted. However it does not necessarily imply that $\tilde{\mathbf{y}}$ has to converge to this particular $\mathbf{f}$.

Definition 3.7. For a non-zero pseudocodeword $\mathbf{f} = [f_1, f_2, \ldots, f_n]$, let e be the smallest number of components f_i such that their sum is at least $\sum_{i=1}^{n} f_i/2$. Then the BSC pseudocodeword weight of $\mathbf{f}$ is defined as

$$w_{\mathrm{BSC}}(\mathbf{f}) = \begin{cases} 2e, & \text{if } \sum_{e} f_i = \sum_{i=1}^{n} f_i/2; \\ 2e-1, & \text{if } \sum_{e} f_i > \sum_{i=1}^{n} f_i/2. \end{cases} \tag{3.171}$$

Definition 3.8. The median $M(\mathbf{f})$ of $\mathbf{f}$ is a binary vector with $\mathrm{supp}(M(\mathbf{f})) = \{i_1, i_2, \ldots, i_e\}$, such that $f_{i_1}, f_{i_2}, \ldots, f_{i_e}$ are the e largest components of $\mathbf{f}$.

Note that the median $M(\mathbf{f})$ is an integer vector, while $\mathbf{f}$ is not.

With this definition, the connection between (3.171) and (3.167) becomes more clear. Rewrite the rightmost part of (3.171) as

$$\sum_{e} f_i - \sum_{i=1}^{n} f_i/2 \geq 0 \tag{3.172}$$

$$\Rightarrow \quad 2\sum_{e} f_i - \sum_{i=1}^{n} f_i \geq 0 \tag{3.173}$$

$$\Rightarrow \quad \sum_{e} f_i - \sum_{n-e} f_i \geq 0 \tag{3.174}$$

$$\Rightarrow \quad \sum_{i \in \mathrm{supp}(M(\mathbf{f}))} f_i - \sum_{i \notin \mathrm{supp}(M(\mathbf{f}))} f_i \geq 0 \tag{3.175}$$

$$\Rightarrow \quad C(M(\mathbf{f}), \mathbf{f}) \leq 0 \tag{3.176}$$

$$\Rightarrow \quad \mathrm{Decode}_{\mathrm{LP}}\left(M(\mathbf{f})\right) \neq \mathbf{0}. \tag{3.177}$$

These weight-e binary vectors cause LP decoding to fail. Finding the ones with minimum weight will enable us calculate the error probability (2.84). This leads to our next definition.

Definition 3.9. The BSC instanton is defined as a binary error vector that will not be corrected by LP decoding, and any vector reduced from this vector by flipping any of its non-zero entries to zero, will converge to $\mathbf{0}$.

In other words, instantons are the "minimal-weight" error patterns under LP decoding. Any vector with support contained in the support of an instanton will be corrected, whereas any vector with support containing the support of an instanton will be decoded to a non-zero pseudocodeword.

Instantons can be found algorithmically as follows.

Step 1. Randomly choose a binary vector as the initial input. If it decodes to a non-zero pseudocodeword $\mathbf{f}$, then calculate its median $M(\mathbf{f})$. This binary vector $M(\mathbf{f})$ cannot be corrected by LP decoding.

Step 2. Use $M(\mathbf{f})$ as an input now. Once again, $M(\mathbf{f})$ does not have to converge to $\mathbf{f}$. If $M(\mathbf{f})$ is decoded to some pseudocodeword $\mathbf{f}'$, it can be shown that the weight of this $\mathbf{f}'$ must be no more than the weight of $\mathbf{f}$ [11].

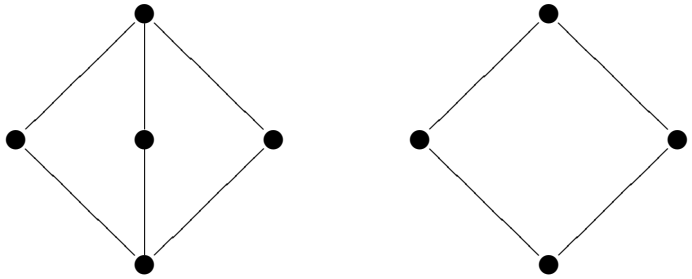
Step 3. If $w_{\mathrm{BSC}}(\mathbf{f}') < w_{\mathrm{BSC}}(\mathbf{f})$, then calculate $M(\mathbf{f}')$ and go back to Step 2. If $w_{\mathrm{BSC}}(\mathbf{f}') = w_{\mathrm{BSC}}(\mathbf{f})$, then flip any one of non-zero bits of $M(\mathbf{f}')$ to zero. Put all these lower-weight binary vectors derived from $M(\mathbf{f}')$ through the LP decoder. If they all converge to $\mathbf{0}$, then $M(\mathbf{f}')$ is what we are looking for, a so-called instanton. If any one of them cannot be corrected, then use it as the new input and go back to Step 2.

It can be shown that the weight of the pseudocodeword used in each iteration is strictly decreasing. Thus this is not an infinite search loop. Moreover, this algorithm terminates in at most $2k_0$ steps, where k_0 is the weight of the initial input binary vector.

We must point out that this algorithm is not guaranteed to find the minimal instantons nor to find all of the instantons. However, by Theorem 2.3 the weight of instantons is lower bounded by $\lceil d_{\mathrm{frac}}/2 \rceil$.

3.9.2 Relationship between LP and MP on BSC

What interest us are the numerical results on the $[155, 64, 20], (3, 5)$ Tanner code, obtained by [13]. Note that 155 distinct weight-5 instantons have been found. All of them share the same subgraph shown in Figure 3.15(a), which is merely an equivalent appearance of the ones shown in Figure 3.9. In addition, the number of this weight-5 instanton matches our record in Table 3.4.



(a) Weight-5 instanton. (b) $(4, 4)$ absorption set.

Figure 3.15: Error patterns of the $[155, 64, 20], (3, 5)$ Tanner code.

First, the topology in Figure 3.15(a) corresponds to the $(5, 3)$ absorption set under message-passing. It seems that LP decoding also tends to fail on absorption sets.

Secondly, by calculating the d_{frac} of the LP polytope of this Tanner code [13], it can be shown that these 155 sets are the minimal-weight error patterns that cannot be corrected by the LP decoder. In other words, any error pattern with weight less than 5 will be corrected by LP decoding. However, removing any one of the three degree-2 variable nodes from Figure 3.15(a) reduces it to a $(4, 4)$ absorption set, as shown in Figure 3.15(b). This weight-4 error pattern will still fail the message-passing (bit flipping) decoder, though it will lead to a weight-5 absorption set. So, in terms of correcting capability, LP decoding outperforms message-passing decoding on the BSC, at least on this code.

We point out that an error pattern for LP decoding may not be an absorption set for message-passing decoding, where such patterns are provided in the following examples.

Example 3.5. Assuming $d_v = 3$, we can construct a weight-9 instanton E as shown in Figure 3.16.

Consider $\mathbf{0}$ being transmitted and this E is received as $\tilde{\mathbf{y}}$ and input to an LP decoder. Considering bit-flipping decoding under BSC, we are able to find a point $\mathbf{f}$ in the polytope Q defined in (2.76), such that the objective function in (2.77) or (3.168) achieves a negative value.

1. Let $f_i = \begin{cases} 0, & i \notin E; \\ \frac{2}{5}, & i \in E. \end{cases}$ Hence (3.168) equals $-9 \times \frac{2}{5} < 0$.
2. We need to show that $\mathbf{f} \in Q$:
   (a) Obviously, the first condition in (2.75) is satisfied.
   (b) For any check node j, let $M(j) = N(j) \bigcap E$, where $N(j)$ is the set of neighboring variable nodes of j. Only the ten check nodes involved in Figure 3.16 need to be considered.

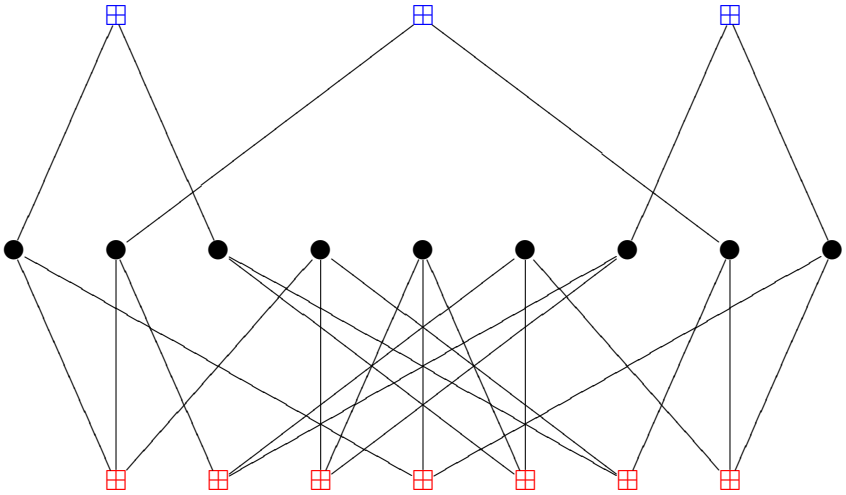


Figure 3.16: A size-9 instanton of LP decoding.

- For the satisfied check nodes j, $|M(j)|$ is even. Let

$$w_{j,S} = \begin{cases} \frac{2}{5}, & S = M(j); \\ \frac{3}{5}, & S = \emptyset; \\ 0, & \text{else.} \end{cases} \tag{3.178}$$

  Then (2.75) is satisfied at these j's.
- Regarding the unsatisfied check nodes j, let $M(j) = \{i_1, i_2, i_3\}$ and

$$w_{j,S} = \begin{cases} \frac{1}{5}, & S = \{i_1, i_2\} \text{ or } \{i_1, i_3\} \text{ or } \{i_2, i_3\}; \\ \frac{2}{5}, & S = \emptyset; \\ 0, & \text{else.} \end{cases} \tag{3.179}$$

  So (2.75) is satisfied at these j's, as well.

Therefore $\mathbf{f} \in Q$.

To be clear, this does not imply that the LP decoder will necessarily return this $\mathbf{f}$ as a decoding output, but rather than that it will definitely not converge to the all-0 codeword.

Note that Figure 3.16 in Example 3.5 shows a cycle-4-free topology. A smaller topology that contains 4-cycles is shown in Example 3.6.

Example 3.6. Still supposing $d_v = 3$, we can construct a weight-6 error patten as shown in Figure 3.17.

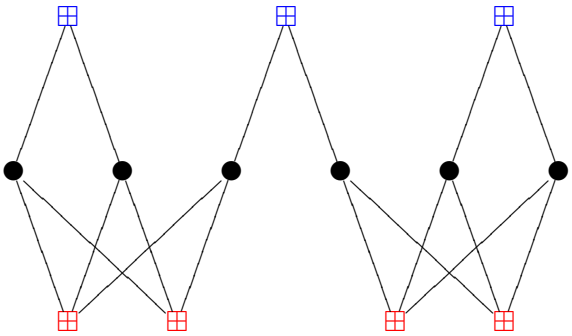


Figure 3.17: A size-6 instanton of LP decoding.

Chapter 4

Error Floor Reduction

After achieving the estimation formula (3.143), this general format helps us look for the contributing factors that come into play to establishing the error rate .

The absorption set structure determines the magnitude of $\mu_{\max}$, which affects how fast the power of $\mu_{\max}$ grows as it is usually greater than one. The coefficients A, B, C and D are the results of operations involving $\mathbf{v}_{\max}$, for instance, as shown in (3.132)–(3.135), and set topology. Hence, altering the code design to eliminate the minimal absorption set will affect the values of these parameters, thus improve the error rate.

The mean of the intrinsics m_λ is solely dependent on the channel characteristics, which is out of our control as an approach for tackling the error floor.

The maximum number of iterations allowed at the decoder, denoted by I, also shows up in (3.143). As we pointed out at the beginning of Chapter 3, it is commonly understood that simply increasing this value will not solve the trapping set problem, but only add complexity to the algorithm. This is because that the absorption set will stabilize after a few iterations. No matter how many more iterations we run the decoder at it, the error pattern will not break up and be corrected. To better demonstrate this, we recorded the accumulated LLRs at the variable nodes of an $(8, 8)$ absorption set of the IEEE 8023.an LDPC code and plotted them against iteration index in Figure 3.8, [64].[1]

The last variable appeared in the estimation formula is $m_{\boldsymbol{\lambda}^{(\mathrm{ex})}}$, the mean of the signals injected to the absorption set through its unsatisfied check nodes, shown in blue color in Figures 3.3(a) and 3.13(a). By observing (3.143), one way to reduce P_{AS} is to raise the value of $m^{(i)}_{\boldsymbol{\lambda}^{(\mathrm{ex})}}$. As a matter of fact, $\boldsymbol{\lambda}^{(\mathrm{ex})}$ carries the deemed correct information accumulated out of the absorption set. Thus, scaling up its value is equivalent to increasing its weight that would increase the chance of correcting the LLRs trapped within the absorption set's network. We also know from (3.143) that this has to be done during the first few iterations. Otherwise the exponential denominator $\mu^i_{\max}$ will diminish the effect quickly.

4.1 Extrinsics Scaling

By observing (3.143), increasing the value of $m^{(i)}_{\boldsymbol{\lambda}^{(\mathrm{ex})}}$ to reduce P_{AS} seems to be the only option since we can do nothing to the other arguments in (3.143) without changing the code

[1] Note that the decoding process would have stopped when the frame is converged, i.e., all variable nodes have positive LLRs. But we kept it running till the maximum iteration number was reached, just to have enough evidence to study the behavior of the LLRs.

design. So if we can boost up the values $\lambda_{ij}^{(\mathrm{ex})}$ which are sent to the absorption set through the unsatisfied check nodes, then P_{AS} can be decreased, therefore lowering the error floor. This has to be done during the first few iterations. Otherwise the exponential denominator $\mu_{\max}^{i}$ will diminish the effect quickly.

Sticking with the Tanner $[155, 64, 20], (3, 5)$ regular LDPC code, we modify the standard iterative SP decoding algorithm as follows:

1. During the initial 4 iterations, multiply the LLRs returning from all unsatisfied check nodes by a factor of 2;

2. After 4 iterations, revert back to the standard SP decoder.

The simulation results are shown in Figure 4.13. Due to the short length of this code, there is no visible turning point between waterfall and error floor regions. So we analyze the composition of the error patterns from the simulation results in Figure 4.14. For a standard SP decoding, the ratio of the dominant absorption set to all error events is approximately 30% at 4 dB and increases to nearly 80% at 6.25 dB, as shown in Figure 4.14(b). Applying the boosting SP will reduce this percentage by approximately 10%, as shown in Figure 4.14(b). The percentage of all absorption sets to the number of decoding failures also decreases 10% as can be seen in Figure 4.14(a). It also indicates that the error floor of this Tanner code starts showing up at 4 dB and is caused by absorption sets and, in particular, the $(8, 2)$ absorption set is the dominant one. Going back to Figure 4.13, it can be seen that the modified SP algorithm outperforms the original SP in the error floor region by nearly an order of magnitude at 6.5 dB.

We observe from Figure 4.13 that the boosting algorithm does not compromise the waterfall region. We also point out that the question of how many iterations to boost and how much to scale depends on the channel SNR and decoder configurations, such as the LLR clipping threshold, maximum iterations, etc., which can be seen in (3.143), as well.

4.2 LLR Range

It caught our attention that applying the boosting SP decoding algorithm will reduce the ratio of the dominant absorption set to all error events by approximately 10% only, compared to a standard SP decoder, as illustrated by a bar plot Figure 4.14(b), [85]. So the absorption sets are still the main contributing error patterns of the modified decoder.

When we look further, we realize that there is an additional contributing factor not shown up explicitly in (3.143), i.e., the LLR clipping threshold. Theoretically, the range of the messages $\lambda_{i}^{(\mathrm{ex})}$ can be unconstrained. However, to reduce the computational complexity, the $\lambda_{i}^{(\mathrm{ex})}$ are typically clipped at a value which is preset at the decoder. This threshold value serves as a hidden variable in (3.143) and will have an effect on the code performance.

The extrinsic message $\lambda^{(\mathrm{ex})}$ is definitely bound by this clipping value. Thanks to the fast growing LLRs in the iterative decoding algorithm, the magnitude of $\lambda^{(\mathrm{ex})}$ will quickly reach the typical clipping limit, say 10, within the first few iterations. In that case, the boosting action presented above has a very limited effect. As a consequence, the error rate of this Tanner code is reduced but still dominated by absorption sets.

This turns our focus on the relationship between LLR range and absorption sets. We also ran simulations of a standard SP decoder with increased clipping threshold, as shown in Figure 4.13, [85]. Then we achieved a significant reduction on the appearance of absorption

sets as error patterns. Fewer than 10%[2] of the observed decoding failures in the error floor region are caused by absorption sets when the LLRs are allowed to grow longer than 10 in magnitude, compared to 80–90% when the clipping threshold is set at 10, as shown in Figure 4.14(a), [85].

Furthermore, if we let the LLR grow freely at the decoder, utilizing the soft-bit decoding in the $\tanh$ domain [37], then there is a noticeable decoding gain, as can be seen in Figure 4.13, which is further supported by the error pattern statistics in Figure. 4.14. In other words, the higher the LLR clipping threshold, the less the threat from the absorption sets. Sooner or later, the extrinsic LLR $\lambda_i^{(\mathrm{ex})}$, which carries the correct information from outside of the absorption set, will grow reliable enough to win the majority vote at the absorption set nodes, thereby correcting this error pattern.

However, from Figure 4.13, there is not much difference between the blue and the green circles, where the LLRs are clipped at ± 10 and ± 100, respectively. Evidently the failures of the decoder with higher clipping threshold are not dominated by the absorption sets at all, as shown in Figure. 4.14. This also explains why the estimation using clipping threshold 100 does not match the simulation results, as shown in Figure 4.12. The black dashed curve in Figure 4.12 represents the contribution from the dominant absorption set, which does not dominate the decoding failures, to the error rate, when the LLRs are clipped at ± 100.

To understand why the bit error rate with raised LLR clipping limit is not improved while absorption sets are not dominating the failures any longer, we examine the numerical evidence hidden during the process of decoding an individual frame, as we did in the analysis for the IEEE 802.3an LDPC code in Figure 3.8 in [64].

Unlike the sum-product and soft-bit decoding [37] algorithms adopted in our earlier work [63, 64, 85], here we apply a corrected min-sum algorithm, which computes the check-to-variable LLR as

$$\lambda_{j\to i} = \left(\min_{l\in V_j\setminus\{i\}} |\lambda_{l\to j}| + \mathrm{CF} \right) \prod_{l\in V_j\setminus\{i\}} \mathrm{sign}\left(\lambda_{l\to j}\right), \tag{4.1}$$

where CF represents a correcting factor and is defined as

$$\mathrm{CF} = \begin{cases} -\dfrac{\ln\left(d_c-1\right)}{4}, & \text{if } \min\limits_{l\in V_j\setminus\{i\}} |\lambda_{l\to j}| \geq \dfrac{3\ln\left(d_c-1\right)}{8}; \\ 0, & \text{otherwise,} \end{cases} \tag{4.2}$$

in the *one-step degree-matched check node approximation* [38], in order to avoid any $\tanh(\cdot)$ calculation that causes a software saturation problem at points ± 1 as illustrated in Figure 4.1. This problem is also pointed out in [5].

Firstly, we caught an input frame that led to an $(8, 2)$ absorption set error event at the min-sum decoder, where the LLRs are limited between ± 10. Figure 4.2(a) shows the accumulated LLRs at the eight absorption set variable nodes against iterations. The errors or the set quickly locks up and stays erroneous for the rest of the iterations. When we pass exactly the same frame to the decoder however increase the LLR clipping threshold to 38, the frame converges to the all-zero codeword, as shown in Figure 4.2(b). Clearly we can see two phases, namely linear growth and bit flipping, the same as the case of the IEEE 802.3an code, shown in Figure 3.8.

[2]It turns out that this 10% can also go away if the maximum number of iterations, I, is also increased in conjunction with raised LLR clipping threshold.

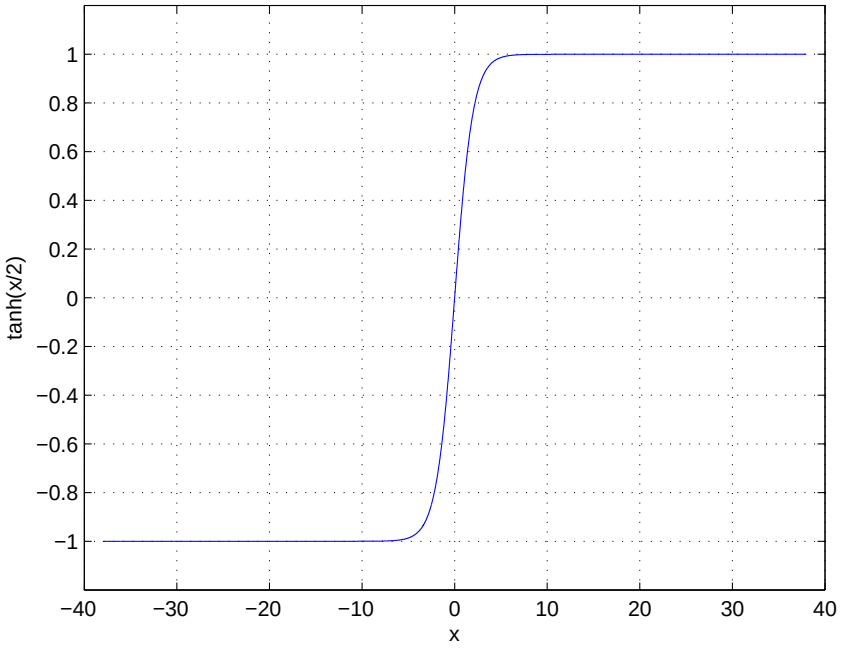


Figure 4.1: The $\tanh(x/2)$ value will be treated as 1 for all $|x| > 55 \ln 2 \approx 38$.

Secondly, we ran the same test on another frame that caused an $(8, 2)$ set when LLRs are confined between ± 10, as shown in Figure 4.3(a). Seemingly, there is no difference from the previous $(8, 2)$ set's behavior, shown in Figure 4.2(a). But when we raised the threshold to 38 for this second frame, the set surely behaves erratically, as seen in Figure 4.3(b). Note that after 50 iterations, some of the set nodes have positive LLRs, while some others do not. Those variable nodes with negative LLRs will be counted as errors but this error event will not be classified as an $(8, 2)$ absorption set.

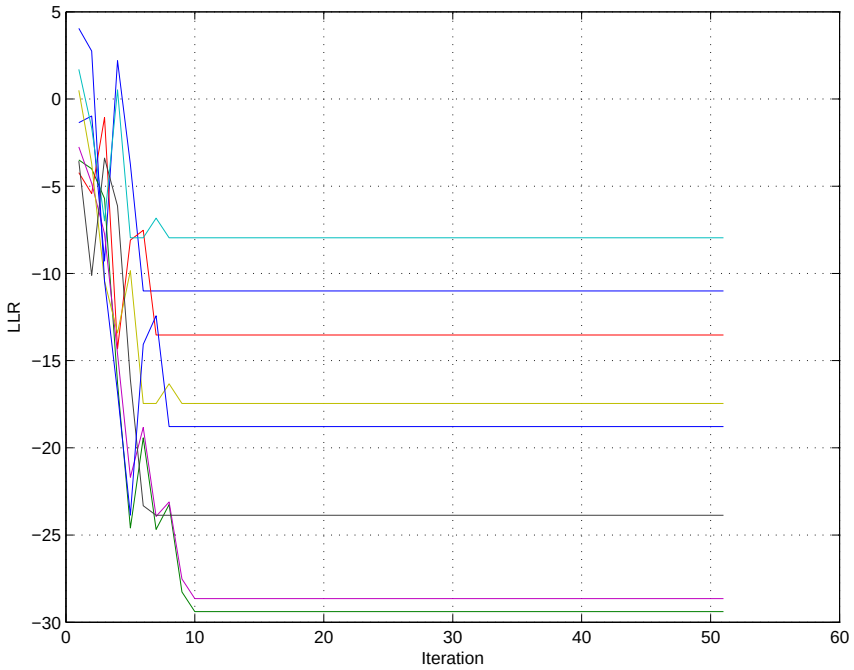
Then we looked further at the extrinsic LLRs entering this absorption set from the two unsatisfied check nodes that are trying to correct the errors. They grow quickly when clipping is at 10, but as erratically as the accumulated LLRs when clipping is 38, as seen in Figure 4.4.

To see how the oscillation evolves, we extended the iteration limit to 500. The second $(8, 2)$ absorption set gets corrected, eventually, after 100 iterations, as shown in Figure 4.5.

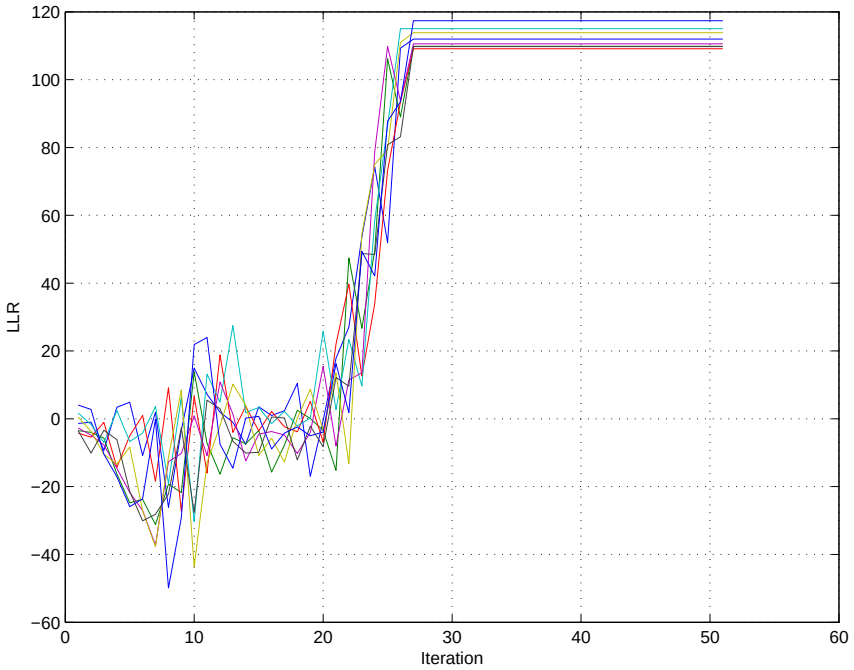
Therefore, the extrinsics need not only space but also time to grow strong enough to overwhelm the accumulated LLRs trapped within the absorption set's network.

The numerical results of the min-sum decoding algorithm with different LLR clipping thresholds and maximum iterations are shown in Figure 4.6. As can be seen, when $I = 50$, the curves are consistent with the ones in Figure 4.13, where sum-product decoding was applied. However, the error rate is significantly improved when I is raised to 500 in combination with LLR clipping at 38.

Figure 4.7 also shows the error rates of the Tanner $[155, 64, 20]$ LDPC code. Once again, with higher LLR clipping value, (3.143) predicts that the error rate will decrease accordingly. This is supported by IS when the clipping threshold is raised from 10 to

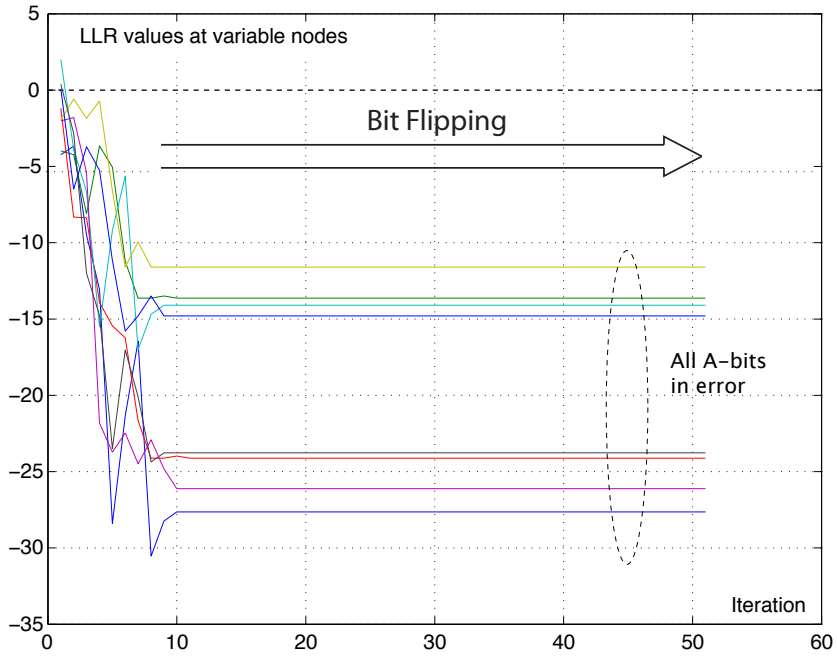


(a) The input frame converges to an $(8, 2)$ absorption set when LLRs are clipped at 10.

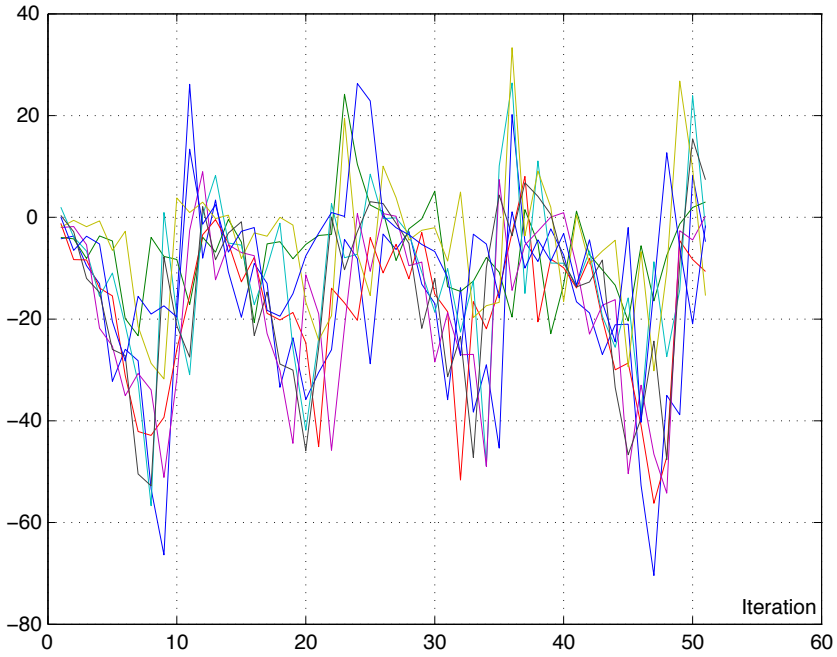


(b) The input frame is corrected when LLRs are clipped at 38.

Figure 4.2: The accumulated LLRs at the first $(8, 2)$ absorption set nodes of the Tanner $[155, 64, 20]$ LDPC code when the input frame is decoded by a min-sum decoder with different clipping thresholds, assuming all-zero codeword is transmitted at 6 dB.

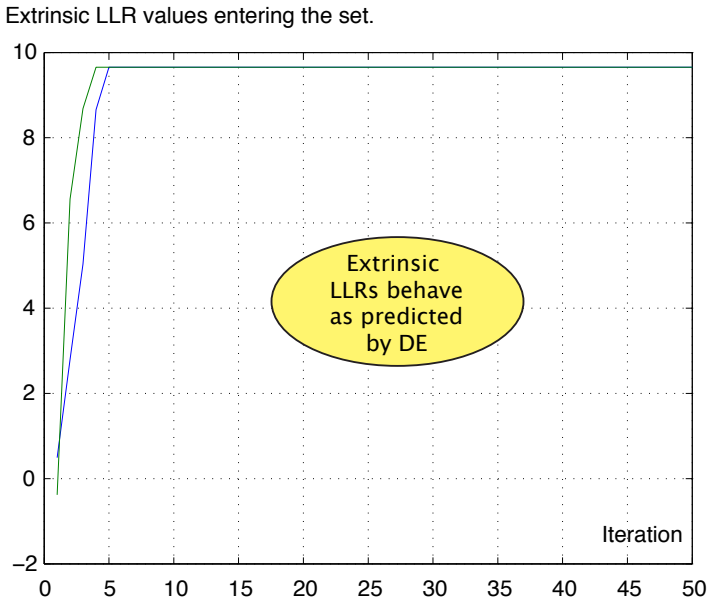


(a) The input frame converges to an $(8, 2)$ absorption set when LLRs are clipped at 10.

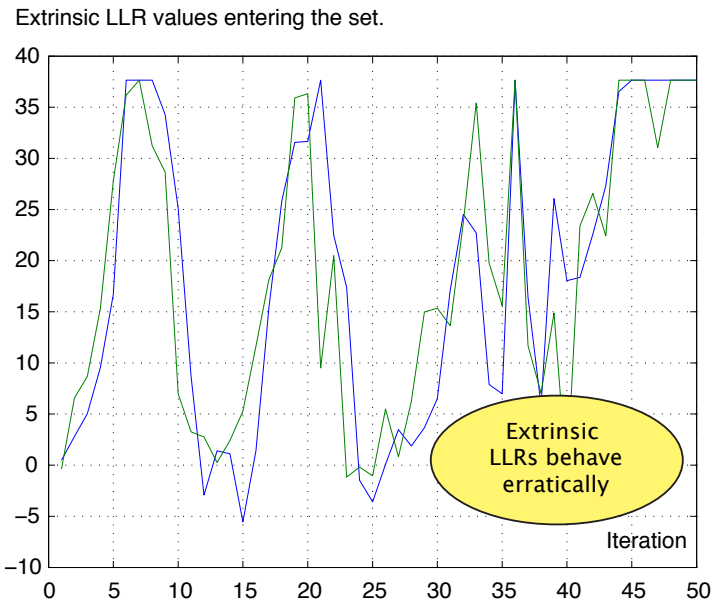


(b) The input frame is not converging when LLRs are clipped at 38 within 50 iterations.

Figure 4.3: The accumulated LLRs at the second $(8, 2)$ absorption set nodes of the Tanner $[155, 64, 20]$ LDPC code when the input framed is decoded by a min-sum decoder with different clipping thresholds, assuming all-zero codeword is transmitted at 6 dB.



(a) The extrinsics reached the LLR clipping limit 10 quickly.



(b) The extrinsics behave erratically when LLRs are clipped at 38.

Figure 4.4: The extrinsic LLRs at the $(8, 2)$ absorption set nodes of the Tanner $[155, 64, 20]$ LDPC code when the input framed is decoded by a min-sum decoder with different clipping thresholds, assuming all-zero codeword is transmitted at 6 dB.

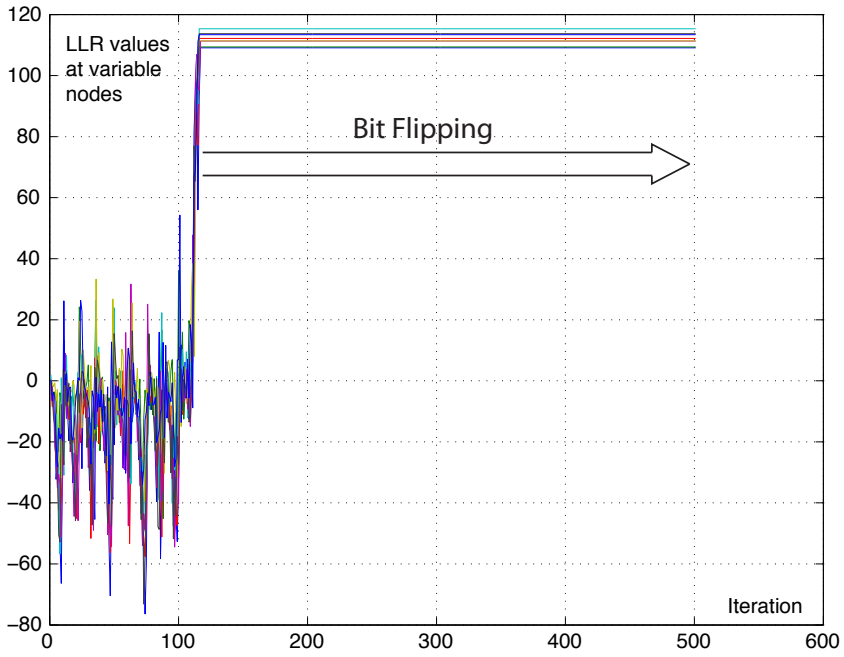


Figure 4.5: The accumulated LLRs at the second $(8, 2)$ absorption set nodes of the Tanner $[155, 64, 20]$ LDPC code when the input framed is decoded by a min-sum decoder with clipping threshold=38 and maximum iteration=500, assuming all-zero codeword is transmitted at 6 dB.

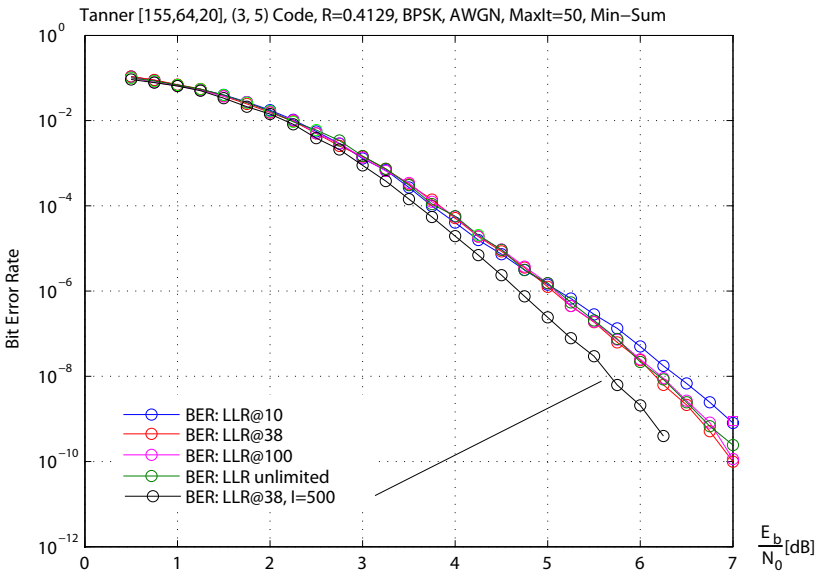


Figure 4.6: The bit error rates of the Tanner $[155, 64, 20], (3, 5)$ LDPC code, assuming min-sum decoding and BPSK modulation on AWGN channel with different decoder configurations.

100. However, the IS results of LLR clipped at 1,000 are not as low as suggested by (3.143), see the black stars in Figure 4.7. This is due to the short length of this code. Shortly after the decoding procedure begins, the LLRs will become so correlated that the extrinsics $\lambda_i^{(\mathrm{ex})}$ start to depart from the behavior predicted by density evolution. However, eventually, the absorption set is still corrected given more iterations are permitted [85]. The qualitative observations made above are therefore valid also for short codes, even though the quantitative prediction power of the error formulas breaks down.

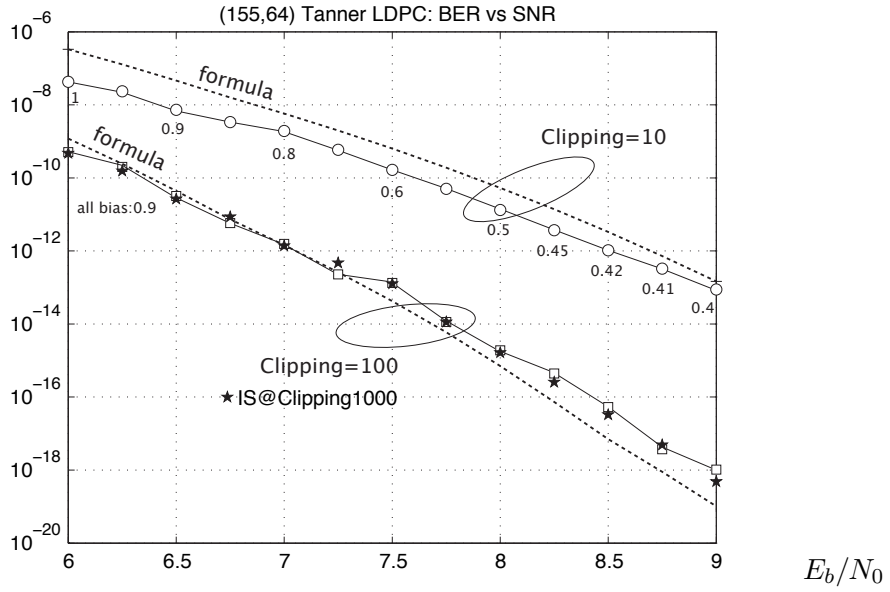
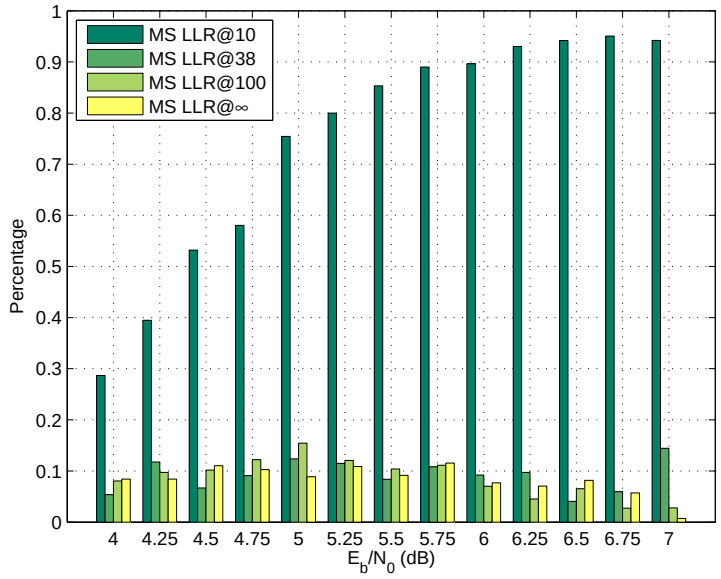


Figure 4.7: Bit error rates of the Tanner $[155, 64, 20]$ LDPC code using both formula (3.143) and importance sampling (IS) with LLR clippings at 10, 100 and 1000, respectively. The maximum iteration number is set to 50.

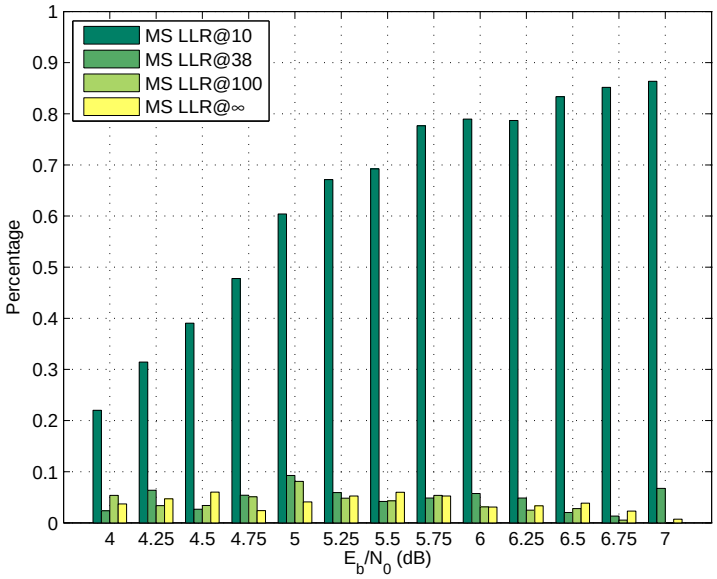
We also analyze the composition of error patterns of the min-sum decoding in Figure 4.8(b), which is consistent with the statistics of the sum-product decoding shown in Figure 4.14(b), as well.

Regarding the IEEE 802.3an LDPC code, all frames causing its $(8, 8)$ absorption sets, that we have captured, can be corrected successfully by raising the LLR clipping threshold to 38 with the iteration number 12 untouched. We believe that this is due to the length of the code and the less trouble-causing absorption set topology. An example is shown in Figures 4.9 and 4.10.

For a bigger code as the IEEE 802.3 LDPC code, we have shown our estimation formula matches the simulations nicely in [63, 64]. If we increase the LLR clipping threshold for this code, the contribution of its dominant absorption set to the error floor also reduces significantly, as shown in Figure 4.11 [87, 86, 60]. The dashed curves plot (3.97) for $I = 10$ and m_λ and $m^{(i)}_{\lambda^{(\mathrm{ex})}}$ are bounded by 10, which is the clipping threshold. The circles are the numerical results of importance sampling (IS) utilizing (4.1) with the same I and LLR clipping threshold. When the threshold is increased to 100, the error rate decreases as (3.97) predicts, also shown in Figure 4.11, and as supported by IS simulation. The error rate further decreases with the even bigger LLR clipping value of 1,000.

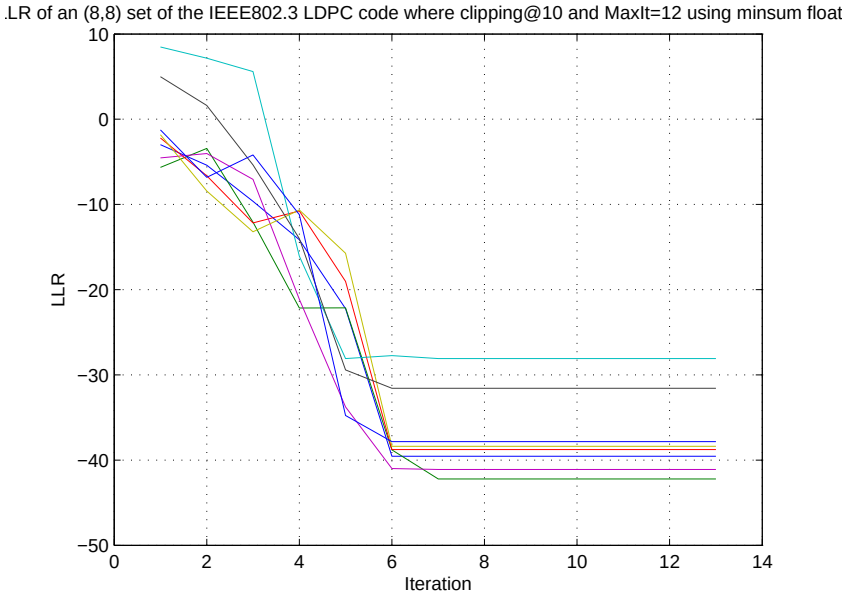


(a) The percent of all absorption sets of all error events.

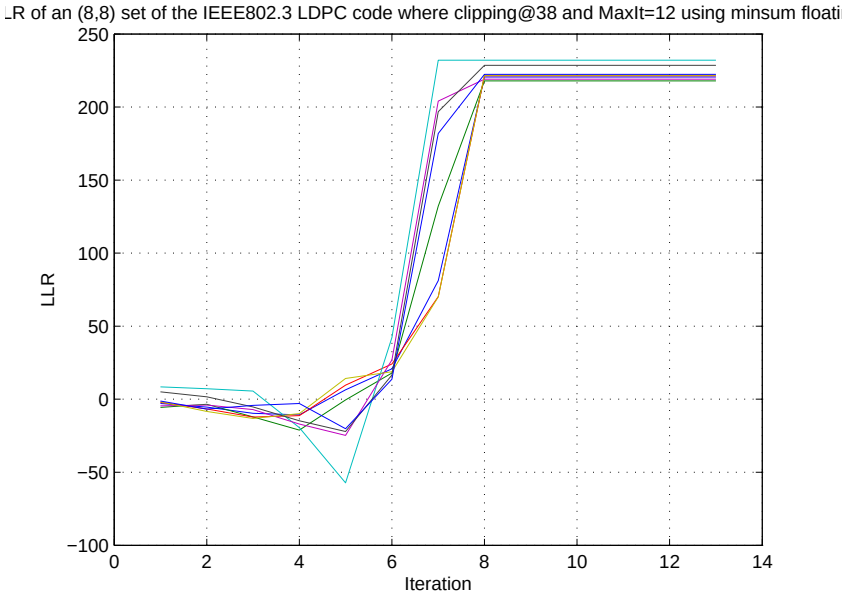


(b) The percent of the $(8, 2)$ absorption sets of all error events.

Figure 4.8: The percentage of absorption sets to the decoding failures of the Tanner $[155, 64, 20]$ LDPC code, assuming min-sum decoding on AWGN channel with maximum iteration=50.



(a) The input frame converges to an $(8, 8)$ absorption set when LLRs are clipped at ± 10.



(b) The input frame is corrected to the all-zero codeword when LLRs are clipped at ± 38.

Figure 4.9: The accumulated LLRs at an $(8, 8)$ absorption set nodes of the IEEE 802.3an $[2048, 1723]$ LDPC code when the input framed is decoded by a min-sum decoder with different clipping thresholds, assuming the all-zero codeword is transmitted at 5 dB.

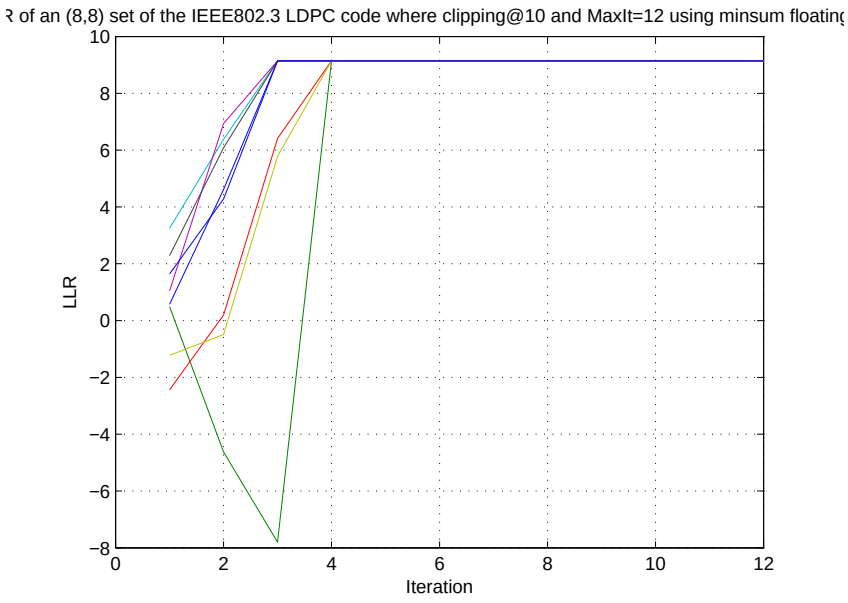


Figure 4.10: The dynamics of the eight extrinsics of the $(8, 8)$ absorption set nodes of the IEEE 802.3an $[2048, 1723]$ LDPC code when the input framed is decoded by a min-sum decoder when the LLR clipping limit is 10, assuming all-zero codeword is transmitted at 5 dB.

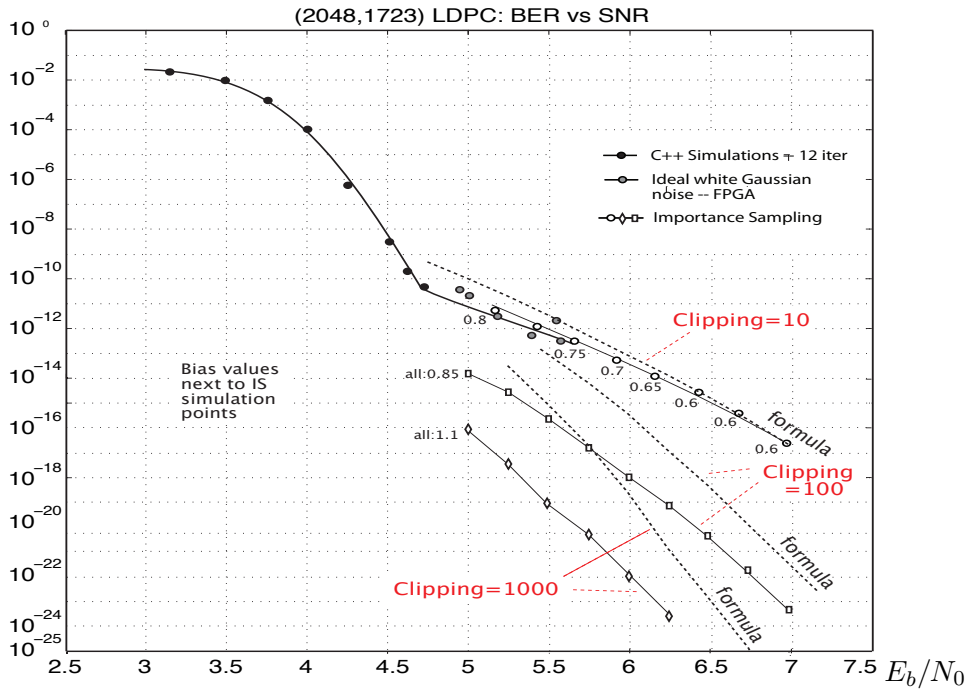


Figure 4.11: Bit error rates of the IEEE 802.3an LDPC code using both formula (3.97) and importance sampling (IS) with LLR clippings at 10, 100 and 1000, respectively. The iteration number is set to 10.

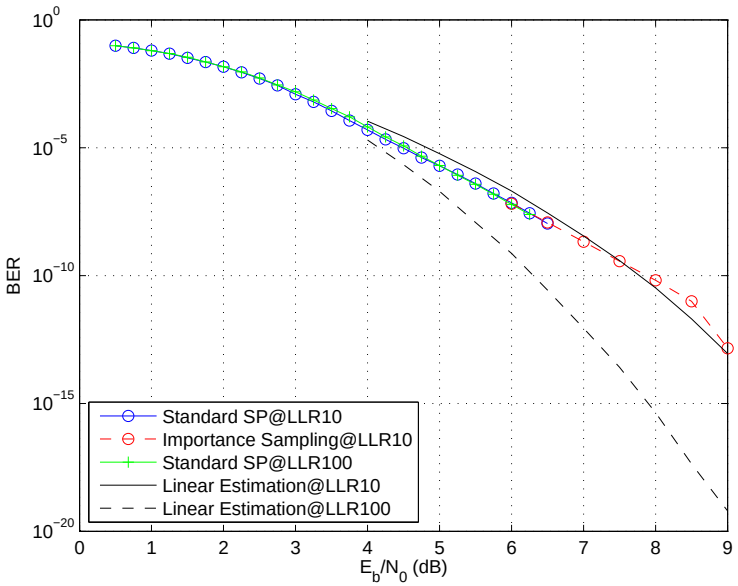


Figure 4.12: The numerical and analytical results of the performance of Tanner $[155, 64, 20], (3, 5)$ LDPC code, assuming BPSK modulation on an AWGN channel and maximum iteration=50.

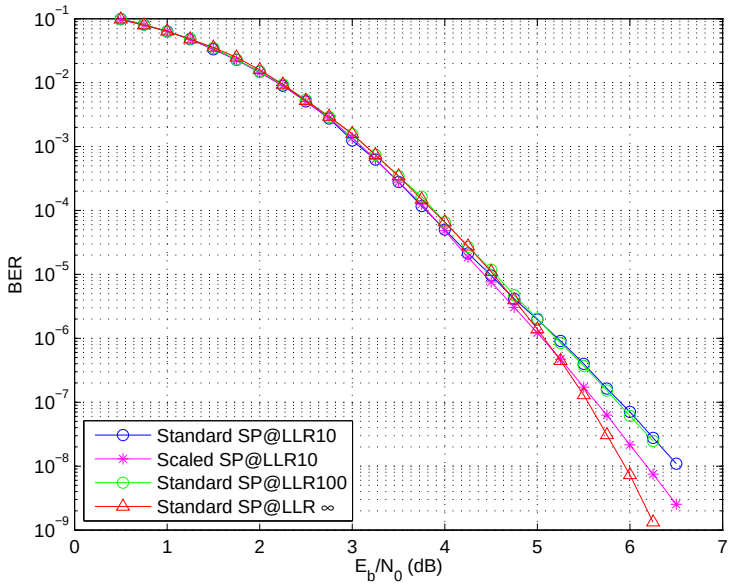
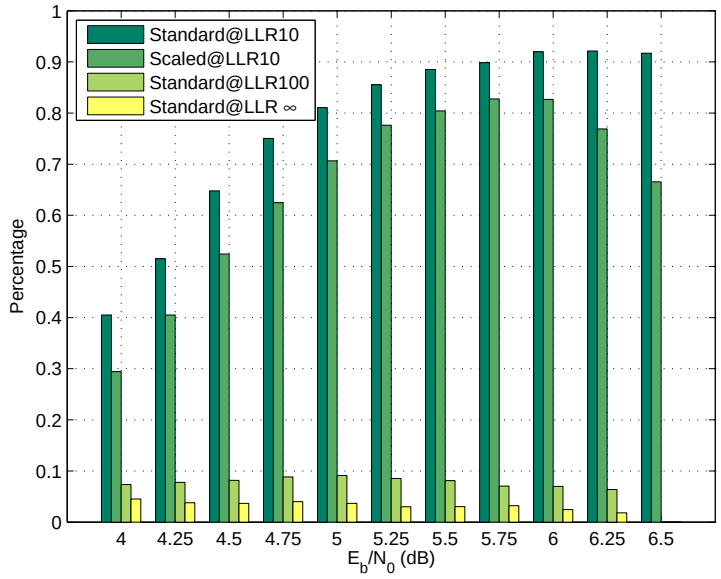
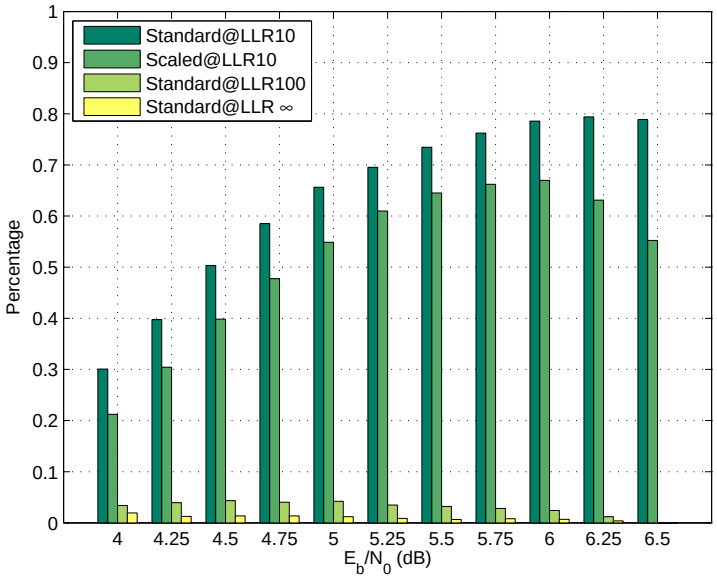


Figure 4.13: The numerical results of the performance of Tanner $[155, 64, 20], (3, 5)$ LDPC code, assuming BPSK modulation on an AWGN channel and maximum iteration=50.



(a) The percent of all absorption sets over all error events.



(b) The percent of the $(8, 2)$ absorption sets over all error events.

Figure 4.14: The percentage of absorption sets to the decoding failures of the Tanner $[155, 64, 20]$ LDPC code, assuming sum-product decoding on AWGN channel with maximum iteration=50.

4.3 Iterations and Complexity

Larger LLR clipping thresholds imply more complexity in practice, since wider bit widths are needed to represent the messages. Figure 4.11 shows how much gain can be achieved by increasing the clipping values, in terms of lowering the error rate of the IEEE 802.3an LDPC code. We note that the theoretical results correlate well with the IS simulations.

Regarding the Tanner $[155, 64, 20]$ code, simply increasing the clipping threshold to over 100 alone has no additional benefit. as shown in Figure 4.7. The correlations among the LLRs of this small code compromise the growth of the extrinsic information entering the absorption sets.

Motivated by these observations, in this section we explore the impact of message quantization and number of iterations on a code's error floor. Arguably, the product of bit-width and number of iterations is an accurate measure of the computational complexity of a decoder since this number is directly related to the switching activity of a digital decoder (see [60]), and hence also the energy expended by the decoder.

In order to verify our theoretical results, we resort again to importance sampling (IS). IS is an ideal tool to explore variations of a decoder, such as finite precision operation, where the impact of design changes need to be explored for ultra-low error rates. One way of increasing the number of error events, or positive counts, is to distort the noise distribution to cause more errors. This is typically done by shifting the mean of the noise towards a convenient boundary of the signal space region where the decoder fails to produce the correct output (mean-shift importance sampling). The key questions are where to shift the transmitted signal and by how much.

A priori knowledge of the dominant error mechanisms is extremely important for proper use of IS, since otherwise a mean shift can actually mask an error by moving the signal further away from the dominant error event. Furthermore, the correct amount of the shift is also important. If the shift value is too small, not enough simulation speed up is achieved; if the shift value is too large, a phenomenon called *over-biasing* causes the IS error estimate to dramatically underestimate the true error contribution by the dominant event. This happens when the biased simulated samples occur too far away from the decision boundary, but inside the error region. These samples are weighed with a index that is too small. Not enough samples are generated close to the decision boundary from where the majority of the actual error contribution originates.

Since absorption sets are examined as the primary causes of the significant events in the error floor region, we add such a mean shift, or *bias*, towards the bits that make up the absorption set. Having identified and enumerated the dominant absorption sets of an LDPC code, we technically need to perform this shift for each absorption set separately, but symmetries can often by exploited in reducing this task.

Richardson [58] used a version of IS where this mean shift assumes a continuous distribution over which the simulations are averaged. This method appears to alleviate the over-biasing, but also obscures the phenomenon. In our approach we carefully choose the mean shift by assuring that small changes do not alter the computed error rate. Mean shift values used for some of our simulations are marked in the figures. We finally wish to note that, unlike sometimes implied in the literature, IS can only properly reproduce an error floor if the causes of the error floor are sufficiently well understood to apply proper biasing. Hence our two-step procedure in Sections 3.4 and 3.5 for the IEEE802.3an LDPC code, and Section 3.6 for the length-155 Tanner code.

Taking into account the bias value and the dominant absorption sets selected in the operation, the estimation formula (1.21) along with the weighting factor (1.23) must be adjusted accordingly, leading to a weight term $w(y) \ll 1$. The combined effect of measuring more significant events and ascribing them lower weight will produce the same error rate measure in (1.21) if the shifting is done correctly. Figures 4.11 and 4.7 show the results of importance sampling for the IEEE 802.3an and the Tanner $[155, 64, 20]$ LDPC codes compared to our formulas for floating point calculations.

In a hardware implementation, however, finite precision arithmetic is used. The more digits used in the decoder, the more power and computational effort is required, but better performance will results. It is therefore vitally important for implementations to understand this cost-benefit tradeoff. For the IEEE 802.3an code, IS simulations with both floating and fixed point calculations at different LLR clipping values are shown in Figure 4.15. Not surprisingly, for smaller clipping values, smaller numbers of bits are required to adequately represent the messages. While 6 bits of quantization are required for a clipping threshold of 10, 10 bits are needed for a clipping threshold of 100, and 14 bits of quantization are required to exploit the full benefit of a clipping threshold of 1000.

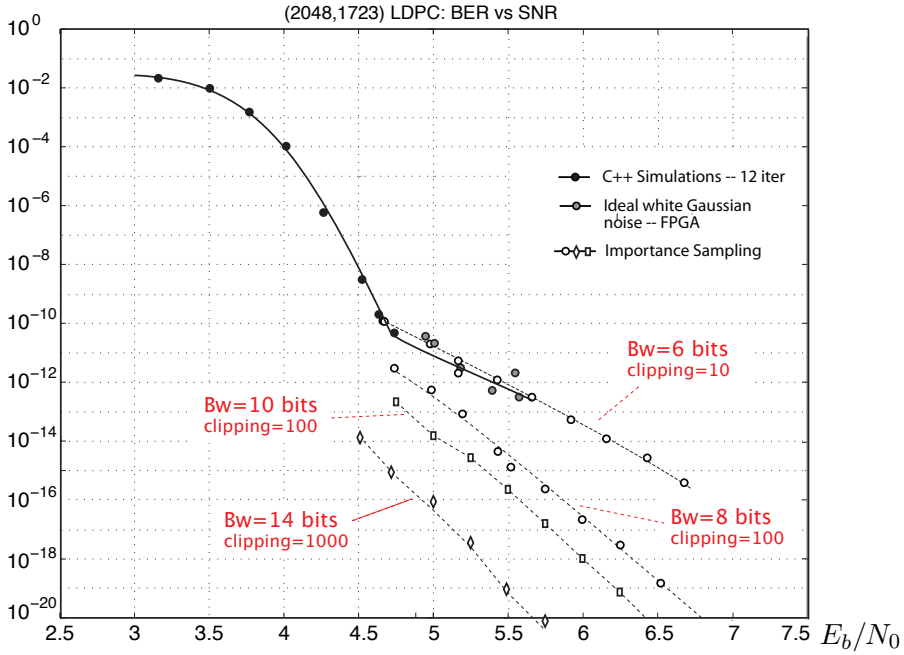


Figure 4.15: Error rates of the IEEE 802.3an LDPC code via importance sampling (IS) with finite precision, where the LLRs are clipped at 10, 100 and 1000, respectively. The maximum iteration number is preset at 10.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this dissertation, we started by reviewing the basics of the low-density parity-check codes, a class of linear block codes. Their sparse parity-check matrices enable efficient software and hardware decoder implementations and therefore make them popular in applications. LDPCs have been the focus of extensive research activities over the last two decades because of their extraordinary performance, their ability to closely approach channel capacity, the excellent distances among codewords, and ease of implementation. Despite all of their good qualities, there are still challenges posed by LDPCs, especially with growing demands on speed and ultra-low error rates.

The efficient iterative message-passing decoding algorithms for each of the popular communications channels, to be exact, the binary erasure, the binary symmetric and the additive white Gaussian noise channels, were reviewed at the beginning of this thesis.

The error rate of LDPC codes decreases quickly as the signal-to-noise ratio increases and is bounded from below by the maximum-likelihood decoding error probability, related to the minimum-weight non-zero codewords. However, when utilizing the sub-optimal, but more practical iterative decoding methods, such as message passing or linear programming, a marked increase of this lower bound tends to appear which has the shape of an error floor. This is caused by inherent structural weaknesses in the code's interconnect network, which the iterative decoder cannot overcome. Once trapped in certain error patterns the decoder cannot recover. Such decoding failures were studied in [81, 28, 26], and initially dubbed trapping sets in [58].

Trapping sets depend not only on the code but also on the channel and the decoding algorithm. Trapping sets of the binary erasure channel are known as stopping sets [16], whereas the dominant trapping sets of LDPC codes on the Gaussian channel are called absorption sets [88], which is also the failure mechanism on the binary symmetric channel. The Gaussian channel differs from the binary symmetric channel and the binary erasure channel in that the error behavior of the LDPC decoder is more complicated.

An absorption set is a set of variable nodes such that the majority of the neighboring check nodes of each variable node in the set are connected to the set an even number of times. Considering Gallager's bit flipping decoding [31], for instance, a majority of messages sent to an absorption set will retain the erroneous sign. Therefore, the decoder will not be able to correct the bits in the absorption set and it will lock up in an erroneous state. It is believed that the "smaller" the absorption set, the more severe its effect on the error

floor phenomenon [58, 48, 63, 64]. This is similar to the observation that lower weight codewords have more impact on the error rate than higher weight codewords. Therefore, minimal absorption sets play a critical role in the error floor.

Due to the growing popularity of LDPC codes, effort have been expended to understand the dynamics of absorption sets [24, 79]. An immediate application of such knowledge is to predict the error floor. This is of great interest especially when the error floor is too low to be simulated even by hardware implementations. Furthermore, the error floor can be lowered by modifying the code design or decoding algorithms targeting the absorption sets [72, 47, 78, 35, 44, 89, 63, 64, 10, 33, 46, 40, 41, 42].

After identifying the dominant absorption sets, we developed a linear model to analyze the error floor error probability. We first identified and enumerated the minimal absorption sets, which dominate the decoding performance in the error floor region. Then, by characterizing the dynamics of the log-likelihood ratios in the absorption set variables using the set structure, and making use of density evolution, we derived a closed-form formula that closely approximates the probability that an absorption set will fall in error.

Both the IEEE 802.3an $[2048, 1723]$ regular $(6, 32)$ LDPC code, which is used for 10 Gbit/s Ethernet over twisted-pair copper cables, and the Tanner $[155, 64, 20]$ regular $(3, 5)$ LDPC code were considered to illustrate our algebraic approach. The estimation from our formula matches the numerical results accurately.

We also developed a systematic method to enumerate the absorption sets of those codes. As a byproduct of the enumeration process, the minimum distance of the IEEE 802.3an code has been tightened to either 12 or 14. However, fully classifying the absorption sets is a largely unsolved, and perhaps unsolvable, problem. Our analysis of the existence of larger sets heavily depends on the results of smaller sets. In addition, classification of absorption set using an unordered array to denote the point degrees in the subgraph induced by an absorption set is of crucial importance, especially when the variable node degree is greater than five. Last but not least, extreme caution must be practiced when exhaustively listing all possible topological structures and determining whether topologies obtained from different approaches are identical or not.

After successfully approximating the error floors, guided by the error probability formula, a simple but effective method to improve the code performance in the error floor region without altering the code construction is proposed, by introducing a boosting factor to the log-likelihood ratios returned from unsatisfied check nodes during the early decoding stages. The effect of this modification is that the likelihood of getting trapped in the absorption set is generally reduced. The effect materializes in the first few iterations since the modification cannot "correct" an absorption set once the error has stabilized. The intention is to prevent the decoder from getting trapped into the absorption sets at the early stages of decoding. Such a modification can be easily implemented and extended to schedules other than flooding [67, 54, 68, 90, 59, 73, 80, 3]. We point out that this approach is different from the ones proposed in [89] and [46], which target "fixing" the absorption sets by post-processing or backtracking the trapping sets, usually as a final decoding stage. Modifying the code structure to eliminate the dominant absorption sets has also been considered by some authors [44].

It is shown that the effect of the dominant trapping sets is reduced, but only a limited number of absorption set error patterns can be eliminated. In other words, the dominance of the absorption sets is reduced but remains.

So we looked further and found out that the impact from absorption sets can be reduced

significantly such that they are not dominating the error rate any more, by extending the computational range of the LLRs at the decoder. Such range is realized by a decoder setting, called the LLR clipping threshold, which serves as a hidden variable in our formula. By allowing the LLRs to grow bigger, the messages from the unsatisfied check nodes of the absorption sets, which we were trying to boost up, will become *strong* enough to eventually overcome the incorrect LLRs trapped within the set's network. In that case, absorption sets will no longer exist as primary error patterns of the iterative decoders. Therefore, the error rates will, again, be dominated by the maximum allowed iterations at the decoder and the low-weight codewords.

In the field of optical communication, 100G is regarded by the industry as the most important transmission rate of the next generation. As the 100G coherent transmission technology has become the mainstream 100G transmission technology, whereas the non-coherent transmission technology has been widely used in the 10G and 40G transmission systems, it makes LDPC-based soft-decision forward error correction (FEC) continuously a good candidate for 100G long-haul transmission [6, 19, 20, 18, 49]. LDPC-based soft-decision FEC will provide a promising solution to further improve the performance of 100G and even 1T transmissions. Therefore, our error floor analysis will become very beneficial in the low error rate region of the high rate transmission schemes.

5.2 Future Work

The linear analysis that we provided is general. However the enumeration of the small absorption sets is code-specific and costs a lot of effort, even though it is already significantly better than any brute-force searching algorithm. It is possible to fill in more gaps in Table 3.2. To do that, future work could find a better way to enumerate them. By better, we mean faster or more efficient, and more generalized.

To further strengthen the generality of our linear analysis, we would like to apply our analysis to more LDPC codes. Tanner codes and MacKay codes are some good options. Then the impact of the LLR clipping threshold and, perhaps, maximum iteration number on the error rate could be tested further.

When moving on to other codes, the $\mathbf{V}$ and $\mathbf{C}$ matrices of their dominant absorption sets may not be as regular as (3.8) and (3.5) of the IEEE 802.3an LDPC code, according to Definition 3.2 of an absorption set. For instance, within the absorption set subgraph, the satisfied check nodes may be connected with four or six variable nodes from the set. This is highly dependent on the code structure, or the parity-check matrix $\mathbf{H}$. When the check nodes send their message back to the neighbor variable nodes in the decoding algorithm, they are not simply performing exchanges any more. But we still need to show that the maximum eigenvalue of $\mathbf{VC}$ is larger than 1, so the LLRs will keep growing at each decoding iteration. It would be convenient to figure out a general method to approximate the maximum eigenvalue, $\mu_{\max}$, which seems to be closely related to the variable node degree, d_v, and the set dimensions, a and b, without calculating the eigenvalues numerically for each absorption set structure. However, by including that approximation into our estimation formula, we expect a degraded result in estimating the error floors, due to the precision lost in approximations.

Furthermore, Richardson attributed the failure mechanism to trapping sets or "near codewords", which are sets with a relatively small number of variable nodes such that the induced sub-graph has only a small number of odd degree check nodes [58]. Apparently

absorption sets are (dominant) trapping sets. However is it true that all trapping sets are absorption sets? The answer seems to be "yes", since we have successfully predicted the error floor by making use of minimal absorption sets and no error pattern other than absorption sets has been seen or, if not impossible, could be constructed. However, to make our analysis more complete, we need to come up with more solid argument for the conjecture.

Another interesting perspective of the error floors is to look at the connection between the iterative message-passing and the optimization-searching linear programming (LP) decoding algorithms.

The LP decoding algorithm has attracted much attention recently since its performance approaches that of optimal ML decoding. Although the message-passing decoding algorithm is the exact decoding algorithm only for cycle-free LDPC codes, it performs very well in practice for real LDPC. However, it does not always converge. In addition, it does not have the ML certificate. But we must remark that in practice, for large block length LDPC codes, when the message-passing decoder outputs a codeword, it is extremely rare for it not to be the ML codeword [25]. From the experiments results [25], it seems that in very low noise conditions LP decoding even outperforms message-passing decoding. So the position of LP decoding lies between ML and MP.

In the binary erasure channel, it is known that message-passing fails if and only if a stopping set exists [7]. While it is shown in [25] that the performance of the LP decoder is equivalent to belief propagation on the BEC, we do not have the same conclusion for the symmetric channel. The error patterns of LP decoding on BSC are named *instantons*, as defined in Definition 3.9. The instantons of the Tanner $[155, 64, 20]$ regular $(3, 5)$ LDPC code correspond to its $(5, 3)$ absorption sets. The correspondence between these two decoding failures of different decoding algorithms remains inconclusive as it is possible to construct an instanton whose topology does not resemble that of an absorption set.

Active research topics in this field include tightening up the relaxation, strengthening the lower bound of d_{frac}, adapting the general LP *Simplex* algorithm to decoding-oriented algorithm, comparing the computational complexity of LP decoding and message-passing decoding, etc. But what we would like to do is to concentrate on linking LP decoding to our analysis, i.e., studying the error patterns of LP decoding, specifically on BSC and AWGN channels. As intantons have been identified as the failure mechanism of LP decoding on BSC channels and the minimum weight instantons correspond to "dominant" absorption sets, we would like to understand the connection between absorption sets and instantons or pseudocodewords. It is also unclear that why LP decoding failed on the absorption sets of the $[155, 64, 20], (3, 5)$ Tanner code. Absorption sets are cycles or unions of cycles, on which the message-passing decoding does not work very well. It looks like that those cycles are causing trouble in LP decoding as well. There seems to be some connection between cycles and pseudocodewords. Thus the topological structure of instantons has to be examined.

We know that pseudocodewords are vertices of the LP polytope Q, and the Q comes from the intersection of all local polytope Q_j's (2.75). The relaxations as part of the constraints might be able to help explain the connection between cycles and instantons. We will also try to get some insights from the intanton search algorithm shown in Chapter 3. Noting that an LP decoder is involved in the search algorithm, the way of reducing the weight by calculating medians could also be applied by the algorithm. This method might also provide us some insight.

Bibliography

[1] S. Abu-Surra, D. DeClercq, D. Divsalar, and W. E. Ryan. Trapping set enumerators for specific LDPC codes. In *Information Theory and Applications Workshop (ITA)*, pages 1–5, UCSD, San Diego, CA, USA, February 2010.

[2] S. Abu-Surra, D. Divsalar, and W. E. Ryan. Enumerators for protograph-based ensembles of LDPC and generalized LDPC codes. *IEEE Trans. Inf. Theory*, 57(2):858–886, February 2011.

[3] K. Baek, H. Lee, C. Choi, S. Kim, and G. E. Sobelman. A high-throughput LDPC decoder architecture for high-rate WPAN systems. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 1311–1314, May 2011.

[4] J. R. Barry, E. A. Lee, and D. G. Messerschmitt. *Digital Communication*. Springer, third edition, 2003.

[5] B. K. Butler and P. H. Siegel. Error floor approximation for LDPC codes in the AWGN channel. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 204 –211, UIUC, Allerton Retreat Center, Monticello, IL, USA, September 2011.

[6] F. Chang, K. Onohara, and T. Mizuochi. Forward error correction for 100 G transport networks. *Communications Magazine, IEEE*, 48(3):S48–S55, March 2010.

[7] Di. Changyan, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke. Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Trans. Inf. Theory*, 48(6):1570–1579, June 2002.

[8] Z. Chen. *Design of Low-Density Parity-Check Convolutional Codes for Efficient VLSI Implementation*. PhD thesis, University of Alberta, January 2009.

[9] M. Chertkov and M. Stepanov. An efficient pseudocodeword search algorithm for linear programming decoding of LDPC codes. *IEEE Trans. Inf. Theory*, 54(4):1514–1520, April 2008.

[10] S. K. Chilappagari, M. Chertkov, M. Stepanov, and B. Vasić. Instanton-based techniques for analysis and reduction of error floors of LDPC codes. *Selected Areas in Communications, IEEE Journal on*, 27(6):855–865, August 2009.

[11] S. K. Chilappagari, M. Chertkov, and B. Vasić. An efficient instanton search algorithm for LP decoding of LDPC codes over the bsc. *IEEE Trans. Inf. Theory*, 57(7):4417 –4426, July 2011.

[12] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasić. Error floors of LDPC codes on the binary symmetric channel. In *Communications, 2006. ICC '06. IEEE International Conference on*, volume 3, pages 1089–1094, June 2006.

[13] S. K. Chilappagari, B. Vasić, M. Stepanov, and M. Chertkov. Analysis of error floor of LDPC codes under LP decoding over the BSC. In *International Symposium on Information Theory, 2009. ISIT 2009. IEEE*, July 2009.

[14] S.-Y. Chung, G. D. Jr. Forney, T. J. Richardson, and R. L. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the shannon limit. *IEEE Commun. Lett.*, 5(2):58–60, February 2001.

[15] S.-Y. Chung, T. J. Richardson, and R. L. Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation. *IEEE Trans. Inf. Theory*, 47(2):657–670, February 2001.

[16] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke. Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Trans. Inf. Theory*, 48(6):1570–1579, June 2002.

[17] D. Divsalar, H. Jin, and R. McEliece. Coding theorems for "turbo-like" codes. *Proc. 36th Annual Allerton Conference on Communications, Control, and Computing*, pages 201–210, September 1998.

[18] I. B. Djordjevic, M. Cvijetic, L. Xu, and T. Wang. Proposal for beyond 100-gb/s optical transmission based on bit-interleaved LDPC-coded modulation. *Photonics Technology Letters, IEEE*, 19(12):874–876, June 15, 2007.

[19] I. B. Djordjevic, O. Milenkovic, and B. Vasic. Generalized low-density parity-check codes for optical communication systems. *Lightwave Technology, Journal of*, 23(5):1939–1946, May 2005.

[20] I. B. Djordjevic, S. Sankaranarayanan, S. K. Chilappagari, and B. Vasic. Low-density parity-check codes for 40-gb/s optical transmission systems. *Selected Topics in Quantum Electronics, IEEE Journal of*, 12(4):555–562, July/August 2006.

[21] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin. A class of low-density parity-check codes constructed based on Reed-Solomon codes with two information symbols. *IEEE Commun. Lett.*, 7(7):317–319, July 2003.

[22] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright. Predicting error floors of structured LDPC codes: deterministic bounds and estimates. *Selected Areas in Communications, IEEE Journal on*, 27(6):908–917, August 2009.

[23] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic. Analysis of absorbing sets for array-based LDPC codes. *Communications, 2007. ICC '07. IEEE International Conference on*, pages 6261–6268, June 2007.

[24] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic. Analysis of absorbing sets and fully absorbing sets for array-based LDPC codes. *IEEE Trans. Inf. Theory*, 56(1):181–201, January 2010.

[25] J. Feldman. *Decoding Error-Correcting Codes via Linear Programming*. PhD thesis, Massachusetts Institute of Technology, September 2003.

[26] G. D. Jr. Forney, R. Kötter, F. R. Kschischang, and A. Reznik. On the effective weights of pseudocodewords for codes defined on graphs with cycles. In *Codes, Systems, and Graphical Models*, pages 101–112, Springer, 2001.

[27] M. P. C. Fossorier. Quasicyclic low-density parity-check codes from circulant permutation matrices. *IEEE Trans. Inf. Theory*, 50(8):1788 – 1793, August 2004.

[28] B. J. Frey, R. Kötter, and A. Vardy. Signal-space characterization of iterative decoding. *IEEE Trans. Inf. Theory*, 47(2):766–781, February 2001.

[29] G. Frobenius. *Üeber Matrizen aus nicht negativen Elementen*. Sitzungsber. Königl. Preuss. Akad., Wiss. (Berlin), Germany, 1912.

[30] R. G. Gallager. Low-density parity-check codes. *IEEE Trans. Inf. Theory*, 8(1):21–28, January 1962.

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

[32] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[33] L. Gong, Y Xu, B. Liu, L. Gui, B. Rong, Y. Wu, and W. Zhang. A modified belief propagation algorithm based on attenuation of the extrinsic LLR. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–5, September 2010.

[34] M. Grtschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, June 1981.

[35] Z. He, S. Roy, and P. Fortier. Lowering error floor of LDPC codes using a joint row-column decoding algorithm. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 920–925, June 2007.

[36] R. Holzlohner, A. Mahadevan, C. R. Menyuk, J. M. Morris, and J. Zweck. Evaluation of the very low ber of fec codes using dual adaptive importance sampling. *IEEE Commun. Lett.*, 9(2):163–165, February 2005.

[37] S. L. Howard, V. C. Gaudet, and C. Schlegel. Soft-bit decoding of regular low-density parity-check codes. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 52(10):646 – 650, October 2005.

[38] S. L. Howard, C. Schlegel, and V. C. Gaudet. Degree-matched check node decoding for regular and irregular LDPCs. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 53(10):1054 –1058, October 2006.

[39] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia. Efficient implementation of the sum-product algorithm for decoding LDPC codes. *Proc. IEEE Globecom, San Antonio, TX*, pages 1036–1036E, November 2001.

[40] Q. Huang, Q. Diao, S. Lin, and K. Abdel-Ghaffar. Cyclic and quasi-cyclic LDPC codes: New developments. In *Information Theory and Applications Workshop (ITA), 2011*, pages 1–10, February 2011.

[41] Q. Huang, Q. Diao, S. Lin, and K. Abdel-Ghaffar. Trapping sets of structured LDPC codes. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1086–1090, August 2011.

[42] Q. Huang, Q. Diao, S. Lin, and K. Abdel-Ghaffar. Cyclic and quasi-cyclic LDPC codes on constrained parity-check matrices and their trapping sets. *IEEE Trans. Inf. Theory*, 58(5):2648–2671, May 2012.

[43] H. Imai. *Essentials of Error-Control Coding Techniques*. Academic Press, Inc., Orlando, FL, USA, 1990.

[44] M. Ivković, S. K. Chilappagari, and B. Vasić. Eliminating trapping sets in low-density parity-check codes by using tanner graph covers. *IEEE Trans. Inf. Theory*, 54(8):3763–3768, August 2008.

[45] A. Jiménez-Felström and K. S. Zigangirov. Time-varying periodic convolutional codes with low-density parity-check matrix. *IEEE Trans. Inf. Theory*, 45(6):2181 –2191, September 1999.

[46] J. Kang, Q. Huang, S. Lin, and K. Abdel-Ghaffar. An iterative decoding algorithm with backtracking to lower the error-floors of LDPC codes. *IEEE Trans. Inf. Theory*, 59(1):64–73, January 2011.

[47] S. Laendner, T. Hehn, O. Milenkovic, and J. B. Huber. Cth02-4: When does one redundant parity-check equation matter? In *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–6, December 2006.

[48] S. Ländner and O. Milenkovic. Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes. In *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, volume 1, pages 630–635, 2005.

[49] Q. Li, L. Yin, and J. Lu. Application of LDPC codes for deep space communication under solar scintillation condition. In *ICOSSSE'11: Proceedings of the 10th WSEAS international conference on System science and simulation in engineering*, pages 191–195, Stevens Point, Wisconsin, USA, October 3–5, 2011. World Scientific and Engineering Academy and Society (WSEAS).

[50] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. 29th Annual ACM Symp. Theory of Computing*, pages 150–159, 1998.

[51] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inf. Theory*, 45(2):399–431, March 1999.

[52] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[53] D. J. C. MacKay and M. C. Davey. Evaluation of Gallager codes for short block length and high rate applications. In B. Marcus and J. Rosenthal, editors, *Codes, Systems and Graphical Models*, volume 123 of *IMA Volumes in Mathematics and its Applications*, pages 113–130. Springer, New York, 2000.

[54] M. M. Mansour and N. R. Shanbhag. A 640-mb/s 2048-bit programmable LDPC decoder chip. *Solid-State Circuits, IEEE Journal of*, 41(3):684–698, March 2006.

[55] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[56] O. Perron. Zur theorie der matrices. *Mathematische Annalen*, 64(2):248–263, June 1907.

[57] E. Psota and L. C. Pérez. The manifestation of stopping sets and absorbing sets as deviations on the computation trees of LDPC codes. *Journal of Electrical and Computer Engineering*, 2010, 2010. 17 pages.

[58] T. J. Richardson. Error floors of LDPC codes. *41st Annual Allerton Conf. on Communications, Control and Computing, Monticello, Illinois, USA*, pages 1426–1435, October 2003.

[59] C. Roth, A. Cevrero, C. Studer, Y. Leblebici, and A. Burg. Area, throughput, and energy-efficiency trade-offs in the VLSI implementation of LDPC decoders. In *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pages 1772–1775, Rio de Janeiro, Brazil, May 2011.

[60] C. Schlegel. From mathematics to physics: The task of building efficient and effective iterative error control decoders. In *the 7th International Symposium on Turbo Codes & Iterative Information Processing*, Gothenburg, Sweden, August 27–31, 2012. Invited plenary talk.

[61] C. Schlegel and L. C. Pérez. *Trellis and Turbo Coding*. John Wiley & Sons, 2003.

[62] C. Schlegel and S. Zhang. Dynamics and performance of LDPC code trapping sets in the error floor region. In *Information Theory and Applications Workshop (ITA)*, UCSD, San Diego, CA, USA, February 2009.

[63] C. Schlegel and S. Zhang. On the dynamics of the error floor behavior in regular LDPC codes. In *Information Theory Workshop, 2009. ITW 2009. IEEE*, pages 243 –247, October 2009.

[64] C. Schlegel and S. Zhang. On the dynamics of the error floor behavior in (regular) LDPC codes. *IEEE Trans. Inf. Theory*, 56(7):3248–3264, July 2010.

[65] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[66] C. E. Shannon and W. Weaver. *A Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, USA, 1963.

[67] E. Sharon, S. Litsyn, and J. Goldberger. An efficient message-passing schedule for LDPC decoding. In *Electrical and Electronics Engineers in Israel, 2004. Proceedings. 2004 23rd IEEE Convention of*, pages 223–226, September 2004.

[68] E. Sharon, S. Litsyn, and J. Goldberger. Efficient serial message-passing schedules for LDPC decoding. *IEEE Trans. Inf. Theory*, 53(11):4076–4091, November 2007.

[69] M. Sipser and D. A. Spielman. Expander codes. *IEEE Trans. Inf. Theory*, 42(6):1710–1722, November 1996.

[70] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Jounal of the ACM*, 51(3):385–463, 2004.

[71] A. Sridharan, D. Truhachev, M. Lentmaier, D. J. Costello, and K. Zigangirov. Distance bounds for an ensemble of LDPC convolutional codes. *IEEE Trans. Inf. Theory*, 53(12):4537–4555, December 2007.

[72] J. Sun, O. Y. Takeshita, and M. P. Fitz. Analysis of trapping sets for LDPC codes using a linear system model. In *42nd Annual Allerton Conference on Communications, Control and Computing*, October 2004.

[73] Y. Sun, G. Wang, and J. R. Cavallaro. Multi-layer parallel decoding algorithm and vlsi architecture for quasi-cyclic LDPC codes. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 1776–1779, May 2011.

[74] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inf. Theory*, 27(5):533–547, September 1981.

[75] R. M. Tanner, D. Sridhara, and T. E. Fuja. A class of group-structured LDPC codes. In *Proc. Int. Symp Communications Theory and Applications*, Ambleside, U.K., July 2001.

[76] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Jr. Costello. LDPC block and convolutional codes based on circulant matrices. *IEEE Trans. Inf. Theory*, 50(12):2966 – 2984, December 2004.

[77] D. Truhachev, K. Zigangirov, and D. J. Costello. Distance bounds for periodically time-varying and tail-biting LDPC convolutional codes. *IEEE Trans. Inf. Theory*, 56(9):4301–4308, September 2010.

[78] N. Varnica and M. Fossorier. Improvements in belief-propagation decoding based on averaging information from decoder and correction of clusters of nodes. *Communications Letters, IEEE*, 10(12):846–848, December 2006.

[79] P. O. Vontobel. A list of references on pseudocodewords, trapping sets, stopping sets, absorption sets.

[80] M. Weiner, B. Nikolic, and Z. Zhang. LDPC decoder architecture for high-data rate personal-area networks. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 1784–1787, May 2011.

[81] N. Wiberg. *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, Sweden, 1996.

[82] B. Xia and W. E. Ryan. On importance sampling for linear block codes. *Communications, 2003. ICC '03. IEEE International Conference on*, 4:2904–2908, May 2003.

[83] M. Yang, W. E. Ryan, and Y. Li. Design of efficiently encodable moderate-length high-rate irregular LDPC codes. *IEEE Trans. Commun.*, 52(4):564–571, April 2004.

[84] L. Zeng, L. Lan, Y. Y. Tai, S. Lin, and K. Abdel-Ghaffar. Construction of LDPC codes for AWGN and binary erasure channels based on finite fields. In *Information Theory Workshop, 2005 IEEE*, pages 273–276, August 29 – September 1, 2005.

[85] S. Zhang and C. Schlegel. Causes and dynamics of LDPC error floors on AWGN channels. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 1025 –1032, UIUC, Allerton Retreat Center, Monticello, IL, USA, September 2011.

[86] S. Zhang and C. Schlegel. Controlling the error floor in LDPC decoding. *IEEE Trans. Commun.*, September 2012. submitted to.

[87] S. Zhang and C. Schlegel. Dispelling the error floor of LDPC codes. In *Information Theory and Applications Workshop (ITA)*, UCSD, San Diego, CA, USA, February 2012.

[88] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright. Gen03-6: Investigation of error floors of structured low-density parity-check codes by hardware emulation. *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, pages 1–6, December 2006.

[89] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright. Lowering LDPC error floors by postprocessing. *Global Telecommunications Conference, 2008. GLOBECOM '08. IEEE*, pages 1–6, December 2008.

[90] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright. Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices. *Communications, IEEE Transactions on*, 57(11):3258–3268, November 2009.

Appendix A

IEEE 802.3an RS-based LDPC Code

A.1 Code Construction

The IEEE 802.3an RS-based Low-Density Parity-Check code belongs to a special class of binary linear block codes, constructed in [21, 84]. We first introduce the general construction procedure, in order to reveal that some of the code properties, such as, minimum cycle length, symmetry, etc., are the intended results of the construction. We generate a tiny code as an example to assist us understanding the steps, for illustration purposes only.

Then the $[2048, 1723]$ IEEE 802.3an LDPC code will be constructed with proper selections of the construction parameters and modules.

A.1.1 A Class of LDPC Codes Based on Reed-Solomon Codes

Consider the Galois field $\mathrm{GF}(p^s)$, where p is a prime and s is a positive integer. Let α be a primitive element of $\mathrm{GF}(p^s)$. Then $\mathrm{GF}(p^s) = \left\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{p^s-2}\right\}$. Let ρ be an integer such that $2 \leq \rho < p^s$. Then the generator polynomial[1] of the cyclic Reed-Soloman (RS) code $\mathcal{C}$ with length $p^s - 1$, dimension $p^s - \rho + 1$ and $d_{\min} = \rho - 1$ is

$$\begin{aligned} g(X) &= \prod_{i=1}^{\rho-2} \left(X - \alpha^i\right) && \text{(A.1)} \\ &= g_0 + g_1 X + g_2 X^2 + \cdots + g_{\rho-3} X^{\rho-3} + X^{\rho-2}, && \text{(A.2)} \end{aligned}$$

where $g_i \in \mathrm{GF}^*(p^s), i = 0, 1, \ldots, \rho - 3$.

The code $\mathcal{C}$ can be shortened by deleting the first $p^s - \rho - 1$ information symbols from each codeword of $\mathcal{C}$. The shortened RS code $\mathcal{C}_b\,[\rho, 2, \rho - 1]$ has a generator matrix:

$$\begin{aligned} \mathbf{G}_b &= \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{\rho-3} & 1 & 0 \\ 0 & g_0 & g_1 & g_2 & \cdots & g_{\rho-3} & 1 \end{bmatrix}_{2\times\rho} && \text{(A.3)} \\ &\triangleq \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix}. && \text{(A.4)} \end{aligned}$$

There are p^{2s} codewords with three different weights, zero, $\rho - 1$ and ρ. As a matter of fact,

$$\mathcal{C}_b = \left\{\mathbf{k}\mathbf{G}_b = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \cdot \mathbf{G}_b \middle| k_1, k_2 \in \mathrm{GF}(p^s)\right\}. \quad \text{(A.5)}$$

[1] Note that this is the generator polynomial of the code, not the generator polynomial of the field.

For example, $\mathbf{r}_1$ and $\mathbf{r}_2$ are two weight-$(\rho-1)$ codewords, and $\mathbf{r}_1+\mathbf{r}_2$ is a weight-ρ codeword. Therefore, any two distinct codewords in $\mathcal{C}_b$ have at most one location with the same code symbol. This can be proved by way of contradiction. Let $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}_b$ and $\mathbf{c}_1 \neq \mathbf{c}_2$. Let $\mathbf{c} = \mathbf{c}_1 + \mathbf{c}_2$. Then $\mathbf{c} \in \mathcal{C}_b$ and $\mathbf{c} \neq \mathbf{0}$. Hence $\text{weight}\,(\mathbf{c}) = \rho - 1$ or ρ, which implies that $\mathbf{c}_1$ and $\mathbf{c}_2$ are different in at least $\rho - 1$ places.

Now let $\mathbf{c} \in \mathcal{C}_b$ with weight ρ. Then the set $\mathcal{C}_b^{(1)} = \left\{\beta\mathbf{c} \middle| \forall \beta \in \mathrm{GF}\,(p^s)\right\}$ contains p^s codewords in $\mathcal{C}_b$. In addition, each of the $p^s - 1$ non-zero codewords has weight ρ. Thus, any two codewords in $\mathcal{C}_b^{(1)}$ differ at every location. To see this, let c_i represent a symbol of the codeword $\mathbf{c}$ and $\beta_1, \beta_2 \in \mathrm{GF}\,(p^s)$. Then $\beta_1 c_i = \beta_2 c_i$ implies $(\beta_1 - \beta_2)c_i \equiv 0 \pmod{p^s}$. Since $c_i \neq 0$, we must have $\beta_1 - \beta_2 \equiv 0 \pmod{p^s}$ or $\beta_1 \equiv \beta_2 \pmod{p^s}$.

Partition $\mathcal{C}_b$ into p^s cosets, $\mathcal{C}_b^{(1)}, \mathcal{C}_b^{(2)}, \ldots, \mathcal{C}_b^{(p^s)}$, based on the subcode $\mathcal{C}_b^{(1)}$.

$$\mathcal{C}_b = \bigcup_{i=1}^{p^s} \mathcal{C}_b^{(i)} \tag{A.6}$$

$$\mathcal{C}_b^{(i)} \bigcap \mathcal{C}_b^{(j)} = \emptyset, \qquad (i \neq j). \tag{A.7}$$

Hence, any two codewords from two sets $\mathcal{C}_b^{(i)}$ and $\mathcal{C}_b^{(j)}$, respectively, have at most one component in common, due to the codeword weight can only be 0, $\rho - 1$ or ρ. In addition, any two codewords in each coset $\mathcal{C}_b^{(i)}$ must differ in all locations. In other words, if we arrange all the p^s codewords $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{p^s}$ of a coset $\mathcal{C}_b^{(i)}$ into a $p^s \times \rho$ array:

$$\mathbf{B}_i = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{p^s} \end{bmatrix} \tag{A.8}$$

$$= \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,\rho} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,\rho} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p^s,1} & c_{p^s,2} & \cdots & c_{p^s,\rho} \end{bmatrix}_{p^s \times \rho}, \qquad i = 1, 2, \ldots, p^s, \tag{A.9}$$

where each row represents a codeword over $\mathrm{GF}\,(p^s)$, then all the p^s elements of any column of the array are different.

Now let us map each element of $\mathrm{GF}\,(p^s)$ to a binary array. Let $\mathbf{z} = [z_0 \quad z_1 \quad \cdots \quad z_{p^s-1}]$ denote a binary p^s-tuple corresponding to the elements of $\mathrm{GF}\,(p^s)$ as

$$\mathbf{z}(0) = [1 \quad 0 \quad \cdots \quad 0] \tag{A.10}$$

$$\mathbf{z}\left(\alpha^i\right) = [0 \quad \cdots \quad 0 \quad \underbrace{1}_{z_{i+1}} \quad 0 \quad \cdots \quad 0], \qquad i = 0, 1, \ldots, p^s - 2. \tag{A.11}$$

Then any codeword $\mathbf{c} = [c_1 \quad c_2 \quad \cdots \quad c_\rho] \in \mathcal{C}_b$ can be represented by a binary ρp^s-tuple:

$$\mathbf{z}(\mathbf{c}) = [\mathbf{z}(c_1) \quad \mathbf{z}(c_2) \quad \cdots \quad \mathbf{z}(c_\rho)]\,. \tag{A.12}$$

Therefore, the array $\mathbf{B}_i$, composed of all p^s codewords from $\mathcal{C}_b^{(i)}$, can be denoted by a $p^s \times \rho p^s$ binary matrix:

$$\mathbf{A}_i = \mathbf{z}(\mathbf{B}_i) \tag{A.13}$$

$$= \begin{bmatrix} \mathbf{z}(\mathbf{c}_1) \\ \mathbf{z}(\mathbf{c}_2) \\ \vdots \\ \mathbf{z}(\mathbf{c}_{p^s}) \end{bmatrix} \tag{A.14}$$

$$= \begin{bmatrix} \mathbf{z}(c_{1,1}) & \mathbf{z}(c_{1,2}) & \cdots & \mathbf{z}(c_{1,\rho}) \\ \mathbf{z}(c_{2,1}) & \mathbf{z}(c_{2,2}) & \cdots & \mathbf{z}(c_{2,\rho}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}(c_{p^s,1}) & \mathbf{z}(c_{p^s,2}) & \cdots & \mathbf{z}(c_{p^s,\rho}) \end{bmatrix}_{p^s \times \rho p^s}, \qquad i = 1, 2, \ldots, p^s. \tag{A.15}$$

Properties of the matrices $\mathbf{A}_i$'s:

I. The row weight of each $\mathbf{A}_i$ is ρ. This is because that each block entry $\mathbf{z}(c_{i,j})$ is a weight-1 length-ρ binary vector.

II. The column weight of each $\mathbf{A}_i$ is 1. Since, within each block column, there are no two distinct blocks $\mathbf{z}(c_{i,j})$ and $\mathbf{z}(c_{k,j})$ that share the non-zero entry. In other words, $\mathbf{A}_i$ consists of a row of ρ $p^s \times p^s$ permutation matrices.

III. Any two rows from two different matrices, $\mathbf{A}_i$ and $\mathbf{A}_j$, respectively, have at most one entry in common, thanks to (A.6)–(A.7) and the weight spectrum of $\mathcal{C}_b$.

Select a positive integer γ such that $1 \leq \gamma \leq p^s$. Then the following matrix represents a (γ, ρ) regular LDPC code.

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_\gamma \end{bmatrix}_{\gamma p^s \times \rho p^s}. \tag{A.16}$$

Properties of this regular LDPC code denoted by the parity-check matrix $\mathbf{H}$:

I. Code length is ρp^s.

II. Variable node degree is $d_v = \gamma$.

III. Check node degree is $d_c = \rho$.

IV. Design code rate is $R = 1 - d_v/d_c = 1 - \gamma/\rho$.

V. The corresponding Tanner graph is free of cycles of length 4, since no two rows in $\mathbf{H}$ have more than one non-zero entry in common.

The following example will illustrate the detail construction steps.

Example A.1. Let $p = 2$ and $s = 2$. Then the field $\mathrm{GF}\left(2^2\right) = \mathbb{Z}/2\mathbb{Z} = \left\{0, 1, \alpha, \alpha^2\right\}$ has $p^s = 4$ elements. We select the monic irreducible polynomial

$$f(x) = x^2 + x + 1 \tag{A.17}$$

over $\mathbb{Z}/2\mathbb{Z}$, i.e., neither 0 or 1 is a root of $f(x)$ over the binary field $\mathrm{GF}(2)$. Therefore we have

$$\alpha^2 + \alpha + 1 = 0 \tag{A.18}$$

or

$$\alpha^2 = \alpha + 1. \tag{A.19}$$

Then

$$\alpha^3 = \alpha \cdot \alpha^2 = \alpha\left(\alpha + 1\right) = \alpha^2 + \alpha = \left(\alpha + 1\right) + \alpha = 1. \tag{A.20}$$

Let $\rho = 3$, so we can construct a shortened RS code $[3, 2, 2]$ $\mathcal{C}_b$. The generator polynomial of this code can be calculated using (A.1).

$$\begin{aligned} g(X) &= \prod_{i=1}^{\rho-2}\left(X - \alpha^i\right) && \text{(A.21)}\\ &= X - \alpha && \text{(A.22)}\\ &= \alpha + X, && \text{(A.23)} \end{aligned}$$

where $g_0 = \alpha$ and $g_1 = 1$. Then the associated generator matrix:

$$\begin{aligned} \mathbf{G}_b &= \begin{bmatrix} g_0 & g_1 & 0 \\ 0 & g_0 & g_1 \end{bmatrix}_{2\times\rho} && \text{(A.24)}\\ &= \begin{bmatrix} \alpha & 1 & 0 \\ 0 & \alpha & 1 \end{bmatrix}_{2\times 3}. && \text{(A.25)} \end{aligned}$$

There are $\left(p^s\right)^2 = 16$ choices of information bits:

$$\mathbf{k} \in \left\{ \begin{array}{cccc} \left[0 \quad 0\right], & \left[0 \quad 1\right], & \left[0 \quad \alpha\right], & \left[0 \quad \alpha^2\right], \\ \left[1 \quad 0\right], & \left[1 \quad 1\right], & \left[1 \quad \alpha\right], & \left[1 \quad \alpha^2\right], \\ \left[\alpha \quad 0\right], & \left[\alpha \quad 1\right], & \left[\alpha \quad \alpha\right], & \left[\alpha \quad \alpha^2\right], \\ \left[\alpha^2 \quad 0\right], & \left[\alpha^2 \quad 1\right], & \left[\alpha^2 \quad \alpha\right], & \left[\alpha^2 \quad \alpha^2\right] \end{array} \right\}. \tag{A.26}$$

Hence the codewords can be enumerated

$$\begin{aligned} \mathcal{C}_b &= \left\{\mathbf{k}\mathbf{G}_b\right\} && \text{(A.27)}\\ &= \left\{ \begin{array}{cccc} \left[0 \quad 0 \quad 0\right], & \left[0 \quad \alpha \quad 1\right], & \left[0 \quad \alpha^2 \quad \alpha\right], & \left[0 \quad 1 \quad \alpha^2\right], \\ \left[\alpha \quad 1 \quad 0\right], & \left[\alpha \quad \alpha^2 \quad 1\right], & \left[\alpha \quad \alpha \quad \alpha\right], & \left[\alpha \quad 0 \quad \alpha^2\right], \\ \left[\alpha^2 \quad \alpha \quad 0\right], & \left[\alpha^2 \quad 0 \quad 1\right], & \left[\alpha^2 \quad 1 \quad \alpha\right], & \left[\alpha^2 \quad \alpha^2 \quad \alpha^2\right], \\ \left[1 \quad \alpha^2 \quad 0\right], & \left[1 \quad 1 \quad 1\right], & \left[1 \quad 0 \quad \alpha\right], & \left[1 \quad \alpha \quad \alpha^2\right] \end{array} \right\} && \text{(A.28)} \end{aligned}$$

We can see the weights are 0, $\rho - 1 = 2$ or $\rho = 3$.

To construct the cosets, let the weight-ρ codeword be $\mathbf{c} = [1 \quad 1 \quad 1]$. Then

$$\begin{aligned} \mathcal{C}_b^{(1)} &= \left\{\beta\mathbf{c} \middle| \forall\beta \in \mathrm{GF}\left(2^2\right)\right\} && \text{(A.29)}\\ &= \left\{\left[0 \quad 0 \quad 0\right], \left[1 \quad 1 \quad 1\right], \left[\alpha \quad \alpha \quad \alpha\right], \left[\alpha^2 \quad \alpha^2 \quad \alpha^2\right]\right\} && \text{(A.30)}\\ \mathcal{C}_b^{(2)} &= \left[0 \quad \alpha \quad 1\right] + \mathcal{C}_b^{(1)} && \text{(A.31)}\\ &= \left\{\left[0 \quad \alpha \quad 1\right], \left[1 \quad \alpha^2 \quad 0\right], \left[\alpha \quad 0 \quad \alpha^2\right], \left[\alpha^2 \quad 1 \quad \alpha\right]\right\} && \text{(A.32)}\\ \mathcal{C}_b^{(3)} &= \alpha\left[0 \quad \alpha \quad 1\right] + \mathcal{C}_b^{(1)} && \text{(A.33)}\\ &= \left\{\left[0 \quad \alpha^2 \quad \alpha\right], \left[1 \quad \alpha \quad \alpha^2\right], \left[\alpha \quad 1 \quad 0\right], \left[\alpha^2 \quad 0 \quad 1\right]\right\} && \text{(A.34)}\\ \mathcal{C}_b^{(4)} &= \alpha^2\left[0 \quad \alpha \quad 1\right] + \mathcal{C}_b^{(1)} && \text{(A.35)}\\ &= \left\{\left[0 \quad 1 \quad \alpha^2\right], \left[1 \quad 0 \quad \alpha\right], \left[\alpha \quad \alpha^2 \quad 1\right], \left[\alpha^2 \quad \alpha \quad 0\right]\right\}. && \text{(A.36)} \end{aligned}$$

First, it can be seen that

$$\mathcal{C}_b = \bigcup_{i=1}^{4} \mathcal{C}_b^{(i)} \tag{A.37}$$

$$\mathcal{C}_b^{(i)} \bigcap \mathcal{C}_b^{(j)} = \emptyset, \qquad (i \neq j). \tag{A.38}$$

Second, if we rewrite the cosets as a $p^s \times \rho$ array,

$$\mathbf{B}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ \alpha & \alpha & \alpha \\ \alpha^2 & \alpha^2 & \alpha^2 \end{bmatrix} \tag{A.39}$$

$$\mathbf{B}_2 = \begin{bmatrix} 0 & \alpha & 1 \\ 1 & \alpha^2 & 0 \\ \alpha & 0 & \alpha^2 \\ \alpha^2 & 1 & \alpha \end{bmatrix} \tag{A.40}$$

$$\mathbf{B}_3 = \begin{bmatrix} 0 & \alpha^2 & \alpha \\ 1 & \alpha & \alpha^2 \\ \alpha & 1 & 0 \\ \alpha^2 & 0 & 1 \end{bmatrix} \tag{A.41}$$

$$\mathbf{B}_4 = \begin{bmatrix} 0 & 1 & \alpha^2 \\ 1 & 0 & \alpha \\ \alpha & \alpha^2 & 1 \\ \alpha^2 & \alpha & 0 \end{bmatrix}, \tag{A.42}$$

where each row represents a codeword. Note all the p^s elements of any column of the array are different.

Let us map the field elements to binary arrays:

$$\mathbf{z}(0) = [1 \quad 0 \quad 0 \quad 0] \tag{A.43}$$

$$\mathbf{z}(1) = [0 \quad 1 \quad 0 \quad 0] \tag{A.44}$$

$$\mathbf{z}(\alpha) = [0 \quad 0 \quad 1 \quad 0] \tag{A.45}$$

$$\mathbf{z}(\alpha^2) = [0 \quad 0 \quad 0 \quad 1]. \tag{A.46}$$

Therefore,

$$\mathbf{A}_1 = \mathbf{z}(\mathbf{B}_1) \tag{A.47}$$

$$= \left[\begin{array}{cccc|cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]_{4\times 12} \tag{A.48}$$

$$\mathbf{A}_2 = \mathbf{z}(\mathbf{B}_2) \tag{A.49}$$

$$= \left[\begin{array}{cccc|cccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]_{4\times 12} \tag{A.50}$$

$$\mathbf{A}_3 = \mathbf{z}(\mathbf{B}_3) \tag{A.51}$$

$$= \left[\begin{array}{cccc|cccc|cccc} 1&0&0&0&0&0&0&1&0&0&1&0\\ 0&1&0&0&0&0&1&0&0&0&0&1\\ 0&0&1&0&0&1&0&0&1&0&0&0\\ 0&0&0&1&1&0&0&0&0&1&0&0 \end{array}\right]_{4\times 12} \tag{A.52}$$

$$\mathbf{A}_4 = \mathbf{z}(\mathbf{B}_4) \tag{A.53}$$

$$= \left[\begin{array}{cccc|cccc|cccc} 1&0&0&0&0&1&0&0&0&0&0&1\\ 0&1&0&0&1&0&0&0&0&0&1&0\\ 0&0&1&0&0&0&0&1&0&1&0&0\\ 0&0&0&1&0&0&1&0&1&0&0&0 \end{array}\right]_{4\times 12}. \tag{A.54}$$

Suppose $\gamma = 2$, then a $(2, 3)$ regular LDPC code can be constructed as

$$\mathbf{H} = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix} \tag{A.55}$$

$$= \left[\begin{array}{cccc|cccc|cccc} 1&&&&1&&&&1&&&\\ &1&&&&1&&&&1&&\\ &&1&&&&1&&&&1&\\ &&&1&&&&1&&&&1\\ \hline 1&&&&&&1&&&1&&\\ &1&&&&&&1&1&&&\\ &&1&&1&&&&&&&1\\ &&&1&&1&&&&&1&\\ \end{array}\right]_{8\times 12}. \tag{A.56}$$

It is clear that each block is a permutation matrix and there are no four non-zero entries located at the four corners of a rectangle, thus it is cycle-4 free. This also concludes this example.

A.1.2 IEEE 802.3an LDPC Code

Regarding the LDPC code used in the IEEE802.3an standard, the parameters are selected as $p = 2$, $s = 6$ and $\rho = 32$. A shortened RS code $[32, 2, 31]$ was constructed over $\mathrm{GF}\left(2^6\right) = \mathbb{Z}/6\mathbb{Z} = \left\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{62}\right\}$. The generator polynomial of this code is

$$g(X) = \prod_{i=1}^{30}\left(X - \alpha^i\right) \tag{A.57}$$

$$= g_0 + g_1 X + g_2 X^2 + \cdots + g_{29} X^{29} + X^{30}, \tag{A.58}$$

which gives the associated generator matrix as

$$\mathbf{G}_b = \begin{bmatrix} g_0 & g_1 & \cdots & g_{29} & 1 & 0\\ 0 & g_0 & g_1 & \cdots & g_{29} & 1\end{bmatrix}_{2\times 32}. \tag{A.59}$$

There are $\left(2^6\right)^2 = 4,096$ codewords in $\mathcal{C}_b$ with weights 0, 31 and 32. It then can be partitioned into $2^6 = 64$ cosets, $\mathcal{C}_b^{(1)}, \mathcal{C}_b^{(2)}, \ldots, \mathcal{C}_b^{(64)}$, each of which has 64 codewords of $\mathcal{C}_b$. Thus the dimensions of the matrices $\mathbf{B}_i, i = 1, 2, \ldots, 64$, over $\mathrm{GF}\left(2^6\right)$ are 64×32. After rewriting the field elements into length-32 binary arrays, we obtain the candidate

matrices $\mathbf{A}_i = \mathbf{z}(\mathbf{B}_i), i = 1, 2, \ldots, 64$, with 64×2048 entries, for constructing the LDPC code. Finally, we select $\gamma = 6$ of $\mathbf{A}_i$'s to be the rows of the parity-check matrix $\mathbf{H}_{384\times 2048}$.

The specifications of the code are listed below.

- Regularity: $d_v = 6, d_c = 32$.
- The number of variable nodes or code length: $N = 2048$.
- The number of check nodes: 384.
- Information bits: $k = 1723$.
- Design code rate: $1 - d_v/d_c = 0.8125$.
- Actual code rate: $R = k/N \approx 0.8413$.
- Minimum distance: $d_{\min} \geq d_v + 1$ and even, so $d_{\min} \geq 8$ [21].
- 4-cycle free: girth $g = 6$.
- Parity-check matrix $\mathbf{H}_{384\times 2048}$ possesses a block structure [2]:

$$\mathbf{H} = \begin{bmatrix} \boldsymbol{\sigma}_{1,1} & \boldsymbol{\sigma}_{1,2} & \boldsymbol{\sigma}_{1,3} & \cdots & \boldsymbol{\sigma}_{1,32} \\ \boldsymbol{\sigma}_{2,1} & \boldsymbol{\sigma}_{2,2} & \boldsymbol{\sigma}_{2,3} & \cdots & \boldsymbol{\sigma}_{2,32} \\ \boldsymbol{\sigma}_{3,1} & \boldsymbol{\sigma}_{3,2} & \boldsymbol{\sigma}_{3,3} & \cdots & \boldsymbol{\sigma}_{3,32} \\ \boldsymbol{\sigma}_{4,1} & \boldsymbol{\sigma}_{4,2} & \boldsymbol{\sigma}_{4,3} & \cdots & \boldsymbol{\sigma}_{4,32} \\ \boldsymbol{\sigma}_{5,1} & \boldsymbol{\sigma}_{5,2} & \boldsymbol{\sigma}_{5,3} & \cdots & \boldsymbol{\sigma}_{5,32} \\ \boldsymbol{\sigma}_{6,1} & \boldsymbol{\sigma}_{6,2} & \boldsymbol{\sigma}_{6,3} & \cdots & \boldsymbol{\sigma}_{6,32} \end{bmatrix}_{384\times 2048}, \tag{A.60}$$

  where each $\boldsymbol{\sigma}_{i,j}$ is a 64×64 permutation matrix.

A.2 Absorption Sets

We have identified the existence of the first few absorption sets of the IEEE 802.3an LDPC code and are able to enumerate the smaller ones. The results are listed in Table A.1, which is the same as Table 3.2 in Chapter 3.

Recall by Definition 3.3 that an ordered pair (a, b) denotes an absorption set, where a is size of the set and b is the number of unsatisfied check nodes attached to the set.

It is believed that the "smaller" the absorption set, the more severe the effect on the error floor phenomenon. Thus our target is to find the minimal absorption sets in terms of a, b and $b/(6a)$.

Since $d_v = 6$, then $a \geq 5$ by the definition of absorption sets and this 802.3an LDPC code is 4-cycle free. In other words, if $a \leq 4$, it is impossible to connect a nodes with at least $4a$ edges without creating any cycles of length 4.

Now let us start with $a = 5$ to show every number in Table A.1 is true in the following subsections.

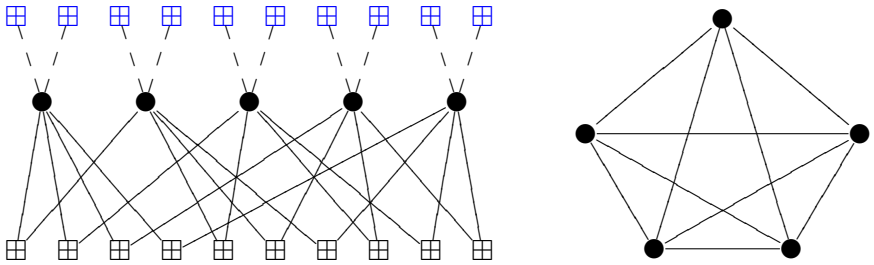
[2] In this write-up we use the $\mathbf{H}$ matrix from [21] rather than the one listed in the IEEE 802.3 standard, since it has a very symmetric structure.

Table A.1: The first few absorption sets of the IEEE 802.3an LDPC code.

a	b	**Existence**	**Multiplicity**	$\frac{b}{ad_v}^3$
< 5		No		
5	10	No		
6	6	No		
	8			
	10			
	12			
7	0	No		
	2			
	4			
	6			
	8			
	10			
	12	Yes	$65,472$	0.2857
	14	Yes	$14,720$	0.3333
8	0	No		
	2			
	4			
	6			
	8	Yes	$14,272$	0.1667
	10	No		
	12	Yes	$44,416$	0.2500
	14	Yes	$88,896$	0.2917
	16	Yes	$661,824$	0.3333
9	0	No		
	2			
	4			
	6			
	8			
	10			
	12	Yes	?	0.2222
	14	Yes (at least from $(10, 10)$)	?	0.2593
	16	Yes	?	0.2963
	18			0.3333
10	0	No		
	2			
	4			
	6			
	8	?		
	10	Yes	> 192?	0.1667
	12	Yes	?	0.2000
	14			0.2333
	16			0.2667
	18			0.3000
	20			0.3333

A.2.1 $a = 5$

Each variable node in such set is able to connect the set at most 4 times without generating any 4-cycle. Clearly there is only one connecting topology as shown in Figure A.1, which is identical to Figure 3.5.



(a) Check nodes shown. (b) Check nodes hidden.

Figure A.1: The only possible topology of $(5, 10)$ absorption sets.

Theorem A.1. *The number of size-*5 *absorption sets is* 0.

Proof. Search the $\mathbf{H}$ matrix by applying the topology, plus making use of some of the properties listed in [23] for array-based LDPC codes. Algorithm 1 was developed and no $(5, 10)$ set was found. □

Algorithm 1 can be used to either determine the existence of absorption sets, or to enumerate all $(5, 10)$ sets. However, we note that it may output duplicates due to the symmetric structure of the absorption sets.

A.2.2 $a = 6$

With more nodes, hence more edges, involved, the graphical combinations of possible absorption sets grow dramatically. Let us develop a couple of notations to help prove the existence of absorption sets in an organized fashion.

Following Definition 3.5, the degree of each vertex v in the topology graph with check nodes hidden, $\mathrm{Deg}(v)$, must satisfy the following constraints:

Definition A.1. Let an unordered array $[\mathrm{Deg}(v_i) : i = 1, 2, \ldots, a]$ denote a class of (a, b) absorption sets. Clearly $\mathrm{Deg}(v_i) \in \{4, 5, 6\}$ and $\sum_{i=1}^{a} \mathrm{Deg}(v_i) = 6a - b$.

We start with the smallest possible b since we would like to find the minimal absorption sets. In addition, because there are only six nodes, then $\mathrm{Deg}(v) \leq 5$ for any v in such sets to avoid 4-cycles.

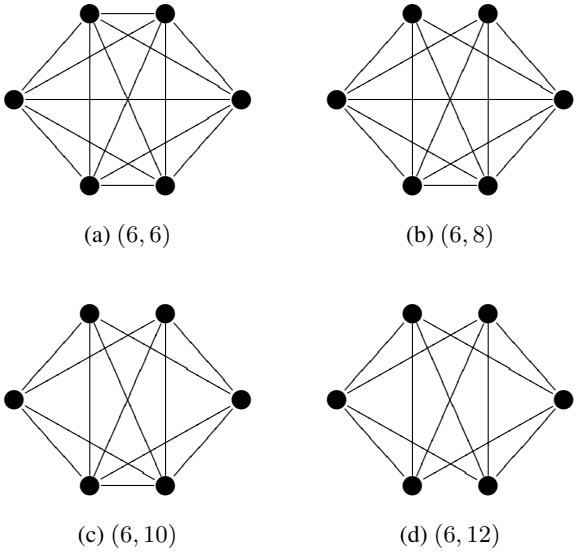
1. $b = 6$: graphically, only one class, $[5, 5, 5, 5, 5, 5]$, exists and there is only one possible topology, which is very symmetric as shown in Figure A.2(a). By removing any node from Figure A.2(a), we will get $[4, 4, 4, 4, 4]$, exactly the one in Figure A.1(b).

2. $b = 8$: there is only one class $[5, 5, 5, 5, 4, 4]$ as shown in Figure A.2(b).

   Removing either degree-4 node will reduce to $[4, 4, 4, 4, 4]$, Figure A.1(b), as well, since there is no connection between the two degree-4 nodes.

3. $b = 10$: there exists only one class $[5, 5, 4, 4, 4, 4]$ and one possible topology shown in Figure A.2(c).

4. $b = 12$: there is only one class $[4, 4, 4, 4, 4, 4]$ and one possible topology shown in Figure A.2(d).



(a) $(6, 6)$ (b) $(6, 8)$

(c) $(6, 10)$ (d) $(6, 12)$

Figure A.2: Possible topologies of $(6, b)$ absorption sets.

Hence only Figures A.2(c)–A.2(d) need to be searched against $\mathbf{H}$.

Theorem A.2. *The number of size-*6 *sets is* 0*.*

Proof. Firstly, $(6, 6)$ and $(6, 8)$ absorption sets do not exist. Otherwise it is contradictory to Theorem A.1. Then, by searching the $\mathbf{H}$ matrix via inputting the topologies Figures A.2(c) and A.2(d) into an Algorithm 1-type of searcher, no such sets are found. □

A.2.3 $a = 7$

A.2.3.1 $b < 12$

Lemma A.3. *The number of* $(7, b)$ *absorption sets with* $b < 12$ *is zero.*

Proof. We apply Definition A.1 and the pigeonhole principle to prove this.

1. $b = 0$. $(7, 0)$ absorption set is a weight-7 codeword. Since $d_{\min} \geq 8$, $b \neq 0$ when $a = 7$.

2. $b = 2$. By the constraints in Definition A.1, there are two classes:

   (a) $[6, 6, 6, 6, 6, 5, 5]$: by removing either degree-5 node, it will give us a $(6, 6)$ absorption set which does not exist by Theorem A.2. Hence this class of $(7, 2)$ absorption set does not exist, either.

(b) $[6, 6, 6, 6, 6, 6, 4]$: this setting graphically cannot exist, because the group of degree-6 nodes emanate out at least six edges which are impossible to be all connected to one degree-4 node.

3. $b = 4$. By the constraints in Definition A.1, there are three classes:

   (a) $[6, 6, 6, 5, 5, 5, 5]$: removing any degree-5 node will give us a $(6, 8)$ absorption set.

   (b) $[6, 6, 6, 6, 5, 5, 4]$: removing the degree-4 node will get us a $(6, 6)$ absorption set.

   (c) $[6, 6, 6, 6, 6, 4, 4]$: this is infeasible since the group of five degree-6 nodes requires ten edges coming out of the couple of degree-4 nodes.

4. $b = 6$. By the constraints in Definition A.1, there are four classes:

   (a) $[6, 5, 5, 5, 5, 5, 5]$: removing any degree-5 node will get us a $(6, 10)$ absorption set.

   (b) $[6, 6, 5, 5, 5, 5, 4]$: removing the degree-4 node will get us a $(6, 8)$ absorption set.

   (c) $[6, 6, 6, 5, 5, 4, 4]$: each of the degree-5 nodes and each of the degree-6 nodes needs at least one and two edges coming out of the two degree-4 nodes, respectively. That makes eight. So there is no connection between the degree-4 nodes. Thus removing either of them will get us a $(6, 8)$ absorption set.

   (d) $[6, 6, 6, 6, 4, 4, 4]$: each of the degree-6 nodes needs three edges coming out of the two degree-4 nodes. That makes twelve. So there is no connection among the three degree-4 nodes. Thus removing any of them will get us a $(6, 8)$ absorption set.

5. $b = 8$. By the constraints in Definition A.1, there are four classes:

   (a) $[5, 5, 5, 5, 5, 5, 4]$: removing the degree-4 node will get us a $(6, 10)$ absorption set.

   (b) $[6, 5, 5, 5, 5, 4, 4]$: let us study the intrinsic connections between the two degree-4 nodes:

Figure A.3: Possible intrinsic connections between two degree-4 nodes.

   i. Not connected as shown in Figure A.3(a). Removing either degree-4 node will reduce to a $(6, 10)$ absorption set.

   ii. Connected as shown in Figure A.3(b). We consider the connections between the two degree-4 nodes and the other five nodes in the set. The other five nodes need at least six edges coming out of the two degree-4 nodes, so

there is no degree-5 node connected to both degree-4 nodes. Thus removing both of them generates a $(5, 10)$ absorption set.

(c) $[6, 6, 5, 5, 4, 4, 4]$: the two degree-5 nodes and the two degree-6 nodes need at least ten edges coming from the three degree-4 nodes. Hence there should be at most one connection among the group of degree-4 nodes:

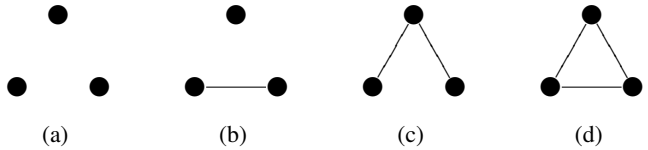


Figure A.4: Possible intrinsic connections among three nodes.

i. No connection as shown in Figure A.4(a). Removing any degree-4 node will reduce to a $(6, 10)$ absorption set.

ii. One connection as shown in Figure A.4(b). Removing the topmost degree-4 node will reduce to a $(6, 10)$ absorption set.

(d) $[6, 6, 6, 4, 4, 4, 4]$: the three degree-6 nodes are all interconnected and, hence, need exactly twelve edges coming from the four degree-4 nodes. This leaves Figure A.5(d) the only feasible case. Then removing either the top or the bottom pair of degree-4 nodes in Figure A.5(d) will reduce it to a $(5, 10)$ absorption set.

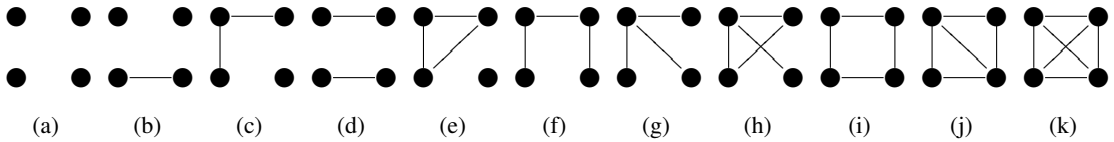


Figure A.5: Possible intrinsic connections among four nodes.

6. $b = 10$. By the constraints in Definition A.1, there are three classes:

(a) $[5, 5, 5, 5, 4, 4, 4]$: the four degree-5 nodes need at least eight edges coming from the three degree-4 nodes. Hence there should be no more than two connections among the group of degree-4 nodes:

i. No connection as shown in Figure A.4(a). Removing any degree-4 node will reduce to a $(6, 12)$ absorption set.

ii. One connection as shown in Figure A.4(b). Removing the topmost degree-4 node will reduce to a $(6, 12)$ absorption set.

iii. Two connections as shown in Figure A.4(c). No node can be removed to get another absorption set. It is straightforward that there is only one possible topology to satisfy this as shown in Figure A.6(a). Therefore we have to go check its existence against the $\mathbf{H}$ matrix.

(b) $[6, 5, 5, 4, 4, 4, 4]$: the two degree-5 nodes and the degree-6 node need twelve or ten edges emanating from the four degree-4 nodes. Hence there should be two or three connections among the group of degree-4 nodes, respectively.

i. Two connections: there are two cases:
   A. Figure A.5(c) is infeasible since the bottom-right node cannot achieve degree 4.
   B. It is straightforward that there is only one possible topology to satisfy Figure A.5(d) as shown in Figure A.6(b).

ii. Three connections: there are three cases:
   A. We claim that Figure A.5(e) is infeasible since the bottom-right node can only have three connections at most.
   B. There is only one possible topology to satisfy Figure A.5(f) as shown in Figure A.6(c).
   C. There is only one possible topology to satisfy Figure A.5(g) as shown in Figure A.6(d).

(c) $[6, 6, 4, 4, 4, 4, 4]$: there is only one possible topology in this class as shown in Figure A.6(e).

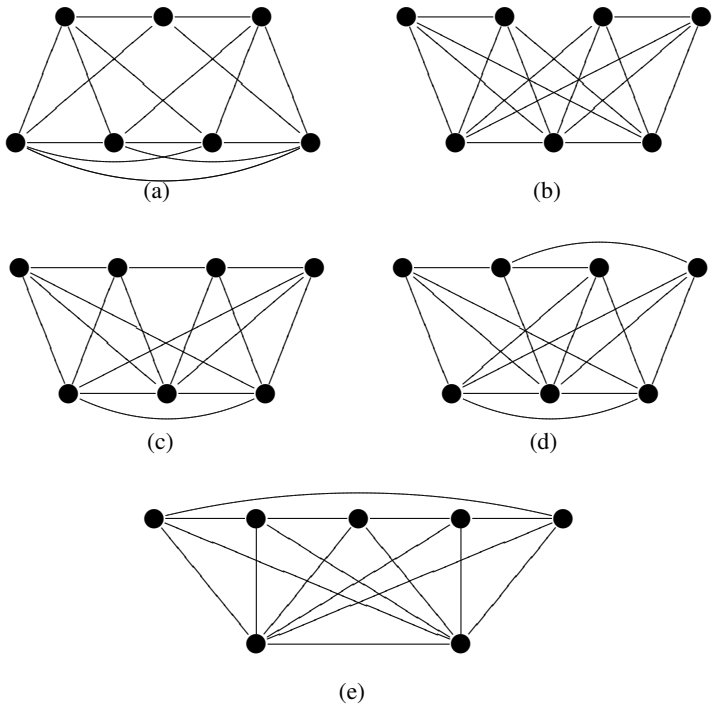


Figure A.6: The possible topologies of $(7, 10)$ absorption sets that cannot be ruled out without searching against the parity-check matrix $\mathbf{H}$.

Note that since a is large enough, the neighboring check nodes are able to be connected to the absorption set four times rather than only two, as shown in Figure A.7, which also falls in the $[6, 5, 5, 4, 4, 4, 4]$ class.

After checking against the $\mathbf{H}$ matrix, the $(7, 10)$ absorption sets listed in Sections 6(a)iii, 6(b)iB, 6(b)iiB, 6(b)iiC, 6c and above, corresponding to topologies shown in Figure A.6 and Figure A.7, do not exist. □

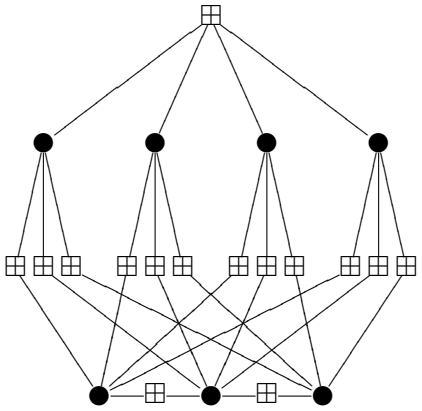


Figure A.7: One check node connecting to a $(7, 10)$ absorption set four times.

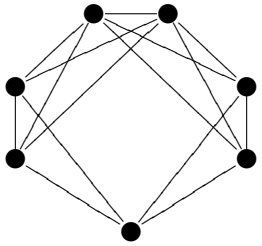


Figure A.8: Topology of the $(7, 12)$ absorption set.

A.2.3.2 $b = 12$

Lemma A.4. *The* $(7, 12)$ *absorption set exists and there are* $65,472$ *of them.*

Proof. The existence can be seen, at least, as a reduction from $(8, 8)$ sets, which will be studied in Section A.2.4.2. As a matter of fact, $179,648$ such sets have been enumerated and all of them share the topology shown in Figure A.8, which is a reduction from the topology of the $(8, 8)$ absorption set shown in Section A.2.4.2.

Even though this topology can be obtained by removing one node from the one of the $(8, 8)$ absorption set, apparently not all $(7, 14)$ sets can be obtained that way. We will see that there are $14,272$ $(8, 8)$ sets and they can generate only $14272 \times 8 = 114176$ $(7, 12)$ absorption sets (no duplicates). Hence there are $179648 - 114176 = 65472$ $(7, 12)$ sets that are not contained in the $(8, 8)$ ones. □

Note that the extrinsic degree is large now. Recall that our priority is to identify the minimal absorption sets. We will see soon in next section that there are smaller b's.

A.2.3.3 $b = 14$

Lemma A.5. *There exist* $14,720$ $(7, 14)$ *absorption sets which share two topologies.*.

Proof. There are three candidate topologies. Let us show how the first two are derived. The third one will be presented at the end of the proof.

Evidently there is only one class: $[4, 4, 4, 4, 4, 4, 4]$. We start with Figure A.9.

There exist two cases.

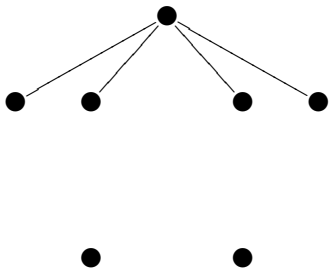


Figure A.9: Step 1 in constructing a $(7, 14)$ absorption set.

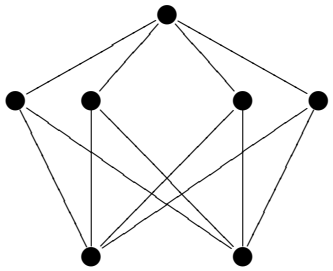


Figure A.10: Step 2 in constructing a $(7, 14)$ absorption set: Case I.

1. Case I: the bottom two nodes are not connected. Then we have Figure A.10.

   By connecting the remaining nodes in the only possible way, we achieve the first possible topology as shown in Figure A.11.

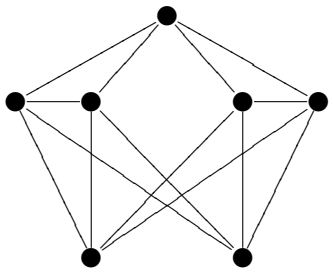


Figure A.11: Case I of $(7, 14)$ absorption sets.

2. In Case II, the bottom two nodes in Figure A.9 are connected as shown in Figure A.12.

   Then the the other bottom node has two options.

   (a) Case II.A as shown in Figure A.13.
   By finishing connecting the remain nodes in the only feasible way, Figure A.14 shows another sketch.

   (b) Case II.B of Figure A.12 is shown in Figure A.15.
   Finishing up the connections, we get another topology in Figure A.16

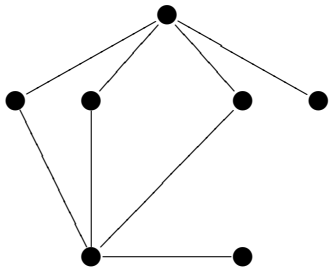


Figure A.12: Step 2 in constructing a $(7, 14)$ absorption set: Case II.

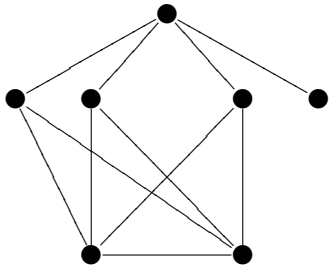


Figure A.13: Step 3 in constructing a $(7, 14)$ absorption set: Case II.A.

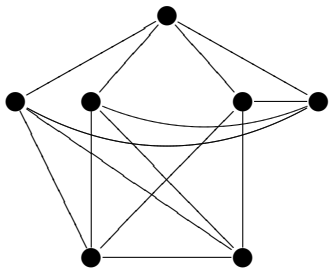


Figure A.14: Case II.A of $(7, 14)$ absorption sets.

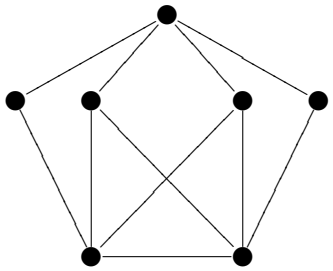


Figure A.15: Step 3 in constructing a $(7, 14)$ absorption set: Case II.B.

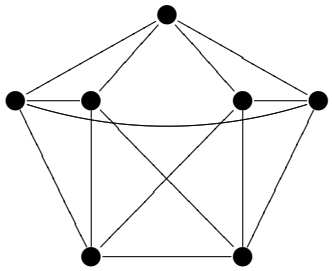
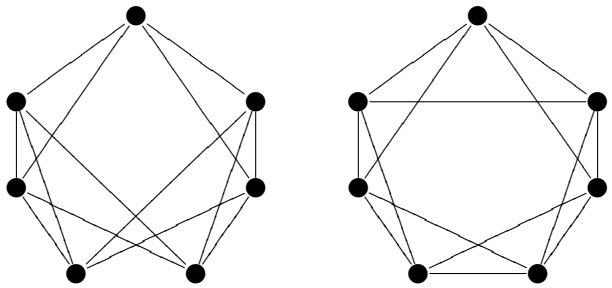


Figure A.16: Case II.B of $(7, 14)$ absorption sets.

By reorganizing the nodes, it turns out that the sketches under Case I and Case II.A are identical. In other words, both Figure A.11 and Figure A.14 can be arranged to the appearance shown in Figure A.17(a). Figure A.16 can also be arranged to a nicer equivalent looking as Figure A.17(b).



(a) Case I or Case II.A. (b) Case II.B.

Figure A.17: Two out of three possible topologies of $(7, 14)$ absorption sets.

Similar to Figure A.7, check nodes can be connected to the set four times as shown in Figure A.18. By searching against the $\mathbf{H}$, such configuration does not exist.

By searching against the $\mathbf{H}$ matrix, there are $2,304$ $(7, 14)$ sets that share the connectivity of Figure A.17(a), $12,416$ of Figure A.17(b) and none of Figure A.18. The average multiplicity of each variable node appeared in such sets is $14720 \times 7 \div 2048 = 50.3125$. Due to the block structure of the $\mathbf{H}$ matrix, certain group of variable nodes do share the same multiplicity, as listed in Table A.2.

□

Combining Lemmas A.3, A.4 and A.5, we obtain

Theorem A.6. *The number of* $(7, b)$ *absorption sets with* $b < 12$ *is* 0*, while both* $(7, 10)$ *and* $(7, 14)$ *sets exist.*

A.2.4 $a = 8$

If $a = 8$, then b could only be even numbers.

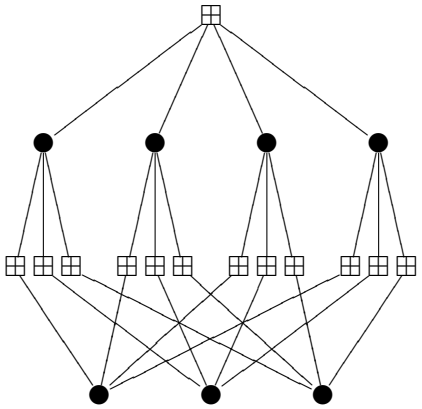


Figure A.18: The last of the three possible topologies of $(7, 14)$ absorption sets where one check node is connected to the set four times.

A.2.4.1 $b < 8$

Lemma A.7. *For $b < 8$, there exits no $(8, b)$ absorption set.*

Proof. We apply Definition A.1 and the pigeonhole principle to prove this.

1. $b = 0$. It will be a class of $[6, 6, 6, 6, 6, 6, 6, 6]$ absorption sets. We obtain the perfect symmetric Figure A.19 again as Figure A.1(b) or Figure A.2(a).

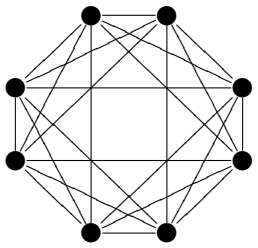


Figure A.19: Only possible topology of $(8, 0)$ absorption set.

   Removing any node will reduce it to a $(7, 6)$ absorption set.

2. $b = 2$. By the constraints in Definition A.1, there are two classes:

   (a) $[6, 6, 6, 6, 6, 6, 5, 5]$: removing either degree-5 node will get us a $(7, 6)$ absorption set.

   (b) $[6, 6, 6, 6, 6, 6, 6, 4]$: removing the degree-4 node will get us a $(7, 4)$ absorption set.

3. $b = 4$. By the constraints in Definition A.1, there are three classes:

   (a) $[6, 6, 6, 6, 5, 5, 5, 5]$: removing any degree-5 node will get us a $(7, 8)$ absorption set.

   (b) $[6, 6, 6, 6, 6, 5, 5, 4]$: removing the degree-4 node will get us a $(7, 6)$ absorption set.

Table A.2: The multiplicity of each variable node in $(7, 14)$ absorption sets.

Group	Variable Node	Multiplicity	Group	Variable Nodes	Multiplicity
1	0—63	49	17	1024—1087	42
2	64—127	64	18	1088—1151	41
3	128—191	61	19	1152—1215	48
4	192—255	69	20	1216—1279	59
5	256—319	33	21	1280—1343	59
6	320—383	71	22	1344—1407	49
7	384—447	53	23	1408—1471	65
8	448—511	49	24	1472—1535	41
9	512—575	49	25	1536—1599	42
10	576—639	37	26	1600—1663	25
11	640—703	44	27	1664—1727	30
12	704—767	50	28	1728—1791	58
13	768—831	54	29	1792—1855	44
14	832—895	49	30	1856—1919	53
15	896—959	67	31	1920—1983	56
16	960—1023	57	32	1984—2047	42

(c) $[6, 6, 6, 6, 6, 6, 4, 4]$: removing both degree-4 nodes will get us a $(6, 10)$ or $(6, 8)$ absorption set, depends on if the degree-4 nodes are incident or not.

4. $b = 6$. By the constraints in Definition A.1, there are four classes:

(a) $[6, 6, 5, 5, 5, 5, 5, 5]$: removing any degree-5 node will get us a $(7, 10)$ absorption set.

(b) $[6, 6, 6, 5, 5, 5, 5, 4]$: removing the degree-4 node will get us a $(7, 8)$ absorption set.

(c) $[6, 6, 6, 6, 5, 5, 4, 4]$: let us study the intrinsic connections between the two degree-4 nodes:

i. Not connected as shown in Figure A.3(a). Removing either degree-4 node will reduce it to a $(7, 8)$ absorption set.

ii. Connected as shown in Figure A.3(b). We consider the connections between the two degree-4 nodes and the other six nodes in the set. There are six edges coming out of the two degree-4 nodes and at least four of the six edges must go to the four degree-6 nodes, respectively. So at most two edges coming out of the two degree-4 nodes can be connected the two degree-5 nodes.

A. If either of the two degree-5 nodes is connected to the two degree-4 nodes at most once, then removing both degree-4 nodes will get us a $(6, 8)$ absorption set.

B. If one degree-5 node is connected to both degree-4 nodes, then removing the other degree-5 node will get us a $(7, 10)$ absorption set.

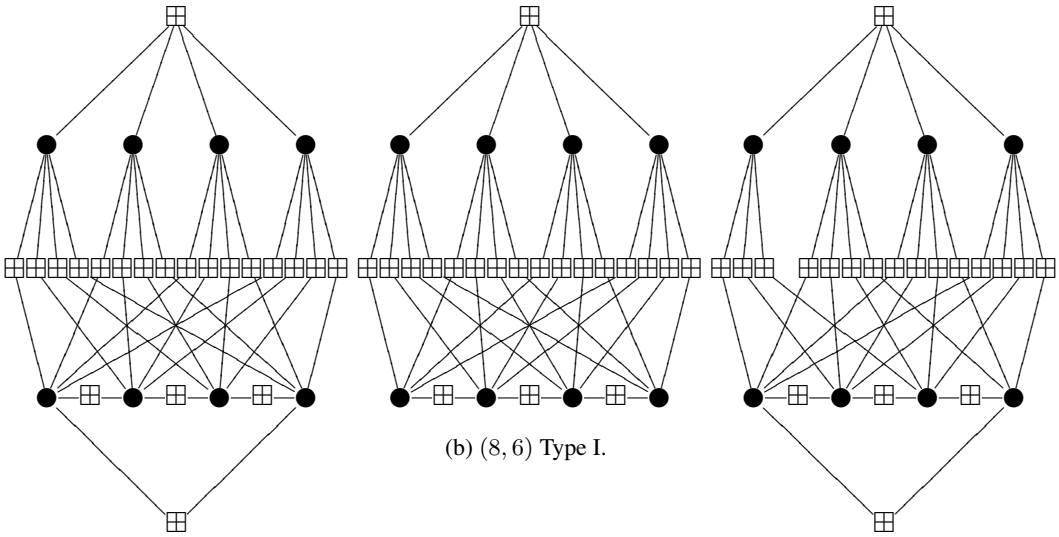
(d) $[6, 6, 6, 6, 6, 4, 4, 4]$: let us study the intrinsic connections between the three degree-4 nodes:

i. No connection as shown in Figure A.4(a). Removing any degree-4 node will reduce it to a $(7, 8)$ absorption set.

ii. One connection as shown in Figure A.4(b). Removing the topmost degree-4 node will reduce it to a $(7, 8)$ absorption set.

iii. More than one connections as shown in Figure A.4(c) and Figure A.4(d). There are at most eight edges coming out of these three degree-4 nodes. However, there are five degree-6 nodes, which require ten edges from the three degree-4 nodes. Thus they will not match each other. These are not feasible connections.

We have exhausted all possibilities above, considering the satisfied check nodes are connected to the absorption set only twice. In Figure A.20, we present the topologies when a check node is connected four times to an absorption set.



(a) $(8, 4)$.

(b) $(8, 6)$ Type I.

(c) $(8, 6)$ Type II.

Figure A.20: One check node connecting to $(8, 4)$ and $(8, 6)$ absorption sets four times.

Figure A.20(a) represents an $(8, 4)$ set of class $[6, 6, 6, 6, 5, 5, 5, 5]$. Removing any degree-5 node will reduce it to a $(7, 10)$ absorption set. Figure A.20(b) denotes an $(8, 6)$ set of class $[6, 6, 5, 5, 5, 5, 5, 5]$. Removing either degree-5 node in the bottom row will reduce it to a $(7, 10)$ absorption set. Figure A.20(a) represents an $(8, 4)$ set of class $[6, 6, 6, 5, 5, 5, 5, 4]$. Removing the degree-4 node will reduce it to a $(7, 10)$ absorption set.

Thus the lemma follows. □

Corollary A.8. *The minimum distance is lower bounded by* 10.

Proof. No $(8, 0)$ absorption set implies that there is no weight-8 codeword, as seen in Figure A.19. Thus, $d_{\min} > 8$, and has to be even [21]. Therefore $d_{\min} \geq 10$. □

A.2.4.2 $b = 8$

Lemma A.9. *There exits no* $(8, 8)$ *absorption set that contains any degree-*6 *variable node.*

Proof. By the constraints in Definition A.1, we have the following four classes, each of which contains at least one degree-4 variable node.

1. $[6, 5, 5, 5, 5, 5, 5, 4]$: removing the degree-4 node will reduce to a $(7, 10)$ absorption set.

2. $[6, 6, 5, 5, 5, 5, 4, 4]$: let us study the intrinsic connections between the two degree-4 nodes:

   (a) Not connected as shown in Figure A.3(a). Removing either degree-4 node will reduce to a $(7, 10)$ absorption set.

   (b) Connected as shown in Figure A.3(b). We consider the connections between the two degree-4 nodes and the other six nodes in the set. There are six edges coming out of the two degree-4 nodes and at least two and at most four of the six edges must go to the two degree-6 nodes, respectively. So at most four and at least two edges coming out of the two degree-4 nodes can be connected the four degree-5 nodes.

      i. Two edges between the group of degree-4 nodes and the group of degree-6 nodes are shown in Figure A.23(a). Note that under the conditions in the these two cases, the two degree-6 nodes are connected to each other and either of them has to be connected to all the degree-5 nodes. In addition, there are four edges coming from the degree-4 nodes to the four degree-5 nodes. Thus,

         A. if there is no degree-5 node sharing the two degree-4 nodes, then removing the two degree-4 nodes will reduce it to a $(6, 10)$ absorption set.

         B. if there is one and only one degree-5 node sharing the two degree-4 nodes, then there exist two topologies as shown in Figure A.21. Removing the two degree-4 nodes and that degree-5 node, all in blue in Figure A.21, will reduce them to $(5, 10)$ absorption sets, respectively.

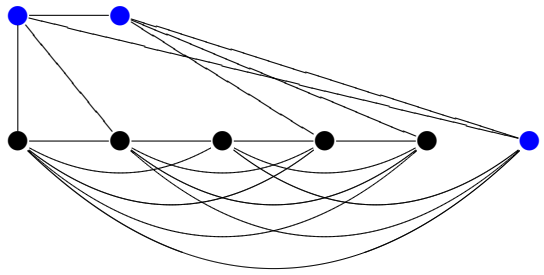
      ii. Three edges between the group of degree-4 nodes and the group of degree-6 nodes are shown in Figure A.23(b). Note that under the conditions in the above two cases, the two degree-6 nodes are connected to each other and the bottom-right degree-6 node has to be connected to all the degree-5 nodes. In addition, there are three edges coming from the degree-4 nodes to the four degree-5 nodes. Thus,

         A. if there is no degree-5 node sharing the two degree-4 nodes, then removing the two degree-4 nodes will reduce it to a $(6, 10)$ absorption set.

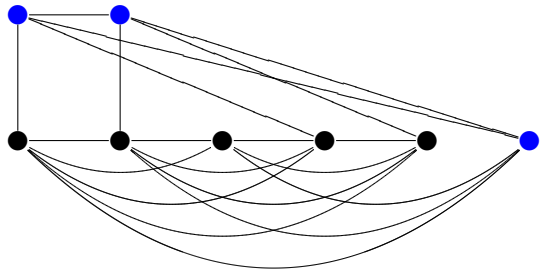
         B. if there is one and at most one degree-5 node sharing the two degree-4 nodes, then the topology can be obtained as shown in Figure A.22(a). Removing the two degree-4 nodes and that degree-5 node will reduce it to a $(5, 10)$ absorption set.

      iii. Four edges between the group of degree-4 nodes and the group of degree-6 nodes are shown in Figure A.23(c).

         A. if there is no degree-5 node sharing the two degree-4 nodes, then removing the two degree-4 nodes will reduce it to a $(6, 10)$ absorption set.



(a)



(b)

Figure A.21: Two possible topologies of $(8, 8)$ absorption set with degree-6 variable nodes.

B. if there is one and at most one degree-5 node sharing the two degree-4 nodes, the topology is shown in Figure A.22(b).
Removing the two degree-4 nodes and that degree-5 node will reduce it to a $(5, 10)$ absorption set.

3. $[6, 6, 6, 5, 5, 4, 4, 4]$: The intrinsic connections among the three degree-4 nodes:

   (a) No connection as shown in Figure A.4(a). Removing any degree-4 node will reduce it to a $(7, 10)$ absorption set.

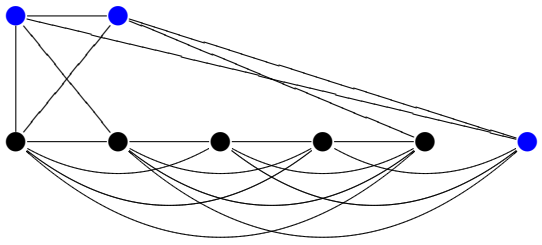
   (b) One connection as shown in Figure A.4(b). Removing the topmost degree-4 node will reduce it to a $(7, 10)$ absorption set.

   (c) Two connections as shown in Figure A.4(c). Now, there are eight edges coming out the group of degree-4 nodes. However, each of the two degree-5 nodes and each of the three degree-6 nodes needs one and two connections from the group of degree-4 nodes, respectively. That makes eight. Thus, removing the three degree-4 nodes will reduce it to a $(5, 10)$ absorption set.

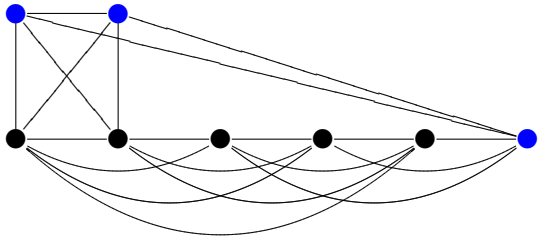
   (d) Three connections as shown in Figure A.4(d). There are only six edges coming out of these three degree-4 nodes. And they will not match the group of degree-4 nodes and the group of the other five nodes remained in the set.

4. $[6, 6, 6, 6, 4, 4, 4, 4]$: The group of degree-6 nodes need at least twelve edges from the group of degree-4 nodes. So

   (a) if there is no connection among the four degree-4 nodes as shown in Figure A.5(a), then removing any degree-4 node will reduce it to a $(7, 10)$ absorption set.

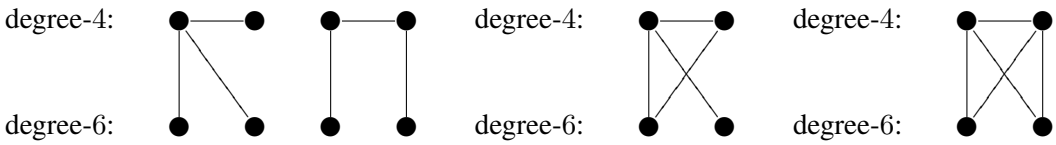


(a)



(b)

Figure A.22: Another two possible topologies of $(8, 8)$ absorption set with degree-6 variable nodes.



(a) (b) (c)

Figure A.23: Possible intrinsic connections between two groups of variable nodes which are grouped by degrees. The top row represents the degree-4 nodes, whereas the bottom is degree-6.

(b) if there is only one connection among the four degree-4 nodes as shown in Figure A.5(b), then removing either topmost degree-4 node will reduce it to a $(7, 10)$ absorption set.

(c) if there are two connections among the four degree-4 nodes, we have two cases:

i. Removing the bottom-right degree-4 node in Figure A.5(c) will reduce it to a $(7, 10)$ absorption set.

ii. Removing either the top or the bottom couple of degree-4 nodes in Figure A.5(d) will reduce it to a $(6, 10)$ absorption set.

(d) if there are more than two connections among the four degree-4 nodes, then there will be less than twelve edges going out from the group of degree-4 nodes to the group of degree-6 nodes.

□

Once again, for an absorption set with this many members, a satisfied check node is able to be incident to more than two variable nodes. The possible connectivities are shown in Figure A.24 and Figure A.25, which can be obtained by removing edges from the ones shown in Figure A.20.

Removing the degree-4 node, the bottom-left degree-4 node or the bottom-right degree-4 node in Figures A.24(a), A.24(c) or A.24(e), respectively, reduces it to a $(7, 10)$ absorption set, while removing the two degree-4 nodes in Figure A.24(f) reduces it to a $(6, 10)$ absorption set. By checking against the $\mathbf{H}$ matrix, the ones shown in Figures A.24(b), A.24(d) and A.25 do not exist.

Note that Figures A.24(b) and A.25 are in the class $[5, 5, 5, 5, 5, 5, 5, 5]$. We claim that, graphically, there exist five possible topologies under this class of the $(8, 8)$ sets. Let us find the other three by restricting that a satisfied check node can only be connected to the set twice.

We start with node ❶ as shown in Figure A.26.

Next, we discuss the connection between ❺ and ❻ in the following two cases.

1. Case I: ❺ and ❻ are not connected. Then both of them have to be connected to the remaining five nodes, as shown in Figure A.27. After connecting the remaining five nodes in the only possible way, we obtain Figure A.28. Reorganize this figure into a symmetric form as Figure A.35(a).

2. Case I: ❺ and ❻ are connected. Then node ❻ has to connect four of the nodes ❷, ❸, ❹, ❼ and ❽. Without loss of generality, assume Figure A.29 is the case.

   Now regarding node ❺, there are two connecting choices for it.

   (a) Case II.A: If ❺ is not connected to ❷, then it has to connect ❶, ❹, ❼ and ❽, as shown in Figure A.30.

   Then node ❷ has only one connecting choice as shown in FigureA.31.

   By connecting the remaining nodes, we obtain another possible topology for an $(8, 8)$ absorption set as shown in FigureA.35(b).

   (b) Case II.B: if ❺ is connected to ❷, then without loss of generality assume it is connected to ❸, ❹ and ❼, as shown in Figure A.32.

   Node ❷ has two choices:

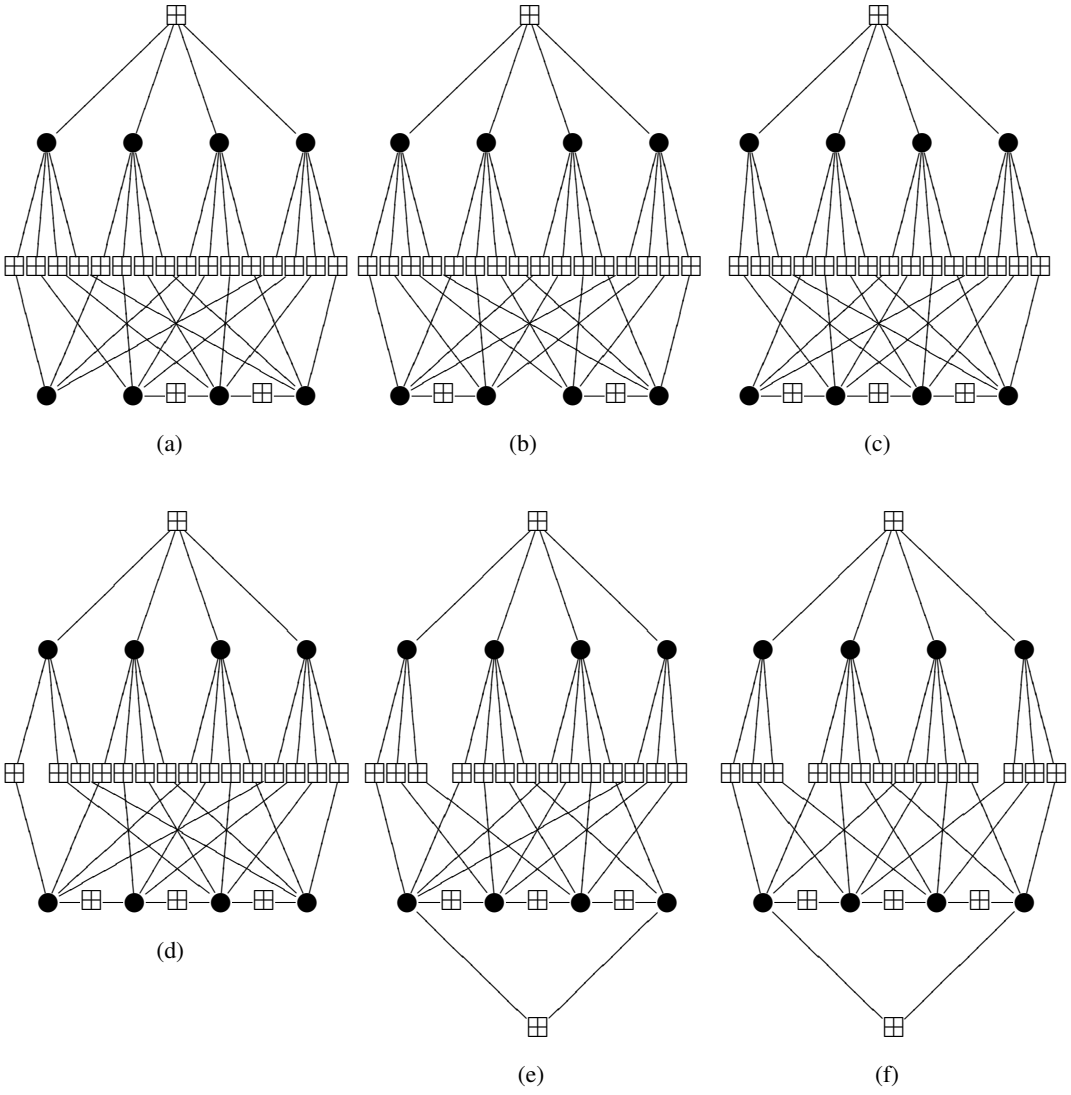


Figure A.24: One check node connecting to $(8, 8)$ sets four times.

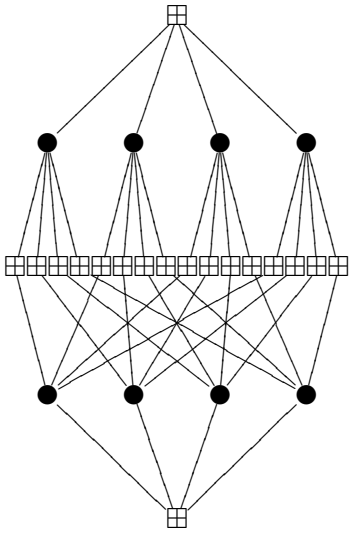


Figure A.25: Two check nodes connecting to $(8, 8)$ set four times.

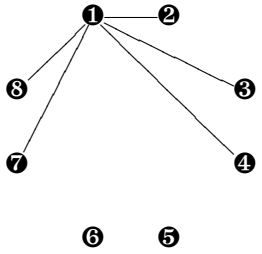


Figure A.26: Step 1 in constructing an $(8, 8)$ absorption set.

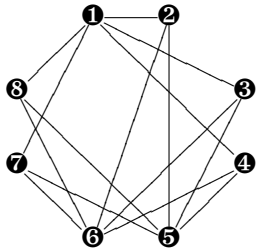


Figure A.27: Step 2 in constructing an $(8, 8)$ absorption set: Case I.

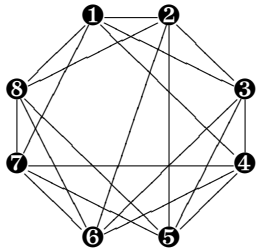


Figure A.28: One topology of an $(8, 8)$ absorption set: Case I.

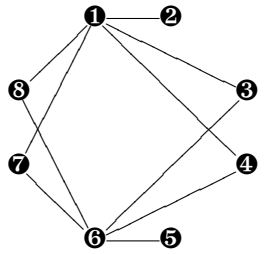


Figure A.29: Step 2 in constructing an $(8, 8)$ absorption set: Case II.

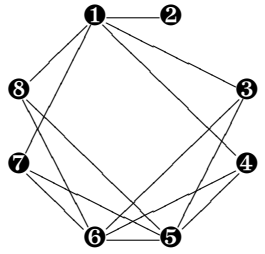


Figure A.30: Step 3 in constructing an $(8, 8)$ absorption set: Case II.A.

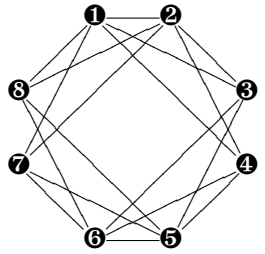


Figure A.31: Step 4 in constructing an $(8, 8)$ absorption set: Case II.A.

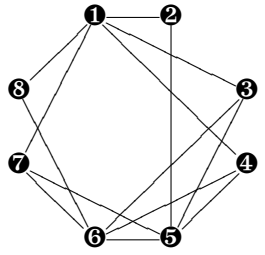


Figure A.32: Step 3 in constructing an $(8, 8)$ absorption set: Case II.B.

i. CaseII.B.1: if ❷ is connected to ❽, without loss of generality, assume it is connected to ❸ and ❹, as shown in Figure A.33.

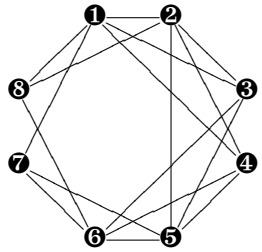


Figure A.33: Step 4 in constructing an $(8, 8)$ absorption set: Case II.B.1.

By connecting the remaining nodes, we obtain Figure A.35(c).

ii. CaseII.B.2: if ❷ is not connected to ❽, we must have the connectivity as shown in Figure A.34.

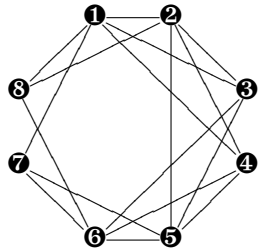
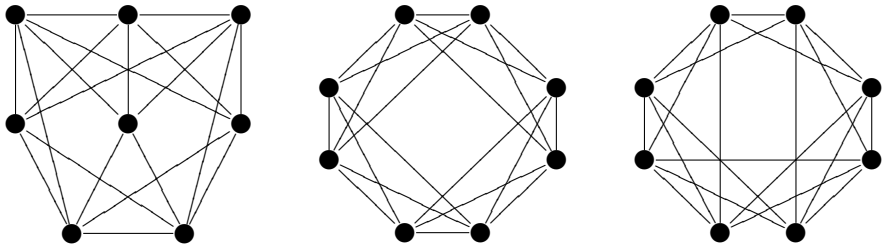


Figure A.34: Step 4 in constructing an $(8, 8)$ absorption set: Case II.B.2.

After connecting the remaining node, this gives us FigureA.35(a) again.

So eventually we obtain the other three possible $(8, 8)$ topologies in Figure A.35.



(a) Case I or Case II.B.2.

(b) Case II.A.

(c) Case II.B.1.

Figure A.35: Another three out of five possible topologies of $(8, 8)$ absorption sets.

Theorem A.10. *By checking the* $\mathbf{H}$ *matrix, the number of* $(8, 8)$ *absorption sets is* $14,272$ *and they all share one topology shown in Figure A.35(b).*

The ratio $\frac{b}{6a} = \frac{1}{6}$. Next possible absorption set with this ratio would be $(10, 10)$. In addition, the dominant eigenvalue of the $\mathbf{VC}$ matrix is 4 and the corresponding eigenvector is an all-1 vector.

In addition, from Figure A.35(b), we can get $(7, 12)$ absorption sets, by removing one node.

A.2.4.3 $b = 10$

The conductivities can be divided into four classes:

1. $[6, 6, 6, 4, 4, 4, 4, 4]$;
2. $[6, 6, 5, 5, 4, 4, 4, 4]$;
3. $[6, 5, 5, 5, 5, 4, 4, 4]$;
4. $[5, 5, 5, 5, 5, 5, 4, 4]$.

By searching the $\mathbf{H}$ matrix, none of them exist.

Note that enumerating all possible topologies like what we did in previous sections does not interest us any longer, since we have found the dominant $(8, 8)$ absorption set. Besides, the enumeration process gets extremely complicated as b and a grow.

A.2.4.4 $b = 12$

We simply skip topology construction and list the existence and multiplicity of the possible three classes below:

1. $[6, 6, 4, 4, 4, 4, 4, 4]$: $11,008$ such sets.
2. $[6, 5, 5, 4, 4, 4, 4, 4]$: none exists.
3. $[5, 5, 5, 5, 4, 4, 4, 4]$: $33,408$ under this class.

So there are $11,008 + 33,408 = 44,416$ $(8, 12)$ absorption sets in total. Some topologies are shown in Figure A.36.

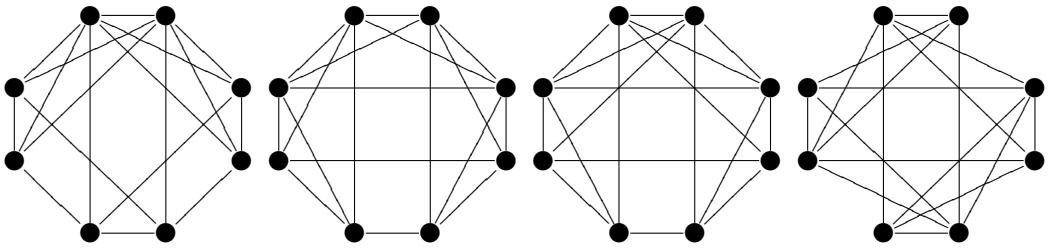


Figure A.36: Some topologies of $(8, 12)$ absorption sets.

A.2.4.5 $b = 14$

The $(8, 14)$ absorption set does exist. There are two classes and all $(8, 14)$ sets fall in the second class:

1. $[6, 4, 4, 4, 4, 4, 4, 4]$: none;
2. $[5, 5, 4, 4, 4, 4, 4, 4]$: there exist 88,896 sets of this class.

A.2.4.6 $b = 16$

There is only one feasible class $[4, 4, 4, 4, 4, 4, 4, 4]$. We start with the sketch shown in Figure A.37.

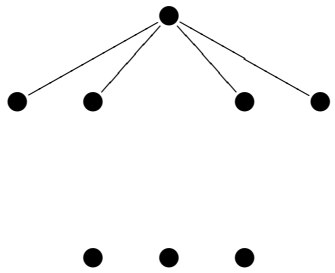


Figure A.37: Step 1 in constructing an $(8, 16)$ absorption set.

Dependent on the connectivity among the bottom three nodes, we have four cases.

1. Case I: there is no connection among them. Then there is only one way to connect all the nodes as shown in Figure A.38.

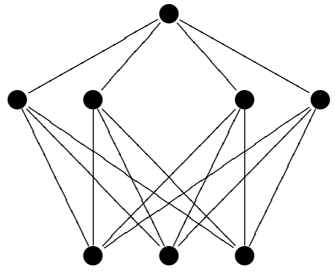


Figure A.38: First topology of $(8, 16)$ absorption sets: Case I.

   Its equivalent symmetric appearances are shown in Figure A.39.

2. Case II: there is only one connection among the three bottom nodes which implies that there is only one connection among the middle level four nodes, as shown in Figure A.40.

   Next, we connect the bottom-right node as shown in Figure A.41.

   Then the rest can be connected as Figure A.42.

   It has some symmetric forms, as well, as shown in Figure A.43.

3. Case III: there are only two connections among the three bottom nodes as shown in Figure A.44.

   In this case, it implies that there are only two connections among the middle level four nodes, which gives us two options.

   (a) Case III.A: the connection shown in Figure A.45.

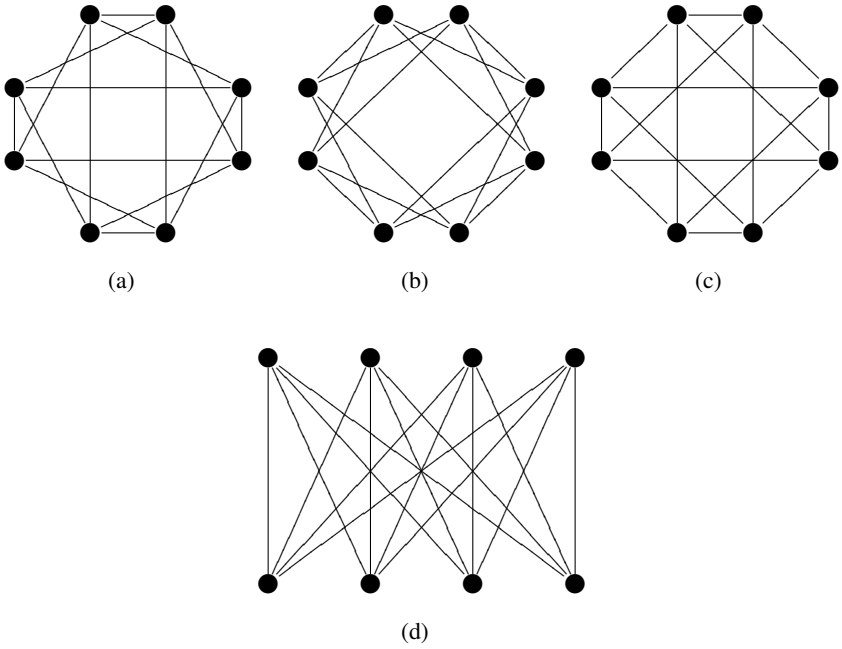


Figure A.39: Equivalent appearances of the first topology of $(8, 16)$ absorption sets: Case I.

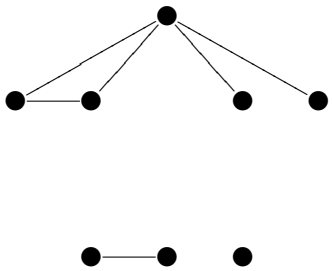


Figure A.40: Step 2 in constructing an $(8, 16)$ absorption set: Case II.

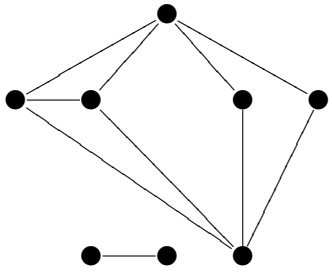


Figure A.41: Step 3 in constructing an $(8, 16)$ absorption set: Case II.

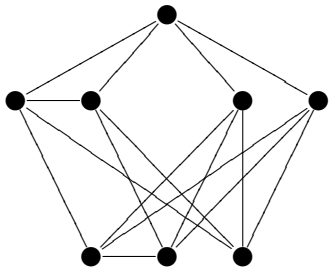


Figure A.42: Second topology of $(8, 16)$ absorption sets: Case II.

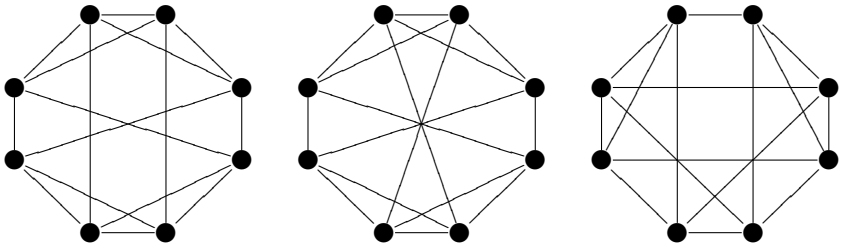


Figure A.43: Equivalent appearances of the second topology of $(8, 16)$ absorption sets: Case II.

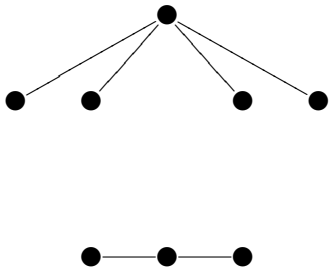


Figure A.44: Step 2 in constructing an $(8, 16)$ absorption set: Case III.

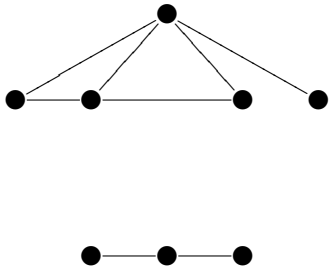


Figure A.45: Step 3 in constructing an $(8, 16)$ absorption set: Case III.A.

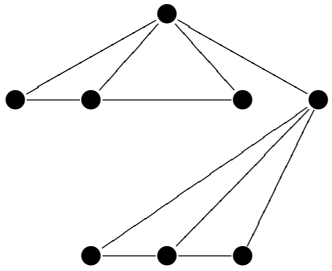


Figure A.46: Step 4 in constructing an $(8, 16)$ absorption set: Case III.A.

Next, we connect the rightmost node in the center row as Figure A.46.
Now the center node of the bottom three has two choices.

i. Case III.A.1: firstly as shown in Figure A.47.

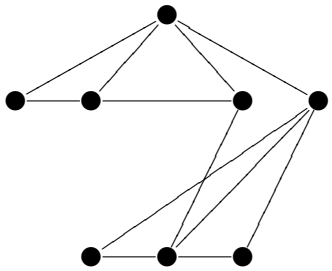


Figure A.47: Step 5 in constructing an $(8, 16)$ absorption set: Case III.A.1.

Then we connect more and obtain Figure A.48.

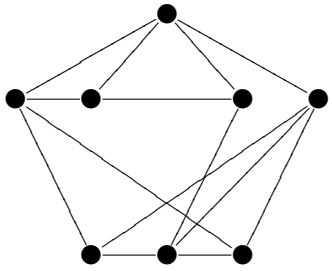


Figure A.48: Step 6 in constructing an $(8, 16)$ absorption set: Case III.A.1.

Finally, we find the third possible topology as shown in Figure A.49, since the bottom-left and bottom-right are symmetric.
We list one of its equivalent topologies in Figure A.50.

ii. Case III.A.2: another choice to connect Figure A.46 is shown in Figure A.51.

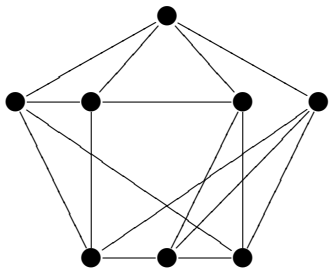


Figure A.49: Third topology of $(8, 16)$ absorption sets: Case III.A.1.

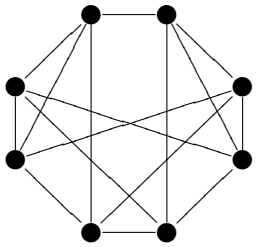


Figure A.50: Equivalent appearance of the third topology of $(8, 16)$ absorption sets: Case III.A.1.

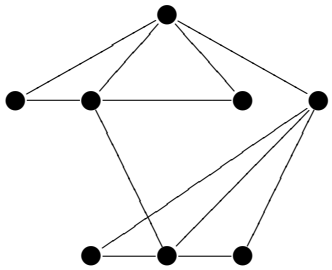


Figure A.51: Step 5 in constructing an $(8, 16)$ absorption set: Case III.A.2.

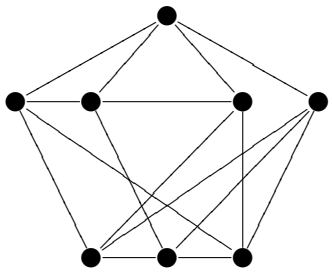


Figure A.52: Another topology of $(8, 16)$ absorption sets: Case III.A.2.

By connecting the rest, this gives us Figure A.52.
Actually, it can be arranged to the ones shown in Figure A.43.

(b) Case III.B: Figure A.44 can also be connected as Figure A.53.

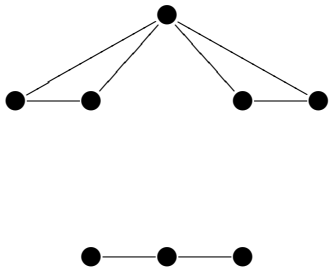


Figure A.53: Step 3 in constructing an $(8, 16)$ absorption set: Case III.B.

Next we have Figure A.54.

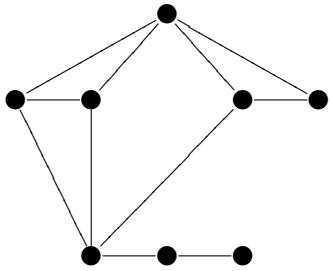


Figure A.54: Step 4 in constructing an $(8, 16)$ absorption set: Case III.B.

Then we connect more in Figure A.55.

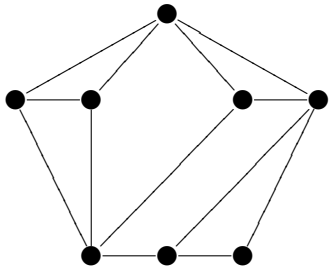


Figure A.55: Step 4 in constructing an $(8, 16)$ absorption set: Case III.B.

The center node of the bottom three has two choices.

i. Case III.B.1: firstly, it can be as Figure A.56.
So we connect the remaining nodes and get Figure A.57.
This is another appearance of Figure A.50.

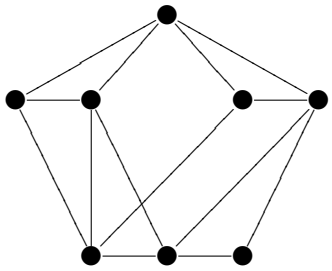


Figure A.56: Step 4 in constructing an $(8, 16)$ absorption set: Case III.B.1.

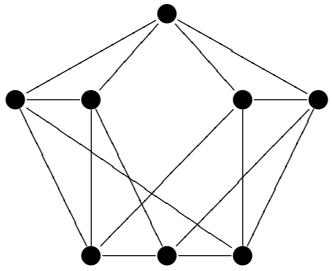


Figure A.57: A topology of $(8, 16)$ absorption sets: Case III.B.1.

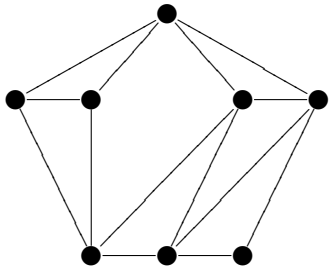


Figure A.58: Step 4 in constructing an $(8, 16)$ absorption set: Case III.B.

ii. Case III.B.2: secondly, Figure A.55 can be connected as Figure A.58.
Then we have another topology shown in Figure A.59.

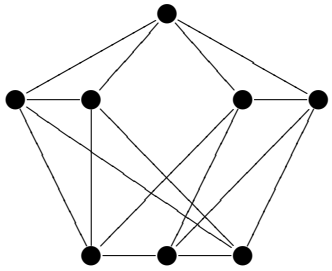


Figure A.59: Another topology of $(8, 16)$ absorption sets: Case III.B.2.

Equivalently, it can be like Figure A.60.

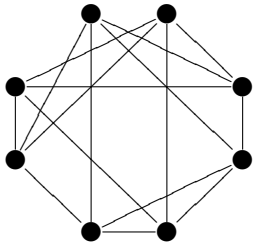


Figure A.60: An equivalent appearance of the topology of the $(8, 16)$ absorption set: Case III.B.2.

4. Case IV: there are three connections among the three bottom nodes of Figure A.37, as shown in Figure A.61.

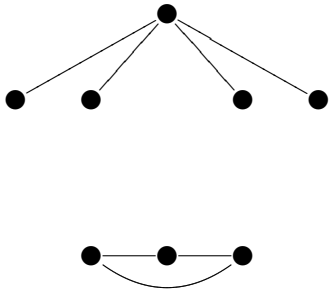


Figure A.61: Step 2 in constructing an $(8, 16)$ absorption set: Case IV.

In this case, it implies that there are only three connections among the middle-row four nodes, which gives us two options.

(a) Case IV.A: first, it is connected as Figure A.62.
Next we obtain Figure A.63.
The second node of the middle level four has two choices.

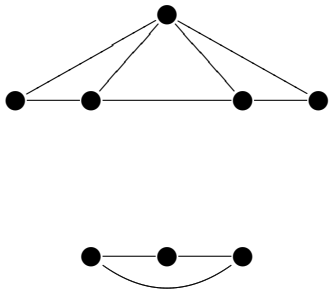


Figure A.62: Step 3 in constructing an $(8, 16)$ absorption set: Case IV.A.

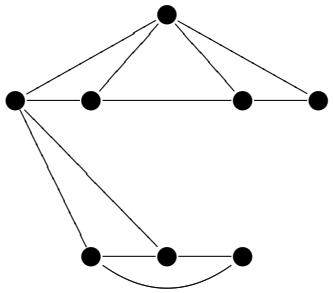


Figure A.63: Step 4 in constructing an $(8, 16)$ absorption set: Case IV.A.

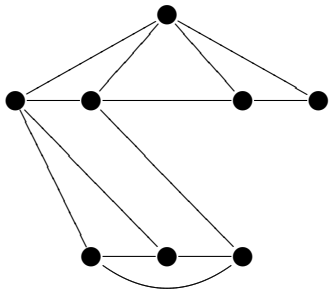


Figure A.64: Step 5 in constructing an $(8, 16)$ absorption set: Case IV.A.1.

i. Case IV.A.1: as shown in Figure A.64.
Now, the third node of the middle level four has two choices.

A. Case IV.A.1.i: first choice is as Figure A.65.

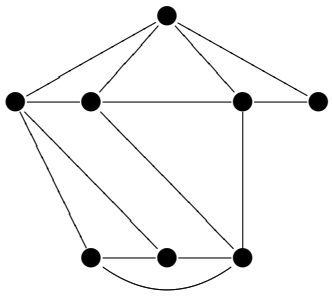


Figure A.65: Step 6 in constructing an $(8, 16)$ absorption set: Case IV.A.1.i.

Hence, we get to another topology shown in Figure A.66.

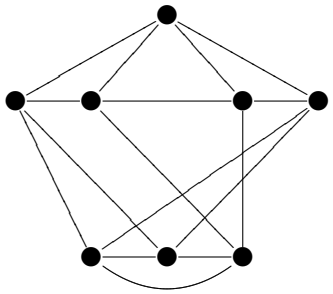


Figure A.66: A topology of $(8, 16)$ absorption sets: Case IV.A.1.i.

This is identical to Figure A.60.

B. Case IV.A.1.ii: another choice is like Figure A.67.

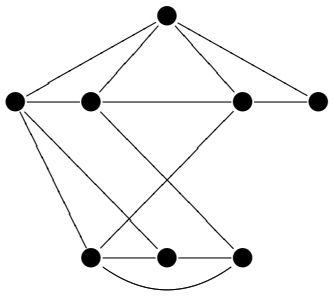


Figure A.67: Step 6 in constructing an $(8, 16)$ absorption set: Case IV.A.1.ii.

Hence, the topology under this case can be achieved in Figure A.68.
This is also the same as Figure A.50.

ii. Case IV.A.2: a case as shown in Figure A.69.
Next, more connections are shown in Figure A.70.

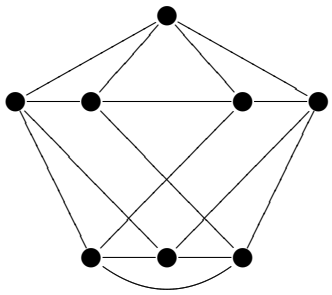


Figure A.68: a topology of the $(8, 16)$ absorption set: Case IV.A.1.ii.

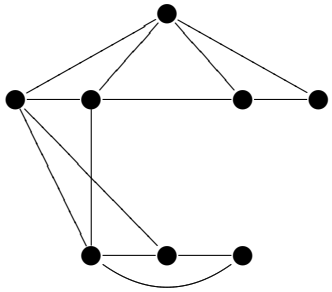


Figure A.69: Step 5 in constructing an $(8, 16)$ absorption set: Case IV.A.2.

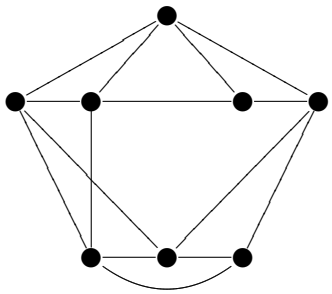


Figure A.70: Step 6 in constructing an $(8, 16)$ absorption set: Case IV.A.2.

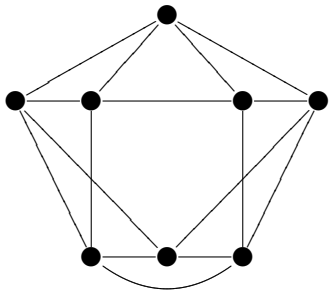


Figure A.71: A topology of $(8, 16)$ absorption sets: Case IV.A.2.

Hence, connecting the rest gives us Figure A.71.
It has a nicer symmetric look shown in Figure A.72.

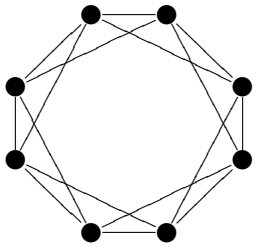


Figure A.72: An equivalent appearance of the $(8, 16)$ absorption set: Case IV.A.2.

(b) Case IV.B: Figure A.61 can also be connected as Figure A.73.

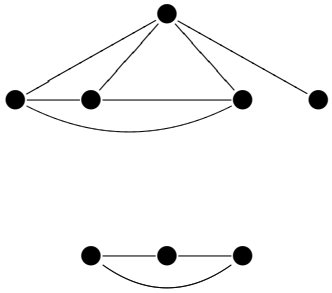


Figure A.73: Step 3 in constructing an $(8, 16)$ absorption set: Case IV.B.

Next, we connect more nodes as Figure A.74.

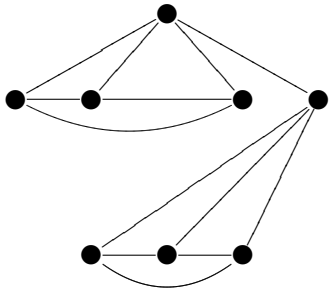


Figure A.74: Step 4 in constructing an $(8, 16)$ absorption set: Case IV.B.

Then we achieve the last possible sketch in Figure A.75.
Equivalently, we rearrange it to Figure A.76.

To summarize, we have narrowed down to (at most) six topologies, as shown in Figure A.77, to search against the $\mathbf{H}$.

By searching them algorithmically, we list the multiplicity of the $(8, 16)$ absorption sets per topology below.

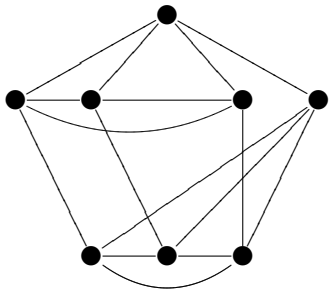


Figure A.75: A topology of $(8, 16)$ absorption sets: Case IV.B.

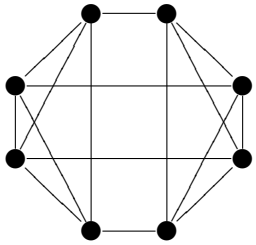
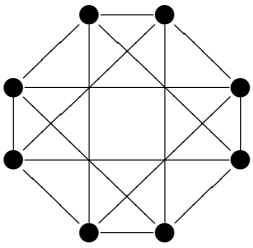
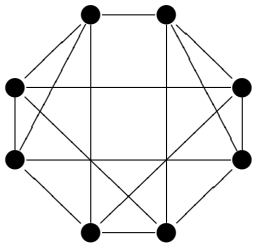


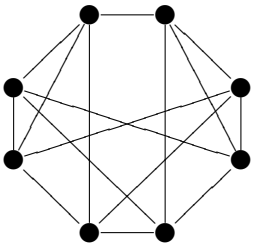
Figure A.76: An equivalent appearance of the $(8, 16)$ absorption set: Case IV.B.



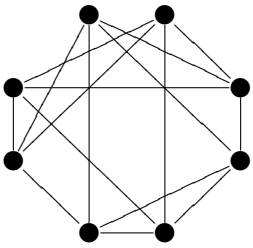
(a) Case I.



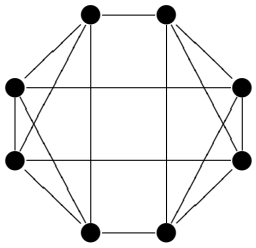
(b) Case II or Case III.A.2.



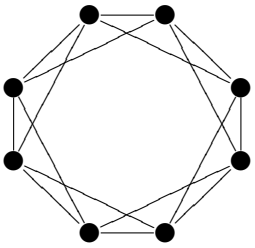
(c) Case III.A.1 or Case III.B.1 or Case IV.A.1.ii.



(d) Case III.B.2 or Case IV.A.1.i.



(e) Case IV.B.



(f) Case IV.A.2.

Figure A.77: Possible topologies of $(8, 16)$ absorption sets.

1. Figure A.77(a): $156,064$;
2. Figures A.77(b) and A.77(c): $58,944$;
3. Figures A.77(d) and A.77(f): $86,272$;
4. Figure A.77(e): $360,544$;

which add up to $661,824$.

A.2.5 $a = 9$

We only try to enumerate the topologies that may have larger $\mu_{\max}$ which affects the error estimation formula, instead of going through all possibilities like what we have done for smaller a, which is not practical any more.

First, we assume only double check nodes connections are allowed. The more complicated cases are studied afterwards.

1. $b = 0$. By Corollary A.8, weight-9 codeword does not exist.
2. $b = 2$. There are two classes:
   (a) $[6, 6, 6, 6, 6, 6, 6, 5, 5]$: removing either degree-5 node will reduce it to an $(8, 6)$ absorption set.
   (b) $[6, 6, 6, 6, 6, 6, 6, 6, 4]$: removing the degree-$4$ node will reduce it to an $(8, 4)$ absorption set.
3. $b = 4$. Three classes:
   (a) $[6, 6, 6, 6, 6, 5, 5, 5, 5]$: removing any degree-$5$ node will reduce it to an $(8, 8)$ absorption set. Since we know that only one type of $(8, 8)$ sets exist, then we know that there is only one possible topology for this class of $(9, 4)$ sets as shown in Figure A.78.

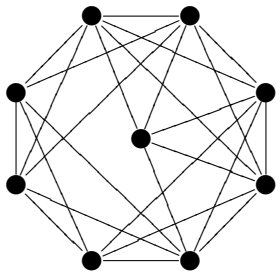


Figure A.78: A possible topology of the $(9, 4)$ absorption sets.

   Algorithmically, we pick a node randomly and check if it is connected with any 5 nodes from $(8, 8)$ sets. We found none that exists.
   (b) $[6, 6, 6, 6, 6, 6, 5, 5, 4]$: removing the degree-$4$ node will reduce it to an $(8, 6)$ absorption set.
   (c) $[6, 6, 6, 6, 6, 6, 6, 4, 4]$: removing both degree-$4$ nodes will reduce it to a $(7, 6)$ or $(7, 8)$ absorption set.

4. $b = 6$. There are four classes:

   (a) $[6, 6, 6, 5, 5, 5, 5, 5, 5]$: removing any degree-5 node will reduce it to an $(8, 10)$ absorption set.

   (b) $[6, 6, 6, 6, 5, 5, 5, 5, 4]$: removing the degree-4 node will reduce it to an $(8, 8)$ absorption set. Since we know that only one type of $(8, 8)$ sets exist, then we know that there is only one possible topology for this $(9, 6)$ class as shown in Figure A.79.

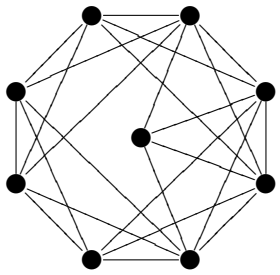


Figure A.79: A possible topology of the $(9, 6)$ absorption sets.

   We pick a node randomly and check if it is connected with any 4 nodes from $(8, 8)$ sets. None of such connectivity exists.

   (c) $[6, 6, 6, 6, 6, 5, 5, 4, 4]$: if the two degree-4 nodes are

      i. Not connected: removing one degree-4 node reduces it to an $(8, 8)$ set with a degree-4 node. Such class of $(8, 8)$ sets does not exist.

      ii. Connected: then let us involve the degree-5 nodes as follows.

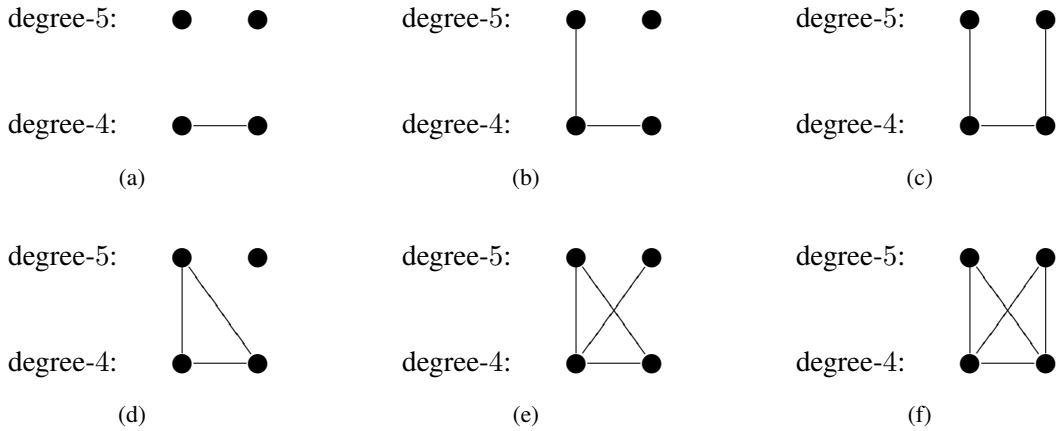


Figure A.80: Possible intrinsic connections between two groups of variable nodes which are grouped by degrees. The top row represents the degree-5 nodes, whereas the bottom degree-4.

      A. No connection between the two groups as shown in Figure A.80(a). Removing both degree-4 nodes gives a $(7, 8)$ set, or removing one degree-5 node gives a $(8, 10)$ set.

B. One connection between the two groups as shown in Figure A.80(b). Removing both degree-4 nodes gives a $(7, 8)$ set, or removing the top right degree-5 node gives an $(8, 10)$ set.

C. Two connections between the two groups, firstly as shown in Figure A.80(c). Removing both degree-4 nodes gives a $(7, 8)$ set. Or as shown in Figure A.80(d). Removing the top right degree-5 node gives an $(8, 10)$ set.

D. Three connections between the two groups as shown in Figure A.80(e). Each of the degree-6 nodes requires at least two connections from the rest, whereas the above topology can provide at most 10. Thus removing the degree-4 and the left degree-5 nodes reduces it to a $(6, 8)$ set.

E. Four connections between the two groups as shown in Figure A.80(f). This is infeasible since it can provide at most 8 connections to the group of degree-6 nodes.

(d) $[6, 6, 6, 6, 6, 6, 4, 4, 4]$: let study internal connectivity within degree-4 nodes.

i. No connection as shown in Figure A.4(a): removing one degree-4 node reduces it to an $(8, 8)$ set with class $[6, 6, 5, 5, 5, 5, 4, 4]$.

ii. One connection as shown in Figure A.4(b): removing the top degree-4 node that is not connected reduces it to an $(8, 8)$ set with class $[6, 6, 5, 5, 5, 5, 4, 4]$.

iii. Two connections as shown in Figure A.4(c): each of the degree-6 nodes requires one, or two connections from the group of degree-4 nodes. In other words, no degree-6 node can be connected to all degree-4 nodes. Thus, removing all degree-4 nodes gives a $(6, 8)$ set.

iv. Three connections as shown in Figure A.4(d): by the same argument in the above item, removing all degree-4 nodes gives a $(6, 6)$ set.

5. $b = 8$. There are five classes.

(a) $[6, 5, 5, 5, 5, 5, 5, 5, 5]$;

(b) $[6, 6, 5, 5, 5, 5, 5, 5, 4]$: removing the degree-4 node reduces it to an $(8, 10)$ set.

(c) $[6, 6, 6, 5, 5, 5, 5, 4, 4]$;

(d) $[6, 6, 6, 6, 5, 5, 4, 4, 4]$;

(e) $[6, 6, 6, 6, 6, 4, 4, 4, 4]$.

However, since we cannot rule out enough cases by connecting them to known absorption sets, actually running a general program to check the possibility containing a degree-6 node is more effective, although brutal. The search program found no $(9, 8)$ sets containing a degree-6 node.

6. $b = 10$. There are five classes.

(a) $[5, 5, 5, 5, 5, 5, 5, 5, 5, 4]$;

(b) $[6, 5, 5, 5, 5, 5, 5, 5, 4, 4]$;

(c) $[6, 6, 5, 5, 5, 5, 4, 4, 4]$;

(d) $[6, 6, 6, 5, 5, 4, 4, 4, 4]$;

(e) $[6, 6, 6, 6, 4, 4, 4, 4, 4]$.

By removing the single degree-4 node in first class reduces it to an $(8, 12)$ set. By checking against the enumerated $(8, 12)$ sets, such connectivity does not exist. Regarding the remaining classes, once again, a general program has found no $(9, 10)$ sets containing a degree-6 node.

7. $b = 12$. There are four classes.

   (a) $[5, 5, 5, 5, 5, 5, 4, 4, 4]$;

   (b) $[6, 5, 5, 5, 5, 4, 4, 4, 4]$;

   (c) $[6, 6, 5, 5, 4, 4, 4, 4, 4]$;

   (d) $[6, 6, 6, 4, 4, 4, 4, 4, 4]$.

   One $(9, 12)$ absorption set is shown in Table A.3 as evidence of the existence.

Table A.3: An example $(9, 12)$ absorption set that falls in the first class.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
1563	29	120	150	217	264	336
1628	40	75	171	248	264	373
176	43	78	174	251	267	376
314	8	125	150	251	300	368
560	40	78	177	199	313	368
1258	29	79	137	213	313	370
1988	30	125	171	213	267	336

8. $b = 14$. There are three classes.

   (a) $[5, 5, 5, 5, 4, 4, 4, 4, 4]$: its existence can be seen by removing one node from the $(10, 10)$ absorption sets found in the following section.

   (b) $[6, 5, 5, 4, 4, 4, 4, 4, 4]$;

   (c) $[6, 6, 4, 4, 4, 4, 4, 4, 4]$.

9. $b = 16$. There are two classes.

   (a) $[5, 5, 4, 4, 4, 4, 4, 4, 4]$;

   (b) $[6, 4, 4, 4, 4, 4, 4, 4, 4]$.

   An example is shown in Table A.4.

10. $b = 18$. There is only one class, $[4, 4, 4, 4, 4, 4, 4, 4, 4]$, and one such set is shown in Table A.5.

Table A.4: An example $(9, 16)$ absorption set that falls in the first class.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
1460	2	79	184	238	307	365
580	7	102	148	253	307	376
890	15	87	181	229	261	349
1194	38	87	148	228	319	360
1775	2	126	176	228	261	338
1881	33	84	176	229	300	360

Table A.5: An example $(9, 18)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
629	15	86	161	248	268	381
1063	1	109	178	236	312	349
504	20	85	174	197	258	338
802	49	86	154	197	300	363
1299	6	124	178	247	305	381
1789	20	109	150	247	265	363

Now, if the satisfied check nodes are allowed to be connected to the set more than twice, we are able to enumerate all possible topologies but cannot prove the existence for all of them by simply removing nodes and relating them to the known smaller sets.

1. If the check nodes are connected to the set 6 times, then there is only one possible topology which gives a $(9, 12)$ absorption set as shown in Figure A.81.

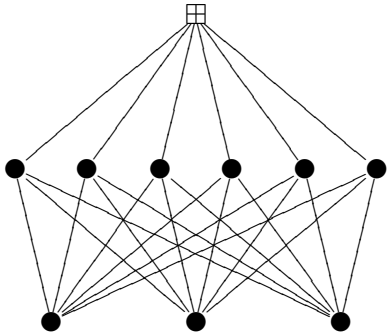


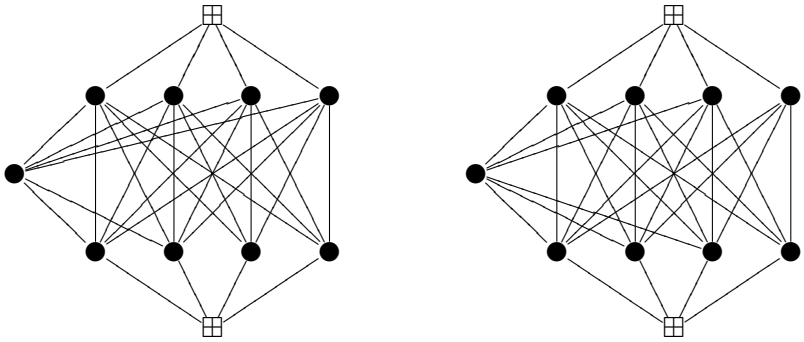
Figure A.81: One check node connecting to a $(9, 12)$ absorption set six times.

It falls in the class $[6, 6, 6, 4, 4, 4, 4, 4, 4]$. Removing any two degree-4 nodes will give us a $(7, 14)$ set.

2. There can be up to two check nodes that are connected 4 times to the set.

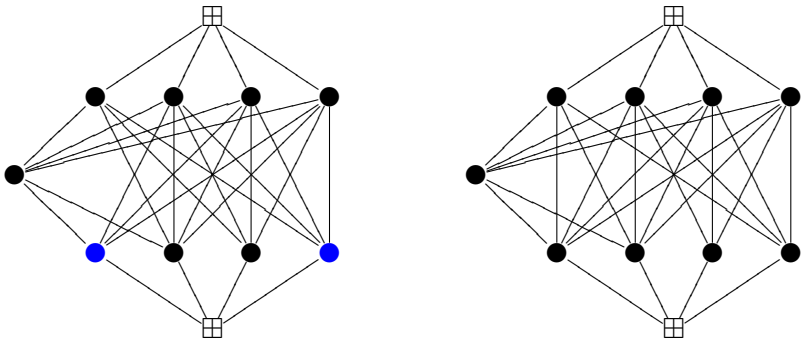
   (a) When there are two such check nodes, they can either share one variable node or not.

      i. First, we sketch the topology that the two check nodes do not share any variable node.
      The smallest would be $(9, 2)$, as shown in Figure A.82, both of which can be reduced to the un-existed $(8, 8)$ sets.



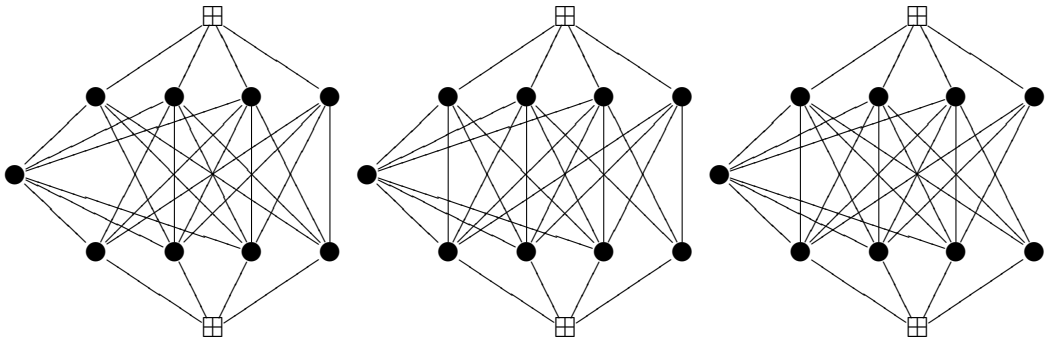
(a) (b)

Figure A.82: Two possible topologies of $(9, 2)$ absorption sets with two check nodes connected to the set four times.



(a) (b)

Figure A.83: Two possible topologies of $(9, 4)$ absorption sets with two check nodes connected to the set four times that are derived from Figure A.82(a).

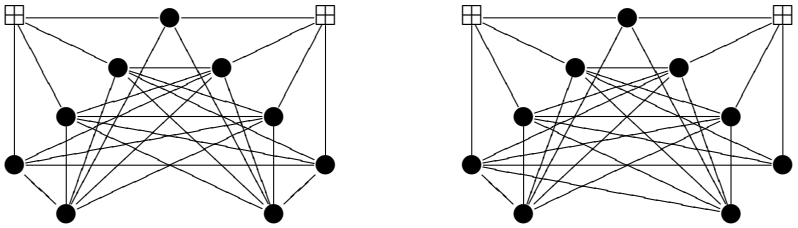


(a) (b) (c)

Figure A.84: Another three possible topologies of $(9, 4)$ absorption sets with two check nodes connected to the set four times that are derived from Figure A.82(b).

Such $(9, 4)$ and $(9, 6)$ sets can be derived by removing one or two edges from them, respectively, as illustrated in Figure A.83 and Figure A.84.
Removing the two blue degree-5 nodes in Figure A.83(a) gives us a $(7, 10)$ set. Removing the degree-4 node in Figure A.83(b) gives an un-existed $(8, 8)$ class. Removing all the degree-5 nodes in Figure A.84(a) gives a $(5, 10)$ set. Removing the degree-4 node in Figure A.84(b) gives an un-existed $(8, 8)$ class. Removing either degree-4 node in Figure A.84(c) gives an un-existed $(8, 8)$ class.

ii. Secondly, if the two check nodes are both connected to one variable node as shown in Figure A.85.



(a) (b)

Figure A.85: Two possible topologies of $(9, 4)$ absorption sets with two check nodes connected to the set four times and sharing one variable node.

Removing the degree-4 node in Figure A.85(a) gives us a $(7, 10)$ set. Removing the two degree-4 nodes in Figure A.85(b) gives us a $(7, 8)$ set.
Also we could derive $(9, 6)$, $(9, 8)$, $(9, 10)$, $(9, 12)$ and $(9, 14)$ from them by removing edge(s). A few example of $(9, 6)$ sets are shown in Figure A.86.

(b) If one check node is connected to the set four times, the smallest possible connectivity results in a weight-9 codeword as shown in Figure A.87 which cannot exist.
Then the $(9, 2)$, $(9, 4)$, $(9, 6)$, $(9, 8)$, $(9, 10)$, $(9, 12)$, $(9, 14)$, $(9, 16)$ and $(9, 18)$ absorption sets can be derived by removing edge(s). We can show that such $(9, 2)$ and $(9, 4)$ absorption sets do not exist.

i. There are two possible $(9, 2)$ set topologies, as shown in Figure A.88.
Removing the bottom-left degree-5 node in Figure A.88(a) gives an $(8, 6)$ set. Removing either degree-5 node in Figure A.88(b) gives an $(8, 6)$ set, as well.

ii. $(9, 4)$ sets are shown in Figure A.89 and Figure A.90.
Removing either degree-5 node in Figure A.89(a) gives an un-existed $(8, 8)$ class. Removing the degree-4 node in Figure A.89(b) gives an $(8, 6)$ set. Removing either bottom degree-5 node in Figure A.89(c) gives an un-existed $(8, 8)$ class. Removing the degree-4 node in Figure A.89(d) gives an $(8, 6)$ set. Removing any bottom degree-5 node in Figure A.89(e) gives an un-existed $(8, 8)$ class.
Removing the degree-4 node in Figure A.90(a) gives an $(8, 6)$ set. Removing any bottom degree-5 node in Figure A.90(b) gives an un-existed $(8, 8)$

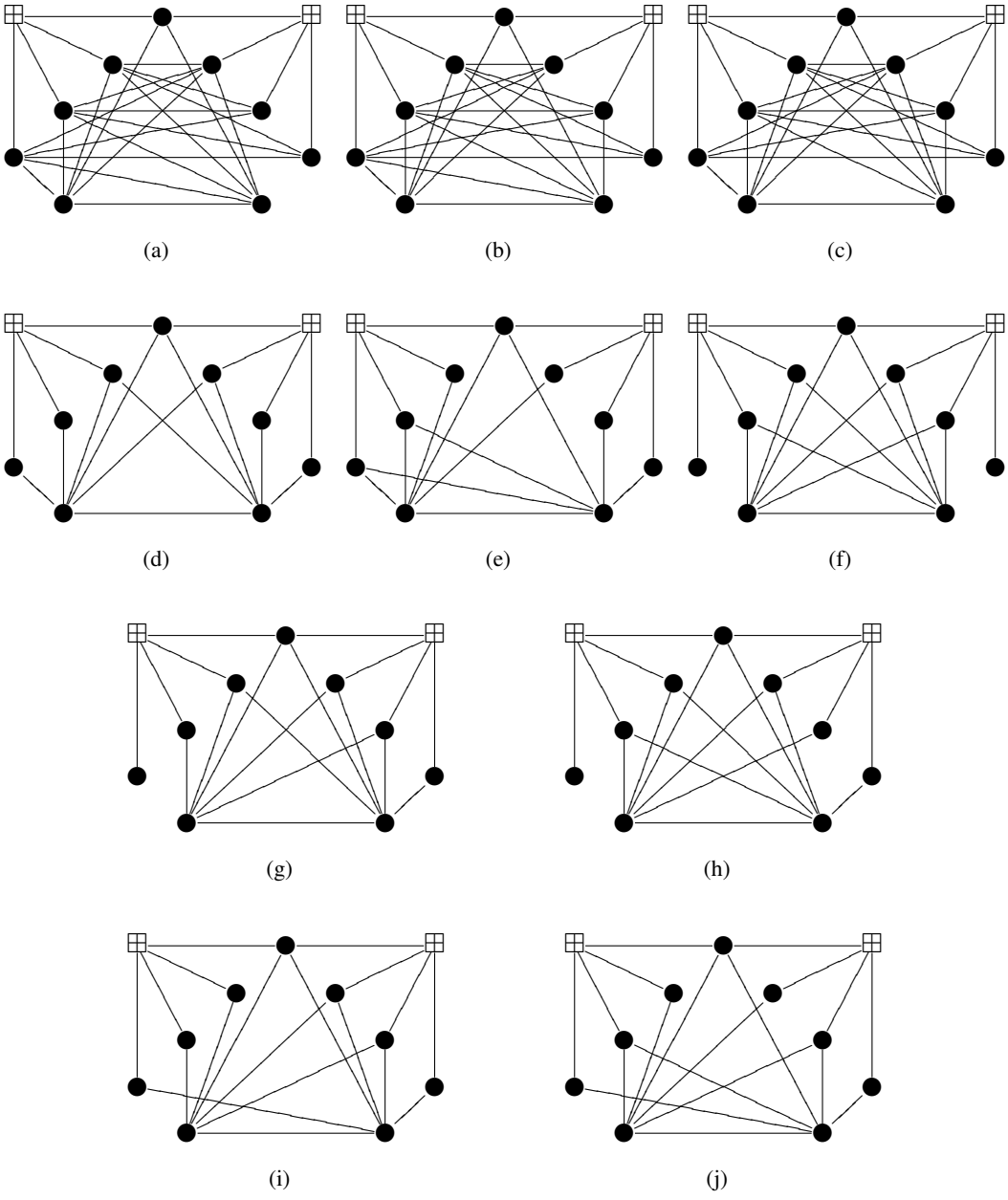


Figure A.86: A few possible topologies of $(9, 6)$ absorption sets with two check nodes connected to the set four times and sharing one variable node.

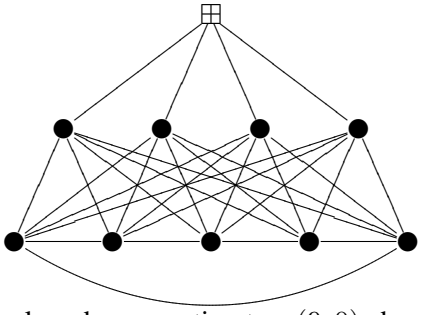
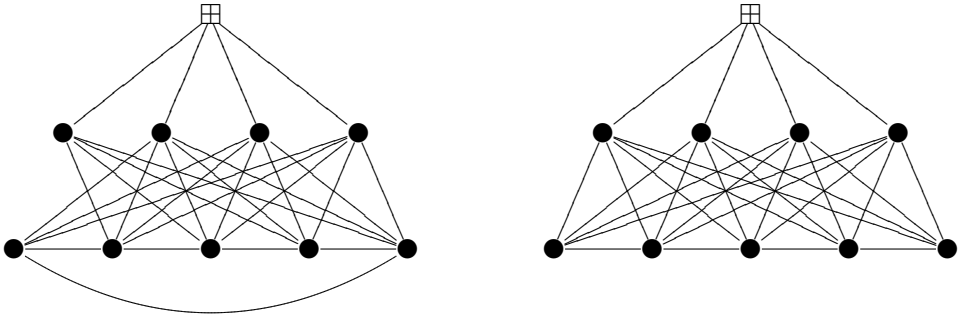


Figure A.87: One check node connecting to a $(9, 0)$ absorption set four times.



(a) (b)

Figure A.88: Two possible topologies of $(9, 2)$ absorption sets with one check node connected to the set four times.

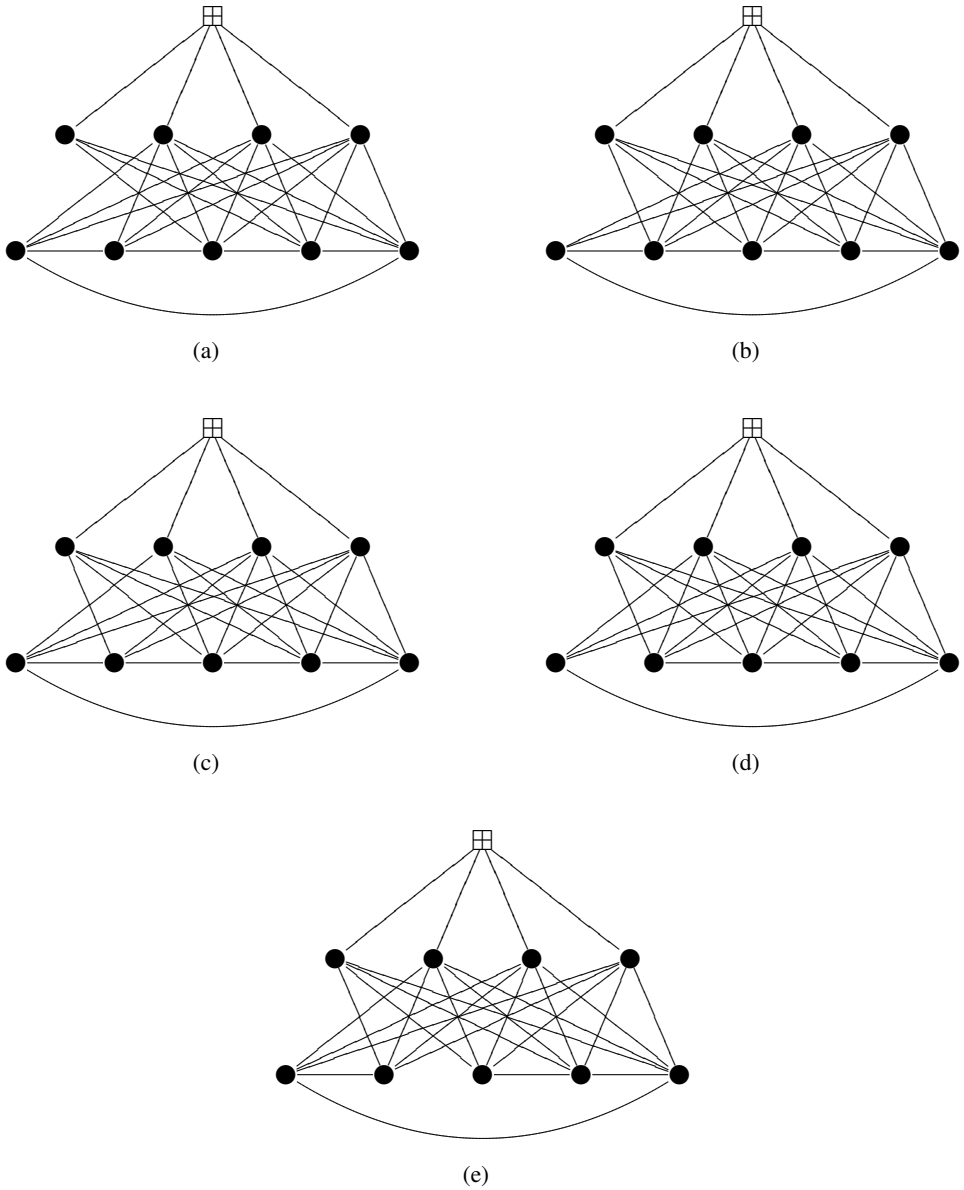


Figure A.89: Five possible topologies of $(9, 4)$ absorption sets with one check node connected to the set four times that are derived from Figure A.88(a).

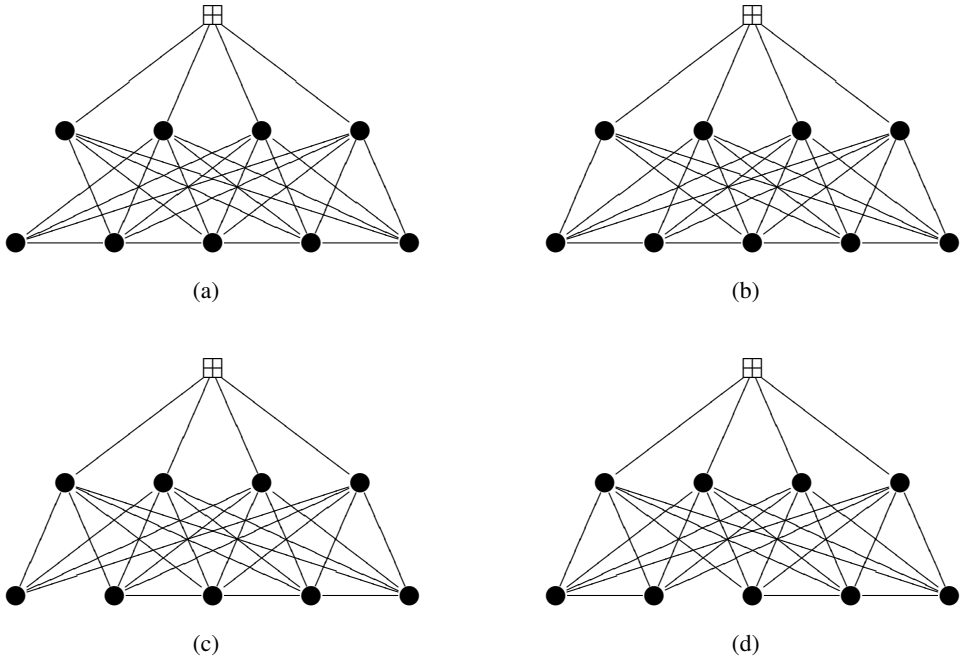


Figure A.90: Another four possible topologies of $(9, 4)$ absorption sets with one check node connected to the set four times that are derived from Figure A.88(b).

class. Removing the degree-4 node in Figure A.90(c) gives an $(8, 6)$ set. Removing any degree-5 node in Figure A.90(d) gives an un-existed $(8, 8)$ class.

A.2.6 $a = 10$

We do not find out the multiplicity of any size-10 absorption sets as it is costly time-wise, but rather trying to figure out the existence of sets with large $\mu_{\max}$ or small b/a ratio. In addition, we only assume that the satisfied check nodes are connected to the set only twice, in this section.

A.2.6.1 $b = 0$

Since a $(10, 0)$ absorption set can be reduced to a $(9, 6)$ set by removing any degree-6 variable node, we are able to further tighten up the lower bound on the minimum distance.

Corollary A.11. *Because there is no weight-*10 *codeword, the minimum distance is bounded by* $d_{\min} \geq 12$.

Note that a weight-14 codeword, as shown in Table A.6[4], has been found during our simulation. Therefore we narrow the minimum distance down to $d_{\min} = 12$ or 14. The authors of [1] have also found $(14, 0)$ absorption sets, and the multiplicity is lower bounded by $1,407$, through an absorption set searching algorithm.

[4]The indexes are according to v2c.txt.

Table A.6: A weight-14 codeword.

Variable Node	Six Neighboring Check Nodes					
131	10	80	150	234	290	329
148	55	112	179	226	298	327
282	33	118	137	199	298	383
315	55	115	162	234	263	379
375	45	122	164	218	287	345
378	42	92	150	226	263	383
833	23	118	164	206	260	329
871	42	66	140	218	262	355
1028	58	112	137	238	287	355
1039	45	115	185	206	262	364
1174	33	66	185	224	290	379
1182	23	122	140	238	312	327
1228	58	80	162	224	260	345
1270	10	92	179	199	312	364

A.2.6.2 $b = 2$

There exist two classes.

1. $[6, 6, 6, 6, 6, 6, 6, 6, 5, 5]$: it can be reduced to a $(9, 6)$ set by removing one degree-5 node.

2. $[6, 6, 6, 6, 6, 6, 6, 6, 6, 4]$: it can be reduced to a $(9, 4)$ set by removing the degree-4 node.

A.2.6.3 $b = 4$

There exist three classes.

1. $[6, 6, 6, 6, 6, 6, 5, 5, 5, 5]$: it can be reduced to a $(9, 8)$ set by removing one degree-5 node.

2. $[6, 6, 6, 6, 6, 6, 6, 5, 5, 4]$: it can be reduced to a $(9, 6)$ set by removing the degree-4 node.

3. $[6, 6, 6, 6, 6, 6, 6, 6, 4, 4]$: it can be reduced to a $(9, 6)$ set by removing one of the degree-4 nodes, if there is no connection between them. Otherwise, removing both gives an $(8, 6)$ set.

A.2.6.4 $b = 6$

There exist four classes.

1. $[6, 6, 6, 6, 5, 5, 5, 5, 5, 5]$: it can be reduced to a $(9, 10)$ set by removing one degree-5 node.

2. $[6,6,6,6,6,5,5,5,5,4]$: it can be reduced to a $(9,8)$ set by removing the degree-4 node.

3. $[6,6,6,6,6,6,5,5,4,4]$: again, let us divide the nodes into two groups: one of degree-6 nodes and another of the rest. Then we study the connections within the second group.

   (a) If there is no connection between the degree-4 nodes, then it can be reduced to a $(9,8)$ set by removing either degree-4 node.

   (b) If the degree-4 nodes are incident, then it becomes tricky. Let us go through all possible connections.

      i. If the degree-4 nodes are not connected with the degree-5 nodes at all, then it can be reduced to a $(9,10)$ set by removing either degree-5 node.

      ii. If the degree-4 nodes are connected with each degree-5 node at most once as shown in Figures A.80(b) and A.80(c), then they can be reduced to an $(8,8)$ set where there is at least one degree-4, by removing both degree-4 nodes, respectively.

      iii. If the degree-4 nodes are connected with each degree-5 node twice, then we have to take into account the connection between the degree-5 nodes.

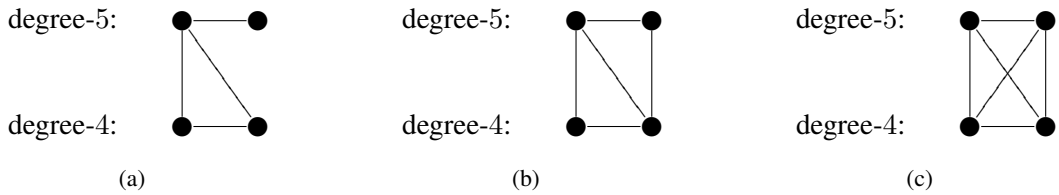


Figure A.91: Possible intrinsic connections between two groups of variable nodes which are grouped by degrees. The top row represents the degree-5 nodes, whereas the bottom degree-4. (Supplementary to Figure A.80.)

A. The topologies shown in Figures A.80(d) and A.91(a) can be reduced to a $(9,10)$ set by removing the top-right degree-5 node, respectively.

B. First, the topology shown in Figure A.80(e) induces a tricker situation. We intend to remove all but the top-right node. If those three are not connected to one degree-6 node from the first group, then it will reduce to a $(7,8)$ set. Otherwise, we have to eliminate that degree-6 node, which is for sure not incident to the top-right degree-5 node and gives us a $(6,8)$ set.
   Secondly, the topology in Figure A.91(b) shows that the second group emanates eight edges. The structure guarantees that removing all of them reduces to a $(6,8)$ set.

C. The group of degree-6 nodes emanates at least six edges, whereas the second group emanates eight or six, respectively. Either case of Figures A.80(f) and A.91(c), it is legitimate to remove the entire second group to get a $(6,8)$ and a $(6,6)$ set, respectively.

4. $[6, 6, 6, 6, 6, 6, 6, 4, 4, 4]$: in terms of degree, the nodes can be divided into two groups. Let us study the intrinsic connection within the degree-4 group.

   (a) If there is no connection as shown in Figure A.4(a), then removing any degree-4 node will reduce it to a $(9, 8)$ absorption set.

   (b) If there is only one connection as shown in Figure A.4(b), then removing the topmost degree-4 node will reduce it to a $(9, 8)$ set.

   (c) If there are two connections as shown in Figure A.4(c), then we want to remove all these three nodes. It would reduce to a $(7, 8)$ absorption set only if none of them are all incident to a single degree-6 node. Otherwise, let us remove that degree-6 node, as well, to get a $(6, 8)$ set. The structure guarantees that it is legal, because the degree-4 nodes now have to be connected with the three degree-6 nodes, respectively, which are not incident to the removed degree-6 node. In addition, the remaining two edges from the bottom two degree-4 nodes cannot be incident to only one of the degree-6 nodes. So removing those fours nodes would result a valid absorption set by definition.

   (d) If all three nodes are connected as shown in Figure A.4(d), then we, once again, try to remove all these three nodes. It would reduce to a $(7, 6)$ set, if none of them are all incident to one degree-6 node. Otherwise, let us also remove that degree-6 node, to get a $(6, 6)$ set. The structure guarantees that it is legal, because the degree-4 nodes now have to be connected with the three degree-6 nodes, respectively, which are not incident to the removed degree-6 node.

A.2.6.5 $b = 8$

The $(10, 8)$ absorption set enumeration is inconclusive. There exist five classes and two of them have been studied.

1. $[6, 6, 5, 5, 5, 5, 5, 5, 5, 5]$;

2. $[6, 6, 6, 5, 5, 5, 5, 5, 5, 4]$: it can be reduced to a $(9, 10)$ set by removing the degree-4 node.

3. $[6, 6, 6, 6, 5, 5, 5, 5, 4, 4]$;

4. $[6, 6, 6, 6, 6, 5, 5, 4, 4, 4]$;

5. $[6, 6, 6, 6, 6, 6, 4, 4, 4, 4]$: let us study the connection within the degree-4 group.

   (a) If there is no connection among the degree-4 nodes at all as shown in Figure A.5(a), then it can be reduced to a $(9, 10)$ set by removing any degree-4 node.

   (b) If there is one edge as shown in Figure A.5(b), then it can be reduced to a $(9, 10)$ set by removing either of the top-row degree-4 nodes.

   (c) If there are two connections, the one shown in Figure A.5(c) can be reduced to a $(9, 10)$ set by removing the bottom-right degree-4 node, whereas another one shown Figure A.5(d) can be reduced to an $(8, 10)$ set by removing either row of degree-4 nodes.

(d) When there are three connections, Figure A.5(e) can be reduced to a $(9, 10)$ set by removing the bottom-right degree-4 node. As of the other two shown in Figures A.5(f) and A.5(g), we will have to remove the entire group. Note that the degree-4 group emanates ten edges to the degree-6 group. If none of the degree-6 nodes connect to the degree-4 group more than twice, three times to be exact, then removing all degree-4 nodes gives a $(6, 10)$ set. Or, if there exists one degree-6 node that is incident to three degree-4 nodes, then remove this node as well as all degree-4 nodes. Each of the remaining nodes connects to the removed ones exactly twice. Thus it reduces to a $(5, 10)$ set.

(e) If there are four edges as shown in Figures A.5(h) and A.5(i), then, this time, the group emanates eight edges which guarantee that no degree-6 node connecting to this group more than twice. Therefore, the topologies can be reduced to a $(6, 8)$ set by removing all degree-4 nodes, respectively.

(f) If there are five edges as shown in Figure A.5(j), now the group emanates six edges so that each degree-6 node connects to this group exactly once. Therefore, the set can be reduced to a $(6, 6)$ set by removing all degree-4 nodes.

(g) The degree-4 nodes cannot be fully connected as shown in Figure A.5(k), since the other group demands at least six edges.

A.2.6.6 $b = 10$

There are six classes and we are sure at least two of them exist.

1. $[5, 5, 5, 5, 5, 5, 5, 5, 5, 5]$: there are 192 $(10, 10)$ absorption sets under this class. After checking thirty-two representative nodes, each of which is from one group, we found that there is only one topology as shown in Figure A.92.

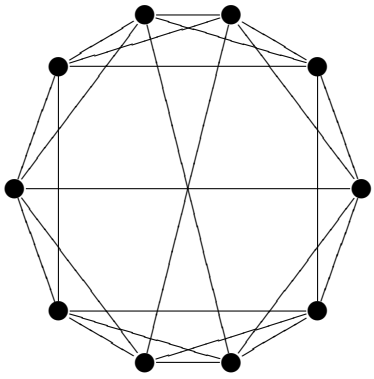


Figure A.92: $(10, 10)$ absorptions sets.

The average multiplicity of each variable node appeared in such sets is $192 \times 10 \div 2048 = 0.9375$. Once again, certain group of variable nodes share the same multiplicity, as listed in Table A.7. As we can see, some groups are not involved at all. Therefore, the average multiplicity of each involved variable node in such sets is $192 \times 10 \div 1408 \approx 1.363636$.

2. $[6, 5, 5, 5, 5, 5, 5, 5, 5, 4]$;

Table A.7: The multiplicity of each variable node in $(10, 10)$ absorption sets.

Group	Variable Nodes	Multiplicity	Group	Variable Nodes	Multiplicity
1	0—63	1	17	1024—1087	1
2	64—127	1	18	1088—1151	0
3	128—191	1	19	1152—1215	0
4	192—255	1	20	1216—1279	2
5	256—319	0	21	1280—1343	2
6	320—383	2	22	1344—1407	0
7	384—447	0	23	1408—1471	1
8	448—511	2	24	1472—1535	1
9	512—575	1	25	1536—1599	1
10	576—639	1	26	1600—1663	0
11	640—703	1	27	1664—1727	2
12	704—767	2	28	1728—1791	2
13	768—831	0	29	1792—1855	1
14	832—895	2	30	1856—1919	1
15	896—959	1	31	1920—1983	0
16	960—1023	0	32	1984—2047	0

3. $[6, 6, 5, 5, 5, 5, 5, 5, 4, 4]$;
4. $[6, 6, 6, 5, 5, 5, 5, 4, 4, 4]$: the existence of this class is shown by Table A.8.
5. $[6, 6, 6, 6, 5, 5, 4, 4, 4, 4]$;
6. $[6, 6, 6, 6, 6, 4, 4, 4, 4, 4]$.

A.2.6.7 $b > 10$

The bigger $(10, 12)$, $(10, 14)$, $(10, 16)$, $(10, 18)$ and $(10, 20)$ absorption sets do exist and an example of each are shown in Tables A.9–A.13.

Table A.8: An example $(10, 10)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
591	56	65	159	207	302	327
1405	32	120	159	225	314	337
1904	0	119	184	249	314	379
210	42	65	160	248	286	335
732	30	118	157	223	312	335
1676	36	77	183	223	286	376
616	30	119	160	194	275	373
834	42	85	157	249	302	373
892	36	118	142	225	275	327

Table A.9: An example $(10, 12)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
1460	2	79	184	238	307	365
1320	46	67	140	248	307	320
1543	51	72	145	253	312	320
931	45	88	160	252	305	376
9	46	110	174	238	302	366
1104	33	67	145	252	282	349
1316	51	88	156	199	302	365

Table A.10: An example $(10, 14)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
1460	2	79	184	238	307	365
931	45	88	160	252	305	376
9	46	110	174	238	302	366
1121	15	110	156	198	315	321
1316	51	88	156	199	302	365
1432	33	121	160	226	315	338
1549	45	121	142	198	300	366

Table A.11: An example $(10, 16)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
1045	23	110	184	199	306	336
176	43	78	174	251	267	376
39	15	79	143	207	271	335
305	23	78	132	201	259	335
1048	19	76	143	253	262	354
1189	43	76	132	234	300	326
1444	19	110	140	207	317	326

Table A.12: An example $(10, 18)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
170	49	124	184	196	283	366
931	45	88	160	252	305	376
253	63	124	174	226	259	336
1121	15	110	156	198	315	321
1432	33	121	160	226	315	338
1549	45	121	142	198	300	366
2016	9	113	156	196	259	349

Table A.13: An example $(10, 20)$ absorption set.

Variable Node	**Six Neighboring Check Nodes**					
0	56	120	184	248	312	376
109	56	79	174	199	300	349
90	15	120	140	253	305	338
358	52	68	184	255	315	327
1056	10	115	135	248	300	333
574	15	80	169	255	316	333
712	52	125	147	198	316	347
862	10	125	174	242	285	325
1207	25	68	140	232	285	356
1837	42	79	147	253	293	356

Appendix B

Tanner Codes

In [75, 76], Tanner introduced a class of regular LDPC codes composed of blocks of shifted identity matrices, named circulant matrices. By carefully configuring the parameters, 4-cycles can be avoided and the constructed code may achieve the upper bound of the minimum distance.

The parity-check matrix $\mathbf{H}$ will have a $d_v \times d_c$ array of circulant permutation matrices:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{0,0} & \mathbf{I}_{0,1} & \cdots & \mathbf{I}_{0,d_c-1} \\ \mathbf{I}_{1,0} & \mathbf{I}_{1,1} & \cdots & \mathbf{I}_{1,d_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_{d_v-1,0} & \mathbf{I}_{d_v-1,1} & \cdots & \mathbf{I}_{d_v-1,d_c-1} \end{bmatrix}_{d_v p \times d_c p} . \tag{B.1}$$

Immediately, we can tell that it is a (d_c, d_v) regular LDPC code. The code-length is $N = d_c p$ and the number of check nodes is $d_v p$. The actual rate R will be slightly higher than the design code rate, $1 - d_v/d_c$, in that within every row blocks, all p rows add to an all-1 vector and it leads to that at least $d_v - 1$ rows are linearly dependent.

The dimension of the circulant matrices p is a prime and also has the property that $d_v|\phi(p)$ and $d_c|\phi(p)$, where $\phi(n)$ is the Euler's totient function that is defined to be the number of positive integers less than or equal to n that are coprime to n. Hence $\phi(p) = p - 1$.

Now, we need to find two parameters m and n, which are nonzero elements from the Galois field GF$(p) = \{0, 1, \ldots, p-1\}$, such that

$$\begin{array}{rcl} o(m) & = & d_c \\ o(n) & = & d_v \end{array} . \tag{B.2}$$

In other words, m and n have to satisfy

$$\begin{array}{rcll} m^{d_c} & \equiv & 1 & (\mathrm{mod}\ p) \\ n^{d_v} & \equiv & 1 & (\mathrm{mod}\ p) \end{array} . \tag{B.3}$$

Conditions $d_v|(p-1)$ and $d_c|(p-1)$ on p guarantee the existence of such m and n.

Then we create a matrix in GF(p) to denote the number of shifts x of each circulant matrix.

$$\mathbf{X} = [x_{ij}] \tag{B.4}$$

$$= \left[m^j n^i\right] \tag{B.5}$$

$$\equiv \begin{bmatrix} 1 & m & m^2 & \cdots & m^{d_c-1} \\ n & mn & m^2n & \cdots & m^{d_c-1}n \\ n^2 & mn^2 & m^2n^2 & \cdots & m^{d_c-1}n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n^{d_v-1} & mn^{d_v-1} & m^2n^{d_v-1} & \cdots & m^{d_c-1}n^{d_v-1} \end{bmatrix}_{d_v \times d_c} \pmod{p}, \quad \text{(B.6)}$$

where $i = 0, 1, \ldots, d_v - 1$ and $j = 0, 1, \ldots, d_c - 1$.

Last, the parity-check matrix is obtained as

$$\mathbf{H} = \mathbf{I}_{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_m & \mathbf{I}_{m^2} & \cdots & \mathbf{I}_{m^{d_c-1}} \\ \mathbf{I}_n & \mathbf{I}_{mn} & \mathbf{I}_{m^2n} & \cdots & \mathbf{I}_{m^{d_c-1}n} \\ \mathbf{I}_{n^2} & \mathbf{I}_{mn^2} & \mathbf{I}_{m^2n^2} & \cdots & \mathbf{I}_{m^{d_c-1}n^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{I}_{n^{d_v-1}} & \mathbf{I}_{mn^{d_v-1}} & \mathbf{I}_{m^2n^{d_v-1}} & \cdots & \mathbf{I}_{m^{d_c-1}n^{d_v-1}} \end{bmatrix}_{d_v p \times d_c p}, \quad \text{(B.7)}$$

where each $\mathbf{I}_x$ denotes a $p \times p$ identity matrix with rows shifted to the left by x positions

The size of the base identity matrix p seems to play an important role during the construction process. In [76], p was extended to non-primes. Also the authors showed that for prime p, the girth of the derived code g is lower bounded by 6, whereas it can be 4 using non-prime p.[1] Either case, it can be shown that g is upper bounded by 12, no matter how big the code grows, [75, 27]. Using the analysis of [53], the minimum distance is bounded by $d_{\min} \leq (d_v + 1)!$. When d_v is small, this bound can be met by carefully choosing the parameters. As a matter of fact, Tanner presented a $[155, 64, 20], (3, 5)$ code with $g = 8$ in the "Recent Results" session of the 2000 International Symposium on Information Theory, in Sorrento, Italy.

Tanner codes come with relatively large girth or/and minimum distance, as to provide a good performance. However, the inherent structure flaw causing the existence of absorption sets, therefore the error floor problem, still cannot be corrected. In the following sections, we will showcase three Tanner codes and analyze their decoding performance.

B.1 Tanner Code $[155, 64, 20], (3, 5)$

Let us start with $(d_v, d_c) = (3, 5)$ with a design rate 0.4. In order to satisfy both $d_v|(p-1), d_c|(p-1)$ and p being a prime, the smallest value will be $p = 31$. Thus the minimal code length will have to be $d_c \times p = 155$. A solution to (B.3) is

$$\begin{array}{rcl} m & = & 2 \\ n & = & 5 \end{array}. \quad \text{(B.8)}$$

Substituting into (B.6) results to

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 5 & 10 & 20 & 9 & 18 \\ 25 & 19 & 7 & 14 & 28 \end{bmatrix}_{3 \times 5}, \quad \text{(B.9)}$$

[1] During the course of our enumeration of absorption sets, including codewords, the multiplicities are multiples of this base matrix dimension p. This suggests that there might be a non-exaustive type of search by making use of the field knowledge plus code design. However, the absorption set topologies are necessary in estimating the error rate, unless a close estimation of the maximum eigenvalue and corresponding eigenvector is available.

which in turn gives the parity-check matrix representing this Tanner code as

$$\mathbf{H} = \mathbf{I}_{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_4 & \mathbf{I}_8 & \mathbf{I}_{16} \\ \mathbf{I}_5 & \mathbf{I}_{10} & \mathbf{I}_{20} & \mathbf{I}_9 & \mathbf{I}_{18} \\ \mathbf{I}_{25} & \mathbf{I}_{19} & \mathbf{I}_7 & \mathbf{I}_{14} & \mathbf{I}_{28} \end{bmatrix}_{93\times 155} . \tag{B.10}$$

The specifications of the code are listed below.

- Regularity: $d_v = 3, d_c = 5$.
- The number of variable nodes or code length: $N = 155$.
- The number of check nodes: 93.
- Information bits: $k = 64$.
- Design code rate: $1 - d_v/d_c = 0.4$.
- Actual code rate: $R = k/N \approx 0.4129$.
- Minimum distance: $d_{\min} = 20$ [76].
- Girth: $g = 8$ [75].
- Block structure of $\mathbf{H}$ [76]:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_4 & \mathbf{I}_8 & \mathbf{I}_{16} \\ \mathbf{I}_5 & \mathbf{I}_{10} & \mathbf{I}_{20} & \mathbf{I}_9 & \mathbf{I}_{18} \\ \mathbf{I}_{25} & \mathbf{I}_{19} & \mathbf{I}_7 & \mathbf{I}_{14} & \mathbf{I}_{28} \end{bmatrix}_{93\times 155} , \tag{B.11}$$

  where each $\mathbf{I}_x$ is derived by shifting the rows of a 31×31 identity matrix cyclically to the left by x positions, in other words, which are all permutation matrices.[2] The binary presentation of $\mathbf{H}$ is depicted in Figure 1.3, where non-zero entries are represented by solid nodes and zeros are not shown.

B.1.1 Absorption Sets

Facts on the first few absorption sets are shown in Table B.1. They are enumerated in the same way as the absorption sets of the IEEE 802.3an LDPC code in Appendix A.2, except it is way easier here and less time consuming thanks to the short length.

Since $g = 8$, then $a \geq 4$ by the definition of absorption sets. Recall we defined an unordered integer array $[\mathrm{Deg}(v_i) : i = 1, 2, \ldots, a]$ to denote a class of (a, b) absorption sets in Definition 3.5. To satisfy the number of edges in the subgraph generated by an absorption set, $\sum_{i=1}^{a} \mathrm{Deg}(v_i) = ad_v - b = 3a - b$, to be even, a and b have to be both even or both odd at the same time.

Now let us start with $a = 4$ to show how the multiplicities in Table B.1 are distributed among every topology.

B.1.1.1 $a = 4$

There are 465 of $(4, 4)$ sets, which all share one topology in Figure B.1. It is actually a length-8 cycle. In addition, each variable node appears exactly $465 \times 4 \div 155 = 12$ times.

[2] Or it can be defined alternatively—see another paper of Tanner's [75].

Table B.1: First few absorption sets of Tanner code [155, 64, 20] (3, 5).

a	b	**Existence**	**Multiplicity**	**Gain:** $\mu_{\max}$
< 4		No		
4	4	Yes	465	1
5	1	No		
	3	Yes	155	
	5		3,720	
6	2	No		
	4	Yes	930	
	6		22,630	1
7	1	No		
	3	Yes	930	
	5		16,275	
	7		140,430	1
8	2		465	1.7870
	4		5,115	
	6		196,540	
	8		823,515	1

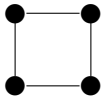
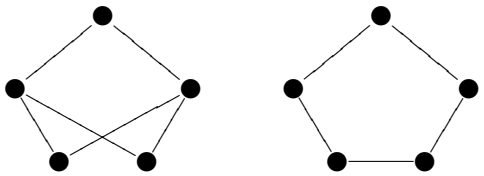


Figure B.1: The only possible topology of $(4, 4)$ absorption sets.

B.1.1.2 $a = 5$

Now b can be 1, 3 or 5. However, since $g = 8$, $b \neq 1$. Therefore, the only possible connections will be the ones shown in Figure B.2.



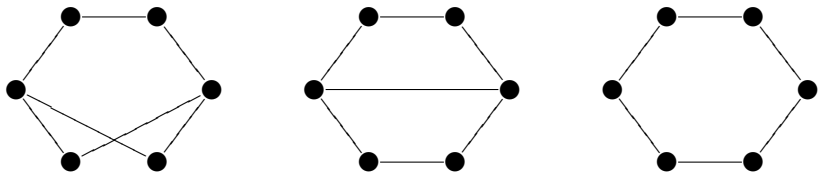
(a) $(5, 3)$ (b) $(5, 5)$

Figure B.2: The only possible topology of $(5, 5)$ absorption sets.

There are 155 of $(5, 3)$ sets. And each variable node appears exactly $155 \times 5 \div 155 = 5$ times. Note that each $(5, 3)$ set can reduce to three $(4, 4)$ sets by removing any of the three degree-2 nodes in Figure B.2(a) with counting possible duplicates. By checking, all 465 $(4, 4)$ sets are contained in $(5, 3)$ sets.

There are 3,720 of $(5, 5)$ sets. And each variable node appears exactly $3720 \times 5 \div 155 = 120$ times. Note that each $(5, 5)$ absorption set is a cycle of length 10, as shown in Figure B.2(b).

B.1.1.3 $a = 6$



(a) $(6, 4)$ Type I (b) $(6, 4)$ Type II (c) $(6, 6)$

Figure B.3: The only possible topologies of $(6, b)$ absorption sets.

There are 22,630 of $(6, 6)$ sets. And each variable node appears exactly $22630 \times 5 \div 155 = 876$ times. No $(6, 4)$ sets with topology Figure B.3(b), but 930 of Figure B.3(a)[3].

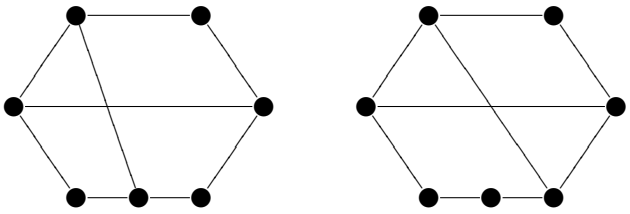
By checking, half of $(5, 3)$ sets can be reduced from $(6, 4)$ sets, by removing either of the bottom two nodes of Figure B.3(a).

B.1.1.4 $a = 7$

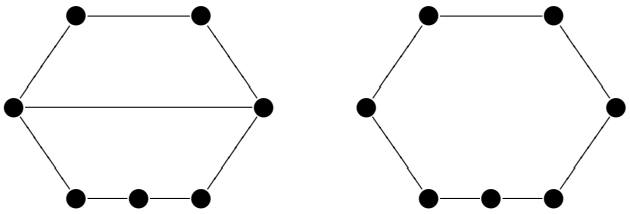
There are 930 of $(7, 3)$ sets. And each variable node appears exactly $930 \times 7 \div 155 = 42$ times. There are 16,275 of $(7, 5)$ sets[4]. And each variable node appears exactly $16275 \times 7 \div 155 = 735$ times. There are 140,430 of $(7, 7)$ sets. And each variable node appears exactly $140430 \times 7 \div 155 = 6342$ times.

[3] $930 \times 6 \div 155 = 36$.

[4] Include 7,440 of Figure B.4(c).



(a) $(7,3)$ Type I (b) $(7,3)$ Type II

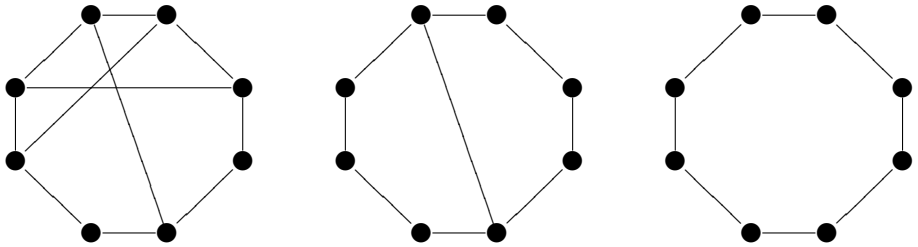


(c) $(7,5)$ (d) $(7,7)$

Figure B.4: Possible topologies of $(7,b)$ absorption sets.

By checking, all $(5,3)$ and $(6,4)$ sets can be reduced from $(7,3)$ sets. All $(5,5)$ sets can be reduced from $(7,5)$ sets. There are $9,300$ $(6,6)$ sets that can be reduced from $(7,5)$ sets.

B.1.1.5 $a = 8$



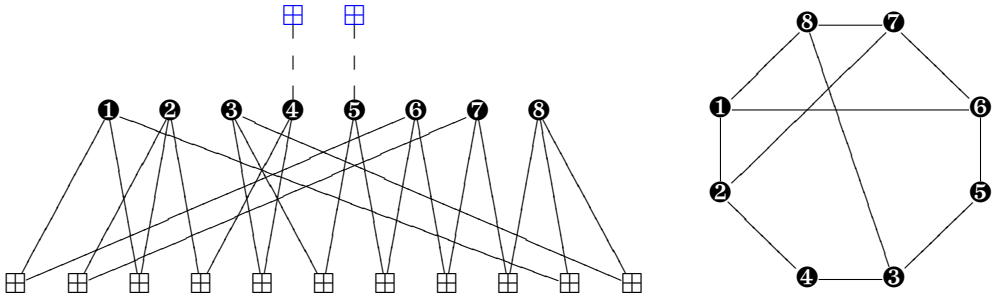
(a) $(8,2)$ (b) $(8,6)$ (c) $(8,8)$

Figure B.5: Possible topologies of $(8,b)$ absorption sets.

There are 465 of $(8,2)$ sets, which all have connectivity Figure B.5(a). And each variable node appears exactly $465 \times 8 \div 155 = 24$ times.

Let us take a closer look at the subgraph induced by this $(8,2)$ absorption set in Figure B.6, which is identical to Figure 3.13. We observe that one $(8,2)$ set contains

- two $(7,3)$ sets by removing either degree-2 node;
- three $(7,5)$ sets by removing any of ❶, ❷ and ❸;
- two $(6,4)$ sets by removing either ❸ and ❹ or ❻ and ❼;



(a) Check nodes shown.

(b) Check nodes hidden.

Figure B.6: The topology of the $(8, 2)$ absorption set.

- two $(6, 6)$ sets by removing either ❶ and ❷ or ❶ and ❽;
- one $(5, 3)$ set by removing ❹, ❺ and ❻;
- four $(5, 5)$ sets: ❶ ❷ ❸ ❹ ❺, ❺ ❻ ❼ ❽ ❶, ❶ ❷ ❼ ❻ ❺, and ❶ ❽ ❸ ❹ ❺;
- three $(4, 4)$ sets: ❶ ❷ ❸ ❽, ❶ ❷ ❼ ❽, and ❷ ❸ ❽ ❼.

As a matter of fact, all $(4, 4)$, $(5, 3)$, $(6, 4)$ and $(7, 3)$ sets are contained in $(8, 2)$ sets. And $1,860$ $(5, 5)$, 930 $(6, 6)$, $1,395$ $(7, 5)$ sets are contained in $(8, 2)$ sets, respectively.

In addition, there are 5,115 of $(8, 4)$ sets. And each variable node appears exactly $5115 \times 8 \div 155 = 264$ times. There are 196,540 of $(8, 6)$ sets. And each variable node appears exactly $196540 \times 8 \div 155 = 10144$ times. There are 823,515 of $(8, 8)$ sets. And each variable node appears exactly $823515 \times 8 \div 155 = 42504$ times.

B.1.2 Error Events Analysis

First we simulated the standard iterative min-sum (MS) decoding using: AWGN channel module, Binary Phase Shift Keying (BPSK) modulation and Max Iteration=50, floating LLR values confined within $[-10, 10]$. All-zero codeword is assumed to be the original transmitted message.

The simulation will not stop until more than one hundred decoding failures are collected. Then the composition of the error patterns is analyzed and tabularized. We run these simulations at SNRs from 4 to 7 dB at step size 0.25 dB.

Then we increase the LLR clipping range at the MS decoder and run all those experiments again. We only found that that the dominance of the error rate by the absorption sets is decreasing dramatically and that results in a lower error floor, which can be seen in Tables B.2–B.14. The raw data displayed in those tables can also be converted to ratios and presented as bar plots to help read the results. Figure 4.8(a) shows that the decoding failures are dominated by absorption sets. Whereas Figures 4.8(b) and B.7 demonstrate that $(8, 2)$ absorption set is the dominant one.

Table B.2: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 4 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 4$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	557,604	510,169	453,019	460,295
Total Error Frames	300	298	298	297
Total Error Bits	3,479	4,070	3,925	4,076
FER	5.38E–04	5.84E–04	6.58E–04	6.45E–04
BER	4.03E–05	5.15E–05	5.59E–05	5.71E–05
Codeword	0	0	0	0
(5,3)	0	1	0	1
(7,3)	2	3	4	6
(7,5)	0	0	0	1
(8,2)	66	7	16	11
(9,3)	0	0	0	2
(9,5)	0	0	0	2
(10,2)	16	1	3	1
(10,4)	0	2	1	0
(12,2)	1	0	0	0
(12,6)	0	2	0	0
(15,7)	1	0	0	0
(17,7)	0	0	0	1
Total Absorption Sets	86	16	24	25
Other Error Frames	214	282	274	272

Table B.3: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 4.25 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 4.25$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	1,323,688	1,298,475	1,220,096	1,211,145
Total Error Frames	299	298	298	297
Total Error Bits	3,231	3,807	3,643	3,869
FER	2.26E–04	2.30E–04	2.44E–04	2.45E–04
BER	1.57E–05	1.89E–05	1.93E–05	2.06E–05
Codeword	0	0	0	0
(4,4)	0	0	2	0
(5,3)	0	3	1	1
(7,3)	2	10	9	5
(7,5)	0	0	1	1
(8,2)	94	19	10	14
(8,4)	0	1	0	0
(9,3)	0	1	0	0
(9,5)	0	1	0	0
(10,2)	22	0	5	2
(12,2)	0	0	1	1
(16,10)	0	0	0	1
Total Absorption Sets	118	35	29	25
Other Error Frames	181	263	269	272

Table B.4: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 4.5 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 4.5$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	2,587,330	2,916,995	2,655,520	2,495,830
Total Error Frames	297	299	295	299
Total Error Bits	2,932	3,845	3,876	3,604
FER	1.15E–04	1.03E–04	1.11E–04	1.20E–04
BER	7.31E–06	8.50E–06	9.42E–06	9.32E–06
Codeword	0	0	0	0
(4,4)	0	1	1	0
(5,3)	0	1	2	1
(6,4)	0	0	1	1
(7,3)	1	2	9	7
(7,5)	0	1	0	0
(8,2)	116	8	10	18
(8,4)	1	4	3	0
(9,3)	1	0	0	1
(10,2)	35	2	1	3
(10,4)	0	0	1	1
(10,6)	0	0	1	0
(12,2)	2	0	0	1
(13,5)	0	1	0	0
(14,2)	1	0	0	0
(14,8)	0	0	1	0
(16,2)	1	0	0	0
Total Absorption Sets	158	20	30	33
Other Error Frames	139	279	265	266

Table B.5: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 4.75 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 4.75$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	5,518,124	7,173,802	6,076,975	6,761,129
Total Error Frames	293	297	295	292
Total Error Bits	2,677	3,962	3,539	3,254
FER	5.31E–05	4.14E–05	4.85E–05	4.32E–05
BER	3.13E–06	3.56E–06	3.76E–06	3.11E–06
Codeword	0	0	0	0
(4,4)	0	2	0	1
(5,3)	1	0	1	1
(6,4)	0	0	1	0
(7,3)	0	8	13	13
(7,5)	0	0	1	1
(8,2)	140	16	15	7
(8,4)	0	0	2	1
(9,3)	0	0	2	1
(10,2)	27	1	1	4
(10,4)	0	0	0	1
(12,2)	1	0	0	0
(18,2)	1	0	0	0
Total Absorption Sets	170	27	36	30
Other Error Frames	123	270	259	262

Table B.6: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 5 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 5$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	11,729,572	18,723,480	15,481,708	15,134,469
Total Error Frames	293	291	295	293
Total Error Bits	2,573	3,639	3,647	3,595
FER	2.50E–05	1.55E–05	1.91E–05	1.94E–05
BER	1.42E–06	1.25E–06	1.52E–06	1.53E–06
Codeword	0	0	0	0
(4,4)	0	0	1	1
(5,3)	0	0	4	0
(5,5)	0	1	1	0
(6,4)	0	0	0	1
(6,6)	0	0	1	0
(7,3)	3	5	7	4
(7,5)	0	2	2	1
(8,2)	177	27	21	12
(9,3)	0	0	0	1
(10,2)	37	0	2	4
(11,3)	1	1	0	1
(12,2)	2	0	1	1
(14,2)	1	0	0	0
Total Absorption Sets	221	36	40	26
Other Error Frames	72	255	255	267

Table B.7: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 5.25 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 5.25$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	24,346,162	44,881,001	48,725,120	42,295,870
Total Error Frames	295	287	290	285
Total Error Bits	2,519	3,115	3,370	3,539
FER	1.21E–05	6.39E–06	5.95E–06	6.74E–06
BER	6.68E–07	4.48E–07	4.46E–07	5.40E–07
Codeword	0	0	0	0
(4,4)	0	2	0	1
(5,3)	1	2	1	1
(5,5)	0	0	1	0
(6,4)	0	0	1	0
(7,3)	0	5	10	10
(7,5)	0	1	0	1
(8,2)	198	17	14	15
(8,4)	0	1	0	0
(9,3)	0	1	2	0
(9,5)	0	1	0	0
(10,2)	36	3	5	3
(12,2)	1	0	1	0
Total Absorption Sets	236	33	35	31
Other Error Frames	59	254	255	254

Table B.8: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 5.5 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 5.5$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	54,449,507	125,913,015	121,893,114	119,291,880
Total Error Frames	286	286	279	284
Total Error Bits	2,396	3,636	3,600	3,764
FER	5.25E–06	2.27E–06	2.29E–06	2.38E–06
BER	2.84E–07	1.86E–07	1.91E–07	2.04E–07
Codeword	0	0	0	0
(4,4)	0	0	1	2
(5,3)	3	0	0	0
(5,5)	0	0	1	0
(6,4)	0	1	0	0
(6,6)	0	1	0	0
(7,3)	2	9	10	3
(7,5)	0	0	2	0
(8,2)	198	12	12	17
(8,4)	0	1	0	0
(9,3)	1	0	0	0
(9,5)	0	0	1	0
(10,2)	37	0	2	4
(12,2)	2	0	0	0
(16,4)	1	0	0	0
Total Absorption Sets	244	24	29	26
Other Error Frames	42	262	250	258

Table B.9: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 5.75 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 5.75$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	117,059,584	340,548,946	331,125,421	352,913,708
Total Error Frames	291	268	279	286
Total Error Bits	2,410	3,288	3,740	4,103
FER	2.49E–06	7.87E–07	8.43E–07	8.10E–07
BER	1.33E–07	6.23E–08	7.29E–08	7.50E–08
Codeword	0	0	0	0
(5,3)	0	1	0	3
(5,5)	0	1	0	1
(6,4)	0	2	3	2
(6,6)	0	0	1	0
(7,3)	1	7	5	8
(7,5)	0	0	0	1
(8,2)	226	13	15	15
(8,4)	0	0	2	1
(9,3)	0	1	0	0
(9,5)	0	0	1	0
(10,2)	32	3	4	2
(10,4)	0	1	0	0
Total Absorption Sets	259	29	31	33
Other Error Frames	32	239	248	253

Table B.10: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 6 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 6$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	305,329,921	1,043,715,615	932,949,538	1,038,563,352
Total Error Frames	290	261	257	260
Total Error Bits	2,371	3,881	3,612	3,495
FER	9.50E–07	2.50E–07	2.75E–07	2.50E–07
BER	5.01E–08	2.40E–08	2.50E–08	2.17E–08
Codeword	0	0	0	0
(4,4)	0	0	2	0
(5,3)	1	0	1	0
(5,5)	0	1	0	1
(6,4)	0	0	0	2
(7,3)	1	5	5	3
(8,2)	229	15	8	8
(8,4)	0	0	1	0
(8,6)	0	0	0	2
(9,3)	1	0	0	0
(9,5)	0	1	0	0
(10,2)	26	2	1	2
(12,2)	1	0	0	1
(14,2)	1	0	0	0
(14,4)	0	0	0	1
Total Absorption Sets	260	24	18	20
Other Error Frames	30	237	239	240

Table B.11: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 6.25 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 6.25$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	826,962,468	3,623,931,874	2,556,722,831	2,688,364,511
Total Error Frames	272	247	243	241
Total Error Bits	2,255	3,519	3,567	3,531
FER	3.29E–07	6.82E–08	9.50E–08	8.96E–08
BER	1.76E–08	6.26E–09	9.00E–09	8.47E–09
Codeword	0	0	0	0
(4,4)	0	1	1	1
(5,3)	1	0	0	2
(5,5)	0	0	0	1
(6,4)	0	1	1	2
(7,3)	3	7	1	3
(8,2)	214	12	6	8
(8,4)	0	1	0	0
(9,3)	3	0	0	0
(10,2)	31	2	2	0
(12,2)	1	0	0	0
Total Absorption Sets	253	24	11	17
Other Error Frames	19	223	232	224

Table B.12: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 6.5 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 6.5$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	1,994,568,368	9,156,248,969	8,488,881,290	8,729,152,294
Total Error Frames	258	197	215	208
Total Error Bits	2,096	3,009	3,512	3,307
FER	1.29E–07	2.15E–08	2.53E–08	2.38E–08
BER	6.78E–09	2.12E–09	2.67E–09	2.44E–09
Codeword	0	0	0	0
(4,4)	0	0	0	1
(5,3)	1	1	2	1
(5,5)	0	0	1	0
(7,3)	1	2	4	4
(7,5)	0	1	0	0
(8,2)	215	4	6	8
(8,4)	0	0	0	1
(9,3)	1	0	0	0
(9,5)	1	0	0	1
(10,2)	24	0	0	0
(10,4)	0	0	1	1
Total Absorption Sets	243	8	14	17
Other Error Frames	15	189	201	191

Table B.13: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 6.75 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 6.75$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	5,534,970,894	28,814,519,101	29,174,130,466	28,453,186,304
Total Error Frames	263	151	183	175
Total Error Bits	2,103	2,291	3,692	2,977
FER	4.75E–08	5.24E–09	6.27E–09	6.15E–09
BER	2.45E–09	5.13E–10	8.16E–10	6.75E–10
Codeword	0	0	0	0
(4,4)	0	1	0	0
(5,3)	1	1	0	1
(6,4)	0	1	0	0
(7,3)	3	1	3	3
(8,2)	224	2	1	4
(8,6)	0	0	0	1
(9,3)	1	0	0	0
(9,5)	0	0	0	1
(9,7)	0	1	0	0
(10,2)	20	2	1	0
(12,2)	1	0	0	0
Total Absorption Sets	250	9	5	10
Other Error Frames	13	142	178	165

Table B.14: A breakdown of error events of Tanner $[155, 64, 20]$ $(3, 5)$ code at 7 dB of a min-sum decoder with different clipping values.

$E_b/N_0 = 7$ dB	Clipping@10	Clipping@38	Clipping@100	Clipping@∞
Total Tested Frames	15,357,623,747	90,768,900,266	161,114,248,732	77,339,128,967
Total Error Frames	242	104	183	141
Total Error Bits	1,903	1,387	3,692	2,910
FER	1.58E–08	1.15E–09	1.14E–09	1.82E–09
BER	7.99E–10	9.86E–11	1.48E–10	2.43E–10
Codeword	0	0	0	0
(5,3)	3	1	0	0
(6,4)	0	1	0	0
(7,3)	6	4	3	0
(7,5)	0	1	0	0
(8,2)	209	7	1	1
(8,6)	0	1	0	0
(9,3)	1	0	0	0
(10,2)	7	0	1	0
(11,3)	1	0	0	0
(12,2)	1	0	0	0
Total Absorption Sets	228	15	5	1
Other Error Frames	14	89	178	140

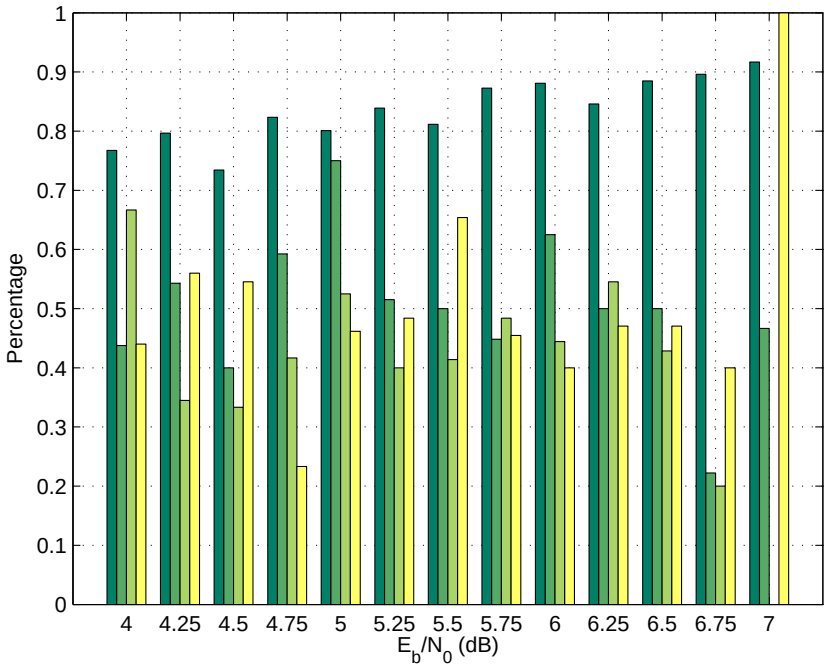


Figure B.7: The percent of $(8, 2)$ absorption sets of all absorption sets of Tanner [155, 64, 20] (3, 5) LDPC code. (Note that it shares the same legends as in Figure 4.8.)

B.2 Tanner Code $[755, 334, 14], (3, 5)$

Using the same $(d_v, d_c) = (3, 5)$ but with a bigger prime $p = 151$ we can construct a longer Tanner code with $N = d_c \times p = 755$. Now a solution to (B.3) is

$$\begin{array}{rcl} m & = & 8 \\ n & = & 32 \end{array} . \tag{B.12}$$

Substituting into (B.6) results to

$$\mathbf{X} = \begin{bmatrix} 1 & 8 & 64 & 59 & 19 \\ 32 & 105 & 85 & 76 & 4 \\ 118 & 38 & 2 & 16 & 128 \end{bmatrix}_{3\times 5} , \tag{B.13}$$

which in turn gives the parity-check matrix representing this Tanner code as

$$\mathbf{H} = \mathbf{I}_{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_8 & \mathbf{I}_{64} & \mathbf{I}_{59} & \mathbf{I}_{19} \\ \mathbf{I}_{32} & \mathbf{I}_{105} & \mathbf{I}_{85} & \mathbf{I}_{76} & \mathbf{I}_4 \\ \mathbf{I}_{118} & \mathbf{I}_{38} & \mathbf{I}_2 & \mathbf{I}_{16} & \mathbf{I}_{128} \end{bmatrix}_{453\times 755} . \tag{B.14}$$

The specifications of the code are listed below.

- Regularity: $d_v = 3, d_c = 5$.
- The number of variable nodes or code length: $N = 755$.
- The number of check nodes: 453.
- Information bits: $k = 334$.
- Design code rate: $1 - d_v/d_c = 0.4$.
- Actual code rate: $R = k/N \approx 0.4424$.
- Minimum distance: $d_{\min} = 14$ [76].
- Girth: $g = 10$ [75].
- Block structure of the parity-check matrix $\mathbf{H}$ [76]:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_8 & \mathbf{I}_{64} & \mathbf{I}_{59} & \mathbf{I}_{19} \\ \mathbf{I}_{32} & \mathbf{I}_{105} & \mathbf{I}_{85} & \mathbf{I}_{76} & \mathbf{I}_4 \\ \mathbf{I}_{118} & \mathbf{I}_{38} & \mathbf{I}_2 & \mathbf{I}_{16} & \mathbf{I}_{128} \end{bmatrix}_{453\times 755} , \tag{B.15}$$

  where each $\mathbf{I}_x$ is derived by shifting the rows of a 151×151 identity matrix cyclically to the left by x positions. The binary presentation of $\mathbf{H}$ is depicted in Figure B.8, where non-zero entries are represented by solid nodes and zeros are not shown.

B.2.1 Absorption Sets

Facts on the first few absorption sets are listed in Table B.15.

By the same reason stated in Appendix B.1.1, both a and b have to be even or odd at the same time. Due to the minimal cycle length $g = 10$, there are no $a < 5$ sets. So let us start with $a = 5$ to show every number in Table B.15 is true.

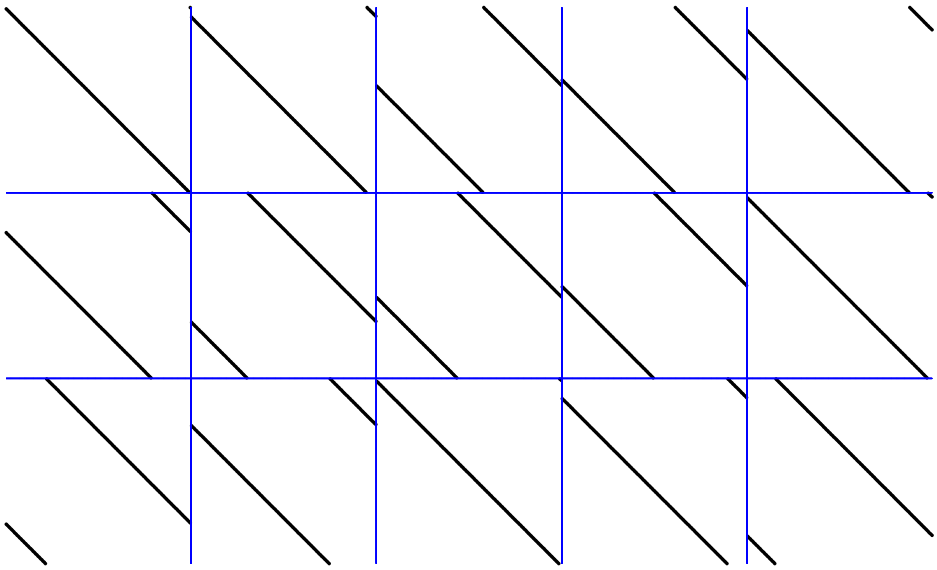
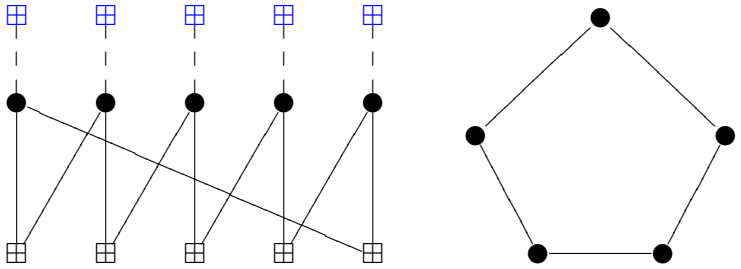


Figure B.8: Parity-check matrix of Tanner [755, 334, 14], (3,5) regular LDPC code.

Table B.15: First few absorption sets of Tanner code [755, 334, 14] (3, 5).

a	b	**Existence**	**Multiplicity**	**Gain:** $\mu_{\max}$
< 5		No		
5	1			
	3			
	5	Yes	$4,530$	1
6	2	No		
	4			
	6	Yes	$21,895$	1
7	1	No		
	3			
	5			
	7	Yes	$163,080$	1
8	2			
	4	No		
	6	Yes	$50,585$	
	8	No		



(a) Check nodes shown. (b) Check nodes hidden.

Figure B.9: The only possible topology of $(5, 5)$ absorption sets.

B.2.1.1 $a = 5$

The only possible connection will be a length-10 cycle as shown in Figure B.9.

So we are looking for the multiplicity of 10-cycles, which is $4,530$. In addition, every variable appears in the sets exactly $4530 \times 5 \div 755 = 30$ times.

B.2.1.2 $a = 6$

Similarly, the possible topology will be a 12-cycle as shown in Figure B.10.

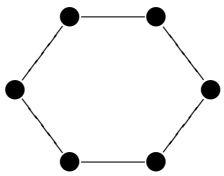
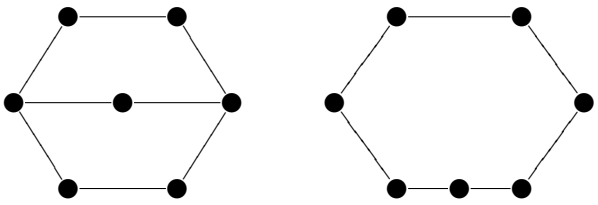


Figure B.10: The only possible topologies of $(6, 6)$ absorption sets.

By searching against the $\mathbf{H}$, the multiplicity of $(6, 6)$ sets is $21,895$ and every variable appears in the sets exactly $21895 \times 6 \div 755 = 174$ times.

B.2.1.3 $a = 7$

Due to the constraint $g = 10$, $b \neq 1$ or 3. So we have the following topologies.

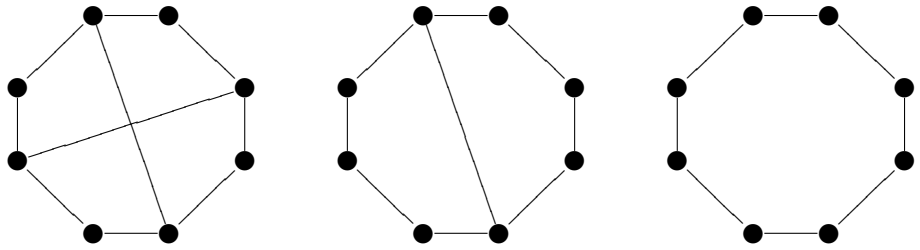


(a) (7,5) (b) (7,7)

Figure B.11: Possible topologies of $(7, b)$ absorption sets.

By searching against the $\mathbf{H}$, only $(7, 7)$ absorption sets exist and its multiplicity is $163,080$. Every variable appears in the sets exactly $163080 \times 7 \div 755 = 1512$ times.

B.2.1.4 $a = 8$



(a) $(8, 4)$ (b) $(8, 6)$ (c) $(8, 8)$

Figure B.12: Three possible topologies of $(8, b)$ absorption sets.

Figure B.12(c) shows a 16-cycle, whereas Figure B.12(b) can be seen as a joint of two 10-cycles.

By searching against the $\mathbf{H}$, the multiplicity of $(8, 6)$ sets is $50,585$[5] and every variable appears in the sets exactly $50585 \times 8 \div 755 = 536$ times.

$(8, 4)$ and $(8, 8)$ sets do not exist.

B.2.2 Error Events

Adopting the same simulation environment used before, we listed the breakdown of all error events gathered at the decoder in Table B.16 and in the format of percentages in Figure B.14. It is evident from Figure B.14 that the weight-$d_{\min}$ codeword is dominating the decoding failures of this code. Actually the slope of the curves at 3 dB in Figure B.13 is approximately 16 and tends to be 14, which equals $d_{\min}$, when the SNR gets higher.

[5]Including $15,855$ of topology Figure B.12(b).

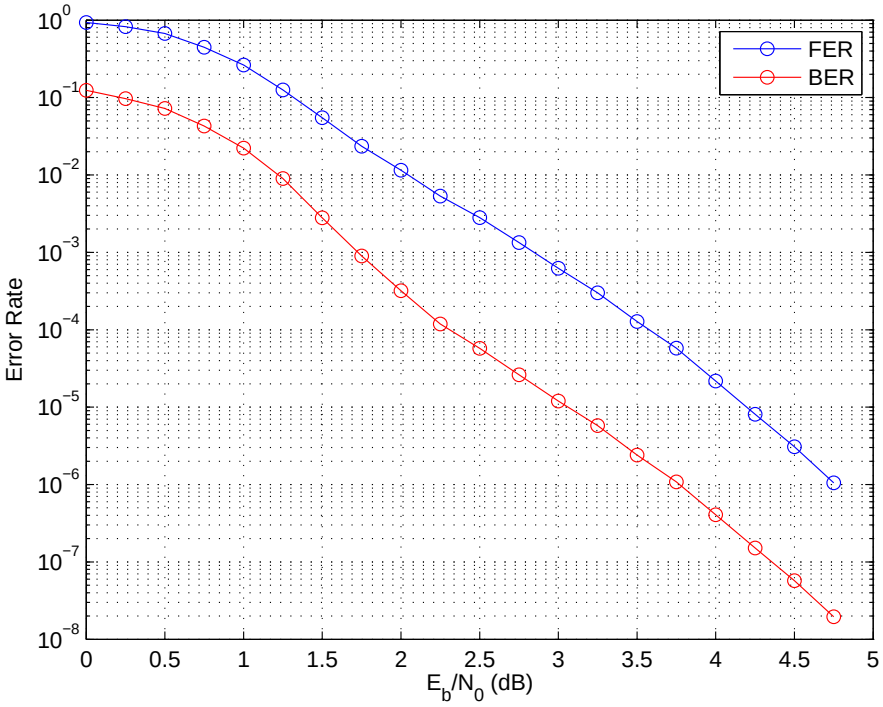


Figure B.13: The error rates of the Tanner [755, 334, 14] (3, 5) code using a standard min-sum decoder with LLR$\in [-10, 10]$ and maximum iteration=50 on AWGN channel.

Table B.16: A breakdown of Tanner [755, 334, 14] (3, 5) error events from a standard min-sum decoder with LLR$\in [-10, 10]$.

E_b/N_0	2.25 dB	2.5 dB	2.75 dB	3 dB	3.25 dB	3.5 dB	3.75 dB	4 dB	4.25 dB	4.5 dB	4.75 dB
Total Tested Frames	187,586	355,072	746,343	1,602,471	3,328,939	7,785,362	17,226,525	45,467,297	121,723,883	320,517,150	930,371,425
Total Error Frames	1,000	999	998	998	1,000	996	996	989	987	987	977
Total Error Bits	16,798	15,373	14,728	14,524	14,474	14,140	14,104	13,966	13,924	13,876	13,718
FER	5.33E–03	2.81E–03	1.34E–03	6.23E–04	3.00E–04	1.28E–04	5.78E–05	2.18E–05	8.11E–06	3.08E–06	1.05E–06
BER	1.19E–04	5.73E–05	2.61E–05	1.20E–05	5.76E–06	2.41E–06	1.08E–06	4.07E–07	1.52E–07	5.73E–08	1.95E–06
$d_{\min}$ Codeword	773	848	896	912	921	961	963	964	966	974	968
Other Codewords	199	146	101	86	79	35	33	25	21	13	9
Absorption Sets	0	0	0	0	0	0	0	0	0	0	0
Other Error Frames	28	5	1	0	0	0	0	0	0	0	0

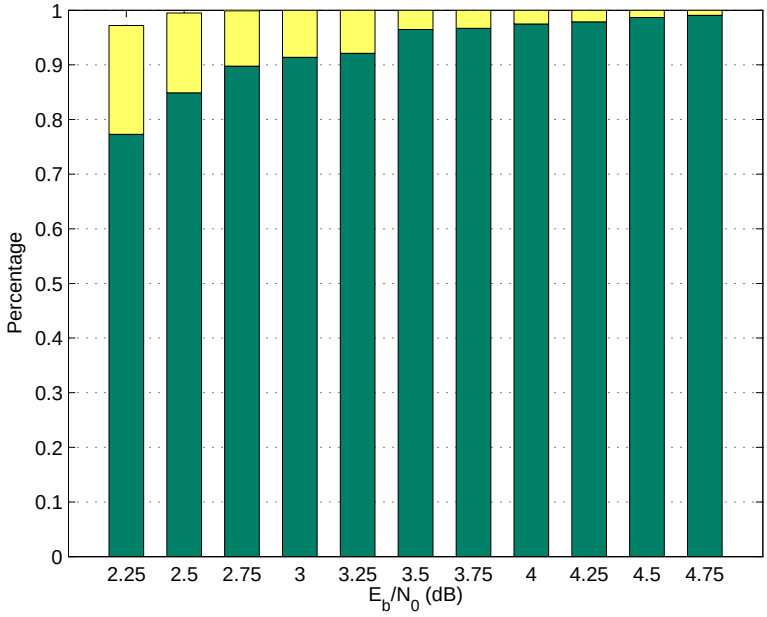


Figure B.14: The percent of codewords of the error events of Tanner [755, 334, 14] (3, 5) LDPC code using a standard min-sum decoder. Note that weight-$d_{\min}$ codeword is represented by green and all other codewords by yellow.

B.2.3 Minimum Weight Codeword

It seems that starting at 1.5 dB, far more codewords, especially the weight-$d_{\min}$ ones, are making appearance than absorption sets. So weight-$d_{\min}$ codewords are dominating the error floor.

One example of the codeword consists of the following variable nodes: $\{2, 75, 146, 153, 155, 226, 231, 290, 297, 326, 470, 477, 555, 557\}$, which has the topology shown in Figure B.15. Noticing that this topology consists of two disjoint $(6, 6)$ absorption sets, we can take advantage of the knowledge of the already enumerated $(6, 6)$ sets to find all codewords with this connectivity. It turns out that there are 755 of them.

However, to make sure all weight-$d_{\min}$ codewords share this topology, we will have to enumerate all minimum weight codewords without specifying any particular connectivity but common constraints only. The algorithm searching for absorption sets applies here finding codewords, but it is beyond our intention in studying the error floors caused by absorption sets. Besides, (non-zero) codewords are deemed as valid output of a decoder. This kind of errors can be treated by improving $d_{\min}$ which involves code design. That concludes our exploration on this Tanner code.

B.3 Tanner Code $[186, 35, 36], (5, 6)$

Now we turn to a low rate Tanner code with $(d_v, d_c) = (5, 6)$. To meet the constraints $d_v|(p-1), d_c|(p-1)$ and p being prime, the smallest candidate will be $p = 31$. A solution

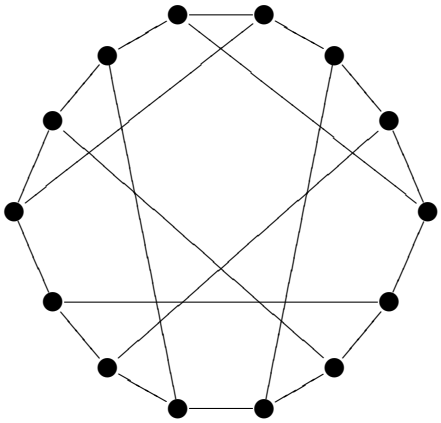


Figure B.15: A topology of $d_{\min}$ codeword of the Tanner $[755, 334, 14]$ $(3, 5)$ LDPC code.

to (B.3) is

$$\begin{array}{rcl} m & = & 6 \\ n & = & 2 \end{array} . \tag{B.16}$$

Substituting into (B.6) results to

$$\mathbf{X} = \begin{bmatrix} 1 & 6 & 5 & 30 & 25 & 26 \\ 2 & 12 & 10 & 29 & 19 & 21 \\ 4 & 24 & 20 & 27 & 7 & 11 \\ 8 & 17 & 9 & 23 & 14 & 22 \\ 16 & 3 & 18 & 15 & 28 & 13 \end{bmatrix}_{5\times 6} , \tag{B.17}$$

which in turn gives the parity-check matrix representing this Tanner code as

$$\mathbf{H} = \mathbf{I}_{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_6 & \mathbf{I}_5 & \mathbf{I}_{30} & \mathbf{I}_{25} & \mathbf{I}_{26} \\ \mathbf{I}_2 & \mathbf{I}_{12} & \mathbf{I}_{10} & \mathbf{I}_{29} & \mathbf{I}_{19} & \mathbf{I}_{21} \\ \mathbf{I}_4 & \mathbf{I}_{24} & \mathbf{I}_{20} & \mathbf{I}_{27} & \mathbf{I}_7 & \mathbf{I}_{11} \\ \mathbf{I}_8 & \mathbf{I}_{17} & \mathbf{I}_9 & \mathbf{I}_{23} & \mathbf{I}_{14} & \mathbf{I}_{22} \\ \mathbf{I}_{16} & \mathbf{I}_3 & \mathbf{I}_{18} & \mathbf{I}_{15} & \mathbf{I}_{28} & \mathbf{I}_{13} \end{bmatrix}_{155\times 186} . \tag{B.18}$$

Note that this matrix makes use of all $\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_{30}$ that are possible to cyclicly shift a dimension 31 identity matrix.

The specifications of the code are listed below.

- Regularity: $d_v = 5, d_c = 6$.
- The number of variable nodes or code length: $N = 186$.
- The number of check nodes: 155.
- Information bits: $k = 35$.
- Design code rate: $1 - d_v/d_c \approx 0.1667$.
- Actual code rate: $R = k/N \approx 0.1882$.
- Minimum distance: $d_{\min} = 36$.

- Girth: $g = 6$.
- Block structure of $\mathbf{H}[76]$[6]:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_6 & \mathbf{I}_5 & \mathbf{I}_{30} & \mathbf{I}_{25} & \mathbf{I}_{26} \\ \mathbf{I}_2 & \mathbf{I}_{12} & \mathbf{I}_{10} & \mathbf{I}_{29} & \mathbf{I}_{19} & \mathbf{I}_{21} \\ \mathbf{I}_4 & \mathbf{I}_{24} & \mathbf{I}_{20} & \mathbf{I}_{27} & \mathbf{I}_7 & \mathbf{I}_{11} \\ \mathbf{I}_8 & \mathbf{I}_{17} & \mathbf{I}_9 & \mathbf{I}_{23} & \mathbf{I}_{14} & \mathbf{I}_{22} \\ \mathbf{I}_{16} & \mathbf{I}_3 & \mathbf{I}_{18} & \mathbf{I}_{15} & \mathbf{I}_{28} & \mathbf{I}_{13} \end{bmatrix}_{155\times186}, \tag{B.19}$$

where each $\mathbf{I}_x$ is derived by shifting the rows of a 31×31 identity matrix cyclically to the left by x positions. The binary presentation of $\mathbf{H}$ is depicted in Figure B.16, where non-zero entries are represented by solid nodes and zeros are not shown.

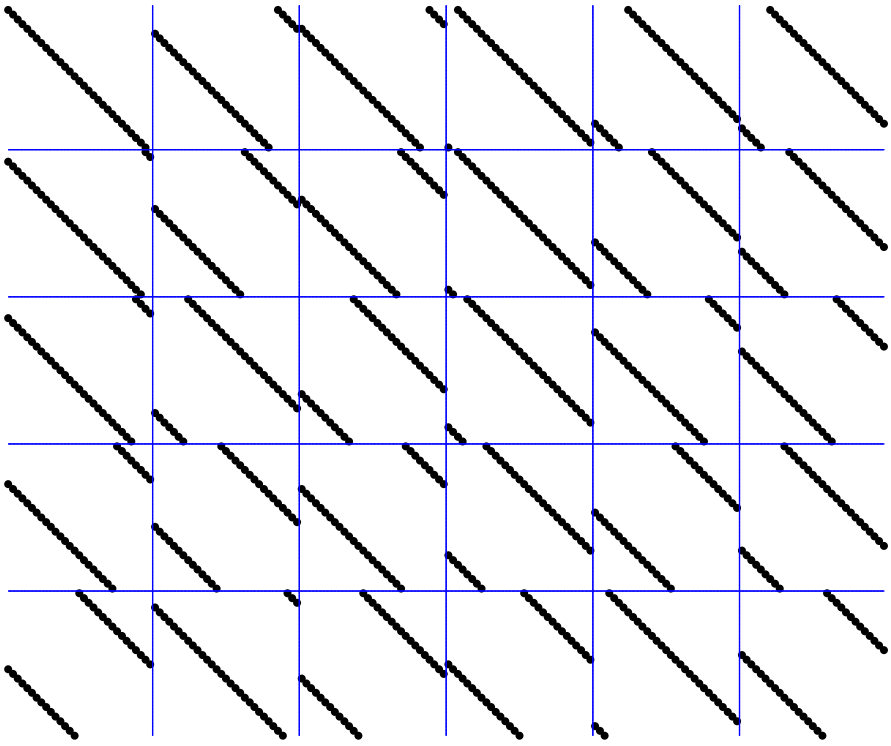


Figure B.16: Parity-check matrix of Tanner $[186, 35, 36]$ regular $(5, 6)$ LDPC code.

B.3.1 Girth

First we checked that no any two rows of $\mathbf{H}$ share more than one non-zero element. Therefore, this code is 4-cycle free. Then a 6-cycle was identified. So the girth of this code is

$$g = 6. \tag{B.20}$$

[6]By carefully choosing the configuration, we avoid constructing 4-cycles.

B.3.2 Absorption Sets

Since $d_v = 5$, then $a \geq 4$ by the definition of absorption sets and it is 4-cycle free. In addition, by the same argument in Appendix B.1.1, both a and b have to be even or odd at the same time. Now let us start with $a = 4$ to show every number in Table B.17 is true.

Table B.17: Absorption sets of Tanner code [186, 35, 36] (5, 6).

a	b	**Existence**	**Multiplicity**	**Gain:** $\mu_{\max}$
< 4		No		
4	8			
5	5			
	7			
	9			
6	0			
	2			
	4			
	6			
	8			
	10			
	12	Yes	$1,860$	2
7	1	No		
	3			
	5			
	7			
	9			
	11			
	13	Yes	$11,160$	
8	0	No		
	2			
	4			
	6			
	8			
	10			
	12	Yes	$1,395$	
	14		$43,710$	
	16		$26,040$	2

B.3.2.1 $a = 4$

By $g = 6$, the only possible absorption set is $(4, 8)$ as shown in Figure B.17 and none exists.

B.3.2.2 $a = 5$

It is readily seen that $5 \leq b \leq 10$ and odd. No topologies shown in Figure B.18 exist.

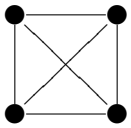
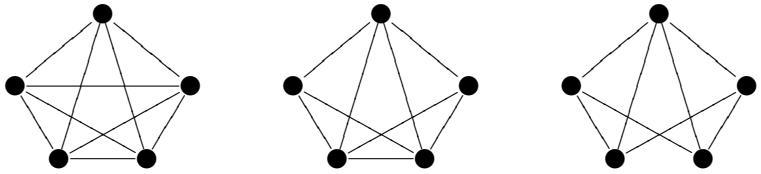


Figure B.17: The only possible topology of $(4, 8)$ absorption sets.



(a) $(5, 5)$ (b) $(5, 7)$ (c) $(5, 9)$

Figure B.18: Three possible topologies of $(5, b)$ absorption sets.

B.3.2.3 $a = 6$

Graphically, $(6, 0)$ absorption set topology exists as shown in Figure B.19. However, since each $(6, 0)$ set can be reduced to $(5, 5)$ absorption sets, shown in Figure B.18(a), by removing any one of the six nodes, there cannot be any $(6, 0)$ absorption set.

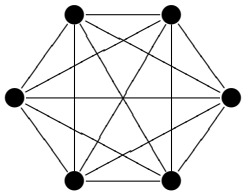
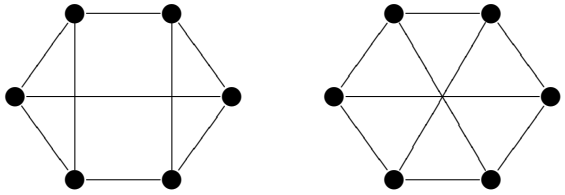


Figure B.19: The only possible topology of $(6, 0)$ absorption set or length-6 codeword.

$(6, 0)$ absorption set topology also represents a length-6 codeword. As a useful corollary, $d_{\min} > 6$. Because $d_{\min}$ has to be even [76], we have $d_{\min} \geq 8$.

By removing edges from Figure B.19, we can construct the rest topologies of $(6, b)$ sets, where $b = 2, 4, 6, 8, 10, 12$, accordingly. There is no $(6, 2)$ set, since it will give $(5, 5)$. However, thanks to the short length of this code, we simply did an exhaustive search rather than a topology analysis one by one. Only $(6, 12)$ sets are found. Theoretically, there exist two connectivities of $(6, 12)$, as shown in Figure B.20.

By searching against the $\mathbf{H}$, the multiplicity is $1,860$ and every variable appears in the sets exactly $1860 \times 6 \div 186 = 60$ times. A topology check confirms that all $(6, 12)$ sets share the connectivity Figure B.20(a), by the existence of 6-cycles and all check nodes degree is either 1 or 2 within the induced subgraph.



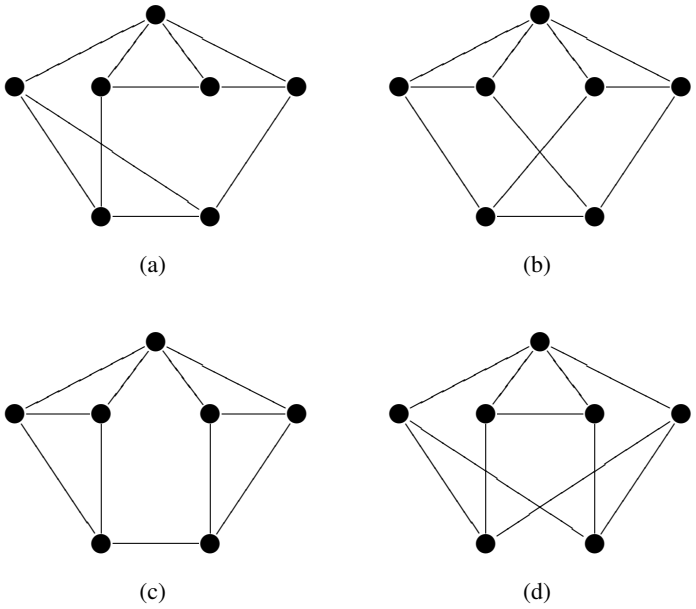
(a) With 6-cycle. (b) Without 6-cycle.

Figure B.20: Two possible topologies of $(6, 12)$ absorption sets.

B.3.2.4 $a = 7$

Both $(7, 1)$ and $(7, 3)$ absorption sets do not exist since they can be reduced to $(6, 4)$ or $(6, 6)$ sets.

There must be at least one degree-4 node in $(7, b)$, where $b = 5, 7, 9, 11, 13$. Again, by exhaustive search, only $(7, 13)$ exists, and the only possible class is $[4, 3, 3, 3, 3, 3, 3]$. There exist four connectivities, as shown in Figure B.21, for this class. By searching against the $\mathbf{H}$, the multiplicity is $11,160$ and every variable appears in the sets exactly $11160 \times 7 \div 186 = 420$ times. It turns out that topology as Figure B.21(d) does not exist and there are $5,580$, $3,720$ and $1,860$ $(7, 13)$ sets as Figure B.21(a)–B.21(c), respectively.



(a) (b)

(c) (d)

Figure B.21: Four possible topologies of $(7, 13)$ absorption sets.

B.3.2.5 $a = 8$

Both $(8, 0)$ and $(8, 2)$ absorption sets do not exist, because they can be reduced to $(7, 5)$ set. Therefore the minimum distance has been pushed further as $d_{\min} \geq 10$ and even.

By searching against the $\mathbf{H}$, there are $1,395$ $(8, 12)$ sets and every variable appears in

the sets exactly $1395 \times 8 \div 186 = 60$ times, $43,710$ $(8, 14)$ sets and every variable node appears $43710 \times 8 \div 186 = 1880$ times, and $26,040$ $(8, 16)$ sets and every variable appears $26040 \times 8 \div 186 = 1120$ times.

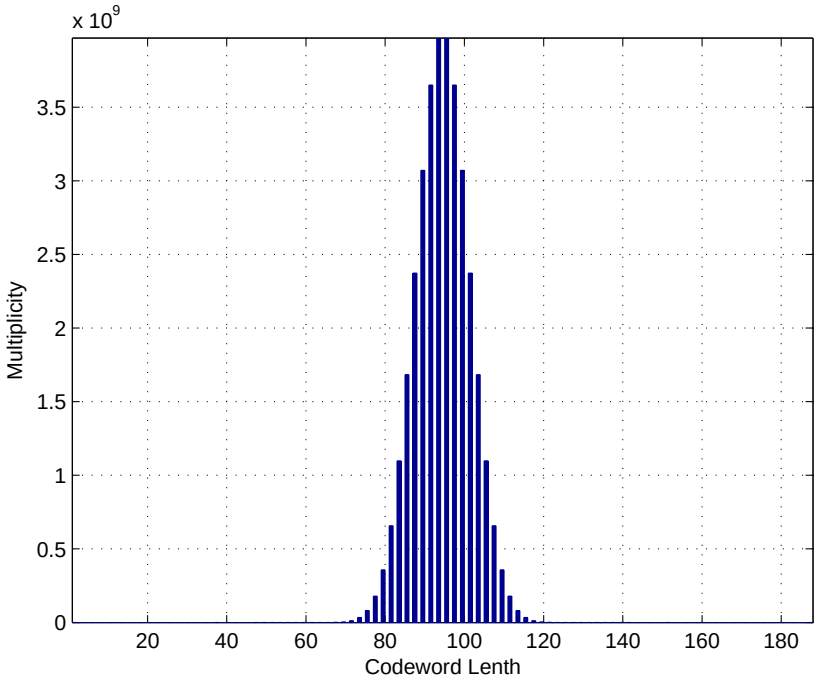
B.3.3 Codeword Spectrum

For a code with this length, it is affordable, nowadays, to find the codeword spectrum, as shown in Table B.18. The minimum distance is $d_{\min} = 36$, and there are 31 weight-$d_{\min}$ codewords.[7] The codeword distribution is also plotted as histograms shown in Figure B.22.

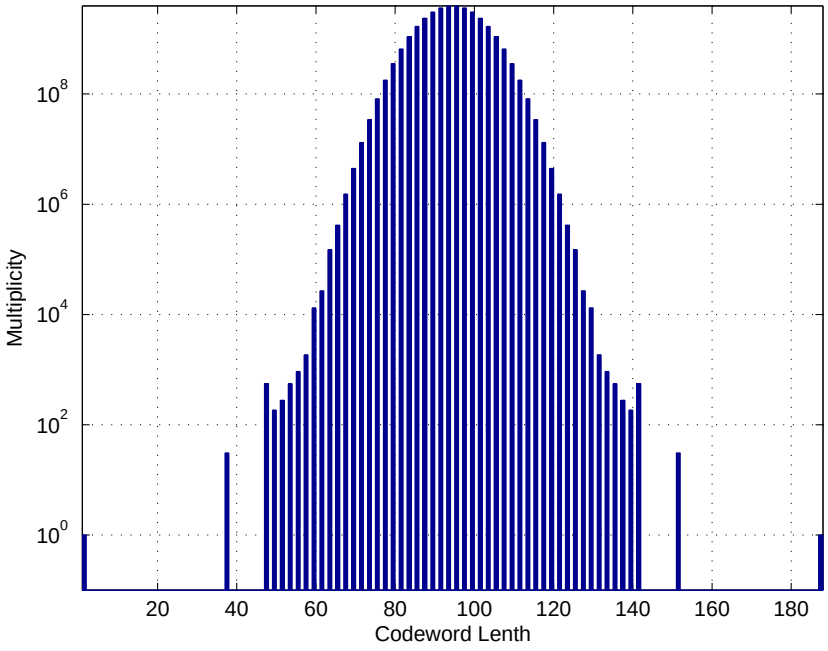
Table B.18: Codeword spectrum of Tanner [186, 35, 36] (5, 6) LDPC code.

Codeword Length	Multiplicity	Codeword Length	Multiplicity
0	1	186	1
36	31	150	31
46	558	140	558
48	186	138	186
50	279	136	279
52	558	134	558
54	930	132	930
56	1,860	130	1,860
58	13,206	128	13,206
60	26,970	126	26,970
62	150,210	124	150,210
64	417,105	122	417,105
66	1,534,097	120	1,534,097
68	4,480,368	118	4,480,368
70	13,254,732	116	13,254,732
72	34,459,786	114	34,459,786
74	81,915,144	112	81,915,144
76	178,960,272	110	178,960,272
78	357,797,598	108	357,797,598
80	657,805,926	106	657,805,926
82	1,098,017,427	104	1,098,017,427
84	1,684,019,138	102	1,684,019,138
86	2,374,135,155	100	2,374,135,155
88	3,071,687,607	98	3,071,687,607
90	3,651,598,128	96	3,651,598,128
92	3,969,591,912	94	3,969,591,912
Total		$34,359,738,368 = 2^{35}$	

[7] Recall that the block permutation matrices of this code has dimension $p = 31$.



(a) Linear scale.



(b) Log scale.

Figure B.22: Histograms of Tanner [186, 35, 36] codewords.

B.3.4 Error Events

The code performance is displayed in Figure B.23. It can be seen, together with Table B.19, that the error patterns are not dominated by absorption sets yet, while the error rate is already very low. This is due to the relatively large minimum distance of this code. In addition, it seems like that $(13, 9)$ absorption set is going to be the dominant one, which is bigger than the typical dominant absorption sets that we have observer. This also can be contributed to the large minimum distance. Therefore this code cannot help us example the relation between absorption sets and error floor.

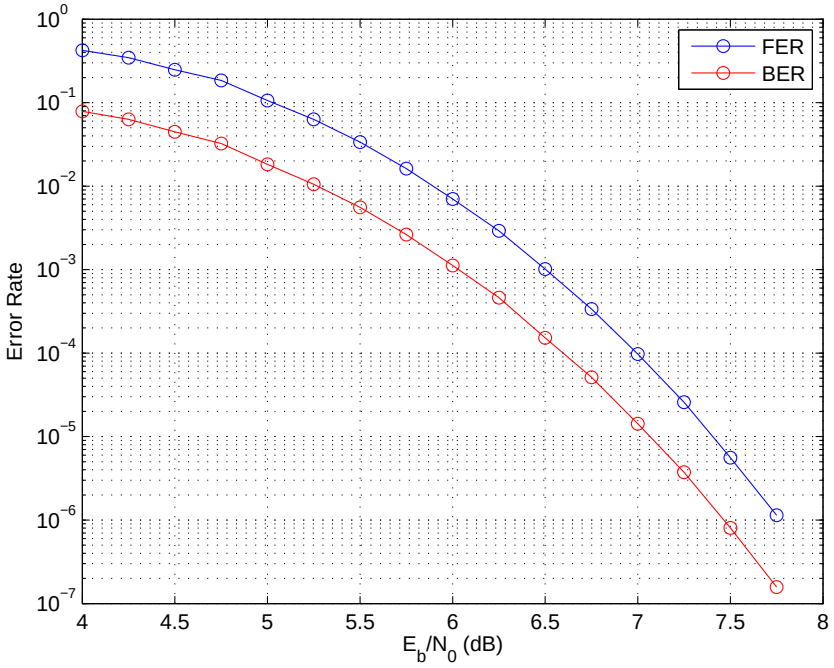


Figure B.23: The error rates of the Tanner [186, 35, 36] (5, 6) code using a standard min-sum decoder with LLR$\in [-10, 10]$ and maximum iteration=50 on AWGN channel.

B.4 Tanner Code $[104, 30, 14], (3, 4)$

Here we try $(d_v, d_c) = (3, 4)$ with a nonprime $p = 26$. Thus the code length is $N = d_c \times p = 104$. A solution to (B.3) is

$$\begin{matrix} m & = & 5 \\ n & = & 9 \end{matrix} . \tag{B.21}$$

Substituting into (B.6) results to

$$\mathbf{X} = \begin{bmatrix} 1 & 5 & 25 & 21 \\ 9 & 19 & 17 & 7 \\ 3 & 15 & 23 & 11 \end{bmatrix}_{3\times 4} , \tag{B.22}$$

Table B.19: A breakdown of Tanner [186, 35, 36] (5, 6) error events from a standard min-sum decoder with LLR$\in [-10, 10]$.

E_b/N_0	7 dB	7.25 dB	7.5 dB	7.75 dB
Total Tested Frames	10,276,165	38,836,263	179,026,287	878,340,338
Total Error Frames	1,000	1,000	1,000	1,000
Total Error Bits	27,190	27,067	26,844	25,714
FER	9.73E–05	2.57E–05	5.59E–06	1.14E–06
BER	1.42E–05	3.75E–06	8.06E–07	1.57E–07
Codewords	0	0	0	0
(11,11)	0	0	1	1
(13,9)	4	7	12	25
(16,10)	0	0	0	1
(17,9)	0	0	0	1
(17,11)	0	0	0	1
Total Absorption Sets	4	7	13	29
Other Error Frames	996	993	987	971

which in turn gives the parity-check matrix representing this Tanner code as

$$\mathbf{H} = \mathbf{I}_{\mathbf{X}} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_5 & \mathbf{I}_{25} & \mathbf{I}_{21} \\ \mathbf{I}_9 & \mathbf{I}_{19} & \mathbf{I}_{17} & \mathbf{I}_7 \\ \mathbf{I}_3 & \mathbf{I}_{15} & \mathbf{I}_{23} & \mathbf{I}_{11} \end{bmatrix}_{78\times 104} . \tag{B.23}$$

The specifications of the code are listed below.

- Regularity: $d_v = 3, d_c = 4$.
- The number of variable nodes or code length: $N = 104$.
- The number of check nodes: 78.
- Information bits: $k = 30$.
- Design code rate: $1 - d_v/d_c = 0.25$.
- Actual code rate: $R = k/N \approx 0.2885$.
- Minimum distance: $d_{\min} = 14$ [76].
- Girth: $g = 6$ [75].
- Block structure of $\mathbf{H}$ [76]:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_5 & \mathbf{I}_{25} & \mathbf{I}_{21} \\ \mathbf{I}_9 & \mathbf{I}_{19} & \mathbf{I}_{17} & \mathbf{I}_7 \\ \mathbf{I}_3 & \mathbf{I}_{15} & \mathbf{I}_{23} & \mathbf{I}_{11} \end{bmatrix}_{78\times 104} , \tag{B.24}$$

  where each $\mathbf{I}_x$ is derived by shifting the rows of a 26×26 identity matrix cyclically to the left by x positions. The binary presentation of $\mathbf{H}$ is depicted in Figure B.24, where non-zero entries are represented by solid nodes and zeros are not shown.

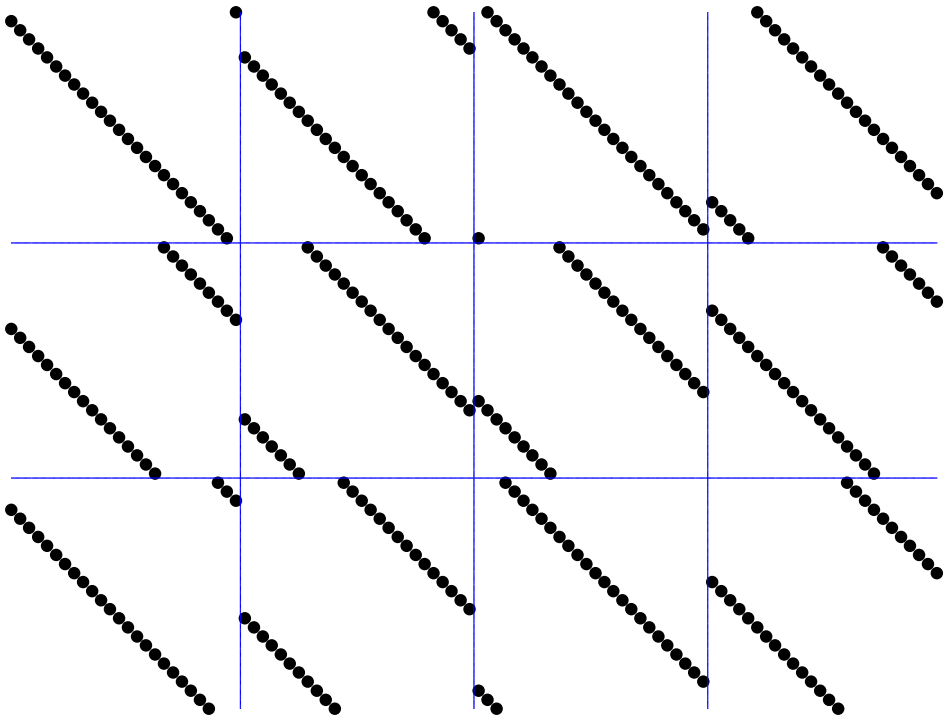


Figure B.24: Parity-check matrix of Tanner $[104, 30, 14]$ regular $(3, 4)$ LDPC code.

B.4.1 Absorption Sets

Facts on the first few absorption sets are shown in Table B.20 and enumeration steps are skipped. Interested readers are referred to the previous sections.

Table B.20: First few absorption sets of Tanner code [104, 30, 14] (3, 4).

a	b	**Existence**	**Multiplicity**	**Gain:** $\mu_{\max}$
< 4		No		
4	4	Yes	465	1
5	1	No		
	3	Yes	155	
	5		3,720	
6	2	No		
	4	Yes	930	
	6		22,630	1
7	1	No		
	3	Yes	930	
	5		16,275	
	7		140,430	1
8	2		465	1.7870
	4		5,115	
	6		196,540	
	8		823,515	1

B.4.2 Codeword Spectrum

The minimum distance of this code is $d_{\min} = 14$, and there are 156 weight-$d_{\min}$ codewords.[8] The codeword distribution is listed in Table B.21. It is also plotted using histograms, as shown in Figure B.25.

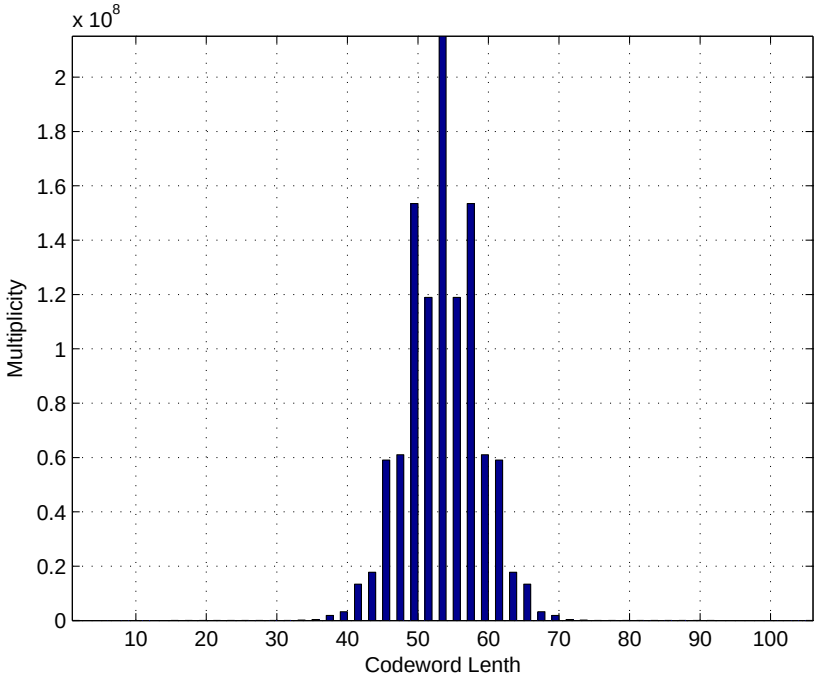
B.4.3 Error Events

The decoding performance of this code is shown in Figure B.26. It can be seen from Table B.22 that the dominance of the $(8, 2)$ absorption set to the error floor starts showing at 7 dB. By the time it completely dominating the decoding failures as the case of the Tanner $[155, 64, 20]$ LDPC code, the error rate will be too low to simulate. Therefore this code does not quality as a good example to analyze the relation between absorption sets and error floor.

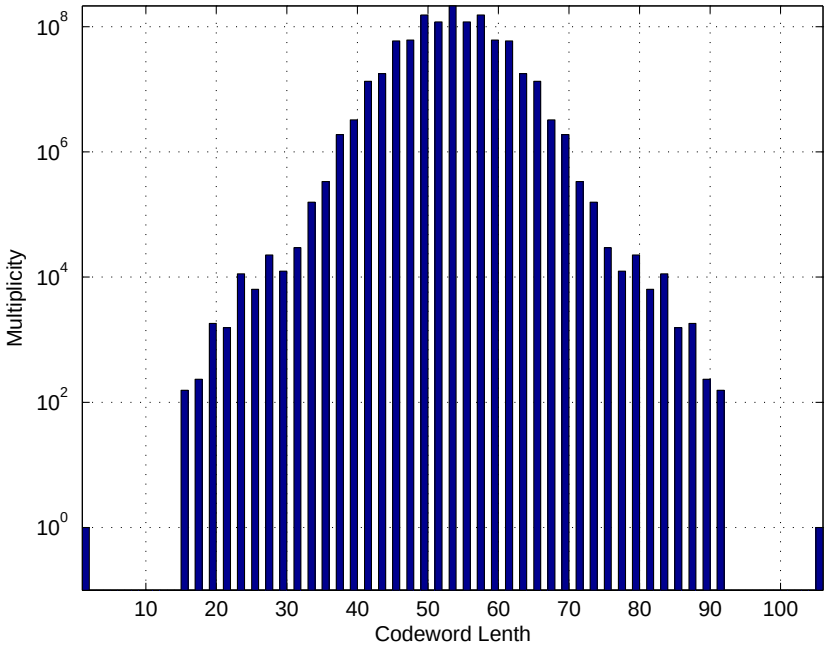
[8] Recall that the block permutation matrices of this code has dimension $p = 26$, and 156 is a multiple of p.

Table B.21: Codeword spectrum of Tanner [104, 30, 14] (3, 4) LDPC code.

Codeword Length	**Multiplicity**	**Codeword Length**	**Multiplicity**
0	1	104	1
14	156	90	156
16	234	88	234
18	1,820	86	1,820
20	1,560	84	1,560
22	11,284	82	11,284
24	6,396	80	6,396
26	22,632	78	22,632
28	12,480	76	12,480
30	29,536	74	29,536
32	157,209	72	157,209
34	336,440	70	336,440
36	1,891,006	68	1,891,006
38	3,238,872	66	3,238,872
40	13,390,468	64	13,390,468
42	17,768,712	62	17,768,712
44	59,044,648	60	59,044,648
46	61,001,460	58	61,001,460
48	153,498,748	56	153,498,748
50	118,915,680	54	118,915,680
52	215,083,140		
Total		$1,073,741,824 = 2^{30}$	



(a) Linear scale.



(b) Log scale.

Figure B.25: Histograms of Tanner [104, 30, 14] codewords.

Table B.22: A breakdown of Tanner [104, 30, 14] (3, 4) error events from a standard min-sum decoder with LLR$\in [-10, 10]$.

E_b/N_0	7 dB	7.25 dB	7.5 dB	7.75 dB
Total Tested Frames	146,688,836	340,220,598	934,396,191	2,491,868,947
Total Error Frames	917	887	855	820
Total Error Bits	6,469	5,770	5,490	5,010
FER	6.25E–06	2.61E–06	9.15E–07	3.29E–07
BER	4.24E–07	1.63E–07	5.65E–08	1.93E–08
$d_{\min}$ Codeword	5	0	3	0
Other Codewords	1	0	1	0
(4,4)	1	5	1	6
(5,3)	0	0	0	1
(5,5)	7	5	5	5
(6,4)	10	8	12	16
(6,6)	1	0	1	2
(7,3)	5	7	7	10
(7,5)	5	3	2	2
(8,2)	25	32	69	84
(8,4)	2	3	4	2
(8,6)	2	1	1	1
(8,8)	1	0	0	0
(9,3)	0	1	0	0
(9,5)	1	1	3	0
(9,7)	1	0	0	0
(10,2)	1	0	8	4
(10,6)	2	2	2	4
(11,3)	0	0	1	0
(11,5)	0	0	1	1
(11,7)	0	0	2	0
(12,2)	1	1	0	2
(13,9)	0	0	1	0
Total Absorption Sets	65	68	120	140
Other Error Frames	846	819	731	680

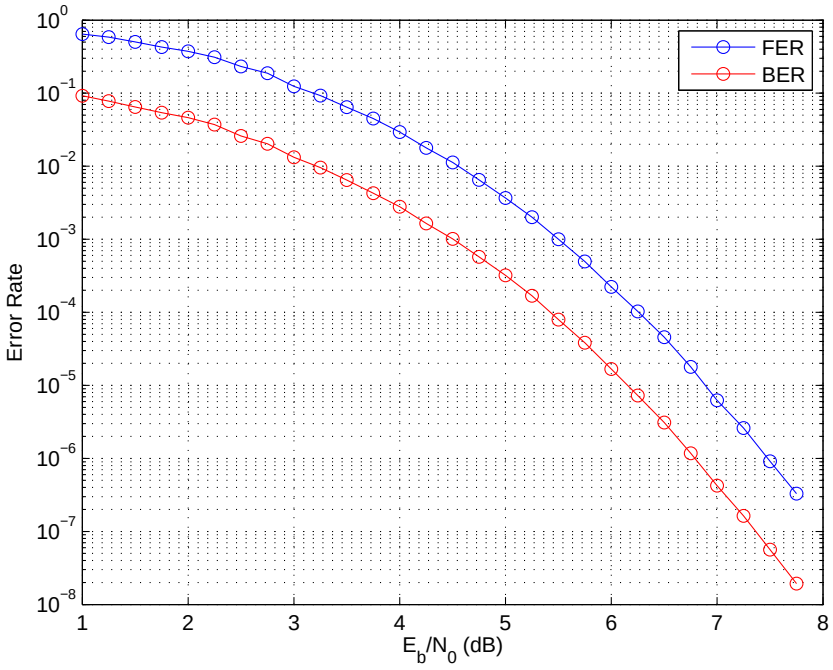


Figure B.26: The error rates of the Tanner [104, 30, 14] (3, 4) code using a standard min-sum decoder with LLR$\in [-10, 10]$ and maximum iteration=50 on AWGN channel.