A Mathematical Approach to Channel Codes with a Diagonal Matrix Structure

David G. M. Mitchell

A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
May 2009

Abstract

Digital communications have now become a fundamental part of modern society. In communications, *channel coding* is an effective way to reduce the information rate down to channel capacity so that the information can be transmitted reliably through the channel. This thesis is devoted to studying the mathematical theory and analysis of channel codes that possess a useful diagonal structure in the parity-check and generator matrices. The first aspect of these codes that is studied is the ability to describe the parity-check matrix of a code with sliding diagonal structure using polynomials. Using this framework, an efficient new method is proposed to obtain a generator matrix G from certain types of parity-check matrices with a so-called defective cyclic block structure. By the nature of this method, G can also be completely described by a polynomial, which leads to efficient encoder design using shift registers. In addition, there is no need for the matrices to be in systematic form, thus avoiding the need for Gaussian elimination.

Following this work, we proceed to explore some of the properties of diagonally structured low-density parity-check (LDPC) convolutional codes. LDPC convolutional codes have been shown to be capable of achieving the same capacity-approaching performance as LDPC block codes with iterative message-passing decoding. The first crucial property studied is the minimum free distance of LDPC convolutional code ensembles, an important parameter contributing to the error-correcting capability of the code. Here, asymptotic methods are used to form lower bounds on the ratio of the free distance to constraint length for several ensembles of asymptotically good, protograph-based LDPC convolutional codes. Further, it is shown that this ratio of free distance to constraint length for such LDPC convolutional codes exceeds the ratio of minimum distance to block length for corresponding LDPC block codes.

Another interesting property of these codes is the way in which the structure affects the performance in the infamous error floor (which occurs at high signal to noise ratio) of the bit error rate curve. It has been suggested that "near-codewords" may be a significant factor affecting decoding failures of LDPC codes over an additive white Gaussian noise (AWGN) channel. A near-codeword is a sequence that satisfies almost all of the check equations. These near-codewords can be associated with so-called 'trapping sets' that exist in the Tanner graph of a code. In the final major contribution of the thesis, trapping sets of protograph-based LDPC convolutional codes are analysed. Here, asymptotic methods are used to calculate a lower bound for the trapping set growth rates for several ensembles of asymptotically good protograph-based LDPC convolutional codes. This value can be used to predict where the error floor will occur for these codes under iterative message-passing decoding.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself in the Department of Electronics and Electrical Engineering at The University of Edinburgh.

David G. M. Mitchell

Acknowledgements

I would like to begin by thanking my advisors at the University of Edinburgh, Prof. Norbert Goertz and Dr. Liam O'Carroll. I thank Prof. Goertz very much for his guidance and help throughout my degree, in particular for explaining many of the important engineering aspects of this field. Prior to my undertaking of this degree, I received a BSc. in Mathematics from the University of Edinburgh. Dr. O'Carroll supervised my honours year project, and also lectured the Algebraic Coding Theory course. This course sparked my interest in channel coding from a mathematical perspective and Dr. O'Carroll's excellent teaching enthused me to undertake the Ph.D. degree. Many thanks for this, and the continued support ever since. I would also like to thank Prof. Goertz and Dr. O'Carroll for helping to develop my research skills and independent thinking.

The award of the John Moyes Lessells Travel Scholarship from the Royal Academy of Edinburgh allowed me to visit the University of Notre Dame in Indiana, USA for three months. There, I was fortunate enough to work with Prof. Daniel J. Costello, Jr., Prof. Kamil Sh. Zigangirov and Dr. Ali E. Pusane. I am extremely grateful to Prof. Costello for his excellent advice and guidance during my time at Notre Dame which has continued to date following my return to Edinburgh. Thanks to Prof. Zigangirov for the helpful electronic comments and suggestions. Many thanks to Dr. Pusane, with whom I have very much enjoyed working with and sharing countless useful discussions.

I would also like to thank my colleagues at the Institute for Digital Communications for useful discussions, advice and for all the welcome distractions. In particular, thanks to Prof. Mike E. Davies for overseeing the latter part of my degree and for his advice during this time. Finally, I would like to thank my parents and family for their unwavering support. Thanks especially to my beautiful fiancée Bridget for her constant encouragement and interest.

This thesis is the result of several collaborations. The results presented in Chapter 3 are joint work with my advisors Dr. O'Carroll and Prof. Goertz and have, in part, previously been published in [MOG07,MOG08]. A selection of the results presented in Chapter 5 is joint work with Dr. Pusane, Prof. Goertz, Prof. Zigangirov and Prof. Costello and has previously been published in [MPZC08, MPGC08]. Thanks also to Ali Pusane for providing the simulation results displayed in Figure 5.8. An extension of this work, detailed in Chapter 6, is a collaboration with Dr. Pusane and Prof. Costello. A selection of the results presented here has previously been published in [MPC09a, MPC09b]. I acknowledge the financial support for the duration of the degree from the Scottish Funding Council for the Joint Research Institute with the Heriot-Watt University, which is a part of the Edinburgh Research Partnership. Thanks to the Royal Society of Edinburgh for the award of the John Moyes Lessells Travel Scholarship, and to the Royal Academy of Engineering for International Travel Grants ITG 08-663 and ITG 09-534.

Contents

List of figures

List of tables

Acronyms and abbreviations

APP	A posteriori probability
ARA	Accumulate-repeat-accumulate
ARJA	Accumulate repeat jagged accumulate
AWGN	Additive white Gaussian noise
BCH	Bose-Chaudhuri-Hocquenghem
BCJR	Bahl, Cocke, Jelinek and Raviv
BEC	Binary erasure channel
BER	Bit-error rate
BF	Bit-flipping
BP	Belief propagation
BPSK	Binary phase shift keying
BSC	Binary symmetric channel
CAP	Copy-and-permute
FER	Frame-error rate
IRA	Irregular repeat-accumulate
LDPC	Low-density parity-check
MAP	Maximum a posteriori probability
ML	Maximum likelihood
PSD	Power spectral density
QC	Quasi-cyclic
RA	Repeat-accumulate
RJA	Repeat jagged accumulate
RM	Reed-Muller
RS	Reed-Solomon
SISO	Soft-input soft-output
SNR	Signal-to-noise ratio
SPA	Sum-product algorithm
XOR	Exclusively-or

Chapter 1
Introduction

1.1 Digital communications

Digital communications have now become a fundamental part of modern society. The need for reliable data transmission and storage systems is paramount for applications such as high speed data networks, mobile phones, the internet, large scale data storage for commercial and governmental use, and many others. A major challenge of any system designer is how to reproduce data reliably in the presence of errors.

In the landmark 1948 paper entitled "A mathematical theory of communication" by Claude Shannon [Sha48], the concept of *information* was quantified and the field we know today as Information Theory was born. Shannon gave a mathematical proof showing that, by proper encoding, reliable transmission (or storage) of information is possible for a wide range of channels provided that the information (storage) rate does not exceed the so-called *capacity* of the channel. The channel capacity is determined by the physical properties of the channel. A model of the overall digital communications system based on the principles of this remarkable paper is displayed in Figure 1.1.

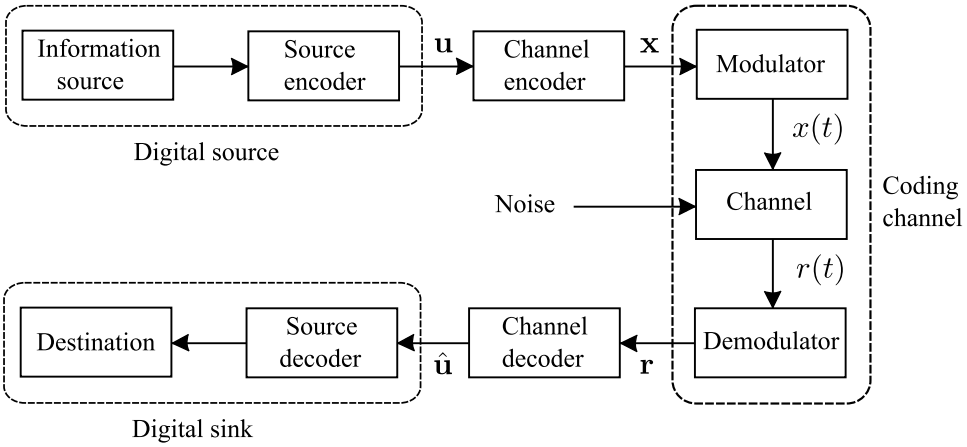


Figure 1.1: *The digital communications system.*

Another important result reported in [Sha48] is that it is possible to separate the processing of information being produced at a source into *source coding* and *channel coding* without suffering any loss in optimality; this separation is shown in Figure 1.1. The information at the source can be produced in a multitude of ways; for example, it could be a person, sound waves, a digital computer, or a number of alternatives. The job of the source encoder is primarily to transform this output into a sequence of binary digits called the *information sequence* $\mathbf{u}$, but also to reduce redundancy in the information, i.e., to minimize the amount of bits per unit of time needed to

represent the source output while maintaining enough information to reconstruct the message later. This operation is typically known as *compression*. In this thesis, we group the information source and source encoder together (and consequently the destination and source decoder) and assume that we are given an information sequence $\mathbf{u}$ directly from the digital source (and give the resulting information estimate $\hat{\mathbf{u}}$ to the digital sink). For more information about source encoding and decoding, the reader is directed to [Ber71, Gra90, CT91].

The reliable transmission that Shannon guaranteed can be achieved by *channel coding*. Here, the channel encoder is employed to add *redundancy* in a predefined mathematical way to the information sequence $\mathbf{u}$, creating a longer sequence (or *codeword*) $\mathbf{x}$ that is to be transmitted over the channel. The motivation of this operation is to use the redundancy later to help with error control when transmitting over a *noisy channel*. The channel encoder is the main focus of this thesis, and is subsequently introduced in more detail in Section 1.3.

In practice, the channel could be, for example, telephone wires, radio, mobile telephony, satellite links or storage media such as magnetic tapes, CDs, optical memory, and so on. All of these examples can be affected by different forms of 'noise'. For our purposes, we use a fixed mathematical model for the channel. Typical models are presented in Section 1.2. In this work, as shown in Figure 1.1, we group the modulator, the channel and the demodulator together. The modulator should transform each symbol of $\mathbf{x}$ into a waveform $x(t)$ of duration T seconds to be transmitted over the channel. The noise will possibly corrupt this signal, and the received waveform $r(t)$ should be transformed into a received sequence $\mathbf{r}$ to pass to the channel decoder. Thus in our simplified system, we pass the codeword $\mathbf{x}$ into the coding channel and receive only the (noisy) vector $\mathbf{r}$ out of the coding channel.

In Section 1.4, the channel decoder block is described briefly. The role of the channel decoder is to pass an estimate $\hat{\mathbf{u}}$ of the original message $\mathbf{u}$ to the digital sink, having received the noisy sequence $\mathbf{r}$. How to decode a noisy sequence is another major challenge to system designers, and many novel and diverse methods have been proposed in the literature. We will discuss later that on the AWGN channel, the way to approach the theoretical Shannon limit is to use so-called soft decisions with iterative decoding. Codes that perform within a few tenths of a decibel of this limit are appropriately named *capacity-approaching codes*. The 1990s brought the arrival of Turbo codes [BGT93, BG96] and the reintroduction of the now-implementable low-density parity-check (LDPC) codes of Gallager [Gal62], the two classes of capacity-approaching codes in use today. This thesis focuses primarily on LDPC codes, in both block and convolutional

form, and Section 1.5 provides a brief but detailed review of these codes. Turbo type codes and concatenated codes, the modern competitors of LDPC codes, are also briefly described in Section 1.6.

This chapter serves as an introduction to the field of digital communications and in particular channel coding. This should set the results of the rest of the thesis in context. We conclude this chapter in Section 1.7 with a summary of the aims and motivation behind the main contributions of the thesis and an outline of the thesis structure.

1.2 Channel models

In this section, we will describe briefly the three standard channels that are used by information theorists, namely the binary symmetric channel (BSC), the binary erasure channel (BEC) and the additive white Gaussian noise (AWGN) channel. Recall that here we are modelling the coding channel block of Figure 1.1. In this thesis we will use only the AWGN channel; however the others are included here for completeness. For full details of these and other common channels, see [CT91].

1.2.1 The binary symmetric channel

The binary symmetric channel is the simplest noisy channel model. The transition probability diagram for the BSC is illustrated in Figure 1.2.

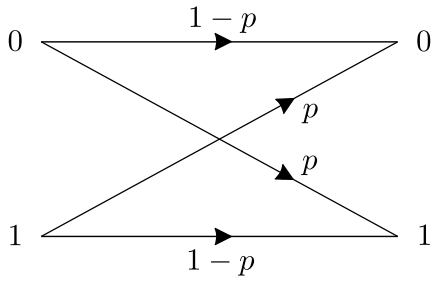


Figure 1.2: *Transition probability diagram for the binary symmetric channel (BSC).*

The input to the channel is binary. Observe from Figure 1.2 that the output is equal to the input with probability $1 - p$. However, with probability p, an error occurs over the channel and a transmitted 0 is received as a 1 and vice versa. Note that, for this channel, the receiver obtains only a binary digit and hence does not know if the received bit is reliable or not. It can easily

be shown (see, for example, [CT91, p. 187]) that the capacity $\mathcal{C}$ for this channel is given by the formula

$$\mathcal{C} = 1 + p\log_2 p + (1-p)\log_2(1-p).$$

1.2.2 The binary erasure channel

The binary erasure channel is another simple noisy channel. The main difference between this channel and the BSC is that information cannot be corrupted. Instead it is either received or it is not. The BEC is displayed in Figure 1.3.

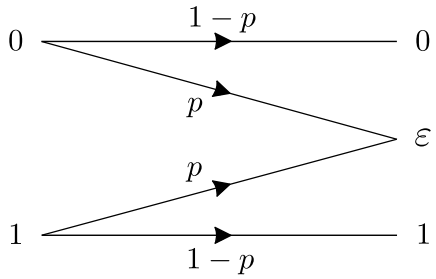


Figure 1.3: *Transition probability diagram for the binary erasure channel (BEC).*

The BEC has binary inputs like the BSC. Again, a bit is successfully transmitted over the channel with probability $1-p$. However, under this model, the bit can be *erased* with probability p. An important factor of this channel model is that there are 3 outputs. I.e., the receiver knows which bits have been erased. The capacity for the BEC is given by the formula $\mathcal{C} = 1 - p$ [CT91, p189].

1.2.3 The additive white Gaussian noise channel

In this thesis, we assume BPSK modulation when using the additive white Gaussian noise channel. The channel is modelled as a binary-input, discrete-time AWGN channel with input power constraint E_s and noise variance of σ^2. Given a codeword $\mathbf{x} = [x_0 \;\; x_1 \;\; \cdots \;\; x_{n-1}]$ of length n that is to be transmitted, the power constraint is given as:

$$\frac{1}{n}\sum_{i=0}^{n-1} x_i^2 \leq E_s\,.$$

The discrete-time binary-input AWGN channel with BPSK modulation as discussed above is illustrated in Figure 1.4.

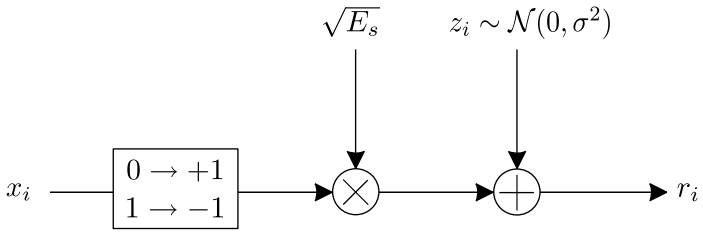


Figure 1.4: *The binary-input additive white Gaussian noise (AWGN) channel.*

For each of the n transmitted binary symbols x_i, $i = 0, \ldots, n-1$, the optimal demodulator generates the real-valued received symbol

$$r_i = (2x_i - 1)\sqrt{E_s} + z_i, \quad i = 0, \ldots, n-1,$$

where z_i corresponds to the noise.[1] Each z_i is an independent, identically distributed (i.i.d.) variable from a Gaussian distribution with mean 0 and variance σ^2. The demodulated output will generally be chosen from a number of fixed symbols designed to suit a particular system. However some systems may prefer the output as a real number. For more details on modulation/demodulation, see [Pro01, LC04]. Note that if we restrict the output to binary symbols only, the channel reduces to a BSC.

In practical situations we use a more complicated continuous-time channel model. For the scope of this thesis however, we consider only the discretised model described above. For a good discussion of the continuous-time signalling concept and the relation between discrete-time and continuous-time modelling, the reader is directed to [Goe07, Pro01]. If the channel that we are trying to model has noise formed from the superposition of a large number of i.i.d. noise sources, then the AWGN channel is a very good model to use.

1.3 Channel coding: structure and encoding

Following Shannon's seminal paper in 1948, tremendous effort has been placed in designing efficient encoding and decoding methods for transmission over noisy channels. Channel codes operate by adding redundancy to the information that is to be transmitted over the channel in a predefined mathematical way. The aim of channel coding is to then use this redundancy to extract the correct information vector from a received noisy vector. In this section, we focus

[1] If the output symbols are quantized in more than two levels, we say that the demodulator makes *soft decisions*. If the symbols are quantized in exactly two levels (i.e., as binary symbols), then this is *hard decision* demodulation. Hard and soft decision decoding will be discussed further in Section 1.4.

on the channel encoder block of Figure 1.1 and introduce two broad categories of channel codes, namely *block* and *convolutional* codes. A system employing an aptly named block code transmits a block of n symbols at every time unit which is acted on independently by the channel encoder/decoder. Convolutional codes, on the other hand, work continuously symbol by symbol.[2]

1.3.1 Block codes

We begin by defining a binary linear block code.[3] For binary codes we work over the finite field (our alphabet) $F = \text{GF}(2) = \{0, 1\}$.[4] Consider the space $V = F^n$ of all n-tuples of 0's and 1's with addition of vectors componentwise modulo 2. For example, let $n = 7$; then

$$[\,0\ 1\ 1\ 0\ 1\ 0\ 1\,] + [\,1\ 1\ 0\ 0\ 1\ 0\ 0\,] = [\,1\ 0\ 1\ 0\ 0\ 0\ 1\,].$$

Definition 1.1 *An (n, k)* ***linear code*** *is the set of all linear combinations of k linearly independent vectors of length n in V.*

Clearly then a (n, k) binary linear code has 2^k vectors or codewords. We regard each vector as a *word* and will write the row vector $\mathbf{x} = [\,x_0\ x_1\ \cdots\ x_{n-1}\,]$ in the form of a word $\mathbf{x} = x_0x_1 \ldots x_{n-1}$. It should be evident from linear algebra (see, e.g., [All97]) that such a set of codewords, C say, is actually a subspace of the vector space V; hence it can be defined by a basis. A matrix whose rows form a set of basis vectors for C is called a *generator matrix*. Note that as a subspace has more than one basis, except in very trivial circumstances, the corresponding code has more than one generator matrix, since any set of k linearly independent codewords can be chosen as rows for the generator matrix.[5]

Example 1.1. Consider the $(5, 3)$ binary linear code C with generator matrix

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

[2] Convolutional codes can be (and often are) terminated. This termination effectively results in a block code.

[3] In this thesis we will consider only linear block codes. There exist some good nonlinear block codes such as Kerdock [Ker72] and Preparata [Pre68] codes. For more information, the reader is directed to [Ple98].

[4] A finite field with q elements (where q is a power of prime p) is called a *Galois field*, named after the mathematician Évariste Galois, and is denoted GF(q). See, for example, [All91] for more details.

[5] The members of the set of generator matrices for C are called *equivalent generator matrices*.

Clearly the three chosen words are linearly independent due to the reduced row echelon structure of G. The associated code C has $2^3 = 8$ distinct codewords and can be found as the set of linear combinations of the three words forming G. Explicitly

$$C = \{00000, 10010, 10101, 11011, 11100, 00111, 01001, 01110\}. \tag{1.1}$$

Further, as C is a vector subspace, we note that the linear combination of any number of codewords from C modulo 2 results in a codeword from C. Indeed, choosing any $k = 3$ linearly independent codewords from C forms an equivalent generator matrix. For example, say $\mathbf{x}_0 = [1\ 0\ 1\ 0\ 1]$, $\mathbf{x}_1 = [1\ 1\ 0\ 1\ 1]$, and $\mathbf{x}_2 = [1\ 1\ 1\ 0\ 0]$. Then we can form

$$G_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

It is easy to check that these vectors are linearly independent and that the rowspace of G_2 forms the set C, i.e., G_2 is also a generator matrix for C. □

As a binary (n, k) linear block code has 2^k codewords, each codeword corresponds to a unique k-tuple $\mathbf{u}$, in the sense that $\mathbf{x} = u_0\mathbf{G}_0 + u_1\mathbf{G}_1 + \ldots u_{k-1}\mathbf{G}_{k-1}$ for $\mathbf{x} \in C$ where $\mathbf{G}_0, \ldots, \mathbf{G}_{k-1}$ are the k basis vectors forming the rows of G. Thus we can encode any binary *information vector* $\mathbf{u}$ of length k as a unique codeword $\mathbf{u}G = \mathbf{x} \in C$. Then the *rate* of an (n, k) linear block code is defined as $R = k/n$ in the sense that in every time unit, k information bits are encoded into a vector of length n to be transmitted over the channel.

Example 1.1 (cont.). Suppose we wish to encode the information vector $\mathbf{u} = [u_0\ \ u_1\ \ u_2] = [1\ 0\ 1]$ using the $(5, 3)$ binary linear code C of rate $R = 3/5$ defined above. We calculate

$$\mathbf{x} = \mathbf{u}G = [1\ 0\ 1] \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} = [1\ 0\ 1\ 0\ 1].$$

The codeword $\mathbf{x} = [1\ 0\ 1\ 0\ 1]$ can then be transmitted over the channel. □

We observe in the above example that the information vector $\mathbf{u}$ forms part of the codeword $\mathbf{x} = [\,\mathbf{u}\ :\ \mathbf{p}\,]$. This phenomenon happens because of the reduced row echelon structure of G.

Such a generator matrix is called *systematic*.[6] The vector $\mathbf{p}$, of length $n-k$, contains the so-called *parity-check* symbols. For an information sequence encoded using a systematic generator matrix the parity-check symbols represent the redundant information transmitted alongside the message to aid error control at the receiver. How to choose the redundant bits to improve error control over a noisy channel is one of the major challenges of code design. The notion of parity-checks leads us to another useful matrix associated with a linear block code.

It is a fact (see, e.g., [LC04, p. 70]) that if G is a $k \times n$ matrix with k linearly independent rows then there exists an $(n-k) \times n$ matrix H with the property that any vector in the row space of G is orthogonal to H, and any vector in the row space of H is orthogonal to G. In other words, the linear combinations of rows of H form the null space of G. We call H the *parity-check matrix* of the linear code C; this can be used to form an alternative definition of a linear block code.

Definition 1.2 *Let H be the parity-check matrix of an (n,k) linear block code with generator matrix G. Then the code C is defined as the set of vectors $\mathbf{x}$ that satisfy $\mathbf{x}H^T = \mathbf{0}$.*

The parity-check matrix proves helpful for error detection: if a received vector $\mathbf{r}$ does not satisfy $\mathbf{r}H^T = \mathbf{0}$ then we know an error has occured. There exists a very useful theorem for obtaining the parity check matrix from a generator matrix or vice versa.

Theorem 1.1 [Ple98, p. 9] *If an (n,k) linear code C has a generator matrix $G = [\mathbb{I}_k : A]$ in systematic form, then a parity check matrix of C is*[7] *$H = [-A^T : \mathbb{I}_{n-k}]$, where A^T is the transpose of A and, for example, $\mathbb{I}_k$ is the $k \times k$ identity matrix.*

Example 1.1 (cont.). In this running example,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \text{ and thus } -A^T = A^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

[6] Note that in general for a generator matrix of rank k to be systematic we simply require the $k \times k$ identity matrix appears in the matrix up to column permutations. The matrix is said to be in *standard systematic form* if the identity matrix exists as a left hand block.

[7] When considering binary codes, $-A^T = A^T$.

We can then form the corresponding parity-check matrix using Theorem 1.1 as follows:

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

and by calculation, $GH^T = 0_{k\times(n-k)}$. □

Note that the 2^{n-k} linear combinations of the rows of the parity-check matrix H described above form a binary $(n, n-k)$ linear code $C^\perp$. This code $C^\perp$ is called the *dual code* of C. By the properties of parity-check and generator matrices we observe that any word $\mathbf{x} \in C^\perp$ satisfies $\mathbf{x}G^T = 0$. In other words, a parity-check matrix for C is a generator matrix for $C^\perp$ and vice versa. See, for example, [LC04] for more details.

The notions of *Hamming weight* and *Hamming distance* were introduced in [Ham50]. The Hamming weight of a codeword is simply the number of non-zero positions in the codeword. The Hamming distance, or metric, $d(\mathbf{x}_0, \mathbf{x}_1)$ between two codewords $\mathbf{x}_0$ and $\mathbf{x}_1$ is the number of positions in which they differ. Hamming used these definitions, in turn, to define the minimum distance d_{min} of a linear block code C as follows:

$$d_{min} = \min\{d(\mathbf{x}_0, \mathbf{x}_1) \mid \mathbf{x}_0, \mathbf{x}_1 \in C, \mathbf{x}_0 \neq \mathbf{x}_1\}.$$

Due to the linearity of C, it can easily be shown that d_{min} is equal to the codeword with the smallest Hamming weight. For example, we see from Equation (1.1) in the running example that $d_{min} = 2$. Often, an (n, k) linear block code with minimum distance d_{min} will be referred to as an (n, k, d_{min}) linear block code. The minumum distance is an very important parameter for error-correction. For example, when considering the binary symmetric channel model, the optimal way to decode is to choose the codeword that is closest in terms of Hamming distance. Thus, an (n, k, d_{min}) linear block code transmitted over the BSC can guarantee the correct decoding of a codeword provided less than or equal to t errors occur where[8]

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

This important parameter motivated the study of algebraic coding theory, which primarily aimed to maximize d_{min} for a given n and k. This formed the main focus of channel coding theorists for over 20 years, following Shannon's famous paper.

[8]The *floor* of the real number x is the largest integer that is not larger than x and is written as $\lfloor x \rfloor$.

1.3.2 Convolutional codes

Convolutional codes were first introduced by Elias [Eli55]. The idea of convolutional coding is fundamentally different to that of block codes. The main difference of a convolutional code is that the encoder contains memory, and thus the output at any time is dependant on both the inputs at that time and a number of previous inputs.[9] As we will observe in this section, good properties of convolutional codes, such as low error probabilities and the equivalent of large minimum distances for block codes, are achieved by increasing the memory order. In this sense, convolutional codes are essentially motivated by the the concept of finding codes that optimize the average performance as a function of encoding and decoding complexity [CF07], i.e., they fall into the category of *probabilistic coding* rather than algebraic coding (this notion will be discussed further in Section 1.4). For a thorough treatment of convolutional codes, the reader is directed to [JZ99b].

An (n, k, m) convolutional encoder of rate $R = k/n$ can be viewed as a linear sequential ciruit with k inputs, n outputs and input memory m. The memory order m implies that inputs will remain in the encoder for m time units after entering. The encoder for a rate $R = 1/2$ binary convolutional code with memory order $m = 2$ is shown in Figure 1.5.

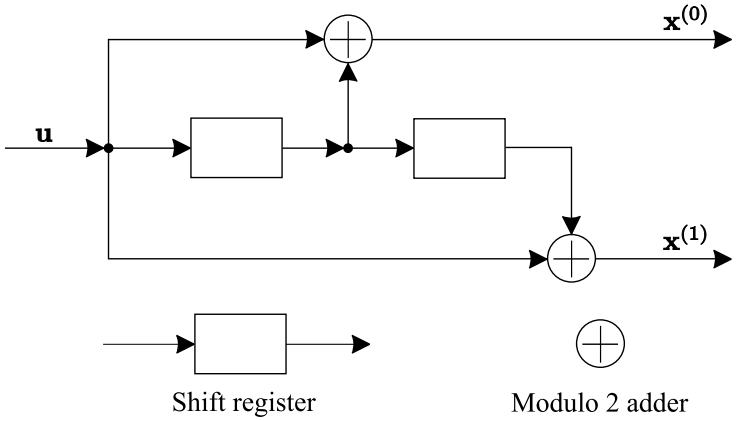


Figure 1.5: *A rate $R = 1/2$ binary convolutional encoder with memory order $m = 2$.*

A shift register receives a binary input symbol at each time unit and outputs this symbol at the next time unit. The modulo 2 adders sum the binary symbols at the inputs and output this value modulo 2 at each time unit. In practice, a modulo 2 adder with two inputs is simply an exclusive-or (XOR) gate, and an adder with more than two inputs can be implemented using

[9] Encoders that use feedback result in so-called *recursive* convolutional codes. In this case, the output at any time unit depends on infinitely many previous inputs.

a sequence of XOR gates. The encoder shown in Figure 1.5 is called a *feedforward* encoder as we have no feedback in the circuit. In order to be concise, we will not consider feedback encoders in this section. The reader is directed to [LC04] for more details and examples.

Example 1.2. Consider the $(2, 1, 2)$ binary feedforward convolutional encoder of rate $R = 1/2$ depicted in Figure 1.5. Suppose we wish to encode the infinite information sequence $\mathbf{u} = [\,1\ \ 1\ \ 0\ \ 1\ \ 0\ \cdots\,]$. The parameters of the encoding circuit according to input sequence $\mathbf{u}$ are displayed in Table 1.1.

Input		Shift register contents		Output symbols	
t	u_t	Register 1	Register 2	$\mathbf{x}^{(0)}$	$\mathbf{x}^{(1)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
-1	0	0	0	0	0
0	1	1	0	1	1
1	1	1	1	0	1
2	0	0	1	1	1
3	1	1	0	1	0
4	0	0	1	1	0
5	0	0	0	0	1
6	0	0	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 1.1: *Parameters of the convolutional encoding circuit of Example 1.2.*

We note that the output at time unit t is given as

$$x_t^{(0)} = u_t + u_{t-1} \ \text{ and } \ x_t^{(1)} = u_t + u_{t-2}.$$

As these equations hold for every time unit, we say the convolutional code is *time invariant*. From these equations we can form *generator sequences* $\mathbf{g}^{(i)}$ for each output.[10] For this example,

$$\mathbf{g}^{(0)} = [\,1\ \ 1\ \ 0\,] \ \text{ and } \ \mathbf{g}^{(1)} = [\,1\ \ 0\ \ 1\,].$$

Then the output sequences can be obtained as

$$\mathbf{x}^{(0)} = \mathbf{u} \circledast \mathbf{g}^{(0)} = [\,1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \cdots\,]$$
$$\mathbf{x}^{(1)} = \mathbf{u} \circledast \mathbf{g}^{(1)} = [\,1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \cdots\,],$$

[10] Alternatively, the generator sequences can be obtained for this example by observing the two impulse responses for $m + 1$ time units when inputting information sequence $\mathbf{u} = [\,1\ \ 0\ \ 0\ \cdots\,]$.

where $\circledast$ denotes discrete convolution and all operations are modulo 2.[11] The two sequences $\mathbf{x}^{(0)}$ and $\mathbf{x}^{(1)}$ are then multiplexed into a single codeword sequence as follows

$$\mathbf{x} = [\, x_0^{(0)} \;\; x_0^{(1)} \;\; x_1^{(0)} \;\; x_1^{(1)} \;\; \cdots \,] = [\, 1 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \cdots],$$

that can be transmitted over the channel. □

For a general rate $R = k/n$ binary feedforward encoder with memory order[12] m, each of the n outputs will have a generator sequence according to each one of the k inputs; i.e., for each output $j = 0, \ldots, n-1$, there exist k generator sequences $\mathbf{g}_0^{(j)}, \mathbf{g}_1^{(j)}, \ldots, \mathbf{g}_{k-1}^{(j)}$ of length $m+1$.

These generator sequences can be used to form a *generator matrix* G for the convolutional code, where

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & & \\ & G_0 & G_1 & G_2 & \cdots & G_m & \\ & & G_0 & G_1 & G_2 & \cdots & G_m \\ & & & \ddots & \ddots & \ddots & \end{bmatrix},$$

where blank space corresponds to zero matrices of the appropriate size and each G_l is a $k \times n$ matrix given as

$$G_l = \begin{bmatrix} g_{0,l}^{(0)} & g_{0,l}^{(1)} & \cdots & g_{0,l}^{(n-1)} \\ g_{1,l}^{(0)} & g_{1,l}^{(1)} & \cdots & g_{1,l}^{(n-1)} \\ \vdots & \vdots & & \vdots \\ g_{k-1,l}^{(0)} & g_{k-1,l}^{(1)} & \cdots & g_{k-1,l}^{(n-1)} \end{bmatrix}.$$

Then for an information sequence $\mathbf{u} = [\, u_0^{(0)} \;\; u_0^{(1)} \;\; \cdots \;\; u_0^{(k-1)} \mid u_1^{(0)} \;\; u_1^{(1)} \;\; \cdots \;\; u_1^{(k-1)} \mid \cdots]$, the codeword $\mathbf{x} = [\, x_0^{(0)} \;\; x_0^{(1)} \;\; \cdots \;\; x_0^{(n-1)} \mid x_1^{(0)} \;\; x_1^{(1)} \;\; \cdots \;\; x_1^{(n-1)} \mid \cdots]$ is given as $\mathbf{x} = \mathbf{u}G$.

Example 1.2 (cont.). Recall that the generator sequences for this rate $R = 1/2$ convolutional encoder were given as $\mathbf{g}^{(0)} = [\, 1 \;\; 1 \;\; 0 \,]$ and $\mathbf{g}^{(1)} = [\, 1 \;\; 0 \;\; 1 \,]$. The associated generator matrix is

[11] To obtain the symbols $x_l^{(j)}$ for $l \geq 0$ using the convolution operation, we calculate $x_l^{(j)} = \sum_{i=0}^{m} u_{l-i} g_i^{(j)}$ for outputs $j = 1, 2$, where $u_{l-i} \triangleq 0$ for all $l < i$ and all operations are modulo 2. Similar expressions exist for general rate $R = k/n$ convolutional encoders. The reader is directed to [LC04] for further details and examples.

[12] One needs to take care to calculate the memory order for more complicated convolutional encoders than the one presented here. See [LC04, p. 459] for a formal definition.

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & & \\ & G_0 & G_1 & G_2 & \\ & & \ddots & \ddots & \ddots \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & & & & & & \\ & & 1 & 1 & 1 & 0 & 0 & 1 & & & & \\ & & & & 1 & 1 & 1 & 0 & 0 & 1 & & \\ & & & & & & 1 & 1 & 1 & 0 & 0 & 1 \\ & & & & & & & & \ddots & \ddots & \ddots & \end{bmatrix}.$$

Thus, for information sequence $\mathbf{u} = [\,1\ 1\ 0\ 1\ 0\ \cdots\,]$, we calculate

$$\mathbf{x} = \mathbf{u}G = [\,1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ \cdots\,],$$

which agrees with the previous calculation found by convolution. □

Recall that when discussing block codes we introduced the notion of *systematic* and *nonsystematic* generator matrices. A generator matrix G of an (n, k) linear block code is in systematic form if the $k \times k$ identity matrix appears in the matrix up to column permutations. This results in the information vector symbols existing unchanged in the codeword to be transmitted. The same phenomenon occurs for convolutional generator matrices. An identity matrix exists in the generator matrix of a convolutional code precisely when each of the k inputs of the associated rate $R = k/n$ convolutional encoder are directly output with no modification. We observe that this is not the case for the generator matrix and associated encoder for Example 1.2. Thus we say the encoder in Figure 1.5 is a $(2, 1, 2)$ *binary nonsystematic feedforward encoder*. For examples of systematic convolutional encoders see, e.g., [LC04].

The notion of a parity-check matrix of a convolutional code is the same as that for a block code. Hence Theorem 1.1 can be adapted to allow one to obtain a parity-check matrix H for a systematic convolutional generator matrix. We may then formulate the alternative definition of a convolutional code as the set of infinite sequences $\mathbf{x}$ that satisfy the equation

$$\mathbf{x}H^T = \mathbf{0}.$$

Another similarity between block codes and convolutional codes is the concept of a minimum distance between codewords/code sequences. For convolutional codes, the (minimum) *free distance* d_{free} is defined as the minimum Hamming distance between any two finite length code sequences, i.e., sequences with only a finite number of non-zero symbols. If one of the two sequences is shorter than the other, an appropriate number of zeros is added to make the

lengths equal. Due to the linearity of convolutional codes, d_{free} is equal to the minimum Hamming weight over all non-zero code sequences $\mathbf{x} \in C$. Similarly to the case of block codes, when considering transmission over the BSC, a convolutional code can guarantee the correct decoding of a received noisy sequence with at most $t = \lfloor (d_{free} - 1)/2 \rfloor$ errors.

1.4 Channel coding: decoding

In this section, we concentrate on the channel decoder block of the digital communications model depicted in Figure 1.1. The role of the channel decoder is to pass an estimate $\hat{\mathbf{u}}$ of the original message $\mathbf{u}$ to the digital sink, having received the noisy sequence $\mathbf{r}$. This estimate is clearly very important to the system and thus the reliability of a coding scheme depends not only on the particular code employed but also the chosen decoding algorithm.

1.4.1 Decoding methods

When transmitting over the AWGN channel, the received noisy sequence $\mathbf{r}$ is real-valued. In order to approach the Shannon limit with coding schemes over this channel it has been found to be necessary to use so-called *soft decision decoding* [Pro01, LC04].[13] When using soft decision decoding, the decoder receives not only the 1 or 0 (hard) decision but also a measure of the reliability of the channel output. Here, clearly the decoder must be able to receive more than 2 different inputs. This will complicate decoder implementation, but it can be shown (see for example [LC04]) that we can gain two to three decibels in transmit power. In this section, we review the basic concepts of two of the standard decoding techniques, namely *maximum likelihood* (ML) decoding and *maximum a posteriori probability* (MAP) decoding, and highlight their relationship.

Recall that there is a one-to-one correspondence between the information sequence $\mathbf{u}$ and the codeword to be transmitted $\mathbf{x}$. The same correspondence follows for the decoder estimates $\hat{\mathbf{x}}$ and $\hat{\mathbf{u}}$. Suppose that the decoder receives the noisy vector $\mathbf{r}$. Then the probability of decoder error can be written as

$$\mathbb{P}(e) = \sum_{\mathbf{r}} \mathbb{P}(e|\mathbf{r})\mathbb{P}(\mathbf{r}) = \sum_{\mathbf{r}} \mathbb{P}(\hat{\mathbf{x}} \neq \mathbf{x}|\mathbf{r})\mathbb{P}(\mathbf{r}),$$

[13] The first recorded algorithm employing soft decision decoding was so-called Wagner decoding [SB54].

where $\mathbb{P}(\mathbf{r})$ is the probability of receiving sequence $\mathbf{r}$. Of course, the probability $\mathbb{P}(\mathbf{r})$ is independent of the decoder. Thus to decode optimally, the decoder should minimize $\mathbb{P}(\hat{\mathbf{x}} \neq \mathbf{x}|\mathbf{r})$, or alternatively maximize $\mathbb{P}(\hat{\mathbf{x}} = \mathbf{x}|\mathbf{r})$. To use the ML rule, one should choose the $\hat{\mathbf{x}} \in C$ such that $\mathbb{P}(\mathbf{r}|\hat{\mathbf{x}})$ is maximum among all $\mathbb{P}(\mathbf{r}|\tilde{\mathbf{x}})$ for $\tilde{\mathbf{x}} \in C$. By following this approach, the word error probability will be minimized, provided that each information sequence $\mathbf{u}$ (and hence each codeword $\mathbf{x}$) is equally likely.

If this condition cannot be assumed, then the word error probability will be minimized by choosing the codeword estimate $\hat{\mathbf{x}} \in C$ that maximizes the *a posteriori probability* (APP) $\mathbb{P}(\hat{\mathbf{x}}|\mathbf{r})$, i.e., one should choose the estimate $\hat{\mathbf{x}} \in C$ such that $\mathbb{P}(\hat{\mathbf{x}}|\mathbf{r})$ is maximum among all $\mathbb{P}(\tilde{\mathbf{x}}|\mathbf{r})$ for $\tilde{\mathbf{x}} \in C$. Using Bayes' Law, we obtain an expression for the MAP rule

$$\mathbb{P}(\mathbf{x}|\mathbf{r}) = \frac{\mathbb{P}(\mathbf{r}|\mathbf{x})\mathbb{P}(\mathbf{x})}{\mathbb{P}(\mathbf{r})}.$$

Again, we arrive at the conclusion that if each codeword is equally likely then maximizing $\mathbb{P}(\mathbf{x}|\mathbf{r})$ is equivalent to maximizing $\mathbb{P}(\mathbf{r}|\mathbf{x})$, i.e., the MAP rule is equivalent to the ML rule. As an alternative to minimizing the error probability of an entire block or sequence, we can define a symbol by symbol decoder. Here, the decoder should minimize the probability $\mathbb{P}(\hat{x}_i \neq x_i|\mathbf{r})$. A symbol by symbol soft decision MAP decoder maximizes the APP $\mathbb{P}(\hat{x}_i|\mathbf{r})$. A hard decision for $\hat{x}_i$ is a probability distribution such that $\mathbb{P}(\hat{x}_i|\mathbf{r})$ can equal either a one or a zero.

Many of the early decoding schemes for algebraic block codes used only hard decision decoding based on the algebraic structure of the code. The metric used here to make the hard decisions is the Hamming distance and thus the goal of algebraic coding theory was primarily to maximize the minimum distance d_{min} for a given block length n and dimension k. The motivation behind probabilistic coding theory, on the other hand, was to focus on the average performance of codes with encoding and decoding complexity in mind. In view of good performance, probabilistic decoders tend to use soft information both at the input and during the decision making process. Convolutional codes were the early focus of probabilistic coding theorists, and their natural trellis structure[14] led to a very efficient ML decoding algorithm that can use soft decisions, called the *Viterbi algorithm* [Vit67]. Probabilistic coding theory was inspired by Shannon's original probabilistic approach to coding and it has proven to be the way to obtain modern capacity-approaching codes by using long random-like codes with iterative soft-input soft-output (SISO) decoding algorithms.

[14] See, for example, [LC04] for definitions and examples of codes with trellis representations.

1.4.2 Modern iterative decoding techniques

In his doctoral thesis, Gallager developed a (hard decision) iterative *bit-flipping* (BF) decoding algorithm for low-density parity-check codes [Gal63]. LDPC codes are now often decoded with an implementation of the popular *sum-product algorithm* (SPA) (see [KFL01]), which is a symbol by symbol SISO iterative message-passing APP decoding algorithm based on *belief propagation* (BP) [Pea82]. The SPA, whilst being a sub-optimal decoding method in practice, has proven to be extremely efficient and effective for decoding LDPC codes. The reliability measures (extrinsic information) computed by the algorithm for each symbol at the end of an iteration are used as inputs for the next iteration. After each iteration, hard decisions are taken and these decisions are then tested by evaluating the parity-check equations. If a codeword is found, the decoding iterations are stopped; otherwise they continue until a pre-specified maximum number of iterations is reached. In the latter case one would declare decoding failure, or just use the hard decision even if this implies bit errors. For more details, see [Mac99,LC04].

In 1974, Bahl, Cocke, Jelinek, and Raviv (BCJR) proposed a SISO MAP iterative decoding algorithm for convolutional codes (or block codes with a trellis structure) [BCJR74]. The BCJR algorithm is more complex than the Viterbi algorithm and, as such, it is generally not employed for decoding convolutional codes. However, the iterative SISO technique means that it is ideal for decoder implementation of capacity-approaching turbo codes. This algorithm is now understood to be an implementation of the SPA on a trellis [KFL01]. LDPC and turbo codes will be discussed further in Sections 1.5 and 1.6 respectively.

To measure the error performance of a code we usually plot the *bit-error probability*, or *bit-error rate* (BER), against the signal-to-noise ratio (SNR). Figure 1.6 shows the typical dynamics of a capacity-approaching code under iterative message-passing decoding compared to an uncoded BPSK transmission. In this figure, the SNR is given as the ratio of the energy per transmitted information bit E_b to the one-sided power spectral density (PSD) N_0 of the channel noise. Another common measure is to plot the *block* or *frame-error rate* (FER) against the SNR. There are two limits on the performance of the code, the *Shannon limit* and the *iterative decoding threshold*.

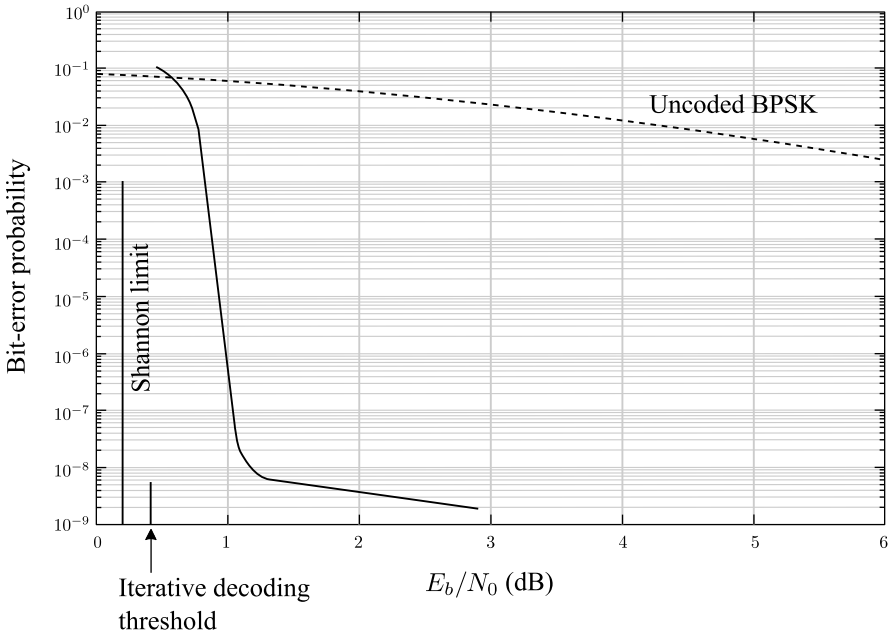


Figure 1.6: *Typical error performance of a modern capacity-approaching code under iterative message-passing decoding.*

The Shannon limit defines channel capacity and is the ultimate limit of error performance. This limit is a direct result of the physical properties of the channel. The iterative decoding threshold differs in that it considers both the structure of a code and the (iterative) decoding algorithm employed. In [RU01a], Richardson and Urbanke presented a technique to obtain the iterative decoding threshold using a technique called *density evolution*. Density evolution tracks the probability density functions of the messages in the Tanner graph of an LDPC code[15] and determines whether or not iterative message-passing decoding will converge for a given SNR. Like the Shannon limit, the iterative decoding threshold limits the performance of an LDPC code. By definition, the iterative decoding threshold cannot exceed the Shannon limit, and if the threshold is equal to the Shannon limit then the code is said to be *capacity-achieving*. Long irregular LDPC codes have been designed that almost achieve capacity [CFRU01].

Capacity-approaching codes have the desirable property that after a certain SNR there is a steep slope called the *waterfall* region of the curve. Here, the error performance improves dramatically with little increase in power. However, we also observe that when the SNR exceeds a particular value the curve typically levels off, forming the so-called *error floor* region. Here we achieve a relatively small improvement in performance for a large increase of power. For

[15]Density evolution is not restricted to LDPC codes. It can be employed for other graph-based codes with associated message-passing decoding. However, for codes with complicated structures such as turbo codes, the procedure becomes intractable.

capacity-approaching codes, the error floor typically occurs at very low BERs and thus cannot be explored by conventional simulations. Recently, the error events contributing to this error floor have been classified as *trapping sets*, graphical sub-structures existing in the Tanner graph of LDPC codes [Ric03]. In this paper, Richardson also proposed a two-stage technique based on trapping sets to predict the error floor performance of LDPC codes. This has sparked a wave of research, as the error floor is a crucial consideration for the system designer. Trapping sets will be discussed further in Chapter 4, and new results on the trapping sets of LDPC convolutional codes are the focus of Chapter 6.

1.5 Low-density parity-check (LDPC) codes

Along with turbo codes, which will be described briefly later in Section 1.6, low-density parity-check block codes form a class of codes which approach the (theoretical) Shannon limit. LDPC codes were first introduced in the 1960s by Gallager [Gal62, Gal63]. However, they were considered impractical at that time, and very little related work was done until Tanner provided a graphical interpretation of the parity-check matrix in 1981 [Tan81b]. More recently, in his Ph.D. Thesis, Wiberg revived interest in LDPC codes and further developed the relation between Tanner graphs and iterative decoding [Wib96].

1.5.1 LDPC block codes

The 1990s brought the rediscovery of LDPC codes from two independent sources: Sipser and Spielman proposed LDPC codes based on so-called *Expander graphs* [SS96] and the second author proposed a linear-time encoding and decoding algorithm with reasonable error performance [Spi96]; and independently, MacKay and Neil proposed *MN* codes [MN97] with a similar structure to that of Gallager codes [Gal62], and showed empirically that near Shannon limit performance could be achieved by codes with very sparse matrices and an iterative decoding belief propagation decoding algorithm [Mac98, Mac99]. Following these important contributions, long LDPC codes using belief propagation decoding have been shown to achieve near capacity performance (see, e.g., [RSU01, RU01a, KLF01, LMSS01]). See, in particular, the noteworthy achievement of designing LDPC codes just 0.0045 dB from the capacity limit of the binary input AWGN channel [CFRU01].

LDPC codes are defined using the second of our approaches to linear codes, namely as the set of sequences $\mathbf{x}$ forming the null space of a parity-check matrix H.

Definition 1.3 *A* $(\mathbf{J}, \mathbf{K})$***-regular low-density parity-check (LDPC) code*** *is defined as the null space of a parity-check matrix* H *where each row of* H *consists of exactly* K *ones, each column of* H *consists of exactly* J *ones, and both* J *and* K *are small compared with the number of rows in* H*.*

Example 1.3. Consider the parity-check matrix

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}_{6\times 12}.$$

The parity-check matrix H has $K = 4$ ones in every row, $J = 2$ ones in every column, and is of full rank. Thus, the set of all binary vectors $\mathbf{x}$ of length $n = 12$ satisfying the equation $\mathbf{x}H^T = \mathbf{0}$ form a $(2, 4)$-regular LDPC block code of rate $R = 1/2$.[16] □

In 1981, Tanner proposed a graphical interpretation of a code by means of a bipartite graph, subsequently called a *Tanner graph* [Tan81b].[17] In the Tanner graph of a parity-check matrix H, each column of H corresponds to a *variable node*, and each row corresponds to a *check* or *constraint node*. If position (i, j) of H is equal to one, then check node i should be connected to variable node j in the Tanner graph, otherwise, there is no edge connecting these nodes. Figure 1.7 shows the Tanner graph of the $(2, 4)$-regular LDPC code of Example 1.3.

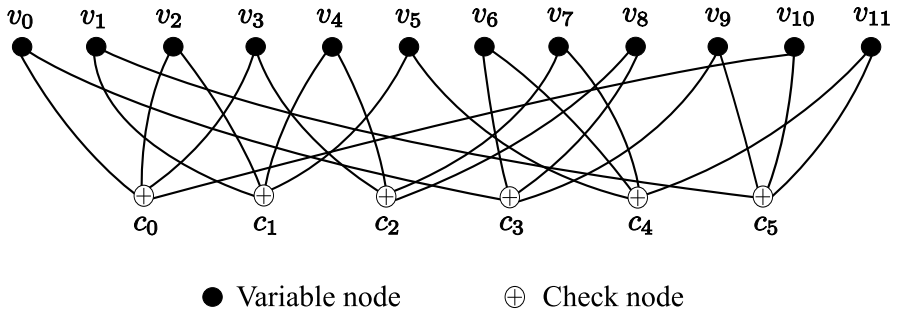


Figure 1.7: *The Tanner graph of a* $(2, 4)$*-regular low-density parity-check code.*

[16] The parity-check matrix of Example 1.3 is much shorter and denser than that of a typical low-density parity-check matrix. Parity-check matrices of LDPC codes with good error control performance typically have a very sparse parity-check matrix and long block length [Mac99].

[17] A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets A and B such that every edge of the graph connects a vertex in set A to a vertex in set B.

It has been shown in the literature that long *irregular* LDPC block codes have extremely good error control performance [LMSS01, RSU01, RU01a, CFRU01]. An irregular LDPC code simply relaxes the LDPC constraints concerning row and column weight. Many such irregular codes use the Tanner graph for design and construction.

Recall that a major factor in error control performance is the minimum distance of the chosen code. In his thesis, Gallager showed that there exist so-called *asymptotically good* (J, K)-regular LDPC code ensembles [Gal63]. A code ensemble is a set of codes which share certain properties (such as common parameters J and K in this case). An asymptotically good ensemble has the desireable property that the minimum distance d_{min} grows linearly with block length n for an average member. Thus we can define the constant *minimum distance growth rate* for such an ensemble as $\delta_{min} = d_{min}/n$. This result is remarkable, as many good code ensembles that are used in practice do not have this property. However, the distance growth rate of Gallager codes is typically not as good as the growth rate indicated by the Gilbert-Varshamov bound [Gil52, Var57] which, for any given rate, proves the existence of at least one code with a particular growth rate. Thus, we hypothesise that LDPC codes ensembles constructed in the way proposed by Gallager are not as good as a randomly constructed code ensemble. However it is possible to decode long length LDPC codes computationally, which is not currently the case for randomly constructed codes.

1.5.2 LDPC convolutional codes

LDPC convolutional codes, the convolutional counterparts of LDPC block codes, were introduced by Jiménez-Felström and Zigangirov in [JZ99a], and they have been shown to have certain advantages compared to LDPC block codes of the same complexity [CPBZ06, CPJD07].[18]

We start with a brief definition of a rate $R = b/c$ binary LDPC convolutional code C (see [JZ99a] for full details). An infinite code sequence $\mathbf{x}$ of an LDPC convolutional code satisfies the equation

$$\mathbf{x}H^T = \mathbf{0},$$

[18] The basic concepts of an LDPC convolutional code were independently described earlier in a patent application by Tanner [Tan81a].

where H^T is the *syndrome former matrix* (i.e., transposed parity-check matrix) and

$$H = \begin{bmatrix} H_0(0) & & & & & \\ H_1(1) & H_0(1) & & & & \\ \vdots & \vdots & & \ddots & & \\ H_{m_s}(m_s) & H_{m_s-1}(m_s) & \cdots & H_0(m_s) & & \\ & H_{m_s}(m_s+1) & H_{m_s-1}(m_s+1) & \cdots & H_0(m_s+1) & \\ & & \ddots & \ddots & & \ddots \end{bmatrix}$$

is the parity-check matrix of the convolutional code C. The submatrices $H_i(t), i = 0, 1, \cdots, m_s$, $t \geq 0$, are binary $(c-b) \times c$ submatrices, given by

$$H_i(t) = \begin{bmatrix} h_i^{(1,1)}(t) & \cdots & h_i^{(1,c)}(t) \\ \vdots & & \vdots \\ h_i^{(c-b,1)}(t) & \cdots & h_i^{(c-b,c)}(t) \end{bmatrix},$$

that satisfy the following properties:

1. $H_i(t) = 0_{(c-b)\times c},\ i < 0$ and $i > m_s,\ \forall\, t$.

2. There is a t such that $H_{m_s}(t) \neq 0_{(c-b)\times c}$.

We call m_s the *syndrome former memory* and $\nu_s = (m_s+1)c$ the *decoding constraint length*. These parameters determine the width of the nonzero diagonal region of H. The sparsity of the parity-check matrix is ensured by demanding that its rows have very low Hamming weight, i.e., $w_H(\mathbf{h}_i) << (m_s+1) \cdot c,\ i > 0$, where $\mathbf{h}_i$ denotes the i-th row of H. The code is said to be *regular* if its parity-check matrix H has exactly J ones in every column and, starting from row $(c-b)m_s + 1$, K ones in every row. The other entries are zeros. We refer to a code with these properties as an (m_s, J, K)-regular LDPC convolutional code, and we note that, in general, the code is time-varying and has rate $R = b/c = 1 - J/K$.[19] A rate $R = b/c$, (m_s, J, K)-regular time-varying LDPC convolutional code is periodic with period T if $H_i(t)$ is periodic, i.e., $H_i(t) = H_i(t+T), \forall\, i, t$, and if $H_i(t) = H_i, \forall\, i, t$, the code is called time-invariant.

[19] If linearly dependent rows exist in the parity-check matrix then the code will have rate $R \geq 1 - J/K$.

An LDPC convolutional code is called *irregular* if its row and column weights are not constant. The notion of degree distribution is used to characterize the variations of check and variable node degrees in the Tanner graph corresponding to an LDPC convolutional code. Similarly to the case of block codes, optimized degree distributions have been used in the literature to design LDPC convolutional codes with good iterative decoding performance (see, e.g., [SSCF03, RKZ06, PZC06, TZF07]).

1.6 Concatenated and turbo type codes

The idea of concatenated codes was first presented by Forney [For66]. Here, two linear block codes were essentially implemented one after another, with similar serial decoding being performed in reverse. The main idea behind this is that the two shorter codes should be easy to encode and decode but the resulting concatenated code will be longer and much more powerful. Turbo codes, introduced by Berrou, Glavieux and Thitimajshima [BGT93, BG96], shocked the coding community by displaying excellent error control performance and reasonable decoding complexity. Turbo codes are a form of concatenated code, but contrary to earlier concatenated codes, turbo codes are *parallel* concatenated codes involving two encoders working in tandem. The good performance of turbo codes can be attributed to the "random-like" structure of the code, as the encoder uses a long interleaver and feedback. However, turbo codes do not have good minimum distance properties. In fact, it has been shown that the minimum distance of a turbo code grows only logarithmically with interleaver length [Bre04], and thus these codes are not asymptotically optimum (unlike some ensembles of asymptotically good LDPC codes), implying the error floor cannot be avoided [CF07].

"Turbo-like" codes called Repeat-Accumulate (RA) codes were proposed by Divsalar, Jin and McEliece in 1998 [DJM98]. These codes were generated by serial concatenation of a repetition code, a long interleaver and a simple convolutional encoder. The advantage of this construction is that one is able to develop a general theory for such codes. Subsequent advances include Irregular Repeat Accumulate (IRA) codes [JKM00] and Accumulate Repeat Accumulate (ARA) codes [ADY07]. These codes also possess sublinear minimum distance, but similar methods have been used to create asymptotically good code ensembles [DDJ06]. These codes all have a simple graphical representation which will be discussed further in Chapter 4, and later used in Chapters 5 and 6.

1.7 Thesis outline and motivation

Motivated primarily by Theorem 1.1, which tells us how to obtain generator matrices and parity-check matrices from one another (after using Gaussian elimination if necessary), in **Part I** of the thesis we provide an efficient alternative to this theorem for codes with particular diagonal structures. This method uses polynomials and associated algebra similar to that used for cyclic codes. In **Chapter 2**, the theory of cyclic codes necessary to understand the contributions of Chapter 3 is presented. Moreover, we review candidate codes with similar diagonal structure that can be used by the proposed method for such codes and describe the existing alternative encoding methods.

In **Chapter 3**, an efficient new method to obtain a generator matrix G from certain types of parity-check matrices H with a so-called defective cyclic block structure is proposed. This novel approach describes parity-check and generator matrices in terms of polynomials. Moreover, using this polynomial algebra, we show that efficient ways to implement the scheme can be devised. In addition, this method is interesting as such, as it allows one to convert H into G without a systematic encoder in between. The method can be used to find sparse generator matrices for LDPC block codes, and the procedure can be implemented efficiently due to the polynomial representation of the matrix H. Further, quasi-cyclic type structures emerging from the non-binary version of the algorithm allow us to consider LDPC convolutional codes (which have an inherent diagonal structure) as candidates, due to the close connections between such block codes and LDPC convolutional codes.

This leads us logically to **Part II** of the thesis, which focuses on LDPC convolutional codes. Motivated by Gallager's result on the minimum distance growth rate of regular LDPC ensembles (detailed at the end of Section 1.5.1), we wish to obtain similar results for LDPC convolutional codes. Protograph-based LDPC block codes [Tho03] provide the framework for this theory. In **Chapter 4** we review previous results and theory concerning the minimum distance and performance of protograph-based codes. The material presented in this chapter is essential for the understanding of Chapters 5 and 6.

In **Chapter 5**, we present new methods to construct LDPC convolutional codes based on protographs. Then, asymptotic methods are used to form lower bounds on the ratio of free distance to constraint length for several ensembles of asymptotically good, protograph-based LDPC convolutional codes, i.e., ensembles that have the property of free distance growing linearly

with constraint length. These results mirror Gallager's results for regular LDPC block code ensembles. In addition, lower bounds on the free distance growth rate are found for two irregular ensembles of practical interest. Finally, it is shown for all the ensembles considered that the free distance to constraint length ratio of the LDPC convolutional codes exceeds the minimum distance to block length ratio of corresponding LDPC block codes.

In **Chapter 6**, we use the framework of protographs again, this time in the context of the study of the error floor of the asymptotically good LDPC convolutional codes derived in Chapter 5. Here, trapping sets of protograph-based LDPC convolutional codes are analysed and we use asymptotic methods to calculate a lower bound for the trapping set growth rates for several ensembles of asymptotically good protograph-based LDPC convolutional codes. Thus showing that the size of the smallest non-empty trapping set (which dominates performance in the error floor region) grows linearly with constraint length for these codes. These results can then be used to predict where the error floor will occur for such codes under iterative message-passing decoding.

Finally, in **Chapter 7**, some concluding remarks are made and recommendations for future research are proposed.

Part I

A New Method of Encoding Block Codes with Polynomials

Chapter 2
Background: Block Codes with Diagonal Structures in the Parity-Check Matrix

Abstract

Cyclic codes, and other such structured codes, have been shown to have good error-correcting properties and efficient encoder and decoder designs. In this chapter, we review previous results from the literature that are essential to put the contributions of Chapter 3 in context. In particular, we focus on block codes with useful encoding structures.

2.1 Introduction

In this chapter we primarily discuss results in the literature on block codes with sliding diagonal structures. As discussed in Section 1.7, we are motivated to study such codes because, in particular, the underlying structure helps make encoding and decoding easier [Meg61]. We begin in Section 2.2 by exploring cyclic codes, the most famous and extensively studied type of code with this structure, and include a brief summary of quasi-cyclic codes in section 2.2.2. In Section 2.3, so-called polynomial codes are introduced which, while not necessarily cyclic, contain several important cyclic codes as subclasses. Non-binary codes and code construction methods are introduced in Section 2.4, with particular attention being paid to codes with the important diagonal structure. Finally, in Section 2.5, we discuss various existing encoding schemes for cyclic and quasi-cyclic codes, and also unstructured LDPC codes. The material contained in this chapter provides a background for setting the new results of Chapter 3 in context.

2.2 Cyclic codes

Cyclic codes were first presented in 1957 by Prange [Pra57]. The important structure of cyclic codes leads to efficient encoding and decoding mechanisms [Meg61], and hence cyclic codes are among the most studied of all codes. Cyclic codes include the important family of Bose-Chaudhuri-Hocquenghem (BCH) codes [Hoc59] [BR60] (of which Reed-Solomon (RS) codes [RS60] can be viewed as a subclass) and finite geometry codes (see [KLF01] and references there for a good review of these codes). In addition, Hamming [Ham50], Golay [Gol49] and shortened Reed-Muller (RM) [Ree54, Mul54] codes can be put into cyclic form. Cyclic codes are also building blocks for other important codes such as Kerdock [Ker72], Preparata [Pre68] and Justesen [Jus72] codes.

2.2.1 The structure of cyclic codes

To begin, we define what is meant by a cyclic code.

Definition 2.1 *A linear block code C of length n over a field F is called* ***cyclic*** *if for any codeword $(c_0 c_1 \dots c_{n-1}) \in C$ we also have $(c_{n-1} c_0 c_1 \dots c_{n-2}) \in C$.*

Let C be a cyclic block code of block length n over a field F. There exists a unique monic polynomial $g(t)$ over the field F such that $g(t)$ *generates* C, denoted by $C = \langle g(t) \rangle$. By this it is meant that C is viewed as a subcode of the factor ring $R_n = F[t]/\langle t^n - 1 \rangle$ (considered as a vector space over F) and in R_n the code C is generated as an ideal by $g(t)$ [MS77, pp. 188-190]. Hence we call $g(t)$ the *generator polynomial* of C.

Let $g(t) = g_0 + g_1 t + \ldots + g_{n-k} t^{n-k}$, $g_0 = g_{n-k} = 1$, where $0 < k < n$. Then a generator matrix G for the code C is formed using $g(t)$, in the following way:

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_{n-k} & & \\ & \ddots & \ddots & & \ddots & \\ & & g_0 & g_1 & \cdots & g_{n-k} \end{bmatrix}_{k \times n}, \tag{2.1}$$

where blank spaces correspond to zeros.

Due to the structure of a cyclic code, there also exists a unique monic polynomial

$$h(t) = \frac{t^n - 1}{g(t)}$$

over F of degree k which we call the *check* or *parity-check polynomial*. Suppose $h(t) = h_0 + h_1 t + \ldots + h_k t^k$. Then it can be shown (see, e.g., [MS77, pp. 194-196]) that a parity-check matrix H for C is given by:

$$H = \begin{bmatrix} & h_k & \ldots & h_1 & h_0 \\ \iddots & & & \iddots & \iddots & \\ h_k & \ldots & h_1 & h_0 & & \end{bmatrix}_{(n-k) \times n}. \tag{2.2}$$

Note that $g_{n-k} = h_k = 1$, $g_0 = h_0 = 1$. This implies that the code C has dimension k and the dual code $C^\perp$ has dimension $n - k$.

Example 2.1. Consider the polynomial $g(t) = 1 + t^2 + t^3$ over the field GF(2). Then $g(t)$ defines a binary $(7, 4)$ cyclic block code C over GF(2). We form a generator matrix from (2.1) as follows:

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}_{k \times n = 4 \times 7}.$$

The resulting $2^k = 16$ codewords can be partitioned into 4 completely cyclic sets

$$\begin{aligned} C =& \{0000000\} + \{1011000, 0101100, 0010110, 0001011, 1000101, 1100010, 0110001\} \\ &+ \{1110100, 0111010, 0011101, 1001110, 0100111, 1010011, 1101001\} \\ &+ \{1111111\}, \end{aligned}$$

thus C satisfies Definition 2.1. We calculate

$$h(t) = \frac{t^n - 1}{g(t)} = \frac{t^7 - 1}{1 + t^2 + t^3} = 1 + t^2 + t^3 + t^4,$$

and can form the corresponding parity-check matrix

$$H = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}_{n-k \times n = 3 \times 7}$$

from equation (2.2). As H is of full rank, we observe that the dual code $C^\perp$ has dimension $n - k = 3$. This code is generated by $h(t)$, and each of the $2^{n-k} = 8$ codewords can be seen to be a cyclic shift of one another. □

2.2.2 Quasi-cyclic codes

As discussed in Section 2.2.1, cyclic codes possess full cyclic symmetry, i.e., cyclically shifting a codeword any number of positions, either left or right, results in another codeword. This structure again enables encoding and decoding to be performed using simple shift registers and logic circuits.

Definition 2.2 *A **quasi-cyclic code** is a linear code for which cyclically shifting a codeword a fixed number $n_0 \neq 1$ (or a multiple of n_0) of symbol positions either to the right or the left results in another codeword.*

It is clear that for $n_0 = 1$, a quasi-cyclic (QC) code is a cyclic code.

Example 2.2 [LC04, p. 185]. Consider the $(9, 3)$ code generated by the following generator matrix:

$$G = \left[\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{array}\right].$$

The 8 codewords of this $(9, 3)$ QC code are displayed in Table 2.1.

$\mathbf{x}_0$	0	0	0	0	0	0	0	0	0
$\mathbf{x}_1$	1	1	1	1	0	0	1	1	0
$\mathbf{x}_2$	1	1	0	1	1	1	1	0	0
$\mathbf{x}_3$	1	0	0	1	1	0	1	1	1
$\mathbf{x}_4$	0	0	1	0	1	1	0	1	0
$\mathbf{x}_5$	0	1	0	0	0	1	0	1	1
$\mathbf{x}_6$	0	1	1	0	1	0	0	0	1
$\mathbf{x}_7$	1	0	1	1	0	1	1	0	1

Table 2.1: *The codewords of the quasi-cyclic code presented in Example 2.2.*

We observe that $n_0 = 3$, and the code is indeed quasi-cyclic. For example consider the codeword $\mathbf{x}_2 = 110111100$. Cycling $\mathbf{x}_2$ one and two positions to the right results in the words 011011110 and 001101111 respectively which are not codewords from C. An encoding circuit for this code is shown in [LC04, p. 186]. □

In general, the performance of QC LDPC block codes compares favourably with that of randomly constructed LDPC codes for short to moderate block lengths. In fact, it has been shown that certain QC LDPC codes perform close to the Shannon limit [CXDL04]. Further, non-binary QC LDPC codes have been shown to have significant coding gains over RS codes of the same lengths and rates decoded with algebraic decoding [SZLA06]. In addition, an LDPC convolutional code (introduced in Section 1.5.2) can be constructed by replicating the constraint structure of the QC LDPC code at infinity [Tan87, ST79, EGSM98]. Recently, good convolutional codes have been constructed in this way [PSVC07]. A comparison between the performances of LDPC convolutional codes and the QC codes on which they are based is discussed in [TSS$^+$04].

2.3 Polynomial codes

Polynomial codes were introduced in 1968 by Kasami, Lin and Peterson [KLP68]. Polynomial codes can be viewed as a generalisation of many important classes of codes such as BCH codes, RS codes, RM codes and finite geometry codes. Shortened cyclic codes [LC04, pp. 179-183], which are not cyclic, are also a type of polynomial code.

The aptly named shortened cyclic codes are obtained by puncturing a cyclic code in a certain way. Consider a cyclic code C of length n and dimension k over a finite field F. Suppose we can form a subset S of the codewords of C that have zeros in the first l positions for some positive integer l. Deleting the first l information symbols from every member of S results in a set C_{short} of 2^{k-l} vectors of length $n-l$. C_{short} is a linear subcode of C, called a *shortened cyclic code*. This follows from the echelon structure of the shortened cyclic generator matrix.

Example 2.3. Consider the $(7, 4)$ cyclic code of Example 2.1. Let us form S by collecting all the codewords with $l = 1$, i.e., all the codewords with a 0 in the first position. Explicitly,

$$S = \{0000000, 0101100, 0010110, 0001011, 0110001, 0111010, 0011101, 0100111\}.$$

Then we obtain the shortened cyclic code

$$C_{short} = \{000000, 101100, 010110, 001011, 110001, 111010, 011101, 100111\},$$

which we observe is a linear code of length $n-l = 6$, with dimension $k-l = 3$, and is certainly not cyclic. □

Shortened cyclic codes are of practical interest as shortening codes permits the designer to obtain codes with a particular length or rate. Moreover, it is shown, for example in [LC04, pp. 179-183], that the same encoding circuit used for the cyclic code can be used for shortened cyclic codes. Recall from Section 2.2 that, in part, the motivation behind studying cyclic codes is the ease of encoding due to their inherent algebra. In Chapter 3, we will examine similar types of structure to that of shortened cyclic codes.

2.4 Codes and constructions over GF(q)

We now consider the general case when $F = GF(q)$, the finite field of q elements. It is known that q must be a power of p where p is a prime (for details see, e.g., [All91]). If q is itself a prime p then F can be thought of as the set of p elements $0, 1, \ldots, p-1$ with arithmetic operations performed modulo p. As motivation, LDPC codes defined over GF(q) of order $q > 2$ have been shown to have significantly improved performance compared to analogous binary LDPC codes when decoded using a decoding algorithm based on belief propagation [DM98].

In this thesis we focus on codes with diagonal structures in the parity-check matrix. Cyclic block codes, introduced in Section 2.2, possess this structure by definition. Possibly the most famous non-binary cyclic codes are Reed-Solomon (RS) codes. RS codes have been widely used for error-control in a variety of applications including deep-space communications [MS94] and coding schemes for compact discs [Imm94]. Moreover, binary versions of RS codes have been shown to be extremely adept (effectively optimal) at correcting burst errors [For71]. However, the drawback here is worse error-correcting performance compared to binary BCH codes. The key to implementing these codes was a decoding algorithm proposed by Berlekamp [Ber68][1]. As a testament to the qualities of these codes, Reed-Solomon codes are the main codes used with an expanded alphabet today, and yet were discovered over 40 years ago. For a detailed review of RS codes, see [WB94].

Low-density parity-check block codes, introduced in Section 1.5.1, are today's standard binary error-correcting block codes. Recently, very good LDPC block code constructions based on finite geometries over GF(q) were proposed by Kou, Lin and Fossorier [KLF00, KLF01] and extended by Tang, Xu, Lin and Abdel-Ghaffar [TXLA05]. This type of code will be either cyclic or quasi-cyclic, thus enabling easy encoding by implementing shift registers. Moreover, some of the long finite geometry LDPC block codes proposed in the above references have error performance close to the Shannon limit (in the region of a few tenths of a decibel). Due to the particular structure of the standard parity-check matrix of an LDPC code (see Section 1.5.1), finite geometries over such fields have an ideal structure to aid construction.

[1] The connection to linear codes was later observed by Massey the following year [Mas69]. The algorithm is known as the *Berlekamp-Massey algorithm*.

2.5 Existing encoding methods

In Chapter 3, a new method of obtaining a generator matrix for a so-called defectively cyclic parity-check matrix will be presented. The method uses a simple polnomial division algorithm similar to that of a cyclic code and outputs a generator matrix defined by a polynomial. Thus we are able to use shift registers to encode the resulting code which is not necessarily cyclic or quasi-cyclic. In this section, we review briefly some of the existing methods of encoding block codes.

To begin, we study an encoding circuit for a cyclic code to illustrate the simple shift register and feedback structure. Encoding and decoding circuits for cyclic codes were presented in [Meg61]. The standard way[2] to encode an (n, k) cyclic code in systematic form involves three steps:

1. We multiply the information polynomial $u(t) = u_0 + u_1 t + \ldots u_{k-1} t^{k-1}$ by t^{n-k};
2. the parity-bits polynomial $p(t)$ for the codeword with information polynomial $u(t)$ is obtained as the remainder when dividing $t^{n-k}u(t)$ by $g(t)$;
3. the codeword is then formed as $x(t) = t^{n-k}u(t) + p(t)$.

The steps in the above algorithm allow us to form a linear $(n-k)$-stage shift register with feedback connections based on the generator polynomial $g(t)$. The linear $n-k=3$ stage shift register with feedback connections for the cyclic code of Example 2.1 is shown in Figure 2.1.

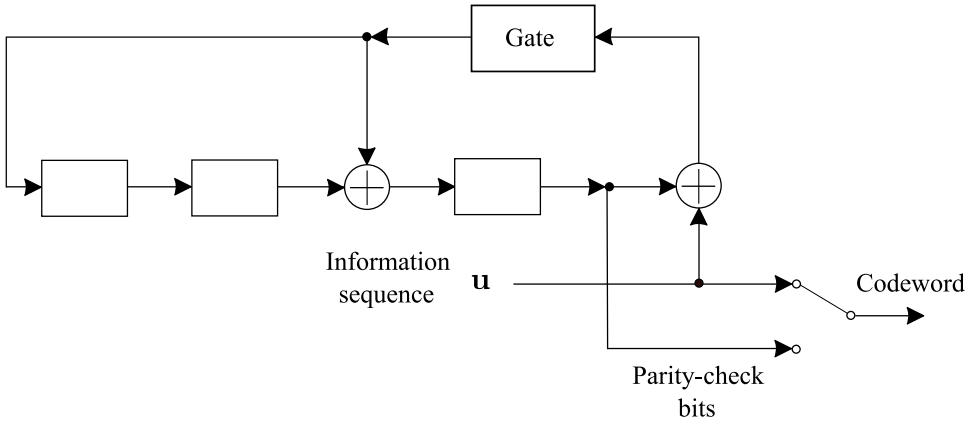


Figure 2.1: *An encoding circuit for the* $(7, 4)$ *cyclic code of Example 2.1.*

[2] A similar method to design encoding circuits based on the parity-check polynomial $h(t)$ exists. In general, circuit designers will choose the encoder based on $h(t)$ if there are more parity-check digits than information digits; otherwise the encoder based on $g(t)$ will be preferred.

Example 2.4. Consider the binary $(7, 4)$ cyclic code of Example 2.1. Suppose, for example, we wish to encode the sequence $\mathbf{u} = [\,1\ \ 1\ \ 0\ \ 1\,]$ arising from $u(t) = 1 + t + t^3$. In systematic form, the generator matrix for this cyclic code is

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix},$$

and we calculate

$$t^{n-k}u(t) + p(t) = t^3(1 + t + t^3) + t^2 = t^2 + t^3 + t^4 + t^6 \equiv [\,0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\,] = \mathbf{u}G.$$

For encoding, we shift the information sequence in to the circuit (and hence directly into the communication channel). Note that shifting the information sequence in to the channel at the front end is equivalent to the multiplication $t^{n-k}u(t) = t^3u(t)$. According to the input information sequence, the contents of the registers will be updated at every step. The state of the registers at every shift for this example is shown in Table 2.2. After the information bits have been input, the gate (and hence the feedback loop) is closed. The registers now contain the polynomial $p(t) = t^2 \equiv [\,0\ \ 0\ \ 1\,]$. We finally shift the parity digits into the channel. Thus the complete codeword is $\mathbf{x} = [\,0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\,]$. □

Shift number	Input bit	Register 1	Register 2	Register 3
0		0	0	0
1	1	1	0	1
2	0	1	1	1
3	1	0	1	1
4	1	0	0	1

Table 2.2: *Shift register state for the encoding circuit of Example 2.4.*

The binary cyclic encoding procedure described above can be modified for encoding non-binary cyclic codes. An encoder design for Reed-Solomon codes was presented by Berlekamp [Ber82]. We have observed that linear shift-register and feedback encoding circuits are particularly efficient for encoding block codes. This provides partial motivation for the study in Chapter 3, where we will look to use this approach to encode non-cyclic codes.

Today, LDPC block codes with long block length have virtually achieved the theoretical Shannon limit [CFRU01]. As discussed in Section 2.2.2, quasi-cyclic LDPC codes have been constructed with capacity-approaching performance [CXDL04]. Further, in Section 2.4, the method of LDPC block code construction based on finite geometries over GF(q) was highlighted. An advantage with this construction method is that the codes possess cyclic or quasi-cyclic structure. A method of obtaining the generator matrices of quasi-cyclic LDPC codes and subsequent efficient shift register based encoding circuits was presented in [LCZ$^+$06].

While the structure of cyclic and quasi-cyclic codes certainly provides efficient encoder design, it was evident from Gallager's seminal paper on LDPC codes [Gal62] that from the performance point of view codes should be "random-like", i.e., the distance distribution of the codewords should resemble that of a randomly constructed code. Indeed, most of the capacity-approaching LDPC codes have a random-like structure [SS96,Mac98,Mac99,CFRU01,RU01a, RSU01, MN97]. However, the drawback is that this randomness will complicate the encoder design. The benchmark encoding algorithm for randomly constructed LDPC block codes was presented by Richardson and Urbanke [RU01b]. This state-of-the-art algorithm requires only $O(g)$ operations per symbol, where $g << n$.

In Chapter 3, we form less constrained generator matrices than those of cyclic or quasi-cyclic codes, yet which can still be defined completely by polynomials. In this sense, they can be viewed as more general codes that can still make use of simple shift register based encoding.

Chapter 3
Efficient Encoding Using Polynomials

Abstract

In this chapter, an efficient new method of obtaining a generator matrix G from certain types of parity-check matrices H with a defective cyclic block structure is proposed. This novel approach describes parity-check and generator matrices in terms of polynomials. Moreover, using this polynomial algebra, we have found efficient ways to implement the scheme. In addition, this method is interesting as such, as it allows us to convert H into G without a systematic encoder in between (i.e., there is no diagonal subpart in the output). This alone is striking as normally G would be dense if we were to form it from the given H by Gaussian elimination. Further, we show that the new algorithm can be used to find generator matrices for non-binary codes possessing a defective cyclic block structure. A selection of the results presented here has been published previously in [MOG07] and [MOG08].

3.1 Introduction

Cyclic codes, which were introduced as early as 1957 [Pra57], are the most studied of all linear block codes. Due to their important cyclic structure, they can be encoded easily using shift registers and practical decoding schemes can be devised to make use of the underlying algebra [Meg61]. Cyclic codes were introduced in Section 2.2, and it was shown there that the generator and parity-check matrices of cyclic codes can be obtained from one other by simple polynomial division.

The generator and parity-check matrices can themselves be described by polynomials $g(t)$ and $h(t)$ respectively, and providing that their product is $t^n - 1$, where n is the block length of the code, the polynomial division algorithm can be implemented easily. In this chapter, an efficient new algorithm is proposed to obtain a generator matrix from a so-called *defective* parity-check matrix. This matrix appears to be cyclic. However the would-be parity-check polynomial $h(t)$ does not satisfy the condition[1] $h(t)|t^n-1$ for such a code length. The generator matrix obtained by this method is, in general, non-systematic and is completely defined by a polynomial (similar to the case of a cyclic code). This inherent structure allows efficient encoding and decoding schemes using shift registers to be employed.

Further, it is shown that this versatile method can be modified easily to work with a parity-check matrix made up of multiple defective blocks. This multi-block structure has been observed to arise frequently in the context of low-density parity-check (LDPC) block codes [LC04, pp. 851-947]. Moreover, the proposed polynomial approach holds over GF(p^s) where p is a prime and s is a non-negative integer. Thus the method can be generalised to the consideration of non-binary codes with this cyclic block structure.

The chapter is structured as follows. In Section 3.2 we begin by considering a parity check matrix made up of a single defective block, and introduce an extension method which efficiently finds a generator matrix even when the check polynomial $h(t)$ does not divide the corresponding $t^n - 1$. Then, in Section 3.3, an approach is detailed for a parity-check matrix made up of two blocks in cyclic form. The method is then extended in Section 3.4 to deal with an arbitrary number of blocks. In Section 3.5, it is shown that we can use properties of the dual code in order to obtain a parity-check matrix from a defective generator matrix without complicating the procedure. Finally, in Section 3.6, the method is generalised to consider non-binary codes

[1] By the notation $h(t)|(t^n - 1)$ it is meant that $h(t)$ divides $t^n - 1$ without remainder. If this does not hold, we write $h(t) \nmid (t^n - 1)$. Since there is no danger of ambiguity we abreviate notation by $h(t)|t^n - 1$.

with this defective cyclic block structure. This extension provides a huge selection of codes of real practical interest that are suitable for the proposed polynomial encoding method.

3.2 The extension method

Definition 3.1 *Suppose we have a parity-check matrix H in cyclic form (defined in equation (2.2)), where $h(t) \nmid t^n - 1$. Then $h(t)$ is called* ***defective****, and the parity-check matrix H is said to define a* ***defectively cyclic code****.*

Suppose we have a defective parity-check matrix H. The proposed method of 'extension' outputs a generator matrix for the code using polynomial division and the truncation of an interim matrix. This can be implemented efficiently using any standard Computer Algebra package. The method begins by finding an $n^* > n$ such that $h(t)$ divides $t^{n^*} - 1$.

We must first show that such an n^* exists and how to find it. Later we will see that finding and working with a large n^* is not necessary, but it is essential to show its existence. That this is indeed the case is the first new result shown in Section 3.2.1 using two standard facts. We can then use this to prove the main new theorem in Section 3.2.2. In Section 3.2.3, we propose a practical refinement of this theorem that enables it to be implemented efficiently.

This optimization allows us to obtain the desired generator matrix directly via polynomial division, so avoiding the theoretical values displayed in Sections 3.2.1 and 3.2.2. In the case of a single defective block, standard methods using Gaussian elimination are also easy to implement due to the cyclic structure of the matrix. However, when we consider stacked blocks (Sections 3.3 and 3.4) this new method is a very efficient alternative.

3.2.1 Finding a suitable n^*

Suppose $h(t)$ is an irreducible polynomial of degree $k > 1$ over GF(2). We then create an extension field GF(2^k) in the usual way [Gar04]. This process produces an extending element α which is a root of $h(t)$. If we throw away the zero element from this field and consider GF$(2^k)\backslash\{0\}$, we are left with an abelian group under multiplication. We now use:

Theorem 3.1 (Lagrange [All97]) *The order*[2] *of an element g of a finite group G divides the order $|G|$ of G.*

So $g^{|G|} = 1$. Hence g is a root of the polynomial $f(t) = t^{|G|} - 1$. Let $G = \text{GF}(2^k)\backslash\{0\}$, so that $|G| = 2^k - 1$. Suppose $g = \alpha$ where α is a root of $h(t)$. Since $f(\alpha) = 0$, it follows that $h(t)|f(t)$.

So for an irreducible polynomial $h(t)$ of degree $k > 1$ over GF(2), we can immediately write down an $n^* = 2^k - 1$ such that $h(t)|t^{n^*} - 1$. If $h(t)$ is not irreducible, we proceed by splitting the polynomial into irreducible parts. We will see that there is a systematic way of choosing n^* which depends on the structure of $h(t)$. To avoid trivialities we assume that $t \nmid h(t)$; this avoids zero columns in H. Let us first recall a required result on the divisors of certain polynomials.

Theorem 3.2 [MS77, pp. 99–106] *Over any field, $x^s - 1|x^r - 1$ if and only if $s|r$.*

If we can reduce $h(t)$ there are two possible cases:

Case (1)*: No repeated irreducible factors*
In this case, suppose $h(t) = q_1(t)q_2(t)\cdots q_s(t)$, where all the $q_i(t)$ are distinct and irreducible over GF(2). Here we apply the above procedure to each of the s factors. Thus for each factor we observe that $q_i(t)|t^{n_i^*} - 1$ for a particular n_i^*. Now, using Theorem 3.2, we observe

$$t^{n_i^*} - 1|t^{n^*} - 1 \text{ if and only if } n_i^*|n^*.$$

Thus if we set $n^* = \text{lcm}(n_1^*, n_2^*, \ldots, n_s^*)$, $h(t)|t^{n^*} - 1$, as each of the distinct factors $q_i(t)$ of $h(t)$ divides $t^{n^*} - 1$.

Case (2)*: Repeated irreducible factors*
If there are repeated irreducible factors, then $h(t) = q_1^{r_1}(t)q_2^{r_2}(t)\cdots q_s^{r_s}(t)$ where all the $q_i(t)$ are distinct and irreducible and at least one of the $r_i \geq 2$. We observe that when working modulo 2

$$(t^k - 1)^2 = t^{2k} + 1.$$

[2]Recall for any element g from a group G if there exists a positive integer m such that $g^m = e$ (where e is the identity element of the group) then the smallest such positive integer is called the *order* of g. If no such m exists we say g is of *infinite order*. The order of a group G is simply the number of elements in the group: this is denoted by $|G|$.

Suppose we write $t^k - 1 = u_1(t)u_2(t)\cdots u_p(t)$, where each of the $u_i(t)$ is irreducible over GF(2). Then

$$\begin{aligned}(t^k-1)^2 &= t^{2k}+1 = \quad u_1^2(t)u_2^2(t)\cdots u_p^2(t),\\ (t^k-1)^4 &= t^{4k}+1 = \quad u_1^4(t)u_2^4(t)\cdots u_p^4(t),\\ &\vdots\end{aligned}$$

In general $t^{2^i k}+1 = u_1^{2^i}(t)u_2^{2^i}(t)\cdots u_p^{2^i}(t)$. So if $q_i(t)$ is a factor of t^k-1 then $q_i^{r_i}(t)$ will be a factor of $t^{2^i k}+1$ if $2^i \geq r_i$.

Thus the approach in this case is as follows. For each of the irreducible factors $q_i(t)$ we find an n_i^* (in the usual way) such that $q_i(t)|t^{n_i^*}-1$ as in case (1). Set $k = \text{lcm}(n_1^*, n_2^*, \ldots, n_s^*)$. Let $r = \max(r_1, r_2, \ldots, r_s)$. Using the above rule we find the least such integer c so that $2^c \geq r$. Finally we set $n^* = k \cdot 2^c$. Then $h(t)|t^{n^*}-1$. Hence we have proved our first new result.

Theorem 3.3 *In the above notation, there exists a positive integer n^* such that $h(t)|t^{n^*}-1$.*

3.2.2 Finding a generator matrix in the defective case

Now we know that an n^* exists, we can state an algorithm to obtain a generator matrix:

1. we begin with a defective parity-check matrix H for a code C of block length n, in the sense that H 'looks' cyclic (with corresponding check polynomial $h(t)$), but $h(t) \nmid t^n - 1$;
2. we find a larger n^* such that $h(t)|t^{n^*}-1$;
3. we then calculate $g(t) = (t^{n^*}-1)/h(t)$;
4. finally, the generator matrix associated to $g(t)$ is truncated from the left so as to leave n columns. This new matrix is a generator matrix of the original code.

Theorem 3.4 *The matrix so formed is a generator matrix of the original code C.*

Proof. The proof of this can be split into three parts. Throughout we will denote the check polynomial by $h(t) = h_0 + h_1 t + \ldots + h_k t^k$ with $h_k = h_0 = 1$, where $0 < k < n$.

Claim 1. The matrix so formed has the appropriate rank and number of columns to be a generator matrix for C.

Proof. Suppose the parity-check matrix H is a $(n-k) \times n$ matrix with cyclic shift structure. This matrix is necessarily of full rank, since H has a backwards echelon structure with $h_k \neq 0$. Let G be a generator matrix for C. Then G is a $k \times n$ matrix with rank k.

Note that the degree of the parity-check polynomial is k, as we require $n-k$ rows in H. Using $t^{n^*}-1$, we calculate the generator polynomial $g(t)$ of the corresponding cyclic code as $g(t) = (t^{n^*}-1)/h(t)$. Thus the degree of the generator polynomial $g(t)$ of this extended matrix is n^*-k. Recall from Section 2.2.1 that the corresponding generator matrix G' is formed by 'sliding' the coefficients of $g(t)$ diagonally downwards as to to fill up the matrix (as in (2.1)). Thus we calculate

$$\text{row rank } G' = n^* - (n^* - k) = k = \text{ row rank } G.$$

Now set G'' to be G' with the first $n^* - n$ columns removed. Then G'' is an $k \times n$ matrix. The final necessary condition is that G'' must be of full rank. Recall that the generator polynomial necessarily has non-zero coefficient g_{n^*-k}. This creates an echelon structure in the cyclic generator matrix G'. We note that the truncation length n is greater than k (from construction of the check matrix). Thus the echelon structure of the cyclic generator matrix G' is preserved in this new matrix G''. □

Claim 2. $G''H^T = 0$.

Proof. Let H' be the $(n^*-k) \times n^*$ parity-check matrix of the cyclic code of block length n^* corresponding to the parity-check polynomial $h(t)$. So

$$H' = \begin{bmatrix} 0 & & h(t) \\ & \swarrow & \\ & & \end{bmatrix}_{(n^*-k)\times n^*},$$

Then $G'H'^T = 0_{k\times(n^*-k)}$ by construction. Now create a new matrix H'' which is equal to the first $n-k$ rows of H',

$$H'' = \begin{bmatrix} 0_{(n-k)\times(n^*-n)} & H \end{bmatrix}_{(n-k)\times n^*}.$$

Observe that $G' {H''}^T = 0_{k\times(n-k)}$.

By construction, the first $n^* - n$ columns of H'' are all zero. Thus they are irrelevant in the calculation. Remove the first n^*-n columns from G' and H'', making G'' and H''' respectively. Observe that

$$H''' = \begin{bmatrix} & & & h(t) \\ & & \swarrow & \\ & & & \end{bmatrix}_{(n-k)\times n} = H.$$

Thus, $G'' {H'''}^T = G'' H^T = 0_{k\times n}$. □

Claim 3. Rowspace G'' = rowspace G.

Proof. As $G'' H^T = 0$ and $G H^T = 0$ we know that each word in G'' and G is perpendicular to each word in H. Thus

$$\begin{aligned} \text{rowspace } G'' &\subseteq \text{null space } H, \\ \text{rowspace } G &\subseteq \text{null space } H. \end{aligned}$$

The null-space of the $(n-k)\times n$ parity-check matrix H (of full rank) must be of dimension k. As both G and G'' have rank k, it follows that rowspace G'' = rowspace G. □ □

Example 3.1. Suppose $h(t) = t^3 + t + 1$ and $n = 5$. We observe that $h(t)$ is irreducible over GF(2) and that $h(t) \nmid t^5 - 1 (\text{mod } 2)$, though the corresponding defective parity check matrix H looks cyclic:

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}_{2\times 5}.$$

We find that $n^* = 7$, so $h(t) | t^7 - 1 (\text{mod } 2)$. Then the generator polynomial for the extended cyclic code is

$$g(t) = (t^7 - 1)/(t^3 + t + 1) = t^4 + t^2 + t + 1.$$

So the extended matrices are

$$H' = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}_{4\times 7}$$

and

$$G' = \left[\begin{array}{cc|ccccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right]_{3\times 7}.$$

Thus by construction $G'H'^T = 0_{3\times 4}$. Recall that the next step is to set H'' equal to the first $n-k$ rows of H'

$$H'' = [\, 0_{2\times 2} \;\; H \,]_{2\times 7}.$$

Clearly now $G'H''^T = 0_{3\times 2}$. The final step is to observe there will be no contribution from the first $n^* - n = 2$ columns. Removing these columns from G' results in G'' (which is the boxed area of G'). It should now be evident that $G''H^T = 0_{3\times 2}$ and hence G'' is a generator matrix for the code because of an argument using rank (as above), or by direct calculation here. □

3.2.3 Optimizing the method

It was noted earlier that the desired n^* could be very large and hence polynomial division would be very time-consuming. Depending on the desired block length n, we can optimize the method to save on calculations. The generator matrix of the extended code is formed by sliding the coefficients of the generator polynomial diagonally downwards (as in (2.1)). Then, as we are truncating this extended matrix, some of the coefficients may be surplus to requirements. This is best observed in an example.

Example 3.2. Let $h(t) = t^{34} + t^6 + t^3 + 1$. We observe $h(t)$ is reducible; explicitly,

$$\begin{aligned} h(t) &= (t^7 + t^5 + t^4 + t^3 + 1)(t^6 + t^5 + t^4 + t + 1) \\ &\quad (t+1)(t^{20} + t^{18} + t^{10} + t^9 + t^6 + t^5 + t^4 + t^2 + 1). \end{aligned}$$

The factors here are all irreducible. Note that there are no repeated factors, hence $h(t)$ is a *Case* (1) polynomial. We calculate

$$n^* = \text{lcm}(2^7 - 1, 2 - 1, 2^6 - 1, 2^{20} - 1) = 2796549525.$$

Suppose $n = 37$. (Typically we will choose a block length and this will define the rank of our generator matrix, since the degree of $h(t)$ is given.) In this case, as the degree of our check

polynomial is $k = 34$, H is a 3×37 matrix and G will be a 34×37 generator matrix. However we observe G' looks as follows, where the entries are the coefficients of $g(t)$:

$$G' = \begin{bmatrix} & \boxed{\begin{matrix} g_{n^*-37} & \cdots & g_{n^*-2} & g_{n^*-1} \\ g_{n^*-38} & g_{n^*-37} & \cdots & g_{n^*-2} \\ \vdots & \vdots & & \vdots \\ g_{n^*-70} & g_{n^*-69} & \cdots & g_{n^*-34} \end{matrix}} \\ \star & \end{bmatrix},$$

where

$$\star = \begin{pmatrix} g_0 & g_1 & & \cdots & & g_{n^*-39} & g_{n^*-38} \\ 0 & g_0 & g_1 & & \cdots & & g_{n^*-39} \\ \vdots & \ddots & \ddots & \ddots & & & \vdots \\ 0 & \dots & 0 & g_0 & g_1 & \cdots & g_{n^*-71} \end{pmatrix}.$$

Observe that the boxed section is the generator matrix G that we want. Thus the majority of the matrix is irrelevant. So to write down G we only require the coefficients of $t^{n^*-1}, \dots, t^{n^*-70}$ from the generator polynomial $g(t)$.

The division to obtain $g(t)$ looks like

$$g(t) = \frac{t^{n^*} - 1}{t^{34} + t^6 + t^3 + 1} = t^{n^*-34} + t^{n^*-62} + t^{n^*-65} + t^{n^*-68} + t^{n^*-90} + \dots.$$

Thus there are only 4 non-zero coefficients to fill in to obtain the boxed area of G', since coefficients of terms of degree $n^* - 71$ or less are in the non-boxed section of G'.

For the purpose of obtaining the necessary number of non-zero coefficients we can work with a smaller value of n^*, denoted $\tilde{n}$. Using this smaller value of n^* will result in a remainder which can be discarded. So we will have

$$t^{\tilde{n}} - 1 = q(t)h(t) + r(t), \tag{3.1}$$

where the polynomial $q(t)$ has the correct coefficients to form the desired generator matrix G (the boxed part of G'). There is a limit to how small we can choose $\tilde{n}$ to be so that $q(t)$ still has the correct coefficients. We observe that in our example we require $\tilde{n} \geq 70$ to include the

coefficient of $t^{\tilde{n}-70}$. Observe that (3.1) in this case becomes

$$t^{70} - 1 = (t^{36} + t^8 + t^5 + t^2)(t^{34} + t^6 + t^3 + 1) + t^{14} + t^8 + t^2 + 1.$$

Thus $q(t) = t^{36}+t^8+t^5+t^2$. Recall that the non-zero coefficients from G' are $t^{n^*-34}, t^{n^*-62}, t^{n^*-65}$ and t^{n^*-68}. Observe that these coefficients match up when substituting $n^* = 70$. So for $\tilde{n} = 70$

$$G = \begin{bmatrix} q_{33} & \cdots & q_{68} & q_{69} \\ q_{32} & q_{33} & \cdots & q_{68} \\ \vdots & \vdots & & \vdots \\ q_0 & q_1 & \cdots & q_{34} \end{bmatrix}.$$

□

Thus the optimized procedure established above can be summarised in the following new Theorem.

Theorem 3.5 *In the above notation given a block length n and a check polynomial of degree k then $\tilde{n} = n + k - 1$.*

3.3 Finding a generator matrix in the case of two defective blocks

3.3.1 Setup

Consider a parity-check matrix of the form

$$H = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}, \tag{3.2}$$

where H_1 and H_2 are in 'cyclic' form, but defective. By that it is meant that the submatrices appear as the form specified in Section 2.2.1 but the associated parity-check polynomials do not both divide $t^n - 1$, where n is the number of columns of H. Let C_1 and C_2 be the codes formed by the null space of the matrices H_1 and H_2 respectively. Then we define the concatenated code [BBF+06]:

$$C_1 \dot{+} C_2 = \{[\mathbf{x}_1 : \mathbf{x}_2] \,|\mathbf{x}_1 \in C_1 \text{ and } \mathbf{x}_2 \in C_2\}.$$

We define the modulo 2 addition of two codes

$$C_1 \oplus C_2 = \{\mathbf{x}_1 + \mathbf{x}_2 \bmod 2 \,|\mathbf{x}_1 \in C_1 \text{ and } \mathbf{x}_2 \in C_2\}.$$

Consider the linear mapping $[\,\mathbf{x}_1 \,:\, \mathbf{x}_2\,] \rightarrow \mathbf{x}_1 \oplus \mathbf{x}_2$ of the concatenated code $C_1 \dot{+} C_2$ to the modulo-2 sum of its two component codes. Using the conservation law of dimensions [All97] we have

$$\dim(C_1 \dot{+} C_2) = \dim(C_1 \oplus C_2) + \dim(C_1 \cap C_2). \tag{3.3}$$

Notice that if $C_1 \cap C_2 = \{0 \ldots 0\}$ then $\dim(C_1 \dot{+} C_2) = \dim(C_1 \oplus C_2)$. Let H_1 and H_2 have corresponding generator matrices G_1 and G_2. We are interested in the null space of the parity-check matrix (3.2); this is the intersection of the null spaces C_1 and C_2. To get a nice result on this code we will see that we can use the method of extension to bring H into a form we can usefully work with.

We have that C_1 and C_2 are the rowspaces of G_1 and G_2 respectively. It is easily seen that

$$C_1 \oplus C_2 = \text{rowspace} \begin{bmatrix} G_1 \\ G_2 \end{bmatrix}.$$

Using the method of extension, we know that for both the check polynomials $h_1(t)$ and $h_2(t)$ associated with the blocks H_1 and H_2 there exists an integer n_i^* such that $h_i(t)|t^{n_i^*} - 1$, $i = 1, 2$. We again suppose that $t \nmid h_i(t)$, $i = 1, 2$. Let $n^* = \text{lcm}\ (n_1^*, n_2^*)$; then $h_i(t)|t^{n^*} - 1$, $i = 1, 2$. We extend (3.2) to an $((n-k_1)+(n-k_2)) \times n^*$ matrix, which consists of two cyclic blocks of size $(n^* - k_i^*) \times n^*$, $i = 1, 2$, where k_i is the degree of polynomial $h_i(t)$. Now we can calculate the generator polynomials $g_1(t)$ and $g_2(t)$ associated with the two extended blocks and form the respective generator matrices in the usual way

$$G_i' = \begin{bmatrix} g_i(t) & & \\ & \searrow & \end{bmatrix}_{k_i \times n^*}.$$

Let the extended codes formed from these matrices be denoted C_1' and C_2' respectively. Note that the sum of two codes is again cyclic and has generator polynomial equal to the greatest common divisor of the individual generator polynomials. This is based on a standard result in the ideal theory of the polynomial ring $F[t]$. If $\langle g_1(t)\rangle$, $\langle g_2(t)\rangle$ are two ideals in $F[t]$, then $\langle g_1(t)\rangle + \langle g_2(t)\rangle = \langle g(t)\rangle$, where $g(t) = \gcd(g_1(t), g_2(t))$. Moreover, if $g_1(t)|t^n - 1$ and

$g_2(t)|t^n - 1$, then these statements continue to hold in the factor ring $R_n = F[t]/\langle t^n - 1\rangle$.

We now introduce notation for the truncation operation. Suppose a code C' of length $n^* > n$ is truncated from the left to be of length n. We denote the new code C (the right hand side of the original code) by $C = C'|_{\text{RHS}}$. Following the work done in Section 3.2, it should be evident that truncating the sum of extended codes (from the left) to be of length n will result in the sum of the required codes. That is,

$$(C_1' \oplus C_2')\big|_{\text{RHS}} = C_1 \oplus C_2.$$

Of course we are interested in the intersection of the codes C_1 and C_2. We have

$$(C_1' \cap C_2')\big|_{\text{RHS}} \subseteq C_1'\big|_{\text{RHS}} \cap C_2'\big|_{\text{RHS}}\,. \tag{3.4}$$

This is easily seen by choosing a codeword $\mathbf{x}$ that exists in the intersection $C_1' \cap C_2'$. Suppose we split $\mathbf{x}$ into left and right parts, so $\mathbf{x} = [\,l : r\,]$ with the part r of length n. Then

$$[\,l : r\,] \in C_1' \cap C_2' \Rightarrow r \in C_1'\big|_{\text{RHS}} \cap C_2'\big|_{\text{RHS}}\,.$$

Note that if we can show equality in the dimensions of the two sides of (3.4) then the two spaces must be equal because one is contained in the other. So we proceed by examining the dimensions in (3.4). We immediately get the inequality

$$\dim\left((C_1' \cap C_2')\big|_{\text{RHS}}\right) \leq \dim\left(C_1'\big|_{\text{RHS}} \cap C_2'\big|_{\text{RHS}}\right) = \dim\left(C_1 \cap C_2\right). \tag{3.5}$$

We can write the left hand side of (3.5) as

$$\begin{aligned}
&\dim\left((C_1' \cap C_2')\big|_{\text{RHS}}\right)\\
&\quad= \dim\left(C_1' \cap C_2'\right) \text{ for a suitably large}^4\, n,\\
&\quad= \dim\left(C_1' \dot{+} C_2'\right) - \dim\left(C_1' \oplus C_2'\right) \text{ from (3.3)},\\
&\quad= \dim\left(C_1'\right) + \dim\left(C_2'\right) - \dim\left(C_1' \oplus C_2'\right),
\end{aligned}$$

[4]We require that n is larger than the degree of the least common multiple of check polynomials $h_1(t)$ and $h_2(t)$.

where in the last line we use the fact that the dimension of the concatenated codes is equal to the sum of the individual dimensions of the codes [BBF+06]. Using this property and (3.3) we now write the right-hand side of (3.5) as

$$\dim\left(C_1 \cap C_2\right) = \dim\left(C_1\right) + \dim\left(C_2\right) - \dim\left(C_1 \oplus C_2\right).$$

As the rank of the generator matrices of the original and the extended codes are equal, the dimensions of the codes must be equal ($\dim(C_i) = \dim(C_i')$). The upshot is that we can now re-write (3.5) as

$$\dim\left(C_1' \oplus C_2'\right) \geq \dim\left(C_1 \oplus C_2\right). \tag{3.6}$$

Now, considering the modulo 2 sum of two codes as rowspaces of stacked generator matrices, we observe

$$\begin{aligned} C_1 \oplus C_2 &= \text{rowspace}\begin{bmatrix} G_1 \\ G_2 \end{bmatrix} = \text{rowspace}\left[\begin{array}{c|c} G_1' & \\ G_2' & \end{array}\right]_{\text{RHS}} \\ &= \left(C_1' \oplus C_2'\right)\big|_{\text{RHS}}. \end{aligned}$$

So as $\left(C_1' \oplus C_2'\right)|_{\text{RHS}} = C_1 \oplus C_2$ their dimensions must be equal. Thus (3.6) becomes

$$\dim\left(C_1' \oplus C_2'\right) \geq \dim\left(\left(C_1' \oplus C_2'\right)\big|_{\text{RHS}}\right).$$

Note that by design the truncation (block) length n is larger than the degree of the check polynomials. Thus truncation leaves the echelon structure intact and we must have equality. Recall that equality here implies that the spaces in (3.4) are equal.

3.3.2 Fundamental method

Given two polynomials $h_1(t)$ and $h_2(t)$, the method is as follows:

- calculate n^* so that $h_1(t), h_2(t) | t^{n^*} - 1$;
- solve $g_i(t) = (t^{n^*} - 1)/h_i(t)$ for $i = 1, 2$;
- solve $g(t) = \text{lcm}(g_1(t), g_2(t))$. This polynomial generates the intersection of the extended codes;

- truncation from the left gives us the desired code, providing n is greater than the degree of $\text{lcm}(h_1(t), h_2(t))$.

3.3.3 Optimizing the method

This process could be fairly time consuming if n^* is large. If we work with polynomials of high degree this method would probably not be beneficial. So we look to alternative means. We aim to get the least common multiple of the two generator polynomials $g_1(t)$ and $g_2(t)$ (possibly of very high degree).

Theorem 3.6 *Given two generator polynomials $g_1(t)$ and $g_2(t)$ as discussed above and a positive integer n^* as defined in Section 3.2.1, we have*

$$\text{lcm}(g_1(t), g_2(t)) = (t^{n^*} - 1)/\text{gcd}(h_1(t), h_2(t)).$$

Proof. The proof is a direct result of setting $s = 2$ in the proof of Theorem 3.7 below. □

Observe that this is now in the form of our original optimization problem. We solve (3.1) for $q(t)$ using $h(t) = \text{gcd}(h_1(t), h_2(t))$, where $\tilde{n}$ is the number of rows plus the number of columns of our desired generator matrix minus one.

Example 3.3. Let

$$\begin{aligned} h_1(t) &= 1 + t + t^2 + t^5 + t^6 + t^7 + t^8 + t^{11}, \\ h_2(t) &= 1 + t^2 + t^4 + t^9 + t^{10} + t^{12} + t^{14} + t^{15}. \end{aligned}$$

We calculate the $\text{lcm}(h_1(t), h_2(t))$ and the $\text{gcd}(h_1(t), h_2(t))$. The degree of the l.c.m. is 16, thus we must choose $n > 16$; let $n = 32$. We know G will be a 10×32 matrix; thus $\tilde{n} = 41$. Now we can solve (3.1) for $q(t)$:

$$t^{41} - 1 = q(t)(1 + t^2 + t^3 + t^4 + t^6 + t^8 + t^9 + t^{10}) + r(t),$$

and we find that

$$\begin{aligned} q(t) =& 1 + t^3 + t^4 + t^8 + t^{10} + t^{13} + t^{14} + t^{15} + t^{19} + t^{21} + t^{22} + t^{23} \\ &+ t^{25} + t^{28} + t^{30} + t^{31}. \end{aligned}$$

These coefficients are entered into the matrix in the following way (with notation as before):

$$G = \begin{bmatrix} q_9 & \cdots & q_{39} & q_{40} \\ q_8 & q_9 & \cdots & q_{39} \\ \vdots & \vdots & & \vdots \\ q_0 & q_1 & \cdots & q_{31} \end{bmatrix}.$$

The rowspace of this matrix G is the code we desire. □

3.4 An arbitrary number of parity-check blocks

The procedure given in the previous section can be extended to deal with an arbitrary number of these defective blocks $H_1, \ldots, H_s$. Suppose we wish to have a parity-check matrix

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_s \end{bmatrix},$$

defined by s polynomials $h_1(t), \ldots, h_s(t)$.

Theorem 3.7 *Given a block length n such that $n >$ degree(lcm$(h_1(t), h_2(t), \ldots, h_s(t))$), the polynomial given by*

$$\text{lcm}(g_1(t), \ldots, g_s(t)) = \frac{(t^{n^*} - 1)}{\text{gcd}(h_1(t), \ldots, h_s(t))}$$

generates the intersection of the extended codes.

Proof. Let $t^{n^*} - 1 = q_1^{r_1}(t) q_2^{r_2}(t) \cdots q_s^{r_s}(t)$ where all the $q_i(t)$ are distinct and irreducible and the $r_i \geq 1$ for $i = 1, \ldots, s$. By the properties of cyclic codes $g_i(t) h_i(t) = t^{n^*} - 1$ for

$i = 1, \ldots, l$. Let the generator polynomials be written as

$$\begin{aligned} g_1(t) &= q_1^{k_{1,1}}(t) q_2^{k_{1,2}}(t) \cdots q_s^{k_{1,s}}(t), \\ &\vdots \\ g_l(t) &= q_1^{k_{l,1}}(t) q_2^{k_{l,2}}(t) \cdots q_s^{k_{l,s}}(t), \end{aligned}$$

for $0 \leq k_{i,j} \leq r_i$, $i = 1, \ldots, l$, $j = 1, \ldots, s$. Let $k'_{i,j} = r_j - k_{i,j}$, to simplify notation. Then

$$\begin{aligned} h_1(t) &= q_1^{k'_{1,1}}(t) q_2^{k'_{1,2}}(t) \cdots q_s^{k'_{1,s}}(t), \\ &\vdots \\ h_l(t) &= q_1^{k'_{l,1}}(t) q_2^{k'_{l,2}}(t) \cdots q_s^{k'_{l,s}}(t). \end{aligned}$$

Then for an arbitrary number l of polynomials, we have

$$\text{gcd}(g_1(t), g_2(t), \ldots, g_l(t)) = q_1^{\min(k_{1,1}, k_{2,1}, \ldots, k_{l,1})} q_2^{\min(k_{1,2}, k_{2,2}, \ldots, k_{l,2})} \ldots q_s^{\min(k_{1,s}, k_{2,s}, \ldots, k_{l,s})}, \tag{3.7}$$

and

$$\text{lcm}(g_1(t), g_2(t), \ldots, g_l(t)) = q_1^{\max(k_{1,1}, k_{2,1}, \ldots, k_{l,1})} q_2^{\max(k_{1,2}, k_{2,2}, \ldots, k_{l,2})} \ldots q_s^{\max(k_{1,s}, k_{2,s}, \ldots, k_{l,s})}. \tag{3.8}$$

Then we use the rule; $\max\{k_{i,j}\} = r_j - \min\{k'_{i,j}\}$, where $1 \leq j \leq s$ is fixed and $i \in \{1, \ldots, l\}$. Thus (3.8) becomes

$$\begin{aligned} \text{lcm}(g_1(t), g_2(t), \ldots, g_l(t)) &= q_1^{r_1 - \min\{k'_{i,1}\}} q_2^{r_2 - \min\{k'_{i,2}\}} \ldots q_s^{r_s - \min\{k'_{i,s}\}}, \\ &= \frac{q_1^{r_1}(t) q_2^{r_2}(t) \ldots q_s^{r_s}(t)}{q_1^{\min\{k'_{i,1}\}} q_2^{\min\{k'_{i,2}\}} \ldots q_s^{\min\{k'_{i,s}\}}}, \\ &= \frac{t^{n^*} - 1}{\text{gcd}(h_1(t), h_2(t)), \ldots, h_l(t))}, \text{ using } (3.7), \end{aligned}$$

where we minimize over $i \in \{1, \ldots, l\}$. □

Let $h(t) = \text{gcd}(h_1(t), h_2(t), \ldots, h_s(t))$ and $k = \text{degree}(h(t))$. To form the $k \times n$ generator matrix G associated with our original matrix H we solve (3.1) for $q(t)$ with $\tilde{n} = k + n - 1$.

Then the coefficients of the polynomial $q(t) = q_0 + q_1 t + \ldots + q_{\tilde{n}} t^{\tilde{n}}$ form G as follows:

$$G = \begin{bmatrix} q_{k-1} & q_k & \cdots & q_{\tilde{n}-1} & q_{\tilde{n}} \\ q_{k-2} & q_{k-1} & \cdots & q_{\tilde{n}-2} & q_{\tilde{n}-1} \\ \vdots & \vdots & & \vdots & \vdots \\ q_0 & q_1 & \cdots & q_{n-2} & q_{n-1} \end{bmatrix}. \tag{3.9}$$

From a practical point of view, the G that is created is non-systematic and we do not need Gaussian elimination, which is an advantage when we are given an LDPC parity-check matrix with a large block length. Moreover, G has a polynomial description which allows for an implemenation of the encoder by simple shift registers rather than by a costly matrix multiplication. When the parity-check matrix can be brought into the form stated above, the result is a very efficient alternative to the method presented in [RU01b].

Example 3.4. We build a stacked parity-check matrix consisting of 3 defective cyclic blocks. Let

$$\begin{aligned} h_1(t) &= 1 + t + t^2 + t^5 + t^6 + t^7 + t^8 + t^{11}, \\ h_2(t) &= 1 + t^2 + t^4 + t^9 + t^{10} + t^{12} + t^{14} + t^{15}, \\ h_3(t) &= 1 + t + t^3 + t^5 + t^6 + t^9 + t^{13} + t^{17}. \end{aligned}$$

Each block of the parity-check matrix is given as:

$$H_i = \begin{bmatrix} & & h_i(t) \\ & \swarrow & \\ & & \end{bmatrix}_{n-\text{degree}(h_i(t)) \times n}.$$

Thus the resulting parity-check matrix is

$$H = \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix}.$$

We now calculate the greatest common divisor of the three polynomials:

$$\gcd(h_1(t), h_2(t), h_3(t)) = 1 + t^2 + t^3 + t^4 + t^6 + t^8 + t^9 + t^{10} = h(t).$$

The degree of the least common multiple of the three polynomials is 18. Thus we must choose an $n > 18$; let $n = 21$.

We know G will be a 10×21 matrix. Thus we set $\tilde{n} = 10 + 21 - 1 = 30$ and solve (3.1) for $q(t)$. This results in the expression

$$q(t) = t^2 + t^3 + t^4 + t^8 + t^{10} + t^{11} + t^{12} + t^{14} + t^{17} + t^{19} + t^{20}.$$

We now fill in matrix (3.9) with the coefficients of $q(t)$ to obtain a generator matrix. □

3.5 Defective generator matrices

It is worth noting that, using properties of the dual code, this method can output a parity-check matrix for a defective generator matrix[3] in the form of (2.1), or indeed, for stacked blocks of defective generator matrices. We use the following theorem:

Theorem 3.8 [LC04, p. 145] *Let C be an (n, k) cyclic code with generator polynomial $g(t)$. The dual code of C is also cyclic and is generated by the polynomial $t^k h(t^{-1})$, where $h(t) = (t^n - 1)/g(t)$.*

Thus given a block length n and a generator polynomial $g(t)$ of degree $n - k$ we calculate the polynomial

$$h'(t) = t^{n-k} g(t^{-1}),$$

and put these coefficients into a parity-check matrix as in (2.2). We now have a problem of the form solved in Section 3.2.

Example 3.5. Let $g(t) = 1 + t + t^4$ and $n = 7$. Note that $g(t) \nmid t^7 - 1$ modulo 2. We see that the resulting defective generator matrix is

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

[3]The defective generator matrix can be viewed as a shortened cyclic code. Shortened cyclic codes were introduced in Section 2.3, and it was discussed there that this type of code is of practical interest for ease of encoding and code design.

We calculate $h'(t) = t^4 g(t^{-1}) = t^4 + t^3 + 1$. Forming a parity-check matrix with $h'(t)$ and $n = 7$ results in

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

This, we observe, is an equivalent matrix[4] to G. The matrix H' is a defective parity-check matrix. Thus we can find an associated generator matrix using our new method, which is actually a parity-check matrix for G. □

The process described above can also be extended to an arbitrary number l of defective generator blocks. For each generator polynomial $g_i(t)$ we find $h'_i(t) = t^{\text{degree}(g_i(t))} g_i(t^{-1})$ for $i = 1, \ldots, l$. This allows us to use the machinery presented in Section 3.4 for multiple parity-check polynomials. The matrix produced by the method is a parity-check matrix for the given stacked generator matrix G.

3.6 Higher order fields

Until now we have considered only codes over GF(2) for simplicity. However the processes described thus far can easily be extended to work over GF(p^s), where p is a prime and s is a non-negative integer. This allows for a larger class of codes to be considered for the method, or indeed, translating to binary (see, e.g., [MS77]), some interesting quasi-cyclic type structures. Quasi-cyclic LDPC codes, described in Section 2.2.2, have been shown in the literature to have excellent performance while possessing a useful structure for encoding/decoding, see, e.g., [CXDL04].

Example 3.6. Let us define $h(t) = 1 + 2t + t^4$ over GF(3). Let $n = 7$. We observe that $h(t) \nmid t^7 - 1$. As the resulting parity-check matrix is necessarily of full rank we know that G will be a 4×7 matrix. Solving $t^{10} - 1 = h(t)q(t) + r(t)$ for $q(t)$ results in $q(t) = 1 + 2t^2 + t^3 + t^6$ and thus

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 1 & 0 & 0 & 1 \end{bmatrix},$$

with $GH^T = 0_{4\times 3}$ over GF(3). □

[4] By the term equivalent matrix, it is meant that linear combinations of the rows result in the same code.

Example 3.7. Let GF$(2^2) = \{a + b\alpha | a, b \in \text{GF}(2), \alpha^2 = 1 + \alpha\}$ be defined in the usual way. Consider the polynomial $h(t) = 1 + \alpha t + \alpha t^3$ and let $n = 7$. It can be shown that $h(t) \nmid t^7 - 1$ and, because H is necessarily of full rank, G will be a 3×7 matrix. Note that under these conditions, we have a *non-binary defectively cyclic code*. We solve $t^9 - 1 = h(t)q(t) + r(t)$ for $q(t)$ as usual. This results in

$$q(t) = \alpha + \alpha^2 t^2 + \alpha t^3 + \alpha^2 t^4 + \alpha^2 t^6,$$

and thus

$$G = \begin{bmatrix} \alpha^2 & \alpha & \alpha^2 & 0 & \alpha^2 & 0 & 0 \\ 0 & \alpha^2 & \alpha & \alpha^2 & 0 & \alpha^2 & 0 \\ \alpha & 0 & \alpha^2 & \alpha & \alpha^2 & 0 & \alpha^2 \end{bmatrix},$$

with $GH^T = 0_{3\times 4}$ over GF(2^2).

If we immediately translate these higher order symbols back in to binary we obtain a matrix with sliding block structure. Here, using binary equivalents $0 = 00$, $1 = 10$, $\alpha = 01$, and $\alpha^2 = 11$, the resulting binary matrix is

$$G_{bin} = \left[\begin{array}{cc:cc:cc:cc:cc:cc:cc} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hdashline 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \end{array}\right].$$

This we observe has sliding 1×2 blocks from top left to bottom right. Using the same mapping, the binary equivalent to H is given as

$$H_{bin} = \left[\begin{array}{cc:cc:cc:cc:cc:cc:cc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right],$$

and we observe that both matrices are of full rank and $G_{bin}H_{bin}^T = 0_{3\times 4}$. □

The matrices formed in this fashion are, in general, non-systematic. In addition, due to the polynomial division approach we have a degree of control over the density of the parity-check and generator matrices by picking appropriate polynomials.

3.7 Conclusions

In this chapter, a new method of obtaining a generator matrix G from certain types of parity-check matrices with a defective cyclic block structure has been presented. The generator matrices so-formed are described in terms of polynomials, and by using polynomial algebra we have found efficient ways to implement the scheme. There is a large class of practical parity-check matrices, e.g., LDPC codes, that are candidates for this method. Further, by careful choice of polynomials we are free to choose any length and rate code we may desire, and the density of G depends solely on its corresponding polynomial. The method was then extended to consider non-binary codes with this cyclic block structure, and it was shown that equivalent binary codes possess a quasi-cyclic type structure. This extension provides a huge selection of codes that are suitable for the proposed polynomial encoding method.

Part II

Asymptotically Good LDPC Convolutional Codes

Chapter 4
Background: Protograph-based Codes

Abstract

Protograph-based LDPC block codes, a subclass of multi-edge type codes, have been shown to have many desirable qualities, such as fast encoding/decoding, low iterative decoding thresholds and linear minimum distance growth. In this chapter, we review previous results from the literature concerning protographs and highlight the theory essential for the discussion in Chapters 5 and 6.

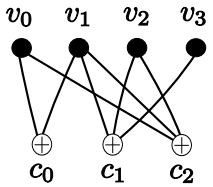
4.1 Introduction

Multi-edge type low-density parity-check codes [RU02, RU03], a generalisation of irregular LDPC codes, have recently inspired the design of LDPC codes with imposed sub-structures. LDPC codes based on a *protograph* [Tho03] (or *projected graph* [RN03]) form a subclass of multi-edge type codes that have been shown in the literature to have many desirable qualities such as fast encoding/decoding, low iterative decoding thresholds and linear minimum distance growth, see for example, [DDJ06, ADY07] and references there.

This chapter is structured as follows. In Section 4.2, we review the protograph construction method and describe the merits of the resulting codes. In Section 4.3, we summarize results by Divsalar [Div06] to calculate ensemble average weight enumerators for protograph-based LDPC codes which will later be used in Chapter 5. Finally, in Section 4.4, the notion of so-called 'trapping sets' will be introduced, and recent results by Abu-Surra, Ryan and Divsalar [ARD07a] concerning ensemble average trapping set enumerators of protograph-based LDPC codes will be presented and discussed. These enumerators will later be used in Chapter 6.

4.2 Protographs

A protograph is a small bipartite graph (a formal definition can be found in [Tho03]). Figure 4.1 shows a protograph and the associated protograph parity-check matrix P.



$$P = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Figure 4.1: *An example of a protograph and associated parity-check matrix.*

4.2.1 Protograph-construction

Suppose a given protograph P has n_v variable nodes and n_c check nodes. A protograph-based (LDPC) block code can be created using the copy-and-permute (CAP) operation [Tho03]. The CAP operation consists of two steps; firstly, we take N copies of the original protograph, then we permute the edges connecting the check and variable nodes, ensuring that the type of edge

connection is maintained. An example of the Tanner graph obtained using the CAP method for the protograph in Fig. 4.1 with $N = 3$ is illustrated in Fig. 4.2.

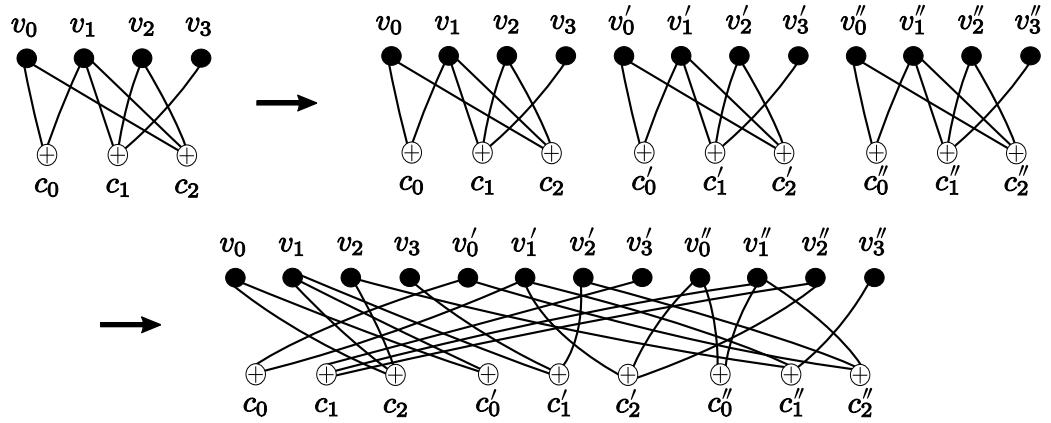


Figure 4.2: *The copy-and-permute operation for a protograph.*

A protograph-based block code with derived (or *lifted*) parity-check matrix H displays the following properties:

- the code has block length $n = Nn_v$;
- the code will have rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ provided that the rows of H are linearly independent;
- each variable and check node maintains the same number and type of edge connection;
- if a sufficiently large positive integer N is chosen, the resulting block code satisfies the low-density condition of an LDPC block code.

The underlying structure of protograph-based LDPC codes is essential for their analysis. This will be discussed further in Section 4.2.2.

The parity-check matrix H corresponding to a protograph-based LDPC block code can be obtained by replacing ones in the underlying protograph parity-check matrix P with $N \times N$ permutation matrices and zeros from P with $N \times N$ all-zeros matrices, where the permutation matrices are chosen randomly and independently. This will be illustrated in the following example.

Example 4.1. Consider the protograph with parity-check matrix

$$P = \begin{array}{c} \\ c_0 \\ c_1 \\ c_2 \end{array} \begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right] \end{array}_{n_c \times n_v = 3 \times 4}$$

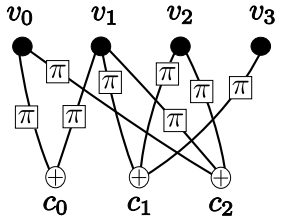
as given in Figure 4.1. The Tanner graph of the protograph-based code after applying the CAP operation displayed in Figure 4.2 can be written as follows:

$$H = \begin{array}{c} \\ c_0 \\ c_0' \\ c_0'' \\ c_1 \\ c_1' \\ c_1'' \\ c_2 \\ c_2' \\ c_2'' \end{array} \begin{array}{c} \begin{array}{ccccccccccc} v_0 & v_0' & v_0'' & v_1 & v_1' & v_1'' & v_2 & v_2' & v_2'' & v_3 & v_3' & v_3'' \end{array} \\ \left[\begin{array}{ccc:ccc:ccc:ccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}.$$

In protograph P, we observe, for example, that variable node v_0 connects to check node c_0. Applying the copy and permute operation, we maintain the protograph structure by ensuring that every variable node of type v_0 connects to a *distinct* check node of type c_0. Thus arranging H as we have done here, the $N \times N = 3 \times 3$ submatrix related to nodes of type v_0 and c_0 is in fact a 3×3 permutation matrix. We observe the same phenomenon for every connection in the base protograph with parity-check matrix P. If there is no connection, i.e., if the corresponding position of P is zero, then there are no connections of this type in H and the resulting 3×3 submatrix is the all-zeros matrix. □

After taking N copies of a protograph, there are of course $N!$ ways to permute the edges (or $N!$ choices of $N \times N$ permutation matrix to insert as a submatrix of H). An ensemble of protograph-based LDPC block codes can be created by allowing the $N \times N$ permutation matrices to vary. Every member of the resulting code ensemble has rate $R = 1 - n_c/n_v$ (provided that the parity-check matrices are of full rank), block length $n = Nn_v$, and the same degree distribution for each of its variable and check nodes as the underlying protograph.

Moreover, allowing N to be sufficiently large results in a protograph-based LDPC block code ensemble. The resulting ensemble of Example 4.1 is illustrated in Figure 4.3, where π denotes a permutor of length N and Q denotes a varying $N \times N$ permutation matrix.



$$P = \begin{bmatrix} Q & Q & 0 & 0 \\ 0 & Q & Q & Q \\ Q & Q & Q & 0 \end{bmatrix}$$

Figure 4.3: *The protograph-based LDPC block code ensemble of Example 4.1.*

We note that in Example 4.1 the row and column weights of P are not constant, so P represents the parity-check matrix of an *irregular* protograph. The resulting ensemble of protograph-based block codes will thus also be irregular since the structure is preserved. Note that it is also possible to consider protograph parity-check matrices P with larger integer entries, which represent parallel edges in the base protograph. In this case, the resulting block in H consists of a sum of $N \times N$ permutation matrices [Tho03]. For binary codes, we ensure that the permutation matrices involved in any summation do not overlap, i.e., none of the entries of H should be larger than one. Thus we require $N \geq p_{max}$ where p_{max} is the largest integer entry in P. This point is illustrated in the following example.

Example 4.2. Consider the irregular protograph with parity check matrix

$$P = \begin{bmatrix} 2 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \end{bmatrix}.$$

Let $N = 3$. Then a resulting member of the protograph-based ensemble could have parity-check matrix

$$H = \left[\begin{array}{ccc:ccc:ccc:ccc} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hdashline 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right].$$

The resulting code has rate $R = 1 - Nn_c/Nn_v = 1/2$ and block length $n = Nn_v = 12$. We observe that the edge connections and types of this protograph-based code are maintained when summing the permutation matrices, and thus the row and column weights remain constant. □

4.2.2 Properties of protograph-based codes

In the previous section we saw that a protograph is a small bipartite graph which can be used to construct a larger code with an underlying structure. As the constructed code has the same rate and degree distribution of variable and check nodes as the base protograph, it also possesses the same iterative decoding threshold. This occurs because the iterative decoding threshold is a function of only the degree distribution. The computation of thresholds can be achieved using the approach presented in [Chu00], which provides an accurate approximation to density evolution. So-called Repeat-Accumulate (RA) [DJM98] codes, Irregular Repeat-Accumulate (IRA) [JKM00] codes and Accumulate-Repeat-Accumulate (ARA) [ADY07] all have simple protograph representations, and constructions described there have been shown to achieve very low iterative decoding thresholds.

However, the construction schemes described above have sublinear minimum distance. In the interests of having a low error floor, it is desirable to construct codes that are *asymptotically good*, i.e., they have the property that the minimum distance grows linearly with block length. Protographs that achieve both linear minimum distance growth and low iterative decoding thresholds are thus of great interest. Good constructions can be found, for example, in [DJDT05, DDJ06]. It can be shown that protograph-based ensembles are asymptotically good by analysing ensemble weight enumerators as the block length $n \to \infty$. In Section 4.3, we review a method proposed by Divsalar [Div06] to calculate ensemble weight enumerators for protograph-based LDPC code ensembles. Here, it is shown again that the protograph structure is crucial for the analysis as we need only consider the base protograph in the combinatorial calculations.

Another factor that can affect the performance of the code in the error floor region of the BER curve are so called *trapping sets* (see, e.g., [Ric03]) or *near codewords*. In Section 4.4 we will formally introduce this concept and review the approach by Abu-Surra, Ryan and Divsalar [ARD07a] where we see again that the protograph structure can be employed successfully in order to calculate ensemble trapping set enumerators. These enumerators can then be used to give an estimate of the error floor for protograph-based LDPC code ensembles.

4.3 Protograph weight enumerators

An *ensemble average weight enumerator* A_d tells us that given a particular Hamming weight d, an average member of the ensemble has A_d codewords of Hamming weight d.[1] In 1962, Gallager calculated asymptotic codeword weight enumerators for regular LDPC block code ensembles [Gal62, Gal63]. Recently, this result has been extended to consider even unstructured irregular ensembles [LS02]. The method of constructing LDPC block codes based on protographs was discussed in Section 4.2.1, and it was was suggested in Section 4.2.2 that the structure of protograph-based codes could prove useful for calculating ensemble weight enumerators. Combinatorial methods of calculating ensemble average weight enumerators for protograph-based block codes have been presented in [FMT05] and [Div06]. These results have since been extended to protograph-based generalised LDPC codes in [ARD07b]. In this section, we summarise the results of [Div06] and derive expressions that will be used in Chapter 5.

4.3.1 Ensemble weight enumerators

Suppose we take N copies of the protograph to form the parity-check matrix for a protograph-based code. Let $A_{\mathbf{w}_i}^{c_i}$ denote the number of sequences that satisfy (all N copies of) check node c_i given *input weight vector* $\mathbf{w}_i = [\, w_{i,0} \;\; w_{i,1} \;\; \ldots \;\; w_{i,q_{c_i}-1} \,]$, where $0 \leq w_{i,j} \leq N$ for each of the j attached variable nodes. The input weight vector defines the weight distribution for the q_{c_i} variable nodes attached to check node c_i, i.e., after the CAP operation, the sum of the (binary) input weights to the N variable nodes of type j (i.e., the N nodes indexed by j) is equal to $w_{i,j}$. For a check node of degree three this can be counted as [Div06]

$$A_{w_0,w_1,w_2} = \binom{N}{s} \frac{s!}{(s-w_0)!(s-w_1)!(s-w_2)!},$$

where $s = \frac{w_0+w_1+w_2}{2}$, $\max\{w_0, w_1, w_2\} \leq s \leq N$ and $w_0 + w_1 + w_2$ is even; otherwise $A_{w_0,w_1,w_2} = 0$. A simple combinatorial argument can be used to prove this, see, e.g., [Div06].

Example 4.3. Figure 4.4 displays a protograph consisting of a degree 3 check node, and corresponding Tanner graph after taking $N = 3$ copies.

[1] Note that ensemble average weight enumerator A_d need not be an integer.

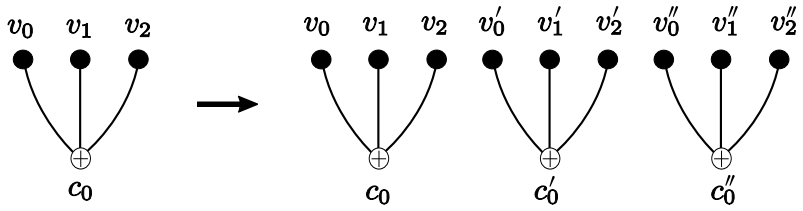


Figure 4.4: *A check node of degree* 3 *before and after the CAP operation with* $N = 3$.

(a) Consider the weight distribution $\mathbf{w} = [w_0 \ w_1 \ w_2] = [2 \ 1 \ 1]$. We calculate $s = (w_0 + w_1 + w_2)/2 = 2$, and $\max\{w_0, w_1, w_2\} = 2 \leq s = 2 \leq N = 3$. Thus, $A_{w_0,w_1,w_2} = 6$ from equation (4.3.1). The resulting codewords are shown in Table 4.1

c_0			c_0'			c_0''		
v_0	v_1	v_2	v_0'	v_1'	v_2'	v_0''	v_1''	v_2''
1	1	0	1	0	1	0	0	0
1	1	0	0	0	0	1	0	1
1	0	1	1	1	0	0	0	0
1	0	1	0	0	0	1	1	0
0	0	0	1	1	0	1	0	1
0	0	0	1	0	1	1	1	0

Table 4.1: *Codewords satisfying the weight distribution of Example 4.3(a).*

We observe that every codeword listed in Table 4.1 satisfies the weight distribution given by input weight vector $\mathbf{w} = [w_0 \ w_1 \ w_2] = [2 \ 1 \ 1]$. The $N = 3$ nodes of type $j = 0$ (i.e., indexed by $j = 0$) have sum $v_0 + v_0' + v_0'' = 2 = w_0$. Similarly, the nodes indexed by $j = 1$ have sum $v_1 + v_1' + v_1'' = 1 = w_1$ and the nodes indexed by $j = 2$ have sum $v_2 + v_2' + v_2'' = 1 = w_2$. It is clear that there are no other ways to satisfy all three check nodes under this weight distribution.

(b) Consider the weight distribution $\mathbf{w} = [w_0 \ w_1 \ w_2] = [1 \ 1 \ 1]$. As $w_0 + w_1 + w_2$ is not even, we have $A_{w_0,w_1,w_2} = 0$. It is clear that there is no way to satisfy the three check nodes with this weight distribution. □

To calculate enumerators for a check node of degree 2 we can simply use the formulae for a check node of degree 3 with weight 0 attached to the third input and the other two weight inputs defined to be equal. This observation is detailed and derived in Proposition A.1. Further, we can obtain enumerators for check nodes with degree larger than three by concatenation of degree 3 check nodes [Div06].

Having obtained an expression for the number of ways to satisfy any particular check node, we may now form an ensemble average weight enumerator. Suppose that a protograph contains m variable nodes connected to the channel and $n_v - m$ punctured variable nodes. Also, suppose that each of the m transmitted variable nodes has an associated weight d_i, where $0 \leq d_i \leq N$ for all i.[2] Let $S_d = \{(d_0, d_1, \ldots, d_{m-1})\}$ be the set of all possible weight distributions such that $d_0 + \ldots + d_{m-1} = d$, and let S_p be the set of all possible weight distributions for the remaining punctured nodes. After performing the CAP operation N times, the ensemble average weight enumerator for codewords with Hamming weight d for a protograph P is then given by [Div06]

$$A_d = \sum_{\{d_k\} \in S_d} \sum_{\{d_j\} \in S_p} A_{\mathbf{d}}, \tag{4.1}$$

where

$$A_{\mathbf{d}} = \frac{\Pi_{i=1}^{n_c} A_{\mathbf{d}_i}^{c_i}}{\Pi_{i=1}^{n_v} \binom{N}{d_i}^{q_{v_i}-1}}$$

is the average number of codewords in the ensemble with a particular weight distribution $\mathbf{d} = (d_0, d_1, \ldots, d_{n_v-1})$. Here, the vector $\mathbf{d}_i$ is the input weight vector to check node c_i.

4.3.2 Asymptotic weight enumerators

The normalized logarithmic asymptotic weight distribution or *asymptotic spectral shape function* of a code ensemble can be written as $r(\delta) = \limsup_{n\to\infty} r_n(\delta)$, where $r_n(\delta) = \frac{\ln(A_d)}{n}$, $\delta = d/n$, d is the Hamming weight, n is the block length, and A_d is the ensemble average weight distribution given in (4.1). For protograph-based codes, we obtain the expression $r(\delta) = \frac{1}{m}\tilde{r}(m\delta)$ where [Div06]

$$\tilde{r}(\tilde{\delta}) = \max_{\{\delta_k\} \in S_{\tilde{\delta}}} \max_{\{\delta_j\} \in S_\pi} \limsup_{N\to\infty} \left(\sum_{i=1}^{n_c} \frac{\ln A_{\mathbf{d}_i}^{c_i}}{N} - \sum_{i=1}^{n_v} \frac{(q_{v_i} - 1) \ln \binom{N}{d_i}}{N} \right),$$

and the sets $S_{\tilde{\delta}}$ and S_π are normalized versions of S_d and S_p respectively (each component is divided by N as $N \to \infty$).

[2] Since we use N copies of the protograph, the weight associated with a particular variable node in the protograph can be as large as N.

By defining the *asymptotic check node weight enumerator* as

$$a^{c_i}(\boldsymbol{\delta}_i) \triangleq \limsup_{N\to\infty} \left(\ln \frac{A^{c_i}_{\mathbf{d}_i}}{N} \right),$$

where $\boldsymbol{\delta}_i$ is the vector $\mathbf{d}_i$ normalized componentwise by N, we obtain an alternative formulation for $r(\delta)$ as follows [Div06]:

$$\tilde{r}(\tilde{\delta}) = \max_{\{\delta_k\}\in S_{\tilde{\delta}}} \max_{\{\delta_j\}\in S_\pi} \left(\sum_{i=1}^{n_c} a^{c_i}(\delta_i) - \sum_{i=1}^{n_v} (q_{v_i}-1) H(\delta_i) \right),$$

where $H(x) = -(1-x)\ln(1-x) - x\ln x$ is the entropy function. Expressions for the asymptotic weight enumerators were stated in [Div06], to help the reader these are derived in Section A.2.

Suppose the first zero crossing of $r(\delta)$ occurs at $\delta = \delta_{min}$. If $r(\delta)$ is negative in the range $0 < \delta < \delta_{min}$, then δ_{min} is called the *minimum distance growth rate* of the code ensemble. By considering the probability

$$\mathbb{P}(d < \delta_{min} n) = \sum_{d=1}^{\delta_{min} n - 1} A_d,$$

it is clear that, as the block length n grows, if $\mathbb{P}(d < \delta_{min} n) << 1$, then we can say with high probability that the majority of codes in the ensemble have a minimum distance that grows linearly with n and that the minimum distance growth rate is δ_{min}.

Example 4.4. Consider the $(3, 6)$-regular ensemble generated by the protograph with parity-check matrix

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Using the method presented in [Div06], we can easily obtain an expression for the asymptotic spectral shape function $r(\delta)$ of the resulting $(3, 6)$-regular LDPC block code ensemble. The asymptotic spectral shape function is plotted against the normalised weight δ in Figure 4.5. We observe that $\delta_{min} = 0.023$ for this ensemble, as originally computed by Gallager [Gal62]. □

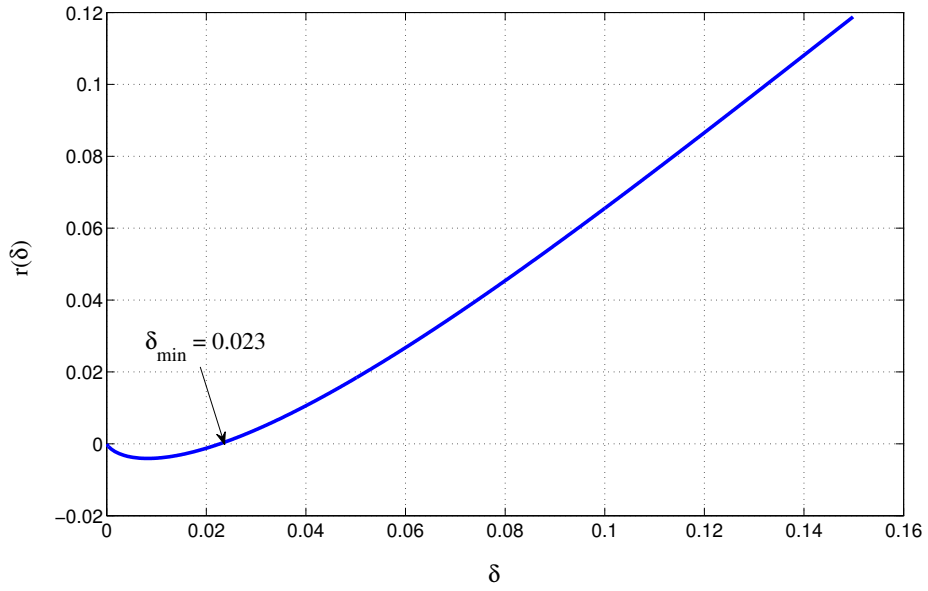


Figure 4.5: *The asymptotic spectral shape function for Example 4.4.*

4.4 Trapping sets of protograph-based codes

In [MP03], MacKay and Postol discovered a "weakness" in the structure of the Margulis construction of a $(3, 6)$-regular Gallager code. Described as *near-codewords*, these small graphical sub-structures existing in the Tanner graph of Low-Density Parity-Check (LDPC) codes cause the iterative decoding algorithm to get trapped in error patterns. These weaknesses were shown to contribute significantly to the performance of the code in the error floor, or high signal-to-noise ratio (SNR), region of the bit error rate (BER) curve. Richardson developed this concept in [Ric03], and defined these structures as *trapping sets*. In this paper, trapping sets were shown to have a strong influence on the position and slope of the error floor and the author proposed a two-stage technique to predict the error floor performance of LDPC codes based on computer search strategies: first, we must search the Tanner graph for candidate trapping sets; then their contribution to the error floor is evaluated. Asymptotic results on trapping set enumerators for both regular and irregular LDPC block code ensembles were published in [MSW07].

Previously, such weaknesses of LDPC codes on the binary erasure channel (BEC) had been defined as *stopping sets* [DPT$^+$02], which can be viewed as trapping sets on the BEC under belief propagation (BP) decoding. Stopping sets also play a role on Gaussian channels but are not dominant. In [ZDN$^+$06], the authors propose another important subclass of trapping

sets called *absorbing sets*. An absorbing set is defined as a decoder-independent combinatorial object that can be studied analytically. So-called *fully absorbing sets* have been shown to be stable under bit-flipping operations [DLZ+09].

In this section, we review the approach and results of Abu-Surra, Ryan and Divsalar [ARD07a], which will later be used to calculate trapping set enumerators for protograph-based LDPC convolutional codes in Chapter 6. In Section 4.4.1, we define general trapping sets and show how a modified protograph structure can be employed successfully in order to calculate easily enumerators of these sets. Section 4.4.2 introduces an important subset of general trapping sets called elementary trapping sets which have been shown to have a large (negative) affect on the performance of codes in the error floor region. Finally, we discuss in Section 4.4.3 how to form ensemble trapping set enumerators, and it is observed that for some asymptotically good ensembles, the smallest trapping set grows linearly with block length.

4.4.1 General trapping set enumerators for protograph-based codes

Definition 4.1 *An (a, b) general trapping set $\tau_{a,b}$ of a bipartite graph is a set of variable nodes of size a which induce a subgraph with exactly b odd-degree check nodes (and an arbitrary number of even-degree check nodes).*

An *ensemble average general trapping set enumerator* $A_{a,b}$ tells us that given particular general trapping set parameters a and b, an average member of the ensemble has $A_{a,b}$ (a, b) general trapping sets. In order to calculate ensemble average general trapping set enumerators for protograph-based block codes, we make use of the combinatorial arguments discussed in Section 4.3 for calculating ensemble average weight enumerators [Div06, FMT05]. The technique involves considering a two-part weight enumerator for a modified protograph with the property that any (a, b) trapping set in the original protograph is a codeword in the modified protograph. We now briefly describe the procedure introduced in [ARD07a]. An auxiliary 'flag' variable node is added to each check node, as displayed in Figure 4.6.

Consider a subset S with cardinality a of the variable nodes $V = \{v_0, v_1, v_2, v_3\}$, for example $a = 3$ and $S = \{v_0, v_1, v_2\}$. We now attach weight 1 to these variable nodes and weight 0 to the remaining nodes in $V \backslash S = \{v_3\}$. We observe that check nodes c_0 and c_1 are satisfied[3],

[3]Recall that a check node corresponds to a row of a parity-check matrix. In this sense, if the check node has an even number of connections the parity-check sum is equal to zero modulo 2 and that particular sum (or check node) is deemed to be *satisfied*. If the check sum is equal to one modulo 2 (or the check node has an odd number of connections) the parity-check equation fails and we say the node is *unsatisfied*.

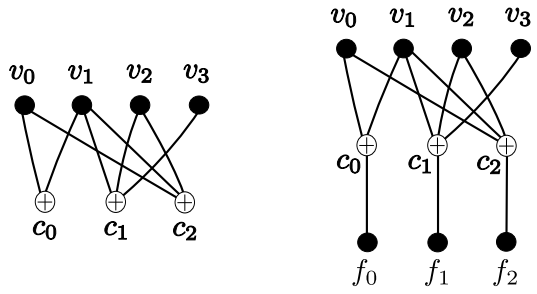


Figure 4.6: *An example protograph and modified version with auxiliary variable nodes.*

since they both have input weight 2, but that check node c_2 (with input weight 3) is unsatisfied. Thus there is $b = 1$ odd-degree (unsatisfied) check node. This is an example of a $(3, 1)$ general trapping set. Thus $\tau_{3,1}$ contains the subset $S = \{v_0, v_1, v_2\}$.

For any subset of variable nodes, we can satisfy any odd-degree check nodes by assigning weight 1 to the corresponding auxiliary variable node. Note that the weight of the variable nodes V is $a = 3$ and the weight of the auxiliary nodes is $b = 1$ for this $(3, 1)$ general trapping set, which suggests that the general trapping sets of a protograph can be enumerated by applying a two-part weight enumerator analysis to the modified protograph. We thus consider a two-part weight enumerator over sets of variable nodes $\{v_0, v_1, \ldots, v_{n_v-1}\}$ and auxiliary nodes $\{f_0, f_1, \ldots, f_{n_f-1}\}$, where n_v is the number of variable nodes in the initial protograph and n_f is the number of auxiliary variable nodes (equal to the number of check nodes n_c). This method of enumerating trapping sets for protograph-based codes is presented in [ARD07a].

4.4.2 Elementary trapping set enumerators

Definition 4.2 *An (a, b) elementary trapping set $\tau^e_{a,b}$ of a bipartite graph is a set of variable nodes of size a which induce a subgraph with only degree-one and degree-two check nodes, and exactly b degree-one check nodes.*

Extensive simulations (see, e.g., [Ric03]) have shown that in fact most of the decoding failures in iterative decoding correspond to elementary trapping sets. Enumerators for elementary trapping sets can be calculated in exactly the same way as those for general trapping sets after introducing extra constraints on the check node degrees; see [ARD07a] for further details. An example elementary trapping set is illustrated in Figure 4.7.

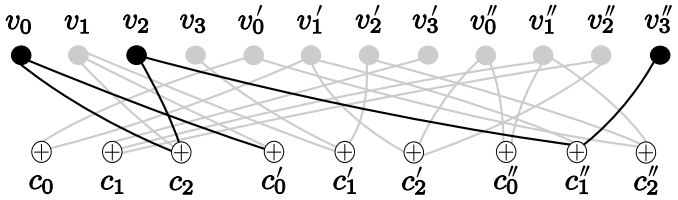


Figure 4.7: *A* $(3, 1)$ *elementary trapping set in the Tanner graph of Example* 4.1.

4.4.3 Ensemble trapping set enumerators

Suppose that a modified protograph contains n_v variable nodes connected to the channel. Also, suppose that each of the n_v transmitted variable nodes has an associated weight d_i, where $0 \leq d_i \leq N$ for all i.[4] Let $S_d = \{(d_0, d_1, \ldots, d_{n_v-1})\}$ be the set of all possible weight distributions such that $d_0 + \ldots + d_{n_v-1} = a$. Finally, suppose that $S_f = \{(f_0, f_1, \ldots, f_{n_f-1})\}$ is the set of all weight distributions such that $f_0 + \ldots + f_{n_f-1} = b$, where $0 \leq f_i \leq N$ for all i. Then the two-part ensemble average trapping set enumerator for the modified protograph is given by

$$A_{a,b} = \sum_{\{d_k\} \in S_d} \sum_{\{f_l\} \in S_f} A_{\mathbf{d}},$$

where $A_{\mathbf{d}}$ is the average number of codewords in the modified ensemble with weight distribution

$$\mathbf{d} = (d_0, d_1, \ldots, d_{n_v-1}, f_0, f_1, \ldots, f_{n_f-1}).$$

The two-part normalized logarithmic *asymptotic trapping set spectral shape function* of a code ensemble can be written as

$$r(\alpha, \beta) = \limsup_{n \to \infty} r_n(\alpha, \beta),$$

where $r_n(\alpha, \beta) = \frac{\ln(A_{a,b})}{n}$, $\alpha = a/n$, $\beta = b/n$, a and b are Hamming weights, n is the block length, and $A_{a,b}$ is the two-part ensemble average weight distribution. Figure 4.8 shows the asymptotic trapping set spectral shape function for the $(3, 6)$-regular LDPC block code ensemble of Example 4.4 for various fixed values of β.

[4] Since we use N copies of the protograph, the weight associated with a particular variable node in the protograph can be as large as N.

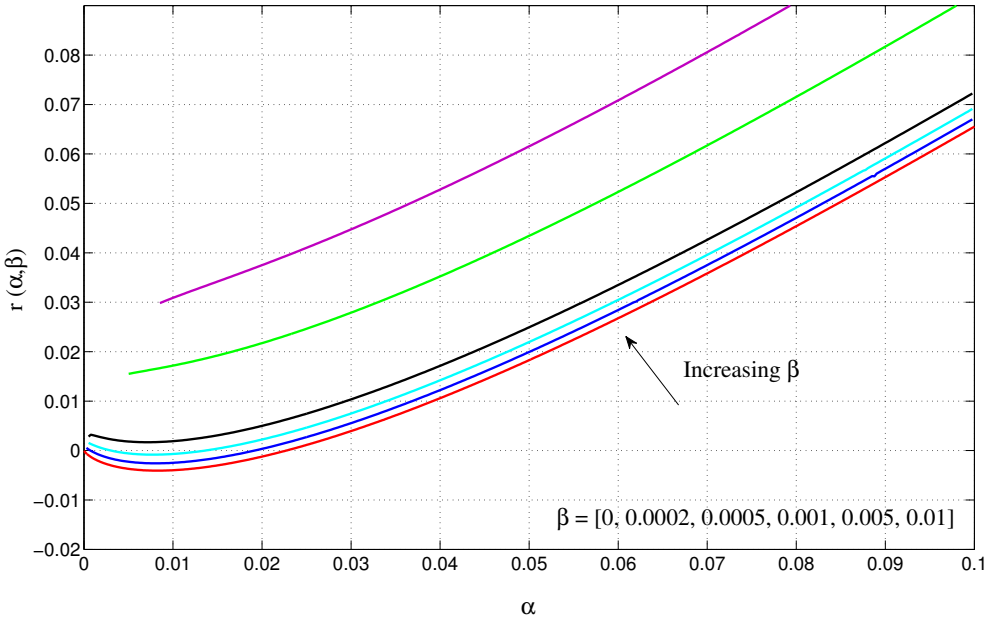


Figure 4.8: *The asymptotic trapping set spectral shape function for Example 4.4 for various fixed values of β.*

Suppose now we are interested in the ratio of b to a for a general (a, b) trapping set enumerator. Let $\Delta = b/a = \beta/\alpha$, $\Delta \in [0, \infty)$. As proposed in [ARD07a], we may now classify the trapping sets as

$$\tau_\Delta = \{\tau_{a,b} | b = \Delta \cdot a\}.$$

For each Δ, we define $d_{ts}(\Delta)$ to be the Δ-*trapping set number*, which is the size of the smallest, non-empty trapping set in τ_Δ. Now consider fixing Δ and plotting the normalized weight α against the two-part asymptotic spectral shape function $r(\alpha, \beta) = r(\alpha, \Delta\alpha)$. Suppose $\alpha > 0$ and the first zero-crossing of $r(\alpha, \beta)$ occurs at $\alpha = \delta_{ts}(\Delta)$. If $r(\alpha, \beta)$ is negative in the range $0 < \alpha < \delta_{ts}(\Delta)$, then the first zero-crossing $\delta_{ts}(\Delta)$ is called the Δ-*trapping set growth rate* of the code ensemble. If $\delta_{ts}(\Delta)$ exists, and if the probability

$$\mathbb{P}(a < \delta_{ts}(\Delta)n) = \sum_{a=1}^{\delta_{ts}(\Delta)n-1} A_{a,b} << 1$$

as the block length n grows, we can say with high probability that the majority of codes in the ensemble have a Δ-trapping set number that increases linearly with n, i.e., $d_{ts}(\Delta) = n\delta_{ts}(\Delta)$. This implies that, for sufficiently large n, a typical member of the ensemble has no small trapping sets. Figure 4.9 shows the asymptotic trapping set spectral shape function for the $(3, 6)$-regular LDPC block code ensemble of Example 4.4 for various fixed values of Δ.

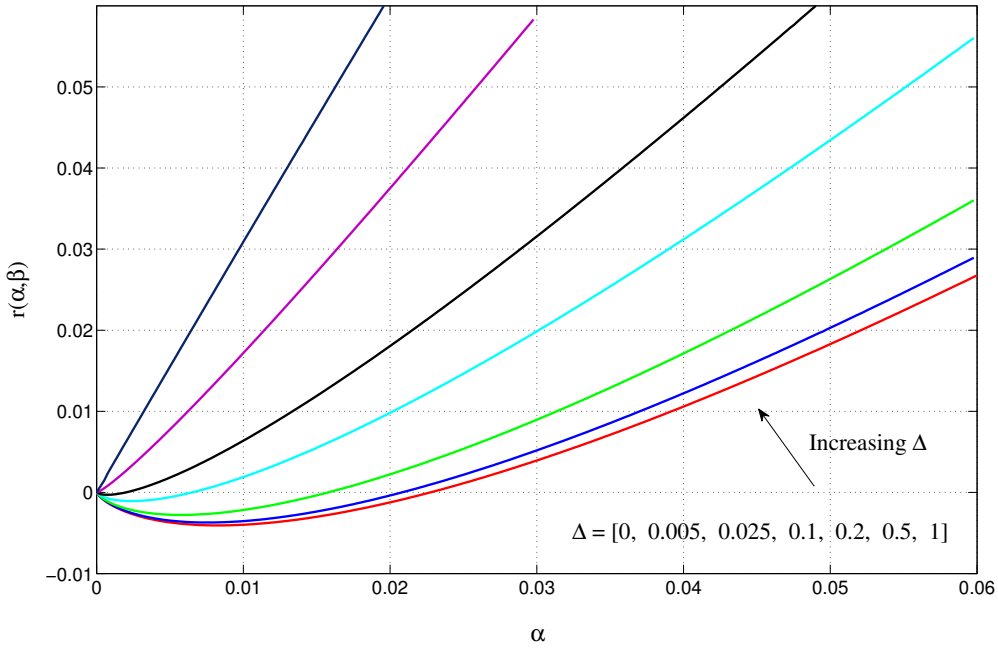


Figure 4.9: *Asymptotic general trapping set enumerators for Example 4.4 for various fixed values of* Δ.

Experimental results have indicated that the failure events that dominate the performance of LDPC codes with iterative decoding in the error floor region of the BER curve can be attributed to small trapping sets [Ric03]. Since small trapping sets contain relatively few variable nodes, it is more likely that all of the variable nodes in a small trapping set have unreliable channel values than in a larger trapping set. Further, examination of the small trapping sets dominating in the error floor region has shown that the sets with low degree check nodes cause the most trouble. This is because of the low connectivity of these check nodes to the rest of the graph, which results in the trapping set not being able to obtain enough independent (and possibly helpful) messages during decoding iterations. Thus an ensemble that guarantees no small trapping sets on average is highly desirable.

Chapter 5
Asymptotically Good LDPC Convolutional Codes based on Protographs

Abstract

LDPC convolutional codes have been shown to be capable of achieving the same capacity-approaching performance as LDPC block codes with iterative message-passing decoding. In this chapter, asymptotic methods are used to form lower bounds on the free distance to constraint length ratio of several ensembles of asymptotically good, protograph-based LDPC convolutional codes. Further, it is shown that the free distance to constraint length ratio of the LDPC convolutional codes exceeds the minimum distance to block length ratio of corresponding LDPC block codes. A selection of the results presented here has previously been published in [MPZC08] and [MPGC08].

5.1 Introduction

In this chapter, we use ensembles of tail-biting LDPC convolutional codes (introduced in [TZF07]) derived from a protograph-based ensemble of LDPC block codes to obtain a lower bound for the free distance of unterminated, asymptotically good, time-varying LDPC convolutional code ensembles, i.e., ensembles that have the property of free distance growing linearly with constraint length.

In the process, it is shown that the minimum distances of ensembles of tail-biting LDPC convolutional codes approach the free distance of an associated unterminated, time-varying LDPC convolutional code family as the block length of the tail-biting ensemble increases. We also show, for rate $1/2$ protograph-based ensembles with regular degree distributions, that the free distance bounds are consistent with those recently derived for more general regular LDPC convolutional code ensembles in [STL$^+$07] and [TZC09]. Further, the relatively low complexity requirements of computing the bound allows us to calculate new free distance bounds that grow linearly with constraint length for values of J and K that have not been previously considered in the literature. We show, for all the (J, K)-regular ensembles considered, that the free distance to constraint length ratio exceeds the minimum distance to block length ratio of the corresponding block codes. Further, for protographs with irregular degree distributions, we obtain new free distance bounds that grow linearly with constraint length and whose free distance to constraint length ratio also exceeds the minimum distance to block length ratio of the corresponding irregular block codes.

The chapter is structured as follows. In Section 5.2, the notion of a convolutional protograph is introduced along with several new construction techniques for protograph-based LDPC convolutional codes. In Section 5.3, the construction of tail-biting LDPC convolutional codes is described. It is then shown that the free distance of a (periodically) time-varying LDPC convolutional code is lower bounded by the minimum distance of the block code formed by terminating it as a tail-biting LDPC convolutional code. Finally, in Section 5.4 new results are presented on the free distance of ensembles of regular and irregular LDPC convolutional codes based on protographs.

5.2 Protograph-based LDPC convolutional codes

The convolutional counterpart of LDPC block codes was described in Section 1.5.2. LDPC convolutional codes have been shown to have certain advantages compared to LDPC block codes of the same complexity [CPBZ06,CPJD07]. In this section, we will use the copy-and-permute operation (introduced in Section 4.2) to construct protograph-based (LDPC) convolutional codes. We achieve this by forming a convolutional protograph that has the same number and type of edge connections as the base block protograph that it is based on, and thus maintains many of the desirable properties of protograph-based codes discussed in Section 4.2.

Throughout this section, we will suppose that a protograph, with protograph parity-check matrix P, consists of n_c check nodes and n_v variable nodes. We wish to form a convolutional protograph and this is achieved by 'unwrapping' the protograph with protograph parity-check matrix P. Let $y = \text{gcd}(n_c, n_v)$. Then, depending on the value of y, we will define different unwrapping procedures.

5.2.1 Unwrapping a protograph with gcd$(n_c, n_v) > 1$

Suppose that we have an $n_c \times n_v$ protograph parity-check matrix P, where $\text{gcd}(n_c, n_v) = y > 1$. We then partition P as a $y \times y$ block matrix as follows:

$$P = \begin{bmatrix} P_{1,1} & \dots & P_{1,y} \\ \vdots & & \vdots \\ P_{y,1} & \dots & P_{y,y} \end{bmatrix},$$

where each block $P_{i,j}$ is of size $n_c/y \times n_v/y$. P can now be separated into a lower triangular part, P_l, and an upper triangular part minus the leading diagonal, P_u. Explicitly,

$$P_l = \begin{bmatrix} P_{1,1} & & & \\ P_{2,1} & P_{2,2} & & \\ \vdots & \vdots & \ddots & \\ P_{y,1} & P_{y,2} & \dots & P_{y,y} \end{bmatrix} \text{ and } P_u = \begin{bmatrix} & P_{1,2} & \dots & P_{1,y} \\ & & \ddots & \vdots \\ & & & P_{y-1,y} \\ & & & \end{bmatrix},$$

where blank spaces correspond to zeros. This operation is called 'cutting' a protograph parity-check matrix.

Rearranging the positions of these two triangular matrices and repeating them indefinitely results in a protograph parity-check matrix P_{cc} of an unterminated, periodically time-varying convolutional code with decoding constraint length $\nu_s = n_v$ and, in general, period $T = y$, given by[1]

$$P_{cc} = \begin{bmatrix} P_l & & & \\ P_u & P_l & & \\ & P_u & P_l & \\ & & \ddots & \ddots \end{bmatrix}. \tag{5.1}$$

Note that the unwrapping procedure described above preserves the row and column weights of the protograph parity-check matrix. The Tanner graph of P_{cc} can be viewed as a *convolutional protograph*.

5.2.2 Examples of unwrapped protograph-based convolutional codes

As mentioned in Section 5.2.1, the period of a convolutional code with protograph (or protograph-based) parity check matrix is dependent on the structure of the initial protograph. Here, we observe some toy examples to illustrate the possible periods and structures of the unwrapped codes.

Example 5.1. Consider the following protograph parity-check matrix,

$$P = \begin{bmatrix} 2 & 1 & 0 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 0 & 1 & 2 \end{bmatrix}.$$

For this example, we calculate $\gcd(n_c, n_v) = \gcd(3, 6) = 3 = y$. Thus the resulting rate $R = 1 - n_c/n_v = 1/2$ convolutional code has infinite parity-check matrix

[1]The period of the convolutional code is dependent on the structure of the protograph. This will be discussed further in Section 5.2.2.

$$P_{cc} = \left[\begin{array}{cccccc|cc|c}
2 & 1 & & & & & & & \\
0 & 1 & 1 & 1 & & & & & \\
1 & 2 & 1 & 0 & 1 & 2 & & & \\
\hline
 & & 0 & 1 & 2 & 1 & 2 & 1 & \\
 & & & & 2 & 1 & 0 & 1 & 1 & 1 \\
 & & & & & & 1 & 2 & 1 & 0 & 1 & 2 \\
\hline
 & & & & & & & \ddots & & \ddots
\end{array}\right].$$

We observe that the decoding constraint length is $\nu_s = n_v = 6$. The submatrices $H_i(t)$ are of size $n_c/y \times n_v/y = 1 \times 2$, and $H_i(t) = H_i(t+3)\ \forall\ i, t$. Thus $T = y = 3$ for this example, and by construction $y|n_v$. □

This periodic property holds for different rate codes, as we observe in the next example.

Example 5.2. Consider the following rate $R = 1/3$ protograph parity-check matrix,

$$P = \left[\begin{array}{ccc|ccc}
1 & 0 & 3 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 2 \\
2 & 1 & 2 & 1 & 1 & 1
\end{array}\right].$$

For this example, we calculate $\gcd(n_c, n_v) = \gcd(4, 6) = 2 = y$. Thus we cut P with step size $n_c/y \times n_v/y = 2 \times 3$ and the resulting rate $R = 1 - n_c/n_v = 1/3$ convolutional code has infinite parity-check matrix

$$P_{cc} = \left[\begin{array}{ccc|ccc|ccc|c}
1 & 0 & 3 & & & & & & & \\
0 & 1 & 1 & & & & & & & \\
1 & 0 & 0 & 1 & 1 & 2 & & & & \\
2 & 1 & 2 & 1 & 1 & 1 & & & & \\
\hline
 & & & 0 & 1 & 1 & 1 & 0 & 3 & \\
 & & & 1 & 1 & 0 & 0 & 1 & 1 & \\
 & & & & & & 1 & 0 & 0 & 1 & 1 & 2 \\
 & & & & & & 2 & 1 & 2 & 1 & 1 & 1 \\
\hline
 & & & & & & & & \ddots & & & \ddots
\end{array}\right].$$

The decoding constraint length is $\nu_s = n_v = 6$. The submatrices $H_i(t)$ are of size $n_c/y \times n_v/y = 2 \times 3$, and $H_i(t) = H_i(t+y) = H_i(t+2)\ \forall\ i, t$. Thus $T = y = 2$ for this example, and by construction $y|n_v$. □

Depending on any sub-structures in P, it is possible to have a smaller period $y' \in \mathbf{Z}^+$ where $y'|y$. This can be seen in the next example.

Example 5.3. Consider the protograph parity-check matrix

$$P = \begin{bmatrix} 1 & 0 & 0 & 1 & 2 & 1 & 1 & 3 \\ 1 & 1 & 2 & 1 & 1 & 0 & 0 & 2 \\ 2 & 1 & 1 & 3 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 1 & 1 & 2 & 1 \end{bmatrix}.$$

We calculate $y = \gcd(4, 8) = 4$, and thus we 'cut' P with step size $n_c/y \times n_v/y = 1 \times 2$. The resulting infinite convolutional protograph parity-check matrix is

$$P_{cc} = \begin{bmatrix} 1 & 0 & & & & & & & & & & & & \\ 1 & 1 & 2 & 1 & & & & & & & & & & \\ 2 & 1 & 1 & 3 & 1 & 0 & & & & & & & & \\ 1 & 0 & 0 & 2 & 1 & 1 & 2 & 1 & & & & & & \\ & & 0 & 1 & 2 & 1 & 1 & 3 & 1 & 0 & & & & \\ & & & & 1 & 0 & 0 & 2 & 1 & 1 & 2 & 1 & & \\ & & & & & & 0 & 1 & 2 & 1 & 1 & 3 & 1 & 0 \\ & & & & & & & & 1 & 0 & 0 & 2 & 1 & 1 \\ & & & & & & & & & & & \ddots & & \ddots \end{bmatrix}.$$

The submatrices $H_i(t)$ are of equal size to the step size $n_c/y \times n_v/y = 1 \times 2$. The period of this convolutional code is actually $y' = 2$ because of the sub-structures in P, and thus $H_i(t) = H_i(t+y') = H_i(t+2)\ \forall\ i, t$. Note that $y'|y$, and thus we also have $H_i(t) = H_i(t+2y') = H_i(t+y) = H_i(t+4)\ \forall\ i, t$. Thus $T = y' = 2$ for this example. □

By construction, the equation

$$H_i(t) = H_i(t+y) \tag{5.2}$$

holds for all i, t. We now formalise the observation that the smallest period y' of P_{cc} must satisfy $y'|y$.

Proposition 5.1 *Suppose we construct a convolutional protograph parity-check matrix P_{cc} as defined in equation (5.1). Let the submatrix size of each $H_i(t)$ be $n_c/y \times n_v/y$, then $H_i(t) = H_i(t+y)\ \forall\ i,t$ from equation (5.2). Suppose $y' \in \mathbb{Z}^+$ is the smallest period of P_{cc}, i.e., $H_i(t) = H_i(t+y')\ \forall\ i,t$ and $y' \leq y$ with no smaller period. Then $y'|y$.*

Proof. Define $y'' = \gcd(y, y')$. If $y'' = y$ then $T = y = y'$ is the smallest period of P_{cc}. If $y'' = y'$ then $T = y''$ and $y'' = y'|y$. We now show that there are no other possible situations. Note that if the convolutional code is periodic with periods y and y' then $H_i(t) = H_i(t + ky + ly')$ $\forall\ i,t$ for any $k,l \in \mathbb{Z}$. If $1 < y'' < y'$, then from the Euclidean algorithm there exist k,l such that $ky + ly' = y''$ and thus $H_i(t) = H_i(t+y'')\ \forall\ i,t$, and we have a contradiction as y' is not the smallest period.

Finally, if y and y' are relatively prime, i.e., $y'' = 1$, then $T = y' = y'' = 1$ (the convolutional code is actually time-invariant). This can be easily seen again from the Euclidean algorithm. If $\gcd(y, y') = 1$ then there exist $k,l \in \mathbb{Z}$ such that $ky + ly' = 1$. Then it follows that $H_i(t) = H_i(t+1)\ \forall\ i,t$. □

Example 5.4. Suppose P_{cc} is periodic with period $T = y = 4$. Suppose there exists $\hat{y} = 3$ such that $H_i(t) = H_i(t+\hat{y})\ \forall\ i,t$. Then $H_i(t) = H_i(t+4k+3l)\ \forall\ i,t$ and for any $k,l \in \mathbb{Z}$. As $\gcd(3,4) = 1$, there exist k,l such that $4k + 3l = 1$. We note that, for example $k = -2$ and $l = 3$ satisfy the equation. We observe that this is indeed a solution, as if the code is periodic with period $y = 4$ then $H_i(t) = H_i(t + (-2) \times 4) = H_i(t-8)\ \forall\ i,t$, and if the code is also periodic with period $\hat{y} = 3$ then $H_i(t) = H_i(t-8) = H_i(t-8+3\times 3) = H_i(t+1)\ \forall\ i,t$. □

5.2.3 Unwrapping a protograph with gcd$(n_c, n_v) = 1$

If $\gcd(n_c, n_v) = 1$, we cannot form a square block matrix larger than 1×1 with equal size blocks. In this case, $P_l = P$ and P_u is the all-zeros matrix of size $n_c \times n_v$. This trivial cut results in a convolutional code with syndrome former memory zero, with repeating blocks of the original protograph on the leading diagonal. We now propose two methods of dealing with this structure.

5.2.3.1 Form an M-cover

Here we create a larger protograph parity-check matrix by using the copy and permute operation M times on P for some small integer M. This results in an $Mn_c \times Mn_v = n'_c \times n'_v$ parity-check matrix where of course the $\gcd(n'_c, n'_v) = M$. The $n'_c \times n'_v$ protograph parity-check matrix can then be cut following the procedure outlined above to form an infinite protograph parity-check matrix P_{cc}, where the step size is $Mn_c/y \times Mn_v/y$. The resulting unterminated, periodically time-varying convolutional code has decoding constraint length $\nu_s = Mn_v$ and, in general, period $T = y$ where each $H_i(t)$ has size $Mn_c/y \times Mn_v/y$. In effect, as there are $M!$ different choices of permutation matrix, this operation creates a mini ensemble of block codes that can be unwrapped to an ensemble of convolutional codes.

5.2.3.2 Use a nonuniform cut

When $\gcd(n_c, n_v) = 1$, we can also form a convolutional code by unwrapping the protograph parity-check matrix using a nonuniform cut. Let the protograph parity-check matrix be written as

$$P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,n_v} \\ \vdots & & \vdots \\ p_{n_c,1} & \cdots & p_{n_c,n_v} \end{bmatrix}.$$

We define a vector $\boldsymbol{\xi}$ consisting of n_c step parameters $\boldsymbol{\xi} = [\xi_1 \ \xi_2 \ \cdots \ \xi_{n_c}]$, where $0 \leq \xi_1 < \xi_{n_c} \leq n_v$, and each $\xi_{i-1} < \xi_i$ for $i = 2, \ldots, n_c$. As in the previous case, we form $n_c \times n_v$ matrices P_l and P_u as follows

- for each ξ_i, $i = 1, \ldots, n_c$, the entries $p_{i,1}$ to p_{i,ξ_i} are copied into the equivalent positions in P_l;
- entries p_{i,ξ_i+1} to p_{i,n_v} are copied, if they exist, into the equivalent positions in P_u;
- the remaining positions in P_l and P_u are set to zero.

An LDPC convolutional code derived from an LDPC block code using a nonuniform cut can be encoded and decoded using conventional encoding and decoding methods with minor modifications. For an LDPC convolutional code obtained using the nonuniform cut $\boldsymbol{\xi} =$

$[\xi_1 \ \xi_2 \ \cdots \ \xi_{n_c}]$, the maximum step width ξ_{max} for the cut is given by

$$\xi_{max} = \max_{i=2,\ldots,n_c} \{\xi_1, \xi_i - \xi_{i-1}\}.$$

Then $\xi_{max} - \xi_i$ columns of zeros are appended immediately to the left of the columns in the original protograph parity-check matrix P corresponding to the steps ξ_i, $i = 1, 2, \ldots, n_c$, to form a modified protograph parity-check matrix P'.

This process is illustrated for a (3,4)-regular protograph with the nonuniform cut $\boldsymbol{\xi} = [2 \ 3 \ 4]$ below:

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \Rightarrow P' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

LDPC convolutional codes unwrapped from P' can be encoded by a conventional LDPC convolutional encoder with the condition that information symbols are not assigned to the all-zeros columns. Thus, these columns correspond to punctured symbols, and the code rate is not affected. At the decoder, a conventional pipeline decoder (see, e.g., [JZ99a]) can be employed to decode the received sequence. No special treatment is necessary for the symbols corresponding to the all-zeros columns, since the column weight of zero ensures that they are not included in any parity-check equations. I.e., the belief-propagation decoding algorithm ignores the corresponding symbols.

We now form the parity-check matrix P_{cc} of an unterminated, periodically time-varying convolutional code from P' as in (5.1). Note that nonuniform cuts do not change the row and column weights of the original parity-check matrix P, and that the decoding constraint length remains constant at $\nu_s = n_v$.

5.2.4 Ensembles of protograph-based LDPC convolutional codes

Ensembles of block codes based on protographs were introduced in Section 4.2. The parity-check matrix of a protograph-based block code H is formed by replacing, for a fixed $N \in \mathbb{Z}^+$, the non-zero entries of P with $N \times N$ permutation matrices (or a summation of $N \times N$ permutation matrices for integer entries larger than 1), where the permutation matrices are

chosen randomly and independently, and replacing the zero entries of P with $N \times N$ all-zeros matrices. This operation maintains the edge connections of the base protograph, and if N is sufficiently large, then H is the parity-check matrix of a protograph-based LDPC block code. So far in this section, we have discussed methods of forming an infinite convolutional protograph with protograph parity-check matrix P_{cc} by unwrapping a base protograph P.

We may now form a family of unterminated, protograph-based, time-varying (LDPC) convolutional codes with decoding constraint length $\nu_s = Nn_v$ in the usual way, using the infinite protograph parity-check matrix P_{cc}. We note that, in general, a member of this convolutional family will have infinite period, as the permutation matrices are chosen randomly and independently. However, if P_{cc} has period $T = y$ then there will exist members of the convolutional family with period $T = ky$ for $k \in \mathbb{Z}^+$, where the submatrices $H_i(t)$ are of size $Nn_c/y \times Nn_v/y$.

5.3 Free distance bounds

In this section we present a method for obtaining a lower bound for the free distance of an ensemble of unterminated, asymptotically good, time-varying LDPC convolutional codes derived from protograph-based LDPC block codes. To proceed, we will make use of a family of tail-biting LDPC convolutional codes with incremental increases in block length. The tail-biting codes will be used as a tool to obtain the desired bound for the free distance of the unterminated codes.

5.3.1 Tail-biting convolutional codes

Consider the parity-check matrix P_{cc} of the protograph-based, unterminated convolutional code introduced in Section 5.2. We now introduce the notion of tail-biting convolutional codes by defining the 'unwrapping factor' λ as the number of times the sliding convolutional structure is repeated before applying tail-biting termination. For $\lambda \geq 1$, the parity-check matrix $P_{tb}^{(\lambda)}$ of the desired tail-biting protograph-based convolutional code with block length λn_v can be written as

$$P_{tb}^{(\lambda)} = \begin{bmatrix} P_l & & & & P_u \\ P_u & P_l & & & \\ & P_u & P_l & & \\ & & \ddots & \ddots & \\ & & & P_u & P_l \end{bmatrix}_{\lambda n_c \times \lambda n_v} . \tag{5.3}$$

Note that the tail-biting convolutional code for $\lambda = 1$ is simply the original protograph-based block code.

5.3.2 A tail-biting LDPC convolutional code ensemble

Given a protograph parity-check matrix P, we generate a family of tail-biting convolutional codes with parity check matrices $P_{tb}^{(\lambda)}$ and increasing block lengths λn_v, $\lambda = 1, 2, \ldots$, using the process described above. Note that this can alternatively be viewed as terminating the convolutional code with protograph parity-check matrix P_{cc} as a tail-biting convolutional code with block length λn_v for $\lambda = 1, 2, \ldots$. Since tail-biting convolutional codes are themselves block codes, we can treat the Tanner graph of $P_{tb}^{(\lambda)}$ as a protograph for each value of λ.

Replacing the entries of this matrix with either $N \times N$ permutation matrices or $N \times N$ all-zeros matrices, as discussed in Section 4.2, creates an ensemble of LDPC block codes which can be analysed asymptotically as N goes to infinity, where the sparsity condition of an LDPC code is satisfied for large N. Note that as the unwrapping factor λ increases, the tail-biting ensemble with protograph parity-check matrix $P_{tb}^{(\lambda)}$ becomes a better representation of the unterminated convolutional code family with protograph parity-check matrix P_{cc}. This is reflected in the weight enumerators, and it is shown in Section 5.4 that increasing λ provides us with minimum distance growth rates that converge to a lower bound for the free distance growth rate of the unterminated convolutional code.

In addition, we note that unwrapping any $P_{tb}^{(\lambda)}$ indefinitely results in the protograph parity-check matrix P_{cc}. Further, after inserting $N \times N$ permutation matrices and all-zeros matrices, each tail-biting LDPC code ensemble in turn can be unwrapped and repeated indefinitely to form an ensemble of unterminated, periodically time-varying LDPC convolutional codes. This ensemble will have rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$, decoding constraint length $\nu_s = Nn_v$ and, in general, period $T = \lambda y$.

5.3.3 A free distance bound

Tail-biting convolutional codes can be used to establish a lower bound for the free distance of the associated unterminated, periodically time-varying convolutional code by showing that the free distance of the unterminated code is bounded below by the minimum distance of any of its tail-biting versions. We can then extend this result to show a similar property for the average free distance of an ensemble of convolutional codes with respect to the average minimum distance of any of its associated ensembles of tail-biting convolutional codes. Finally, we will use this principle to find a lower bound for the average free distance for a family of unterminated, time-varying, LDPC convolutional codes based on an infinite protograph.

The method used to prove the following bound is based on a similar proof of a minimum distance bound first presented in [SPVC09]. The reader may find it helpful to consult Example 5.5 below when considering Theorem 5.2 and its proof.

Theorem 5.2 *Consider a rate $R = 1 - n_c/n_v$ unterminated, periodically time-varying convolutional code with decoding constraint length $\nu_s = n_v$ and period $T = y$ specified by (5.1) (see Section 5.2.1). Let $d_{min}^{(\lambda)}$ be the minimum distance of the associated tail-biting convolutional code with block length $n = \lambda n_v$ and unwrapping factor $\lambda > 0$. Then the free distance d_{free} of the unterminated convolutional code is lower bounded by $d_{min}^{(\lambda)}$ for any unwrapping factor λ. I.e.,*

$$d_{free} \geq d_{min}^{(\lambda)}, \quad \forall \lambda > 0.$$

Proof. Consider the unterminated convolutional code C whose parity-check matrix is given in the form (5.1). We can consider any codeword $\mathbf{x} \in C$ with Hamming weight l as a set of distinct variable nodes in the Tanner graph of the parity-check matrix, i.e.,

$$\mathbf{x} \equiv X = \{v_{I(0)}, v_{I(1)}, \cdots, v_{I(l-1)}\}.$$

The equivalence arises as follows: if we apply weight one to each of the variable nodes in X and weight zero to the remaining variable nodes of (5.1), then all of the check nodes will be satisfied (i.e. have even weight).

Here, $\mathbf{I} = [\, I(0) \;\; I(1) \;\; \cdots \;\; I(l-1)\,]$ is an index vector of length l and the elements $v_{I(j)}$ are unique by definition. Consider the modular index vector $\hat{\mathbf{I}}$ obtained by letting $\hat{I}(i) = I(i) \bmod \lambda n_v$, for $i = 0, 1, \ldots, l-1$.

Proposition. The set of variable nodes

$$\hat{X} = \left\{ v_0 k_0, v_1 k_1, \ldots, v_{\lambda n_v - 1} k_{\lambda n_v - 1} \;\middle|\; k_i = \sum_{j=0}^{l-1} \kappa_{\hat{I}(j),i} \mod 2 \right\}$$

corresponds to a codeword in the tail-biting convolutional code with parity-check matrix given in (5.3), where $\kappa_{a,b}$ is the Kronecker Delta function[2]: $\kappa_{a,b} = 1$ iff $a = b$ and zero otherwise.

Proof. The modular operation defined above corresponds to wrapping the unterminated convolutional code as a tail-biting convolutional code with unwrapping factor λ, with parity-check matrix given in (5.3). Consider the infinite concatenated parity-check matrix

$$P_\infty^{(\lambda)} = \left[P_{tb}^{(\lambda)} \;\middle|\; P_{tb}^{(\lambda)} \;\middle|\; \ldots \right] = \left[\begin{array}{ccccc|ccccc|c} P_l & & & & P_u & P_l & & & & P_u & \\ P_u & P_l & & & & P_u & P_l & & & & \\ & P_u & P_l & & & & P_u & P_l & & & \ldots \\ & & \ddots & \ddots & & & & \ddots & \ddots & & \\ & & & P_u & P_l & & & & P_u & P_l & \end{array}\right].$$

It is clear that any codeword $\mathbf{x} \in C$ satisfies $\mathbf{x} \cdot \left(P_\infty^{(\lambda)}\right)^T = \mathbf{0}$, as the same symbols checked in (5.1) are also checked by $P_\infty^{(\lambda)}$ in addition to some extra symbols. Suppose now that we partition $\mathbf{x}$ into sections of length λn_v, i.e,

$$\mathbf{x} = \left[\begin{array}{cccc|cccc|c} x_0 & x_1 & \ldots & x_{\lambda n_v - 1} & x_{\lambda n_v} & x_{\lambda n_v + 1} & \ldots & x_{2\lambda n_v - 1} & \ldots \end{array} \right] = \left[\mathbf{x}_0 \;\middle|\; \mathbf{x}_1 \;\middle|\; \ldots \right],$$

where each $\mathbf{x}_i$ has length λn_v. Consider the vector $\hat{\mathbf{x}} = \sum_{k=0}^{\infty} \mathbf{x}_i \bmod 2$ of length λn_v. Because of the repeated structure of $P_\infty^{(\lambda)}$, we observe that $\hat{\mathbf{x}} \cdot (P_{tb}^{(\lambda)})^T = \mathbf{0}$ and $\hat{\mathbf{x}} \equiv \hat{X}$. □

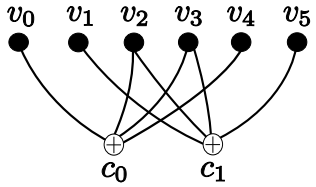
Suppose the codeword $\hat{\mathbf{x}}$ has Hamming weight $\hat{l}$. By the construction of $\hat{\mathbf{x}}$, clearly $\hat{l} \leq l$, as if any of the elements of the modular index vector $\hat{\mathbf{I}}$ are repeated then we obtain a reduction in $\hat{l}$. If the elements of $\hat{\mathbf{I}}$ are unique, then we say that the original codeword $\mathbf{x}$ (with Hamming weight l) has not been affected by the wrapping procedure and therefore $\hat{l}$ equals l. If, however, there are $k > 1$ copies of the same index value in $\hat{\mathbf{I}}$, then the value of $\hat{l}$ is reduced by $2\lfloor \frac{k}{2} \rfloor$. This corresponds to the equivalent case when multiple variable nodes in the codeword are wrapped back on top of each other. If there is an even number of such nodes at the same position, none

[2]Here, we avoid the standard notation 'δ' for the Kronecker Delta function as in this work δ is used to denote distance growth.

of them survives as a member of the tail-biting codeword $\hat{\mathbf{x}}$, and if the repetition value k is odd, only one node survives. These reductions are cumulative over all of the repetitive index values in $\hat{\mathbf{I}}$.

This implies that every codeword in the unterminated convolutional code induces a codeword in any of its tail-biting terminated codes that has Hamming distance at most as large as the original one. The main result now follows by wrapping back the unterminated codeword $\hat{\mathbf{x}}$ with minimum Hamming weight. □

Example 5.5. Consider the following protograph and associated protograph parity-check matrix



$$P = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 5.1: *An example protograph and associated protograph parity-check matrix.*

We form the parity-check matrix of the unterminated periodically time-varying convolutional code with rate $R = 1 - n_c/n_v = 2/3$, decoding constraint length $\nu_s = n_v = 6$ and period $T = y = 2$ by unwrapping P with step (submatrix) size $n_c/y \times n_v/y = 1 \times 3$ as described in Section 5.2.1. The convolutional code parity-check matrix is given as follows:

$$P_{cc} = \begin{bmatrix} 1 & 0 & 1 & & & & & & & & & & & & & \\ 0 & 1 & 1 & 1 & 0 & 1 & & & & & & & & & & \\ & & & 1 & 1 & 0 & 1 & 0 & 1 & & & & & & & \\ & & & & & & 0 & 1 & 1 & 1 & 0 & 1 & & & & \\ & & & & & & & & & 1 & 1 & 0 & 1 & 0 & 1 & & \\ & & & & & & & & & & & & 0 & 1 & 1 & 1 & 0 & 1 \\ & & & & & & & & & & & & & & \ddots & & & \ddots \end{bmatrix}, \tag{5.4}$$

where blank space corresponds to zeros. Consider the following infinite convolutional codeword sequence with Hamming weight $l = 9$:

$$\mathbf{x} = [1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ \cdots\ ].$$

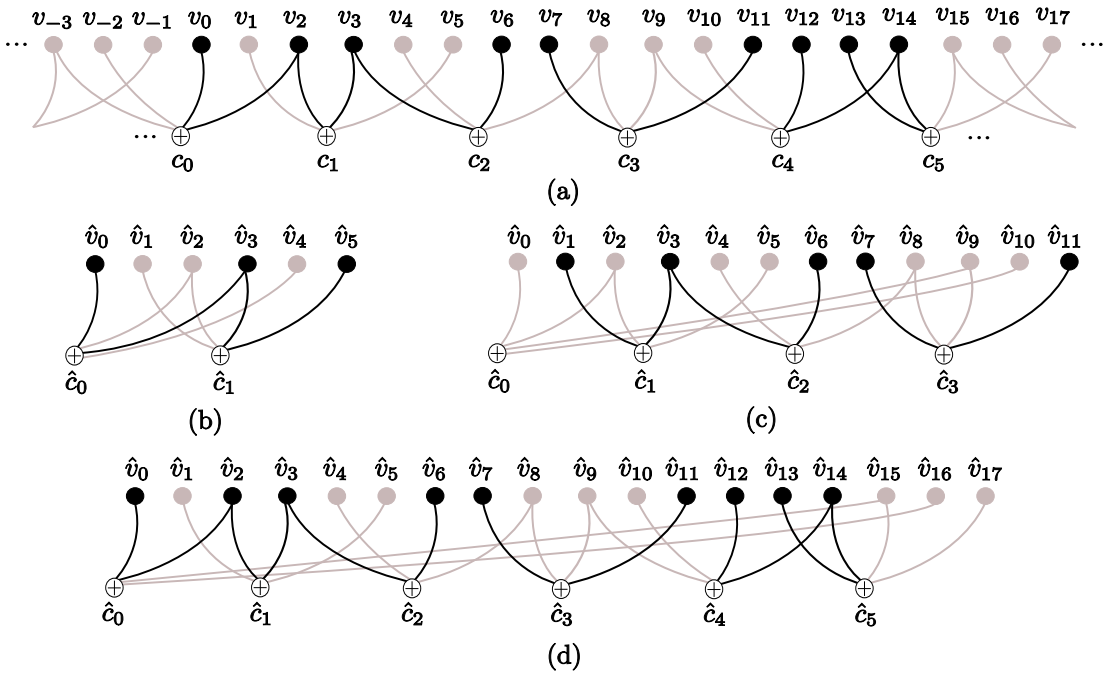


Figure 5.2: *A codeword sequence (a) in the infinite Tanner graph of the unterminated convolutional code of Example* 5.5 *and associated wrapped codewords of the tail-biting convolutional codes with unwrapping factors (b)* $\lambda = 1$*, (c)* $\lambda = 2$*, and (d)* $\lambda = 3$*.*

Suppose, without loss of generality, that the equivalent set of variable nodes is

$$X = \{v_0, v_2, v_3, v_6, v_7, v_{11}, v_{12}, v_{13}, v_{14}\},$$

with associated index vector $\mathbf{I} = [0\ \ 2\ \ 3\ \ 6\ \ 7\ \ 11\ \ 12\ \ 13\ \ 14]$. The set X is highlighted in Fig. 5.2(a). We now consider wrapping the convolutional code onto itself with different unwrapping factors λ.

- $\boldsymbol{\lambda = 1}$. Here, we wrap the convolutional codeword $\mathbf{x}$ onto itself with modulus $\lambda n_v = 6$. The resulting modular index vector is $\hat{\mathbf{I}} = [0\ \ 2\ \ 3\ \ 0\ \ 1\ \ 5\ \ 0\ \ 1\ \ 2]$. Observe that index values $\hat{I}(0)$, $\hat{I}(3)$ and $\hat{I}(6)$ equal 0. I.e., $k = 3$ for this index value, and we obtain the reduction value of $2\lfloor \frac{3}{2} \rfloor = 2$. Since k is odd, only one occurence of the resulting variable node appears in the reduced codeword. As regards the other index values, $\hat{I}(4) = \hat{I}(7) = 1$ and $\hat{I}(1) = \hat{I}(8) = 2$. We observe that $k = 2$ in both these cases, which results in a reduction value of 2 for both index values. Thus the variable nodes corresponding to these index values do not appear in the reduced codeword. Finally, index values $\hat{I}(2) = 3$ and $\hat{I}(5) = 5$ appear only once, so both of the corresponding variable nodes are present

in the reduced codeword. Then $\hat{\mathbf{I}}$ represents the set of variable nodes $\hat{X} = \{\hat{v}_0, \hat{v}_3, \hat{v}_5\}$ in the Tanner graph of the tail-biting convolutional code with unwrapping factor $\lambda = 1$. We observe that $\hat{X} \equiv \hat{\mathbf{x}} = [1\ 0\ 0\ 1\ 0\ 1]$ is a codeword with Hamming weight $\hat{l} = 3$ in the tail-biting convolutional code with parity-check matrix $P_{tb}^{(1)}$. This is highlighted in Fig. 5.2(b).

- $\boldsymbol{\lambda = 2}$. With modulus $\lambda n_v = 12$, we calculate the resulting modular index vector as $\hat{\mathbf{I}} = [0\ 2\ 3\ 6\ 7\ 11\ 0\ 1\ 2]$. We obtain the reduction value of 2 for the following index values: $\hat{I}(0) = \hat{I}(6) = 0$ and $\hat{I}(1) = \hat{I}(8) = 2$ respectively. The resulting codeword with Hamming weight $\hat{l} = 5$ is represented by $\hat{X} = \{\hat{v}_1, \hat{v}_3, \hat{v}_6, \hat{v}_7, \hat{v}_{11}\}$. This is a codeword of the tail-biting convolutional code with parity-check matrix $P_{tb}^{(2)}$ shown in Fig. 5.2(c).

- $\boldsymbol{\lambda \geq 3}$. With modulus λn_v for $\lambda \geq 3$, the unwrapping factor is large enough that the modular index vector is $\hat{\mathbf{I}} = \mathbf{I} = [0\ 2\ 3\ 6\ 7\ 11\ 12\ 13\ 14]$. Here we obtain no reduction for any of the index values and the resulting codeword with Hamming weight $\hat{l} = l = 9$ is represented by $\hat{X} = X = \{\hat{v}_0, \hat{v}_2, \hat{v}_3, \hat{v}_6, \hat{v}_7, \hat{v}_{11}, \hat{v}_{12}, \hat{v}_{13}, \hat{v}_{14}\}$. This is a codeword of the tail-biting convolutional code with parity-check matrix $P_{tb}^{(\lambda)}$ for $\lambda \geq 3$. The codeword $\hat{X} = X$ is illustrated in Fig. 5.2(d) for $\lambda = 3$. □

Example 5.6. Consider the rate $R = 1/3$ convolutional code with $(3, 1, 2)$ binary nonsystematic feedforward convolutional encoder given in Fig. 5.3.

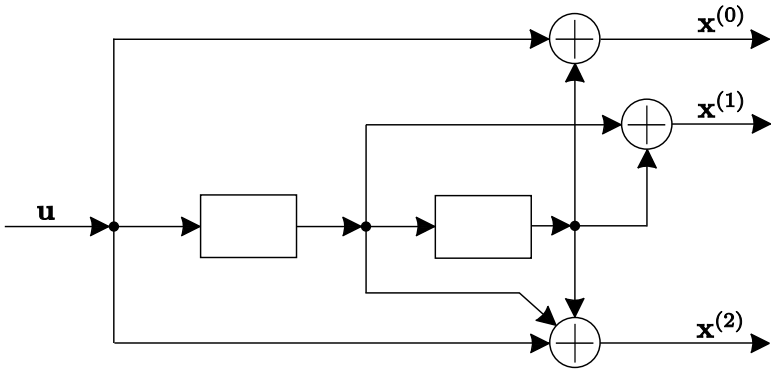


Figure 5.3: *A rate $R = 1/3$ binary nonsystematic feedforward convolutional encoder for the convolutional code of Example* 5.6.

The associated generator matrix is

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & & & & \\ & & & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & \\ & & & & & \ddots & & & \ddots & & & \ddots & \end{bmatrix},$$

and it can be shown easily that this convolutional code has free distance $d_{free} = 7$. The codeword sequence with minimum weight is $\mathbf{x} = [1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ \cdots]$ generated by information sequence $\mathbf{u} = [1\ 0\ 0\ \cdots]$. The corresponding infinite parity check matrix can be written as

$$P_{cc} = \begin{bmatrix} 0 & 0 & 0 & & & & & & & & & & \\ 1 & 1 & 1 & & & & & & & & & & \\ 1 & 1 & 1 & 0 & 0 & 0 & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 1 & & & & & & & \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & & & & \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & & & & \\ & & & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \\ & & & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & \\ & & & & & \ddots & & & \ddots & & & \ddots & \end{bmatrix}.$$

The parity-check matrix P_{cc} can be viewed as an unwrapped protograph parity-check matrix P, where

$$P = P_{tb}^{(1)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix},$$

and we unwrap P with step size $n_c/y \times n_v/y = 2 \times 3$.

The convolutional codeword $\mathbf{x}$ with minimum Hamming weight $l = 7$ can be written as the set of variable nodes

$$\mathbf{x} \equiv X = \{v_0, v_2, v_4, v_5, v_6, v_7, v_8\}.$$

In this example, we observe that with modulus $\lambda n_v = 9\lambda$ the modular index vector $\hat{\mathbf{I}}$ equals $\mathbf{I}$ for all $\lambda \geq 1$. Thus the corresponding wrapped codeword $\hat{\mathbf{x}}$ in any of the associated tail-biting convolutional codes also has Hamming weight $\hat{l} = l = 7$.

However, consider the convolutional codeword

$$X = \{v_0, v_2, v_3, v_4, v_8, v_9, v_{12}, v_{13}, v_{14}\},$$

generated by information sequence $\mathbf{u} = [\ 1 \quad 1 \quad 1 \quad 0 \quad \ldots\ ]$. For $\lambda = 1$, with modulus $\lambda n_v = 9$, the associated modular index vector is $\hat{\mathbf{I}} = [0\ 2\ 3\ 4\ 8\ 0\ 3\ 4\ 5]$. This vector corresponds to the minimum Hamming weight codeword of the tail-biting convolutional code $\hat{X} = \{v_2, v_5, v_8\}$. □

After performing the copy-and-permute operation to a protograph, the following corollary for protograph-based convolutional codes arises. Here, the low-density criteria of an LDPC code is satisfied if we choose a sufficiently large positive integer N.

Corollary 5.3 *Fix a positive integer N and let a protograph-based (LDPC) block code with parity-check matrix H be formed by replacing each one in the protograph parity-check matrix P with an $N \times N$ permutation matrix chosen randomly and independently and each zero with an $N \times N$ all zero matrix as described in Section 5.2. Consider forming a rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ unterminated, periodically time-varying (LDPC) convolutional code with decoding constraint length $\nu_s = Nn_v$ and period $T = y$ by unwrapping H with step size $Nn_v/y \times Nn_c/y$. Let $d_{min}^{(\lambda)}$ be the minimum distance of the associated (LDPC) tail-biting convolutional code with block length $n = \lambda N n_v$ and unwrapping factor $\lambda > 0$. Then the free distance d_{free} of the unterminated (LDPC) convolutional code is lower bounded by $d_{min}^{(\lambda)}$ for any unwrapping factor λ.*

Proof. The result follows directly by replacing n_v by Nn_v in the proof of Theorem 5.2. □

Example 5.7. Consider the protograph parity-check matrix of Example 5.5. Let $N = 2$. Then the resulting $N! = 2$ choices of permutation matrices are

$$\left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right\}.$$

Picking matrices randomly and independently from this set, a typical protograph-based block

code parity-check matrix can be given as follows:

$$H = H_{tb}^{(1)} = \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]. \tag{5.5}$$

The resulting parity-check matrix of a convolutional code, after unwrapping with step size $Nn_c/y \times Nn_v/y = 2 \times 6$, is

$$H_{cc} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & & & & & & & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 1 & & & & & & & & & & & & \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & & & & & & \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & & & & & & \\ & & & & & & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ & & & & & & & & & \ddots & & & & & & \ddots & & \end{bmatrix}. \tag{5.6}$$

This convolutional code has rate $R = 1 - Nn_c/Nn_v = 2/3$, decoding constraint length $\nu_s = Nn_v = 12$ and period $T = y = 2$, where the submatrix size is 2×6.

Consider the convolutional codeword

$$X = \{v_0, v_3, v_4, v_6, v_7, v_{12}, v_{13}, v_{14}, v_{15}, v_{22}, v_{23}\},$$

with corresponding index vector $\mathbf{I} = [0\ \ 3\ \ 4\ \ 6\ \ 7\ \ 12\ \ 13\ \ 14\ \ 15\ \ 22\ \ 23]$ and Hamming weight $l = 11$. For unwrapping factor $\lambda = 1$, we calculate the resulting modular index vector with modulus $\lambda Nn_v = 12$ as $\hat{\mathbf{I}} = [0\ \ 3\ \ 4\ \ 6\ \ 7\ \ 0\ \ 1\ \ 2\ \ 3\ \ 10\ \ 11]$. We observe that the resulting codeword in the tail-biting convolutional code with block length $n = \lambda Nn_v = 12$ is

$$X = \{v_1, v_2, v_4, v_6, v_7, v_{10}, v_{11}\} \equiv [0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1] = \hat{\mathbf{x}},$$

which has Hamming weight $\hat{l} = 7$.

For $\lambda \geq 2$, we calculate $\hat{\mathbf{I}} = \mathbf{I} = [0\ 3\ 4\ 6\ 7\ 12\ 13\ 14\ 15\ 22\ 23]$, and thus the corresponding codeword in the tail-biting convolutional code with block length $n = \lambda N n_v = 12\lambda$ has Hamming weight $\hat{l} = l = 11$. □

Corollary 5.4 *Consider forming the parity-check matrix H described in Corollary 5.3. By letting the $N \times N$ permutation matrices vary over the $N!$ different choices we obtain an ensemble of protograph-based (LDPC) block codes. The associated ensemble of rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ unterminated, periodically time-varying (LDPC) convolutional code with decoding constraint length $\nu_s = Nn_v$ and period $T = y$ can be obtained by unwrapping each parity-check matrix H in the block code ensemble with step size $Nn_v/y \times Nn_c/y$. Denote the average minimum distance of the associated (LDPC) tail-biting convolutional code ensemble with block length $n = \lambda N n_v$ and unwrapping factor $\lambda > 0$ by $\overline{d}_{min}^{(\lambda)}$. Then the average free distance $\overline{d}_{free}$ of the unterminated (LDPC) convolutional code ensemble is bounded below by $\overline{d}_{min}^{(\lambda)}$ for any unwrapping factor λ.*

Proof. Consider unwrapping each member of the protograph-based block code ensemble indefinitely to form the associated ensemble of rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ unterminated, periodically time-varying (LDPC) convolutional code ensemble with decoding constraint length $\nu_s = Nn_v$ and period $T = y$. Now fix $\lambda > 0$, i.e., we fix a termination length for every member of the convolutional ensemble. It follows from Corollary 5.3 that each member of the (LDPC) convolutional code ensemble satisfies $d_{free} \geq d_{min}^{(\lambda)}$, and thus the the average free distance $\overline{d}_{free}$ of the unterminated (LDPC) convolutional code ensemble is bounded below by the average minimum distance $\overline{d}_{min}^{(\lambda)}$ of any ensemble of tail-biting convolutional codes for any fixed unwrapping factor λ and any permutation matrix size N. □

Note that, because there is no danger of ambiguity, from now on we will drop the overline notation when discussing ensemble average distances. Thus for an ensemble of convolutional codes $\overline{d}_{free} = d_{free}$, and for an ensemble of tail-biting convolutional codes with unwrapping factor λ, $\overline{d}_{min}^{(\lambda)} = d_{min}^{(\lambda)}$. Note that for an ensemble average the value of d_{free} (respectively d_{min}) need not be an integer.

Remark 5.5 *By considering only $\lambda = 1$ in Corollary 5.4, it is easily seen that the average free distance of a protograph-based unterminated (LDPC) convolutional code ensemble is bounded below by the average minimum distance of the protograph-based (LDPC) block code ensemble on which it is based.*

Theorem 5.2 and the subsequent Corollaries 5.3 and 5.4 allow us to prove our main result of this section. We wish to bound the average free distance of an unterminated, time-varying LDPC convolutional code family with rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ and decoding constraint length $\nu_s = Nn_v$ obtained by replacing ones with $N \times N$ permutation matrices and zeros with $N \times N$ all-zeros matrices in the convolutional protograph parity-check matrix P_{cc} defined in equation (5.1).

As λ increases, we observe that the tail-biting convolutional code with parity-check matrix $P_{tb}^{(\lambda)}$ defined in (5.3) becomes a better representation of the associated unterminated convolutional code, with $\lambda \to \infty$ corresponding to the unterminated convolutional code with parity-check matrix P_{cc}. In the following theorem, we show that the free distance of an unterminated, time-varying LDPC convolutional code family based on protograph parity-check matrix P_{cc} can be bounded below by the minimum distance of protograph-based block code ensemble with protograph parity-check matrix $P_{tb}^{(\lambda)}$.

Theorem 5.6 *Consider forming a family of unterminated, periodically time-varying (LDPC) convolutional codes with rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$, constraint length $\nu_s = Nn_v$, and period $T = \lambda y$ from (5.3), as described in Section 5.3.2. Let $d_{free}^{(\lambda)}$ be the free distance of the (LDPC) convolutional code ensemble with $T = \lambda y$, and let $d_{free} = \max_{\lambda > 0} d_{free}^{(\lambda)}$, which we call the free distance of the unterminated convolutional code family. Then, for any tail-biting termination of the unterminated convolutional code family with unwrapping factor λ, d_{free} is bounded below by $d_{min}^{(\lambda)}$ for any λ, where $d_{min}^{(\lambda)}$ is the average minimum distance of the protograph-based block code ensemble with parity-check matrix $P_{tb}^{(\lambda)}$. I.e.,*

$$d_{free} \geq d_{min}^{(\lambda)}, \quad \forall \lambda > 0. \tag{5.7}$$

Proof. Fix λ and consider the protograph parity-check matrix $P_{tb}^{(\lambda)}$ of the tail-biting convolutional code defined in (5.3). If we consider this matrix as the matrix P in Corollary 5.3 and as the base protograph parity-check matrix generating the ensemble defined in Corollary 5.4, then both results hold with the modification that the unwrapped convolutional code (resp. convolutional code ensemble) has, in general, period $T = \lambda y$, where the submatrix size is now $Nn_c/y \times Nn_v/y$. Thus $d_{free}^{(\lambda)} \geq d_{min}^{(\lambda)}$ for any $\lambda > 0$. The result then follows directly from the definition of d_{free}. □

Example 5.8. Consider the infinite protograph parity-check matrix P_{cc} of Example 5.5 defined in equation (5.4). The resulting ensemble of time-varying (LDPC) convolutional codes will have rate $R = 1 - n_c/n_v = 2/3$ and decoding constraint length $\nu_s = Nn_v = 6N$. Note that, in general, any member of the ensemble will have infinite period for $N > 1$. Suppose $N = 2$. Then a typical member of the unterminated convolutional code family could have a parity-check matrix H_{cc} as follows:

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & & & & & & & & & & & & & & & & & & \\
0 & 1 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & & & & & & \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & & & & & & & & & & & & \\
 & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & \\
 & & & & & & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & & & & & & \\
 & & & & & & & & & & & & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \\
 & & & & & & & & & & & & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & \ddots \\
 & & & & & & & & & & & & & & & & & & 0 & 1 & 1 & 0 & 0 & 0 & \\
 & & & & & & & & & & & & & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & \ddots \\
 & & & & & & & & & & & & & & & & & & & & & & & & \\
\end{bmatrix},$$

where the submatrices $H_i(t)$ are of size 2×6. Note that the difference between this matrix and that given in equation (5.6) is that the submatrices need not satisfy the equality $H_i(t) = H_i(t+2)$ for all i and t. If we terminate H_{cc} with 'unwrapping factor' $\lambda = 1$ we obtain the terminated parity-check matrix

$$H_{tb}^{(1)} = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}.$$

This matrix is a member of the ensemble with base protograph parity-check matrix $P_{tb}^{(1)}$ and $N = 2$, as we saw in equation (5.5). Thus $d_{free} \geq d_{free}^{(1)} \geq d_{min}^{(1)}$, where d_{free} is the free distance of the unterminated convolutional code with parity-check matrix H_{cc}, $d_{free}^{(1)}$ is the free distance of the unterminated convolutional code ensemble obtained by unwrapping $H_{tb}^{(1)}$ with step size $Nn_c/y \times Nn_v/y = 2 \times 6$ and $d_{min}^{(1)}$ is the minimum distance of the tail-biting convolutional ensemble with parity-check matrix $H_{tb}^{(1)}$. This holds for any choice of H_{cc}.

Terminating H_{cc} with 'unwrapping factor' $\lambda = 2$ results in the parity-check matrix $H_{tb}^{(2)}$ given as follows:

$$\left[\begin{array}{cccccc|cccccccccccc|cccccc}
1 & 0 & 0 & 0 & 1 & 0 & & & & & & & & & & & & & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & & & & & & & & & & & & & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & & & & & & & \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & & & & & & & & & & & & \\
\hline
 & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & & & & & & \\
 & & & & & & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & & & & & & \\
\hline
 & & & & & & & & & & & & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 & & & & & & & & & & & & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}\right].$$

This can also be viewed as a protograph-based parity-check matrix from the ensemble formed from $P_{tb}^{(2)}$ by taking $N = 2$. Thus, $d_{free} \geq d_{free}^{(2)} \geq d_{min}^{(2)}$ for any member of the unterminated convolutional code family H_{cc}. □

5.3.4 The free distance growth rate

The distance growth rate δ_{min} of a block code ensemble is defined as its asymptotic minimum distance to block length ratio. For the protograph-based tail-biting LDPC convolutional code ensembles with unwrapping factor λ defined in Section 5.3.2, this ratio is therefore given as

$$\delta_{min}^{(\lambda)} = \frac{d_{min}^{(\lambda)}}{n} = \frac{d_{min}^{(\lambda)}}{\lambda N n_v} = \frac{d_{min}^{(\lambda)}}{\lambda \nu_s}.$$

Using (5.7) we obtain

$$\delta_{min}^{(\lambda)} \leq \frac{d_{free}}{\lambda \nu_s}, \tag{5.8}$$

where d_{free} is the free distance of the associated unterminated, time-varying LDPC convolutional code family. It is important to note that for convolutional codes, the length of the shortest codeword is equal to the encoding constraint length ν_e, which in general differs from the decoding constraint length ν_s. Assuming minimal encoder and syndrome former matrices, the relation between ν_e and ν_s can be expressed as

$$\nu_e = \frac{1-R}{R}\nu_s, \tag{5.9}$$

which implies that, for code rates less than $1/2$, the encoding constraint length is larger than the decoding constraint length, and vice versa for code rates greater than $1/2$.

Combining (5.8) and (5.9) gives us the desired bound for the free distance growth rate

$$\delta_{free} \geq \frac{R}{1-R}\lambda\delta_{min}^{(\lambda)}, \tag{5.10}$$

where $\delta_{free} = d_{free}/\nu_e$ is the free distance growth rate of the unterminated LDPC convolutional code family [3].

One must be careful in comparing the distance growth rates of codes with different underlying structures. A fair basis for comparison generally requires equating the complexity of encoding and/or decoding of the two codes. Traditionally, the minimum distance growth rate of block codes is measured relative to block length, whereas constraint length is used to measure the free distance growth rate of convolutional codes. These measures are based on the complexity of decoding both types of codes on a trellis. Indeed, the typical number of states required to decode a block code on a trellis is exponential in the block length, and similarly the number of states required to decode a convolutional code is exponential in the constraint length. This has been an accepted basis of comparing block and convolutional codes for decades, since maximum-likelihood decoding can be implemented on a trellis for both types of codes.

The definition of decoding complexity is different, however, for LDPC codes. The sparsity of their parity-check matrices, along with the iterative message-passing decoding algorithm typically employed, implies that the decoding complexity per symbol depends on the degree distribution of the variable and check nodes and is independent of both the block length and the constraint length. The cutting technique described in Section 5.2 preserves the degree distribution of the underlying LDPC block code, and thus the decoding complexity per symbol is the same for the block and convolutional codes considered in this thesis.

Also, for randomly constructed LDPC block codes, state-of-the-art encoding algorithms require only $O(g)$ operations per symbol, where $g << n$ [RU01b], whereas for LDPC convolutional codes, if the parity-check matrix satisfies the conditions listed in Section 1.5.2, the number of encoding operations per symbol is only $O(1)$ [PJS$^+$09]. Here again, the encoding complexity per symbol is essentially independent of both the block length and the constraint length.

[3] If the syndrome former matrix is not in minimal form, (5.9) results in an upper bound for ν_e, which implies that δ_{free} is underestimated in this case.

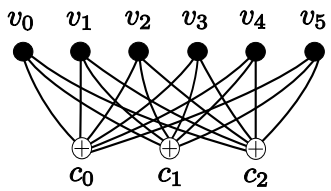
Hence, to compare the distance growth rates of LDPC block and convolutional codes, we consider the hardware complexity of implementing the encoding and decoding operations in hardware. Typical hardware storage requirements for both LDPC block encoders and decoders are proportional to the block length n. The corresponding hardware storage requirements for LDPC convolutional encoders and decoders are proportional to the decoding constraint length [PJS+09].[4]

5.4 Bound computations

We now present results on the free distance growth rate for ensembles of asymptotically good LDPC convolutional codes based on protographs. We first consider a wide variety of regular ensembles (including all the regular ensembles originally considered by Gallager [Gal62]), and for each we calculate a lower bound for the free distance to constraint length ratio δ_{free}. We begin by considering the regular ensembles with $\gcd(n_c, n_v) > 1$. Then the methods of Section 5.2.3 are considered for the regular ensembles with $\gcd(n_c, n_v) = 1$. Results for these ensembles are then presented and discussed. Finally, we look at two irregular ensembles of practical interest. For these ensembles, we calculate a lower bound for the free distance to constraint length ratio δ_{free} and compare it to the block code minimum distance growth rate δ_{min} previously calculated in the literature [DDJ06].

5.4.1 Regular ensembles with $\gcd(n_c, n_v) > 1$

Example 5.9. Consider the rate $R = 1/2$, $(3, 6)$-regular LDPC code ensemble based on the following protograph:



.

For this example, the minimum distance growth rate is $\delta_{min} = 0.023$, as originally computed by Gallager [Gal62]. A family of tail-biting $(3, 6)$-regular LDPC convolutional code ensembles can be generated according to the following cut:

[4] For rates other than $1/2$, encoding constraint lengths may be preferred to decoding constraint lengths.

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

For $\lambda = 2, 3, \ldots, 8$, the minimum distance growth rate δ_{min} was calculated for the tail-biting LDPC convolutional code ensembles using the approach outlined in Section 5.3.2 and the method of [Div06]. These growth rates are shown in Fig. 5.4, along with the corresponding lower bound for the free distance growth rate δ_{free} of the associated ensemble of unterminated, periodically time-varying LDPC convolutional codes. For this rate $R = 1/2$ ensemble, the lower bound for δ_{free} is simply $\delta_{free} \geq \frac{R}{1-R}\lambda\delta_{min} = \lambda\delta_{min}$, since $\frac{R}{1-R} = 1$ in this case.

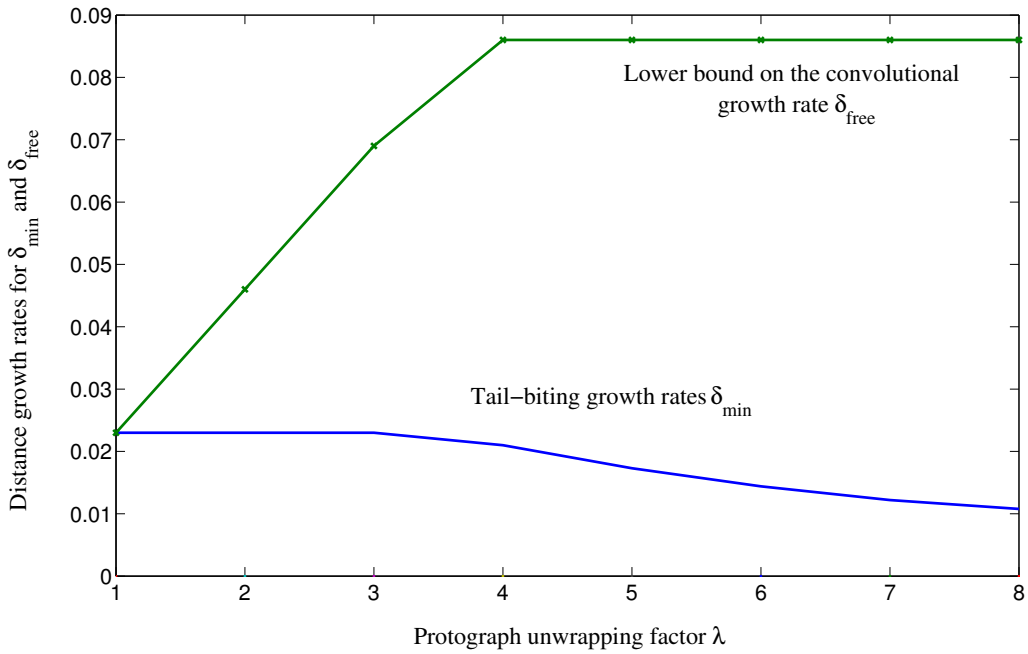


Figure 5.4: *Distance growth rates for Example* 5.9.

We observe that, once the unwrapping factor λ of the tail-biting convolutional codes exceeds 3, the lower bound for δ_{free} levels off at $\delta_{free} \geq 0.086$, which agrees with the results presented in [STL+07] and [TZC09], and represents a significant increase over the value of δ_{min}. In this case, the minimum weight codeword in the unterminated convolutional code also appears as a codeword in the tail-biting code. □

Example 5.10. Consider a rate $R = 1/3$, $(4, 6)$-regular LDPC code ensemble. The minimum distance growth rate for this ensemble is $\delta_{min} = 0.128$ [Gal62]. We form a protograph in the usual fashion, creating four check nodes, each of which connect to all six variable nodes, and we observe that $\gcd(4, 6) = 2$. The protograph parity-check matrix and defined cut are displayed below:

$$P = \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right].$$

For this rate $R = 1/3$ ensemble, the lower bound for δ_{free} is $\delta_{free} \geq \frac{R}{1-R}\lambda\delta_{min} = \frac{1}{2}\lambda\delta_{min}$. We observe that, as in Example 5.9, the minimum distance growth rates calculated for increasing λ provide us with a lower bound $\delta_{free} \geq 0.197$ on the free distance growth rate of the convolutional code ensemble, which exceeds the value of δ_{min}. □

Example 5.11. Consider a rate $R = 1/2$, $(4, 8)$-regular LDPC code ensemble. The minimum distance growth rate for this ensemble is $\delta_{min} = 0.063$ [Gal62]. The protograph parity-check matrix is cut along the diagonal in steps of 1×2. For this rate $R = 1/2$ ensemble, the lower bound for δ_{free} is $\delta_{free} \geq \frac{R}{1-R}\lambda\delta_{min} = \lambda\delta_{min}$, and we obtain the lower bound $\delta_{free} \geq 0.191$ on the free distance growth rate of the convolutional code ensemble, which is again significantly larger than δ_{min} and agrees with the results in [STL$^+$07] and [TZC09]. □

We have now calculated a lower bound for the free distance growth rate for all the regular ensembles with $\gcd(n_c, n_v) > 1$ that Gallager considered in his celebrated work [Gal62]. For the following regular ensembles with $\gcd(n_c, n_v) > 1$, the trivial all-ones protograph parity-check matrix of size $n_c \times n_v$ was formed as in the previous examples. The minimum distance growth rate δ_{min} of each LDPC block code ensemble was calculated using the usual spectral shape analysis and the resulting lower bounds $\delta_{free} \geq \frac{R}{1-R}\lambda\delta_{min}$ for the free distance growth rate are presented in Table 5.1.

For each ensemble considered, the lower bound for δ_{free} is larger than δ_{min} for the block code ensemble. This is illustrated in Fig. 5.5, where the distance growth rates of each regular LDPC code ensemble are compared to the corresponding lower bound for the optimal growth rate for general block [Gil52, Var57] and convolutional [Cos74] codes.

Ensemble	Rate R	δ_{min}	Lower bound for δ_{free}
$(3, 9)$	$2/3$	0.0054	0.0372
$(3, 12)$	$3/4$	0.0021	0.0202
$(4, 10)$	$3/5$	0.037	0.134
$(4, 12)$	$2/3$	0.024	0.132
$(5, 10)$	$1/2$	0.084	0.238

Table 5.1: *Lower bounds on the free distance growth rate for regular ensembles with* $gcd(n_c, n_v) = 1$.

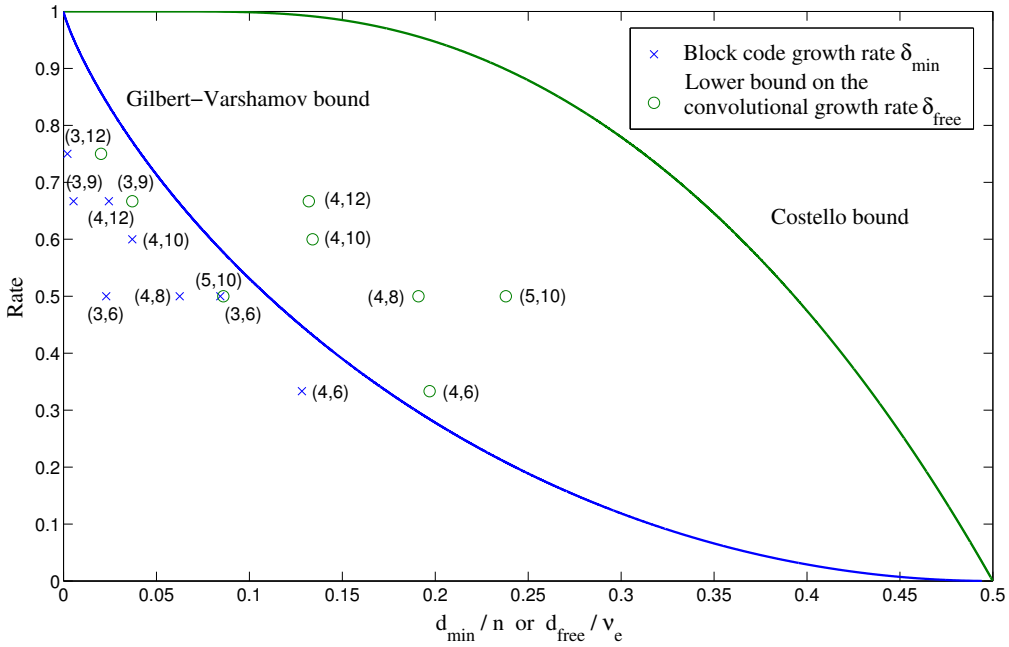


Figure 5.5: *Comparison of calculated growth rates for regular ensembles with* $gcd(n_c, n_v) > 1$ *with the Gilbert-Varshamov bound for block code minimum distance growth rates and the Costello bound for convolutional code free distance growth rates.*

5.4.2 Regular ensembles with gcd$(n_c, n_v) = 1$

We now present results for the two methods of unwrapping a protograph with gcd$(n_c, n_v) = 1$ introduced in Section 5.2.3.

Example 5.12. Consider a rate $R = 2/5$, $(3, 5)$-regular ensemble. The minimum distance growth rate for this ensemble is $\delta_{min} = 0.045$ [Gal62]. For this rate $R = 2/5$ ensemble, the lower bound for δ_{free} is $\delta_{free} \geq \frac{R}{1-R}\lambda\delta_{min} = \frac{2}{3}\lambda\delta_{min}$. The first approach was to form a two-cover of the regular protograph. The resulting mini-ensemble has $2^{n_v n_c} = 2^{15}$ members.

Fifty distinct members were chosen randomly. The resulting lower bounds calculated for δ_{free} are shown in a box plot in Fig. 5.6.

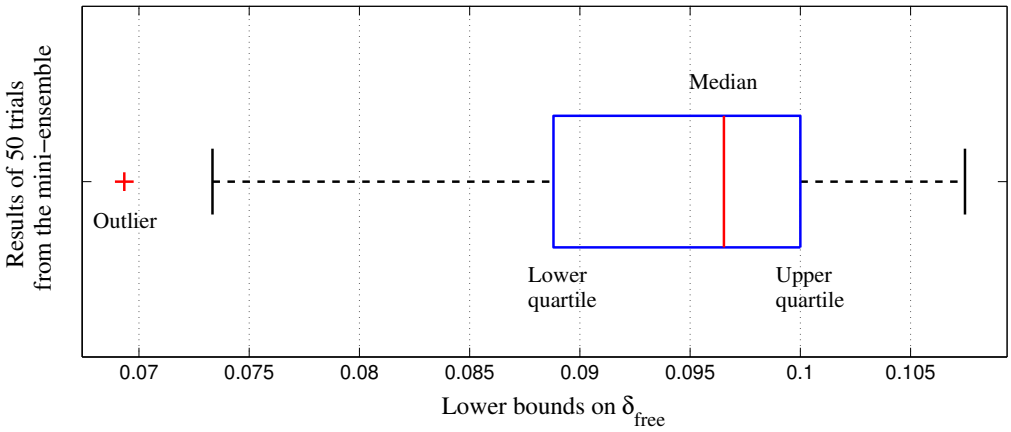


Figure 5.6: *Free distance growth rates for* 50 *mini-ensemble members.*

We observe a fairly large spread in results from the mini-ensemble. The median from the fifty trials is $\delta_{free} \geq 0.097$. We also observe that the smallest lower bound found is statistically an outlier, since it lies reasonably far away from the lower quartile. Note that this smallest lower bound ($\delta_{free} \geq 0.069$) is larger than the block code growth rate $\delta_{min} = 0.045$. Also, the best lower bound, $\delta_{free} \geq 0.108$, is significantly larger than δ_{min}.

We now consider the following two nonuniform cuts of the standard protograph parity-check matrix for the regular $(3, 5)$ ensemble:

$$P_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \text{ and } P_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

with corresponding cutting vectors $\boldsymbol{\xi}_1 = [2\ \ 4\ \ 5]$ and $\boldsymbol{\xi}_2 = [1\ \ 2\ \ 3]$. We calculate a lower bound of $\delta_{free} \geq 0.119$ for cut one and $\delta_{free} \geq 0.111$ for cut two. Note that both nonuniform cuts give larger lower bounds on δ_{free} than the mini-ensemble method. □

For the remaining regular ensembles with $\gcd(n_c, n_v) = 1$, we used the nonuniform cut method. The resulting bounds are given in Table 5.2.

Ensemble	Rate R	Cut $\boldsymbol{\xi}$	δ_{min}	Lower bound for δ_{free}
$(3, 4)$	$1/4$	$[2\ 3\ 4]$	0.112 [Gal62]	0.177
$(3, 7)$	$4/7$	$[2\ 4\ 7]$	0.0129	0.0616
$(3, 8)$	$1/4$	$[3\ 6\ 8]$	0.0081	0.0467
$(4, 5)$	$1/5$	$[2\ 3\ 4\ 5]$	0.210 [Gal62]	0.266
$(5, 6)$	$1/6$	$[2\ 3\ 4\ 5\ 6]$	0.254 [Gal62]	0.317
$(5, 8)$	$3/8$	$[2\ 4\ 6\ 7\ 8]$	0.136	0.249

Table 5.2: *Lower bounds on the free distance growth rate for regular ensembles with a nonuniform cut.*

For each ensemble considered, the lower bound for δ_{free} is larger than δ_{min} for the block code ensemble. We may now update Fig. 5.5 to include the calculated distance growth rates[5] for regular ensembles with $\gcd(n_c, n_v) = 1$. This is illustrated in Fig. 5.7.

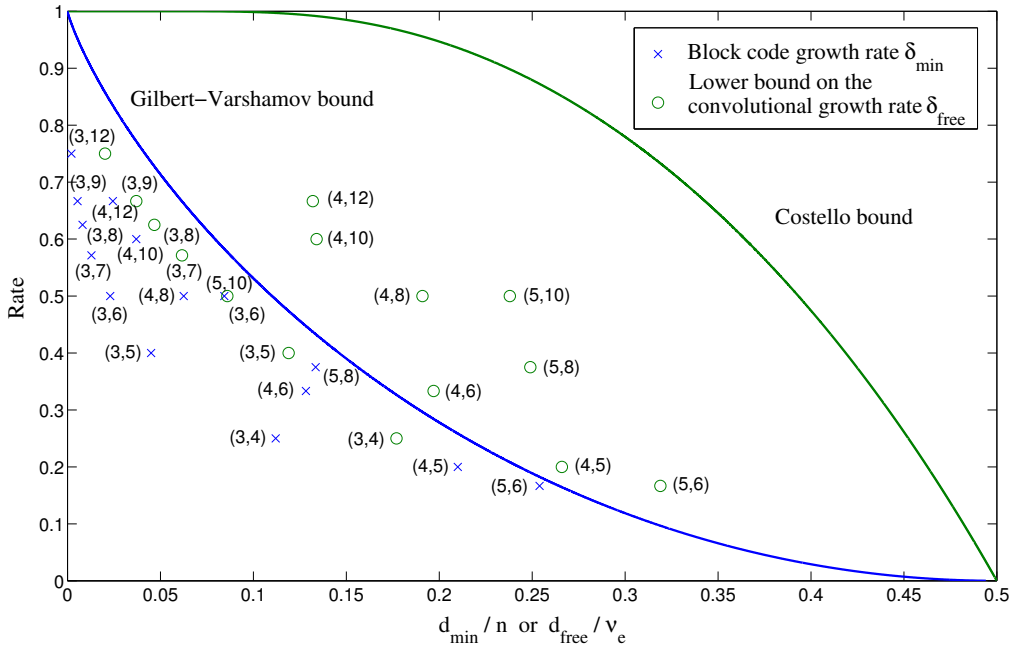
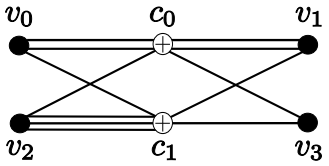


Figure 5.7: *Comparison of calculated growth rates with the Gilbert-Varshamov bound for block code minimum distance growth rates and the Costello bound for convolutional code free distance growth rates.*

[5]Note that it may be possible to improve the lower bound for δ_{free} for these ensembles by choosing a better nonuniform cut.

5.4.3 Irregular ensembles

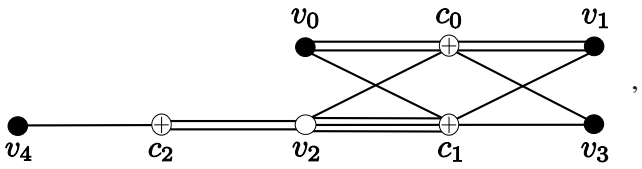
Example 5.13. The following irregular protograph is from the Repeat Jagged Accumulate (RJA) family [DDJ06]. It was shown to have a good iterative decoding threshold ($\gamma_{\text{iter}} = 1.0$ dB) while maintaining linear minimum distance growth ($\delta_{min} = 0.013$). We display below the associated P matrix and cut used to generate the family of tail-biting LDPC convolutional code ensembles.



$$\leftrightsquigarrow \qquad P = \left[\begin{array}{cc|cc} 2 & 2 & 1 & 1 \\ \hline 1 & 1 & 3 & 1 \end{array}\right].$$

We observe that, as in previous examples, the minimum distance growth rates calculated for increasing λ provide us with a lower bound for the free distance growth rate of the convolutional code ensemble using (5.10). The lower bound was calculated as $\delta_{free} \geq 0.057$ (for $\lambda \geq 5$), significantly larger than the minimum distance growth rate δ_{min} of the underlying block code ensemble. □

Example 5.14 The following irregular protograph is from the Accumulate Repeat Jagged Accumulate family (ARJA) [DDJ06]:



,

where the white circle represents a punctured variable node. This protograph is of significant practical interest, since it was shown to have $\delta_{min} = 0.015$ and iterative decoding threshold $\gamma_{\text{iter}} = 0.628$, i.e., pre-coding the protograph of Example 5.13 provides an improvement in both values.

In this ARJA example, the protograph matrix P is of size $n_c \times n_v = 3 \times 5$. We observe that $\gcd(n_c, n_v) = 1$, and thus we have the trivial cut mentioned in Section 5.3.1. Here we choose to copy and permute P to generate a mini ensemble of block codes. Results are shown for one particular member of the mini-ensemble with $M = 2$, but a change in performance can be obtained by varying the particular permutation chosen. Increasing λ for the chosen permutation results in a lower bound, found using (5.10), of $\delta_{free} \geq 0.053$ for $\lambda \geq 4$. Again, we observe a significant increase in δ_{free} compared to δ_{min}. □

5.4.4 Simulation results

Simulation results for LDPC block and convolutional codes based on the protograph of Example 5.14 were obtained assuming BPSK modulation and an additive white Gaussian noise (AWGN) channel. All decoders were allowed a maximum of 100 iterations and the block code decoders employed a syndrome-check based stopping rule. As a result of their block structure, tail-biting LDPC convolutional codes were decoded using standard LDPC block decoders employing a belief-propagation decoding algorithm. The LDPC convolutional code, on the other hand, was decoded by a sliding-window based belief-propagation decoder [JZ99a]. The resulting bit error rate (BER) performance is shown in Fig.5.8.

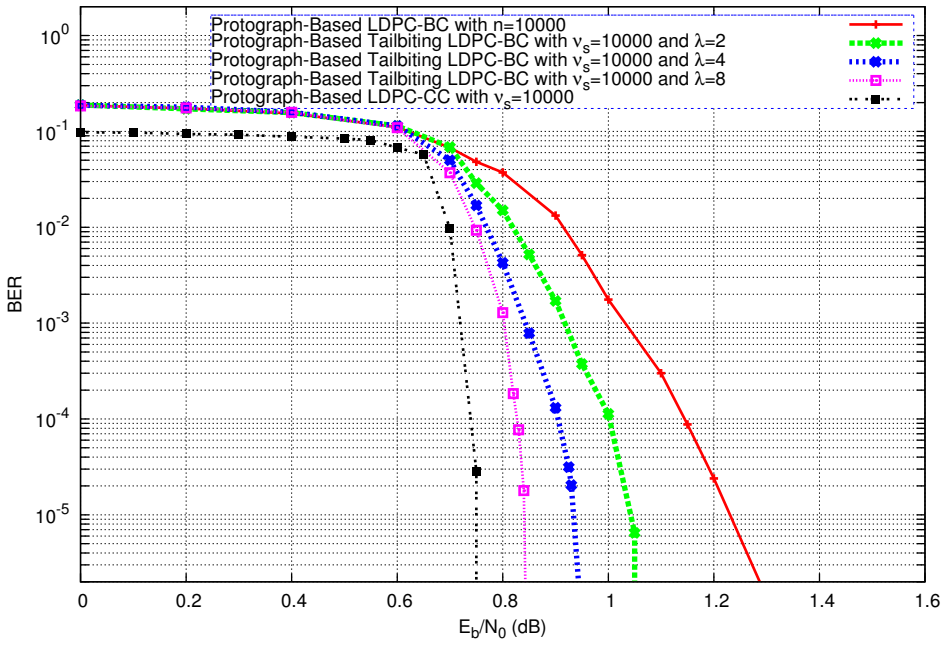


Figure 5.8: *Simulation results for Example* 5.14.

We note that the protograph-based tail-biting LDPC convolutional codes outperform the underlying protograph-based LDPC block code (which can also be seen as a tail-biting code with unwrapping factor $\lambda = 1$). Larger unwrapping factors yield improved error performance, eventually approaching the performance of the unterminated convolutional code, which can be seen as a tail-biting code with an infinitely large unwrapping factor. We also note that no error floor is observed for the convolutional code, which is expected, since the code ensemble is asymptotically good and has a relatively large ($\delta_{free} \geq 0.053$) distance growth rate.

We also note that the performance of the unterminated LDPC convolutional code is consistent with the iterative decoding threshold computed for the underlying protograph. At a moderate constraint length of 10000, the unterminated code achieves 10^{-5} BER at roughly 0.12 dB away from the threshold, and with larger block (constraint) lengths, the performance will improve even further. This is expected, since both the unterminated and the tail-biting convolutional codes preserve the same degree distribution as the underlying protograph.

5.5 Conclusions

In this chapter, asymptotic methods were used to calculate a lower bound for free distance that grows linearly with constraint length for several ensembles of regular and irregular, unterminated, protograph-based time-varying LDPC convolutional codes. It was shown that the free distance growth rates of the regular LDPC convolutional code ensembles exceed the minimum distance growth rates of the corresponding regular LDPC block code ensembles. When $\gcd(n_c, n_v) = 1$, two new methods were proposed to unwrap the protograph parity-check matrix in order to obtain the best possible lower bound for δ_{free}. The results suggest that we typically obtain better lower bounds by performing nonuniform cuts. Further, we observed that for an irregular ensemble of practical interest the performance of the protograph-based LDPC convolutional codes is consistent with the iterative decoding thresholds of the underlying protographs.

Chapter 6
Trapping Set Analysis for Protograph-based LDPC Convolutional Codes

Abstract

It has been suggested that "near-codewords" may be a significant factor affecting decoding failures of LDPC codes over the AWGN channel. A near-codeword is a sequence that satisfies almost all of the check equations. These near-codewords can be associated with so-called 'trapping sets' (as introduced in Section 4.4) that exist in the Tanner graph of a code. In this chapter, trapping sets of protograph-based LDPC convolutional codes are analysed. LDPC convolutional codes have been shown to be capable of achieving the same capacity-approaching performance as LDPC block codes with iterative message-passing decoding. Further, in Chapter 5 it was shown that some ensembles of LDPC convolutional codes are asymptotically good, in the sense that the average free distance grows linearly with constraint length. Here, asymptotic methods are used to calculate a lower bound for the trapping set growth rates for several ensembles of asymptotically good protograph-based LDPC convolutional codes. This can be used to predict where the error floor will occur for these codes under iterative message-passing decoding. A selection of the results presented here has previously been published in [MPC09a] and [MPC09b].

6.1 Introduction

Trapping sets, as introduced in Section 4.4, are graphical sub-structures existing in the Tanner graph of Low-Density Parity-Check (LDPC) codes. Known initially as *near-codewords* [MP03], they were used to analyse the performance of LDPC codes in the error-floor, or high signal-to-noise ratio (SNR), region of the bit error rate (BER) curve. In [Ric03], Richardson developed these concepts and proposed a two-stage technique to predict the error floor performance of LDPC codes based on trapping sets.

The analysis used in Chapter 5 to calculate ensemble average weight enumerators can be extended to the problem of finding ensemble average trapping set enumerators. In this chapter, building on work by Abu-Surra, Ryan, and Divsalar [ARD07a] (discussed in Chapter 4), asymptotic methods are used to calculate a lower bound for the average trapping set enumerators for several ensembles of regular, asymptotically good, protograph-based LDPC convolutional codes. As in Chapter 5, we will use ensembles of tail-biting LDPC convolutional codes derived from a single protograph-based ensemble of LDPC block codes to obtain a lower bound for the average trapping set enumerators of unterminated, asymptotically good, periodically time-varying LDPC convolutional code ensembles. In the process, we show that the average trapping set enumerators of ensembles of tail-biting LDPC convolutional codes approach the average trapping set enumerator of an associated LDPC convolutional code ensemble as the block length of the tail-biting ensemble increases.

The chapter is structured as follows. In Section 6.2, we modify our previous construction of tail-biting LDPC convolutional codes to enable their use for protograph trapping set enumeration. We then show that the smallest trapping set of a time-varying LDPC convolutional code ensemble can be bounded below by the smallest trapping set of the LDPC block code ensemble formed by tail-biting termination of the convolutional codes. Finally, in Section 6.3, we present new results on the trapping set enumerators of several ensembles of LDPC convolutional codes based on protographs.

6.2 Trapping set enumerators for LDPC convolutional codes

Methods to form a family of protograph-based LDPC convolutional codes from a convolutional protograph were presented in Section 5.2. In this section, a new method is presented which obtains a lower bound for the Δ-trapping set number of an ensemble of unterminated, asymptotically good, time-varying LDPC convolutional codes derived from protograph-based LDPC block codes. To proceed, we form a family of tail-biting LDPC convolutional codes with incremental increases in block length. The tail-biting codes are then used as a tool to obtain the desired bound for the Δ-trapping set number of the unterminated codes.

6.2.1 Tail-biting convolutional codes for trapping set enumeration

Consider the parity-check matrix P_{cc} of the protograph-based, unterminated convolutional code family introduced in Section 5.2. Recall that, in Chapter 5, tail-biting convolutional codes with block length $n = \lambda n_v$ were defined by their unwrapping factor λ. The unwrapping factor can be viewed as the number of times the sliding convolutional structure is repeated before applying tail-biting termination. For $\lambda \geq 1$, the parity-check matrix $P_{tb}^{(\lambda)}$ of the desired tail-biting protograph-based convolutional code with block length λn_v can be written as

$$P_{tb}^{(\lambda)} = \begin{bmatrix} P_l & & & & P_u \\ P_u & P_l & & & \\ & P_u & P_l & & \\ & & \ddots & \ddots & \\ & & & P_u & P_l \end{bmatrix}_{\lambda n_c \times \lambda n_v} . \tag{6.1}$$

Note that the tail-biting convolutional code for $\lambda = 1$ is simply the original protograph-based block code.

Replacing the entries of this matrix with either $N \times N$ permutation matrices, or $N \times N$ all-zeros matrices (as discussed in Section 4.2), creates an ensemble of LDPC codes that can be analysed asymptotically as N goes to infinity, where the sparsity condition of an LDPC code is satisfied for large N. Further, each block code ensemble based on $P_{tb}^{(\lambda)}$ can be unwrapped to form an ensemble of unterminated, periodically time-varying, LDPC convolutional codes with decoding constraint length $\nu_s = Nn_v$ and, in general, period $T = \lambda y$, where the submatrices $H_i(t)$ are of size $Nn_c/y \times Nn_v/y$.

To study the average general trapping set enumerators of these block code ensembles, we add auxiliary flag variables to the protograph following the procedure detailed in Section 4.4. The resulting protograph-based modified parity-check matrix is given by

$$\tilde{P}_{tb}^{(\lambda)} = \left[\begin{array}{ccccc|c} P_l & & & & P_u & \\ P_u & P_l & & & & \\ & P_u & P_l & & & \mathbb{I}_{\lambda n_c} \\ & & \ddots & \ddots & & \\ & & & P_u & P_l & \end{array}\right]_{\lambda n_c \times \lambda(n_v+n_c)},$$

where $\mathbb{I}_n$ is the $n \times n$ identity matrix. For any λ, we can now follow the procedure detailed in Section 4.4 to calculate the Δ-trapping set number $d_{ts}^{(\lambda)}(\Delta)$ for the ensemble of LDPC tail-biting convolutional codes based on the protograph parity-check matrix $P_{tb}^{(\lambda)}$.

As discussed in Chapter 5, as the unwrapping factor λ increases, the protograph parity-check matrix $P_{tb}^{(\lambda)}$ becomes a better representation of the infinite convolutional protograph parity-check matrix P_{cc}. To calculate the general trapping set number of the convolutional code family we should analyse the modified convolutional protograph associated with P_{cc}, which we denote by $\tilde{P}_{cc}$. An example of a modified convolutional protograph is shown in Figure 6.1(a).

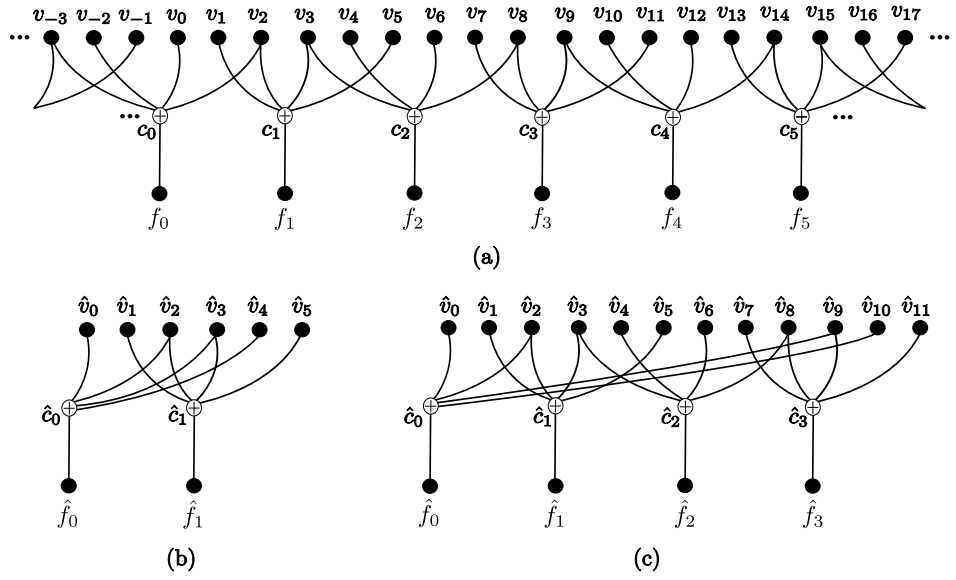


Figure 6.1: *An example of a modified convolutional protograph (a), and corresponding tail-biting versions for (b) $\lambda = 1$, and (c) $\lambda = 2$.*

Note that as λ increases, the modified tail-biting protograph parity-check matrix $\tilde{P}_{tb}^{(\lambda)}$ becomes a better representation of $\tilde{P}_{cc}$. This is reflected in the general trapping set enumerators, and we will show in Section 6.2.2 that the trapping set numbers $d_{ts}^{(\lambda)}(\Delta)$ obtained for increasing λ can be used to form a lower bound for the general trapping set number of the unterminated LDPC convolutional code family.

6.2.2 A lower bound for the convolutional Δ-trapping set number

In this section we present a technique to obtain a lower bound for the convolutional Δ-trapping set number $d_{ccts}(\Delta)$ by showing that $d_{ccts}(\Delta)$ must be at least as large as the Δ-trapping set number $d_{ts}^{(\lambda)}(\Delta)$ of an ensemble of block codes obtained by tail-biting termination of the underlying convolutional code with unwrapping factor λ. Due to the close connection between general trapping sets and codewords of modified codes, the structure of this section follows closely to that of Section 5.3.3. The reader may find it helpful to consult Example 6.1 when considering Theorem 6.1 and its proof.

Theorem 6.1 *Consider forming an unterminated, periodically time-varying convolutional code with rate $R = 1 - n_c/n_v$, constraint length $\nu_s = n_v$ and period $T = y$ from (5.1), as described in Section 5.2. Let $d_{ts}^{(\lambda)}(\Delta)$ be the Δ-trapping set number of a tail-biting convolutional code with block length $n = \lambda n_v$ formed by terminating the associated convolutional code with unwrapping factor $\lambda > 0$. Then, for any $\Delta \geq 0$, the Δ-trapping set number $d_{ccts}(\Delta)$ of the unterminated LDPC convolutional code is bounded below by $d_{ts}^{(\lambda)}(\Delta)$ for any unwrapping factor λ; i.e.,*

$$d_{ccts}(\Delta) \geq d_{ts}^{(\lambda)}(\Delta).$$

Proof. Consider the unterminated convolutional code whose parity-check matrix is given in (5.1). Now take any (a, b) general trapping set that consists of a set of a variable nodes

$$V_{ts} = \{v_{I(0)}, v_{I(1)}, \cdots, v_{I(a-1)}\}, \tag{6.2}$$

that result in a set of exactly b odd-degree check nodes

$$C_{ts} = \{c_{J(0)}, c_{J(1)}, \cdots, c_{J(b-1)}\} \tag{6.3}$$

in the induced bipartite subgraph. Here, the vectors $\mathbf{I} = [I(0)\ \ I(1)\ \cdots\ I(a-1)]$ and $\mathbf{J} = [J(0)\ \ J(1)\ \cdots\ J(b-1)]$ are index vectors of length a and b, respectively, and the entries of each vector are unique. Consider the modular index vector $\hat{\mathbf{I}}$ (resp. $\hat{\mathbf{J}}$) obtained by letting $\hat{I}(i) = I(i) \bmod \lambda n_v$, for $i = 0, 1, \ldots, a-1$ (resp. $\hat{J}(j) = J(j) \bmod \lambda n_c$, for $j = 0, 1, \ldots, b-1$). This operation corresponds to wrapping the unterminated convolutional code to a tail-biting convolutional code with unwrapping factor λ whose parity-check matrix is given in (6.1).

Proposition. The sets

$$\hat{V}_{ts} = \left\{ v_0 k_0, v_1 k_1, \ldots, v_{\lambda n_v - 1} k_{\lambda n_v - 1} \,\middle|\, k_i = \sum_{j=0}^{a-1} \kappa_{\hat{I}(j),i} \bmod 2 \right\} \tag{6.4}$$

$$\hat{C}_{ts} = \left\{ c_0 l_0, c_1 l_1, \ldots, c_{\lambda n_c - 1} l_{\lambda n_c - 1} \,\middle|\, l_i = \sum_{j=0}^{b-1} \kappa_{\hat{J}(j),i} \bmod 2 \right\} \tag{6.5}$$

yield an $(\hat{a}, \hat{b})$ general trapping set in the Tanner graph of the tail-biting code, where $\kappa_{a,b}$ is the Kronecker Delta function: $\kappa_{a,b} = 1$ iff $a = b$, and zero otherwise.

Proof. Recall that an $(\hat{a}, \hat{b})$ general trapping set is a set of $\hat{a}$ variable nodes that induce $\hat{b}$ odd-degree check nodes. The odd degree check nodes can be satisfied by setting appropriate auxiliary flag variables to have weight one. Thus, combining (6.4) and (6.5), the proposition is equivalent to the claim that

$$\begin{aligned} \hat{X}_{ts} &= \{v_0 k_0, v_1 k_1, \ldots, v_{\lambda n_v - 1} k_{\lambda n_v - 1}, f_0 l_0, f_1 l_1, \ldots, f_{\lambda n_c - 1} l_{\lambda n_c - 1}\} \\ &\equiv \left[\begin{array}{cccccccc} k_0 & k_1 & \ldots & k_{\lambda n_v - 1} & l_0 & l_1 & \ldots & l_{\lambda n_c - 1} \end{array} \right] = \hat{\mathbf{x}}_{ts}, \end{aligned}$$

where $k_i = \sum_{j=0}^{a-1} \kappa_{\hat{I}(j),i} \bmod 2$ and $l_i = \sum_{j=0}^{b-1} \kappa_{\hat{J}(j),i} \bmod 2$, is a codeword of the modified tail-biting convolutional code with parity-check matrix $\tilde{P}_{tb}^{(\lambda)}$. Here, the auxiliary variable node f_i is attached to check node c_i for $i = 0, \ldots, \lambda n_v$.

The (a, b) general trapping set of the convolutional code with parity-check matrix P_{cc} defined by equations (6.2) and (6.3) can equivalently be viewed as a codeword[1] $\mathbf{x}_{ts}$ of the modified convolutional code with parity-check matrix $\tilde{P}_{cc}$, where

$$\mathbf{x}_{ts} = \left[\begin{array}{cccccc|cccccc|c} v_0 & \ldots & v_{\lambda n_v - 1} & f_0 & \ldots & f_{\lambda n_c - 1} & v_{\lambda n_v} & \ldots & v_{2\lambda n_v - 1} & f_{\lambda n_c} & \ldots & f_{2\lambda n_c - 1} & \ldots \end{array} \right].$$

[1] This will be demonstrated in Example 6.1.

Here, we continue the notation that $v_i = 1$ if $i \in \mathbf{I}$, otherwise $v_i = 0$ (resp. $f_j = 1$ if $j \in \mathbf{J}$, otherwise $f_j = 0$). Consider the infinite concatenated parity-check matrix

$$\tilde{P}_{\infty}^{(\lambda)} = \left[\tilde{P}_{tb}^{(\lambda)} \,\middle|\, \tilde{P}_{tb}^{(\lambda)} \,\middle|\, \dots \right].$$

It is clear that any codeword $\mathbf{x}_{ts}$ in the modified convolutional code satisfies $\mathbf{x}_{ts} \cdot \left(\tilde{P}_{\infty}^{(\lambda)} \right)^T = \mathbf{0}$. Suppose now that we partition $\mathbf{x}_{ts}$ into sections of length $\lambda(n_v + n_c)$, i.e,

$$\begin{aligned} \mathbf{x}_{ts} &= \left[\; v_0 \;\; \dots \;\; v_{\lambda n_v - 1} \;\; f_0 \;\; \dots \;\; f_{\lambda n_c - 1} \,\middle|\, v_{\lambda n_v} \;\; \dots \;\; v_{2\lambda n_v - 1} \;\; f_{\lambda n_c} \;\; \dots \;\; f_{2\lambda n_c - 1} \,\middle|\, \dots \; \right] \\ &= \left[\; \mathbf{x}_0 \,\middle|\, \mathbf{x}_1 \,\middle|\, \dots \; \right], \end{aligned}$$

where each $\mathbf{x}_i$ has length $\lambda(n_v + n_c)$. Consider the vector $\hat{\mathbf{x}}_{ts} = \sum_{k=0}^{\infty} \mathbf{x}_i \bmod 2$ of length $\lambda(n_v + n_c)$. Due to the repeated structure of $\tilde{P}_{\infty}^{(\lambda)}$, we observe that $\hat{\mathbf{x}}_{ts} \cdot (\tilde{P}_{tb}^{(\lambda)})^T = \mathbf{0}$. Thus proving that $\hat{\mathbf{x}}_{ts}$ is a codeword of the modified ensemble with parity-check matrix $\tilde{P}_{tb}^{(\lambda)}$, or equivalently, an $(\hat{a}, \hat{b})$ general trapping set of the tail-biting convolutional code with parity-check matrix $P_{tb}^{(\lambda)}$ □

The values of $\hat{a}$ and $\hat{b}$ are determined as follows: if the elements of the modular index vectors $\hat{\mathbf{I}}$ and $\hat{\mathbf{J}}$ are still unique, then we say that the original (a, b) general trapping set has not been affected by the wrapping procedure and therefore $\hat{a}$ equals a and $\hat{b}$ equals b. If, however, there are $k > 1$ copies of the same index value in $\hat{\mathbf{I}}$ (resp. $\hat{\mathbf{J}}$), then the value of $\hat{a}$ (resp. $\hat{b}$) is reduced by $2\lfloor \frac{k}{2} \rfloor$. This corresponds to the case when multiple variable or check nodes in the trapping set are wrapped back on top of each other. If there is an even number of such nodes at the same position, none of them survive as a member of the tail-biting $(\hat{a}, \hat{b})$ trapping set, and if the repetition value k is odd, only one node survives. These reductions are cumulative over all the repetitive index values in $\hat{\mathbf{I}}$ and $\hat{\mathbf{J}}$.

This implies that a general trapping set in the unterminated convolutional code induces a general trapping set in any of its tail-biting terminated codes that has at most the same size as the original one. It follows that the smallest general trapping set size of an unterminated convolutional code is at least as large as the smallest general trapping set size of any of its tail-biting terminated convolutional codes, i.e., codes with arbitrary unwrapping factor λ. □

Example 6.1. Consider the following $(3, 6)$-regular protograph parity-check matrix of Exercise 5.9 given as follows:

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{6.6}$$

We can form the convolutional protograph by unwrapping P with step size $n_c/y \times n_v/y = 1 \times 2$. The appropriate cut is displayed in equation (6.6). Consider the set of variable nodes

$$V_{ts} = \{v_0, v_2, v_6, v_8, v_{10}, v_{11}, v_{14}\}$$

in the resulting convolutional protograph displayed in Figure 6.2(a). We observe that the seven nodes in this set induce five odd-degree check nodes ($C_{ts} = \{c_0, c_6, c_7, c_8, c_9\}$). I.e., the resulting subgraph corresponds to a $(7, 5)$ general trapping set in the convolutional protograph with parity-check matrix P_{cc}. This trapping set is highlighted in Figure 6.2(a). We observe in Figure 6.2(b) that after modifying the convolutional prototograph and setting the auxiliary variables to appropriate values, this $(7, 5)$ trapping set can be viewed as a codeword in the modified convolutional protograph.

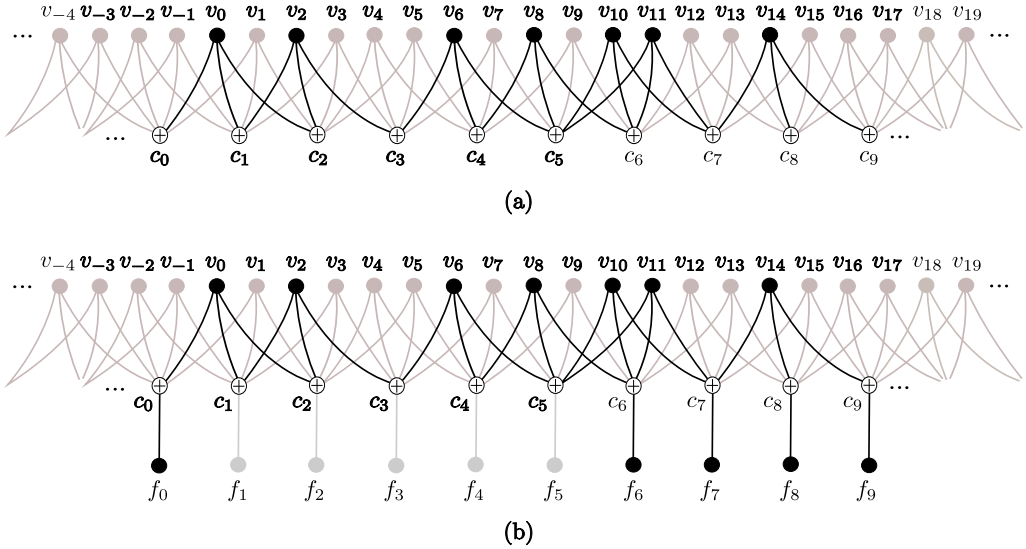


Figure 6.2: *(a) A $(7, 5)$ general trapping set in the Tanner graph of a $(3, 6)$-regular convolutional code with parity-check matrix P_{cc}, and (b) the corresponding codeword in the modified convolutional ensemble with parity-check matrix $\tilde{P}_{cc}$.*

Suppose, without loss of generality, that the $(7, 5)$ general trapping set has index vectors $\mathbf{I} = [0\ 2\ 6\ 8\ 10\ 11\ 14]$ and $\mathbf{J} = [0\ 6\ 7\ 8\ 9]$. Consider setting $\lambda = 2$. By wrapping the unterminated code onto itself with modulus $2n_v = 12$ for variable nodes and with modulus $2n_c = 6$ for check nodes, we obtain a sequence in the $(3, 6)$-regular tail-biting convolutional code with unwrapping factor $\lambda = 2$. The resulting modular index vectors, with modulus 12 and 6 respectively, are $\hat{\mathbf{I}} = [0\ 2\ 6\ 8\ 10\ 11\ 2]$ and $\hat{\mathbf{J}} = [0\ 0\ 1\ 2\ 3]$. Observe that index values $\hat{I}(1)$ and $\hat{I}(6)$ equal 2, i.e., $k = 2$ for this index value, and we obtain the reduction value of $2\lfloor\frac{2}{2}\rfloor = 2$. Since k is even, the variable nodes corresponding to these index values do not appear in the reduced trapping set, and hence $\hat{\mathbf{I}}$ represents the set of variable nodes $\{\hat{v}_0, \hat{v}_6, \hat{v}_8, \hat{v}_{10}, \hat{v}_{11}\}$ in the Tanner graph of a $(3, 6)$-regular tail-biting convolutional code and $\hat{a} = a - 2 = 5$. The same reduction in $\hat{b}$ is obtained as a result of the equality $\hat{J}(0) = \hat{J}(1) = 0$. Hence $\hat{b} = b - 2 = 3$, and the reduced set of check nodes represented by $\hat{\mathbf{J}}$ is $\{\hat{c}_1, \hat{c}_2, \hat{c}_3\}$. The corresponding sequence is

$$\hat{V}_{ts} = \{\hat{v}_0, \hat{v}_6, \hat{v}_8, \hat{v}_{10}, \hat{v}_{11}\} \equiv [1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1],$$

and it induces a subgraph in the tail-biting convolutional code graph with three odd-degree check nodes ($\hat{c}_1$, $\hat{c}_2$ and $\hat{c}_3$); i.e., the modular index vectors $\hat{\mathbf{I}}$ and $\hat{\mathbf{J}}$ represent a $(5, 3)$ general trapping set in the Tanner graph of a $(3, 6)$-regular tail-biting convolutional code, as illustrated in Fig. 6.3.

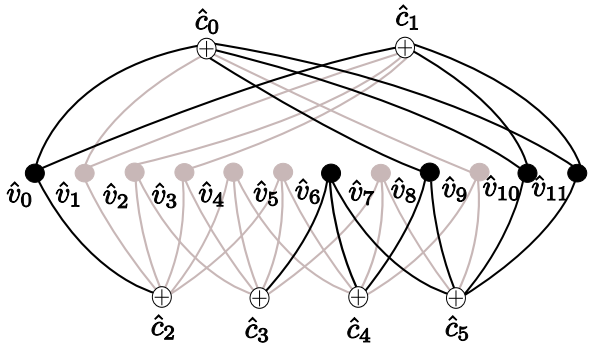


Figure 6.3: *A $(5, 3)$ general trapping set in the Tanner graph of a $(3, 6)$-regular tail-biting convolutional code with unwrapping factor $\lambda = 2$.*

Setting the unwrapping factor $\lambda = 1$, with modulus $n_v = 6$ and $n_c = 3$ respectively, the resulting modular index vectors are $\hat{\mathbf{I}} = [0\ 2\ 0\ 2\ 4\ 5\ 2]$ and $\hat{\mathbf{J}} = [0\ 0\ 1\ 2\ 0]$. The set of variable nodes corresponding to $\hat{\mathbf{I}}$ is $\hat{V}_{ts} = \{\hat{v}_2, \hat{v}_4, \hat{v}_5\}$. We observe from Fig. 6.4 that $\hat{V}_{ts}$ induces three odd-degree check nodes $\hat{C}_{ts} = \{\hat{c}_0, \hat{c}_1, \hat{c}_2\}$. It is easy to check that the set $\hat{C}_{ts}$ corresponds to the modular index vector $\hat{\mathbf{J}}$.

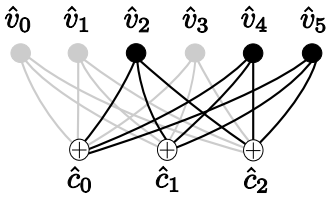


Figure 6.4: *A $(3,3)$ general trapping set in the Tanner graph of a $(3,6)$-regular tail-biting convolutional code with unwrapping factor $\lambda = 1$.*

For the unwrapping factor $\lambda = 4$, with modulus $3n_v = 18$ and $3n_c = 9$ respectively, we obtain the modular index vectors $\hat{\mathbf{I}} = [0\ 2\ 6\ 8\ 10\ 11\ 14]$ and $\hat{\mathbf{J}} = [0\ 6\ 7\ 8\ 0]$. The only reduction occurs for $J(0) = J(4) = 0$. Thus in the tail-biting convolutional code with unwrapping factor $\lambda = 4$, this results in a $(7,3)$ general trapping set. Finally, for $\lambda \geq 5$, with modulus λn_v and λn_c respectively, we obtain the modular vectors $\hat{\mathbf{I}} = \mathbf{I}$ and $\hat{\mathbf{J}} = \mathbf{J}$; i.e., there is no reduction in the $(7,5)$ general trapping set of the tail-biting convolutional code with unwrapping factor $\lambda \geq 5$. □

After performing the copy-and-permute operation to a protograph, the following corollary for protograph-based convolutional codes results. Here, the low-density criteria for an LDPC code is satisfied if we choose a sufficiently large positive integer N.

Corollary 6.2 *Fix $N \in \mathbb{Z}$, $N > 0$, and let a protograph-based (LDPC) block code with parity-check matrix H be formed by replacing each one in the protograph parity-check matrix P with an $N \times N$ permutation matrix chosen randomly and independently and each zero with an $N \times N$ all-zeros matrix as described in Section 4.2. Consider forming a rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ unterminated, periodically time-varying (LDPC) convolutional code with decoding constraint length $\nu_s = Nn_v$ and, in general, period $T = y$, by unwrapping H with step size $Nn_v/y \times Nn_c/y$. Let $d_{ts}^{(\lambda)}(\Delta)$ be the Δ-trapping set number of the associated (LDPC) tail-biting convolutional code with block length $n = \lambda N n_v$ and unwrapping factor $\lambda > 0$. Then, for any $\Delta \geq 0$, the Δ-trapping set number $d_{ccts}(\Delta)$ of the unterminated LDPC convolutional code is bounded below by $d_{ts}^{(\lambda)}(\Delta)$, for any unwrapping factor λ.*

Proof. The result follows directly by replacing n_v by Nn_v and n_c by Nn_c in the proof of Theorem 6.1. □

We have now shown that the Δ-trapping set number $d_{ccts}(\Delta)$ of an unterminated, periodically time-varying, protograph-based (LDPC) convolutional code obtained by unwrapping a

protograph-based block code is bounded below by the Δ-trapping set number of any associated (LDPC) tail-biting convolutional code. We now formalise the more general result that the same result holds for ensemble averages.

Corollary 6.3 *Consider forming the parity-check matrix H described in Corollary 6.2. By letting the $N \times N$ permutation matrices vary over the $N!$ different choices, we obtain an ensemble of protograph-based (LDPC) block codes. The associated ensemble of rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$ unterminated, periodically time-varying (LDPC) convolutional codes with decoding constraint length $\nu_s = Nn_v$ and, in general, period $T = y$ can be obtained by unwrapping each parity-check matrix H in the block code ensemble with step size $Nn_v/y \times Nn_c/y$. Denote by $\overline{d}_{ts}^{(\lambda)}(\Delta)$ the average Δ-trapping set number of the associated (LDPC) tail-biting convolutional code ensemble with block length $n = \lambda N n_v$ and unwrapping factor $\lambda > 0$. Then the average Δ-trapping set number $\overline{d}_{ccts}(\Delta)$ of the unterminated (LDPC) convolutional code ensemble is bounded below by $\overline{d}_{ts}^{(\lambda)}(\Delta)$ for any unwrapping factor λ.*

Proof. By using Corollary 6.2, it is clear that for every member of the ensemble $d_{ccts}(\Delta) \geq d_{ts}^{(\lambda)}(\Delta)$ for any $\Delta > 0$ and any unwrapping factor λ. The result $\overline{d}_{ccts}(\Delta) \geq \overline{d}_{ts}^{(\lambda)}(\Delta)$ then follows directly. □

As there is no danger of ambiguity, from now on we will drop the overline notation when discussing ensemble average trapping set numbers. Thus for an ensemble of convolutional codes $\overline{d}_{ccts}(\Delta) = d_{ccts}(\Delta)$, and for an ensemble of tail-biting convolutional codes with unwrapping factor λ, $\overline{d}_{ts}^{(\lambda)}(\Delta) = d_{ts}^{(\lambda)}(\Delta)$. Note that ensemble average trapping set numbers $d_{ccts}(\Delta)$ and $d_{ts}^{(\lambda)}(\Delta)$ need not be integers.

We can now prove our main result to demonstrate a lower bound for the Δ-trapping set number of the unterminated, protograph-based (LDPC) convolutional code based on the convolutional protograph P_{cc} defined in Section 5.1.

Theorem 6.4 *Consider forming a family of unterminated, periodically time-varying LDPC convolutional codes with rate $R = 1 - Nn_c/Nn_v = 1 - n_c/n_v$, constraint length $\nu_s = Nn_v$, and period $T = \lambda y$ from (6.1), as described in Section 6.2.1. Let $d_{ccts}^{(\lambda)}(\Delta)$ be the Δ-trapping set number of the code with $T = \lambda y$, and let $d_{ccts}(\Delta) = \max_{\lambda > 0} d_{ccts}^{(\lambda)}(\Delta)$, which we call the Δ-trapping set number of the unterminated convolutional code family. Then, for any tail-biting termination with unwrapping factor λ, $d_{ccts}(\Delta)$ is bounded below by $d_{ts}^{(\lambda)}(\Delta)$ for any λ; i.e.,*

$$d_{ccts}(\Delta) \geq d_{ts}^{(\lambda)}(\Delta). \tag{6.7}$$

Proof. It follows from Corollary 6.3 that $d_{ccts}^{(\lambda)}(\Delta) \geq d_{ts}^{(\lambda)}(\Delta)$ for any λ. The result then follows directly from the definition of $d_{ccts}(\Delta)$. □

Intuitively, as λ increases, the tail-biting code becomes a better representation of its associated unterminated convolutional code, with $\lambda \to \infty$ corresponding to a non-periodically time-varying convolutional code. This is reflected in the average general trapping set enumerators, and it will be shown in Section 6.3 that increasing λ provides us with Δ-trapping set growth rates $\delta_{ts}^{(\lambda)}(\Delta)$ that converge to a lower bound for $\delta_{ccts}(\Delta)$, which we call the Δ-trapping set growth rate of the unterminated convolutional code family.

So far in this chapter we have considered only general trapping sets. However, there exist an important subset of these sets called *elementary trapping sets*. These were introduced formally in Section 4.4.2, and it was shown there that in order to calculate elementary trapping set enumerators for protograph-based codes we require a slightly different analytical model than the one used for general trapping sets. One such method was presented in [ARD07a]. Essentially, an (a, b) elementary trapping set is an (a, b) general trapping set with the restriction that all of the induced check nodes must have either degree one or two, and there must be exactly b degree-one check nodes. Under this modified analysis, the same principles that have been discussed in this section hold.

Moreover, it has been shown that, in general, small trapping sets have the largest effect on the error floor [Ric03, MSW07]. This is because small trapping sets contain relatively few variable nodes, and thus it is more likely that all of the variable nodes in a small trapping set have unreliable channel values than in a larger trapping set. Further, examinations of the small trapping sets dominating in the error floor region have shown that the sets with low degree check nodes cause the most trouble. This is because of the low connectivity of these check nodes to the rest of the graph, which results in the trapping set not being able to obtain enough independent (and possibly helpful) messages during decoding iterations. Extensive simulations (see, e.g., [Ric03]) have shown that in fact most of the decoding failures in iterative decoding correspond to elementary trapping sets.

As was discussed in [ARD07a], it is expected that the lower bounds calculated in Section 6.3 for the Δ-trapping set number $d_{ccts}(\Delta)$ of protograph-based LDPC convolutional code ensembles obtained using general trapping sets will be approximately the same for those calculated using elementary trapping sets. This is because for small a and b most of the general trapping sets are

in fact elementary. However, similar results to those presented in this chapter for elementary trapping sets should most definitely be the subject of future research. This will be discussed further in Chapter 7.

6.2.3 The convolutional Δ-trapping set growth rate

As noted in Section 4.4.3, the Δ-trapping set growth rate of a block code ensemble is defined as its Δ-trapping set number to block length ratio. For the protograph-based tail-biting LDPC convolutional code ensembles defined in Section 6.2.1, this ratio is therefore given as

$$\delta_{ts}^{(\lambda)}(\Delta) = \frac{d_{ts}^{(\lambda)}(\Delta)}{n} = \frac{d_{ts}^{(\lambda)}(\Delta)}{\lambda N n_v} = \frac{d_{ts}^{(\lambda)}(\Delta)}{\lambda \nu_s}.$$

Using (6.7), we then obtain

$$\delta_{ts}^{(\lambda)}(\Delta) \leq \frac{d_{ccts}^{(\lambda)}(\Delta)}{\lambda \nu_s} \leq \frac{d_{ccts}(\Delta)}{\lambda \nu_s}. \tag{6.8}$$

We note that, for convolutional codes, the length of the shortest codeword is equal to the encoding constraint length ν_e, which in general differs from the decoding constraint length ν_s. Assuming minimal encoder and syndrome former matrices, the relationship between ν_e and ν_s can be expressed as

$$\nu_e = \frac{1-R}{R}\nu_s, \tag{6.9}$$

which implies that, for code rates less than $1/2$, the encoding constraint length is larger than the decoding constraint length, and vice versa for code rates greater than $1/2$.

Combining (6.8) and (6.9) gives us the desired lower bound

$$\delta_{ccts}(\Delta) \geq \frac{R}{1-R}\lambda\delta_{ts}^{(\lambda)}(\Delta),$$

where $\delta_{ccts}(\Delta) = d_{ccts}(\Delta)/\nu_e$ is the Δ-trapping set growth rate[2] of the unterminated LDPC convolutional code family.

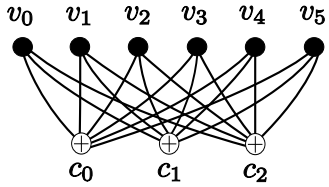
[2]If the syndrome former matrix is not in minimal form, (6.9) results in an upper bound for ν_e, which implies that $\delta_{ccts}(\Delta)$ is underestimated in this case.

6.3 Trapping set analysis

We now present a trapping set analysis for several asymptotically good ensembles of unterminated, time-varying LDPC convolutional codes. As described in Section 6.2, we make use of ensembles of tail-biting LDPC convolutional codes to obtain a lower bound for the desired Δ-trapping set growth rate of the associated unterminated convolutional code family.

6.3.1 Regular ensembles with gcd$(n_c, n_v) > 1$

Example 6.2. Consider a $(3, 6)$-regular LDPC code with the following protograph and associated parity-check matrix:



$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

A family of rate $R = 1/2$ tail-biting LDPC convolutional code ensembles can be generated according to the displayed cut. We now proceed to calculate the Δ-trapping set growth rate $\delta_{ts}^{(\lambda)}(\Delta)$ for the modified tail-biting convolutional code ensembles with base parity-check matrices $\tilde{P}_{tb}^{(\lambda)}$ for various fixed values of Δ and increasing values of the unwrapping factor λ. Note that setting $\Delta = \beta/\alpha = 0$ corresponds to the minimum distance growth rate problem of Chapter 5. Thus, for $\lambda = 1$, which corresponds to the $(3, 6)$-regular block code ensemble, $\delta_{ts}^{(1)}(0) = \delta_{min} = 0.023$, where δ_{min} is the minimum distance growth rate for the $(3, 6)$-regular ensemble, as originally calculated by Gallager [Gal62]. Further, for larger values of λ, the value for $\delta_{ts}^{(\lambda)}(0)$ agrees with the earlier results for minimum distance growth rates of tail-biting convolutional codes given in Example 5.9.

As Δ ranges from 0 to ∞, the points $(\delta_{ts}^{(\lambda)}(\Delta), \Delta \cdot \delta_{ts}^{(\lambda)}(\Delta))$ trace out the so-called *zero-contour curve* for a protograph-based block code ensemble [ARD07a]. Note that we are particularly interested in small values of Δ, since this means that the ratio of odd-degree check nodes to variable nodes inducing them is small. This implies that there are relatively few odd-degree check nodes available to pass messages to the large number of variable nodes. The zero-contour curves for Example 6.2 are shown in Figure 6.5, and the Δ-trapping set growth rates are highlighted for $\Delta = 0.02$.

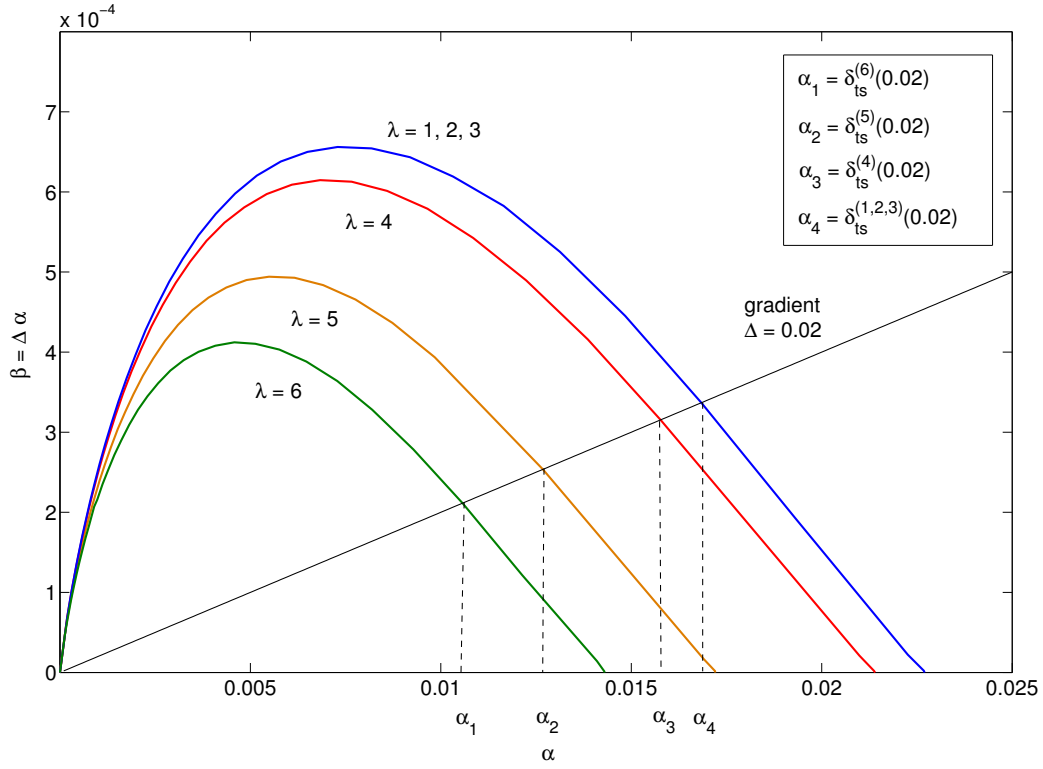


Figure 6.5: *Zero-contour curves for Example 6.2.*

The zero-contour curve is key to understanding the role of trapping sets in iterative decoding. Code ensembles with large Δ-trapping set numbers $d_{ts}^{(\lambda)}(\Delta)$ are of primary interest, since small trapping sets dominate iterative decoding performance in the error floor [Ric03]. Thus we want the Δ-trapping set growth rate $\delta_{ts}^{(\lambda)}(\Delta)$ to exist and be as large as possible for each value of Δ. We observe in Fig. 6.5 that $\delta_{ts}^{(\lambda_1)}(\Delta) \leq \delta_{ts}^{(\lambda_2)}(\Delta)$ for $\lambda_1 > \lambda_2$. This is analogous to the decrease in the minimum distance growth rate with increasing λ observed in Section 5.4. If a zero-contour curve of ensemble A is always below the zero-contour curve of ensemble B, then, in general, we would expect a code drawn from ensemble A to exhibit poorer error floor performance than one drawn from ensemble B. Thus we would expect worse error floor performance with increasing λ for the tail-biting convolutional code ensembles.[3] For this (3,6)-regular ensemble, the lower bound for $\delta_{ccts}(\Delta)$ is simply $\delta_{ccts}(\Delta) \geq \frac{R}{1-R}\lambda\delta_{ts}^{(\lambda)}(\Delta) = \lambda\delta_{ts}^{(\lambda)}$, since $\frac{R}{1-R} = 1$ in this case. The Δ-trapping set growth rates for Example 6.2 are plotted in Fig. 6.6 for $\Delta = 0$, 0.01 and 0.05.

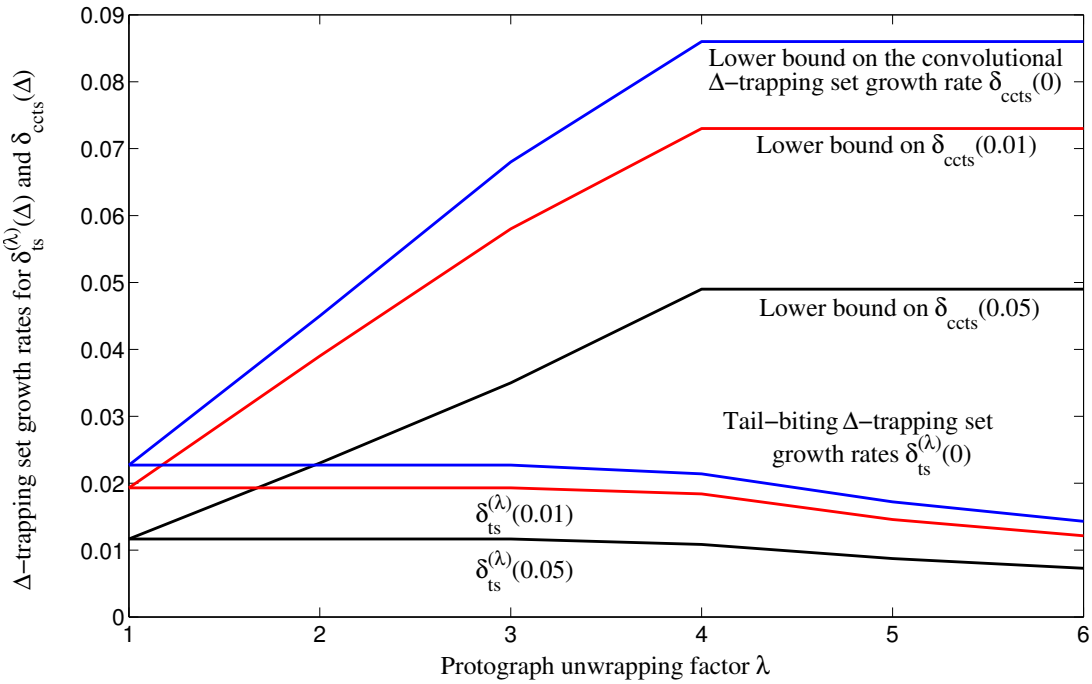


Figure 6.6: *Δ-trapping set growth rates for Example 6.2.*

[3] We observe from the zero-contour curves of Example 6.2 that increasing λ results in smaller Δ-trapping set growth rates for $\lambda \geq 3$. However, we must be careful in this case to remember that the block lengths also increase and the Δ-trapping set number is $d_{ts}^{(\lambda)}(\Delta) = n\delta_{ts}^{(\lambda)}(\Delta) = N\lambda n_v \delta_{ts}^{(\lambda)}(\Delta)$.

We observe that, once the unwrapping factor λ of the tail-biting convolutional code ensemble exceeds 3, the lower bound for $\delta_{ccts}(\Delta)$ levels off for each distinct value of Δ. We also observe a significant increase in the value of $\delta_{ccts}(\Delta)$ compared to $\delta_{ts}^{(1)}(\Delta)$, the Δ-trapping set growth rate of the underlying block code ensemble. □

Example 6.3. Consider the rate $R = 2/3$, $(3, 9)$-regular LDPC code ensemble. We form a protograph in the usual fashion, creating three check nodes, each of which connect to all nine variable nodes, and we observe that $\gcd(3, 9) = 3$. The protograph parity-check matrix and defined cut are displayed below:

$$P = \left[\begin{array}{\overline{ccc}|ccc|ccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right].$$

For this rate $R = 2/3$ ensemble, the lower bound for $\delta_{ccts}(\Delta)$ is $\delta_{ccts}(\Delta) \geq \frac{R}{1-R}\lambda\delta_{ts}^{(\lambda)}(\Delta) = 2\lambda\delta_{ts}^{(\lambda)}(\Delta)$. We observe that, as in Example 6.2, the Δ-trapping set growth rates calculated for fixed values of Δ with increasing λ provide us with a lower bound on the convolutional Δ-trapping set growth rate, which exceeds the value of $\delta_{ts}^{(1)}(\Delta)$, the Δ-trapping set growth rate of the protograph-based LDPC block code ensemble. The bounds calculated for several values of Δ are given in Table 6.1. □

Δ	$\delta_{ts}^{(1)}(\Delta)$	lower bound for $\delta_{ccts}(\Delta)$
0	0.0054	0.037
0.01	0.0046	0.031
0.05	0.0027	0.018
0.10	0.0014	0.010

Table 6.1: *Block code Δ-trapping set growth rates and lower bounds on the convolutional Δ-trapping set growth rates for several values of Δ for the regular $(3, 9)$-ensemble.*

6.3.2 Regular ensembles with gcd$(n_c, n_v) = 1$

For the following $R = 1/4$, $(3, 4)$-regular and $R = 2/5$, $(3, 5)$-regular ensembles, we form the trivial 'all-ones' protograph parity-check matrix of size $n_c \times n_v$. The protograph is then unwrapped according to the nonuniform cutting vector $\boldsymbol{\xi}$ method as described in Section 5.2. The nonuniform cuts chosen are consistent with those previously presented in Chapter 5, and they are displayed in Fig. 6.7.

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \text{ and } P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

(a) (b)

Figure 6.7: *Protograph parity-check matrices with associated nonuniform cut for (a) the* $(3, 4)$*-regular ensemble, and (b) the* $(3, 5)$*-regular ensemble.*

Just as we expect the tail-biting ensemble zero-contour curve values $\delta_{ts}^{(\lambda)}(0)$ to correspond to minimum distance growth rates of the associated block code ensembles, we expect that the lower bound for the free distance growth rate δ_{free} of the regular convolutional codes (found in Chapter 5) based on the displayed cuts correspond to the value $\delta_{ccts}(0)$ of the convolutional lower bound zero-contour curve. This can be seen in Figure 6.8, where the lower bounds on $\delta_{ccts}(\Delta) = \alpha$ are plotted against $\Delta\alpha = \beta$ to form a convolutional lower bound zero-contour curve for each of the regular convolutional ensembles considered.

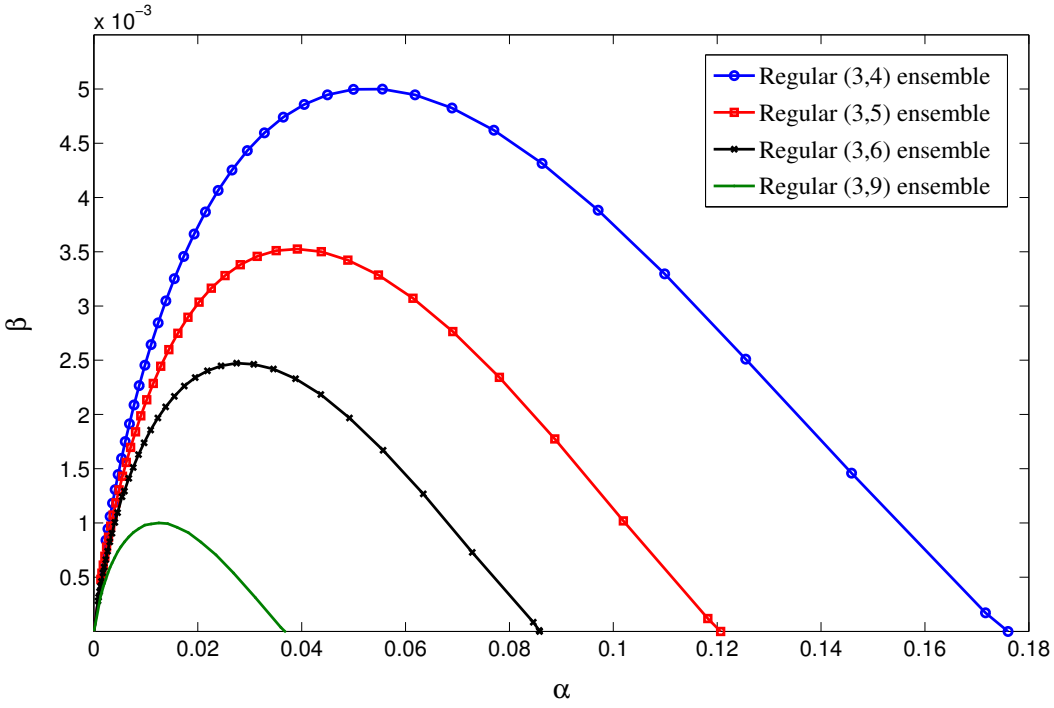
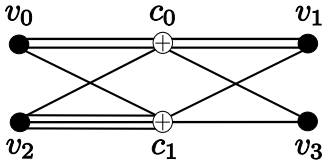


Figure 6.8: *Lower bounds on* $\delta_{ccts}(\Delta)$ *for several regular ensembles.*

We observe that, in general, as the rate increases the convolutional lower bound zero-contour curves become lower, indicating the likelihood of worse error-floor performance. This is consistent with the conclusions drawn from the free distance bounds presented in Section 5.4.

6.3.3 An irregular ensemble

Example 6.4. The following irregular protograph is from the Repeat Jagged Accumulate (RJA) family [DDJ06]. In this paper, it was shown to have a good iterative decoding threshold ($\gamma_{iter} = 1.0$ dB) while maintaining linear minimum distance growth ($\delta_{min} = 0.013$). We display below the associated P matrix and cut used to generate the family of tail-biting LDPC convolutional code ensembles.



$$\leftrightsquigarrow \qquad P = \left[\begin{array}{cc|cc} 2 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \end{array}\right].$$

A lower bound for the free distance growth rate of the unterminated LDPC convolutional code family, $\delta_{free} \geq 0.057$, obtained according to this cut was presented in Example 5.13. We observe that, as in previous examples, the Δ-trapping set growth rates calculated for increasing λ provide us with a lower bound for the Δ-trapping set growth rate of the unterminated convolutional code ensemble, for each value of Δ. The bounds calculated for several values of Δ are given below in Table 6.2. Recall that $\delta_{ts}^{(1)}(\Delta)$ is the Δ-trapping set growth rate of the protograph-based LDPC block code ensemble. □

Δ	$\delta_{ts}^{(1)}(\Delta)$	lower bound for $\delta_{ccts}(\Delta)$
0	0.013	0.057
0.01	0.010	0.046
0.04	0.005	0.026
0.12	0.001	0.005

Table 6.2: *Results for the lower bounds on $\delta_{ccts}(\Delta)$ calculated for several values of Δ for Example 6.4.*

6.4 Conclusions

In this chapter, asymptotic methods were used to calculate a lower bound for the Δ-trapping set number that grows linearly with constraint length for several regular, asymptotically good ensembles of unterminated, protograph-based, time-varying LDPC convolutional codes. It was shown that the Δ-trapping set growth rates of the LDPC convolutional code ensembles exceed the growth rates of the corresponding LDPC block code ensembles on which they are based. These large trapping set growth rates suggest that the LDPC convolutional codes will exhibit good iterative decoding performance in the error floor.

Chapter 7
Concluding Remarks

7.1 Summary of main contributions

Various aspects of channel codes with diagonal structures have been studied in this thesis. To begin with, so-called defectively cyclic block codes were defined. The parity-check matrices of these codes have a sliding diagonal structure and as such can be completely described by a polynomial. The first major result presented in the thesis was, for such codes, an efficient alternative to the standard method of obtaining a generator matrix G from a parity-check matrix H using Gaussian elimination. The new method uses simple polynomial algebra resulting from the diagonal structure, and the resulting G can be completely defined by a polynomial. In general G will be non-systematic, and the polynomial representation leads to efficient encoder implementation using shift registers. Further, the polynomial description enables the system designer to choose any code length and code rate desired. Importantly, the density of G depends solely on its corresponding polynomial.

The method was subsequently extended with little added complexity to handle a more complicated parity-check matrix structure consisting of an arbitrary finite number of stacked defective blocks. This allows a large class of practical parity-check matrices to be considered as candidates for the method, e.g., those of LDPC block codes. The method itself is not particular to binary codes and there is a natural extension to codes over GF(q) with this cyclic block structure. For these codes it was shown that the induced binary codes possess a quasi-cyclic type structure.

The well-known links between quasi-cyclic codes and convolutional codes motivated the study of Part II of the thesis. LDPC convolutional codes have an inherent diagonal structure by nature, and in this part of the thesis we focus on two structural properties of these codes. The first interesting property is the minimum free distance of LDPC convolutional code ensembles. In Chapter 5, looking to mirror Gallager's results for LDPC block code ensembles, asymptotic methods were used to calculate a lower bound for free distance that grows linearly with constraint length for several ensembles of regular and irregular, unterminated, protograph-based time-varying LDPC convolutional codes. It was shown that the free distance growth rates of the LDPC convolutional code ensembles exceed the minimum distance growth rates of the corresponding LDPC block code ensembles. Further, it was observed that for an irregular ensemble of practical interest the performance of the protograph-based LDPC convolutional codes is consistent with the iterative decoding thresholds of the underlying protographs.

The next property of LDPC convolutional codes studied is the so-called trapping sets. Trapping sets are graphical sub-structures that exist in the Tanner graph of the diagonal parity-check matrix. As discussed in Chapter 4, small trapping sets have been shown to have significant effect on the error floor of modern capacity-approaching codes under iterative message-passing decoding. In Chapter 6, asymptotic methods were used to calculate a lower bound for the Δ-trapping set number that grows linearly with constraint length for several regular and one irregular asymptotically good ensembles of unterminated, protograph-based, time-varying LDPC convolutional codes. This shows that the size of the smallest non-empty trapping set (which dominates performance in the error floor region) grows linearly with constraint length for these codes. Finally, it was shown that the Δ-trapping set growth rates of the LDPC convolutional code ensembles exceed the growth rates of the corresponding LDPC block code ensembles on which they are based. These large trapping set growth rates suggest that the LDPC convolutional codes will exhibit good iterative decoding performance in the error floor.

7.2 Recommendations for future research

Following the summary of the new contributions of the thesis, this section details a selection of the worthwhile future research directions to continue this body of work.

7.2.1 Part I: Efficient encoding of block codes using polynomials

- **Selection of polynomials for code design**. It was mentioned that the choice of polynomials dictates the length and rate of the code. In addition, the density of the matrices is also directly related to the choice of polynomials. In particular, it would be of interest to use this polynomial framework to design LDPC parity-check matrices with sparse parity-check and generator matrices. In general, sparsity of the generator matrix is not automatic when using traditional Gaussian elimination techniques, and this property is desirable when considering encoder complexity.

- **Exploration of quasi-cyclic type structures**. We observed that defective quasi-cyclic structures emerge from the non-binary version of the method after translating back to binary symbols. Quasi-cyclic LDPC codes have been shown in the literature to have excellent error control performance, thus it would be of interest to compare the performance of defective quasi-cyclic LDPC codes, especially as they share the efficient shift register encoding.

7.2.2 Part II: Asymptotically good LDPC convolutional codes based on protographs

- **Unwrapping procedures of protographs with gcd$(\mathbf{n_c}, \mathbf{n_v}) = \mathbf{1}$.** Under this condition, two new methods were proposed in Chapter 5 to unwrap the protograph parity-check matrix in order to obtain the best possible lower bound for the free distance growth rate δ_{free}. The results suggest that we typically obtain better lower bounds by performing nonuniform cuts. It would be of interest to develop the theory regarding these methods to see if there is an optimal nonuniform cut, or how to choose the best member of the mini-ensemble.

- **The study of elementary trapping sets**. An obvious extension of the theory and results of Chapter 6 would be to consider the elementary trapping sets of protograph-based LDPC convolutional codes. This is not a trivial extension, requiring a slightly different model and associated theory. As discussed in Section 6.2.2, it is expected that the results of this analysis should be approximately the same as those for general trapping sets, because the smallest trapping sets tend to be of elementary type.

Appendix A

Derivations of Protograph Enumeration Results

A.1 A finite check node weight enumerator for a degree two check node

Proposition A.1 *Suppose we have a check node of degree* 2 *with input weight vector* $\mathbf{w} = (w_0, w_1)$ *and we take* N *copies of the protograph. Thus* $0 \leq w_0, w_1 \leq N$*. Then to satisfy each of the checks we must have* $w_0 = w_1$ *and*

$$A_{w_0,w_1} = \binom{N}{w_0}.$$

Proof. Suppose without loss of generality that $w_0 > w_1 \geq 0$. Each check node will have one connection to a variable node of type w_0 and one connection to a variable node of type w_1.

1. If $w_1 = 0$, then we cannot satisfy any of the check nodes with input weight one from a variable node of type w_0 (and by definition there must be at least one such node).

2. If $w_1 > 0$, then we can satisfy one of the N check nodes immediately. This leaves $N-1$ remaining check nodes to be satisfied with weights $w_0 - 1$ and $w_1 - 1$ to be distributed over variable nodes of type w_0 and w_1 respectively, where $w_0 - 1 > w_1 - 1 \geq 0$. We repeat this procedure and eventually arrive the situation where $w_0 - p > w_1 - p = 0$ for some $1 \leq p \leq N - 1$, hence we cannot satisfy all of the check nodes because of statement 1.

The result now follows that if $w_0 = w_1$ we can satisfy w_0 distinct check nodes from a choice of N. □

A.2 Asymptotic weight enumerators

Recall from Section 4.3.1 that an expression for the number of sequences that satisfy a degree three check node is given as follows [Div06]:

$$A_{w_0,w_1,w_2} = \binom{N}{s} \frac{s!}{(s-w_0)!(s-w_1)!(s-w_2)!},$$

where $s = \frac{w_0+w_1+w_2}{2}$, $\max\{w_0, w_1, w_2\} \leq s \leq N$ and $w_0 + w_1 + w_2$ is even; otherwise $A_{w_0,w_1,w_2} = 0$. We define the asymptotic check node weight enumerator for a degree 3 check node to be

$$a(\epsilon_0, \epsilon_1, \epsilon_2) \triangleq \limsup_{N\to\infty} \frac{\ln(A_{w_0,w_1,w_2})}{N}, \tag{A.1}$$

where $\epsilon_i = w_i/N$.

Define $\sigma = (\epsilon_1 + \epsilon_2 + \epsilon_3)/2$ such that $\max\{\epsilon_1, \epsilon_2, \epsilon_3\} \leq \sigma \leq 1$. We now derive the expression stated in [Div06] for equation (A.1):

$$\begin{aligned}
& \limsup_{N\to\infty} \frac{\ln(A_{w_0,w_1,w_2})}{N} = \limsup_{N\to\infty} \frac{1}{N} \ln\left(\binom{N}{s} \frac{s!}{(s-w_0)!(s-w_1)!(s-w_2)!}\right) \\
= \; & \limsup_{N\to\infty} \frac{1}{N} \ln\left(\frac{N!}{(N-s)!(s-w_0)!(s-w_1)!(s-w_2)!}\right) \\
= \; & \limsup_{N\to\infty} \frac{1}{N} \left(\ln N! - \ln(N-s)! - \ln(s-w_0)! - \ln(s-w_1)! - \ln(s-w_2)!\right) \\
\approx^{1} \; & \limsup_{N\to\infty} \frac{1}{N} \left(N\ln N - N - (N-s)\ln(N-s) + (N-s) - (s-w_0)\ln(s-w_0)\right. \\
& \left. +(s-w_0) - (s-w_1)\ln(s-w_1) + (s-w_1) - (s-w_2)\ln(s-w_2) + (s-w_2)\right) \\
= \; & \limsup_{N\to\infty} \left(\ln N - (1-\sigma)\ln(N(1-\sigma)) - (\sigma-\epsilon_1)\ln(N(\sigma-\epsilon_1))\right. \\
& \left. - (\sigma-\epsilon_2)\ln(N(\sigma-\epsilon_2)) - (\sigma-\epsilon_3)\ln(N(\sigma-\epsilon_3))\right) \\
= \; & -(1-\sigma)\ln(1-\sigma) - (\sigma-\epsilon_1)\ln(\sigma-\epsilon_1) - (\sigma-\epsilon_2)\ln(\sigma-\epsilon_2) \\
& -(\sigma-\epsilon_3)\ln(\sigma-\epsilon_3) \\
= \; & -\left(1 - \sum_{i=1}^{3}(\sigma-\epsilon_i)\right)\ln\left(1 - \sum_{i=1}^{3}(\sigma-\epsilon_i)\right) - \sum_{i=1}^{3}(\sigma-\epsilon_i)\ln(\sigma-\epsilon_i)
\end{aligned} \tag{A.2}$$

[1] Using Stirling's approximation $\ln(n!) \approx n\ln(n) - n$. Note that better approximations exist but as n becomes very large the additional terms tend to zero.

Using the results of Section A.1, an asymptotic check node weight enumerator for a degree two check node can be written as follows:

$$a(\epsilon_0, \epsilon_1) \triangleq \limsup_{N\to\infty} \frac{\ln(A_{w_0,w_1})}{N}, \tag{A.3}$$

where $\epsilon_i = w_i/N$. An expression for (A.3) can be derived explicitly as above for a check node of degree three. Alternatively, we can use the asymptotic degree three check node weight enumerator (A.2): if $\epsilon_0 \neq \epsilon_1$ then $a(\epsilon_0, \epsilon_1) = 0$, otherwise

$$a(\epsilon_0, \epsilon_1) = a(\epsilon_0, \epsilon_0, 0) = -(1-\epsilon_0)\ln(1-\epsilon_0) - (\epsilon_0)\ln(\epsilon_0) = H(\epsilon_0),$$

where $H(x) = -(1-x)\ln(1-x) - x\ln x$ is the entropy function.

Mirroring the case for finite check node weight enumerators, expressions for the asymptotic weight enumerators for check nodes of degree larger than three can be obtained by concatenation of the expression derived here for a check node of degree three. For further details, see [Div06].

References

[ADY07] A. Abbasfar, D Divsalar, and K. Yao, *Accumulate repeat accumulate codes*, IEEE Transactions on Communications **55** (2007), no. 4, 692–702.

[All91] R. B. J. T. Allenby, *Rings, fields and groups: An introduction to abstract algebra*, Butterworth-Heinemann, 1991.

[All97] ———, *Linear algebra*, Butterworth-Heinemann, 1997.

[ARD07a] S. Abu-Surra, W. E. Ryan, and D Divsalar, *Ensemble trapping set enumerators for protograph-based LDPC codes*, Proc. Forty-fifth Annual Allerton Conference (Allerton, IL), Sept. 2007.

[ARD07b] ———, *Ensemble weight enumerators for protograph-based generalized LDPC codes*, Proc. Information Theory and Applications Workshop (San Diego, CA), Jan. 2007.

[BBF+06] A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert, and A. Wassermann, *Error-correcting linear codes: Classification by isometry and applications*, Springer, 2006.

[BCJR74] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, *Optimal decoding of linear codes for minimizing symbol error rate*, IEEE Transactions on Information Theory **20** (1974), no. 2, 284–287.

[Ber68] E. R. Berlekamp, *Algebraic coding theory*, McGraw-Hill, Inc., 1968.

[Ber71] T. Berger, *Rate distortion theory*, Prentice Hall, Englewood Cliffs, NJ, 1971.

[Ber82] E. R. Berlekamp, *Bit-serial Reed-Solomon encoders*, IEEE Transactions on Information Theory **28** (1982), no. 6, 869–874.

[BG96] C. Berrou and A. Glavieux, *Near optimum error correcting coding and decoding: turbo codes*, IEEE Transactions on Communications **44** (1996), no. 10, 1261–1271.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima, *Near Shannon limit error-correcting coding and decoding: turbo-codes*, Proc. IEEE International Conference on Communications (Geneva, Switzerland), May 1993.

[BR60] R. C. Bose and C. R. Ray-Chaudhuri, *On a class of error correcting binary group codes*, Information and Control **3** (1960), 68–79.

[Bre04] M. Breiling, *A logarithmic upper bound on the minimum distance of turbo codes*, IEEE Transactions on Information Theory **50** (2004), no. 8, 1692–710.

[CF07] D. J. Costello, Jr. and G. D. Forney, Jr., *Channel coding: the road to channel capacity*, Proceedings of the IEEE **95** (2007), no. 6, 1150–1177.

[CFRU01] S. Y. Chung, G. D. Forney, Jr., T. Richardson, and R. L. Urbanke, *On the design of low density parity check codes within 0.0045 db of the Shannon limit*, IEEE Communications Letters **5** (2001), 58–60.

[Chu00] S. Y. Chung, *On the construction of some capacity-approaching coding schemes*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, Sept. 2000.

[Cos74] D. J. Costello, Jr., *Free distance bounds for convolutional codes*, IEEE Transactions on Information Theory **20** (1974), no. 3, 356–365.

[CPBZ06] D. J. Costello, Jr., A. E. Pusane, S. Bates, and K. Sh. Zigangirov, *A comparison between LDPC block and convolutional codes*, Proc. Information Theory and Applications Workshop (CA, USA), Feb. 2006.

[CPJD07] D. J. Costello, Jr., A. E. Pusane, C. R. Jones, and D. Divsalar, *A comparison of ARA- and protograph-based LDPC block and convolutional codes*, Proc. Information Theory and Applications Workshop (CA, USA), Feb. 2007.

[CT91] T. Cover and J. Thomas, *Elements of information theory*, Wiley Inter-Science, New York, 1991.

[CXDL04] L. Chen, J. Xu, I. Djurdjevic, and S. Lin, *Near-Shannon-limit quasi-cyclic low-density parity-check codes*, IEEE Transactions on Communications **52** (2004), no. 7, 1038–1042.

[DDJ06] D. Divsalar, S. Dolinar, and C. Jones, *Construction of protograph LDPC codes with linear minimum distance*, Proc. IEEE International Symposium on Information Theory (Seattle, WA), July 2006.

[Div06] D. Divsalar, *Ensemble weight enumerators for protograph LDPC codes*, Proc. IEEE International Symposium on Information Theory (Seattle, WA), July 2006.

[DJDT05] D Divsalar, C. Jones, S. Dolinar, and J. Thorpe, *Protograph based LDPC codes with minimum distance linearly growing with block size*, Proc. IEEE Global Telecommunications Conference (St. Louis, MO), Dec. 2005.

[DJM98] D Divsalar, H. Jin, and R. McEliece, *Coding theorems for turbo-like codes*, Proc. Thirty-sixth Annual Allerton Conference (Allerton, IL), Sept. 1998.

[DLZ+09] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolić, and M. Wainwright, *Predicting error floors of structured LDPC codes: Deterministic bounds and estimates*, IEEE Journal on Selected Areas in Communications **27** (2009), no. 6, 908–917.

[DM98] M. C. Davey and D. J. C. MacKay, *Low-density parity check codes over GF(q)*, IEEE Communications Letters **2** (1998), no. 6, 165–167.

[DPT+02] 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 Transactions on Information Theory **48** (2002), no. 6, 1570–1579.

[EGSM98] M. Esmaeili, T. A. Gulliver, N. P. Secord, and S. A. Mahmoud, *A link between quasi-cyclic codes and convolutional codes*, IEEE Transactions on Information Theory **44** (1998), no. 1, 431–435.

[Eli55] P. Elias, *Coding for noisy channels*, IRE International Convention Record (1955), no. 4, 37–46.

[FMT05] S. L. Fogal, R. McEliece, and J. Thorpe, *Enumerators for protograph ensembles of LDPC codes*, Proc. IEEE International Symposium on Information Theory (Adelaide, Australia), Sept. 2005.

[For66] G. D. Forney, Jr., *Concatenated codes*, MIT Press, Cambridge, MA, 1966.

[For71] ———, *Burst-correcting codes for the classic bursty channel*, IEEE Transactions on Communications Technology **19** (1971), no. 5, 772–781.

[Gal62] R. G. Gallager, *Low-density parity-check codes*, IRE Transactions on Information Theory **8** (1962), no. 1, 21–28.

[Gal63] ———, *Low-density parity-check codes*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1963.

[Gar04] P. B. Garrett, *The mathematics of coding theory : Information, compression, error correction, and finite fields*, Pearson/Prentice Hall, 2004.

[Gil52] E. N. Gilbert, *A comparison of signalling alphabets*, Bell System Technical Journal **31** (1952), 504–522.

[Goe07] N. Goertz, *Joint source-channel coding of discrete-time signals with continuous amplitudes*, Imperial College Press/World Scientific, London, UK, 2007.

[Gol49] M. J. E. Golay, *Notes on digital coding*, Proceedings of the IEEE **37** (1949), 657.

[Gra90] R. Gray, *Source coding theory*, Kluwer Academic, Boston, MA, 1990.

[Ham50] R. W. Hamming, *Error detecting and error correcting codes*, Bell System Technical Journal **29** (1950), no. 2, 147–160.

[Hoc59] A. Hocquenghem, *Codes correcteurs d'erreurs*, Chiffres **2** (1959), 147–156.

[Imm94] K. A. S. Immink, *Reed-Solomon codes and the compact disc*, in "Reed-Solomon codes and their applications", ch. 4, pp. 41–59, IEEE Press, Piscataway, NJ, 1994.

[JKM00] H. Jin, A. Khandekar, and R. McEliece, *Irregular repeat-accumulate codes*, Proc. International Symposium on Turbo Codes and Related Topics (Brest, France), Sept. 2000.

[Jus72] J. Justesen, *A class of constructive asymptotically good algebraic codes*, IEEE Transactions on Information Theory **18** (1972), no. 5, 652–656.

[JZ99a] A. Jiménez-Felström and K. Sh. Zigangirov, *Time-varying periodic convolutional codes with low-density parity-check matrices*, IEEE Transactions on Information Theory **45** (1999), no. 6, 2181–2191.

[JZ99b] R. Johannesson and K. Sh. Zigangirov, *Fundamentals of convolutional coding*, IEEE Press, Piscataway, NJ, 1999.

[Ker72] A. M. Kerdock, *A class of low-rate non-linear binary codes*, Information and Control **20** (1972), 182–187.

[KFL01] F. R. Kschischang, B. J. Frey, and H. Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Transactions on Information Theory **47** (2001), no. 2, 498–519.

[KLF00] Y. Kou, S. Lin, and M. P. C. Fossorier, *Construction of low density parity check codes: a geometric approach*, Proc. International Symposium on Turbo Codes and Related Topics (Brest, France), Sept. 2000.

[KLF01] ———, *Low-density parity-check codes based on finite geometries: a rediscovery and new results*, IEEE Transactions on Information Theory **47** (2001), no. 7, 2711–2736.

[KLP68] T. Kasami, S. Lin, and W. W. Peterson, *Polynomial codes*, IEEE Transactions on Information Theory **14** (1968), no. 6, 807–814.

[LC04] S. Lin and D. J. Costello, Jr., *Error control coding: Fundamentals and applications*, second ed., Prentice Hall, Englewood Cliffs, NJ, 2004.

[LCZ+06] Z. Li, L. Chen, L. Zeng, S. Lin, and Fong. W. H., *Efficient encoding of quasi-cyclic low-density parity-check codes*, IEEE Transactions on Communications **54** (2006), no. 1, 71–81.

[LMSS01] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, *Improved low-density parity-check codes using irregular graphs*, IEEE Transactions on Information Theory **47** (2001), no. 2, 585–598.

[LS02] S. Litsyn and V. Shevelev, *On ensembles of low-density parity-check codes: asymptotic distance distributions*, IEEE Transactions on Information Theory **48** (2002), no. 4, 887–908.

[Mac98] D. J. C. MacKay, *Gallager codes that are better than turbo codes*, Proc. Thirty-sixth Allerton Conference on Communication, Control, and Computing (Allerton, IL), Sept. 1998.

[Mac99] ———, *Good error-correcting codes based on very sparse matrices*, IEEE Transactions on Information Theory **45** (1999), no. 2, 399–431.

[Mas69] J. Massey, *Shift-register synthesis and BCH decoding*, IEEE Transactions on Information Theory **15** (1969), no. 1, 122–127.

[Meg61] J. E. Meggitt, *Error correcting codes and their implementation for data transmission systems*, IRE Transactions on Information Theory **7** (1961), no. 4, 234–244.

[MN97] D. J. C. MacKay and R. M. Neal, *Near Shannon limit performance of low density parity check codes*, Electronics Letters **33** (1997), no. 6, 457–458.

[MOG07] D. G. M. Mitchell, L. O'Carroll, and N. Goertz, *A new method of encoding block codes with polynomials*, Proc. International Symposium on Communications Theory and Applications (Ambleside, England), July 2007.

[MOG08] ———, *Towards efficient encoding using polynomials*, Proc. International ITG Conference on Source and Channel Coding (Ulm, Germany), Jan. 2008.

[MP03] D. J. C. MacKay and M. S. Postol, *Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes*, Electronic Notes in Theoretical Computer Science **74** (2003), 97–104.

[MPC09a] D. G. M. Mitchell, A. E. Pusane, and D. J. Costello, Jr., *Asymptotic trapping set analysis of regular protograph-based LDPC convolutional codes*, Proc. Information Theory and Applications Workshop (San Diego, CA), Feb. 2009.

[MPC09b] ———, *Trapping set analysis of protograph-based LDPC convolutional codes*, Proc. IEEE International Symposium on Information Theory (Seoul, Korea), July 2009.

[MPGC08] D. G. M. Mitchell, A. E. Pusane, N. Goertz, and D. J. Costello, Jr., *Free distance bounds for protograph-based regular LDPC convolutional codes*, Proc. International Symposium on Turbo Codes and Related Topics (Lausanne, Switzerland), Aug. 2008.

[MPZC08] D. G. M. Mitchell, A. E. Pusane, K. Sh. Zigangirov, and D. J. Costello, Jr., *Asymptotically good LDPC convolutional codes based on protographs*, Proc. IEEE International Symposium on Information Theory (Toronto, Canada), July 2008.

[MS77] F. J. Macwilliams and N. J. A. Sloane, *The theory of error-correcting codes*, Elsevier Science Publishers, 1977.

[MS94] R. McEliece and L. Swanson, *Reed-Solomon codes and the exploration of the solar system*, in "Reed-Solomon codes and their applications", ch. 3, pp. 25–40, IEEE Press, Piscataway, NJ, 1994.

[MSW07] O. Milenkovic, E. Soljanin, and P. Whiting, *Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles*, IEEE Transactions on Information Theory **53** (2007), no. 1, 39–55.

[Mul54] D. E. Muller, *Applications of boolean algebra to switching circuits design and to error correction*, IRE Transactions on Computers **3** (1954), 6–12.

[Pea82] J. Pearl, *Reverend Bayes on inference engines: a distributed hierarchical approach*, Proc. AAAI Conference on Artificial Intelligence (Pittsburgh, PA), Aug. 1982.

[PJS+09] A. E. Pusane, A. Jiménez-Feltström, A. Sridharan, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., *Implementation aspects of LDPC convolutional codes*, IEEE Transactions on Communications **56** (2009), no. 7, 1060–1069.

[Ple98] Vera Pless, *Introduction to the theory of error-correcting codes*, third ed., John Wiley & Sons, Inc., 1998.

[Pra57] E. Prange, *Cyclic error-correcting codes in two symbols*, Tech. Report AFCRC-TN-57-103, Air Force Cambridge Research Centre, Cambridge, MA, Sept. 1957.

[Pre68] F. P. Preparata, *A class of optimum nonlinear double-error-correcting codes*, Information and Control **13** (1968), 378–400.

[Pro01] J. G. Proakis, *Digital communications*, fourth ed., McGraw-Hill, New York, NY, 2001.

[PSVC07] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, Jr., *On deriving good LDPC convolutional codes from QC LDPC block codes*, Proc. IEEE International Symposium on Information Theory (Nice, France), June 2007.

[PZC06] A. E. Pusane, K. Sh. Zigangirov, and D. J. Costello, Jr., *Construction of irregular LDPC convolutional codes with fast decoding*, Proc. IEEE International Conference on Communications (Istanbul, Turkey), June 2006.

[Ree54] I. S. Reed, *A class of multiple-error-correcting codes and the decoding scheme*, IRE Transactions on Information Theory **4** (1954), 38–49.

[Ric03] T. J. Richardson, *Error-floors of LDPC codes*, Proc. Forty-first Annual Allerton Conference (Allerton, IL), Sept. 2003.

[RKZ06] G. Richter, M. Kaupper, and K. Sh. Zigangirov, *Irregular low-density parity-check convolutional codes based on protographs*, Proc. IEEE International Symposium on Information Theory (Seattle, WA), July 2006.

[RN03] T. J. Richardson and V. Novichkov, *Methods and apparatus for decoding LDPC codes*, United States Patent 6633856, October 2003.

[RS60] I. S. Reed and G. Solomon, *Polynomial codes over certain finite fields*, Journal of the Society for Industrial and Applied Mathematics **8** (1960), no. 2, 300–304.

[RSU01] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, *Design of capacity approaching irregular low-density parity-check codes*, IEEE Transactions on Information Theory **47** (2001), no. 2, 619–637.

[RU01a] T. J. Richardson and R. L. Urbanke, *The capacity of low-density parity-check codes under message-passing decoding*, IEEE Transactions on Information Theory **47** (2001), no. 2, 599–618.

[RU01b] ———, *Efficient encoding of low-density parity-check codes*, IEEE Transactions on Information Theory **47** (2001), no. 2, 638–656.

[RU02] ———, *Multi-edge type LDPC codes*, Workshop honouring Prof. Bob McEliece on his 60th birthday (Pasadena, CA), May 2002.

[RU03] ———, *The renaissance of Gallager's low-density parity-check codes*, IEEE Communications Magazine (Aug. 2003), 126–131.

[SB54] R. A. Silverman and M Balser, *Coding for constant-data-rate systems*, Transactions of the IRE Professional Group on Information Theory **4** (1954), no. 4, 50–63.

[Sha48] C. E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27** (1948), 379–423 623–656.

[Spi96] D. A. Spielman, *Linear-time encodable and decodable error-correcting codes*, IEEE Transactions on Information Theory **42** (1996), no. 6, 1723–1731.

[SPVC09] R. Smarandache, A. E. Pusane, P. O. Vontobel, and D. J. Costello, Jr., *Pseudo-codeword analysis for LDPC convolutional codes*, IEEE Transactions on Information Theory **55** (2009), no. 6, 2577–2598.

[SS96] M. Sipser and D. A. Spielman, *Expander codes*, IEEE Transactions on Information Theory **42** (1996), no. 6, 1710–1722.

[SSCF03] A. Sridharan, D. Sridhara, D. J. Costello, Jr., and T. E. Fuja, *A construction for irregular low density parity check codes*, Proc. IEEE International Symposium on Information Theory (Yokohama, Japan), June 2003.

[ST79] G. Solomon and H. C. A. Tilborg, *A connection between block and convolutional codes*, SIAM Journal on Applied Mathematics **37** (1979), no. 2, 358–369.

[STL+07] A. Sridharan, D. Truhachev, M. Lentmaier, D. J. Costello, Jr., and K. Sh. Zigangirov, *Distance bounds for an ensemble of LDPC convolutional codes*, IEEE Transactions on Information Theory **53** (2007), no. 12, 4537–4555.

[SZLA06] S. Song, L. Zeng, S. Lin, and K. A. S. Abdel-Ghaffar, *Algebraic constructions of nonbinary quasi-cyclic LDPC codes*, IEEE International Symposium on Information Theory (Seattle, WA), July 2006.

[Tan81a] R. M. Tanner, *Error-correcting coding system*, United States Patent 4295218, 1981.

[Tan81b] ———, *A recursive approach to low complexity codes*, IEEE Transactions on Information Theory **27** (Sept. 1981), no. 5, 533–547.

[Tan87] ———, *Convolutional codes from quasi-cyclic codes: A link between the theories of block and convolutional codes*, Tech. report, University of California Santa Cruz, 1987.

[Tho03] J. Thorpe, *Low-density parity-check (LDPC) codes constructed from protographs*, INP Progress Report 42-154, Jet Propulsion Laboratory, Pasadena, CA, Aug. 2003.

[TSS+04] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, Jr., *LDPC block and convolutional codes based on circulant matrices*, IEEE Transactions on Information Theory **50** (2004), no. 12, 2966–2984.

[TXLA05] H. Tang, J. Xu, S. Lin, and K. A. S. Abdel-Ghaffar, *Codes on finite geometries*, IEEE Transactions on Information Theory **51** (2005), no. 7, 572–596.

[TZC09] D. Truhachev, K. Sh. Zigangirov, and D. J. Costello, Jr., *Distance bounds for periodically time-varying and tail-biting LDPC convolutional codes*, IEEE Transactions on Information Theory (2009), (submitted). Available online at `http://luur.lub.lu.se/luur?func=downloadFile&fileOId=1054996`.

[TZF07] M. Tavares, K. Sh. Zigangirov, and G. Fettweis, *Tail-biting LDPC convolutional codes*, Proc. IEEE International Symposium on Information Theory (Nice, France), June 2007.

[Var57] R. R. Varshamov, *Estimate of the number of signals in error correcting codes*, Doklady Akademii Nauk SSSR **117** (1957), no. 5, 739–741.

[Vit67] A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Transactions on Information Theory **13** (1967), no. 2, 260–269.

[WB94] S. B. Wicker and V. K. Bhargava (eds.), *Reed-Solomon codes and their applications*, IEEE Press, 1994.

[Wib96] N. Wiberg, *Codes and decoding on general graphs*, Ph.D. thesis, Linköping University, Linköping, Sweden, 1996.

[ZDN+06] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, and M. J. Wainwright, *Investigation of error floors of structured low-density parity-check codes by hardware emulation*, Proc. IEEE Global Telecommunications Conference (San Francisco, CA), Nov. 2006.