Analysis and Design of Low-Density Parity-Check Convolutional Codes

Author: **Hua Zhou**

Matriculation number: **0828809**

Institute of Telecommunications

Faculty of Electrical Engineering and Information Technology

Vienna University of Technology

Supervisor: **Norbert Görtz**

A dissertation submitted for the degree of

Doctor of Technical Sciences
(*Doktor der technischen Wissenschaften*)

February 25, 2013

institute of telecommunications

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Acknowledgements

I would like to begin my acknowledgements with my deepest gratitude to my advisor Prof. Norbert Görtz for his patient guidance during my research and study. Throughout my doctoral work he encouraged me to develop independent thinking and skills of writing academic papers and giving technical presentations. Moreover, he was always ready to help me with any difficulty in my life during my stay in Vienna. Without his consistent supervision and support, I could not have come to the stage of writing up my dissertation. I am grateful for his knowledge and trust to take me as a student doing research. Also, I thank Vienna University of Technology for sponsoring me as a university assistant.

I spent one semester visiting the University of Notre Dame, I gained plenty of ideas from the discussion with Prof. Daniel J. Costello, Jr., his broad knowledge and deep understanding of coding theory inspired my endless interest to explore the area of coding theory. Dr. David Mitchell, another one I would like to thank for being a great research partner. I learned many things from his religious attitude towards science and we got many fruitful ideas from the discussion. In addition, I thank Prof. Daniel J. Costello, Jr. and the CRC computer center for offering me extra two-year PC facilities for simulations after my leaving.

I also thank Dr. Roxana Smarandache, Dr. Michael Lentmaier, and Dr. Ali E. Pusane for their time and attention put into discussions.

I would like to thank my friends met at Notre Dame for their friendliness, hospitalities and nicely taking care of me: Lai Wei, Zhanwei Sun, Christian Koller, Xincheng Zhang, Jun Chen, and the couple Keith and Susan. Without them, my stay at Notre Dame could have been difficult. Especially, I thank Keith and Susan for their kindness of offering me a bedroom and driving me to the supermarket.

To my colleagues at the Multimedia System lab of Vienna University of Technology: Gerhard Doblinger, Geetha Ramachandran, Johannes Gonter,

Wolfgang Gartner, Mehdi Mortazawi Molu, and Pratana Kukieattikool, I was very lucky to work with them and I will miss every funny moment during the coffee break. And I also would like to thank my colleagues at the institute that I have not mentioned already, including Qi Wang, Mingming Gan, Zhinan Xu, Josep Colom-Ikuno, Michal Šimko, Georg Kail, and many others.

"Prosperity makes friends, adversity tries them". To my best friends met in Vienna: Junfeng Liu, Xiaoran Zhang, Kongzhao Li, Boyu Zhang, Huichao Sun, Lina Liu, Zhiquan Li, Xiaoyan Meng, Congcong Yang, Hui Xue, and Minjian Zhou. It is my great fortune to have you being my friends. I will miss the memory of our interesting chatting, the group traveling, and the expectation of the future. Vienna leaves me many unforgettable images, and you are the scene of the images. Hope you all have a well and prospering future.

Last but not least, I would like to thank my wife Yanan and my parents who have always been the most important part of my life. Their throughout support and encouragement have always been the strength of my life.

Declaration of originality

I declare that this dissertation is my original work except where stated. This dissertation has never been submitted for any degree or examination to any other University in the world.

Signature: Hua Zhou

Date: February 25, 2013

Abstract

Low-density parity-check block codes (LDPC-BCs) were first discovered by Gallager in the early 1960s. As counterparts, LDPC convolutional codes (LDPC-CCs) were first proposed in 1999 [1]. They can be considered as conventional convolutional codes with fairly large memory. Using pipeline decoding [2], it has been shown that they are suitable for practical implementation with continuous transmission as well as, via encoder termination, for block transmission in frames of arbitrary size [3] without an increase in computational complexity compared to their block code counterparts. Time-invariant or time-varying LDPC-CCs can be defined by polynomial-domain syndrome former matrices $\mathbf{H}^T(D)$ (transposed parity-check matrices in polynomial form), that can be derived from corresponding Quasi-Cyclic (QC) LDPC-BCs or unwrapping LDPC-BCs.

LDPC-CCs can be efficiently decoded by a pipelined sub-optimal Sum Product Algorithm (SPA). It suffers, however, from convergence problems, due to cycles in the Tanner graph. To improve the decoding performance, we analyze the cycle properties, based on the connections between monomials in the polynomial-domain syndrome former matrix. According to the relationship of cycle structures between time- and polynomial-domain syndrome former matrices, we present a cycle counter algorithm to enumerate cycles in time-*invariant* and time-*varying* LDPC-CCs. Due to some specific structures in the weight matrix of a polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$, some cycles are *un*avoidable no matter what monomials are placed in the weight matrix. It is shown that large-weight entries in $\mathbf{H}^T(D)$ lead to small girth, while monomial or empty entries, which can break short unavoidable cycles, may result in large girth.

Another important characteristic of LDPC-CCs is the *distance spectrum*. However, it is complicated to compute the precise distance spectrum of LDPC-CCs by, for example, the BEAST algorithm due to the large memory of LDPC-CCs. An estimation of the distance spectrum of time-invariant

LDPC-CCs is obtained by splitting the polynomial-domain syndrome former matrix into submatrices representing "super codes" and then evaluating the linear dependence between codewords of the corresponding super codes. This estimation results in an upper bound on the minimum *free distance* of the original code and, additionally, a lower bound on the number of codewords A_w with Hamming weight w.

To adapt to the changing conditions of time-varying channels and to allow transceivers to employ the same encoder/decoder pair, a family of robust rate-compatible (RC) punctured low-density parity-check convolutional codes (LDPC-CCs) is derived from a time-invariant LDPC-CC mother code by periodically puncturing encoded bits (variable nodes) with respect to several criteria: (1) ensuring the recoverability of punctured variable nodes, (2) minimizing the number of completely punctured cycle trapping sets (CPCTSs), and (3) minimizing the number of punctured variable nodes involved in short cycles. The influence of (1) and (3) on iterative decoding performance is felt most strongly in the waterfall region of the bit-error-rate (BER) curve, while (2) has a larger effect in the error floor, or high signal-to-noise ratio (SNR), region. We show that the length of the puncturing period is an important parameter when designing high-rate punctured codes and, moreover, that extending the puncturing period can improve the decoding performance and extend the range of compatible rates. As examples, we obtain families of RC LDPC-CCs from several time-invariant LDPC-CC mother codes with monomial and binomial entries in their polynomial syndrome former matrices.

Contents

List of Figures

List of Symbols

Frequently Used Symbols

$\bar{v}_s$	constraint length
$\tau_{d,f}$	a (d, f) trapping set
$\mathbf{H}^T$	time-domain syndrome former matrix
$\mathbf{H}^T(D)$	polynomial-domain syndrome former matrix
$\mathbf{a}$	puncturing pattern
c_j	check node corresponds the j-th column of $\mathbf{H}^T$ for LDPC-BCs
c_t^k	variable node corresponds the $tq + k$-th column of $\mathbf{H}^T$ for LDPC-CCs
m_s	syndrome former memory
R	code rate
R'	punctured code rate
$r_w(\mathbf{t})$	a cycle of length w
s_{ij}^p	the power of the p-th monomial of the polynomial entry in the i-th row and j-th column of $\mathbf{H}^T(D)$ for time-invariant LDPC-CCs
$s_{ij}^p(t)$	the power of the p-th monomial of the polynomial entry in the i-th row and j-th column of the t-th polynomial submatrix $\mathbf{H}_t^T(D)$ for time-varying LDPC-CCs
v_i	variable node corresponds the i-th row of $\mathbf{H}^T$ for LDPC-BCs
v_t^j	variable node corresponds the $tc + j$-th row of $\mathbf{H}^T$ for LDPC-CCs
$\mathbf{B}$	weight matrix

Acronyms

AWGN additive white Gaussian noise

BER bit-error-rate

BPSK binary phase shift keyed

∞-SR an unrecoverable punctured variable node

m-SR a punctured variable node recovered in the m-th decoding iteration

BEC binary erasure channel

BIOSM binary-input output-symmetric memoryless

BP belief propagation

CPCTSs completely punctured cycle trapping sets

FEC forward error correction

FIFO first-in first-out

GF Galois field

LDPC-BCs low-density parity-check block codes

LDPC-CCs low-density parity-check convolutional codes

LLRs Log-likelihood ratios

MAP maximum a posteriori probability

ML maximum likelihood

PCCC parallel concatenated convolutional code

QC quasi-cyclic

RF radio-frequency

SNR signal-to-noise ratio

SPA sum-product algorithm

RC rate-compatible

Chapter 1

Introduction

1.1 Reliable data transmission system

Data transmission or digital transmission is the physical transfer of data (a digital bit stream) over a point-to-point or point-to-multipoint communication channel. Examples of such channels are copper wires, optical fibers, wireless communication channels, and storage media. In recent years, there has been an increasing demand for efficient and reliable digital data transmission and storage as a result of the emergence of large-scale, high-speed data networks for exchange, processing, and storage of digital information. Error-free data transmission and storage are difficult to achieve for practical communications because of the noise and interference in the channel. Then the concern is how to lower the error rate as much as possible so that the reliability of communications is acceptable. According to Shannon's theory, by proper coding applied to the data, errors induced by a noisy channel or storage medium can be reduced to any desired level, as long as the information rate is less than the capacity of the channel. Since then, a lot of effort went into the problem of devising good coding schemes approaching Shannon limit.

Modern digital communication design requires a pair of transmitter and receiver that spans a radio link. The data information is transferred from the transmitter to the receiver. A simple and typical transmission system is illustrated by the block diagram shown in Fig. 1.1. The transmitter consists of the information source, source encoder, channel encoder, and modulator. And the receiver is made up of a demodulator, channel decoder, source decoder, and the destination.

In the transmitter, the *information source* is encoded by the *source encoder* into a sequence of binary digits (bits) $\mathbf{u}$ called *information sequence* which is further encoded

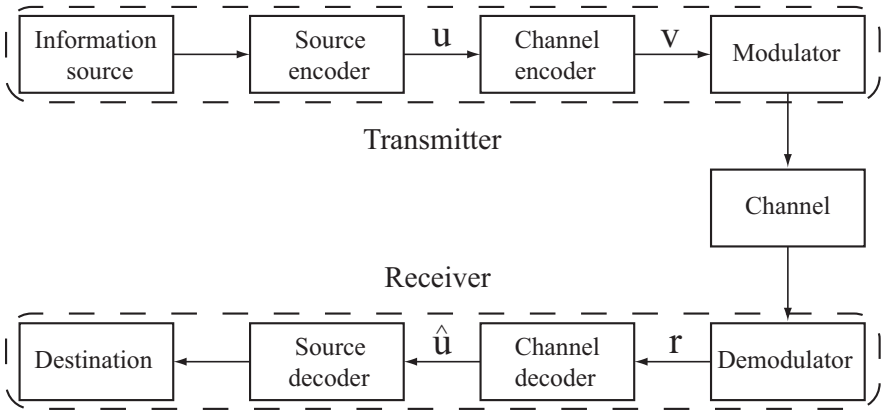


Figure 1.1: Block diagram of a simple and typical data transmission system [4].

by the *channel encoder* into a discrete sequence $\mathbf{v}$ called a *codeword* by adding redundancy bits. Channel encoding makes the data information described by the codewords robust against noise interference. Discrete symbols, for example a digital bit stream (codewords) or an analog audio signal, are not suitable for transmission over a physical channel or recording on a digital storage medium. Hence in the *modulator*, each symbol is modulated into a waveform. It conveys a message signal inside another signal that can be physically transmitted. For example, modulation of a sine waveform is used to transform a baseband message signal into a passband signal, such as low-frequency audio signal into a radio-frequency signal (RF signal). The modulated waveform enters the channel and is corrupted by noise and interference.

In the receiver, the demodulator conveys the corrupted waveform back to a set of symbols $\mathbf{r}$ called *received sequence* that corresponds to the codeword $\mathbf{v}$. Due to the noisy environment of the channel, the received sequence $\mathbf{r}$ could be different from the codeword $\mathbf{v}$. The *channel decoder* converts received sequence $\mathbf{r}$ into an *estimated information sequence* $\hat{\mathbf{u}}$. The *source decoder* then transforms the estimated information sequence $\hat{\mathbf{u}}$ into an estimate of the information source and delivers this estimate to the *destination*.

1.2 Channel coding

This dissertation focuses on channel coding which is involved in the channel encoder/decoder of the block diagram in Fig. 1.1. In communication or storage system, channel coding or forward error correction (FEC), is a technique used for controlling errors in

data transmission over unreliable or noisy channels. The central idea is encoding the information sequence in a redundant way by using a channel code.

The redundancy allows the receiver to detect a limited number of errors, which may occur anywhere in the sequence of symbols, and often to correct these errors. Channel coding gives the receiver the ability to correct errors without a reverse channel to request retransmission of data, but at the cost of a fixed, higher forward channel bandwidth. Channel coding is therefore applied in situations where retransmissions are costly or impossible, such as one-way communication links and when broadcasting to multiple receivers.

Richard Hamming pioneered this field in the 1940s and invented the first error-correcting code in 1950: the *Hamming* $(7, 4)$ *code*. *Reed-Solomon* (RS) codes are non-binary cyclic error-correcting codes invented by Irving S. Reed and Gustave Solomon. Reed-Solomon coding is very widely used in mass storage systems, such as CD and DVD, to correct the burst errors. *Fountain codes* (also known as rateless erasure codes) are a class of erasure codes with the property that a potentially limitless sequence of encoding symbols can be generated from a given set of source symbols such that the original source symbols can ideally be recovered from any subset of the encoding symbols of size equal to or only slightly larger than the number of source symbols. The term fountain or rateless refers to the fact that these codes do not exhibit a fixed code rate. LT codes were the first practical realization of fountain codes. Raptor codes and Online codes were subsequently introduced, and achieve linear time encoding and decoding complexity through a pre-coding stage of the input symbols. *Convolutional codes* are used extensively in numerous applications in order to achieve reliable data transfer, including digital video, radio, mobile communication, and satellite communication. These codes are often implemented in concatenation with a hard-decision code, particularly Reed Solomon. Prior to turbo codes, such constructions were the most efficient, coming closest to the Shannon limit.

Apart from above channel codings, *Turbo codes* [6] and *low-density parity-check* (LDPC) codes [7] are two popular channel coding schemes with capacity-approaching property. Turbo codes were invented in 1993 by Berrou *et al.* together with a suitable iterative soft-decoding scheme. They combine two or more relatively simple convolutional codes and interleavers to produce a block code that can perform to within a fraction of a decibel away from Shannon limit. The first class of turbo codes was the parallel concatenated convolutional code (PCCC). Since then, many other classes of turbo-like codes have been discovered, including serial versions and repeat-accumulate codes.

Besides turbo codes, LDPC codes form another class of capacity-approaching channel codes that were first discovered by Gallager in his doctoral dissertation at the MIT in the early 1960s. Unfortunately, this great discovery did not attract much attention until the late 1990s, except for the work by Tanner in 1981, who worked on the interpretation of LDPC codes from a graphical point of view, which is called Tanner graph nowadays. LDPC codes can be separated into two categories: LDPC block codes (LDPC-BCs) and LDPC convolutional codes (LDPC-CCs). LDPC-BCs can also be obtained from LDPC-CCs by termination or tail-biting. In [2], it has been shown that LDPC-CCs are suitable for practical implementation with continuous transmission as well as, via encoder termination or tail-biting, for block transmission in frames of arbitrary size [3] without an increase in computational complexity compared to their block code counterparts. Therefore, it is of great interest to analyze the characteristics of LDPC-CCs.

The characteristics of LDPC codes, include *girth* (the length of shortest cycles), distance spectrum (including the minimum (free) distance), decoding threshold [8], trapping sets [9], pseudocodewords [10], and others. In this dissertation, we focus on the analysis of short cycles including the structure of avoidable short cycles, distance spectrum including the estimate of minimum free distance, and cycle trapping sets, which are a subset of trapping sets. Based on these three aspects, we proposed a puncturing algorithm to obtain rate-compatible (RC) LDPC-CCs that can be used for wireless communication to adapt to the time-varying conditions of the channel. Unlike the family of RC convolutional codes first introduced in [11] by periodically puncturing encoded bits chosen with respect to a distance spectrum criterion, our puncturing selection criterion is based on a couple of other coding characteristics.

1.3 Outline

The rest of the dissertation is organized as follows:

- In Chapter 2, we give definitions of both time-invariant and time-varying LDPC-CCs in the time- and polynomial-domains. We introduce the encoding implementation of LDPC-CCs by means of the polynomial *syndrome former matrix*. In terms of the decoding algorithm, we first basically review the sum product algorithm (SPA [12]) which is typically used for LDPC-BCs. Then, based on the SPA, we illustrate how the sliding window decoder [1] works for LDPC-CCs.

- In Chapter 3, we discuss the cycle formations of time-invariant and time-varying

LDPC-CCs in both the time- and polynomial-domain syndrome former matrices. Rather than counting cycles in the large-scale semi-infinite time-domain syndrome former matrix, we enumerate the cycles in the compact polynomial syndrome former matrix of finite size. Moreover, we analyze the structure of unavoidable cycles of length ranging from 6 to 20. As a result, we illustrate a set of weight matrices associated with the polynomial syndrome former matrices that have upper bounds on the girth (the length of the shortest cycle). A cycle counter algorithm is proposed to enumerate cycles for time-invariant and time-varying LDPC-CCs. Based on this algorithm, a method is introduced to generate LDPC-CCs with desired girth.

- Due to the large syndrome former memory of LDPC-CCs, it is difficult to compute precise distance spectra of LDPC-CCs. For example, the complexity of applying the BEAST algorithm [13] is proportional to the syndrome former memory. In Chapter 4, we estimate the distance spectra of LDPC-CCs by evaluating the linear dependence between the codewords of *super codes*. We obtain an upper bound on the minimum free distance and an lower bound on the number of codewords with certain Hamming weight.

- In Chapter 5 we propose a puncturing algorithm to obtain *rate-compatible* (RC) LDPC-CCs by periodically puncturing encoded bits with respect to some puncturing criteria, i.e., (1) ensuring the recoverability of punctured variable nodes, (2) minimizing the number of *completely punctured cycle trapping sets* (CPCTSs), and (3) minimizing the number of punctured variable nodes involved in short cycles. One of the advantages of this method compared to designing a puncturing scheme based on the distance spectrum, which is difficult to compute in practice, is that these properties can be precisely and efficiently calculated.

- In Chapter 6, we conclude our work and summarize the major contribution. We also suggest a number of ideas for future research in this area.

Chapter 2

Low-Density Parity-Check Convolutional Codes

2.1 Time-domain LDPC convolutional codes

A rate $R = b/c$ LDPC-CC can be described as the set of sequences $\mathbf{v}$ satisfying $\mathbf{v}\cdot\mathbf{H}^T = \mathbf{0}$,[1] where

$$\begin{aligned}\mathbf{v} &= (\mathbf{v}_0,\mathbf{v}_1,\ldots,\mathbf{v}_t,\ldots)\\ &= ((v_0^{(1)},v_0^{(2)},\ldots,v_0^{(c)}),\ldots,(v_t^{(1)},v_t^{(2)},\ldots,v_t^{(c)}),\ldots),\end{aligned} \tag{2.1}$$

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{H}_0^T(0) & \mathbf{H}_1^T(1) & \cdots & \mathbf{H}_{m_s}^T(m_s) & & & \\ & \mathbf{H}_0^T(1) & \cdots & \mathbf{H}_{m_s-1}^T(m_s) & \mathbf{H}_{m_s}^T(m_s+1) & & \\ & & \ddots & \vdots & \vdots & \ddots & \\ & & & \mathbf{H}_0^T(m_s) & \mathbf{H}_1^T(m_s+1) & \cdots & \mathbf{H}_{m_s}^T(m_s+m_s) \\ & & & & \ddots & & \ddots \end{bmatrix}, \tag{2.2}$$

$$\mathbf{H}_i^T(t) = \begin{bmatrix} h_{1,1}^{i,t} & \cdots & h_{1,p}^{i,t} \\ \vdots & \ddots & \vdots \\ h_{c,1}^{i,t} & \cdots & h_{c,p}^{i,t} \end{bmatrix}, \tag{2.3}$$

and $p \triangleq c-b$, the blank spaces correspond to zeros. The transposed parity check matrix $\mathbf{H}^T$, called the *syndrome former matrix*, is made up of a set of binary submatrices $\mathbf{H}_i^T(t)$, $t\in\mathbb{Z}$, $i = 0, 1, \ldots, m_s$, each of size $c\times p$, given by (2.3). m_s is called the *syndrome former memory*, and the associated *constraint length*, defined as $\bar{v}_s = (m_s + 1)\cdot c$, is

[1] Note that all the LDPC-CCs considered throughout the dissertation are binary. Therefore, an entry in $\mathbf{v}$ or $\mathbf{H}^T$ is either "0" or "1" and the matrix product is based on modulo 2.

proportional to the decoding complexity of pipeline decoding [1]. If, beginning with the block of q columns at time $t = m_s$, $\mathbf{H}^T$ contains exactly J ones in each row and K ones in each column, then the code is called an (m_s, J, K)-regular LDPC-CC. Note that a row in the syndrome former matrix refers to a variable node while a column corresponds to a check (constraint) node in the Tanner graph representation of $\mathbf{H}^T$. Specifically, a variable node, denoted as $v_t^{(j)}$, $j = 1, 2, \ldots, c$, corresponds to row $tc+j$ of (2.2), while a check node, denoted as $c_t^{(k)}$, $k = 1, 2, \ldots, q$, corresponds to column $tq + k$ of (2.2). For time-*invariant* LDPC-CCs, the binary submatrices in $\mathbf{H}^T$ are constant, i.e., $\mathbf{H}_i^T(t) = \mathbf{H}_i^T$, $\forall i, t$, while for periodically time-*varying* LDPC-CCs with period T the submatrices repeat periodically, so that $\mathbf{H}_i^T(t) = \mathbf{H}_i^T(t + T)$. In the dissertation, LDPC-CCs are assumed to be time-invariant unless stated otherwise.

2.2 Polynomial-domain LDPC-CCs

Given the time-domain syndrome former matrix $\mathbf{H}^T$ of a *time-invariant* LDPC-CC in (2.2) with rate $R = b/c$ and syndrome former memory m_s, the corresponding polynomial-domain syndrome former matrix, also called polynomial syndrome former matrix, is given by [14]:

$$\begin{aligned} \mathbf{H}^T(D) &= \textstyle\sum_{n=0}^{m_s} \mathbf{H}_n^T \cdot D^n \\ &= \begin{bmatrix} h_{1,1}(D) & h_{1,2}(D) & \cdots & h_{1,p}(D) \\ h_{2,1}(D) & h_{2,2}(D) & \cdots & h_{2,p}(D) \\ \vdots & \vdots & \ddots & \vdots \\ h_{c,1}(D) & h_{c,2}(D) & \cdots & h_{c,p}(D) \end{bmatrix}, \end{aligned} \tag{2.4}$$

Where $h_{i,j}(D)$, $i = 1, 2, \ldots, c$, $j = 1, 2, \ldots, p$, is an polynomial entry. The maximum power of D in (2.4) is defined as m_s. The corresponding *weight matrix* $\mathbf{B}$ associated with $\mathbf{H}^T(D)$ is defined as:

$$\mathbf{B} = \mathcal{W}(\mathbf{H}^T(D)) = \begin{bmatrix} \mathcal{W}(h_{1,1}(D)) & \mathcal{W}(h_{1,2}(D)) & \cdots & \mathcal{W}(h_{1,p}(D)) \\ \mathcal{W}(h_{2,1}(D)) & \mathcal{W}(h_{2,2}(D)) & \cdots & \mathcal{W}(h_{2,p}(D)) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{W}(h_{c,1}(D)) & \mathcal{W}(h_{c,2}(D)) & \cdots & \mathcal{W}(h_{c,p}(D)) \end{bmatrix}, \tag{2.5}$$

where $\mathcal{W}(h_{i,j}(D))$ indicates the *weight* or *degree* of the polynomial entry $h_{i,j}(D)$. The weight of a polynomial is defined as the number of additive terms with different powers

of D. For instance, the degree of $1 + D + D^2$ is three, while the degree of $D + D^3$ is two. When the degree of a polynomial is one and two, it is also called a *monomial* or a *binomial*, respectively. If the sum of the elements in each row of $\mathbf{B}$ is J and the sum of the elements in each column of $\mathbf{B}$ is K, the associated LDPC-CC is called an (m_s, J, K)-regular time-invariant LDPC-CC.

For a rate $R = b/c$ periodically *time-varying* LDPC-CC with period T, given the time-domain syndrome former matrix $\mathbf{H}^T$ in (2.2) and syndrome former memory m_s, the polynomial-domain syndrome former matrix is given by

$$\mathbf{H}^T(D) = \begin{bmatrix} \mathbf{H}_1^T(D) & & & & & & \\ & \mathbf{H}_2^T(D) & & & & & \\ & & \ddots & & & & \\ & & & \mathbf{H}_i^T(D) & & & \\ & & & & \ddots & & \\ & & & & & \mathbf{H}_T^T(D) & \end{bmatrix}, \tag{2.6}$$

where $i = 1, 2, \ldots, T$, and blank spaces correspond to empty entries. Each polynomial sub-matrix $\mathbf{H}_i^T(D)$ is defined as

$$\mathbf{H}_i^T(D) = \sum_{n=0}^{m_s} \mathbf{H}_n^T(i+n) \cdot D^n \tag{2.7}$$

with $\mathbf{H}_n^T(i+n)$ given in (2.2). Finally, $\mathbf{H}^T(D)$ consists of T distinct polynomial sub-matrices, and each is of size $c \times (c-b)$. Similar to the time-invariant LDPC-CCs, we can also form the weight matrix $\mathbf{B}$ for time-varying LDPC-CCs. It is of the same size as the polynomial matrix in (2.6). Each entry in the base matrix $\mathbf{B}$ is the weight of the corresponding polynomial in $\mathbf{H}^T(D)$. Note that the weight of an empty entry is zero. To some extent, a rate $R = b/c$ periodically time-varying LDPC-CC with period T defined in (2.6) can be considered as a time-invariant LDPC-CC with rate $R = (bT)/(cT)$. In other words, we consider the polynomial matrix (2.6) in the form of (2.4) with many empty entries.

The c-tuple of a codeword $\mathbf{V}(D)$ in the polynomial-domain codeword set $\overline{\mathbf{V}}(D)$ is given by

$$\mathbf{V}(D) \triangleq [\mathbf{v}^{(0)}(D), \mathbf{v}^{(1)}(D), \ldots, \mathbf{v}^{(c-1)}(D)], \tag{2.8}$$

i.e., $\mathbf{V}(D)\mathbf{H}^T(D) = \mathbf{0}(D)$. After multiplexing, the codeword can also be expressed

as [4]

$$\mathbf{v}(D) \triangleq \mathbf{v}^{(0)}(D^c) + D \cdot \mathbf{v}^{(1)}(D^c) + \cdots + D^{c-1} \cdot \mathbf{v}^{(c-1)}(D^c), \tag{2.9}$$

which is a member of the multiplexed codeword set $\overline{\mathbf{v}}(D)$. Due to the repeating syndrome former matrix structure of time-invariant LDPC-CCs, a periodically shifted codeword $D^n\mathbf{V}(D)$, or $D^{c \cdot n}\mathbf{v}(D)$, $n \in \mathbb{Z}$, is also a codeword, and consequently it satisfies the constraint imposed by the polynomial syndrome former matrix, i.e.,

$$D^n\mathbf{V}(D)\mathbf{H}^T(D) = \mathbf{0}(D). \tag{2.10}$$

2.3 Time-invariant LDPC-CCs

A particular type of time-invariant LDPC-CCs is derived from the quasi-cyclic (QC) LDPC block codes (LDPC-BCs). Their syndrome former matrices are made up of a set of circulant matrices.[1] Unlike the random LDPC-BCs, QC LDPC-BCs have geometrical structure in their syndrome former matrices, which facilitates the practical implementation and makes them easier to evaluate the coding properties, such as number of cycles, trapping sets, and minimum distance. Tanner et al. [5] presented a method to obtain QC LDPC-BCs. Based on the structure of the obtained QC LDPC-BCs, they also introduced a family of LDPC-CCs. Details are given as follows.

In the Galois field GF(m), with m prime, the nonzero elements of GF(m) form a cyclic multiplicative group. As described in [5], we assume that a and b are two nonzero elements with multiplicative orders[2] $o(a) = K$ and $o(b) = J$, respectively. With the (s,t)-th element $P_{s,t} = b^s a^t \mod m$, $s = 0, 1, ..., J-1$, and $t = 0, 1, ..., K-1$, we form the matrix P of size $K \times J$ with elements in GF(m):

$$P = \begin{bmatrix} 1 & b & \cdots & b^{J-1} \\ a & ab & \cdots & ab^{J-1} \\ a^2 & a^2b & \cdots & a^2b^{J-1} \\ \cdots & \cdots & \cdots & \cdots \\ a^{K-1} & a^{K-1}b & \cdots & a^{K-1}b^{J-1} \end{bmatrix}. \tag{2.11}$$

Note that all products in the matrix are "modulo m". The syndrome former matrix (transposed parity-check matrix) $\mathbf{H}^T$ of a QC LDPC-BC is generated from (2.11) as

[1] Circulant matrices are cyclically shifted identity matrices.

[2] In GF(m), the multiplicative order of a modulo m is the smallest positive integer k satisfying $a^k \equiv 1 (\mod m)$.

follows:

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_b & \cdots & \mathbf{I}_{b^{J-1}} \\ \mathbf{I}_a & \mathbf{I}_{ab} & \cdots & \mathbf{I}_{ab^{J-1}} \\ \mathbf{I}_{a^2} & \mathbf{I}_{a^2b} & \cdots & \mathbf{I}_{a^2b^{J-1}} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{I}_{a^{K-1}} & \mathbf{I}_{a^{K-1}b} & \cdots & \mathbf{I}_{a^{K-1}b^{J-1}} \end{bmatrix} \tag{2.12}$$

The matrices $\mathbf{I}_{P_{s,t}}$ are $m \times m$ identity matrices cyclically shifted to the left by $P_{s,t} - 1$ positions. Note that, therefore, $\mathbf{I}_1$ corresponds to an "unshifted" standard identity matrix.

The syndrome former matrix $\mathbf{H}^T$ is of size $Km \times Jm$, with Km being the block length of the code and Jm being the number of parity-check constraints, some of which may be linearly dependent resulting in rank deficiency in the syndrome former matrix. Due to the construction of shifted identity matrices, each row of $\mathbf{H}^T$ contains J ones while every column contains K ones, resulting in a (J, K)-regular QC LDPC-BC. As we have K (cyclically shifted) identity matrices in every m columns of $\mathbf{H}^T$, we can add all of these columns in such a block of m columns and obtain one "all-one" column every m columns. As we have J of such blocks in $\mathbf{H}^T$, we obtain the number J "all-one" columns of which, trivially, $J - 1$ are linearly dependent. Hence, the $\mathbf{H}^T$ matrix has, by construction, at least $J - 1$ linearly dependent columns, so we obtain the rate of the QC LDPC block code: $R \geq \frac{K \cdot m - J \cdot m + (J-1)}{K \cdot m} = 1 - \frac{J}{K} + \frac{J-1}{K \cdot m}$.

As explained in [5], LDPC-CCs can be formed by a matrix that consists of monomial entries, with each monomial entry describing the first column of the corresponding shifted identity sub-matrix in $\mathbf{H}^T$. For instance, if the first *column* of a *horizontally left-shifted* identity matrix is $[00010]^T$ (this means three *horizontal* shifts to the left), it is represented by the polynomial D^3. Therefore, given the syndrome former matrix in (2.12), the associated polynomial syndrome former matrix for the LDPC-CC is described as[1]

$$\mathbf{H}^T(D) = \begin{bmatrix} D^0 & D^{b-1} & \cdots & D^{b^{J-1}-1} \\ D^{a-1} & D^{ab-1} & \cdots & D^{ab^{J-1}-1} \\ D^{a^2-1} & D^{a^2b-1} & \cdots & D^{a^2b^{J-1}-1} \\ \cdots & \cdots & \cdots & \cdots \\ D^{a^{K-1}-1} & D^{a^{K-1}b-1} & \cdots & D^{a^{K-1}b^{J-1}-1} \end{bmatrix}. \tag{2.13}$$

[1] Note that the subscripts of the matrix elements in (2.12) do not carry over directly as powers of D in (2.13): we have to subtract "1" due to the definition of I_p (see above) indicating an identity matrix that is cyclically shifted $p - 1$ (not p) positions to the left.

A polynomial syndrome former matrix generated in this way produces a (m_s, J, K)-regular time-invariant LDPC-CC; it has exactly J monomials in each row, K monomials in each column, i.e., the corresponding weight matrix has all one entries, and m_s is defined as the maximum power of D in $\mathbf{H}^T(D)$. Throughout the dissertation, we call such codes with only monomial entries in the polynomial syndrome former matrix (m_s, J, K) Tanner LDPC-CCs. A remarkable property of such LDPC-CCs over the corresponding QC LDPC-BC is that there are (usually [5, p. 2970]) no linear dependent rows in the time-domain $\mathbf{H}^T$ matrix leading to a code rate of exactly $R = 1 - J/K$.

Note that when we design LDPC-CCs, we can focus on the structure of the weight matrix and the entries in the polynomial syndrome former matrix without considering the corresponding QC LDPC-BCs. In addition, the polynomial entries in (2.13) are not necessarily monomials.

Example 2.3.1. Here we use the method summarized above to generate a Tanner LDPC-CC (this code can be found in [5]). We pick the field GF(7) and the elements $a = 2$ and $b = 6$. As $K = 3$ is the smallest positive integer $K > 0$ for which $a^K \bmod 7 = 1$, the order of a is $K \doteq o(a) = 3$; similarly, the order of b is $J \doteq o(b) = 2$. From this we obtain by (2.12)

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_6 \\ \mathbf{I}_2 & \mathbf{I}_5 \\ \mathbf{I}_4 & \mathbf{I}_3 \end{bmatrix}_{3\cdot 7 \times 2\cdot 7} \tag{2.14}$$

for the syndrome former matrix of the QC LDPC-BC. By (2.13) we obtain the polynomial syndrome former matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} D^0 & D^5 \\ D^1 & D^4 \\ D^3 & D^2 \end{bmatrix} \tag{2.15}$$

or equivalently,

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix} \tag{2.16}$$

for the corresponding LDPC-CC.[1] (2.15) and (2.16) are referred to $(5, 2, 3)$ and $(3, 2, 3)$ Tanner LDPC-CCs, respectively. Both of them define a code of rate $R = 1 - J/K =$

[1] The equivalence is being discussed in Section 2.3.1.

$1 - 2/3 = 1/3$. However, (2.16) defines a code with smaller syndrome former memory than (2.15) does, which is desired as large m_s results in high decoding complexity. ■

2.3.1 Tanner graph of LDPC codes

A bipartite Tanner graph (factor graph) is used to describe LDPC codes. It consists of two types of nodes: the variable nodes denoted by v_i and the check nodes denoted by c_j. A variable node v_i corresponds the i-th row of the $\mathbf{H}^T$ matrix, while a check node c_j corresponds the j-th column of $\mathbf{H}^T$. Variable node v_i and check node c_j are connected by an edge whenever the entry h_{ij} is "1", where h_{ij} indicates the entry in the i-th row and j-th column of $\mathbf{H}^T$. In this section, two example codes are used to illustrate the Tanner graph representations of a QC LDPC-BC and an LDPC-CC.

Example 2.3.2. The QC LDPC-BC given by (2.14) has the syndrome former matrix

$$\mathbf{H}^T = \left[\begin{array}{ccccccc|ccccccc} 1&0&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&1&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&1&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&1&0&0&0&0&0\\ \hline 0&0&0&0&0&0&1&0&0&0&1&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&1&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&1&0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&1&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&1&0&0&0&0\\ \hline 0&0&0&0&1&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&1&0&0 \end{array}\right]. \tag{2.17}$$

Based on the above description, the corresponding Tanner graph of this code is shown in Fig. 2.1.

In Fig. 2.1, each variable node is connected to two neighboring check nodes, while each check node contains three neighboring variable nodes. It is consistent with the fact that the row weight and column weight[1] in $\mathbf{H}^T$ are two and three, respectively. In the Tanner graph, there is a set of edges colored in blue forming a cycle of length 12. When designing LDPC codes, short cycles should be removed since they degrade the decoding performance. The length of the shortest cycles in the Tanner graph is called *girth* of an LDPC code. Details about cycles in LDPC codes are discussed in Chapter 3. ■

[1]The weight of a row or column is referred to the number of "1"s involved in it.

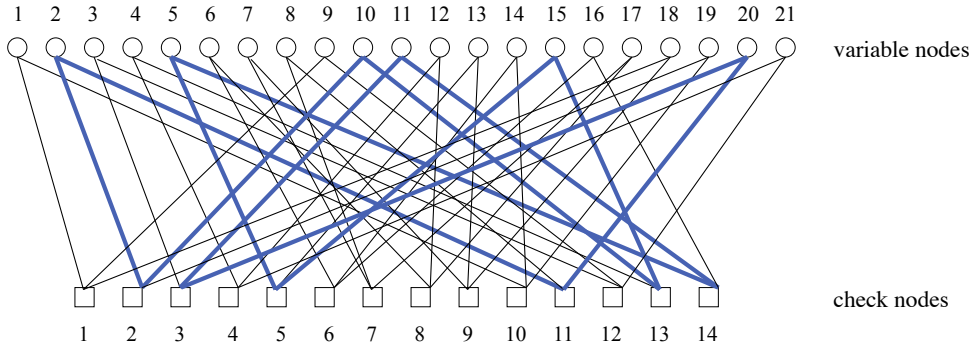


Figure 2.1: Tanner graph of the LDPC code in (2.17).

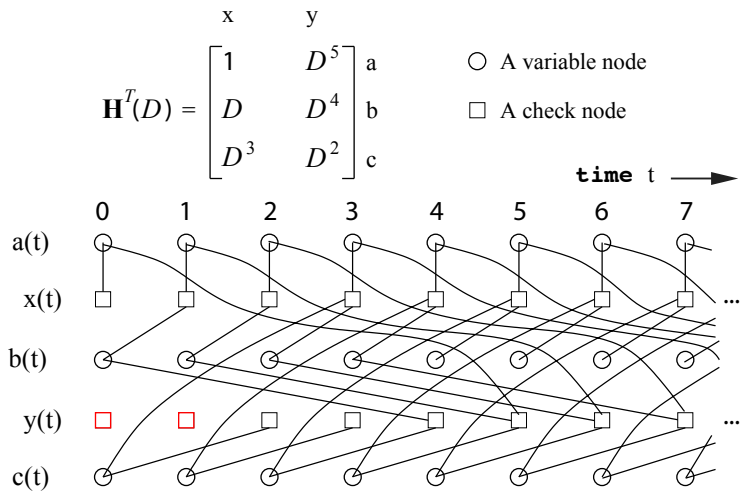


Figure 2.2: Tanner graph of the $(5, 2, 3)$ LDPC-CC example code in (2.15).

Example 2.3.3. We continue with the LDPC-CC defined by (2.15), its Tanner graph representation is given in Fig. 2.2.

The equivalence of the codes in (2.15) and (2.16) can be explained by the Tanner graph. It follows directly from (2.15) with "D" being interpreted as a "delay" between two time steps. We denote by `a,b,c` the three code bits (variable nodes) which are generated by the encoder for every input data bit time-indexed by `t`; the two parity checks which must be fulfilled at each time `t` are denoted by `x,y`. The check `x` at time `t` is given by `x(t)=a(t)+b(t-1)+c(t-3)` (all "+"-operations are "modulo-2") which directly reflects the first column in the syndrome former matrix (2.15). Similarly, for the check `y` at time `t`, we obtain `y(t)=a(t-5)+b(t-4)+c(t-2)`.

Conversely applying the transformation described by (2.4) to the $\mathbf{H}^T(D)$ matrix in

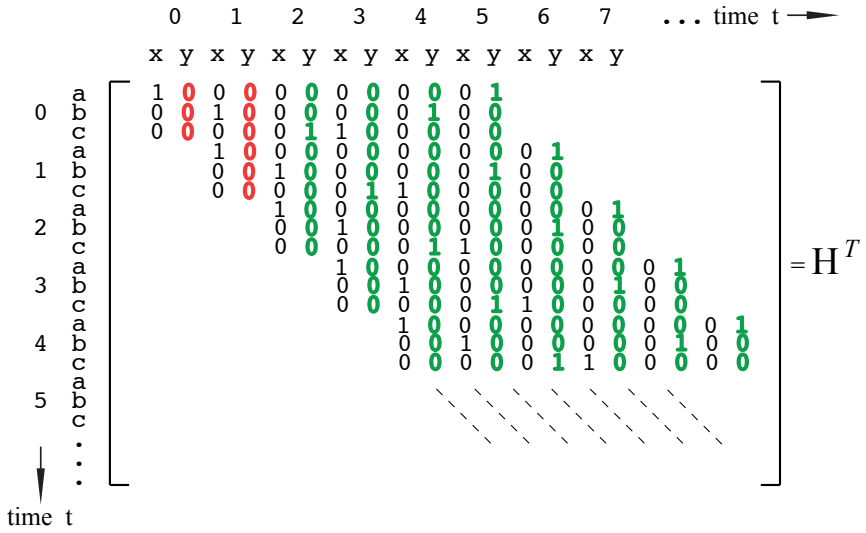


Figure 2.3: Syndrome former matrix of the $(5, 2, 3)$ Tanner LDPC-CC.

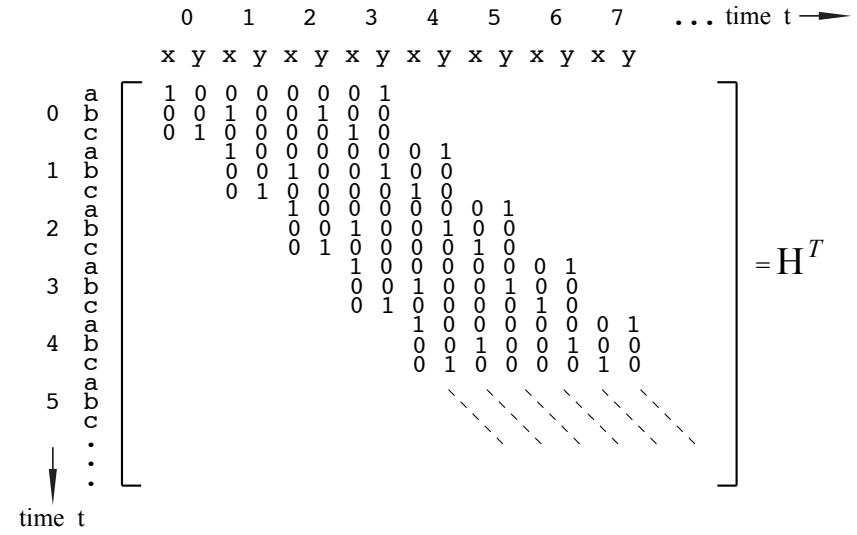


Figure 2.4: Syndrome former matrix of the $(3, 2, 3)$ Tanner LDPC-CC.

(2.15), we obtain the syndrome former matrix presented in Fig. 2.3, where the blank spaces are all zeros. The syndrome former matrix shows two checks `y(0),y(1)` (colored in red) that are not connected to any code bits, i.e., these two columns are all-zero vectors. They correspond to the two unconnected checks (colored in red) in the y-path in Fig. 2.2. As the code is not changed by permuting columns in the syndrome former matrix, we can remove the "empty" checks and move the corresponding checks (colored in green) at later time steps upwards. As a result, we obtain the new syndrome former matrix in Fig. 2.4, which is the time-domain syndrome former matrix of the code in (2.16). Thus, this process corresponds to pulling out common factors $D^l, l > 0$ from any column of $\mathbf{H}^T(D)$ [15]. Removing common factors reduces the syndrome former memory of LDPC-CCs. For example, compared with the code in (2.15) the syndrome former memory of the code defined in (2.16) is reduced from $m_s = 5$ to $m_s = 3$. ■

2.4 Time-varying LDPC-CCs

In [1], the derivation of periodically time-varying LDPC-CCs from LDPC-BCs was proposed. The principle is to unwrap the parity-check or syndrome former matrix of an LDPC-BC and duplicate the unwrapped matrix to infinity along the diagonal. This unwrapping method has been further discussed by Pusane [16]. Normally, given a QC LDPC-BC of rate $R = b/c$ and the identity matrix of size $m \times m$, the unwrapping step size is chosen to be $ck \times (c-b)k$, $0<k\leqslant m$, $k\in\mathbb{Z}^+$; when k=1, the period of the obtained time-varying LDPC-CC is $T = m$.

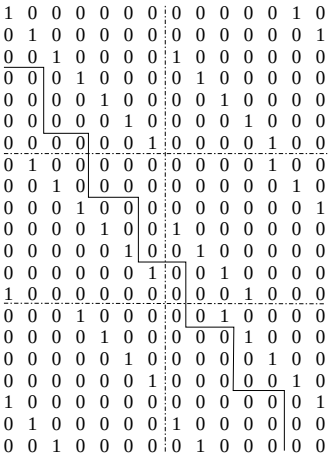


Figure 2.5: Syndrome former matrix of the QC LDPC-BC in (2.14).

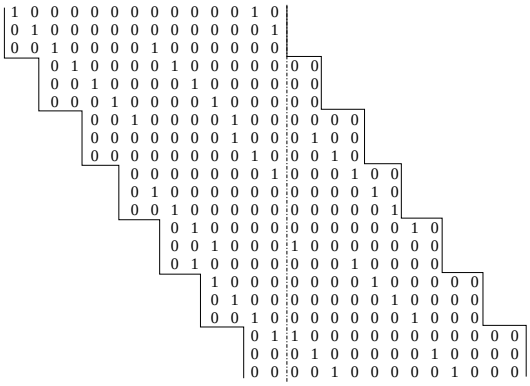


Figure 2.6: Unwrapped QC LDPC-BC in (2.14)

To illustrate the unwrapping technique, we apply it to the rate $R = 1/3$ QC LDPC-BC in (2.14) whose transposed parity-check matrix is given in Fig. 2.5. The unwrapping step size is set to be $ck \times (c-b)k = 3 \cdot 1 \times (3-1) \cdot 1$. As indicated in Fig. 2.5, we "cut" the syndrome former matrix into two pieces, whereby the cutting pattern is such that we repeatedly move two units to the right and then three units down. Having applied this "diagonal cut", we copy and paste the upper part to the bottom of the lower part resulting in the matrix in Fig. 2.6. Repeating the matrix in Fig. 2.6 along the diagonal, we obtain a rate $R = 1/3$ time-varying LDPC-CC with period $T = 7$. Note that, time-varying LDPC-CCs derived in this way maintain the row and column weights and the code rate is the same as for the block codes.

Considering the time-domain syndrome former matrix partially given in Fig. 2.6, according to (2.6) and (2.7), we obtain the corresponding polynomial-domain syndrome former matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} \mathbf{H}_1^T(D) & & & & & & & \\ & \mathbf{H}_2^T(D) & & & & & & \\ & & \mathbf{H}_3^T(D) & & & & & \\ & & & \mathbf{H}_4^T(D) & & & & \\ & & & & \mathbf{H}_5^T(D) & & & \\ & & & & & \mathbf{H}_6^T(D) & & \\ & & & & & & \mathbf{H}_7^T(D) \end{bmatrix}, \tag{2.18}$$

where blank spaces correspond to empty entries. It consists of $T = 7$ polynomial

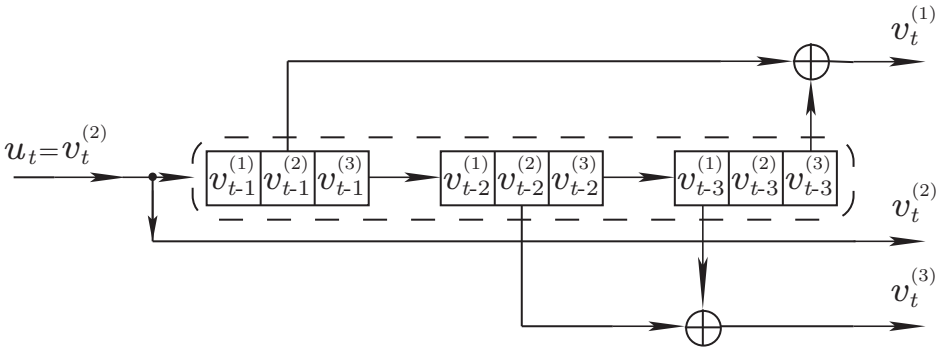


Figure 2.7: Encoder for the $(3, 2, 3)$ Tanner LDPC-CC based on polynomial syndrome former matrix.

sub-matrices, and they are given by

$$\mathbf{H}_1^T(D) = \begin{bmatrix} D^6+1 & 0 \\ 0 & D^6+1 \\ D & D^3 \end{bmatrix}, \mathbf{H}_2^T(D) = \begin{bmatrix} D^3 & 1 \\ D & D^3 \\ D^4 & D \end{bmatrix}, \mathbf{H}_3^T(D) = \begin{bmatrix} D & D^3 \\ 0 & D^5+D^3 \\ D^6+D^4 & 0 \end{bmatrix}$$

$$\mathbf{H}_4^T(D) = \begin{bmatrix} 0 & D^5+D^3 \\ D^6 & 1 \\ D & D^6 \end{bmatrix}, \mathbf{H}_5^T(D) = \begin{bmatrix} D^6 & 1 \\ D^3+D & 0 \\ 0 & D^4+1 \end{bmatrix}, \mathbf{H}_6^T(D) = \begin{bmatrix} D^4+1 & 0 \\ 0 & D^4+1 \\ D^5+D & 0 \end{bmatrix},$$

$$\mathbf{H}_7^T(D) = \begin{bmatrix} D & 1 \\ 0 & D^4+D \\ D^5+D^2 & 0 \end{bmatrix}.$$

The obtained polynomial syndrome former matrix has 2 and 3 monomials in each row and column, respectively, resulting in a time-varying $(m_s = 6, J = 2, K = 3)$-regular $R = 1/3$ LDPC-CC.

2.5 Encoding of LDPC-CCs

LDPC-BCs can be encoded by the generator matrix which is obtained from its syndrome former (or parity-check) matrix by means of Gaussian elimination. Similarly, for the convolutional codes a generator matrix $\mathbf{G}(D)$ can be obtained by Gaussian elimination of $\mathbf{H}^T(D)$. However, it is also possible to directly encode LDPC-CCs using the polynomial syndrome former matrices.

Recall the definition of LDPC-CCs in (2.2) that satisfies $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$. This constraint imposed by the syndrome former matrix can be written as

$$\mathbf{v}_t \mathbf{H}_0^T(t) + \mathbf{v}_{t-1} \mathbf{H}_1^T(t) + \cdots + \mathbf{v}_{t-m_s} \mathbf{H}_{m_s}^T(t) = 0, t \in \mathbb{Z}^*. \tag{2.19}$$

Motivated by (2.19), an encoder is derived for the LDPC-CC. Here we use an example to show how the encoder works.

Example 2.5.1. Consider the rate $R = 1/3$ $(3, 2, 3)$ Tanner LDPC-CC given in (2.16). Its polynomial syndrome former matrix $\mathbf{H}^T(D)$ can be written according to

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} + D \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} + D^2 \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + D^3 \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}. \tag{2.20}$$

Let $\mathbf{v}_t = (v_t^1, v_t^2, v_t^3)$ represent the three bits of a codeword at time unit t in a code sequence. According to (2.2) and (2.19), it follows that

$$\mathbf{v}_t \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} + \mathbf{v}_{t-1} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} + \mathbf{v}_{t-2} \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + \mathbf{v}_{t-3} \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} = \mathbf{0}_{3\times 2}. \tag{2.21}$$

Now, if we let $v_t^{(2)}$ correspond to an information bit u_t then the corresponding parity-check can be determined by solving (2.21), i.e.,

$$\begin{cases} v_t^{(1)} = v_{t-1}^{(2)} + v_{t-3}^{(3)} \\ v_t^{(3)} = v_{t-2}^{(2)} + v_{t-3}^{(1)} \end{cases}$$

Based on above equations, the encoding system can be easily implemented as shown Fig. 2.7. The system continuously encodes information bits into codewords by exploiting the relationship between the input information bit and the bits of codewords in different time slots. ■

2.6 Decoding of LDPC-CCs

LDPC codes are iteratively decoded based on the *maximum a posteriori probability* (MAP) algorithm. LDPC-BCs have a finite topology in the Tanner graphs. The iterative decoding algorithm runs over the graph until a codeword is found or the maximum number of iteration is reached. However, due to the infinite topology of the Tanner graphs of LDPC-CCs, the decoding algorithm used for LDPC-BCs cannot be directly applied to LDPC-CCs. Instead, a so-called *pipeline decoding algorithm* is derived based on the traditional *sum-product algorithm* (SPA). Before discussing the decoding algorithm for LDPC-CCs, we review the decoding method for LDPC-BCs.

2.6.1 Sum-product algorithm for LDPC-BCs

LDPC-BCs are usually decoded by the so-called sum-product algorithm (SPA), which is an iterative belief-propagation, or message-passing algorithm, over the Tanner graph (factor graph) of the code. The SPA is a soft decision algorithm that iteratively computes the probability of a transmitted bit, given the received bit value.

As an input of the decoder, the input bit probability is called a *prior probability* because it is known in advance before running the LDPC decoder. The bit probabilities returned by the decoder are called a *posteriori probabilities*. In the case of the SPA these probabilities are expressed as *Log-likelihood ratios* (LLRs) to greatly reduce the complexity of the algorithm and make the decoding algorithm numerical stable. In the Tanner graph representation, these LLRs are called *messages* that propagate between variable nodes and check nodes.

For a binary variable x it is easy to find $p(x = 1)$ given $p(x = 0)$, since $p(x = 1) = 1 - p(x = 0)$ and so we only need to store one probability value for x. LLRs are used to represent both probabilities for a binary variable by a single value:

$$L(x) = \log \frac{p(x = 0)}{p(x = 1)}, \tag{2.22}$$

where we use the natural logarithm $\log_e$. If $p(x = 0) > p(x = 1)$ then $L(x)$ is positive and the greater the difference between $p(x = 0)$ and $p(x = 1)$, i.e., the more sure we are that $x = 0$, and the larger the positive value for $L(x)$. Conversely, if $p(x = 1) > p(x = 0)$ then $L(x)$ is negative and the greater the difference between $P(x = 0)$ and $p(x = 1)$ the larger the negative value for $L(x)$ [12]. Thus the sign of $L(x)$ provides the hard decision on x and the magnitude $|L(x)|$ represents the reliability of this decision.

The aim of SPA decoding is to compute the maximum a posteriori probability for each codeword bit, $P_i = P(x_i|N)$, which is the probability that the i-th codeword bit (corresponds to the variable node v_i of a codeword) is a 1 conditional on the event N that all parity-check constraints are satisfied. The information about bit x_i (variable node v_i) received from the parity-checks (neighboring check nodes) is called *extrinsic information* for x_i.

Regarding the scheduling of decoding LDPC-BCs, it mainly includes the flooding algorithm, layer decoding [17, 18], informed dynamic decoding [19–21]. Now we discuss the most traditional one, the flooding algorithm, in detail. It consists of two consecutive steps, i.e., (1) updating all the variable nodes and (2) updating all the check nodes in the Tanner graph. This two-step updating process is iteratively applied until the maximum

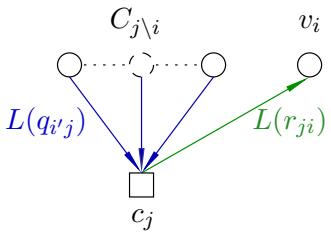


Figure 2.8: Update the message passing from the check node c_j to the variable node v_i.

number of iterations is reached or a codeword has been found, i.e., all the parity-check constraints are satisfied.

Assume the received channel value is y_i given x_i is transmitted over binary input additive white Gaussian noise (BI-AWGN) channel.[1] At the beginning of the flooding algorithm, the a priori LLR is assigned to the variable node v_i according to

$$L(x_i|y_i) = 4y_i\frac{E_s}{N_0} = 4y_i\frac{E_bR}{N_0}, \tag{2.23}$$

where $\frac{E_s}{N_0}$ is the signal-to-noise ratio (SNR) of the energy per symbol, $\frac{E_b}{N_0}$ is the SNR of the energy per information bit, and R is the code rate. Then, a message passing from the variable node v_i to its neighboring check node c_j denoted by $L(q_{ij})$, where $j \in V_i$ and V_i is the subscript index set of all the check nodes connecting to variable node v_i, is initialized by $L(x_i|y_i)$.

After initialization, we repeat the following three-step updating process:

1. update each check node, i.e., updating each message passing from a check node to a variable node by using

$$L(r_{ji}) = 2\tanh^{-1}\Big(\prod_{i'\in C_{j\backslash i}} \tanh\big(\frac{1}{2}L(q_{i'j})\big)\Big), \tag{2.24}$$

   where $L(r_{ji})$ represents the message passing from check node c_j to variable v_i, $L(q_{i'j})$ represents the message passing from variable node $v_{i'}$ to check node c_j, and $C_{j\backslash i}$ is, except for the variable node v_i, the index set of variable nodes connecting to the check node c_j. The Tanner graph representation of updating a check node is given in Fig. 2.8;

[1] Note that x_i is the value of the symbol, corresponding to the variable node v_i in the Tanner graph, after mapping in the modulator of the transmitter in Fig. 1.1 while y_i indicates the corrupted value of x_i by channel noise or interference.

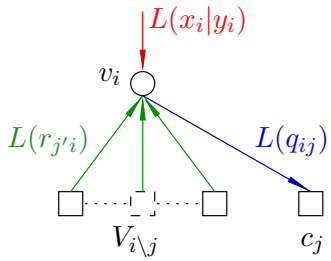


Figure 2.9: Update the message passing from the variable node v_i to the check node c_j.

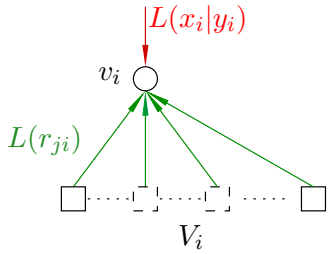


Figure 2.10: Take a decision for the variable node v_i.

2. Update each variable node, i.e., updating each message passing from a variable node to a check node by using

$$L(q_{ij}) = L(x_i|y_i) + \sum_{j' \in V_{i\backslash j}} L(r_{j'i}). \tag{2.25}$$

Similar explanations for the notations in (2.25) can be obtained by referring to those for (2.24). Fig. 2.9 illustrates the Tanner graph representation of updating a variable node. Note that when updating a variable node, the initialization message $L(x_i|y_i)$ is also involved. Steps (1) and (2) form one iteration of the decoding process;

3. At the end of each iteration, we take a decision for each variable node v_i by using the sign of the total LLR value

$$L(Q_i) = L(x_i|y_i) + \sum_{j \in V_i} L(r_{ji}). \tag{2.26}$$

It is illustrated in Fig. 2.10. Since we made the assumption of BPSK modulation, if the sign of $L(Q_i)$ is negative, we decode the transmitted bit x_i associated with

v_i as a "1"; if the sign of $L(Q_i)$ is positive, the transmitted bit x_i is decoded as a "0". The details of flooding algorithm is described in Algorithm 1.[1]

Algorithm 1 The flooding Algorithm

- Initialization: Set the maximum number of decoding iterations I and assign the priori probability to each variable node in the Tanner graph with respect to (2.23).
- Step 1: Update all the check nodes by passing all the messages from check nodes to corresponding variable nodes using (2.25).
- Step 2: Update all the variable nodes by passing all the messages from variable nodes to corresponding check nodes using (2.24).
- Step 3: According to (2.26), take the hard decision for each bit (variable node) in the sequence $\mathbf{v}$. If $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$ or the maximum number of iterations is reached, go to Step 4; else, go to Step 2.
- Step 4: Output the sequence $\mathbf{v}$ as the decoded codeword.

2.6.2 Decoding of LDPC-CCs

The syndrome former matrix $\mathbf{H}^T$ (see (2.2)) of an LDPC-CC is semi-infinite. This means in the Tanner graph there is an infinite number of variable nodes and check nodes. As a result, the flooding decoding algorithm cannot be directly applied to the Tanner graph. In 1999, Felström and Zigangirov [1] introduced the *pipeline decoder* for LDPC-CCs also called the *sliding window decoder*. The decoding process looks like a window of finite size sliding along the diagonal of the syndrome former matrix.

Given an LDPC-CC defined by (2.2) of rate $R = b/c$, the corresponding sliding window is of size $c(m_s+1)I \times p(2m_s+1)I$, where $p = c-b$, m_s is the syndrome former memory of the code and I is the number of iterations used in the decoder. The sliding window decoder consist of I processors, each one corresponds to one iteration of the decoding process. During the window sliding over the matrix, the I processors work in parallel and each variable node (or check node) sequentially passes through the I processors. It results in all the messages are updated by I times as each variable node (or check node) is updated by I times. The decoding complexity is dominated by the size of the sliding window. In other words, the decoding complexity is proportional to the value of the syndrome former memory m_s and the number of iterations I.

[1] Note that, in Algorithm 1 Step 1 and Step 2 can be switched.

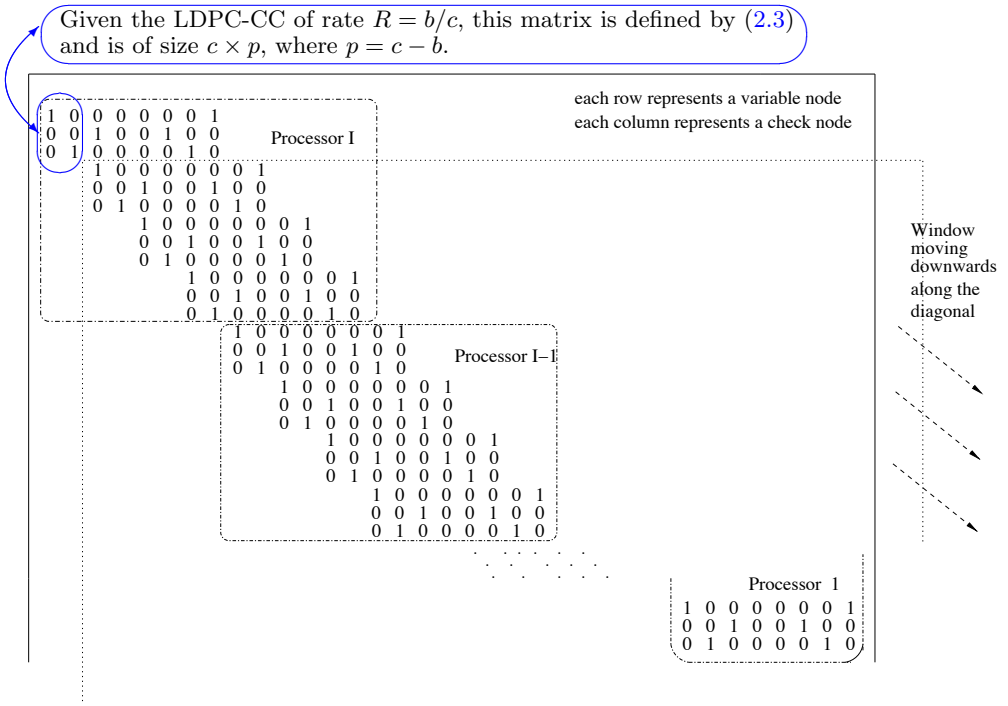


Figure 2.11: Sliding window decoder for the $(3, 2, 3)$ Tanner LDPC-CCs.

The sliding window decoder follows the first-in first-out (FIFO) rule. At every time unit, there are c received bits entering the decoder at the bottom right corner of the window, in the meanwhile, there are c decoded bits leaving the decoder from the top left corner of the window. Note that, only the first and last c rows in the sliding window are involved in the output (decoded) and input (received) bits. After each time unit, the sliding window moves downwards along the diagonal of the matrix $\mathbf{H}^T$. Specifically, it moves to the right and to the bottom of the syndrome former matrix by p and c symbols positions[1], respectively.

Example 2.6.1. To illustrate the sliding window decoder of LDPC-CCs, we use the $(3, 2, 3)$ rate $R = b/c = 1/3$ Tanner LDPC-CC whose time-domain syndrome former matrix is given in Fig. 2.4 as an example. The syndrome former memory of this code is $m_s = 3$. Assuming that the number of iterations is set to be I, the structure of the sliding window decoder overlapping the syndrome former matrix $\mathbf{H}^T$ is presented in Fig. 2.11. It is observed that the window contains I processors, each one is a matrix of size $c(m_s + 1) \times p(2m_s + 1) = 12 \times 14$ and represents one decoding iteration. At each

[1]Here one symbol means one entry "1" or "0" in the syndrome former matrix $\mathbf{H}^T$.

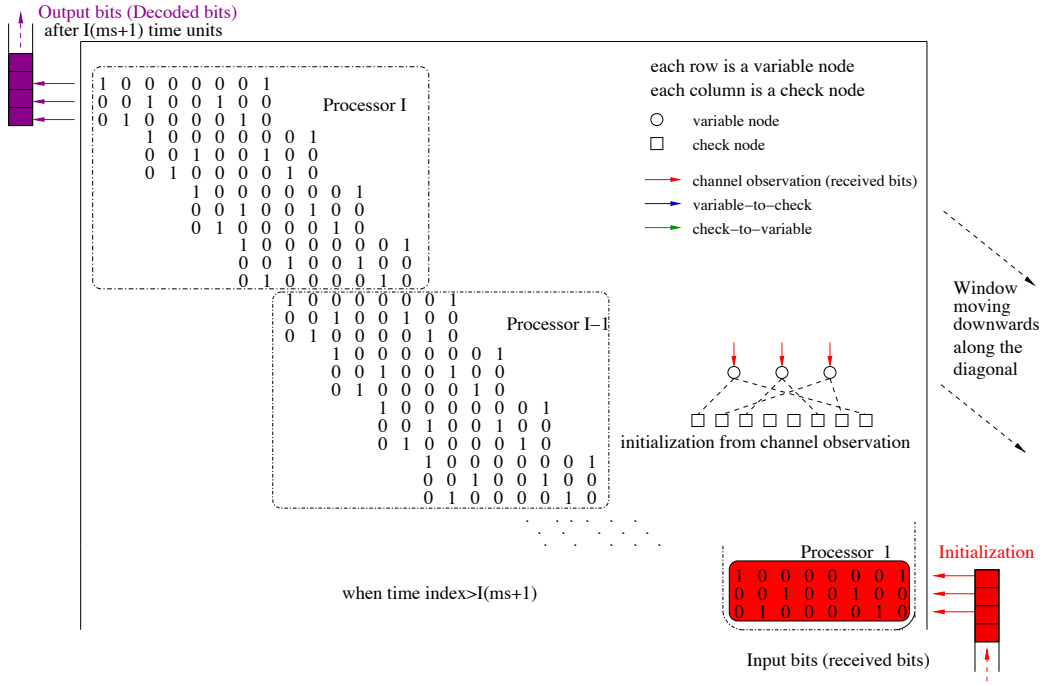


Figure 2.12: Initialization of the sliding window decoder.

time unit, the sliding window decoder processes the following three updating steps:

- *Initialize the sliding window decoder.*

  As shown in Fig. 2.12, in this case, at each time unit there are 3 received bits at the bottom right corner of the window. The corresponding submatrix in the window involved in the initialization step is highlighted in red in Fig. 2.12. The computation of the initialization of variable nodes is given in (2.23). Fig. 2.12 also shows the corresponding Tanner graph representation of the messages passing above the highlighted submatrix.

- *Update the corresponding messages from variable nodes to check nodes, i.e., update the corresponding variable nodes.*

  In Fig. 2.13, those columns in the sliding window involved in this step are highlighted in blue. Specifically, any column in the sliding window whose column index i satisfies $i \in [m_s q + 1 + j(m_s + 1)q, (m_s + 1)q + j(m_s + 1)q]$, where $i \in \mathbb{Z}^+$ and $j = 0, 1, 2, \ldots, I - 1$, participates in updating the messages from variable nodes to check nodes. In other words, the centering p columns in each processor

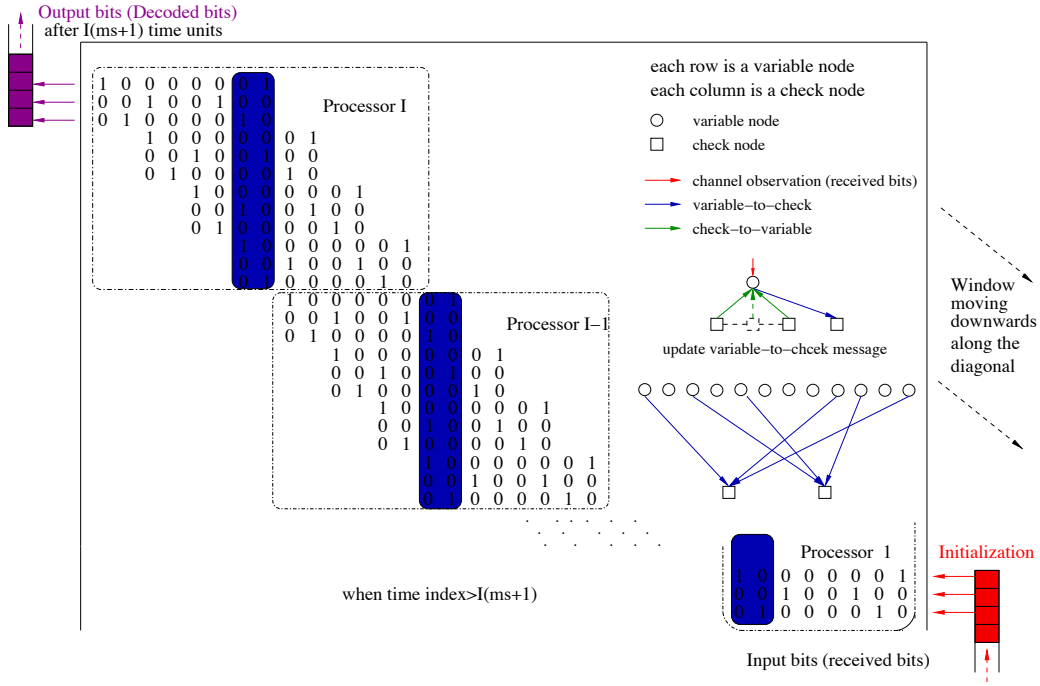


Figure 2.13: Partially update the variable nodes.

update messages with respect to (2.25). It is also shown in Fig. 2.13 the message propagation from the Tanner graph point of view.

- *Update the corresponding messages from check nodes to variable nodes, i.e., update the corresponding check nodes, and output c decoded bits at the top left corner of the sliding window.*

  This step contains two different types of message updating: (i) In the processors numbered from 1 to $I-1$, update messages from check nodes to variable nodes using (2.24). It refers to those rows in the sliding window with row index k lies in the range $k \in [c(m_s+1)+1+j(m_s+1)c, c(m_s+1)+c+j(m_s+1)c]$, where $k \in \mathbb{Z}^+$ and $j = 1, 2, \ldots, I-1$. It is colored in green in the sliding window in Fig. 2.14. (ii) However, in the Processor I, according to (2.26), take hard decisions for every $c = 3$ decoded bits as the output of the sliding window. It is colored in purple as shown in Fig. 2.14.

After the three-step updating, i.e., after one time unit, the sliding window moves to the right and to the bottom by $p = 2$ and $c = 3$ symbols, respectively. In the next time unit, another c received bit enter the sliding window at the bottom right corner

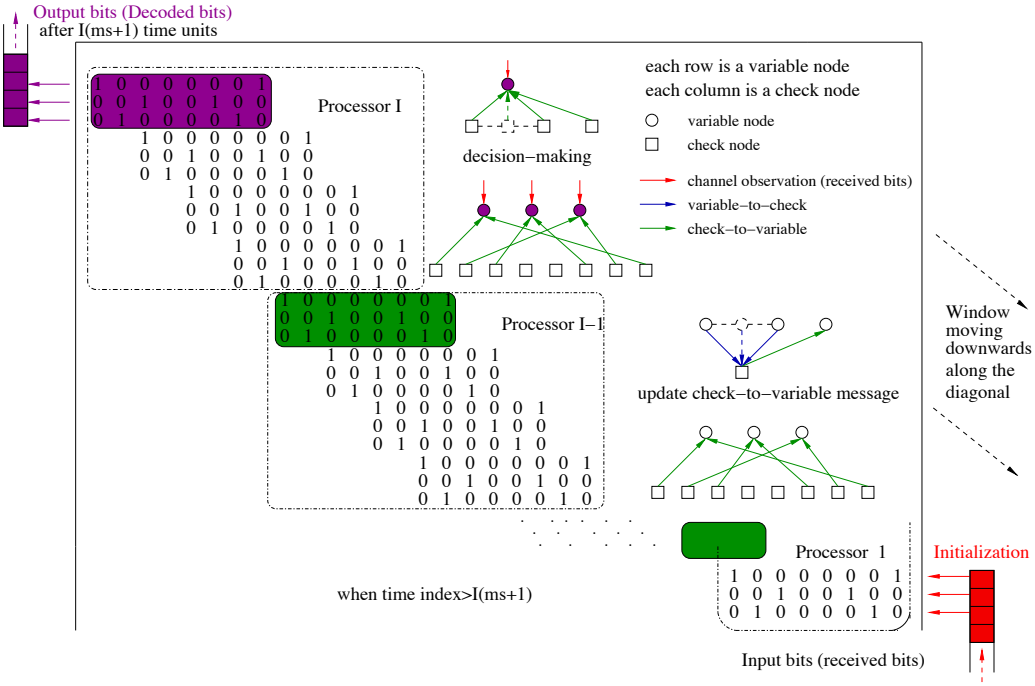


Figure 2.14: Partially update the check nodes and make decisions for the decoded bits.

of the window as input bits. Then it repeats the above three-step updating process. Considering the period starting with a bit (variable node) entering the window and ending with the bit leaving it, the bit (variable node) is totally updated by I times. ■

Note that, before the sliding window output decoded bits, there is a delay of $I(m_s+1)$ time units for the very beginning input bits to pass through the entire sliding window. After such delay, the sliding window decoder continuously produce c output bits at each time unit. Compared to LDPC-BC decoding, it significantly reduces the time of waiting for receiving the entire block length for every codeword.

Chapter 3

Cycle analysis of LDPC-CCs

The sum-product algorithm (SPA), also known as belief propagation, can be viewed as applying Bayes' rule locally and iteratively on the Tanner graph of LDPC codes to calculate approximate marginal a posteriori probabilities for these codes. If the Tanner graph has no cycles, then it can be proved that the SPA computes marginal posterior probabilities exactly. However, LDPC codes of interest contain cycles in the Tanner graphs and short cycles degrade decoding performance because a message sent by a node along a cycle propagates back to the original node after a small number of iterations, which destroys the assumption of dependence among messages of SPA. Specifically, short cycles prevent the SPA from converging, in addition, short cycles forming so-called *trapping sets* results in an error floor in the decoding performance. Intuitively, the larger the girth (length of the shortest cycle), the better the decoding performance will be. However, due to the finite block length and the practically motivated requirement for structured codes, for example, regular or partial regular quasi-cyclic (QC) LDPC codes, there are many cycles not avoidable.

In order to improve decoding performance, there have been lots of efforts to construct LDPC codes with large girth. Fossorier [22] discussed the cycle formations in QC LDPC-BCs that are made up of a set of circulant permutation matrices. Large girth QC LDPC codes were presented in [22–25] from an algebraic-structure point-of-view. Some optimization and greedy search algorithms were used in [26, 27] to find QC LDPC-BCs with large girth. Recently, Esmaeili *et al.* [28] introduced four classes of maximum-girth geometrically structured column-weight-two regular QC LDPC-BCs. Other work on the constructions of QC LDPC-BCs with girth at least six can be found in [29, 30]. A cycle counting algorithm was introduced in [31] to count short cycles in protograph based QC LDPC-BCs. Based on the message passing in the Tanner graphs

of QC protograph-based LDPC codes, an efficient method for counting short cycles was presented in [32].

Low-density parity-check *convolutional codes* (LDPC-CCs) were first proposed in [1] and some construction are based on QC LDPC-BCs. Using pipeline decoding [2], it has been shown that they are suitable for practical implementation with continuous transmission as well as, via encoder termination, block transmission in frames of arbitrary size [3] without an increase in computational complexity compared to their block code counterparts. In this chapter we analyze the cycle formation in LDPC-CCs so as to design good LDPC-CCs containing no short cycles.

The work of this chapter were published in [14, 33–35].

3.1 Cycles in time-invariant LDPC-CCs

A cycle in an LDPC-CC consists of a set of edges and vertices. Vertices refer to the entries "1" in the time-domain syndrome former matrix $\mathbf{H}^T$ or monomials[1] in the polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$ that are connected by vertical and horizontal edges. Correspondingly, in the Tanner graph, vertices refer to variable nodes and check nodes, and they are alternatively connected by edges. If we define the "delay" as the difference $t - t'$ between the subscripts of two variable nodes $v_t^{(j)}$ and $v_{t'}^{(j')}$, then

- a horizontal edge in $\mathbf{H}^T$ corresponds to a connection between two monomials in the same row of $\mathbf{H}^T(D)$: such a connection links two check nodes via a variable node in the Tanner graph representation, there is no "delay" incurred as there is only one variable node;
- a vertical edge in $\mathbf{H}^T$ corresponds to a connection between two monomials in the same column of $\mathbf{H}^T(D)$: such a connection links two variable nodes via a check node, the "delay" incurred equals the difference between the "powers" of the two connected monomials.

Alternatively, if we considers the "delay" with respect to check nodes, we can also define that a horizontal connection results in a "delay" equals the difference between the "powers" of the two connected monomials, while a vertical connection results in no "delay". Note that when a path forms a cycle, either the accumulated "delays" between variable nodes or between check nodes is zero. To simplify the description, we only need to consider one of them. Here we work on the formation of cycles based on the former one.

[1] A monomial is essentially denoted as D^l with the integer $l \geq 0$.

Due to the semi-infinite size of the time-domain matrix $\mathbf{H}^T$ of LDPC-CCs and the repeated identical structure of cycles, it is difficult to analyze the edge connections in the cycles. Instead, we use the polynomial-domain matrix $\mathbf{H}^T(D)$ of finite size to form the cycles. The formal definition of the delay denoted by $\Delta(s_{ij}^p, s_{kl}^q)$ between two monomials s_{ij}^p and s_{kl}^q in $\mathbf{H}^T(D)$ is given by

$$\Delta(s_{ij}^p, s_{kl}^q) \doteq \Delta(s_{ij}^p, s_{il}^{q'}) + \Delta(s_{il}^{q'}, s_{kl}^q), \tag{3.1}$$

where $i \neq k$, $j \neq l$ and s_{ij}^p indicates the p-th monomial in the polynomial entry with row and column indices i and j in $\mathbf{H}^T(D)$, respectively.[1] Similar explanation applies to $s_{il}^{q'}$ and s_{kl}^q. According to (3.1), a possible move from one monomial s_{ij}^p to any other monomial s_{kl}^q is decomposed into a horizontal move $s_{ij}^p \to s_{il}^{q'}$ with zero delay, i.e.,

$$\Delta(s_{ij}^p, s_{il}^{q'}) \doteq 0 \tag{3.2}$$

and a vertical move $s_{il}^{q'} \to s_{kl}^q$ with the delay

$$\Delta(s_{il}^{q'}, s_{kl}^q) \doteq s_{kl}^{q'} - s_{il}^q \,. \tag{3.3}$$

Note that the delay $\Delta(s_{il}^{q'}, s_{kl}^q)$ for the vertical move has a sign, which identifies whether the path moves "forward" or "backward" in time when progressing along a vertical edge. When we form a path through the polynomial syndrome former matrix, we use the delay noted on the branches when the path has the same direction as indicated by the "arrow", and we invert the sign when traverse along a branch in opposite direction.

A path $\mathcal{P}$ – which is a sequence of pairs of monomials from the polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$ – forms a cycle of length $2L$, when

$$\sum_{\forall\{s,s'\}\in\mathcal{P}} \Delta(s, s') = 0 \tag{3.4}$$

[1] Note throughout the dissertation monomials involved in a polynomial entry are placed in descending order. For example, in the polynomial entry D^3+1, D^3 and 1 are the first and second monomials, respectively.

with the path given by

$$\mathcal{P} = \Big\{ \underbrace{\{\overbrace{s_{j_1k_1}^{p_1}}^{\text{start}}, s_{j_1k_2}^{q_1}\}}_{\text{horizontal move}}, \underbrace{\{s_{j_1k_2}^{q_1}, s_{j_2k_2}^{p_2}\}}_{\text{vertical}}, \underbrace{\{s_{j_2k_2}^{p_2}, s_{j_2k_3}^{q_2}\}}_{\text{horizontal}}, ... \\ \underbrace{\{s_{j_{L-1}k_{L-1}}^{p_{L-1}}, s_{j_{L-1}k_L}^{q_{L-1}}\}}_{\text{horizontal}} \underbrace{\{s_{j_{L-1}k_L}^{q_{L-1}} = s_{j_{L-1}k_1}^{q_{L-1}}, \overbrace{s_{j_Lk_L}^{p_L} = s_{j_1k_1}^{p_1}}^{\text{end}}\}}_{\text{vertical}} \Big\} \quad (3.5)$$

where $j_x \in \{1, ..., j\}$, $k_y \in \{1, ..., k\}$, $p_x \in \mathbb{Z}^+$, $q_y \in \mathbb{Z}^+$, and $x \in \{1, ..., L\}$ and $y \in \{1, ..., L\}$. It is important to note that the first and the last element in the path have to be the same in order for $\mathcal{P}$ to form a cycle. This condition is, however, only necessary but not sufficient, as in addition the sum of the delay in (3.4) must also be zero. Moreover, the elements on the path $\mathcal{P}$ need not to be distinct because the same monomials (taken from $\mathbf{H}^T(D)$) on the path could represent different entries of "1" in the time-domain syndrome former matrix $\mathbf{H}^T$ (due to different total delays on the path). In this chapter, $D^{s_{ij}^p}$ and s_{ij}^p are alternatively used to denote a monomial.[1]

Note that a path in $\mathbf{H}^T(D)$ of size $c \times (c - b)$ satisfying the conditions in (3.4) and (3.5) forms a type of cycles of length $2L$: it rather represents a set of cycles in the time-domain syndrome former matrix $\mathbf{H}^T$ that have the same structure and are periodically shifted by c and $c - b$ entry (symbol) positions in the row and column of $\mathbf{H}^T$, respectively. We consider this set of cycles as one type and counted only once when computing the number of cycles in LDPC-CCs. This is shown as an example in Figs. 3.1 and 3.2.

To simplify the description of cycles in LDPC-CCs, we use the example code defined by (2.14) to illustrate the formation of cycles in time-invariant LDPC-CCs. Given the polynomial syndrome former matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix}, \quad (3.6)$$

we obtain the time-domain syndrome former matrix and a cycle of length 12 in Fig. 3.1. The same cycle is also depicted in the corresponding polynomial matrix $\mathbf{H}^T(D)$ in Fig. 3.3, where we have removed the common factor D^2 from the right column. Re-

[1] Note that, if the polynomial entry with row and column indices i and j in $\mathbf{H}^T(D)$ is a monomial, it is denoted by s_{ij}^1 or simply s_{ij}.

moving such common factor leads to smaller syndrome former memory and reduces decoding complexity. In the polynomial syndrome former matrix we can reach from a given monomial to any other monomial by one vertical and one horizontal move with some delay (see Fig. 3.4 for an illustration). When we form a path in $\mathbf{H}^T(D)$, we use the delay noted on the branches when the path has the same direction as indicated by the "arrow" in Fig. 3.4, and we invert the sign when traverse along a branch in the opposite direction.

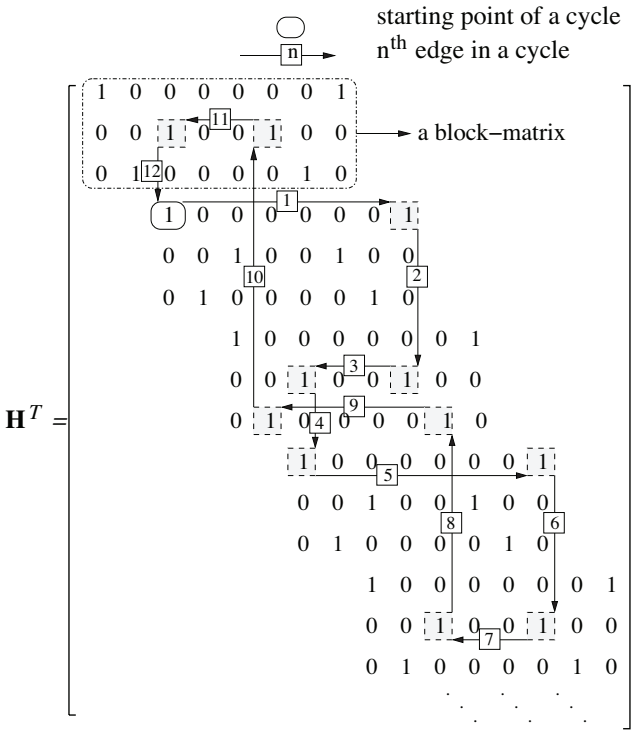


Figure 3.1: A cycle of length 12 in the time-domain syndrome former matrix $\mathbf{H}^T$ of the time-invariant LDPC-CC in (3.6)

The cycle of length 12 in the time-domain syndrome former matrix $\mathbf{H}^T$ in Fig. 3.1, it is also described by a "closed path" in the polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$ in Fig. 3.3. We apply the delays indicated in Fig. 3.4 to the path and calculate

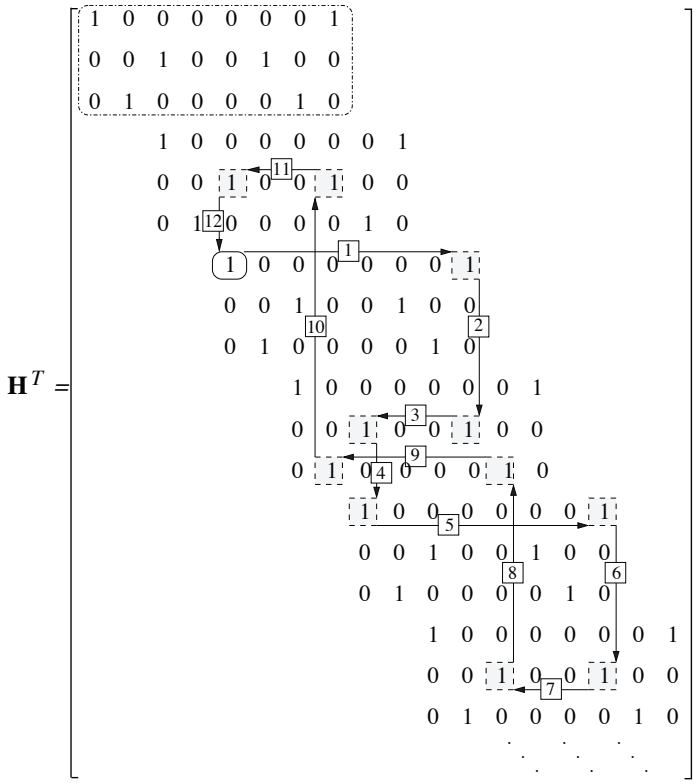


Figure 3.2: The periodically shifted cycle of the one in Fig. 3.1

the sum of the delays which are accumulated along the path as:

$$\sum_{\forall\{s,s'\}\in\mathcal{P}} \Delta(s,s') = \left\{ \begin{array}{lcll} \overbrace{\Delta(s^1_{11},s^1_{12})}^{\text{horizontal moves}} & + & \overbrace{\Delta(s^1_{12},s^1_{22})}^{\text{vertical moves}} & + \\ \Delta(s^1_{22},s^1_{21}) & + & \Delta(s^1_{21},s^1_{11}) & + \\ \Delta(s^1_{11},s^1_{12}) & + & \Delta(s^1_{12},s^1_{22}) & + \\ \Delta(s^1_{22},s^1_{21}) & + & \Delta(s^1_{21},s^1_{31}) & + \\ \Delta(s^1_{31},s^1_{32}) & + & \Delta(s^1_{32},s^1_{22}) & + \\ \Delta(s^1_{22},s^1_{21}) & + & \Delta(s^1_{21},s^1_{11}) & \end{array} \right. . \tag{3.7}$$

The path consists of 6 horizontal and 6 vertical moves. With respect to (3.2) and (3.3), we have "zero" delay for all horizontal moves and certain delay for the vertical moves.

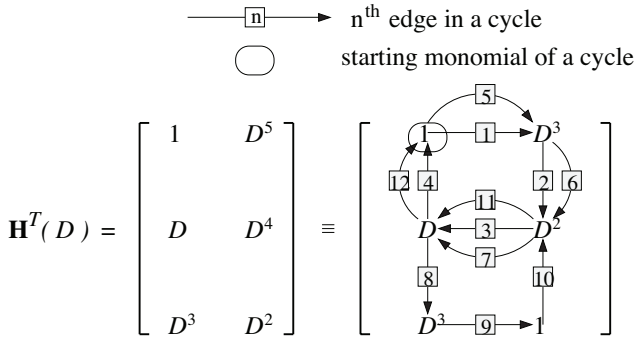


Figure 3.3: Polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$ of the time-invariant LDPC-CC in (3.6)

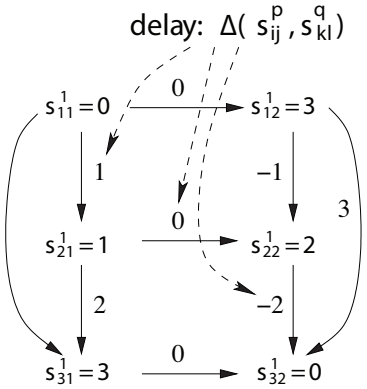


Figure 3.4: Delays for code example derived from syndrome former matrix

As a result, we obtain

$$\sum_{\forall\{s,s'\}\in\mathcal{P}} \Delta(s,s') = \begin{cases} 0 + (s_{22}^1 - s_{12}^1) + \\ 0 + (s_{11}^1 - s_{21}^1) + \\ 0 + (s_{22}^1 - s_{12}^1) + \\ 0 + (s_{31}^1 - s_{21}^1) + \\ 0 + (s_{22}^1 - s_{32}^1) + \\ 0 + (s_{11}^1 - s_{21}^1) \end{cases}$$

If we plug in all the delays from the example in Fig. 3.4 we have

$$\sum_{\forall\{s,s'\}\in\mathcal{P}} \Delta(s,s') = \left\{ \begin{array}{ll} (2-3) & + \\ (0-1) & + \\ (2-3) & + \\ (3-1) & + \\ (2-0) & + \\ (0-1) & \end{array} \right\} = 0, \tag{3.8}$$

which confirms the condition in (3.13). Hence this cycle of length 12 does indeed exist. In Fig. 3.2, we show the periodically shifted version of the cycle in Fig. 3.1. These two cycles have the same description in $\mathbf{H}^T(D)$. As we move along the diagonal of $\mathbf{H}^T$, there are infinite number of such cycles. Therefore, when computing the cycle enumerators, they are considered as one type and are just counted once.

3.2 Cycles in time-varying LDPC-CCs

In this section, we analyze the cycle structures in time-varying LDPC-CCs which have been discussed in Section 2.4. For the simplicity of edge connections in cycles, we use an auxiliary polynomial matrix denoted by $\mathbf{H}_A^T(D)$ derived from $\mathbf{H}^T(D)$ in (2.6) as

$$\mathbf{H}_A^T(D) = \begin{bmatrix} \mathbf{H}_1^T(D) \\ \mathbf{H}_2^T(D) \\ \vdots \\ \mathbf{H}_i^T(D) \\ \vdots \\ \mathbf{H}_T^T(D) \end{bmatrix}, \tag{3.9}$$

where $\mathbf{H}_i^T(D)$ is the i-th polynomial sub-matrix in $\mathbf{H}^T(D)$ of a time-varying LDPC-CC with period T. Similar to time-invariant LDPC-CCs, two monomials with powers $s_{ij}^p(t)$ and $s_{ij'}^q(t)$ from the same row of the polynomial sub-matrix in (3.9) of a time-varying LDPC-CC connect to each other without delay, i.e.,

$$\Delta(s_{ij}^p(t), s_{ij'}^q(t)) \doteq 0 \tag{3.10}$$

where $s_{ij}^p(t)$ refers to the power of the monomial in the polynomial entry, which is in the i-th row and j-th column of the t-th polynomial sub-matrix $\mathbf{H}_t^T(D)$, while p indicates

the p-th monomial of this polynomial entry.

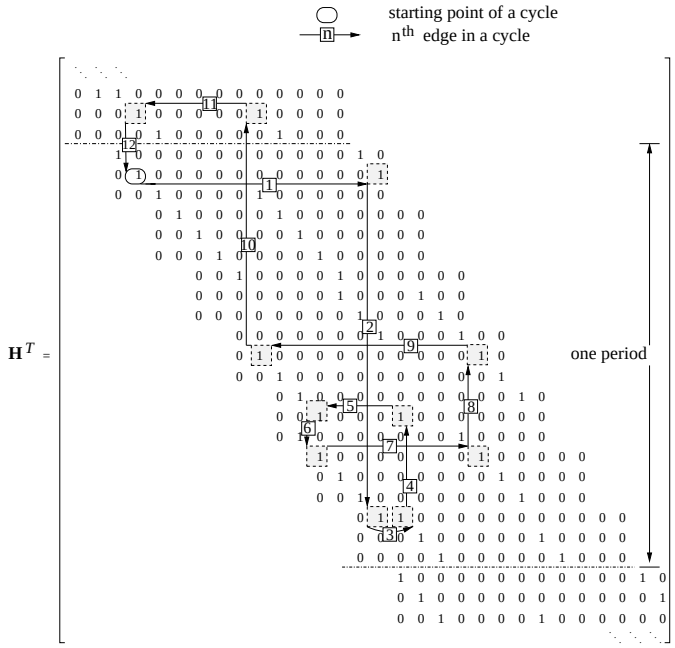


Figure 3.5: A cycle of length 12 in the time-domain syndrome former matrix of the time-varying LDPC-CC derived by unwrapping the QC LDPC-BC in (2.14)

In time-invariant LDPC-CCs, monomials in the same column are always connected. However, for a time-varying LDPC-CC with period T, two monomials with powers $s_{ij}^{p}(t)$ and $s_{i'j}^{q}(t')$ in the same column of $\mathbf{H}_{A}^{T}(D)$ are connected if and only if

$$(t - t' + s_{ij}^{p}(t) - s_{i'j}^{q}(t')) \mod T = 0, \tag{3.11}$$

and the delay is given by

$$\Delta(s_{ij}^{p}(t), s_{i'j}^{q}(t')) \doteq s_{i'j}^{q}(t') - s_{ij}^{p}(t) \; . \tag{3.12}$$

Note that t could be equal to t' for time-varying LDPC-CCs. This condition is also suitable for time-invariant LDPC-CCs because for arbitrary two monomials in time-invariant LDPC-CCs, they have $t = t'$ and $T = 1$. Thus the condition in (3.11) is always satisfied.

A path $\mathcal{P}_v$ in $\mathbf{H}^{T}(D)$ of a time-varying LDPC-CC – which is a sequence of pairs of monomials from the auxiliary polynomial syndrome former matrix $\mathbf{H}_{A}^{T}(D)$– forms a

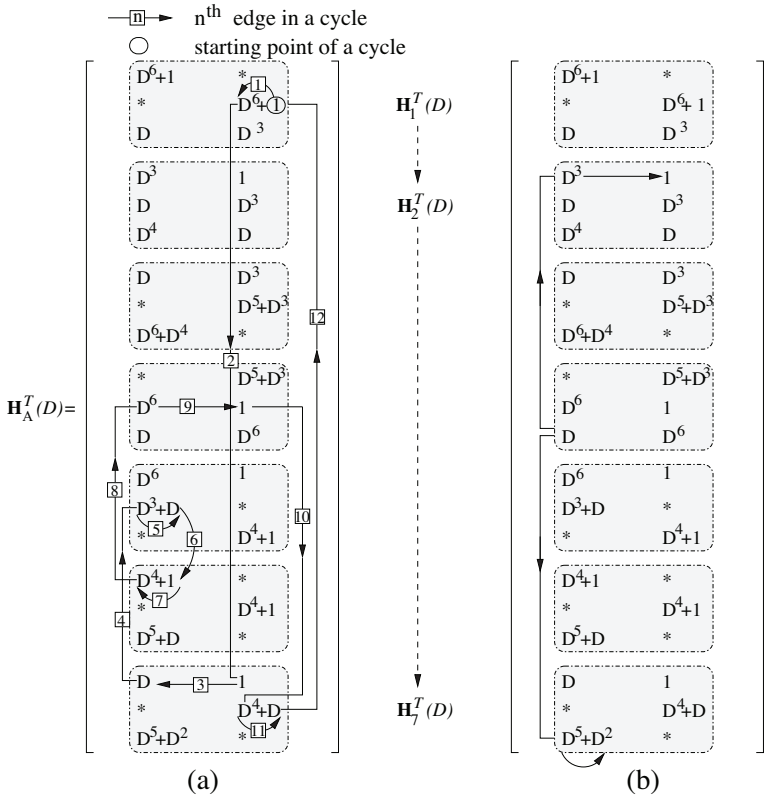


Figure 3.6: (a) A cycle of length 12 in the auxiliary polynomial-domain syndrome former matrix of the obtained time-varying LDPC-CC in Fig. 3.5; (b) An illustration for the connectivity between monomials in the auxiliary polynomial-domain syndrome former matrix.

cycle of length $2 \times L$, when

$$\sum_{\forall\{s,s'\}\in\mathcal{P}_v} \Delta(s, s') = 0 \tag{3.13}$$

with the path given by

$$\mathcal{P}_v = \{ \underbrace{\{\overbrace{s_{i_1 j_1}^{p_1}(t_1)}^{\text{start}}, s_{i_1 j_2}^{q_1}(t_1)\}}_{\text{horizontal}}, \underbrace{\{s_{i_1 j_2}^{q_1}(t_1), s_{i_2 j_2}^{p_2}(t_2)\}}_{\text{vertical}}, \ldots \underbrace{\{s_{i_{L-1} j_{L-1}}^{p_{L-1}}(t_{L-1}), s_{i_{L-1} j_L}^{q_{L-1}}(t_{L-1})\}}_{\text{horizontal}} \underbrace{\{s_{i_{L-1} j_L = j_1}^{q_{L-1}}(t_{L-1}), \overbrace{s_{i_1 j_1}^{p_L = p_1}(t_L = t_1)}^{\text{end}}\}}_{\text{vertical}} \} \tag{3.14}$$

with $i_x \in \{1, ..., c\}$, $j_y \in \{1, ..., c-b\}$, $t_x \in \{1, ..., T\}$, $p_x \in \mathbb{Z}^+$, $q_y \in \mathbb{Z}^+$ and $x \in \{1, ..., L\}$ and $y \in \{1, ..., L\}$. As for time-*invariant* LDPC-CCs, the first and the last elements in the path have to be the same in order for $\mathcal{P}_v$ to form a cycle.

To illustrate the above concept, we use a time-varying LDPC-CC derived from a QC LDPC-BC to show the cycle structure. Apply the unwrapping process to the QC LDPC-BC in (2.14) with k being set to one, we have one period of the time-domain one period syndrome former matrix of a time-varying LDPC-CC and a cycle of length 12 in Fig. 3.5. Applying formulas (2.6) and (2.7) to the matrix in Fig. 3.5, we obtain the auxiliary polynomial syndrome former matrix $\mathbf{H}_A^T(D)$ and the corresponding 12-cycle mapping from Fig. 3.5 to Fig. 3.6(a). The auxiliary polynomial syndrome former matrix contains seven polynomial sub-matrices, i.e., period $T = 7$, and each is of size 3×2. According to (3.13) and (3.14), the accumulated delay of the path in Fig. 3.6(a) is given as

$$\sum_{\forall\{s,s'\}\in\mathcal{P}_v} \Delta(s, s') = \left\{ \begin{array}{llll} \overbrace{\Delta(s_{22}^2(1), s_{22}^1(1))}^{\text{horizontal moves}} & + & \overbrace{\Delta(s_{22}^1(1), s_{12}^1(7))}^{\text{vertical moves}} & + \\ \Delta(s_{12}^1(7), s_{11}^1(7)) & + & \Delta(s_{11}^1(7), s_{21}^1(5)) & + \\ \Delta(s_{21}^1(5), s_{21}^2(5)) & + & \Delta(s_{21}^2(5), s_{11}^2(6)) & + \\ \Delta(s_{11}^2(6), s_{11}^1(6)) & + & \Delta(s_{11}^1(6), s_{21}^1(4)) & + \\ \Delta(s_{21}^1(4), s_{22}^1(4)) & + & \Delta(s_{22}^1(4), s_{22}^1(7)) & + \\ \Delta(s_{22}^1(7), s_{22}^2(7)) & + & \Delta(s_{22}^2(7), s_{22}^2(1)) & \end{array} \right.$$

With (3.10) we have "zero" delay for all horizontal moves and we use (3.12) for the

vertical moves. We obtain

$$\sum_{\forall\{s,s'\}\in\mathcal{P}_v} \Delta(s,s') = \left\{ \begin{array}{lllll} 0 & + & (0-6) & + \\ 0 & + & (3-1) & + \\ 0 & + & (0-1) & + \\ 0 & + & (6-4) & + \\ 0 & + & (4-0) & + \\ 0 & + & (0-1) & \end{array} \right\} = 0$$

which confirms the condition in (3.13): this 12-cycle does indeed exist (see Fig. 3.5).

3.3 An algorithm for counting cycles

In Sections 3.1 and 3.2, we have discussed how cycles are formed for time-invariant and time-varying LDPC-CCs both in the time- and polynomial-domains. In this section, we introduce a cycle counter algorithm to track cycles in an LDPC-CC given its polynomial syndrome former matrix. Given a "starting" monomial as shown in Fig. 3.3 and Fig. 3.6(a), this algorithm extends all the possible two consecutive edges (vertical and horizontal) with delays defined in (3.10) and (3.12); we consider these two path extensions as "one" iteration. The connectivity of monomials in the same column of the auxiliary polynomial syndrome former matrix is given in (3.11) for time-varying LDPC-CCs.

Before introducing the algorithm, the notation is defined as follows:

- $s_{ij}^{p}(t)$: the power of the p-th monomial in the polynomial entry which is in the i-th row and the j-th column of the polynomial sub-matrix $\mathbf{H}_t^T(D)$. We refer to $s_{ij}^{p}(t)$ as "a monomial" or "the power of a monomial", alternatively.
- $\mathcal{N}_v(s_{ij}^{p}(t))$: all the "neighbors" vertically connected to the monomial $s_{ij}^{p}(t)$ in $\mathbf{H}^T(D)$, a "neighbors" means they are connected according to (3.11).
- $\mathcal{N}_h(s_{ij}^{p}(t))$: all the "neighbors" horizontally connected to the monomial $s_{ij}^{p}(t)$ in $\mathbf{H}^T(D)$.
- $\mathbf{N}_t\{i,j\}\{1,p\}$: current accumulated delays (accumulated powers' sum (APS)) along all the paths temporarily intermittent at the monomial $s_{ij}^{p}(t)$ [14].
- $\mathcal{W}(\mathbf{N}_t\{i,j\}\{1,q\})$: number of paths temporarily intermittent at the monomial $s_{ij}^{q}(t)$.

- **S**: the set of starting monomials; for each iteration, after extending all the possible two consecutive paths (vertical and horizontal) from the starting monomials, the set of starting monomials is redefined as those monomials that all the paths temporarily end with in this iteration. In the following, we give an example to show the updating of starting monomials.

The main challenge of this cycle counter algorithm lies in generating a register, which informs a monomial of its connected neighboring monomials with respect to (3.11). Each iteration consists of two-step updates. Firstly, update the set of starting monomials by vertically extending the paths to all their connectible neighboring monomials. Afterwards, update the monomials that currently all the paths are intermittent at by extending paths horizontally. At the end of each iteration, the path extension history is cleared for this iteration and then the set of starting monomials is redefined as those monomials that all the horizontally extended paths currently end with. The process runs iteratively until the maximum testing cycle length is achieved; full details can be found in Algorithm 2.

As an example shown in Fig. 3.6(b), we select $s_{31}^{1}(4)=D$ as the starting monomial in the first iteration, i.e., $\mathbf{S}=s_{31}^{1}(4)$. By checking (3.11), $s_{31}^{1}(4)$ has two available neighboring monomials. First of all, the cycle counter algorithm updates monomial $s_{31}^{1}(4)$ by extending the two vertical available connections to $s_{11}^{1}(2)$ and $s_{31}^{1}(7)$, respectively. Then it updates monomials $s_{11}^{1}(2)$ and $s_{31}^{1}(7)$ by extending paths to $s_{12}^{1}(2)$ and $s_{31}^{2}(7)$. Before starting the next iteration, the set of starting monomials is renewed as the monomials all the paths end with, i.e., $\mathbf{S}=\{s_{12}^{1}(2), s_{31}^{2}(7)\}$, and the previous search history is cleared.

In the first iteration of Step 2 in Algorithm 2, we test every monomial in $\mathbf{H}^{T}(D)$ as a starting monomial, which results in repetitious tracking of a cycle. In addition, short cycles with length i in LDPC-CCs usually have i distinct vertices in the time-domain syndrome former matrix. Therefore, C_i/i in Step 3 gives the exact number of short cycles. However, for larger i it is possible that a cycle of length i consists of less than i distinct vertices when a cycle contains smaller inner cycles. This explains the ceil manipulation $\lceil\rceil$ in Step 3 of the algorithm.

We applied the cycle counter algorithm to some time-invariant LDPC-CCs derived from the corresponding (J=3,K=5) QC LDPC-BCs and time-varying LDPC-CCs derived from unwrapping (J=3,K=5) QC LDPC-BCs in [5] with unwrapping step size $(c-b)\times c$. The cycle enumerators of these time-invariant and time-varying LDPC-CCs are shown in Table 3.1 with the number of cycles for time-varying LDPC-CCs normalized by period T (details are given in [14, 33]), which makes sense to compare cycle

Algorithm 2 Cycle counter algorithm for time-varying LDPC-CCs

- Step 1: Initialization
  1. Define the maximum cycle length as $2 \times L$ for examining and the initial path length as $l = 0$.
  2. Generate a $KT \times J$ empty cell $\mathbf{N}\{\}$ used to update the accumulative powers' sum (APS) and create a cycle counter C_i recording number of cycles with different lengths i, $i = 4, 6, 8, ..., 2 \times L$.
  3. Generate a monomials' "connectivity cell", which is used as a reference for $\mathcal{N}_v(s_{ij}^p(t))$ to indicate the indices of neighbors of $s_{ij}^p(t)$ in the polynomial syndrome former.
- Step 2: Main function.

for each monomial $s_{i^*j^*}^{q^*}(t^*)$ in $\mathbf{H}^T(D)$ **do**
  $\mathbf{S}$=$s_{i^*j^*}^{q^*}(t^*)$; $\mathbf{N}_{t^*}\{i^*, j^*\}\{1, q^*\}$=0;
  while $l < L$ **do**
    for each $s_{ij}^q(t) \in \mathbf{S}$ **do**
      %vertical path extensions
      for each $s_{i'j}^p(t') \in \mathcal{N}_v(s_{ij}^q(t))$ **do**
        if $\mathbf{N}_t\{i, j\}\{1, q\}$=$\varnothing$ **then**
          $\mathbf{N}_t\{i, j\}\{1, q\}$=0;
        end if
        k=$\mathcal{W}(\mathbf{N}_t\{i, j\}\{1, q\})$;
        $\mathbf{T_1}$=$s_{i'j}^p(t')$-$s_{ij}^q(t)$+$\mathbf{N}_t\{i, j\}\{1, q\}(1{:}k)$;
        $\mathbf{N}_{t'}\{i', j\}\{1, p\}$=$[\mathbf{N}_{t'}\{i', j\}\{1, p\}|\mathbf{T_1}]$;
        %horizontal path extensions
        for each $s_{i'j'}^r(t') \in \mathcal{N}_h(s_{i'j}^p(t'))$ **do**
          if $i' \neq j'|r \neq p$ **then**
            $\mathbf{T_2}$=$\mathbf{N}_{t'}\{i', j\}\{1, p\}$(end-$k$+1:end);
            $\mathbf{N}_{t'}\{i', j'\}\{1, r\}$=$[\mathbf{N}_{t'}\{i', j'\}\{1, r\}|\mathbf{T_2}]$;
            $\mathbf{S}$=$\mathbf{S} \cup s_{i'j'}^r(t')$;
          end if
        end for
        %clear history of horizontal path extensions
        $\mathbf{N}_{t'}\{i', j\}\{1, p\}$(end-$k$+1:end)=[];
      end for
      $\mathbf{S}$=$\mathbf{S} \cap \overline{s_{ij}^q(t)}$; %renew starting monomials
      %clear history of vertical path extensions
      $\mathbf{N}_t\{i, j\}\{1, q\}(1{:}k)$=[];
    end for
    $\mathbf{S}$=unique($\mathbf{S}$); %remove repetitive monomials
    Calculate zeros in $\mathbf{N}_{t^*}\{i^*, j^*\}\{1, q^*\}$;
    l=l+2; C_l=C_l+n;
  end while
end for

- Step 3: Process cycle counter C_i.
  The number of cycles with length-i in this syndrome former is defined as $\lceil C_i/i \rceil$. C_i is the number of cycles for length-i.

Table 3.1: Cycle enumerators of LDPC-CCs

time-invariant				time-variant		
m/m_s	8-cycle		10-cycle		12-cycle	
31/21	11	4.55	62	42.39	351	228.77
61/57	0	0	21	15.07	148	118.06
181/134	0	0	0	0	67	46.35
211/187	0	0	0	0	68	54.57
241/204	0	0	0	0	52	44.83

Note that, common factors have been removed from the polynomial-domain syndrome former matrices for time-invariant LDPC-CCs. C_i is normalized for time-varying LDPC-CCs, i.e., C_i/T, where T=m, due to unwrapping size k=1.

properties with those of time-invariant LDPC-CCs.

In Table 3.1, m_s is the syndrome former memory of the code, m is the size of circulant matrix indicating which QC LDPC-BC the corresponding LDPC-CC is derived from (see [5]), and n-cycles represent cycles of length n. Columns in dark gray and light gray correspond to the cycle enumerators of time-varying and time-invariant LDPC-CCs, respectively.

Generally speaking (and as expected), time-varying LDPC-CCs are superior to time-invariant ones with respect to the small number of short cycles.

3.4 Unavoidable cycles in time-invariant LDPC-CCs

Recalling the description in Section 3.1, it seems that cycles can be eliminated by choosing suitable power index for each monomial entry in $\mathbf{H}^T(D)$ that cannot enable any path to satisfy the condition in (3.4). However, we find that there are some unavoidable cycles, no matter what the "powers" of the monomials in the polynomial-domain syndrome former matrix are. To better understand the existence of these cycles, we illustrate the unavoidable cycles with lengths ranging from 6 to 20 from a geometric perspective.

3.4.1 Unavoidable cycles of length 6, 8, 10, and 12

Time-invariant LDPC-CCs derived from corresponding QC LDPC-BCs in Chapter 2 only consist of monomial entries (i.e., their weight is one) in the polynomial syndrome former matrix $\mathbf{H}^T(D)$. In this section, cycle properties are discussed for LDPC-CCs with entry weight larger than one in $\mathbf{H}^T(D)$. In [15], two girth theorems have been

arithmetically proved for the case that the weight of any polynomial entry is larger than one. Rather than from arithmetical point of view, we analyze the geometrical structure of short unavoidable cycles in the polynomial syndrome former matrix.

Property 1. If the weight of any polynomial entry in the polynomial-domain syndrome former matrix is larger than two then the girth of the corresponding LDPC-CC has at most girth 6.

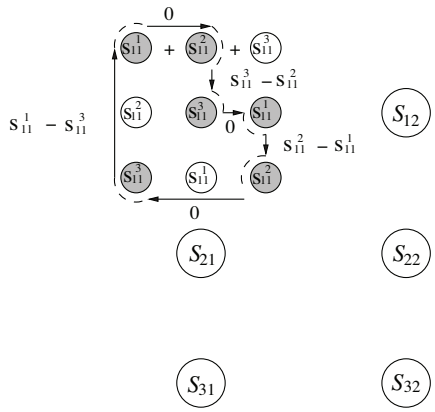


Figure 3.7: Unavoidable cycles of length 6 in $\mathbf{H}^T(D)$ with an entry of weight three

An unavoidable cycle of length 6 is shown in Fig. 3.7: the small circles arranged in the 3×3 block represent the polynomial entry $D^{s^1_{11}} + D^{s^2_{11}} + D^{s^3_{11}}$. In other words, the polynomial entry in the first row and the first column of $\mathbf{H}^T(D)$ consists of three monomials. Considering the property that monomials in the same row or column connect to each other with delays (3.2) or (3.3), it can be applied to this weight-three polynomial entry as well. For the convenience of the graphical representation of the connections, $D^{s^1_{11}} + D^{s^2_{11}} + D^{s^3_{11}}$ is arranged in the form of a "matrix" of size 3×3. Filled circles are the vertices involved in a cycle. After summing up all the delays along the path, delays involved in the path are cancelled by "themselves", and according to (3.4), we hence have a cycle. Fig. 3.7, therefore, shows that such cycles of length 6 always exist in the polynomial syndrome former matrix with *any* polynomial entry of weight of three.

Property 2. If the weight of any two polynomial entries in the same column or row of the polynomial-domain syndrome former matrix is of weight two, then the corresponding LDPC-CC has at most girth 8.

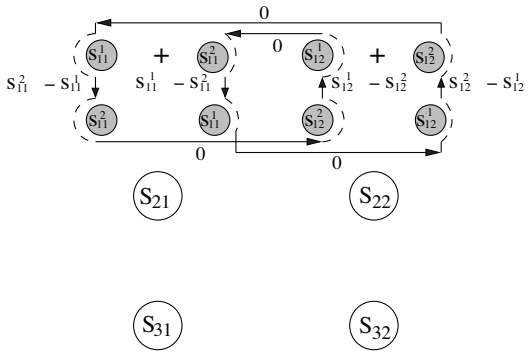


Figure 3.8: Unavoidable cycles of length 8 in $\mathbf{H}^T(D)$ with two polynomial entries of weight two in the same row

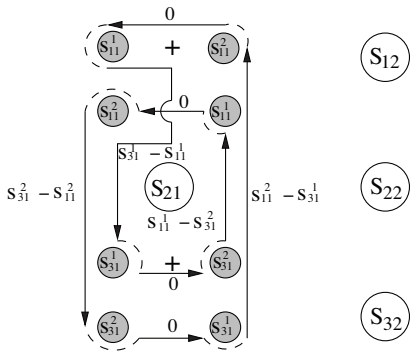


Figure 3.9: Unavoidable cycles of length 8 in $\mathbf{H}^T(D)$ with two polynomial entries of weight two in the same column

Similar as above, we show in Fig. 3.8 and Fig. 3.9 that, if we accumulate the delays along the path, the result is zero and we obtain unavoidable cycles of length 8. Note that this property is only valid when these two polynomial entries are placed in the same row or column. Moreover, a unique weight-two polynomial entry in $\mathbf{H}^T(D)$ causes unavoidable cycles of length 10 as shown in Fig. 3.10; the formal statement is as follow.

Property 3. If there exists one submatrix of size 2×2 in $\mathbf{H}^T(D)$ containing one polynomial entry of weight two while others are of weight one, then the corresponding LDPC-CC has at most girth 10.

Property 4. If there exists one submatrix of size 3×2 or 2×3 in $\mathbf{H}^T(D)$ containing only monomial entries and no empty entries in the submatrix, then the corresponding LDPC-CC has at most girth 12.

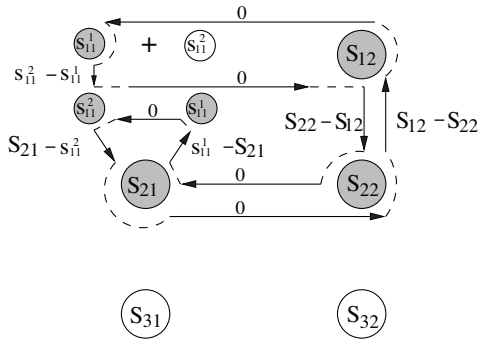


Figure 3.10: Unavoidable cycles of length 10 in $\mathbf{H}^T(D)$ with one polynomial entry of weight two

Similar to the cycle of length 12 in Fig. 3.3, in Figs. 3.11 and 3.12 we demonstrate by simple graphical means that *any* LDPC-CC that can be described by a time-*in*variant polynomial-domain syndrome former matrix has to have cycles of length 12, if the underlying QC code has $J \geq 2$ and $K \geq 3$. We use 3×2 (in Fig. 3.11) and 2×3 (in

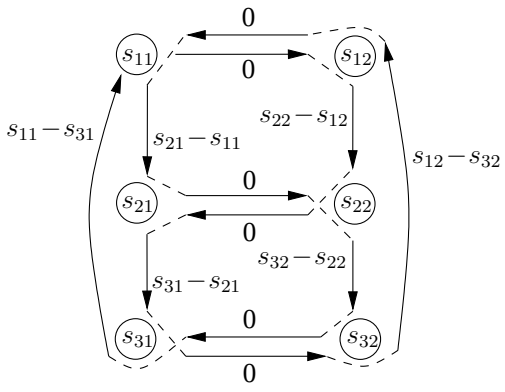


Figure 3.11: Unavoidable Cycles of length 12 in the polynomial-domain syndrome former matrix (i)

Fig. 3.12) submatrices of $\mathbf{H}^T(D)$ to demonstrate how cycles of length 12 are formed. Those cycles appear, regardless of the "powers" of the monomials in the matrix: it is a structural property, common to all time-invariant LDPC-CCs based on a QC block code construction by circulant matrices.

In Fig. 3.11 we only need to form the sum of the delays along the path starting at element s_{11}: no matter which values the elements s_{ij} have, the path delay sum is

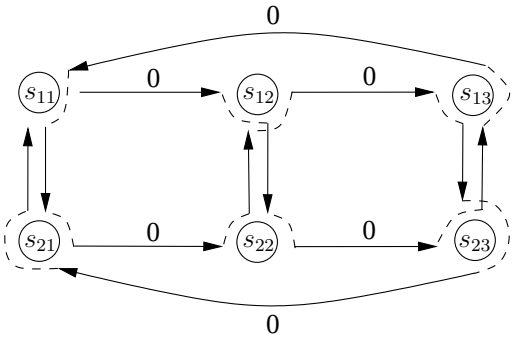


Figure 3.12: Unavoidable Cycles of length 12 in the polynomial-domain syndrome former matrix (ii)

always zero, i.e.,

$$\sum_{\forall\{s,s'\}\in\mathcal{P}'} \Delta(s,s') = \left\{ \begin{array}{lll} 0 & + & (s_{22}-s_{12}) \quad + \\ 0 & + & (s_{31}-s_{21}) \quad + \\ 0 & + & (s_{12}-s_{32}) \quad + \\ 0 & + & (s_{21}-s_{11}) \quad + \\ 0 & + & (s_{32}-s_{22}) \quad + \\ 0 & + & (s_{11}-s_{31}) \end{array} \right\} = 0, \tag{3.15}$$

therefore, we have a cycle. The same applies to the second case depicted in Fig. 3.12. The situation is even simpler than in the first case, as all vertical branches (which incur a delay) are traversed twice and in opposite directions: therefore, the sum of the delays is again zero.

In Fig. 3.13 we show a full list of all such submatrices for a $(3,5)$ LDPC-CC. From Fig. 3.13 we conclude that a lower bound for the number of structurally different non-repetitive cycles of length 12 of an (m_s, J, K)-regular LDPC-CC with $J \geq 2$, $K \geq 3$, and only monomial entries in $\mathbf{H}^T(D)$ can be computed analytically according to

$$N_{12\text{-Cycles}} \geq \binom{J}{2} \cdot \binom{K}{3} + \binom{J}{3} \cdot \binom{K}{2}, \tag{3.16}$$

where $N_{n-\text{Cycles}}$ is the number of non-repetitive cycles of length n. For a $(3,5)$ LDPC-CC we obtain $N_{12\text{-Cycles}} \geq \binom{3}{2} \cdot \binom{5}{3} + \binom{3}{3} \cdot \binom{5}{2} = 40$ which is confirmed by the full list in Fig. 3.13. It should be noted, however, that depending on the choice of the monomials in the polynomial syndrome former matrix the number of cycles of length 12 may be much larger than the lower bound. In Table 3.2 [33], it shows the cycle calculations

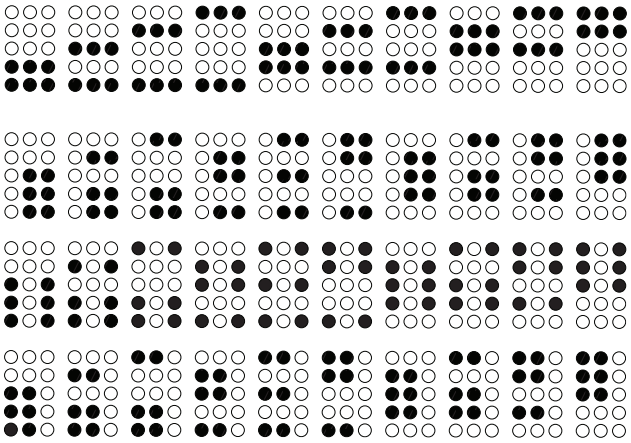


Figure 3.13: Filled circles correspond to submatrices in Figs. 3.11 and 3.12 forming unavoidable cycles of length 12 in $\mathbf{H}^T(D)$ of a $(m_s, 3, 5)$ time-invariant Tanner LDPC-CC

of a couple of code examples. The $(2005, 3, 5)$-code with $m/m_s = 2311/2005$ is a particularly interesting example: this code fulfills the lower bound (3.16) and it does not have any cycles of length 4, 6, 8, and 10. Therefore this time-invariant LDPC-CC is "perfect" with respect to its cycle properties and cannot be improved. This, however, does not make a statement about the minimum free distance or distance spectrum of this code.

3.4.2 Unavoidable cycles of length larger than 12

Above we have discussed unavoidable cycles with length up to 12 for time-invariant LDPC-CCs with nonempty entries in the polynomial syndrome former matrix. To achieve a girth larger than 12, empty entries are required to break shorter cycles in the polynomial syndrome former matrix. Similar to the unavoidable cycles of length 12 in Fig. 3.11, we illustrate by Figs. A.1 – A.12 in Appendix A that some structures lead to unavoidable cycles with lengths 14, 16, 18, and 20. A circle in these figures refers to a nonempty monomial entry. We take Fig. A.3 as an example: it is a submatrix of size 3×3 with two empty entries for the positions S_{12} and S_{33}. We sum up all the delays along the path in each figure: it satisfies the condition in (3.4) regardless of the values of the monomials.

In terms of the structure of unavoidable cycles in the polynomial syndrome former matrix, we conclude the following property for time-invariant LDPC-CCs (see also [25, 34–36]).

Property 5. In the polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$ of a time-invariant LDPC-CC, if there exists a polynomial submatrix whose weight matrix $\mathbf{B}_i$ (see 2.5) belongs to one (including permuted or transposed versions) of following categories:

$$
\begin{aligned}
\mathbf{B}_6 &= \begin{bmatrix} 3 \end{bmatrix} \\
\mathbf{B}_8 &= \begin{bmatrix} 2 & 2 \end{bmatrix} \\
\mathbf{B}_{10} &= \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \\
\mathbf{B}_{12} &= \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix} \\
\mathbf{B}_{14} &= \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \\
\mathbf{B}_{16} &= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \\
\mathbf{B}_{18} &= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \\
\mathbf{B}_{20} &= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}
\end{aligned}
\tag{3.17}
$$

then the girth of the corresponding LDPC-CC is upper bounded by i, i.e., $g \leq i$.

This property offers an overview of the girth bounds for time-invariant LDPC-CCs. For the polynomial syndrome former matrix with low entry density we expect the code, at least to some extent, to have large girth. However, low entry density in $\mathbf{H}^T(D)$ results in low column and row weights in the time-domain syndrome former matrix, which reduces the minimum *free distance* of LDPC-CCs. Therefore, there is

an tradeoff between girth and free distance. Minimum free distance is defined as the minimum Hamming distance between any two codewords, and found by experiences it is often related to the error floor of decoding performance while girth properties dominate the waterfall region.

3.5 Design time-invariant LDPC-CCs with large girth

3.5.1 The design algorithm

Based on the structural properties of unavoidable cycles (see Section 3.4), we propose an algorithm to design polynomial-domain syndrome former matrices of time-invariant LDPC-CCs with a given (desired) girth. Given the desired girth g, before applying the algorithm we need to find a weight matrix $\mathbf{B}_g$ resulting in no unavoidable cycles of length smaller than g. Then choose suitable monomials in the corresponding matrix $\mathbf{H}^T(D)$ using the flow chart algorithm described in Fig. 3.14. It contains two steps:

- sequentially test each monomial entry in the polynomial syndrome former matrix by means of the cycle counter algorithm in Algorithm 2 until the girth of this code is larger than or equal to the given girth;
- keep the girth constant and try to reduce the value of the syndrome former memory m_s as well as the number of shortest cycles.

The main principle of the algorithm is based on the idea that if each monomial in the polynomial syndrome former matrix $\mathbf{H}^T(D)$ is not involved in a cycle of length l, then the associated LDPC-CC is free of l-cycles. Given the desired girth g, syndrome former memory m_s, and considering the girth properties in the weight matrices of Property 5, we generate a polynomial syndrome former matrix whose structure contains no unavoidable cycles smaller than the selected girth g. Afterwards, we apply the above two steps to place suitable entries in $\mathbf{H}^T(D)$.

In the first step, each monomial is sequentially tested by assigning a power index smaller than m_s to it, until no cycles smaller than the girth g passing through this monomial exist. If no power indices smaller than the syndrome former memory m_s are available, m_s is increased until a suitable power index has been found.

In the second step, given the obtained matrix $\mathbf{H}^T(D)$ from the first step, we sequentially test each monomial again to check whether there is a monomial power index (smaller than the current m_s) that results in a reduction of the number (indicated by Ng in Fig. 3.14) of cycles with length g. This process is repeated until the maximum

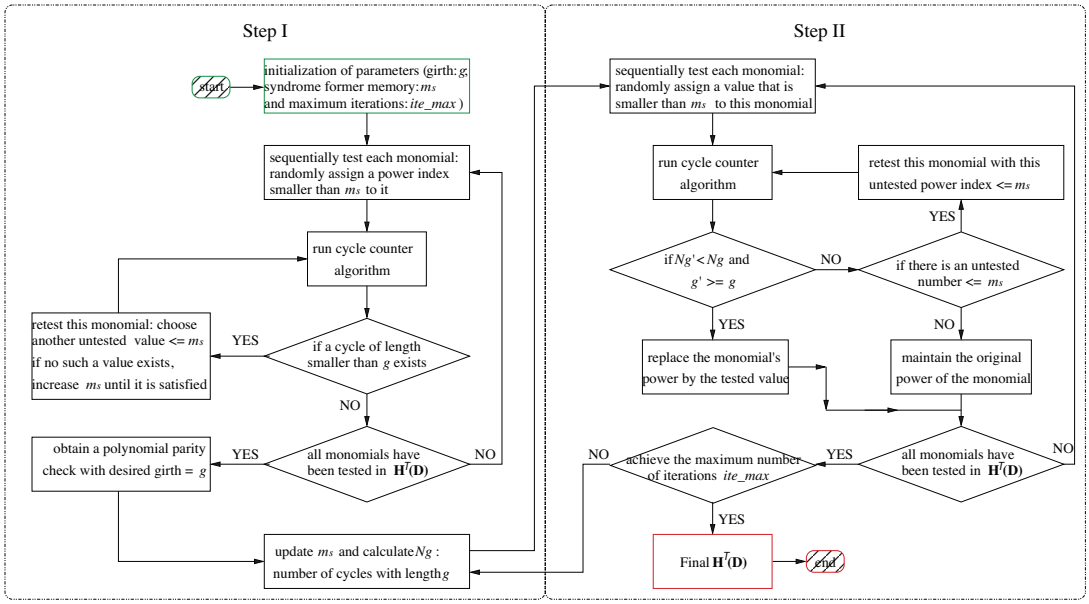


Figure 3.14: Algorithm for code design

number of iterations is reached. One iteration is defined as the procedure of testing all the monomials in $\mathbf{H}^T(D)$ once as shown in Step II in Fig. 3.14. After each iteration, the syndrome former memory m_s is updated according to the definition in Section 2.2 and the number of cycles of length g is calculated by using Algorithm 2.

3.5.2 Simulation results

In Section 3.4 we have shown that the maximum achievable girth of a $(m_s, 3, 5)$ time-invariant Tanner LDPC-CC without empty entries in $\mathbf{H}^T(D)$ is twelve and that the minimum amount of 12-cycles is $\binom{3}{2} \cdot \binom{5}{3} + \binom{3}{3} \cdot \binom{5}{2} = 40$. In Table 3.2, the $(2005, 3, 5)$-code with $m/m_s = 2311/2005$ confirms these properties. However, large syndrome former memory m_s causes high decoding complexity. To save on that, we apply the algorithm in Fig. 3.14 to produce a 5×3 polynomial matrix for the $(m_s, 3, 5)$ LDPC-CC with only monomial entries and no empty entries. Since there are submatrices of size 3×2 and 2×3, the maximum achievable girth of this codes is twelve. Rather than $m_s = 2005$, we obtain by applying our algorithm the polynomial syndrome former

matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} D^{166} & D^{12} & D^{27} \\ D^{181} & D^{95} & 1 \\ D^{19} & 1 & D^{185} \\ 1 & D^{154} & D^{117} \\ D^{58} & D^{138} & D^{170} \end{bmatrix} \tag{3.18}$$

for a $(185, 3, 5)$ LDPC-CC with much smaller syndrome former memory $m_s = 185$. Yet, the code maintains good cycle properties, i.e., it has girth 12 and 40 (smallest possible number) 12-cycles as shown in Table 3.2 in the bottom row.

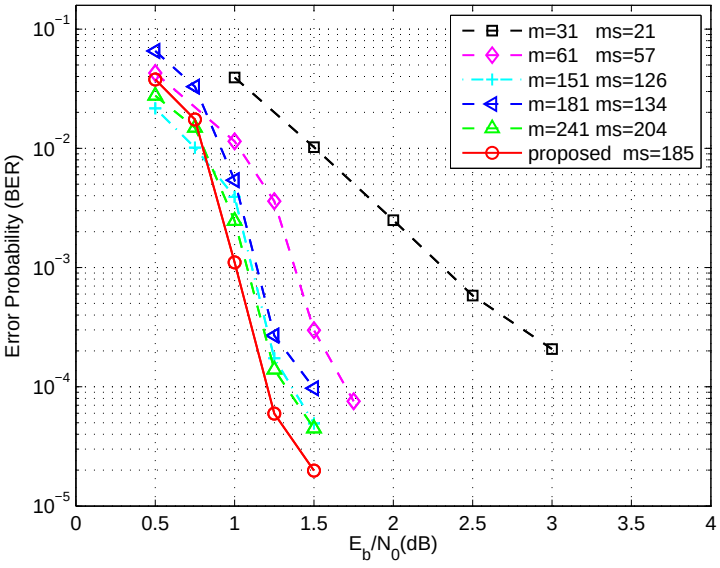


Figure 3.15: Decoding performance of some $(m_s, 3, 5)$ time-invariant LDPC-CCs of rate $R = 2/5$ with the syndrome former memory m_s and the size m of the circulant matrix indicating which QC codes the LDPC-CC is derived from (see [5])

Fig. 3.15 presents the decoding performance of the set of $(m_s, 3, 5)$ time-invariant LDPC-CCs: the dashed curves correspond to those LDPC-CCs derived from corresponding QC LDPC-BCs [5] and the solid curve describes the performance of the proposed code in (3.18). Each of the codes has a polynomial syndrome former matrix of size 5×3 with no empty entries and each entry being of weight one. The polynomial syndrome former matrices of these $(m_s, 3, 5)$ codes with syndrome former memories 21, 57, 126, 134 and 204 are defined by $\overline{\mathbf{H}}_1^T(D)$, $\overline{\mathbf{H}}_2^T(D)$, $\overline{\mathbf{H}}_3^T(D)$, $\overline{\mathbf{H}}_4^T(D)$ and $\overline{\mathbf{H}}_5^T(D)$,

Table 3.2: Cycle enumerators of some (3,5) time-invariant LDPC-CCs of rate $R = 2/5$ with the syndrome former memory m_s and the size m of the circulant matrix indicating which QC codes the LDPC-CC is derived from (see [5])

m/m_s	girth	8-cycles	10-cycles	12-cycles
31/21	8	11	62	351
61/57	10	0	21	148
151/126	10	0	3	55
181/134	12	0	0	67
241/204	12	0	0	52
2311/2005	12	0	0	40
-/185	12	0	0	40

Note: common factors have been removed from $\mathbf{H}^T(D)$.

respectively. They are given as

$$\overline{\mathbf{H}}_1^T(D) = \begin{bmatrix} 1 & 1 & D^{18} \\ D^1 & D^5 & D^{12} \\ D^3 & D^{15} & 1 \\ D^7 & D^4 & D^7 \\ D^{15} & D^{13} & D^{21} \end{bmatrix}, \overline{\mathbf{H}}_2^T(D) = \begin{bmatrix} 1 & 1 & D^{35} \\ D^8 & D^{43} & D^{45} \\ D^{19} & D^3 & D^{13} \\ D^{57} & D^9 & D^{30} \\ D^{33} & D^2 & 1 \end{bmatrix},$$

$$\overline{\mathbf{H}}_3^T(D) = \begin{bmatrix} 1 & D^{28} & D^{116} \\ D^7 & D^{101} & D^{36} \\ D^{63} & D^{81} & 1 \\ D^{58} & D^{72} & D^{14} \\ D^{18} & 1 & D^{126} \end{bmatrix}, \overline{\mathbf{H}}_4^T(D) = \begin{bmatrix} 1 & D^{23} & D^{127} \\ D^{41} & 1 & D^{109} \\ D^{134} & D^{120} & D^{77} \\ D^{58} & D^{92} & 1 \\ D^{124} & D^2 & D^{24} \end{bmatrix}$$

and

$$\overline{\mathbf{H}}_5^T(D) = \begin{bmatrix} 1 & 1 & D^{171} \\ D^{86} & D^{85} & 1 \\ D^{97} & D^9 & D^{65} \\ D^{90} & D^{145} & D^{177} \\ D^{204} & D^{168} & D^{40} \end{bmatrix}.$$

Regarding the cycle enumerators in Table 3.2, the proposed code (gray-shaded in the last row of Table 3.2) is superior to those derived from QC LDPC-BCs. This is also verified by the improvement in the waterfall region as shown in Fig. 3.15. Because of poor free distance, all codes suffer from error floors at the BER value of around 10^{-5}. The simulation was carried out over a binary phase-shift keying (BPSK) modulation AWGN channel with a maximum of 50 decoding iterations of the on-demand variable node activation [37] sum-product pipeline [15] decoding algorithm for LDPC-CCs.

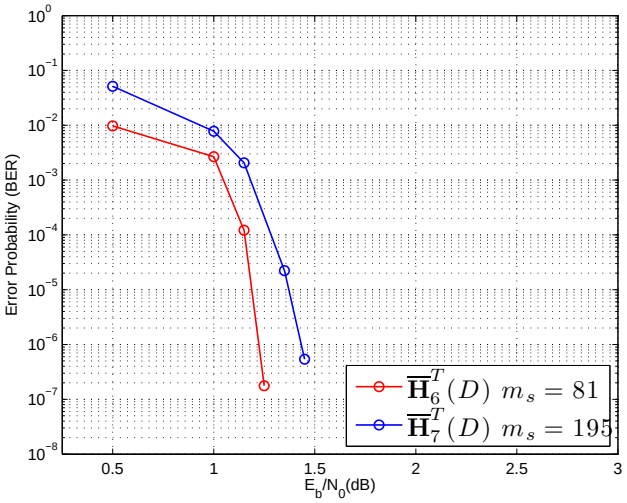


Figure 3.16: Decoding performance of two proposed time-invariant LDPC-CCs

Moreover, to break the structure of unavoidable cycles of length 12, we proposed two weight matrices

$$\mathbf{B}_{12} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{B}_{14} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.19}$$

that contain no submatrices forming unavoidable cycles of length smaller than 12 and 14 (see (3.17)), respectively. Applying the algorithm in Fig. 3.14 to the above two weight matrices, we obtain two regular $(m_s, 3, 4)$ time-invariant LDPC-CCs with polynomial syndrome former matrices

$$\overline{\mathbf{H}}_6^T(D) = \begin{bmatrix} D^{32} & & D^{27} & & D^{60} & \\ & D^{72} & & 1 & & 1 \\ D^{42} & & D^{26} & & & D^{21} \\ & D^{46} & & D^{81} & 1 & \\ 1 & & & D^{55} & D^{79} & \\ & D^{29} & D^{39} & & & D^{25} \\ D^{25} & & & D^{75} & D^{80} & \\ & 1 & 1 & & & D^{78} \end{bmatrix} \tag{3.20}$$

and

$$\overline{\mathbf{H}}_7^T(D) = \begin{bmatrix} & & & D^{57} & D^{34} & D^{195} \\ 1 & & & & D^{174} & D^{148} \\ & D^{139} & & D^{71} & & 1 \\ & & D^{74} & D^{162} & 1 & \\ D^{116} & 1 & & & & D^{160} \\ D^{171} & & D^{33} & & D^{104} & \\ & D^{42} & D^{123} & 1 & & \\ D^{122} & D^{163} & 1 & & & \end{bmatrix}. \tag{3.21}$$

which have girth of 12 and 14, respectively, and they have the same column and row degree distribution in their weight matrices. Their decoding performances are shown in Fig. 3.16. As we expected, $\overline{\mathbf{H}}_7^T(D)$ has better decoding performance in the waterfall region than $\overline{\mathbf{H}}_6^T(D)$ since the girth of $\overline{\mathbf{H}}_7^T(D)$ is larger than that of $\overline{\mathbf{H}}_6^T(D)$.

3.6 Conclusion

In this chapter, we have analyzed the cycle structures for time-invariant and time-varying LDPC-CCs derived from the corresponding QC LDPC-BCs. A cycle counting algorithm is introduced to compute the number of short cycles. Moreover, unavoidable cycles due to destructive structures of polynomial syndrome former matrices are presented from the graphical point-of-view. As a result of the unavoidable cycles, we obtain upper bounds on the girth of time-invariant LDPC-CCs. Based on the formations of unavoidable cycles, an algorithm is presented to generate time-invariant LDPC-CCs achieving desired girth. In adition, the cycle properties were compared between time-invariant and time-varying LDPC-CCs. As we expected, time-varying LDPC-CCs contain less cycles than time-invariant LDPC-CCs.

Chapter 4

Distance Spectrum Estimation of LDPC Convolutional Codes

4.1 Introduction

Low-density parity-check convolutional codes (LDPC-CCs) were first proposed in [1]. Using pipeline decoding [2], it has been shown that they are suitable for practical implementation with continuous transmission as well as, via encoder termination, block transmission in frames of arbitrary size [3] without an increase in computational complexity compared to their block code counterparts.

LDPC-CCs can be separated into two categories, time-invariant and time-varying LDPC-CCs, with respect to the structure of their syndrome former matrices. In this chapter, we focus on time-invariant LDPC-CCs, derived from corresponding QC LDPC-BC [5]. This particular category of polynomial based LDPC-CCs can be considered as conventional convolutional codes with high memory order that results in the sparsity in the syndrome former matrix.

For any convolutional code $\mathcal{C}$ with minimum free distance d_f, the weight spectrum is described by a set $\{A_w | w \geq d_f, w \in \mathbb{Z}^+\}$, where A_w, the codeword weight enumerator, is the number of codewords with Hamming weight w. Together with the minimum free distance, the distance spectrum is an important property of convolutional codes for estimating their performance under a variety of decoding algorithms. Bahl *et al.* [38] presented an algorithm to compute the free distance based on a state-transition diagram. In [39], Rouanne and Costello introduced a bidirectional tree search algorithm to calculate the distance spectrum of convolutionally encoded trellis codes. Subsequently, Bocharova *et al.* [13] presented a more efficient bidirectional tree search algorithm called

BEAST. Sridharan *et al.* [40] showed that for sufficiently large constraint lengths v_s, the minmum distances d_{min} grows linearly with v_s and the ratio of d_{min}/v_s approaches the ratio d_f/v_s for large block length. In [41], existence type lower bounds on the free distance of periodically time-varying LDPC-CCs and on the minimum distance of tail-biting LDPC-CCs are derived. It is demonstrated that the bound on free distance of periodically time-varying LDPC-CCs approaches the bound on free distance of general (nonperiodic) time-varying LDPC-CCs as the period increases. Using asymptotic methods, Mitchell *et al.* [9] obtained lower bounds on the free distance to constraint length ratio, they showed that several ensembles of regular and irregular LDPC-CCs derived from protograph-based LDPC block codes (LDPC-BCs) have the property that the free distance grows linearly with respect to the constraint length, i.e., the ensembles are asymptotically good. In this chapter, we present an algorithm to estimate the distance spectrum of time-invariant LDPC-CCs derived from corresponding quasi-cyclic (QC) LDPC block codes. Time-invariant LDPC-CCs can be considered as conventional convolutional codes with large memory order that are characterized by the sparsity of the syndrome former matrix.

To estimate the distance spectrum of a time-invariant LDPC-CC, we first split the columns of the polynomial syndrome former matrix into submatrices and compute the distance spectrum for each of the "super codes" defined by the submatrices, where each super code contains all the codewords of the original code. Following this, we apply a so-called *linear dependence evaluation* method to investigate the linear dependence between the low weight codewords of the different super codes. This technique provides an estimate of the distance spectrum of the original LDPC-CC, and in particular we obtain an upper bound on the minimum free distance and a lower bound on the number of codewords A_w with Hamming weight w. In order to reduce the complexity of estimating the distance spectrum, the technique is applied to the finite, compact, *polynomial* syndrome former matrix rather than the semi-infinite time-domain syndrome former matrix. We also show that, for some example codes, the estimation technique is highly accurate, resulting in an exact calculation of the free distance d_f.

The rest of the chapter is organized as follows: in Section 4.2, we present some properties of the Hamming weights of codewords in LDPC-CCs defined with respect to the (polynomial) syndrome former matrix. The algorithm to estimate the distance spectrum of polynomial-based LDPC-CCs is presented in Section 4.3. This is followed by some illustrative examples in Section 4.4.

The work of this chapter were published in [42].

4.2 Hamming weights of codewords in LDPC-CCs

In this section we present some properties of the Hamming weights of codewords in time-invariant LDPC-CCs defined with respect to the (polynomial) syndrome former matrix, which form the basis for estimating the distance spectrum in Section 4.3.

4.2.1 Properties of the Hamming weight

In order to introduce some useful notation and terminology, we begin by stating a well-known property of convolutional codes as Theorem 2 and giving a simple proof of this result.

Property 1. Let $\mathcal{C}$ be a convolutional code with syndrome former matrix $\mathbf{H}^T$. For a codeword $\mathbf{v}$ of Hamming weight l, there exist l corresponding rows in $\mathbf{H}^T$ such that the sum of these l rows is the zero vector. Conversely, if there exist l rows in $\mathbf{H}^T$ whose sum is the zero vector, there exists a codeword $\mathbf{v}$ of Hamming weight l.

Proof: The constraint imposed by the syndrome former matrix, i.e., $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$, can be written as

$$\mathbf{v}_t \mathbf{H}_0^T(t) + \mathbf{v}_{t-1}\mathbf{H}_1^T(t) + \cdots + \mathbf{v}_{t-m_s}\mathbf{H}_{m_s}^T(t) = \mathbf{0}, t \in \mathbb{Z}. \tag{4.1}$$

Each horizontal block submatrix of (2.2), which corresponds to the symbols of $\mathbf{v}_t$, is given by

$$\begin{bmatrix} \mathbf{H}_0^T(t) & \mathbf{H}_0^T(t+1) & \cdots & \mathbf{H}_{m_s}^T(t+m_s) \end{bmatrix}, \tag{4.2}$$

where $t \in \mathbb{Z}$. Then, expanding (4.2) using (2.3) we obtain

$$\begin{bmatrix} h_{1,1}^{0,t} & \cdots & h_{1,p}^{0,t} & \cdots & h_{1,1}^{m_s,t} & \cdots & h_{1,p}^{m_s,t} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{c,1}^{0,t} & \cdots & h_{c,p}^{0,t} & \cdots & h_{c,1}^{m_s,t} & \cdots & h_{c,p}^{m_s,t} \end{bmatrix}. \tag{4.3}$$

Assuming that codeword $\mathbf{v}$ has l nonzero components, let $v_{t_1}^{(c_1)} = v_{t_2}^{(c_2)} = \cdots = v_{t_l}^{(c_l)} = 1$ be the l nonzero components, where $t_i \in \mathbb{Z}$, $c_i \in \mathbb{Z}^+$, and $c_i \leq c$, $1 \leq i \leq l$. After expanding each horizontal block submatrix (4.2) using (2.3), consider forming the submatrix $\mathbf{H}_{\mathbf{v}}^T$ corresponding exactly to the l rows indexed by the codeword bits $v_{t_1}^{(c_1)}$, $v_{t_2}^{(c_2)}$,..., $v_{t_l}^{(c_l)}$, i.e.,

$$\mathbf{H}_{\mathbf{v}}^T = \begin{bmatrix} h_{c_1,1}^{0,t_1} \cdots h_{c_1,p}^{0,t_1} \cdots h_{c_1,1}^{m_s,t_1} & \cdots & h_{c_1,p}^{m_s,t_1} & & \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ & h_{c_l,1}^{0,t_l} & \cdots & h_{c_l,p}^{0,t_l} & \cdots & h_{c_l,1}^{m_s,t_l} & \cdots & h_{c_l,p}^{m_s,t_l} \end{bmatrix}. \tag{4.4}$$

As a result of (4.1), it follows that the sum of the rows of (4.4) results in the zero vector. The codeword $\mathbf{v}$ satisfies $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$ and thus indicates that a codeword with Hamming weight l corresponds exactly to l rows in $\mathbf{H}^T$ that sum to the zero vector. A similar argument can be used to prove the converse. ■

Property 1 also applies to time-invariant convolutional codes defined with respect to the (polynomial) syndrome former matrix. Assuming that codeword $\mathbf{v}$ has l nonzero components, let $v_{t_1}^{(c_1)} = v_{t_2}^{(c_2)} = \cdots = v_{t_l}^{(c_l)} = 1$ be the l nonzero components, where $t_i \in \mathbb{Z}$, $c_i \in \mathbb{Z}^+$, and $c_i \leq c$, $1 \leq i \leq l$. Then $\mathbf{v}$ can be described in the polynomial domain as $\mathbf{v}(D) = D^{(c \cdot t_1 + c_1)} + D^{(c \cdot t_2 + c_2)} + \cdots + D^{(c \cdot t_l + c_l)}$, and to satisfy the constraint in (2.10), we have

$$\sum_{i=1}^{l} \mathbf{h}_{c_i}(D) \cdot D^{t_i} = \mathbf{0}(D), \tag{4.5}$$

where $\mathbf{h}_{c_i}(D)$ is the c_i-th row of $\mathbf{H}^T(D)$ in (2.4).Throughout the chapter, we refer to $\mathbf{h}_{c_i}(D) \cdot D^{t_i}$ as an extended row of $\mathbf{H}^T(D)$. Based on above description, we obtain following property.

Property 2. Let $\mathcal{C}(D)$ be a polynomial-domain LDPC-CC with polynomial syndrome former matrix $\mathbf{H}^T(D)$. For a codeword $\mathbf{v}$(D) of Hamming weight l, there exist corresponding l extended rows of $\mathbf{H}^T(D)$ such that the sum of these l extended rows is the matrix of all-zero sequences $\mathbf{0}$(D). Conversely, if there exist l extended rows of $\mathbf{H}^T(D)$ whose sum is the matrix of all-zero sequences $\mathbf{0}$(D), then there exists a codeword $\mathbf{v}(D)$ of Hamming weight l.

4.2.2 An example

Given the $R = 1/3$ QC LDPC-BC given by Tanner *et al.* in [5], we can form an associated $R = 1/3$ time-invariant LDPC-CC with polynomial-domain syndrome former matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} \mathbf{h}_1(D) \\ \mathbf{h}_2(D) \\ \mathbf{h}_3(D) \end{bmatrix} = \begin{bmatrix} h_{1,1}(D) & h_{1,2}(D) \\ h_{2,1}(D) & h_{2,2}(D) \\ h_{3,1}(D) & h_{3,2}(D) \end{bmatrix} = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix}, \tag{4.6}$$

and time-domain syndrome former matrix

$$\mathbf{H}^T = \left[\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right\} t_1{=}0$$

$$\left[\begin{array}{cccccccc}
\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right\} t_2{=}1$$

$$\left[\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0}
\end{array}\right\} t_3{=}t_4{=}2$$

$$\left[\begin{array}{cccccccc}
\mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right\} t_5{=}3$$

$$\left[\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right\} t_6{=}4$$

$$\left[\begin{array}{cccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots
\end{array}\right] . \tag{4.7}$$

Consider the six emboldened rows of (4.7) that correspond to the time indices $t_1 = 0$, $t_2 = 1$, $t_3 = t_4 = 2$, $t_5 = 3$, and $t_6 = 4$ and the row indices $c_1 = 2$, $c_2 = 1$, $c_3 = 2$, $c_4 = 3$, $c_5 = 1$, and $c_6 = 2$, respectively. These six row vectors can be used to construct the submatrix $\mathbf{H}_{\mathbf{v}}^T$ as

$$\mathbf{H}_{\mathbf{v}}^T = \left[\begin{array}{cccccccccccccccc}
\mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & & & & & & & & \\
 & & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & & & & & & \\
 & & & & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & & & & \\
 & & & & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & & & & \\
 & & & & & & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & & \\
 & & & & & & & & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0}
\end{array}\right] . \tag{4.8}$$

Note that summing the rows of (4.8) modulo 2 results in the zero vector. The corresponding polynomial form is

$$\begin{aligned}
\textstyle\sum_{i=1}^{6} \mathbf{h}_{c_i}(D) \times D^{t_i} &= \mathbf{h}_2(D) \cdot D^0 + \mathbf{h}_1(D) \cdot D^1 + \mathbf{h}_2(D) \cdot D^2 + \\
&\quad \mathbf{h}_3(D) \cdot D^2 + \mathbf{h}_1(D) \cdot D^3 + \mathbf{h}_2(D) \cdot D^4 \\
&= \begin{bmatrix} D & D^2 \end{bmatrix} + \begin{bmatrix} D & D^4 \end{bmatrix} + \begin{bmatrix} D^3 & D^4 \end{bmatrix} + \\
&\quad \begin{bmatrix} D^5 & D^2 \end{bmatrix} + \begin{bmatrix} D^3 & D^6 \end{bmatrix} + \begin{bmatrix} D^5 & D^6 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0 \end{bmatrix} = \mathbf{0}(D).
\end{aligned}$$

Therefore, the sequence $\mathbf{v}(D) = D^2 + D^4 + D^8 + D^9 + D^{10} + D^{14}$ is a codeword with Hamming weight 6 in this LDPC-CC.

4.3 Estimating the distance spectrum of LDPC-CCs using linear dependence evaluation

The distance spectrum is an important property of a convolutional code needed to estimate its performance under a variety of decoding algorithms. However, compared to conventional convolutional codes, it is much more complex to calculate the distance spectrum of LDPC-CCs. This is because the low density requirement of the syndrome former matrix of LDPC-CCs results in a comparatively large memory. In this section, we will introduce a strategy to estimate the first several nonzero codeword enumerators A_w in the distance spectrum of time-invariant LDPC-CCs.

Based on Property 2, a straightforward way to calculate the initial portion of the distance spectrum of an LDPC-CC is to find the smallest sets of variable nodes that satisfy the constraint that their corresponding rows in $\mathbf{H}^T$, or extended rows in $\mathbf{H}^T(D)$, sum up to the zero vector, or the vector of all-zero sequences, respectively. However, searching for all possible combinations is very time consuming. To improve the efficiency, we introduce a novel two-step solution.

First, if a row vector r_1 is involved in a row vector set $\overline{r}$ of $\mathbf{H}^T$ that sums to the zero vector, then, for each '1' in r_1, there must exist a row vector $r_2 \in \overline{r}$ that has a '1' in the same column in order to cancel the '1' in r_1 (as shown in the previous example). Recall that a row vector in $\mathbf{H}^T$ corresponds to a variable node, and consequently, instead of searching the entire set of rows in $\mathbf{H}^T$, only neighboring variable nodes are considered in the search to cancel '1's, where two variable nodes are called neighbors if and only if they have at least one constraint node in common. Due to the semi-infinite size of the time-domain syndrome former matrix, it is inconvenient to analyze such connections among variable nodes. However, as described in Section 2.1, we can convert $\mathbf{H}^T$ to a finite, compact, polynomial syndrome former matrix that facilitates analysis.

Second, rather than computing the distance spectrum in the original code (with a relatively large free distance), we split the columns of the polynomial syndrome former matrix into several submatrices and calculate the distance spectrum of each of the resulting convolutional "super codes". Given the distance spectra of the super codes, we can then form an estimate of the distance spectrum for the original code by evaluating the linear dependence between the codewords of the super codes. The linear dependence evaluation method will be described later in this section.

The splitting concept is explained as follows. Given a polynomial syndrome former matrix $\mathbf{H}^T(D)$, we partition it into a sequence of I polynomial syndrome former submatrices $\mathbf{H}_{p_i}^T(D)$ of size $c\times q$,[1] $q\leq p$, $q\in\mathbb{Z}^+$, $p_i\in\mathbb{Z}^+$, $1\leq i\leq I$, $i\in\mathbb{Z}$, where each $\mathbf{H}_{p_i}^T(D)$ defines a super code and each column of $\mathbf{H}^T(D)$ appears in at least one of the submatrices $\mathbf{H}_{p_i}^T(D)$.[2] If, among the super codes, there is a common codeword $\mathbf{W}(D)$, we obtain

$$\left.\begin{array}{c}\mathbf{W}(D)\cdot\mathbf{H}_{p_1}^T(D)=\mathbf{0}(D)\\ \vdots\\ \mathbf{W}(D)\cdot\mathbf{H}_{p_I}^T(D)=\mathbf{0}(D)\end{array}\right\}\Rightarrow\mathbf{W}(D)\cdot\mathbf{H}^T(D)=\mathbf{0}(D), \tag{4.9}$$

i.e., $\mathbf{W}(D)$ is also a codeword of the original LDPC-CC with syndrome former matrix $\mathbf{H}^T(D)$. Some examples of this splitting procedure are given in Section 4.4.

Finding common codewords in the super codes is still a time consuming task if we try to search through all possible codewords, since the super codes are also LDPC-CCs, and thus they contain codewords of infinite length. However, we conduct the search for common codewords by evaluating the linear dependence between codewords of the super codes in conjunction with information from their corresponding distance spectra.

Let $\mathbf{v}_{p_j}^{l_s}(D)$, $1\leq j\leq I$, $j\in\mathbb{Z}$, and $l_s\in\mathbb{Z}^+$, be a codeword in the codeword set $\overline{\mathbf{v}}_{p_j}(D)$ that satisfies $\mathbf{v}_{p_j}^{l_s}(D)\mathbf{H}_{p_j}^T(D)=\mathbf{0}(D)$. Now suppose that there exists a common codeword $\mathbf{v}(D)$ among all the super codes that can be formed as the modulo 2 sum of some number of periodically shifted codewords in each super code with polynomial syndrome former matrix $\mathbf{H}_{p_j}^T(D)$, $1\leq j\leq I$, i.e., there is a set of shift parameter vectors $\{\mathbf{m}_{p_1},\mathbf{m}_{p_2},\ldots,\mathbf{m}_{p_I}\}$, where

$$\mathbf{m}_{p_j}=(m_{j1},m_{j2},\ldots,m_{jL_j}), \tag{4.10}$$

$m_{js}\in\mathbb{Z}$, $1\leq s\leq L_j$, and a set of codewords

$$\{(\mathbf{v}_{p_1}^{l_1}(D),\ldots,\mathbf{v}_{p_1}^{l_{L_1}}(D)),\ldots,(\mathbf{v}_{p_I}^{l_1}(D),\ldots,\mathbf{v}_{p_I}^{l_{L_I}}(D))\} \tag{4.11}$$

that satisfy

$$\left\{\begin{array}{c}\sum_{s=1}^{L_1}D^{c\cdot m_{1s}}\mathbf{v}_{p_1}^{l_s}(D)=\mathbf{v}(D)\\ \vdots\\ \sum_{s=1}^{L_I}D^{c\cdot m_{Is}}\mathbf{v}_{p_I}^{l_s}(D)=\mathbf{v}(D)\end{array}\right. . \tag{4.12}$$

[1] In this chapter, $\mathbf{H}_{p_i}^T(D)$ refers to a super code, but *not* to the polynomial submatrix in the syndrome former matrix of a time-varying LDPC-CC as defined in Chapter 2.

[2] Note that $I\times q$ may be greater than p, so some columns of $\mathbf{H}^T(D)$ may appear in more than one submatrix $\mathbf{H}_{p_i}^T(D)$.

Then it follows that the common codeword $\mathbf{v}(D)$ is also a codeword in the original LDPC-CC, and thus this codeword contributes to the codeword weight enumerator A_w. In searching for these sets of shift parameter vectors, we set a maximum value for m_{jl_j} to ensure termination of the search. We call this process of finding common codewords a *linear dependence evaluation* of codewords in the distance spectra of the super codes. Since linear dependence evaluation does not guarantee that the obtained common codeword has minimum weight, the value that we obtain for the minimum weight codeword is an upper bound on the free distance of the original LDPC-CC, and the number of codewords obtained for any given Hamming weight is a lower bound on the number of codewords of this weight. Note that the linear dependence evaluation method is more likely to give an accurate estimate of the number of codewords with low Hamming weight. This is because, as we increase the Hamming weight, the number of codewords to consider also increases which, in turn, greatly increases the search complexity.

In estimating the distance spectrum, only multiplexed codewords $\mathbf{v}(D)$ whose minimum degree[1] is smaller than or equal to c are included. Thus periodically shifted codewords are not included in the estimate of the distance spectrum. In addition, if there are $n+1$ codewords $\{\mathbf{v}^0(D), \mathbf{v}^1(D), \ldots, \mathbf{v}^n(D) | n > 1, n \in \mathbb{Z}^+\}$ in the codeword set $\overline{\mathbf{v}}(D)$ that satisfy

$$\mathbf{v}^0(D) = \textstyle\sum_{k=1}^{n} \mathbf{v}^k(D) \tag{4.13}$$

and

$$w(\mathbf{v}^0(D)) = w(\textstyle\sum_{k=1}^{n-1} \mathbf{v}^k(D)) + w(\mathbf{v}^n(D)), \tag{4.14}$$

where $w(\mathbf{v}^k(D))$ indicates the Hamming weight of codeword $\mathbf{v}^k(D)$, then the codeword $\mathbf{v}^0(D)$ is not counted in the codeword weight enumerator $A_{w(\mathbf{v}^0(D))}$, i.e., linear combinations of codewords with nonoverlapping ones are not counted as separate codewords. (Note: Notationally, $\mathbf{v}^0(D)$,$\mathbf{v}^1(D)$,...,$\mathbf{v}^n(D)$ are multiplexed codewords, while $\mathbf{v}^{(i)}(D)$ in (2.8) is the ith element in the c-tuple of the codeword $\mathbf{V}(D)$, $0 \leq i \leq c-1$.)

Even though the linear dependence evaluation method only gives an estimate of the distance spectrum, it is shown in Section 4.4 that this technique yields the exact free distance for a well-known class of LDPC-CCs and, moreover, all codewords of low Hamming weight are accounted for in the evaluation.

[1]The *minimum degree* of a polynomial is defined as the lowest exponent of a term with non-zero coefficient. For example, the minimum degree of $D + D^3 + D^6$ is one.

4.4 Application of linear dependence evaluation to some example codes

4.4.1 The Tanner $(21, 3, 5)$ LDPC convolutional code

To illustrate the application of the linear dependence evaluation method to finding common codewords among super codes, the Tanner $(21, 3, 5)$, $R = (c - p)/c = 2/5$ LDPC convolutional code [5] is chosen as an example. This time-invariant code has polynomial syndrome former matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & 1 & D^{18} \\ D & D^5 & D^{12} \\ D^3 & D^{15} & 1 \\ D^7 & D^4 & D^7 \\ D^{15} & D^{13} & D^{21} \end{bmatrix}. \tag{4.15}$$

(Note that the common factors of D have been removed from the code in [5] for simplicity). First, we split the polynomial syndrome former matrix of the original code into two submatrices $\mathbf{H}_1^T(D)$ and $\mathbf{H}_2^T(D)$ as follows[1]:

$$\mathbf{H}_1^T(D) = \begin{bmatrix} 1 & 1 \\ D & D^5 \\ D^3 & D^{15} \\ D^7 & D^4 \\ D^{15} & D^{13} \end{bmatrix}, \tag{4.16}$$

$$\mathbf{H}_2^T(D) = \begin{bmatrix} 1 & D^{18} \\ D^5 & D^{12} \\ D^{15} & 1 \\ D^4 & D^7 \\ D^{13} & D^{21} \end{bmatrix}. \tag{4.17}$$

Next, we compute the distance spectrum of each super code based on the concept introduced in Section 4.3. The first three nonzero codeword weight enumerators are given in Table I. We observe that each super code has minimum free distance 6 and that, additionally, no codewords with odd Hamming weight exist in the spectrum. Given these two distance spectra, if there is a common codeword that is a sum of periodically

[1]Note that, as described in Section 4.3, this is an example where there is a repeated column in $\mathbf{H}_1^T(D)$ and $\mathbf{H}_2^T(D)$, since $I \times q = 2 \times 2 > p = 3$.

Table 4.1: Distance spectra of super codes

Super codes	d_f	Codeword weight enumerator A_w		
		A_6	A_8	A_{10}
$\mathbf{H}_1^T(D)$	6	22	158	1817
$\mathbf{H}_2^T(D)$	6	12	68	924

Codewords with minimum free weight		
$\mathbf{H}_1^T(D) : \mathbf{v}_1^l(D) \in \overline{\mathbf{v}}_1(D)$		$\mathbf{H}_2^T(D) : \mathbf{v}_2^l(D) \in \overline{\mathbf{v}}_2(D)$
$\mathbf{v}_1^1(D) \to \mathbf{v}_1^{12}(D)$	$\mathbf{v}_1^{13}(D) \to \mathbf{v}_1^{22}(D)$	$\mathbf{v}_2^1(D) \to \mathbf{v}_2^{12}(D)$
$[3, 12, 32, 36, 52, 56]$	$[5, 15, 33, 42, 76, 82]$	$[2, 26, 57, 63, 81, 112]$
$[3, 12, 19, 38, 59, 87]$	$[5, 35, 53, 66, 72, 102]$	$[2, 26, 29, 46, 77, 84]$
$[3, 12, 15, 19, 47, 54]$	$[5, 29, 42, 44, 46, 61]$	$[2, 9, 43, 63, 92, 119]$
$[3, 12, 23, 36, 72, 76]$	$[5, 49, 64, 72, 81, 96]$	$[2, 26, 63, 118, 136, 167]$
$[4, 17, 21, 24, 32, 56]$	$[5, 49, 63, 77, 84, 112]$	$[4, 21, 38, 111, 113, 169]$
$[4, 8, 21, 23, 64, 96]$	$[5, 20, 49, 59, 76, 81]$	$[5, 30, 55, 66, 74, 99]$
$[5, 33, 35, 42, 72, 106]$	$[5, 25, 62, 66, 72, 96]$	$[5, 25, 42, 67, 69, 74]$
$[5, 28, 37, 44, 49, 77]$	$[5, 33, 42, 45, 63, 112]$	$[5, 47, 60, 66, 71, 97]$
$[5, 10, 15, 49, 54, 76]$	$[5, 53, 63, 65, 66, 136]$	$[5, 66, 74, 80, 91, 124]$
$[5, 40, 49, 72, 77, 79]$	$[5, 49, 63, 68, 80, 119]$	$[5, 49, 83, 95, 108, 164]$
$[5, 15, 62, 66, 76, 86]$	–	$[5, 42, 103, 108, 115, 157]$
$[5, 44, 49, 57, 61, 81]$	–	$[5, 66, 108, 158, 170, 181]$

shifted codewords in each super code, then this codeword is also a codeword in the original code. In this example, only codewords with minimum free weight from each super code are used to evaluate linear dependence. Nevertheless, we see that this already gives a good estimate of the distance spectrum of the original code. Due to space limitations, only the power indices are listed in Tables 4.1 and 4.2; for example, if a codeword in the polynomial domain is $D^1 + D^3 + D^6$, it is represented as $[1, 3, 6]$.

By applying the linear dependence evaluation method to the codewords with free weight 6 in the super codes, i.e., the 12 codewords from $\mathbf{H}_2^T(D)$ and the 22 codewords from $\mathbf{H}_1^T(D)$ as given in Table 4.1, we obtain an estimate of the distance spectrum of the original code, which is given in Table 4.2.[1] We estimate that the code with polynomial syndrome former matrix given by (4.15) has free distance 24, which is consistent with the free distance given in [5], and the estimated numbers of codewords with Hamming weight 24, 26, 28, 30, and 32 are 6, 5, 8, 34, and 53, respectively. Moreover, we find all

[1] The distance spectrum of this code has also been computed recently in [43].

Table 4.2: Estimated distance spectrum of the original LDPC-CC

(m_s, J, K)	d_f	Codeword weight enumerator A_w				
		A_{24}	A_{26}	A_{28}	A_{30}	A_{32}
$(21, 3, 5)$	24	6	5	8	34	53

Codewords with minimum free weight: $\mathbf{v}^l(D) \in \overline{\mathbf{v}}(D)$
$\mathbf{v}^1(D) = [4, 17, 21, 24, 32, 38, 47, 56, 58, 71, 74, 78, 91, 93, 107,$ $111, 113, 122, 129, 134, 148, 166, 169, 197]$
$\mathbf{v}^2(D) = [5, 30, 49, 60, 65, 72, 74, 77, 83, 96, 101, 102, 104, 119,$ $127, 132, 134, 136, 147, 153, 167, 171, 174, 202]$
$\mathbf{v}^3(D) = [5, 20, 35, 49, 59, 76, 81, 83, 93, 95, 96, 104, 108, 110,$ $121, 123, 154, 164, 166, 168, 173, 185, 196, 224]$
$\mathbf{v}^4(D) = [5, 25, 45, 60, 62, 66, 72, 74, 80, 87, 89, 91, 94, 96, 99,$ $102, 115, 116, 121, 124, 126, 147, 152, 154]$
$\mathbf{v}^5(D) = [5, 40, 49, 55, 72, 75, 77, 79, 83, 92, 95, 108, 113, 117,$ $119, 124, 133, 138, 143, 150, 162, 164, 189, 192]$
$\mathbf{v}^6(D) = [5, 25, 62, 66, 72, 75, 96, 108, 117, 123, 128, 130, 133,$ $135, 136, 141, 158, 167, 170, 177, 181, 188, 206, 237]$

six codewords with minimum weight. For example, the codeword $\mathbf{v}^1(D)$ is a common codeword of both super codes, since there is a set of shift parameter vectors

$$\left\{ \begin{array}{l} \mathbf{m}_1 = (m_{11}, m_{12}, m_{13}, m_{14}) = (22, 7, 0, 14), \\ \mathbf{m}_2 = (m_{21}, m_{22}, m_{23}, m_{24}) = (9, 3, 6, 0) \end{array} \right\}$$

and a set of codewords (for simplicity, the notation 'D' is ignored)

$$\left\{ \begin{array}{l} (\mathbf{v}_1^{l_1}, \mathbf{v}_1^{l_2}, \mathbf{v}_1^{l_3}, \mathbf{v}_1^{l_4}) = (\mathbf{v}_1^2, \mathbf{v}_1^4, \mathbf{v}_1^5, \mathbf{v}_1^6), \\ (\mathbf{v}_2^{l_1}, \mathbf{v}_2^{l_2}, \mathbf{v}_2^{l_3}, \mathbf{v}_2^{l_4}) = (\mathbf{v}_2^2, \mathbf{v}_2^3, \mathbf{v}_2^4, \mathbf{v}_2^5) \end{array} \right\}$$

in the super codes $\mathbf{H}_1^T(D)$ and $\mathbf{H}_2^T(D)$, respectively, that give

$$\begin{cases} D^{5\cdot 22}\mathbf{v}_1^2(D) + D^{5\cdot 7}\mathbf{v}_1^4(D) + D^{5\cdot 0}\mathbf{v}_1^5(D) + D^{5\cdot 14}\mathbf{v}_1^6(D) = \mathbf{v}^1(D) \\ D^{5\cdot 9}\mathbf{v}_2^2(D) + D^{5\cdot 3}\mathbf{v}_2^3(D) + D^{5\cdot 6}\mathbf{v}_2^4(D) + D^{5\cdot 0}\mathbf{v}_2^5(D) = \mathbf{v}^1(D) \end{cases},$$

i.e., condition (4.12) is satisfied. Similar linear dependence evaluations can be formulated for codewords $\mathbf{v}^2(D)$, ..., $\mathbf{v}^6(D)$.

Table 4.3: Estimated distance spectra of some Tanner (m_s,3,5) LDPC-CCs

(m_s, J, K)	$(57, 3, 5)$		$(126, 3, 5)$		$(204, 3, 5)$	
$\mathbf{H}^T(D)$	$\begin{bmatrix} 0 & 0 & 35 \\ 8 & 43 & 45 \\ 19 & 3 & 13 \\ 57 & 9 & 30 \\ 33 & 2 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 28 & 116 \\ 7 & 101 & 36 \\ 63 & 81 & 0 \\ 58 & 72 & 14 \\ 18 & 0 & 126 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 & 171 \\ 86 & 85 & 0 \\ 97 & 9 & 106 \\ 90 & 145 & 177 \\ 204 & 168 & 40 \end{bmatrix}$	
d_f	24		24		24	
Codeword weight enumerator A_w	A_{24}	5	A_{24}	5	A_{24}	5
	A_{36}	21	A_{28}	1	A_{36}	17
	A_{40}	1	A_{34}	2	A_{40}	4
	A_{42}	37	A_{36}	19	A_{44}	187
	A_{44}	171	A_{40}	1	A_{46}	390
	A_{46}	286	A_{42}	9	A_{52}	339

Note: 'D' is ignored for simplicity, and '0' means the polynomial D^0 or 1.

4.4.2 Other examples

The distance spectra of some other Tanner (m_s,3,5) LDPC convolutional codes can be obtained in the same way. Using the QC-LDPC block codes presented in [5], we construct the polynomial domain syndrome former matrix for each of the corresponding time-invariant LDPC-CCs and apply the linear dependence evaluation method to the associated super codes to estimate the distance spectrum of the original LDPC-CC. Results for three such codes are shown in Table 4.3. Interestingly, we find that all of these (m_s,3,5) LDPC-CCs have exactly five codewords with free distance 24. We note that, together with the code discussed in Section 4.4.1, each of the (m_s,3,5)-regular LDPC-CCs without empty entries[1] in the polynomial syndrome former matrix $\mathbf{H}^T(D)$ of size 5×3 has minimum free distance upper bounded by 24 and has at least five codewords with this weight. We also note that, using the bound of Mackay and Davey [44], any corresponding (J, K)-regular QC-LDPC block code has its minimum distance bounded above by $(J + 1)! = (3 + 1)! = 24$. As we find $d_f = 24$ for these (m_s,3,5) Tanner LDPC-CCs, it is an exact value.

Even though all of these $(m_s, 3, 5)$-regular LDPC-CCs have the same estimated free distance, the weight distributions turn out to be different. The first six nonzero codeword weight enumerators of these codes are listed in Table 4.3. The distance spectra of all three codes have codewords with even Hamming weights ranging from 24 to 52.

[1] An empty entry in $\mathbf{H}^T(D)$ refers to a polynomial entry that is the all-zero sequence $\mathbf{0}$(D).

Compared to the $(57, 3, 5)$ code, the $(126, 3, 5)$ code contains fewer codewords in this range. However, compared to the $(57, 3, 5)$ and $(126, 3, 5)$ codes, and despite increasing the syndrome former memory up to 204, the distance spectrum of the $(204, 3, 5)$ code does not show any improvement.

Finally, we present some examples that demonstrate the accuracy of the method for codes with larger free distance. QC-LDPC-BCs based on pre-lifted protographs were introduced in [45] to improve the girth and the minimum Hamming distance. A family of regular (3,4) QC-LDPC-BCs pre-lifted from a 3×4 base matrix is given by

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} & \mathbf{P}_1 & \mathbf{0} & \mathbf{0} & \mathbf{Q}_1 & \mathbf{0} & \mathbf{R}_1 \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{P}_2 & \mathbf{Q}_2 & \mathbf{0} & \mathbf{R}_2 & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{S}_1 & \mathbf{T}_1 & \mathbf{0} & \mathbf{U}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{S}_2 & \mathbf{0} & \mathbf{0} & \mathbf{T}_2 & \mathbf{0} & \mathbf{U}_2 \end{bmatrix}, \tag{4.18}$$

where $\mathbf{P}_i$, $\mathbf{Q}_i$, $\mathbf{R}_i$, $\mathbf{S}_i$, $\mathbf{T}_i$, and $\mathbf{U}_i$, $i = 1, 2$, are permutation matrices, $\mathbf{I}$ is the identity matrix, $\mathbf{0}$ is the all-zero matrix, and each matrix is of size $r \times r$. By choosing the permutation matrices $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{T}_1$, $\mathbf{T}_2$, $\mathbf{U}_1$, and $\mathbf{U}_2$ as the circulant matrices $\mathbf{I}_1$, $\mathbf{I}_5$, $\mathbf{I}_2$, $\mathbf{I}_{10}$, $\mathbf{I}_4$, $\mathbf{I}_{20}$, $\mathbf{I}_7$, $\mathbf{I}_3$, $\mathbf{I}_{14}$, $\mathbf{I}_6$, $\mathbf{I}_{28}$, and $\mathbf{I}_9$,[1] respectively, and setting the circulant size to $r = 41$, we obtain a QC-LDPC-BC with minimum distance bounded by $38 \leq d_{min} \leq 48$ (found using MAGMA [46]). Similarly, choosing the twelve permutation matrices as the circulant matrices $\mathbf{I}_1$, $\mathbf{I}_5$, $\mathbf{I}_{10}$, $\mathbf{I}_{10}$, $\mathbf{I}_{13}$, $\mathbf{I}_{13}$, $\mathbf{I}_7$, $\mathbf{I}_7$, $\mathbf{I}_{11}$, $\mathbf{I}_{11}$, $\mathbf{I}_2$, and $\mathbf{I}_4$, respectively, and setting the circulant size to $r = 49$, results in a QC-LDPC-BC with minimum distance bounded by $32 \leq d_{min} \leq 56$.

Two time-invariant LDPC-CCs with polynomial syndrome former matrices $\mathbf{H}_3^T(D)$ and $\mathbf{H}_4^T(D)$ can be obtained from $\mathbf{H}$ by replacing '$\mathbf{I}$' with '1' and '$\mathbf{I}_n$' with the delay operator 'D^{n-1}'. Applying the linear dependence evaluation method to these two LDPC-CCs, we find that the minimum free distances of the LDPC-CCs represented by $\mathbf{H}_3^T(D)$ and $\mathbf{H}_4^T(D)$ are upper bounded by 48 and 56, respectively. This result is consistent with the fact that minimum free distance of an LDPC-CC is an upper bound on the minimum distance of the corresponding QC-LDPC-BC [5].

Even though the polynomial syndrome former matrices of all the example codes in this chapter consist of only monomials, this technique can also be applied to more general polynomial matrices, i.e., the entries of $\mathbf{H}^T(D)$ can have weight greater than

[1] The notation $\mathbf{I}_a$ is used to denote the $r \times r$ identity matrix with each row cyclically shifted to the left by $a - 1$ positions.

one. This generalization does not increase the complexity of the algorithm.

4.5 Conclusion

In this chapter, a novel approach has been introduced to estimate the distance spectrum of time-invariant LDPC-CCs that are defined by polynomial syndrome former matrices. The concept is to split the polynomial syndrome former matrix into submatrices, each of which defines a super code. By applying the linear dependence evaluation method introduced in this chapter to the codewords of the super codes, we obtain an estimate of the distance spectrum of the original code, i.e., we obtain an upper bound on the minimum free distance and a lower bound on the number of codewords A_w with Hamming weight w. The free distance bound was shown to be exact for the Tanner (21,3,5)-regular time-invariant LDPC-CC.

In contrast to the BEAST algorithm [13], which can be used to obtain exact A_w values for convolutional codes with relatively short memories, the complexity of the proposed algorithm does not depend on the constraint length v_s or the syndrome former memory m_s; instead, the complexity depends on the free distance and the density (row and column weight J and K) of $\mathbf{H}^T$. As a result, unlike BEAST, the linear dependence evaluation method is well suited for application to practical time-invariant LDPC-CCs that have low-density syndrome former matrices and large memories.

Chapter 5

Rate-compatible punctured LDPC convolutional codes

5.1 Introduction

Rate-compatible (RC) channel codes, a form of variable rate coding, are of great practical interest since they can adapt to the changing conditions of time-varying channels and allow transceivers to employ the same encoder/decoder pair. As competitors to RC turbo codes, RC low-density parity-check block codes (LDPC-BCs) have recently been investigated using code modifying techniques such as nulling or shortening, extending, puncturing, and combining [47] - [48].

By nulling information bits and puncturing parity check bits, Tian *et al.* [47] obtained RC LDPC-BCs with the same degree distribution as the original code. A combination of puncturing and extending was used in [49] and [50] to overcome the degradation in decoding performance caused when a large number of bits are punctured in order to obtain high rate LDPC-BCs. Ha *et al.* [51,52] then proposed a puncturing scheme to obtain RC LDPC-BCs based on puncturing nodes that require fewer decoding iterations to be recovered, referred to as the *recoverability* of punctured variable nodes criterion in this chapter. Based on Ha's algorithm, Kim *et al.* [53] proposed a way to design good LDPC-BCs as mother codes that are efficiently encodable and well-suited for RC puncturing. Other methods of puncturing LDPC-BCs can be found in [54–56]. To maintain a constant code blocklength, Casado *et al.* [57] presented a method to obtain multiple rate LDPC-BCs by combining rows of the lowest-rate parity-check matrix. As an alternative to applying puncturing techniques to parity-check matrices, RC linear codes were constructed in [58] by concatenating low-density generator matrices. To im-

prove the decoding performance of RC LDPC-BCs, a layered belief propagation (BP) algorithm [18] was shown to accelerate decoding convergence by means of sequentially updating check nodes, and so-called successive maximization (SM) was proposed in [48] to design universally capacity approaching RC sequences of LDPC code ensembles over the general class of binary-input output-symmetric memoryless (BIOSM) channels.

Costello *et al.* [2] have shown that LDPC convolutional codes (LDPC-CCs) [1] can be implemented without an increase in computational complexity compared to corresponding LDPC-BCs. In addition, it has recently been shown that terminated LDPC-CC ensembles, also known as *spatially coupled* LDPC codes, have substantially better belief propagation (BP) decoding thresholds compared to corresponding block, or *uncoupled*, code ensembles [8, 59]. Therefore, extending the concept of conventional RC convolutional codes to RC punctured LDPC-CCs is of practical interest. As counterparts to RC block codes, RC convolutional codes were first introduced in [11] by periodically puncturing encoded bits chosen with respect to a distance spectrum criterion. The distance spectra of low memory of LDPC-CCs can be precisely computed by the BEAST algorithm [13], or estimated using the techniques proposed presented in Chapter 4; however, it is infeasible to precisely compute the distance spectra of practically interesting LDPC-CCs with large syndrome former memories. Moreover, distance spectrum is important for maximum likelihood (ML) decoding, whereas using (sub-optimal) iterative decoding techniques, such as BP decoding, there are many other graph-based properties that affect decoding performance, such as short cycles in the Tanner graph and trapping sets [60].

Recently, by successively extending the graph of a high-rate mother code, Nguyen *et al.* [61] presented a method to obtain RC protograph-based LDPC codes with low encoding complexity and iterative decoding thresholds within 0.2 dB of capacity, and Si *et al.* [62] constructed a family of RC LDPC-CCs for Type-II HARQ systems and a family of RC LDPC-CCs that achieve the capacity of the BEC. Rather than extending the graph of a code, in this chapter we propose a method of obtaining a family of RC punctured LDPC-CCs by analyzing several properties of the mother LDPC-CC and then periodically puncturing encoded bits with respect to the following criteria: (1) ensuring the recoverability of punctured variable nodes, (2) minimizing the number of *completely punctured cycle trapping sets* (CPCTSs), and (3) minimizing the number of punctured variable nodes involved in short cycles. One of the advantages of this method compared to designing a puncturing scheme based on the distance spectrum is that these properties can be precisely and efficiently calculated. As proposed in [11], compatible rates are realized by incrementally modifying the puncturing pattern of the

previous lower rate punctured code, i.e., the higher rate codes are embedded in the lower rate codes, which is efficient for the practical implementation.

The rest of the chapter is organized as follows: Section 5.2 defines cycle enumerators and cycle trapping sets and explains the recoverability of punctured variable nodes. Details about the puncturing criteria and the proposed puncturing algorithm are presented in Section 5.3. As examples of the algorithm, in Section 5.4, two families of RC punctured LDPC-CCs with rates 4/9, 4/8, ... 4/5, and 8/20, 8/19, ..., 8/9 are obtained from the $(21, 3, 5)$ Tanner LDPC-CC [5] by choosing a puncturing period of $P = 2$ and $P = 4$, respectively. In Section 5.5, we consider the (higher memory) $(57, 3, 5)$ Tanner LDPC-CC as a mother code, and in Section 5.6 we apply the algorithm successfully to a practically interesting LDPC-CC with monomial and binomial entries in the polynomial syndrome former matrix.

The work of this chapter were published in [63, 64].

5.2 Cycle enumerators and cycle trapping sets

In Chapter 3, we have discussed the cycle properties of LDPC-CCs. In this section, we introduce cycle enumerators and so-called cycle trapping sets for LDPC-CCs and review the concept of recoverability of punctured variable nodes presented in [52] from a convolutional code viewpoint. These three properties form the basis of the design criteria used to obtain the RC punctured LDPC-CCs introduced in Section 5.3.

5.2.1 Cycle enumerators of LDPC-CCs

Definition 1. A cycle in the Tanner graph of an LDPC-CC is defined as a finite alternating sequence of nodes (variable nodes or check nodes) connected by edges, beginning and ending with the same node, such that no node appears more than once. The length of the shortest cycle is called the girth of an LDPC-CC and is denoted by g.

A cycle $r_w(\boldsymbol{t})$ of length w, where $\boldsymbol{t} = (t_1, t_2, \ldots, t_w)$, is denoted by

$$r_w(\boldsymbol{t}) = \{v_{t_1}^{(j_1)}, c_{t_2}^{(k_1)}, \ldots, v_{t_{w-1}}^{(j_{w/2})}, c_{t_w}^{(k_{w/2})}\}, \tag{5.1}$$

where $j_i \in \{1, 2, \ldots, c\}$, $k_i \in \{1, 2, \ldots, q\}$, $i = 1, 2 \ldots, w/2$, $w \in \{4, 6, 8, \ldots\}$. The cycle $r_w(\boldsymbol{t})$ contains $w/2$ variable nodes $\{v_{t_1}^{(j_1)}, v_{t_3}^{(j_2)}, \ldots, v_{t_{w-1}}^{(j_{w/2})}\}$ and $w/2$ check nodes $\{c_{t_2}^{(k_1)}, c_{t_4}^{(k_2)}, \ldots, c_{t_w}^{(k_{w/2})}\}$. Since the time-domain syndrome former matrix is semi-infinite, an LDPC-CC has an infinite number of cycles. However, due to the periodically repeating identical structure of the Tanner graph, given a particular cycle $r_w(\boldsymbol{t})$, the set

of periodically shifted cycles $\{r_w(t_1+t,\ldots,t_w+t)|t\in\mathbb{Z}^*,\ t_i+t\geqslant 0, i=1,2,\ldots,w\}$ can be considered as one *type*.

Definition 2. The cycle enumerator for an LDPC-CC is denoted as R_w, where R_w indicates the number of types of cycles of length w, i.e., periodically shifted cycles are only counted once.

To obtain the cycle enumerators of an LDPC-CC, we use the cycle counting method introduced in Section 3.3. For example, applying this method to the $(21,3,5)$ Tanner LDPC-CC with polynomial syndrome former matrix[1]

$$\mathbf{H}^T(D)=\begin{bmatrix}1 & 1 & D^{18}\\ D & D^5 & D^{12}\\ D^3 & D^{15} & 1\\ D^7 & D^4 & D^7\\ D^{15} & D^{13} & D^{21}\end{bmatrix}, \tag{5.2}$$

the first three non-zero cycle enumerators are given by $R_8=11$, $R_{10}=62$, and $R_{12}=351$, i.e., the number of types of cycles of length 8, 10, and 12 are 11, 62, and 351, respectively. Consequently, the girth of this code is 8. All the 11 cycles of length 8 are shown in Table 5.1 (for simplicity, only variable nodes involved in short cycles are presented). One of the 11 cycles in R_8, whose variable nodes and check nodes are connected by solid lines in bold in the Tanner graph representation, is shown in Fig. 5.1. Given $\boldsymbol{t}=(t_1=0,t_2=1,t_3=1,t_4=19,t_5=7,t_6=8,t_7=1,t_8=5)$, the cycle $r_8(\boldsymbol{t})=\{v_0^{(2)},c_1^{(1)},v_1^{(1)},c_{19}^{(3)},v_7^{(2)},c_8^{(1)},v_1^{(4)},c_5^{(2)}\}$, consists of the set of variable nodes $\{v_0^{(2)},v_1^{(4)},v_7^{(2)},v_1^{(1)}\}$ and the set of check nodes $\{c_1^{(1)},c_5^{(2)},c_8^{(1)},c_{19}^{(3)}\}$. In the remainder of the chapter, we simply refer to R_w as the number of cycles of length w rather than the number of types of cycles of length w.

5.2.2 Cycle trapping sets of LDPC-CCs

Definition 3. A (d,f) trapping set $\tau_{d,f}$ of a Tanner graph is a set of variable nodes of size d that induces a subgraph with exactly f odd-degree check nodes (and an arbitrary number of even-degree check nodes).[2]

[1] Note that the common factors of D have been removed from the code in [5] for simplicity.

[2] We define a trapping set as a topology of the bipartite graph, i.e., it does not depend on a particular decoder (see [60]).

Table 5.1: Cycles in the (21,3,5) Tanner LDPC-CC

Cycle enumerators	R_8	R_{10}	R_{12}
	11	62	351
Cycles in R_8			
$\{v_0^{(2)}, v_1^{(1)}, v_7^{(2)}, v_1^{(4)}\}$		$\{v_0^{(2)}, v_5^{(1)}, v_5^{(4)}, v_{11}^{(2)}\}$	
$\{v_0^{(3)}, v_0^{(5)}, v_3^{(1)}, v_{15}^{(1)}\}$		$\{v_0^{(3)}, v_2^{(2)}, v_2^{(5)}, v_{14}^{(3)}\}$	
$\{v_0^{(4)}, v_3^{(4)}, v_7^{(1)}, v_7^{(3)}\}$		$\{v_0^{(4)}, v_4^{(1)}, v_4^{(3)}, v_{15}^{(4)}\}$	
$\{v_0^{(4)}, v_7^{(1)}, v_7^{(3)}, v_{19}^{(4)}\}$		$\{v_0^{(5)}, v_1^{(5)}, v_9^{(2)}, v_9^{(4)}\}$	
$\{v_0^{(5)}, v_5^{(5)}, v_{14}^{(2)}, v_{14}^{(4)}\}$		$\{v_0^{(5)}, v_6^{(5)}, v_{14}^{(2)}, v_{14}^{(4)}\}$	
$\{v_0^{(2)}, v_4^{(2)}, v_5^{(1)}, v_5^{(4)}\}$			

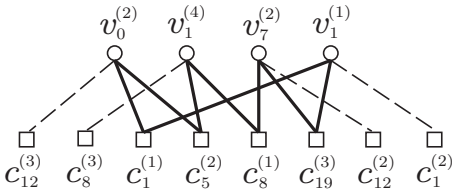


Figure 5.1: The $(4, 4)$ cycle trapping set derived from the cycle $r_8(\boldsymbol{t})$.

Definition 4. If the d variable nodes involved in a cycle of length $w = 2d$ induce a subgraph with $f > 0$ odd-degree check nodes, then the trapping set defined by the subgraph is called a $(w/2, f)$ cycle trapping set.

Trapping sets derived from short cycles, or a union of short cycles, have been shown to be the most dominant ones in terms of decoding performance in the high signal-to-noise ratio (SNR) region for the binary symmetric channel [65]. An illustration of the failure of iterative Bit Flipping (BF) decoding in the trapping set is shown in [12].

Consider the cycle $r_8(\boldsymbol{t})$ highlighted in Fig. 5.1. By attaching the remaining neighboring check nodes (connected by dashed lines) to each of the variable nodes in the subgraph, we obtain an induced subgraph that defines a $(4, 4)$ cycle trapping set. The degree-one check node set $\{c_{12}^{(3)}, c_8^{(3)}, c_{12}^{(2)}, c_1^{(2)}\}$ corresponds to the $f = 4$ odd-degree check nodes in the trapping set and the set $\{c_1^{(1)}, c_5^{(2)}, c_8^{(1)}, c_{19}^{(3)}\}$ are the $w/2 = d = 4$ even-degree check nodes. Thus each cycle in the Tanner graph corresponds to a unique cycle trapping set. Consequently, there are 11, 62, and 351 cycle trapping sets in the code defined by (5.2), containing exactly 4, 5, and 6 variable nodes, respectively.

5.2.3 m-step-recoverable (m-SR) punctured nodes

We define $\mathcal{N}(v_t^{(j)})$ as the neighboring check node set of the variable node $v_t^{(j)}$, and $\mathcal{N}(v_t^{(j)}) \setminus c_{t'}^{(k)}$ as the neighboring check node set excluding check node $c_{t'}^{(k)}$. Similar definitions apply to $\mathcal{N}(c_t^{(k)})$ and $\mathcal{N}(c_t^{(k)}) \setminus v_{t'}^{(j)}$.

In [52], the concept of a *recovery tree* was introduced to determine the number of iterations required to 'recover' a punctured variable node $v_t^{(j)}$ for LDPC-BCs. The recovery tree is obtained using a two-step process: (1) connect the punctured variable node $v_t^{(j)}$ to all of the neighboring check nodes in $\mathcal{N}(v_t^{(j)})$; and (2) then connect each check node $c_{t'}^{(k)} \in \mathcal{N}(v_t^{(j)})$ to the variable nodes $\mathcal{N}(c_{t'}^{(k)}) \setminus v_t^{(j)}$. Steps (1) and (2) form one layer of the recovery tree. This two-step process is repeated until every branch in the tree terminates with an unpunctured variable node. A check node $c_{t'}^{(k)}$ in $\mathcal{N}(v_t^{(j)})$ is called *reliable* with respect to $v_t^{(j)}$ if all of the variable nodes in $\mathcal{N}(c_{t'}^{(k)}) \setminus v_t^{(j)}$ are unpunctured nodes. Otherwise, it is called an *unreliable* check node. The minimum number of layers in the recovery tree required to have at least one reliable check node is the number of iterations needed to recover a punctured variable node $v_t^{(j)}$.

The unreliable check nodes can be explained by the *min-sum* decoding algorithm though standard sum-product algorithm (SPA) [1] is used in the simulation in this dissertation. According to the min-sum iterative decoding algorithm of LDPC codes [66,67], the message passing from a check node $c_{t'}^{(k)}$ to a variable node $v_t^{(j)}$ in the Tanner graph is dominated by the minimum absolute LLR value among the messages passing from the variable nodes in the set $\mathcal{N}(c_{t'}^{(k)}) \setminus v_t^{(j)}$ to the check node $c_{t'}^{(k)}$, i.e.,

$$|m(c_{t'}^{(k)} \to v_t^{(j)})| = \min_{v_{\bar{t}}^{(j')} \in \mathcal{N}(c_{t'}^{(k)}) \setminus v_t^{(j)}} |m(v_{\bar{t}}^{(j')} \to c_{t'}^{(k)})|, \tag{5.3}$$

where $m(c_{t'}^{(k)} \to v_t^{(j)})$ indicates the message in LLR passing from the check node $c_{t'}^{(k)}$ to the variable node $v_t^{(j)}$, while $m(v_{\bar{t}}^{(j')} \to c_{t'}^{(k)})$ is referred to the message in LLR passing from the variable node $v_{\bar{t}}^{(j')}$ to the check node $c_{t'}^{(k)}$. The reliability of a message is determined by the absolute LLR value. The larger the LLR value is, the higher reliability it has. In addition, the decoder sets the initial LLR value of a punctured node to 0. Consequently, if a variable in $\mathcal{N}(c_{t'}^{(k)}) \setminus v_t^{(j)}$ is punctured, according to (5.3), the message $m(c_{t'}^{(k)} \to v_t^{(j)})$ is equal to the message of lowest reliability passing from the punctured node. When applying SPA, the situation is even worse, i.e., the message $m(c_{t'}^{(k)} \to v_t^{(j)})$ is smaller than the lowest reliability passing from the punctured node. Therefore, the check node $c_{t'}^{(k)}$ is unreliable with respect to $v_t^{(j)}$.

Definition 5. A punctured variable node $v_t^{(j)}$ is recovered in the mth iteration, $m \geqslant 1$,

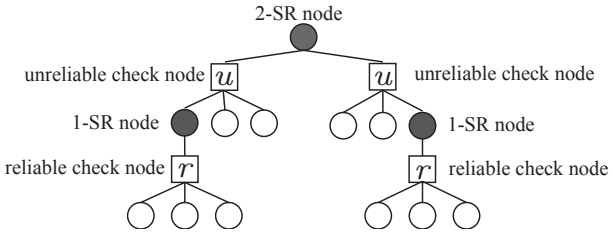


Figure 5.2: The recovery tree of a 2-SR punctured variable node.

by the iterative decoding algorithm and is called an m-step-recoverable (m-SR) node [52] if $v_t^{(j)}$ has at least one neighboring check node $c_{t'}^{(k)}$ in $\mathcal{N}(v_t^{(j)})$ such that the set $\mathcal{N}(c_{t'}^{(k)}) \setminus v_t^{(j)}$ contains at least one (m-1)-SR variable node and all of the other variable nodes in the set are n-SR nodes, where $0 \leq n \leq m-1$.

An unpunctured variable node is described as a 0-SR node. If a punctured variable node cannot be recovered in a finite number of steps, it is called *unrecoverable* and is described as an ∞-SR node. In this case, the recovery tree of the punctured variable node has no reliable check nodes. Fig. 5.2 illustrates an example of the recovery tree of a 2-SR punctured variable node. It has two unreliable check nodes, and each of them is connected to a 1-SR variable node. The squares filled with u and r denote unreliable and reliable check nodes, respectively. Filled circles correspond to punctured variable nodes, and unfilled circles are unpunctured variable nodes. Further details of the recovery tree and m-SR nodes can be found in [52].

5.3 RC Punctured LDPC-CCs

In this section, we present a method to construct RC punctured LDPC-CCs by analyzing several properties of the Tanner graphs of punctured LDPC-CCs. The advantages of this method compared to a distance spectrum based technique are that each of the properties can be precisely and efficiently calculated, even for codes with large syndrome former memories, and that these graph-based properties are directly related to the performance of iterative decoding.

Note that all of the LDPC-CCs used in this chapter are non-systematic. In the decoder, in order to correctly decode codewords back to the original message, apart from decoding the unpunctured variable nodes, we also need to recover the punctured variable nodes. As a result, when counting bits in error for BERs, both the punctured and unpunctured bits in codewords are considered.

5.3.1 Puncturing patterns of LDPC-CCs

Given a polynomial-domain syndrome former matrix of size $c \times q$, a puncturing pattern with period P is defined by the $P \times c$ matrix

$$\mathbf{a} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,c-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,c-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{P-1,0} & a_{P-1,1} & \cdots & a_{P-1,c-1} \end{bmatrix}, \tag{5.4}$$

where $a_{x,y} \in \{0, 1\}$, $x = 0, 1, \ldots, P-1$, $y = 0, 1, \ldots, c-1$. Each row of the puncturing pattern corresponds to the c encoded bits (variable nodes) at one time unit. The "1" and "0" in (5.4) imply puncturing and transmission of the associated bits, respectively. If l is the total number of ones in the puncturing pattern, $l < Pq$, the resulting punctured code has rate $R' = (Pb)/(Pc - l)$, where $b = c - q$.

5.3.2 Criteria used to design RC punctured LDPC-CCs

We now introduce the design criteria used to obtain RC punctured LDPC-CCs from a "mother code" (original unpunctured code). The compatible rates are obtained by a set of nested puncturing patterns, i.e., a higher-rate punctured code is obtained from the previous lower-rate code by including an extra punctured entry in the puncturing pattern of the previous code. This results in a RC family of codes derived from the same mother code [11].

Given a total number of punctured entries l in $\mathbf{a}$ and a puncturing period P, there are $\binom{Pc}{l}$ possible puncturing patterns that result in the same punctured code rate. However, their decoding performance varies. The best puncturing pattern will be chosen based on the criteria described in the remainder of this section.

5.3.2.1 Ensuring the recoverability of punctured variable nodes

A puncturing pattern $\mathbf{a}_i$ that generates unrecoverable punctured variable nodes, i.e., ∞-SR nodes, results in poor decoding performance in both the low and high SNR regions, since ∞-SR nodes are unable to offer reliable information to help recover other punctured nodes or to help correct unpunctured nodes received in error. We will demonstrate the dramatic degradation in decoding performance caused by ∞-SR nodes in Section 5.4.

In addition to the extreme case of an ∞-SR node, we also wish to avoid m-SR nodes with large m. The recoverability of an m-SR node is determined by the value of m. The smaller the value of m, the easier it is to recover a punctured variable node.

Definition 6. We define the element $C_m^{\mathbf{a}_i}$ in the vector $\boldsymbol{E}_c^{\mathbf{a}_i}=\begin{bmatrix} C_1^{\mathbf{a}_i} & C_2^{\mathbf{a}_i} & \cdots & C_{N_c}^{\mathbf{a}_i} \end{bmatrix}$, where $1 \leq m \leq N_c$ and $m, N_c \in \mathbb{Z}^+$, as the enumerator of m-SR nodes in the punctured code within one period of the puncturing pattern $\mathbf{a}_i$, and $C_\infty^{\mathbf{a}_i}$ is defined as the enumerator of unrecoverable punctured variable nodes.

Given a puncturing pattern $\mathbf{a}_i$ and the number l of punctured entries, N_c (the number of elements in the vector $\boldsymbol{E}_c^{\mathbf{a}_i}$) is the smallest integer that satisfies $\sum_{m=1}^{N_c} C_m^{\mathbf{a}_i} + C_\infty^{\mathbf{a}_i} = l$, $N_c \in \mathbb{Z}^+$.

A puncturing pattern $\mathbf{a}_i$ is said to be superior to puncturing pattern $\mathbf{a}_j$, $i \neq j$, with respect to the recoverability of punctured variable nodes if there exists an $M \in \mathbb{Z}^+$ such that

$$\begin{cases} C_m^{\mathbf{a}_i} = C_m^{\mathbf{a}_j}, & 1 \leq m \leq M-1 \\ C_m^{\mathbf{a}_i} > C_m^{\mathbf{a}_j}, & m = M \end{cases} . \tag{5.5}$$

In other words, puncturing pattern $\mathbf{a}_i$ is superior to pattern $\mathbf{a}_j$ if the enumerator vector of $\mathbf{a}_i$ is more weighted toward smaller values of m than that of $\mathbf{a}_j$.

5.3.2.2 Minimize the number of Completely Punctured Cycle Trapping Sets (CPCTSs)

If all of the variable nodes in a cycle trapping set are punctured, then, because of the uncertainty of the nodes in the set and the limited connectivity available to obtain helpful information from nodes outside the set, the probability that the associated symbols are decoded correctly decreases, resulting in a high error floor.

Definition 7. Given a puncturing pattern $\mathbf{a}_i$, if a cycle $r_w(\boldsymbol{t})$ has all of its $w/2$ variable nodes punctured, it corresponds to a CPCTS $\tau_{w/2,f}$, $f \in \mathbb{Z}^*$. We define the element $\mathfrak{T}_{w/2}^{\mathbf{a}_i}$ in the vector $\boldsymbol{E}_\tau^{\mathbf{a}_i}=\begin{bmatrix} \mathfrak{T}_{g/2}^{\mathbf{a}_i} & \mathfrak{T}_{(g+2)/2}^{\mathbf{a}_i} & \cdots & \mathfrak{T}_{N_\tau/2}^{\mathbf{a}_i} \end{bmatrix}$, where $w = g, g+2, g+4, \ldots, N_\tau$ and $g/2$, $N_\tau/2 \in \mathbb{Z}^+$, as the enumerator of the total number of CPCTSs $\tau_{w/2,f}$, $\forall f$, corresponding to the puncturing pattern $\mathbf{a}_i$.

The number of CPCTSs associated with a particular puncturing pattern is obtained by checking each cycle $r_w(\boldsymbol{t})$ to see if it has all of its variable nodes punctured. In Algorithm 3 (pp. 79), the sum of the elements of $\boldsymbol{E}_\tau^{\mathbf{a}_i}$ is used to compare different puncturing patterns.

5.3.2.3 Minimize the number of punctured variable nodes involved in short cycles

Short cycles existing in the Tanner graph of LDPCs are known to hinder the convergence behavior of the suboptimal iterative sum-product decoding algorithm [68].

Puncturing variable nodes in short cycles makes the situation even worse, since unreliable information is passed around a short loop. Consequently, we seek puncturing patterns that have few punctured variable nodes involved in short cycles.

Definition 8. We define the element $B_w^{\mathbf{a}_i}$ in the vector $\boldsymbol{E}_b^{\mathbf{a}_i} = \begin{bmatrix} B_g^{\mathbf{a}_i} & B_{g+2}^{\mathbf{a}_i} & \cdots & B_{N_b}^{\mathbf{a}_i} \end{bmatrix}$, where $w = g, g+2, g+4, \ldots, N_b$ and g, $N_b \in \mathbb{Z}^+$, as the enumerator of the total number of punctured variable nodes involved in all cycles of length w for the puncturing pattern $\mathbf{a}_i$.

Given a cycle $r_w(\boldsymbol{t})$ and a puncturing pattern $\mathbf{a}_i$ with period P, $B_w^{\mathbf{a}_i}$ is obtained by calculating the number of punctured variable nodes in each of the cycles in the set $\{r_w(\boldsymbol{t}), r_w(\boldsymbol{t}+\mathbf{1}), \ldots, r_w(\boldsymbol{t}+\mathbf{1}\cdot(P-1))\}$.[1] For example, for the cycle $r_8(\boldsymbol{t})$ highlighted in Fig. 5.1 (pp. 72), given the puncturing pattern $\mathbf{a}_0 = [10000; 00010]$ with $P = 2$,[2] the set of punctured variable nodes is $\{v_{0+nP}^{(1)}, v_{1+nP}^{(4)} | n = 0, 1\}$. Therefore, in the sets of variable nodes $\{v_0^{(2)}, v_1^{(4)}, v_7^{(2)}, v_1^{(1)}\}$ and $\{v_1^{(2)}, v_2^{(4)}, v_8^{(2)}, v_2^{(1)}\}$ belonging to cycles $r_8(\boldsymbol{t})$ and $r_8(\boldsymbol{t}+\mathbf{1})$, variable nodes $v_1^{(4)}$ and $v_2^{(1)}$ are punctured when $n = 0$ and $n = 1$, respectively. Applying this concept to the 11 cycles of length 8 (shown in Table 5.1 (pp. 72)) for this code, there are a total of 8 and 12 punctured variable nodes as shown in Table 5.2 in the cycles corresponding to $n = 0$ and $n = 1$, respectively. Therefore, the total number of punctured variable nodes involved in cycles of length 8 is $B_8^{\mathbf{a}_0} = 8 + 12 = 20$. When comparing puncturing patterns, we will compare the cumulative number of punctured nodes involved in short cycles, i.e., $\sum_w B_w^{\mathbf{a}_i}$ for small w.

5.3.3 Designing RC punctured LDPC-CCs

Based on the three criteria introduced in Section 5.3.2, we search for puncturing patterns $\mathbf{a}_i$ with no ∞-SR nodes and the smallest possible entries for the enumerators $\boldsymbol{E}_c^{\mathbf{a}_i}$, $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and $\boldsymbol{E}_b^{\mathbf{a}_i}$. Given a mother LDPC-CC of rate $R = b/c$, a search procedure to obtain a rate $R' = Pb/(Pc-l)$ punctured LDPC-CC based on these three criteria is described in Algorithm 3.

The influence of Steps 1 and 3 on decoding performance is felt most strongly in the waterfall region of the bit-error-rate (BER) curve, while Step 2 has a larger effect on the error floor. Also, for low rate punctured codes, even though Step 1 (specifically Step

[1] $\mathbf{1}$ is the all-one vector of length w, where w is the number of variable or check nodes involved in the corresponding cycle.

[2] In the remainder of the chapter, the puncturing pattern in (5.4) is presented as a vector with rows separated by ";".

Table 5.2: Punctured variable nodes in $r_8(\boldsymbol{t})$ and $r_8(\boldsymbol{t}+\mathbf{1})$ for the code with $\mathbf{H}^T(D)$ given in (5.2)

Puncturing pattern		P	l
$[10000; 00010]$		2	2
Punctured nodes in $r_8(\boldsymbol{t})$		Punctured nodes in $r_8(\boldsymbol{t}+\mathbf{1})$	
$\{v_1^{(4)}\}$	$\{v_5^{(4)}\}$	$\{v_2^{(1)}\}$	$\{v_6^{(1)}\}$
$\varnothing$	$\varnothing$	$\{v_4^{(1)}, v_{16}^{(1)}\}$	$\varnothing$
$\{v_3^{(4)}\}$	$\{v_4^{(1)}, v_{15}^{(4)}\}$	$\{v_1^{(4)}, v_8^{(1)}\}$	$\{v_1^{(4)}\}$
$\{v_{19}^{(4)}\}$	$\{v_9^{(4)}\}$	$\{v_1^{(4)}, v_8^{(1)}\}$	$\varnothing$
$\varnothing$	$\varnothing$	$\{v_{15}^{(4)}\}$	$\{v_{15}^{(4)}\}$
$\{v_5^{(4)}\}$		$\{v_6^{(1)}\}$	

1.2) is more critical, the selection of puncturing patterns is typically dominated by Step 3, since there is often no difference in $\boldsymbol{E}_c^{\mathbf{a}_i}$ among the various puncturing patterns under consideration. As the punctured code rate increases, the choice of the best puncturing pattern relies more heavily on Step 1.2, since the enumerator vectors $\boldsymbol{E}_c^{\mathbf{a}_i}$ show more variation in this case.

The complexity of Algorithm 3 is dominated by the computation involved in determining the variable nodes that participate in each of the short cycles. This computation only needs to be done once, however, for a given mother code, since the list of variable nodes involved in each cycle is stored. Subsequently, for each puncturing pattern, the enumerators $C_\infty^{\mathbf{a}_i}$, $\boldsymbol{E}_c^{\mathbf{a}_i}$, $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and $\boldsymbol{E}_b^{\mathbf{a}_i}$ can be efficiently computed and compared, even for large puncturing periods P. For example, it took approximately 30 seconds to obtain the list of variable nodes involved in the cycles of length 8, 10, and 12 for the $(21, 3, 5)$ Tanner LDPC-CC using a computer with a 2 GHz Processor and 2 GB of memory.

Algorithm 3 Search for RC punctured LDPC-CCs

1: Initialization: Given a mother LDPC-CC of rate $R = b/c$, set the value of the puncturing period P and the number of punctured entries l to correspond to the desired rate $R' = Pb/(Pc - l)$. If a puncturing pattern $\mathbf{a}$ with $k < l$ non-zero entries has been selected at a previous stage, form $A_P^l = \{\mathbf{a}_i\}$, where A_P^l is the set of all non-equivalent puncturing patterns $\mathbf{a}_i$ that can be obtained from $\mathbf{a}$ by including $l - k$ additional punctured entries.[1] If no previous puncturing pattern has been selected, set $\mathbf{a} = \mathbf{0}$ and $k = 0$ and generate A_P^l as described above. Given are the girth g of the mother code and set the parameters for the maximum cycle lengths N_τ and N_b that will be used to calculate the enumerator vectors $\boldsymbol{E}_\tau^{\mathbf{a}_i}$ and $\boldsymbol{E}_b^{\mathbf{a}_i}$, respectively. For each puncturing pattern $\mathbf{a}_i \in A_P^l$, calculate the enumerators $C_\infty^{\mathbf{a}_i}$, $\boldsymbol{E}_c^{\mathbf{a}_i}$, $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and $\boldsymbol{E}_b^{\mathbf{a}_i}$ and store them. Initialize $\mathcal{A}_j = \varnothing$, where $j = 1, 2, 3, 4$.

2: Step 1: *Ensure the recoverability of punctured variable nodes.*

3: Step 1.1: *Choose the puncturing patterns with the fewest ∞-SR nodes.* Calculate $\bar{C}_\infty = \min\{C_\infty^{\mathbf{a}_i} | \mathbf{a}_i \in A_P^l\}$. For each puncturing pattern $\mathbf{a}_i \in A_p^l$, if $C_\infty^{\mathbf{a}_i} = \bar{C}_\infty$, $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \cup \mathbf{a}_i$.

4: Step 1.2: *Choose puncturing patterns with the best recoverability of punctured variable nodes.* Based on $\boldsymbol{E}_c^{\mathbf{a}_i}$, if a pattern $\mathbf{a}_i \in \mathcal{A}_1$ is superior to (see (5.5)) or has the same recoverability as any other patterns in $\mathcal{A}_1 \setminus \mathbf{a}_i$, $\mathcal{A}_2 \leftarrow \mathcal{A}_2 \cup \mathbf{a}_i$.

5: Step 2: *Minimize the number of CPCTSs.* Calculate $\bar{\tau} = \min\{\sum_w \mathcal{T}_{w/2}^{\mathbf{a}_i} | \mathbf{a}_i \in \mathcal{A}_1\}$, where $w = g, g + 2, \ldots, N_\tau$. For each puncturing pattern $\mathbf{a}_i \in \mathcal{A}_2$, if $\sum_w \mathcal{T}_{w/2}^{\mathbf{a}_i} > \bar{\tau}$, $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \mathbf{a}_i$; else, $\mathcal{A}_3 \leftarrow \mathcal{A}_3 \cup \mathbf{a}_i$. If $\mathcal{A}_3 = \varnothing$, go to Step 1.2; else, go to Step 3. (Note that the loop between Steps 1.2 and 2 is guaranteed to be terminated as every loop reduces the size of set $\mathcal{A}_1$, finally there will be a pattern in $\mathcal{A}_1$ satisfying $\sum_w \mathcal{T}_{w/2}^{\mathbf{a}_i} \leq \bar{\tau}$.)

6: Step 3: *Minimize the number of punctured variable nodes involved in short cycles.* Calculate $\bar{B} = \min\{\sum_w B_w^{\mathbf{a}_i} | \mathbf{a}_i \in \mathcal{A}_3\}$, where $w = g, g + 2, \ldots, N_b$. For each puncturing pattern $\mathbf{a}_i \in \mathcal{A}_3$, if $\sum_w B_w^{\mathbf{a}_i} = \bar{B}$, $\mathcal{A}_4 \leftarrow \mathcal{A}_4 \cup \mathbf{a}_i$. If $|\mathcal{A}_4| = 1$, select this pattern; otherwise, if $|\mathcal{A}_4| > 1$, randomly choose a puncturing pattern in $\mathcal{A}_4$.

5.4 RC punctured LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC

In this section, the $(21, 3, 5)$ Tanner LDPC-CC is chosen as the mother code to illustrate the process of obtaining RC punctured LDPC-CCs. The polynomial syndrome former matrix is given by (5.2) and the code has rate $R = 2/5$, or equivalently $R = 4/10$. We

[1] Row permutations of a puncturing pattern $\mathbf{a}$ are considered to be *equivalent*. For example, the puncturing pattern $[10000; 00000]$ is equivalent to $[00000; 10000]$.

A_2^1		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, and 12		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
1	[10000; 00000]	0	[1]	[0 0 0]	[8 59 412]	479
2	[01000; 00000]	0	[1]	[0 0 0]	[10 72 447]	529
3	∗[00100; 00000]	0	[1]	[0 0 0]	[6 48 367]	421
4	[00010; 00000]	0	[1]	[0 0 0]	[12 71 426]	509
5	[00001; 00000]	0	[1]	[0 0 0]	[8 60 454]	522

Table 5.3: Enumerators for the puncturing patterns of the $(21, 3, 5)$ Tanner LDPC-CC with rate $R' = 4/9$ obtained using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

set the puncturing period $P = 2$ for demonstration and denote the puncturing pattern of the unpunctured code by $\mathbf{a} = \mathbf{0} = [00000; 00000]$. Then we can obtain a family of RC LDPC-CCs with rates 4/9, 4/8, 4/7, 4/6, and 4/5 by incrementally including one extra punctured entry in the puncturing pattern of the previous lower rate code, i.e., by changing one element in the previous puncturing pattern from "0" to "1". For this example, only the cycles of length 8, 10, and 12 were used to calculate elements in $\boldsymbol{E}_b^{\mathbf{a}_i}$ and $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, i.e., we set $N_b = N_\tau = 12$ and $g = 8$ in Algorithm 3.

5.4.1 Rate $R' = 4/9$ punctured codes

We begin by placing $l = 1$ punctured entry in the puncturing pattern $[00000; 00000]$. Consequently, there are $\binom{Pc}{l} = \binom{10}{1} = 10$ different possible puncturing patterns; but only 5 are not equivalent, i.e., $A_2^1 = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5\}$. The enumerators calculated for A_2^1 are shown in Table 5.3. Following Algorithm 3, we find that all of the puncturing patterns contain zero ∞-SR nodes and one 1-SR node. Thus, all the patterns have the same recoverability of punctured nodes and are equally good choices with respect to Step 1. Moreover, none of the patterns contain any CPCTSs (Step 2). However, according to Step 3, we find that puncturing pattern $\mathbf{a}_3$ (marked by an asterisk in Table 5.3) is superior to the others, since it has the smallest number of punctured variable nodes involved in short cycles.

Simulation curves for the five puncturing patterns of the rate $R' = 4/9$ punctured $(21, 3, 5)$ Tanner LDPC-CC are given in Fig. 5.3. All of the simulations in this dissertation were carried out assuming binary phase-shift keyed (BPSK) modulation on an

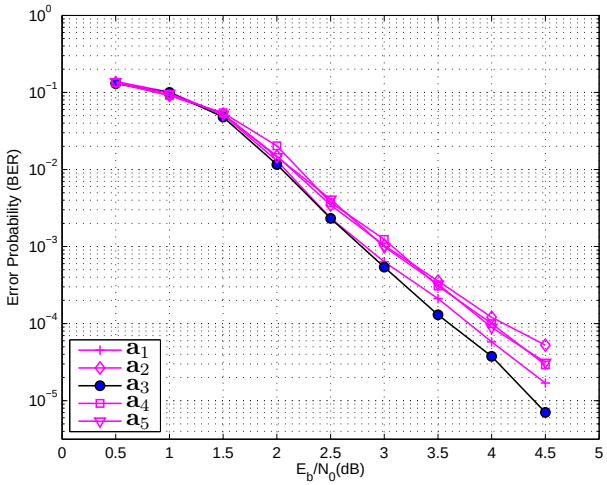


Figure 5.3: BERs of the punctured LDPC-CCs of (5.2) with rate R'=4/9.

additive white Gaussian noise (AWGN) channel with 50 iterations of the on-demand variable node activation [37] sum-product pipeline decoding algorithm [1] for LDPC-CCs.

The simulation results confirm the superiority of puncturing pattern $\mathbf{a}_3$ compared to the other patterns. In particular, we see that $\mathbf{a}_3$ has approximately 0.1 to 0.5dB SNR gains over the other puncturing patterns for a BER of 10^{-4}. In addition, we find that patterns $\mathbf{a}_1$ and $\mathbf{a}_2$ perform as the second best and the worst patterns, respectively. This is consistent with the fact that $\mathbf{a}_1$ and $\mathbf{a}_2$ have the second least and the largest numbers of punctured variable nodes in short cycles, viz., $\sum \boldsymbol{E}_b^{\mathbf{a}_1} = 479$ and $\sum \boldsymbol{E}_b^{\mathbf{a}_2} = 529$. Pattern $\mathbf{a}_5$ has, in total, more punctured nodes involved in cycles than $\mathbf{a}_4$; however, from the enumerator vector $\boldsymbol{E}_b^{\mathbf{a}_i}$, we see that $\mathbf{a}_5$ has fewer punctured nodes for cycles of length 8 and 10. This trade-off results in $\mathbf{a}_4$ and $\mathbf{a}_5$ having almost the same BER performance.

5.4.2 Rate $R' = 4/8$ punctured codes

Based on the previous rate $R' = 4/9$ code with puncturing pattern $\mathbf{a}_3 = [00100; 00000]$, a punctured code with rate $R' = 4/8$ is obtained by replacing one "0" in $\mathbf{a}_3$ with a "1". Note that the rate 4/8 code is rate-compatible with the rate 4/9 code. There are 9 possibilities for A_2^2, as shown in Table 5.4.

Of the puncturing patterns in A_2^2, $\mathbf{a}_7$ results in two punctured variable nodes within

A_2^2		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, and 12		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
6	[10100; 00000]	0	[2]	[0 0 0]	[14 107 779]	900
7	[01100; 00000]	2	[0]	[0 0 1]	[16 120 814]	950
8	[00110; 00000]	0	[2]	[0 0 0]	[18 119 793]	930
9	[00101; 00000]	0	[2]	[0 0 0]	[14 108 821]	943
10	[00100; 10000]	0	[2]	[0 0 0]	[14 107 779]	900
11	[00100; 01000]	0	[2]	[0 0 0]	[16 120 814]	950
12	$*$[00100; 00100]	0	[2]	[0 0 0]	[12 96 734]	842
13	[00100; 00010]	0	[2]	[0 0 0]	[18 19 793]	930
14	[00100; 00001]	0	[2]	[0 0 0]	[14 108 821]	943

Table 5.4: Enumerators for the puncturing patterns of the $(21, 3, 5)$ Tanner LDPC-CC with rate $R' = 4/8$ obtained using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

one puncturing period. These two punctured variable nodes are denoted by $v_t^{(2)}$ and $v_t^{(3)}$. According to Algorithm 3, $\mathbf{a}_7$ is immediately eliminated at Step 1.1 as a result of the two unrecoverable punctured variable nodes (additionally, is the only pattern with a CPCTS). The punctured variable nodes $v_t^{(2)}$ and $v_t^{(3)}$ resulting from pattern $\mathbf{a}_7$ are unrecoverable since each of their recovery trees has no reliable check nodes. Figs. 5.4 and 5.5 show one iteration of the recovery tree for the punctured variable nodes $v_t^{(2)}$ and $v_t^{(3)}$, respectively. Observe that the neighboring check nodes of $v_t^{(2)}$, i.e., $c_{t+1}^{(1)}$, $c_{t+5}^{(2)}$, and $c_{t+12}^{(3)}$, are all unreliable because they are connected, in turn, to punctured variable nodes $v_{t-2}^{(3)}$, $v_{t-10}^{(3)}$, and $v_{t+12}^{(3)}$, respectively. Similarly, $\mathcal{N}(v_t^{(3)})$, i.e., $c_{t+3}^{(1)}$, $c_{t+15}^{(2)}$, and $c_t^{(3)}$, are unreliable because they are connected to punctured variable nodes $v_{t+2}^{(2)}$, $v_{t+10}^{(2)}$, and $v_{t-12}^{(2)}$, respectively. To obtain the complete recovery tree of $v_t^{(2)}$, the punctured variable nodes $v_{t-2}^{(3)}$, $v_{t-10}^{(3)}$, and $v_{t+12}^{(3)}$ in Fig. 5.4 are replaced by the tree for $v_t^{(3)}$ (with the time indices changed accordingly). By extending the tree at each level in this fashion, we obtain the complete recovery tree of $v_t^{(2)}$ in Fig. 5.6. It is observed that every check node in the recovery tree is unreliable, and consequently punctured node $v_t^{(2)}$ is unrecoverable. A similar explanation applies to the punctured variable node $v_t^{(3)}$ for $\mathbf{a}_7$.

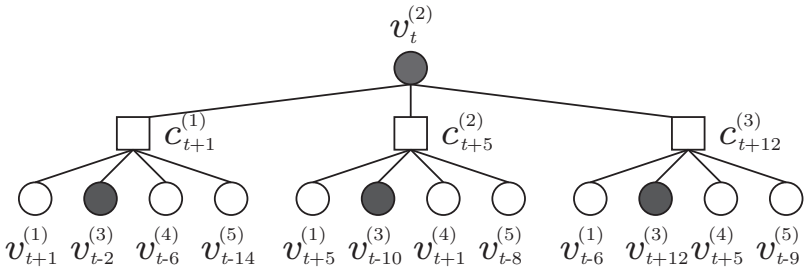


Figure 5.4: One iteration of the recovery tree for punctured variable node $v_t^{(2)}$ and puncturing pattern $\mathbf{a}_7$.

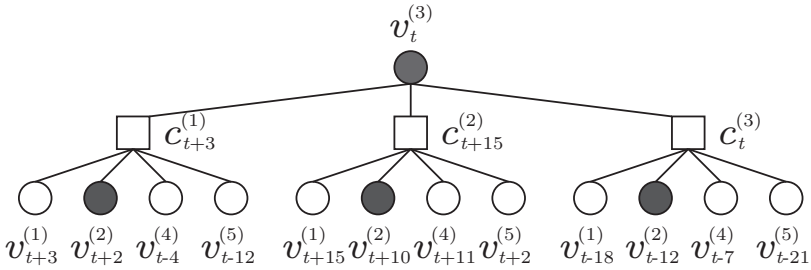


Figure 5.5: One iteration of the recovery tree for punctured variable node $v_t^{(3)}$ and puncturing pattern $\mathbf{a}_7$.

Fig. 5.7 presents the simulation results for the rate 4/8 punctured codes with puncturing patterns in A_2^2. Due to the existence of two ∞-SR nodes, puncturing pattern $\mathbf{a}_7$ has the worst performance, with a visibly inferior BER curve compared to the others. At Step 1, we find that all the puncturing patterns in $A_2^2 \backslash \mathbf{a}_7$ have the same recoverability of punctured nodes, i.e., they all have zero ∞-SR nodes and two 1-SR nodes. In addition, no CPCTSs exist for puncturing patterns in $A_2^2 \backslash \mathbf{a}_7$. However, at Step 3, we observe that pattern $\mathbf{a}_{12}$ has the smallest number of punctured nodes involved in short cycles, and we see in Fig. 5.7 that this pattern also results in the lowest BER, confirming our choice.

5.4.3 RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 2$

For each higher rate, we follow the design procedure of Algorithm 3 to choose the best candidate puncturing pattern. As in the rate 4/9 and 4/8 cases, our choice was confirmed to be the best via code simulations. Details of the properties of punctured codes and corresponding simulations for each code rate are given in Appendix B. Finally,

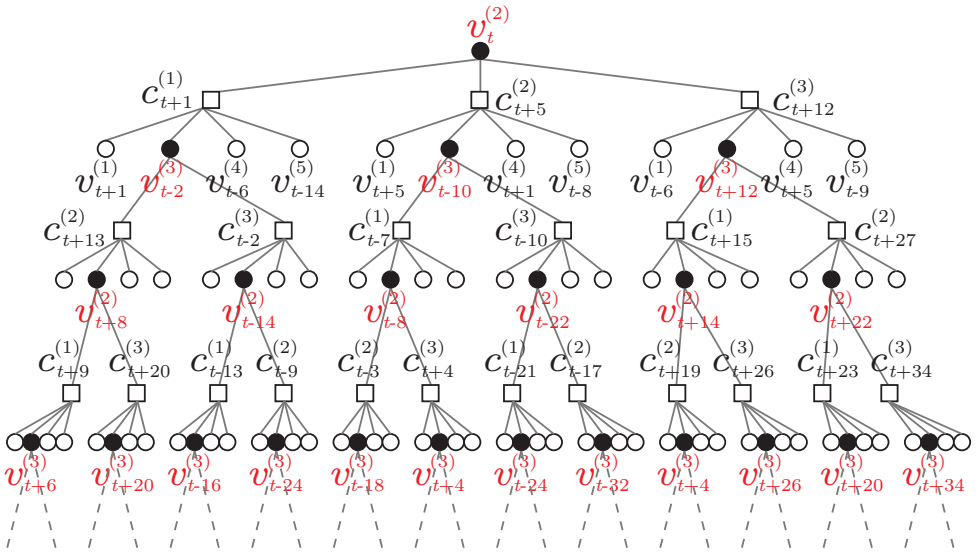


Figure 5.6: Recovery tree of the punctured node $v_t^{(2)}$ for the puncturing pattern $\mathbf{a}_7$.

we obtain six RC LDPC-CCs with enumerators given in Table 5.5. As the number of punctured entries increases, the recoverability of punctured variable nodes degrades and the number of punctured variable nodes involved in short cycles increases. CPCTSs only appear in the high rate (4/6 and 4/5) punctured codes. In other words, the characteristics of punctured codes get worse as the punctured code rate increases.

Fig. 5.8 shows the simulated performance of the family of RC punctured codes derived from the $(21, 3, 5)$ Tanner LDPC-CC. The BER curves from left to right correspond to the codes with rates 4/10, 4/9, 4/8, 4/7, 4/6, and 4/5, respectively. As expected, we observe that the decoding performance gets worse as the punctured code rate increases. We also see that, due to the large number of punctured entries in the puncturing pattern for the code of rate $R' = 4/5$, ∞-SR nodes are unavoidable (no puncturing pattern with $P = 2$ and $l = 5$ exists that avoids creating ∞-SR nodes). As a result, the BER curve of the associated code is significantly worse than the others. In addition, the chosen $R' = 4/5$ punctured code has many CPCTSs, in particular, $\boldsymbol{E}_{\tau}^{\mathbf{a}_{18}} = [1\ \ 1\ \ 6]$. Together with the ∞-SR nodes, this results in poor BER performance and the existence of a high error floor.

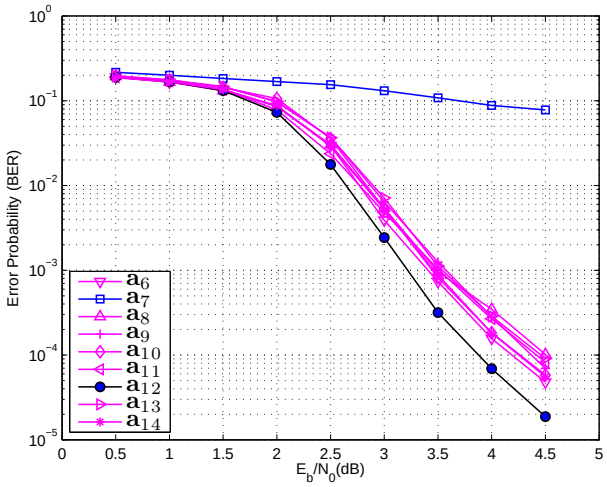


Figure 5.7: BERs of the punctured LDPC-CCs of (5.2) with rate $R' = 4/8$.

R'	i	$\mathbf{a}_i$	$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 8$,10, and 12		
					$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
4/10	18	[00000; 00000]	0	-	[0 0 0]	[0 0 0]	0
4/9	3	[00100; 00000]	0	[1]	[0 0 0]	[6 48 367]	421
4/8	12	[00100; 00100]	0	[2]	[0 0 0]	[12 96 734]	842
4/7	15	[10100; 00100]	0	[2 1]	[0 0 0]	[20 155 1146]	1321
4/6	16	[10100; 00110]	0	[2 2]	[1 0 0]	[32 226 1572]	1830
4/5	17	[10100; 01110]	2	[1 1 1]	[1 1 6]	[42 298 2019]	2359

Table 5.5: Enumerators for the RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 2$ obtained using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

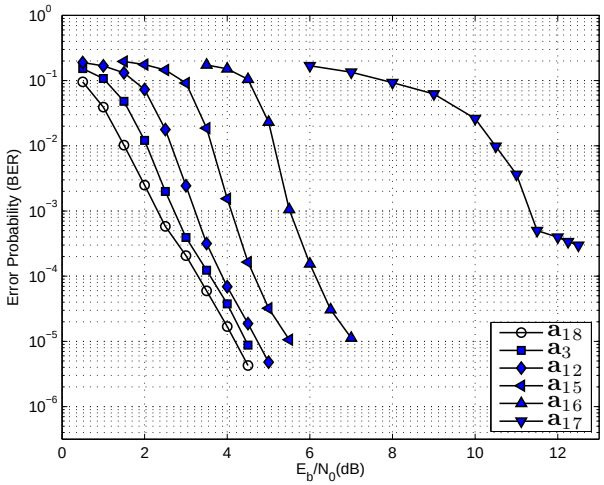


Figure 5.8: BERs of the RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner code with puncturing period $P = 2$.

5.4.4 RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 4$

By increasing the puncturing period to four, i.e., $P = 4$, and applying Algorithm 3 to the same (21, 3, 5) Tanner LDPC-CC, we obtain a family of punctured RC LDPC-CCs with rates 8/20, 8/19, 8/18, 8/17, 8/16, 8/15, 8/14, 8/13, 8/12, 8/11, 8/10, and 8/9. The corresponding puncturing patterns of these codes are given in Table 5.6. Compared to the family of punctured codes with $P = 2$ in Section 5.4.3, increasing the puncturing period to $P = 4$ extends the range of punctured code rates and has more intermediate rates.

The enumerators for these RC LDPC-CCs are also listed in Table 5.6 and the simulated BER performance is shown in Fig. 5.9, where, as with the examples in Sections 5.4.1 and 5.4.2, each punctured code selected was confirmed to be the best by simulation. Similar to the situation for the RC codes in Section 5.4.3, the properties of these codes tend to be worse as the rate increases, i.e., as the number of punctured entries increases. Consequently, we see in Fig. 5.9 that, just as in the case of $P = 2$, the decoding performance becomes worse and the gap between neighboring BER curves gets larger as we increase the punctured code rate. Among these punctured codes, only the code with the highest rate $R' = 8/9$ contains ∞-SR nodes. Together with its large number of CPCTSs ($\boldsymbol{E}_{\tau}^{\mathbf{a}_{30}} = [4\ \ 7\ \ 9]$), this results in a BER curve that remains constant

R'	i	$\mathbf{a}_i$	$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, and 12		
					$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
8/20	19	[00000; 00000; 00000; 00000]	0	-	000	0 0 0	0
8/19	20	[00100; 00000; 00000; 00000]	0	1	000	6 48 367	421
8/18	21	[00100; 00100; 00000; 00000]	0	2	000	12 96 734	842
8/17	22	[00100; 00100; 00100; 00000]	0	3	000	18 144 1101	1263
8/16	23	[00100; 00100; 00100; 00100]	0	4	000	24 192 1468	1684
8/15	24	[10100; 00100; 00100; 00100]	0	4 1	000	32 251 1880	2163
8/14	25	[10100; 00100; 10100; 00100]	0	4 2	000	40 310 2292	2642
8/13	26	[10100; 00110; 10100; 00100]	0	4 3	000	52 381 2718	3151
8/12	27	[10100; 00110; 11100; 00100]	0	3 3 2	000	62 453 3165	3680
8/11	28	[10100; 00110; 11101; 00100]	0	1111221	100	70 513 3619	4202
8/10	29	[10100; 00110; 11101; 00110]	0	111111121	334	82 584 4045	4711
8/9	30	[10100; 10110; 11101; 00110]	8	1 1 1	479	90 643 4457	5190

Table 5.6: Enumerators for the RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 4$ using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

at a BER of $\approx 10^{-1}$ with no evidence of a waterfall region even though the SNR is increased to beyond 12 dB.

Importantly, we see here that extending the puncturing period allows us to avoid ∞-SR nodes over a wider range of code rates. Recall from Section 5.4.3 that there are two unavoidable ∞-SR nodes in the code of rate 4/5 with $P = 2$; however, with $P = 4$ the code of rate 8/10 whose puncturing pattern is $\mathbf{a}_{29} = [10100; 00110; 11101; 00110]$ is free of ∞-SR nodes, resulting in about 2.1 dB gain at a BER of 10^{-3} and no discernible error floor down to 10^{-6} compared to the rate 4/5 code (see Fig. 5.10).

5.4.5 Discussion of puncturing criteria ordering

To show the existence of error floors occurring in the high SNR region caused by CPCTSs, the decoding performance of two codes of punctured code rate $R' = 8/10$ with

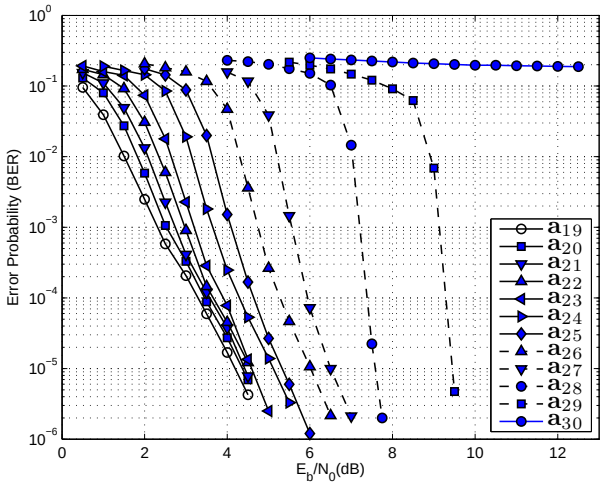


Figure 5.9: BERs of the RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 4$.

patterns $\mathbf{a}_{31} = [10100; 00111; 11100; 00110]$ and $\mathbf{a}_{32} = [10100; 01111; 10100; 00110]$,[1] obtained by puncturing the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 4$, are presented in Fig. 5.10. They have the same recoverability of punctured nodes ($C_{\infty}^{\mathbf{a}_{31}} = C_{\infty}^{\mathbf{a}_{32}} = 0$ and $\boldsymbol{E}_c^{\mathbf{a}_{31}} = \boldsymbol{E}_c^{\mathbf{a}_{32}} = [3\ 2\ 1\ 1\ 2\ 1]$) and the same number of punctured variable nodes involved in short cycles ($\sum \boldsymbol{E}_b^{\mathbf{a}_{31}} = \sum \boldsymbol{E}_b^{\mathbf{a}_{32}} = 4711$). Consequently, they have similar decoding performance in the waterfall region, as shown in Fig. 5.10. Compared to the code with puncturing pattern $\mathbf{a}_{29}$, however, patterns $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$ have better recoverability of punctured nodes ($\boldsymbol{E}_c^{\mathbf{a}_{31}} = \boldsymbol{E}_c^{\mathbf{a}_{32}} = [3\ 2\ 1\ 1\ 2\ 1]$ is superior to $\boldsymbol{E}_c^{\mathbf{a}_{29}} = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 2\ 1]$), and it is confirmed by the BER curves in Fig. 5.10 that they have better decoding performance in the waterfall region than $\mathbf{a}_{29}$. However, in terms of CPCTSs, the codes with patterns $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$ ($\boldsymbol{E}_{\tau}^{\mathbf{a}_{31}} = [3\ 2\ 12]$ and $\boldsymbol{E}_{\tau}^{\mathbf{a}_{32}} = [4\ 4\ 18]$) have many more CPCTSs than the code with pattern $\mathbf{a}_{29}$ ($\boldsymbol{E}_{\tau}^{\mathbf{a}_{29}} = [3\ 3\ 4]$). Consequently, the simulation curves in Fig. 5.10 show that the codes with patterns $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$ suffer from error floors at BER values of 10^{-6} and 10^{-5}, respectively, while there is no observed error floor for the code with pattern $\mathbf{a}_{29}$ in this range.

Note that, even though the patterns $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$ have better decoding performance in the waterfall region for $R' = 8/10$, following Algorithm 3, $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$ cannot be derived from the lower rate ($R' = 8/11$) pattern $\mathbf{a}_{28}$, i.e., they are not rate compatible to

[1] Note that, unlike the puncturing pattern $\mathbf{a}_{29}$, the patterns $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$ are not derived from the same lower rate punctured code, i.e., they are not RC puncturing patterns.

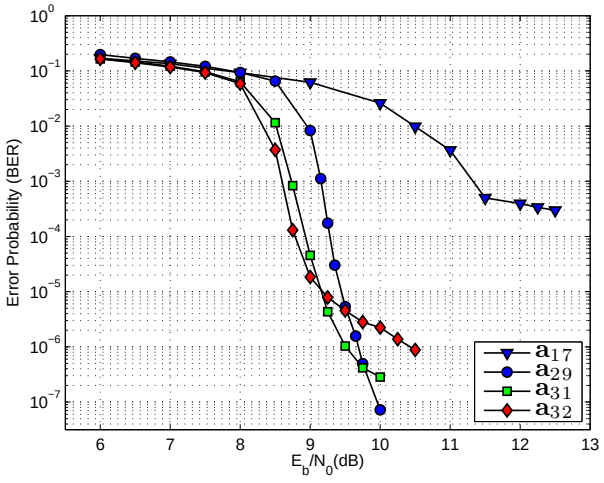


Figure 5.10: BERs of some punctured LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 2$ or $P = 4$.

pattern $\mathbf{a}_{28}$. Instead, we selected $\mathbf{a}_{29}$ as the best rate compatible candidate. This choice was made because in Algorithm 3 we consider the criterion of minimizing the number of CPCTSs to be more important than the criterion of ensuring the recoverability of m-SR nodes, where m is a finite number. In this regard, we note that Algorithm 3 can be adapted depending on the code designers' preference of improved performance in the waterfall region versus the location of the error floor. In our case, Algorithm 3 has been designed with an emphasis on selecting the robust RC puncturing patterns that yield low error floors over a wide range of code rates.

Finally, for comparison, we show in Fig. 5.10 the $R' = 4/5$ punctured code obtained in Section 5.4.3 with puncturing pattern $\mathbf{a}_{18} = [10100; 01110]$, puncturing period $P = 2$, and CPCTS enumerator $\boldsymbol{E}_{\tau}^{\mathbf{a}_{18}} = [1\ \ 1\ \ 6]$. This pattern is equivalent to a $P = 4$ pattern with $\mathbf{a}_{33} = [\mathbf{a}_{18}; \mathbf{a}_{18}] = [10100; 01110; 10100; 01110]$ whose corresponding CPCTS and ∞-SR enumerators are $\boldsymbol{E}_{\tau}^{\mathbf{a}_{33}} = [2\ \ 2\ \ 12]$ and $C_{\infty}^{\mathbf{a}_{33}} = 4$, respectively. Compared to the puncturing patterns $\mathbf{a}_{31}$ and $\mathbf{a}_{32}$, pattern $\mathbf{a}_{33}$ has fewer CPCTSs; however, the corresponding $R' = 8/10$ code (equivalent to the $R' = 4/5$ code) displays a much higher error floor, as shown in Fig. 5.10. This is because of the existence of the four ∞-SR nodes in $\mathbf{a}_{33}$, while $\mathbf{a}_{29}$, $\mathbf{a}_{31}$, and $\mathbf{a}_{32}$ have none. In other words, even though minimizing the number of CPCTSs is crucial to avoid the early onset of an error floor, the avoidance of ∞-SR nodes is of even greater importance (and hence the emphasis on Step 1.1 in Algorithm 1).

R'	i	$\mathbf{a}_i$	$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14		
					$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
6/15	34	[00000; 00000; 00000]	0	-	[000]	[0 0 0]	0
6/14	35	[00010; 00000; 00000]	0	[1]	[000]	[18 161 1241]	1420
6/13	36	[00010; 00010; 00000]	0	[2]	[000]	[36 322 2482]	2840
6/12	37	[00010; 00010; 00010]	0	[3]	[000]	[54 483 3723]	4260
6/11	38	[00110; 00010; 00010]	0	[3 1]	[000]	[74 657 5021]	5752
6/10	39	[00110; 01010; 00010]	0	[3 2]	[000]	[97 834 6322]	7253
6/9	40	[01110; 01010; 00010]	0	[1 2 2 1]	[000]	[120 1011 7623]	8754
6/8	41	[01111; 01010; 00010]	0	[1 1 1 1 2 1]	[002]	[141 1199 9019]	10359
6/7	42	[01111; 01010; 00110]	2	[1 1 1 1 2]	[2736]	[161 1373 10317]	11851

Table 5.7: Enumerators for the RC LDPC-CCs derived from the $(57, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 3$ using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

5.5 RC punctured LDPC-CCs derived from the $(57, 3, 5)$ Tanner LDPC-CC

The puncturing algorithm can also be applied to higher memory codes. In this section, we use the $(57, 3, 5)$ Tanner LDPC-CC with rate $R = 2/5$ as the mother code. This code has a syndrome former memory almost three times as large as that of the $(21, 3, 5)$ Tanner LDPC-CC. The polynomial syndrome former matrix is given by

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & 1 & D^{35} \\ D^8 & D^{43} & D^{45} \\ D^{19} & D^3 & D^{13} \\ D^{57} & D^9 & D^{30} \\ D^{33} & D^2 & 1 \end{bmatrix}. \tag{5.6}$$

The graph has a girth of 10 and the first three nonzero cycle enumerators are $R_{10} = 21$, $R_{12} = 148$, and $R_{14} = 947$. The puncturing period is set to $P = 3$, and cycles of length 10, 12, and 14 were used to compute the enumerators, i.e., we set $N_b = N_\tau = 14$ and $g = 10$ in Algorithm 3. It took less than one minute to obtain the list of variable nodes involved in the cycles of length 10, 12, and 14 and to compute the enumerators used for code selection using a computer with a 2 GHz Processor and 2 GB of memory. The enumerators and the decoding performance of the obtained RC

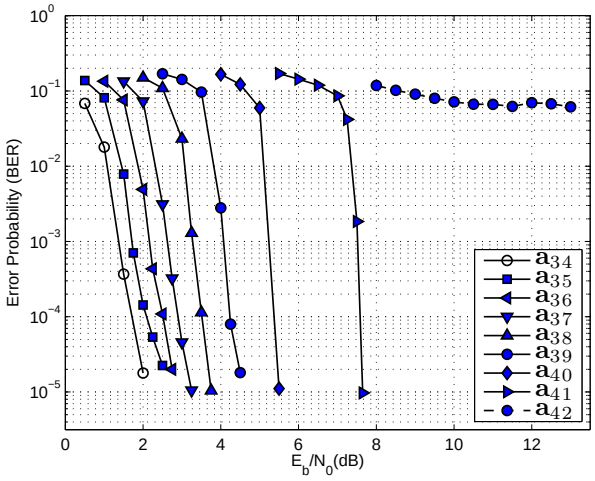


Figure 5.11: BERs of the RC LDPC-CCs derived from the $(57, 3, 5)$ Tanner code with puncturing period $P = 3$.

LDPC-CCs are presented in Table 5.7 and Fig. 5.11, respectively. The BER curves from left to right in Fig. 5.11 correspond to codes of rates 6/15, 6/14, 6/13, 6/12, 6/11, 6/10, 6/9, 6/8, and 6/7, respectively. Details of the process of obtaining the family of RC punctured LDPC-CCs from the $(57, 3, 5)$ Tanner LDPC-CC are illustrated in Appendix C.

After evaluating the enumerators and decoding performance of the RC codes derived from the $(57, 3, 5)$ Tanner LDPC-CCs, we arrive at conclusions similar to the examples in Section 5.4, i.e., a family of robust RC LDPC-CCs with good performance is obtained until the first occurrence of unavoidable ∞-SR nodes. Comparing the decoding performance in Fig. 5.11 to that in Fig. 5.8 or Fig. 5.9, the RC codes derived from the $(57, 3, 5)$ Tanner LDPC-CC generally have a steeper waterfall region than those derived from the $(21, 3, 5)$ Tanner LDPC-CC, due to the strength of the mother code. If the mother code has better decoding performance, typically the corresponding RC codes will also benefit.

5.6 Application of the puncturing algorithm to general time-invariant LDPC-CCs

So far, the puncturing algorithm has been applied to some example codes whose polynomial syndrome former matrices contain only monomial entries. As a final example, we

R'	i	$\mathbf{a}_i$	$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, 12, and 14		
					$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
2/8	43	[0000; 0000]	0	-	[0 0 0 0]	[0 0 0 0]	0
2/7	44	[0010; 0000]	0	[1]	[0 0 0 0]	[0 3 18 45]	66
2/6	45	[0010; 0010]	0	[2]	[0 0 0 0]	[0 6 36 90]	132
2/5	46	[0011; 0010]	0	[2 1]	[0 0 0 0]	[4 12 48 151]	215
2/4	47	[0111; 0010]	0	[2 2]	[0 0 0 0]	[7 18 63 211]	299
2/3	48	[0111; 0110]	0	[1 1 1 1]	[1 0 0 3]	[10 24 78 271]	383

Table 5.8: Enumerators for the RC LDPC-CCs derived from the LDPC-CC in (5.7) with puncturing period $P = 2$ using cycles of length $w = 8$, 10, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

demonstrate the compatibility of the puncturing algorithm with more general LDPC-CCs. In particular, we apply it to an LDPC-CC whose polynomial syndrome former matrix consists of both monomials and binomials, as well as some zero entries. This code, found in [69], is defined by the polynomial syndrome former matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} D + D^2 & D^5 & 0 \\ 0 & D^9 & D^{19} + D^{25} \\ D^4 & D^{10} + D^{20} & 0 \\ D^8 & 0 & D^7 + D^{14} \end{bmatrix}. \tag{5.7}$$

It has rate $R = 1/4$ and a girth of 8. The first four nonzero cycle enumerators are $R_8 = 2$, $R_{10} = 4$, $R_{12} = 10$, and $R_{14} = 31$. Given the puncturing period $P = 2$ and setting $N_b = N_\tau = 14$ and $g = 8$ in Algorithm 3, we obtain a family of RC punctured LDPC-CCs with rates 2/8, 2/7, 2/6, 2/5, 2/4, and 2/3. The calculated enumerators for the selection criteria are given in Table 5.8. Fig. 5.12 presents the BER curves of the RC codes with ascending rates from left to right. We observe that all the codes are free of ∞-SR nodes. The highest rate $R' = 2/3$ code contains four CPCTSs. Compared to other codes, it has worse recoverability of punctured variable nodes. This causes a large gap in performance between the codes of rates 2/4 and 2/3 and a high error floor in the rate 2/3 case. In order to improve the performance of the highest rate code and maintain rate compatibility, it would again be necessary to increase the puncturing

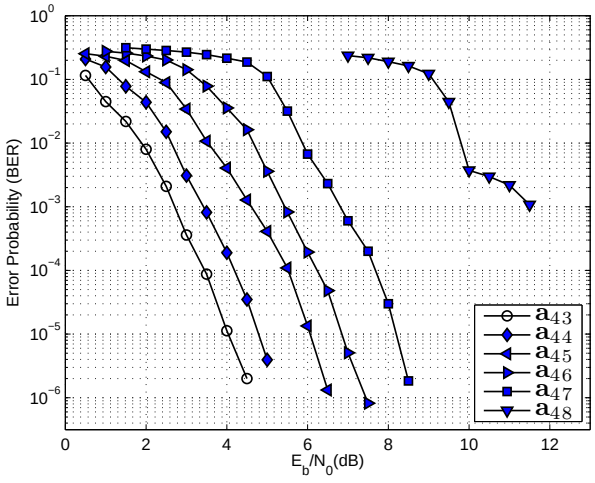


Figure 5.12: BERs of the RC LDPC-CCs derived from the LDPC-CC in (5.7) with puncturing period $P = 2$.

period to $P > 2$.

5.7 Conclusions

Given a mother time-invariant LDPC-CC and a puncturing period P, a novel method has been presented to find robust RC punctured LDPC-CCs by periodically puncturing encoded bits (variable nodes). The positions of the punctured entries in the puncturing pattern are determined using several criteria: (1) ensure good recoverability of punctured variable nodes, (2) minimize the number of CPCTSs, and (3) minimize the number of punctured variable nodes involved in short cycles. Crucially, these criteria can be efficiently and precisely computed, even for practically interesting time-invariant LDPC-CCs with large syndrome former memories, and we do not require the computation of code distance spectra. The influence of criteria (1) and (3) on iterative decoding performance is strongest in the waterfall region, while (2) has a larger effect on the position of the error floor. We observed that the most important criterion is to avoid punctured unrecoverable variable nodes, i.e., ∞-SR nodes. As the punctured code rate increases, the RC codes demonstrate robustly good decoding performance, and the robustness is maintained until a rate is reached where ∞-SR nodes cannot be avoided. Moreover, we showed that increasing the puncturing period P has the effect of eliminating ∞-SR nodes in high rate punctured codes, in addition to extending the range

of compatible rates and increasing the number of intermediate rates.

Finally, we demonstrated the versatility of the puncturing algorithm by applying it to several example mother codes with different puncturing periods, syndrome former memories, and syndrome former matrix structures.

Chapter 6

Conclusions and further work

In the dissertation, we analyzed the cycle structure of LDPC-CCs, obtained girth bounds as a result of unavoidable cycles, and we estimated the distance spectrum and the minimum free distance of LDPC-CCs. By analyzing the characteristics of LDPC-CCs, we proposed a puncturing algorithm to obtain rate compatible (RC) LDPC-CCs by periodically puncturing encoded bits. Some remarkable conclusions are given as follows.

Firstly, the relationship of cycle formations between time- and polynomial-domain syndrome former matrices for both time-invariant and time-varying LDPC-CCs has been analyzed. Time-invariant LDPC-CCs are defined by $\mathbf{H}^T(D)$ that is obtained from QC LDPC-BCs by replacing the circulant matrices '$\mathbf{I}$' in $\mathbf{H}^T$ with '1' and '$\mathbf{I}_n$' with the delay operator 'D^{n-1}', while time-varying LDPC-CCs can be derived by unwrapping the matrix $\mathbf{H}^T$. Based on the connectivities between monomials in the polynomial-domain syndrome former matrix $\mathbf{H}^T(D)$, we presented a cycle counter algorithm to examine the cycle enumerators of time-invariant and time-varying LDPC-CCs. The maximum achievable cycle length in the numerical search (limited by algorithmic complexity) depends on the row and column weights of $\mathbf{H}^T(D)$ as well as the period T. In addition, even though the cycle counter algorithm is proposed for LDPC-CCs, it is universally applicable to any convolutional code defined by a polynomial matrix.

Moreover, we investigated some unavoid able cycles with lengths ranging from 6 to 20 that are caused by destructive structures in the polynomial matrix $\mathbf{H}^T(D)$, no matter what powers of monomials are chosen. Based on the analysis we conjecture that, to obtain a "good" LDPC-CC with respect to its (large) girth, rather than polynomial entries with large weight, monomials and empty entries are preferred or required in $\mathbf{H}^T(D)$ since small cycles are avoidable, and larger girth is achieved.

Secondly, rather than to compute the exact distance spectrum of LDPC-CCs, a novel approach has been introduced to estimate the distance spectrum of time-invariant LDPC-CCs that are defined by polynomial syndrome former matrices. The concept is to split the polynomial syndrome former matrix into submatrices, each of which defines a super code. By applying the linear dependence evaluation method introduced in Chapter 4 to the codewords of the super codes, we obtain an estimate of the distance spectrum of the original code, i.e., we obtain an upper bound on the minimum free distance and a lower bound on the number of codewords A_w with Hamming weight w. The free distance bound was shown to be exact for the Tanner $(21, 3, 5)$-regular time-invariant LDPC-CC. In addition, after testing some Tanner $(m_s, 3, 5)$ LDPC-CCs with only monomials and no empty entries in $\mathbf{H}^T(D)$, we find that all of these codes have free distance 24 and at least 5 minimum weight codewords.

In contrast to the BEAST algorithm [13], which can be used to obtain exact A_w values for convolutional codes with relatively short memories, the complexity of the proposed algorithm does not depend on the constraint length v_s or the syndrome former memory m_s; instead, the complexity depends on the free distance and the density (row and column weight J and K) of $\mathbf{H}^T$. As a result, unlike BEAST, the linear dependence evaluation method is well suited for application to practical time-invariant LDPC-CCs that have low-density syndrome former matrices and large memories.

Finally, given a mother time-invariant LDPC-CC and puncturing period P, a method has been presented to find robust RC punctured LDPC-CCs by periodically puncturing encoded bits (variable nodes). The RC higher rate punctured code is obtained by placing an extra punctured entry in the puncturing pattern of the lower rate punctured code. The positions of the punctured entries in the puncturing pattern are determined considering the recoverability of punctured variable nodes, the number of CPCTSs, and the number of punctured variable nodes involved in short cycles. We observed that the most important criterion is to avoid unrecoverable punctured variable nodes, i.e., ∞-SR nodes. As the punctured code rate increases, the RC codes demonstrate robustly good decoding performance, and the robustness is maintained until a rate is reached where ∞-SR nodes cannot be avoided. Moreover, we showed that increasing the puncturing period P has the effect of eliminating ∞-SR nodes in high rate punctured codes, in addition to extending the range of compatible rates and increasing the number of intermediate rates.

For future work, we have several open questions to continue with:

- We would like to design good LDPC-CCs as mother codes based on the girth bounds and the distance spectrum estimation investigated in Chapters 3 and 4,

respectively. Also, we would like to apply the puncturing algorithm to these mother codes and compare the decoding performance of the obtained RC LDPC-CCs with the RC LDPC codes used in DVB-2 standard.

- The puncturing algorithm may be extended to QC LDPC-BCs, especially we are interested in unrecoverable punctured nodes in punctured QC LDPC-BCs of finite block length.

- Given a mother code and applying the puncturing algorithm, we wonder what the maximum punctured code rate is that can be obtained without any unrecoverable punctured variable nodes. We have an intuition that the maximum punctured code rate is determined by the weight matrix $\mathbf{B}$ of the polynomial syndrome former matrix $\mathbf{H}^T(D)$.

Appendix A

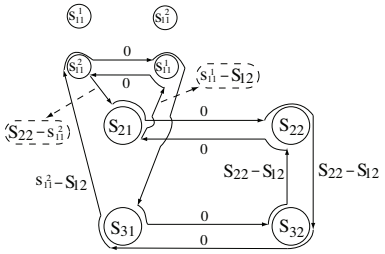
Graphical representations of unavoidable cycles for time-invariant LDPC-CCs

In Chapter 3, we have shown some unavoidable cycles of length up to 12 for time-invariant LDPC-CCs. Together with the weight matrices, here we give the graphical structures of some other unavoidable cycles of lengths 12, 14, 16, 18, and 20 in the polynomial submatrix of a polynomial syndrome former matrix $\mathbf{H}^T(D)$. In each figure, the unavoidable cycle is illustrated on the left side and the corresponding weight matrix is given on the right side. If any entry, let us say b_{ij}, in the weight matrix $\mathbf{B}$ is larger than one, then the corresponding polynomial entry in $\mathbf{H}^T(D)$ can be written as $D^{s_{ij}^1} + D^{s_{ij}^2} \ldots + D^{s_{ij}^{b_{ij}}}$. For the convenience of graphical representation, we arrange the b_{ij} monomials in the form of a "matrix" of size $b_{ij} \times b_{ij}$, i.e.,

$$\begin{bmatrix} D^{s_{ij}^1} & D^{s_{ij}^2} & \cdots & D^{s_{ij}^{b_{ij}}} \\ D^{s_{ij}^2} & D^{s_{ij}^3} & \cdots & D^{s_{ij}^1} \\ \vdots & \vdots & \vdots & \\ D^{s_{ij}^{b_{ij}}} & D^{s_{ij}^1} & \cdots & D^{s_{ij}^{b_{ij}-1}} \end{bmatrix}, \tag{A.1}$$

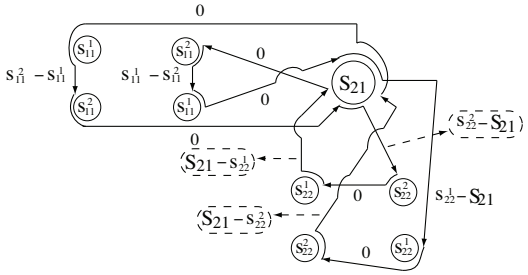
where starting from the second row, each row is the shifted version of the upper one to the left by one monomial entry. For example, in Fig. A.1 the entry b_{11} in $\mathbf{B}_{12}$ is 2, as a result the polynomial entry $D^{s_{11}^1} + D^{s_{11}^2}$ in the graph is replaced by a 2×2 "matrix" formed by $D^{s_{11}^1}$ and $D^{s_{11}^2}$. Note that, for simplicity the notation D is removed in the

graphical representation. Similar explanation applies to Figs. A.2 and A.4. By summing all the delays along each path in Figs. A.1-A.12, they all satisfy the conditions in (3.4) and (3.5). Thus all these cycles are unavoidable no matter what "powers" are chosen for the monomials. Hence when designing LDPC-CCs to achieving the desired girth, the corresponding weight matrix should be free of respective submatrices in (3.17).



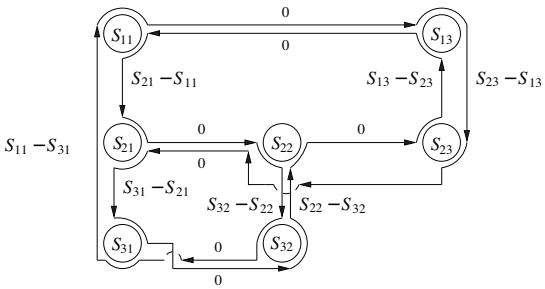
$$\mathbf{B}_{12} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Figure A.1: Unavoidable cycles of length 12 in a 3×2 polynomial submatrix (i)



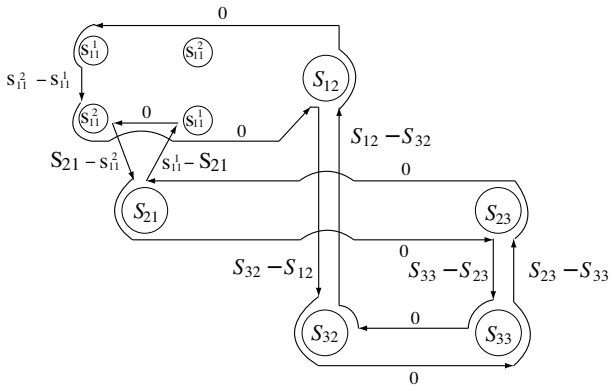
$$\mathbf{B}_{12} = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$$

Figure A.2: Unavoidable cycles of length 12 in a 2×2 polynomial submatrix (ii)



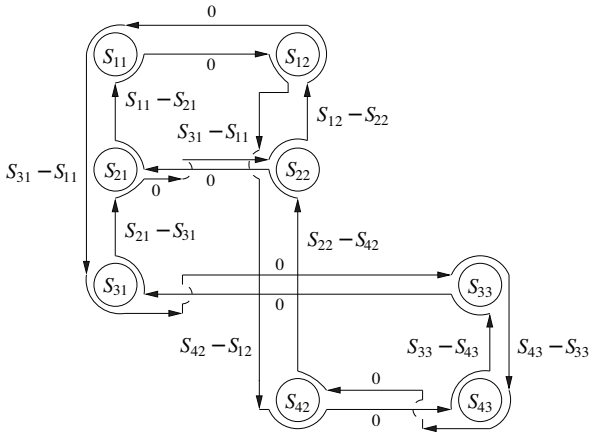
$$\mathbf{B}_{14} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Figure A.3: Unavoidable cycles of length 14 in a 3×3 polynomial submatrix (i)



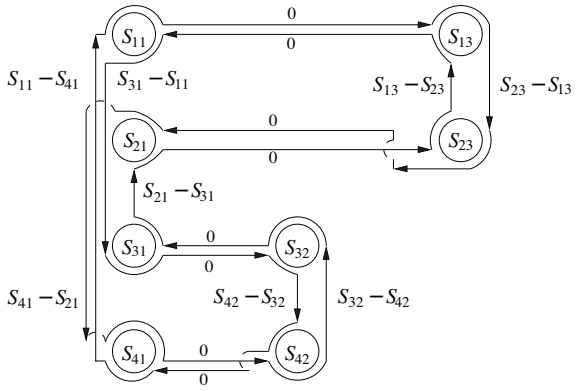
$$\mathbf{B}_{14} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Figure A.4: Unavoidable cycles of length 14 in a 3×3 polynomial submatrix (ii)



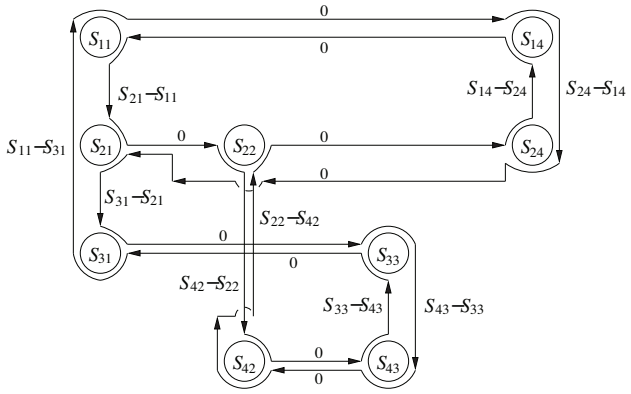
$$\mathbf{B}_{16} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Figure A.5: Unavoidable cycles of length 16 in a 4×3 polynomial submatrix (i)



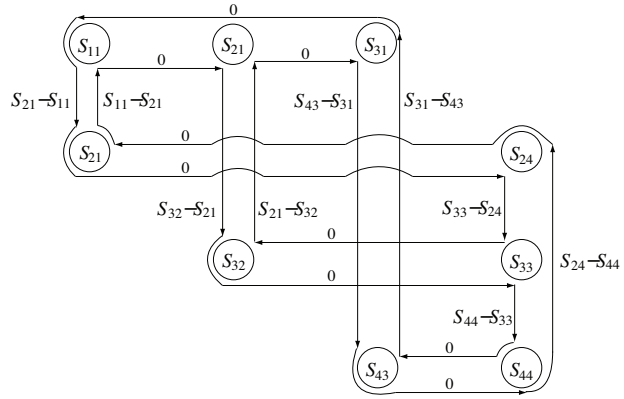
$$\mathbf{B}_{16} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Figure A.6: Unavoidable cycles of length 16 in a 4×3 polynomial submatrix (ii)



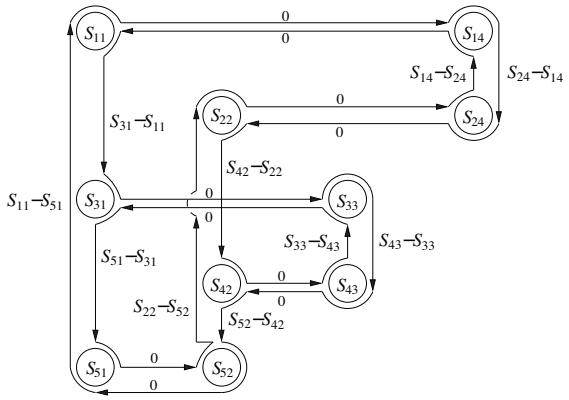
$$\mathbf{B}_{18} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure A.7: Unavoidable cycles of length 18 in a 4×4 polynomial submatrix (i)



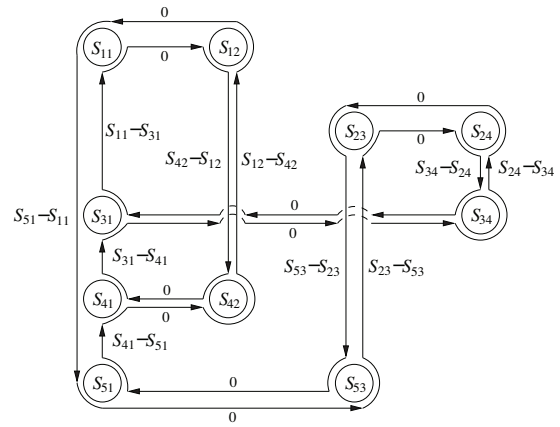
$$\mathbf{B}_{18} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Figure A.8: Unavoidable cycles of length 18 in a 4×4 polynomial submatrix (ii)



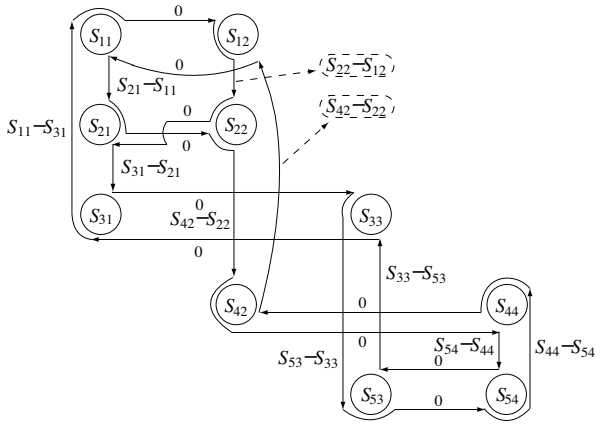
$$\mathbf{B}_{20} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure A.9: Unavoidable cycles of length 20 in a 5×4 polynomial submatrix (i)



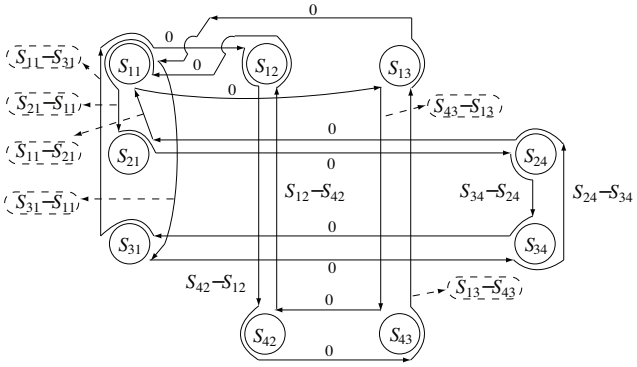
$$\mathbf{B}_{20} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure A.10: Unavoidable cycles of length 20 in a 5×4 polynomial submatrix (ii)



$$\mathbf{B}_{20} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Figure A.11: Unavoidable cycles of length 20 in a 5×4 polynomial submatrix (iii)



$$\mathbf{B}_{20} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure A.12: Unavoidable cycles of length 20 in a 4×4 polynomial submatrix (iv)

Appendix B

RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC

Following the punctured code of rate $R' = 4/8$ with puncturing pattern $\mathbf{a}_{12} = [00100; 00100]$ (details can be found in Section 5.4.2), we give the rest of the family of RC LDPC-CCs derived from the $(21, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 2$.

B.1 The rate $R' = 4/7$ punctured code

Based on the puncturing pattern $\mathbf{a}_{12}$, four RC punctured codes of rate $4/7$ are derived. Their properties are presented in Table B.1. By running Algorithm 3, puncturing pattern $\mathbf{a}_{50}$ is eliminated at Step 1.1 as it contains two ∞-SR nodes. The only difference among patterns $\mathbf{a}_{49}$,[1] $\mathbf{a}_{51}$, and $\mathbf{a}_{52}$ lies in the number of punctured variable nodes involved in short cycles. Thus we chose $\mathbf{a}_{49}$ as the best one since it results in the smallest number of punctured nodes in short cycles. Correspondingly, their decoding performance are shown in Fig. B.1. $\mathbf{a}_{49}$ is the best one with the lowest BER curve, while $\mathbf{a}_{50}$ is the worst one as a result of the two ∞-SR nodes. Even though $\mathbf{a}_{52}$ has larger accumulated value in $E_b^{\mathbf{a}_i}$ than that of $\mathbf{a}_{51}$, $\mathbf{a}_{52}$ contains less punctured nodes in short cycles of length 8 and 10, i.e., $B_8^{\mathbf{a}_{52}} = 20$ and $B_{10}^{\mathbf{a}_{52}} = 156$ which are smaller than $B_8^{\mathbf{a}_{51}} = 24$ and $B_{10}^{\mathbf{a}_{51}} = 167$, respectively. Consequently, $\mathbf{a}_{52}$ has better decoding performance than $\mathbf{a}_{51}$ in the high SNR region.

[1] Note that $\mathbf{a}_{49}$ is the same as the puncturing pattern $\mathbf{a}_{15}$ in Section 5.4.3.

Puncturing pattern of previous lower rate code: [00100; 00100]						
A_2^3		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, and 12		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
49	$*$[10100; 00100]	0	[2 1]	[0 0 0]	[20 155 1146]	1321
50	[01100; 00100]	2	[1]	[0 0 1]	[22 168 1181]	1371
51	[00110; 00100]	0	[2 1]	[0 0 0]	[24 167 1160]	1351
52	[00101; 00100]	0	[2 1]	[0 0 0]	[20 156 1188]	1364

Table B.1: Enumerators for the puncturing patterns of the $(21, 3, 5)$ Tanner LDPC-CC with rate $R' = 4/7$ obtained using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

B.2 The rate $R' = 4/6$ punctured code

Continue the process of obtaining RC punctured code with the puncturing pattern $\mathbf{a}_{49}$, seven RC punctured LDPC-CCs of rate $R' = 4/6$ are obtained by introducing an extra puncture entry in $\mathbf{a}_{49}$. Details of these codes are shown in Table B.2. Regarding all the puncturing patterns in A_2^4, only $\mathbf{a}_{54}$ and $\mathbf{a}_{58}$ are free of ∞-SR nodes,[1] and moreover, $\mathbf{a}_{58}$ is superior to $\mathbf{a}_{54}$ as it has better recoverability of punctured variable nodes at Step 1.2 of Algorithm 3. As a result, we chose $\mathbf{a}_{58}$ as the best candidate puncturing pattern for rate $R' = 4/6$. However, comparing the enumerators between $\mathbf{a}_{54}$ and $\mathbf{a}_{58}$ in Table B.2, they have very similar properties which is consistent with the observation in Fig. B.2 that they have nearly the same decoding performance.

In term of the puncturing patterns in $A_2^4 \backslash \{\mathbf{a}_{54}, \mathbf{a}_{58}\}$, they all have unrecoverable punctured variable nodes. It causes that their decoding performance suffer from very high error floors. In addition, the error floor of $\mathbf{a}_{55}$ and $\mathbf{a}_{56}$ are higher than that of $\mathbf{a}_{53}$, $\mathbf{a}_{57}$, and $\mathbf{a}_{59}$ since puncturing patterns $\mathbf{a}_{55}$, $\mathbf{a}_{56}$ contain twice the number of ∞-SR nodes than those of $\mathbf{a}_{53}$, $\mathbf{a}_{57}$, and $\mathbf{a}_{59}$.

B.3 The rate $R' = 4/5$ punctured code

Finally, we obtain the highest rate $R' = 4/5$ punctured codes with puncturing patterns in the set A_2^5. There are six such codes with enumerators presented in Table B.3. All these codes have ∞-SR nodes. Specifically, the codes with patterns in the set $\{\mathbf{a}_{61}, \mathbf{a}_{62}, \mathbf{a}_{63}\}$ and the set $\{\mathbf{a}_{60}, \mathbf{a}_{64}, \mathbf{a}_{65}\}$ have five and two ∞-SR nodes, respectively.[2]

[1] Note that $\mathbf{a}_{58}$ is the same as the puncturing pattern $\mathbf{a}_{16}$ in Section 5.4.3.
[2] Note that $\mathbf{a}_{64}$ is the same as the puncturing pattern $\mathbf{a}_{17}$ in Section 5.4.3.

Puncturing pattern of previous lower rate code: [10100; 00100]						
A_2^4		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, and 12		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
53	[11100; 00100]	2	[1 1]	[0 0 1]	[30 227 1593]	1850
54	[10110; 00100]	0	[1 2 1]	[0 1 0]	[32 226 1572]	1830
55	[10101; 00100]	4	[0]	[0 0 0]	[28 215 1600]	1843
56	[10100; 10100]	4	[0]	[0 0 2]	[28 214 1558]	1800
57	[10100; 01100]	2	[1 1]	[0 0 3]	[30 227 1593]	1850
58	*[10100; 00110]	0	[2 2]	[1 0 0]	[32 226 1572]	1830
59	[10100; 00101]	2	[2 0 0]	[1 1 2]	[28 215 1600]	1843

Table B.2: Enumerators for the puncturing patterns of the $(21, 3, 5)$ Tanner LDPC-CC with rate $R' = 4/6$ obtained using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

Puncturing pattern of previous lower rate code: [10100; 00110]						
A_2^5		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 8,10, and 12		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
60	[11100; 00110]	2	[1 1 1]	[1 2 6]	[42 298 2019]	2359
61	[10110; 00110]	5	[0]	[3 4 3]	[44 297 1998]	2339
62	[10101; 00110]	5	[0]	[1 0 1]	[40 286 2026]	2352
63	[10100; 10110]	5	[0]	[1 2 3]	[40 285 1984]	2309
64	*[10100; 01110]	2	[1 1 1]	[1 1 6]	[42 298 2019]	2359
65	[10100; 00111]	2	[2 1]	[2 1 9]	[40 286 2026]	2352

Table B.3: Enumerators for the puncturing patterns of the $(21, 3, 5)$ Tanner LDPC-CC with rate $R' = 4/5$ obtained using cycles of length $w = 8$, 10, and 12: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

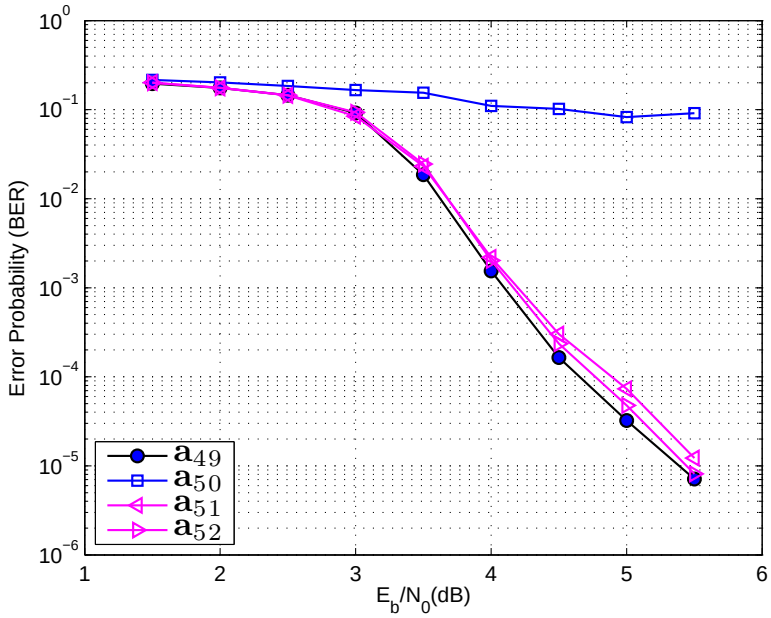


Figure B.1: BERs of the punctured LDPC-CCs of (5.2) with rate R'=4/7.

According the designing criteria, $\mathbf{a}_{64}$ is chosen as the best puncturing pattern for this rate, however, it is not of practical interest due to the existence of error floor. Fig. B.3 presents the decoding performance of these codes. It is observed that $\mathbf{a}_{60}$ and $\mathbf{a}_{64}$ have lower error floors than other puncturing patterns, because they have fewer ∞-SR nodes. $\mathbf{a}_{65}$ contains exactly the same number of ∞-SR nodes as $\mathbf{a}_{60}$ and $\mathbf{a}_{64}$, however, it contains many more CPCTSs. As a result, its error floor is higher than that of $\mathbf{a}_{60}$ and $\mathbf{a}_{64}$ and merges with that of the puncturing patterns in the set $\{\mathbf{a}_{61}, \mathbf{a}_{62}, \mathbf{a}_{63}\}$.

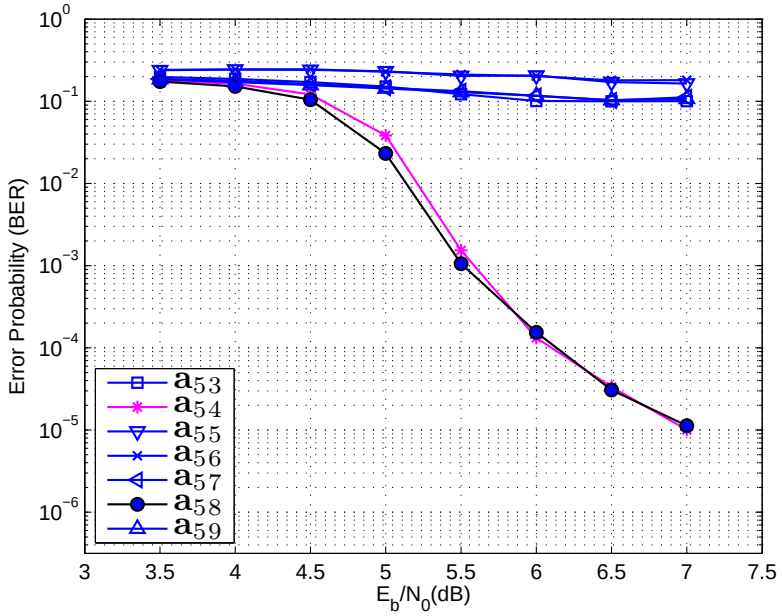


Figure B.2: BERs of the punctured LDPC-CCs of (5.2) with rate R'=4/6.

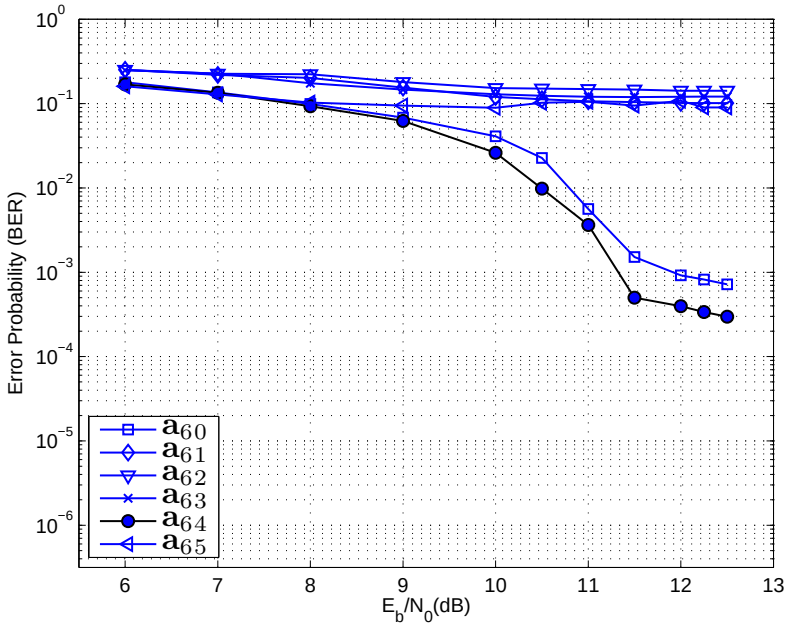


Figure B.3: BERs of the punctured LDPC-CCs of (5.2) with rate R'=4/5.

Appendix C

RC LDPC-CCs derived from the $(57, 3, 5)$ Tanner LDPC-CC

Here we give the process of obtaining the family of RC punctured LDPC-CCs derived from the $(57, 3, 5)$ Tanner LDPC-CC with puncturing period $P = 3$. The polynomial-domain syndrome former matrix of the $(57, 3, 5)$ Tanner LDPC-CC is given by (5.6). It is of rate $R = 2/5$ or $R = 6/15$. The puncturing pattern of the unpunctured mother code is $[00000; 00000; 00000]$.

C.1 The rate $R' = 6/14$ punctured code

We begin by placing $l = 1$ punctured entry in the puncturing pattern of the mother code, there are fifteen options available. However, vertically cyclically shifted puncturing patterns are considered as one type. Hence we obtain five non-equivalent puncturing patterns in the set A_3^1. The enumerators for these patterns are listed in Table C.1. They are all free of ∞-SR nodes and CPCTSs and they have the same properties of recovering punctured nodes. According to Algorithm 3, the pattern selection is determined by the punctured variable nodes in short cycles. By comparing parameter $\boldsymbol{E}_b^{\mathbf{a}_i}$ among puncturing patterns in A_3^1, $\mathbf{a}_{69}$ is selected as the best one for rate $R' = 6/14$,[1] because it contains the smallest number of punctured nodes in shorts cycles though its decoding performance is nearly the same as others as shown in Fig. C.1.

[1] Note that $\mathbf{a}_{69}$ is the same as the puncturing pattern $\mathbf{a}_{35}$ in Section 5.5.

C.2 The rate $R' = 6/13$ punctured code

In order to maintain compatible rates, further punctured codes are sequentially obtained by placing an extra punctured entry in the puncturing pattern of the previous lower rate punctured code. Thirteen punctured codes of rate 6/13 with puncturing patterns in A_3^2 are obtained from $\mathbf{a}_{69}$. Observing the data in Table C.2, punctured codes in A_3^2 have the same enumerators in $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, $\boldsymbol{E}_c^{\mathbf{a}_i}$ and $C_\infty^{\mathbf{a}_i}$. $\mathbf{a}_{78}$ has the smallest number of punctured nodes in short cycles,[1] i.e., $\sum \boldsymbol{E}_b^{\mathbf{a}_{78}} = 2840$. It leads to the better decoding performance over others in Fig. C.2. Consequently, the code with puncturing pattern $\mathbf{a}_{78}$ is chosen as the optimal one for the punctured code rate $R' = 6/13$.

C.3 The punctured codes with rates 6/12, 6/11, 6/10, 6/9, 6/8, and 6/7

Based on Algorithm 3, similar explanations apply to the following RC punctured codes. Here we do not give repetitive comments on the comparisons for each rate. Readers could form the idea by evaluating the properties of codes in Tables C.3-C.8 and simulations in Figs. C.3-C.8, where the puncturing patterns $\mathbf{a}_{69}$, $\mathbf{a}_{78}$, $\mathbf{a}_{95}$, $\mathbf{a}_{99}$, $\mathbf{a}_{105}$, $\mathbf{a}_{112}$, $\mathbf{a}_{123}$, and $\mathbf{a}_{136}$ are the same as the puncturing patterns $\mathbf{a}_{35}$, $\mathbf{a}_{36}$, $\mathbf{a}_{37}$, $\mathbf{a}_{38}$, $\mathbf{a}_{39}$, $\mathbf{a}_{40}$, $\mathbf{a}_{41}$, and $\mathbf{a}_{42}$ in Section 5.5, respectively. We can see that, as the puncturing code rate increases, the enumerators $\boldsymbol{E}_\tau^{\mathbf{a}_i}$ and $\boldsymbol{E}_c^{\mathbf{a}_i}$ of different puncturing patterns of the same rate starts to varying. In other words, for the low rate punctured codes, the selection of puncturing patterns is dominated by enumerator $\boldsymbol{E}_b^{\mathbf{a}_i}$, while for the high rate punctured codes, the enumerators $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, $\boldsymbol{E}_c^{\mathbf{a}_i}$ have more influence on the puncturing algorithm. However, $C_\infty^{\mathbf{a}_i}$ is the most important selection criterion and affect the selection entirely.

[1] Note that $\mathbf{a}_{78}$ is the same as the puncturing pattern $\mathbf{a}_{36}$ in Section 5.5.

Puncturing pattern of previous lower rate code: $[00000; 00000; 00000]$								
A_3^1		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14				
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$			$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
66	$[10000; 00000; 00000]$	0	$[1]$	$[0 \quad 0 \quad 0]$	$[23$	188	$1393]$	1604
67	$[01000; 00000; 00000]$	0	$[1]$	$[0 \quad 0 \quad 0]$	$[23$	177	$1301]$	1501
68	$[00100; 00000; 00000]$	0	$[1]$	$[0 \quad 0 \quad 0]$	$[20$	174	$1298]$	1492
69∗	$[00010; 00000; 00000]$	0	$[1]$	$[0 \quad 0 \quad 0]$	$[18$	161	$1241]$	1420
70	$[00001; 00000; 00000]$	0	$[1]$	$[0 \quad 0 \quad 0]$	$[21$	188	$1396]$	1605

Table C.1: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/14$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

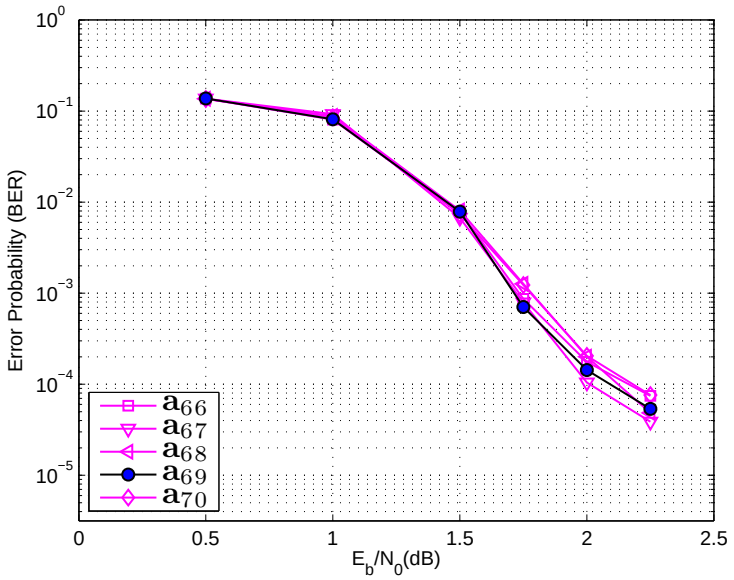


Figure C.1: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/14$.

Puncturing pattern of previous lower rate code: [00010; 00000; 00000]										
A_3^1		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	w = 10,12, and 14						
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$			$\boldsymbol{E}_b^{\mathbf{a}_i}$			$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
71	[10010; 00000; 00000]	0	[2]	[0	0	0]	[41	349	2634]	3024
72	[01010; 00000; 00000]	0	[2]	[0	0	0]	[41	338	2542]	2921
73	[00110; 00000; 00000]	0	[2]	[0	0	0]	[38	335	2539]	2912
74	[00011; 00000; 00000]	0	[2]	[0	0	0]	[39	349	2637]	3025
75	[00010; 10000; 00000]	0	[2]	[0	0	0]	[41	349	2634]	3024
76	[00010; 01000; 00000]	0	[2]	[0	0	0]	[41	338	2542]	2921
77	[00010; 00100; 00000]	0	[2]	[0	0	0]	[38	335	2539]	2912
78∗	[00010; 00010; 00000]	0	[2]	[0	0	0]	[36	322	2482]	2840
79	[00010; 00001; 00000]	0	[2]	[0	0	0]	[39	349	2637]	3025
80	[00010; 00000; 10000]	0	[2]	[0	0	0]	[41	349	2634]	3024
81	[00010; 00000; 01000]	0	[2]	[0	0	0]	[41	338	2542]	2921
82	[00010; 00000; 00100]	0	[2]	[0	0	0]	[38	335	2539]	2912
83	[00010; 00000; 00001]	0	[2]	[0	0	0]	[39	349	2637]	3025

Table C.2: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/13$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

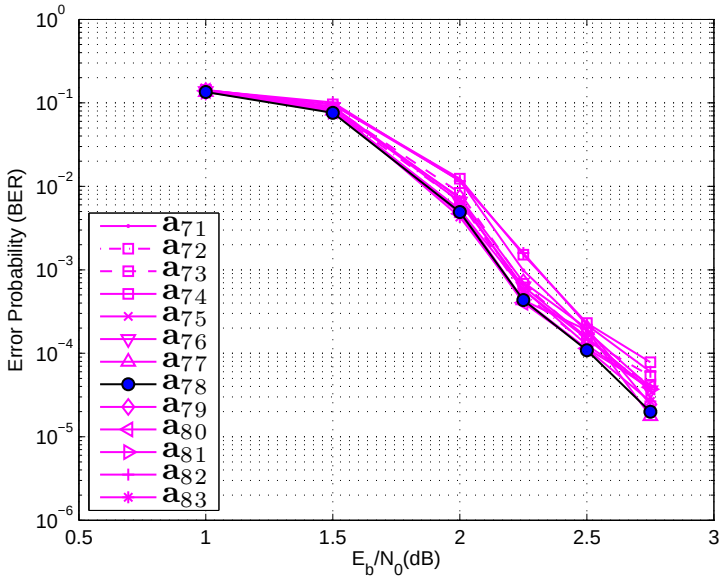


Figure C.2: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/13$.

Puncturing pattern of previous lower rate code: $[00010; 00010; 00000]$						
A_3^3		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
84	$[10010; 00010; 00000]$	0	$[3]$	$[0\ 0\ 0]$	$[59\ 510\ 3875]$	4444
85	$[01010; 00010; 00000]$	0	$[3]$	$[0\ 0\ 0]$	$[59\ 499\ 3783]$	4341
86	$[00110; 00010; 00000]$	0	$[2\ 1]$	$[0\ 0\ 0]$	$[56\ 496\ 3780]$	4332
87	$[00011; 00010; 00000]$	0	$[3]$	$[0\ 0\ 0]$	$[57\ 510\ 3878]$	4445
88	$[00010; 10010; 00000]$	0	$[2\ 1]$	$[0\ 0\ 0]$	$[59\ 510\ 3875]$	4444
89	$[00010; 01010; 00000]$	0	$[3]$	$[0\ 0\ 0]$	$[59\ 499\ 3783]$	4341
90	$[00010; 00110; 00000]$	0	$[3]$	$[0\ 0\ 0]$	$[56\ 496\ 3780]$	4332
91	$[00010; 00011; 00000]$	0	$[2\ 1]$	$[0\ 0\ 0]$	$[57\ 510\ 3878]$	4445
92	$[00010; 00010; 10000]$	0	$[3]$	$[0\ 0\ 0]$	$[59\ 510\ 3875]$	4444
93	$[00010; 00010; 01000]$	0	$[3]$	$[0\ 0\ 0]$	$[59\ 499\ 3783]$	4341
94	$[00010; 00010; 00100]$	0	$[3]$	$[0\ 0\ 0]$	$[56\ 496\ 3780]$	4332
95	$*[00010; 00010; 00010]$	0	$[3]$	$[0\ 0\ 0]$	$[54\ 483\ 3723]$	4260
96	$[00010; 00010; 00001]$	0	$[3]$	$[0\ 0\ 0]$	$[57\ 510\ 3878]$	4445

Table C.3: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/12$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

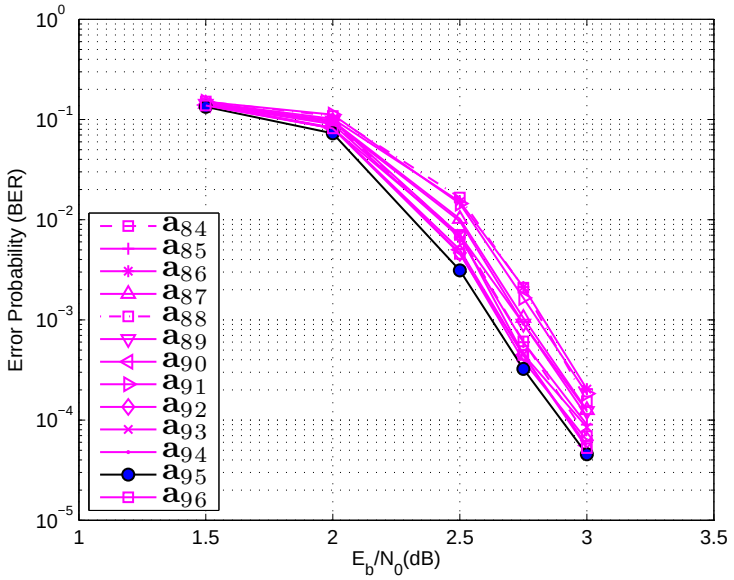


Figure C.3: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/12$.

Puncturing pattern of previous lower rate code: $[00010; 00010; 00010]$						
A_3^4		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
97	$[10010; 00010; 00010]$	0	$[3 \quad 1]$	$[0 \quad 0 \quad 0]$	$[77 \quad 671 \quad 5116]$	5864
98	$[01010; 00010; 00010]$	0	$[3 \quad 1]$	$[0 \quad 0 \quad 0]$	$[77 \quad 660 \quad 5024]$	5761
99	$*[00110; 00010; 00010]$	0	$[3 \quad 1]$	$[0 \quad 0 \quad 0]$	$[74 \quad 657 \quad 5021]$	5752
100	$[00011; 00010; 00010]$	0	$[3 \quad 1]$	$[0 \quad 0 \quad 0]$	$[75 \quad 671 \quad 5119]$	5865

Table C.4: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/11$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

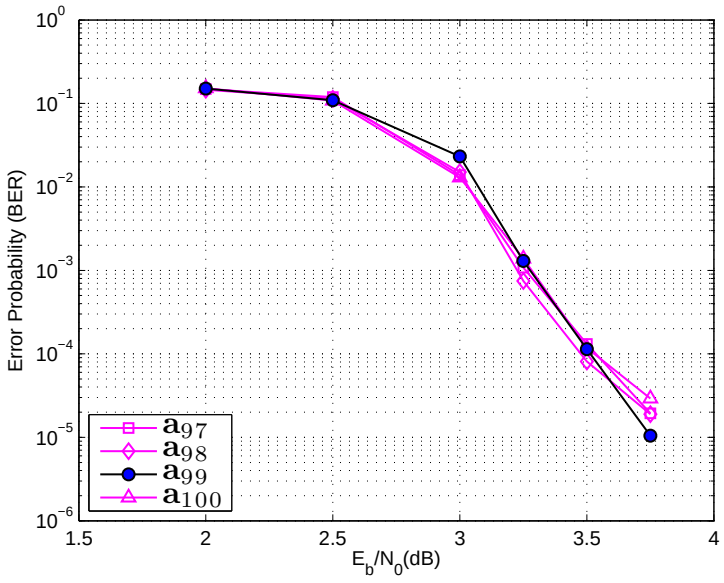


Figure C.4: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/11$.

Puncturing pattern of previous lower rate code: $[00110; 00010; 00010]$						
A_3^5		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
101	$[10110; 00010; 00010]$	0	$[3\ \ 2]$	$[0\ \ 0\ \ 0]$	$[97\ \ 845\ \ 6414]$	7356
102	$[01110; 00010; 00010]$	0	$[2\ \ 2\ \ 1]$	$[0\ \ 0\ \ 0]$	$[97\ \ 834\ \ 6322]$	7253
103	$[00111; 00010; 00010]$	0	$[2\ \ 2\ \ 1]$	$[0\ \ 0\ \ 0]$	$[95\ \ 845\ \ 6417]$	7357
104	$[00110; 10010; 00010]$	0	$[2\ \ 2\ \ 1]$	$[0\ \ 0\ \ 1]$	$[97\ \ 845\ \ 6414]$	7356
105	$*[00110; 01010; 00010]$	0	$[3\ \ 2]$	$[0\ \ 0\ \ 0]$	$[97\ \ 834\ \ 6322]$	7253
106	$[00110; 00110; 00010]$	0	$[2\ \ 2\ \ 1]$	$[0\ \ 1\ \ 0]$	$[94\ \ 831\ \ 6319]$	7244
107	$[00110; 00011; 00010]$	2	$[3]$	$[1\ \ 3\ \ 6]$	$[95\ \ 845\ \ 6417]$	7357
108	$[00110; 00010; 10010]$	0	$[3\ \ 2]$	$[0\ \ 0\ \ 0]$	$[97\ \ 845\ \ 6414]$	7356
109	$[00110; 00010; 01010]$	0	$[3\ \ 2]$	$[0\ \ 0\ \ 1]$	$[97\ \ 834\ \ 6322]$	7253
110	$[00110; 00010; 00011]$	0	$[2\ \ 2\ \ 1]$	$[0\ \ 0\ \ 0]$	$[95\ \ 845\ \ 6417]$	7357

Table C.5: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/10$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

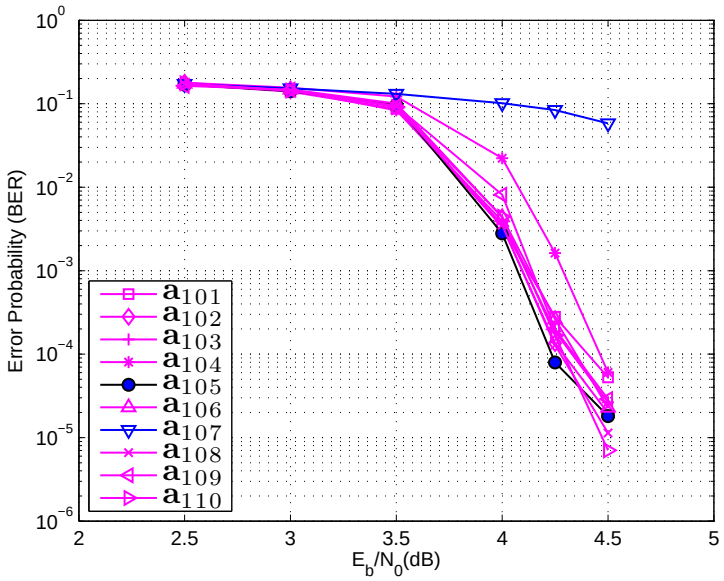


Figure C.5: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/10$.

Puncturing pattern of previous lower rate code: $[00110; 01010; 00010]$							
A_3^6		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14			
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$		$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
111	$[10110; 01010; 00010]$	0	$[3\ \ 3]$	$[0\ \ 2\ \ 1]$	$[120\ \ 1022\ \ 7715]$		8857
112	$*[01110; 01010; 00010]$	0	$[1\ \ 2\ \ 2\ \ 1]$	$[0\ \ 0\ \ 0]$	$[120\ \ 1011\ \ 7623]$		8754
113	$[00111; 01010; 00010]$	0	$[2\ \ 1\ \ 2\ \ 1]$	$[0\ \ 0\ \ 1]$	$[118\ \ 1022\ \ 7718]$		8858
114	$[00110; 11010; 00010]$	0	$[1\ \ 1\ \ 1\ \ 2\ \ 1]$	$[0\ \ 0\ \ 2]$	$[120\ \ 1022\ \ 7715]$		8857
115	$[00110; 01110; 00010]$	0	$[1\ \ 2\ \ 2\ \ 1]$	$[0\ \ 1\ \ 1]$	$[117\ \ 1008\ \ 7620]$		8745
116	$[00110; 01011; 00010]$	2	$[3\ \ 1]$	$[1\ \ 3\ \ 6]$	$[118\ \ 1022\ \ 7718]$		8858
117	$[00110;01010;10010]$	0	$[3\ \ 3]$	$[0\ \ 0\ \ 1]$	$[120\ \ 1022\ \ 7715]$		8857
118	$[00110; 01010; 01010]$	0	$[3\ \ 2\ \ 1]$	$[0\ \ 0\ \ 2]$	$[120\ \ 1011\ \ 7623]$		8754
119	$[00110; 01010; 00110]$	0	$[2\ \ 1\ \ 2\ \ 1]$	$[0\ \ 2\ \ 2]$	$[117\ \ 1008\ \ 7620]$		8745
120	$[00110; 01010; 00011]$	0	$[1\ \ 2\ \ 2\ \ 1]$	$[0\ \ 0\ \ 0]$	$[118\ \ 1022\ \ 7718]$		8858

Table C.6: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/9$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

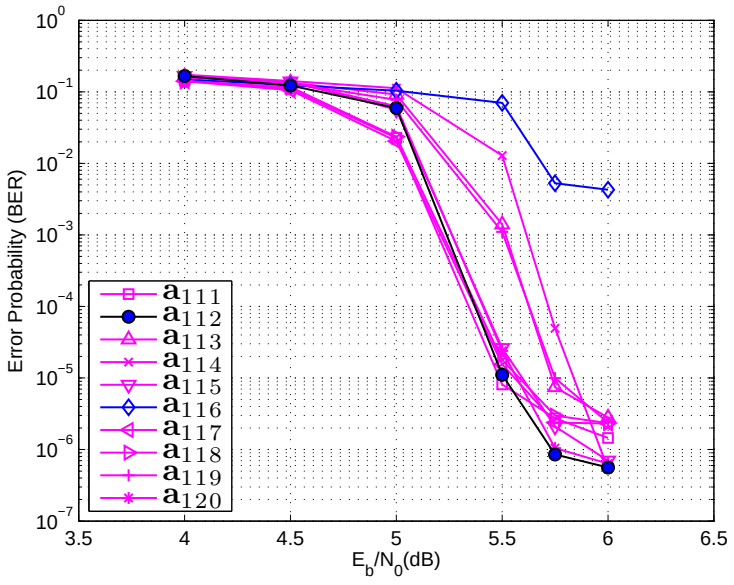


Figure C.6: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/9$.

Puncturing pattern of previous lower rate code: $[01110; 01010; 00010]$						
A_3^7		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10$,12, and 14		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum_w \boldsymbol{E}_b^{\mathbf{a}_i}$
121	$[11110; 01010; 00010]$	7	$[0]$	$[0\ 2\ 1]$	$[143\ 1199\ 9016]$	10358
122	$*[01111; 01010; 00010]$	0	$[1\ 1\ 1\ 1\ 2\ 1]$	$[0\ 0\ 2]$	$[141\ 1199\ 9019]$	10359
123	$[01110; 11010; 00010]$	0	$[1\ 2\ 1\ 2\ 1]$	$[0\ 0\ 5]$	$[143\ 1199\ 9016]$	10358
124	$[01110; 01110; 00010]$	7	$[0]$	$[1\ 3\ 6]$	$[140\ 1185\ 8921]$	10246
125	$[01110; 01011; 00010]$	2	$[1\ 2\ 2]$	$[1\ 3\ 7]$	$[141\ 1199\ 9019]$	10359
126	$[01110; 01010; 10010]$	6	$[1]$	$[1\ 0\ 1]$	$[143\ 1199\ 9016]$	10358
127	$[01110; 01010; 01010]$	7	$[0]$	$[0\ 4\ 2]$	$[143\ 1188\ 8924]$	10255
128	$[01110; 01010; 00110]$	0	$[1\ 1\ 1\ 1\ 2\ 1]$	$[0\ 2\ 11]$	$[140\ 1185\ 8921]$	10246
129	$[01110; 01010; 00011]$	7	$[0]$	$[0\ 0\ 1]$	$[141\ 1199\ 9019]$	10359

Table C.7: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/8$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

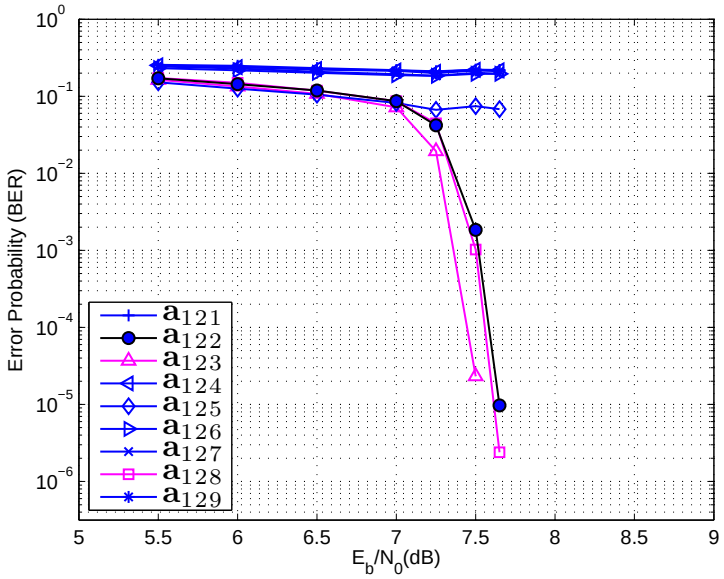


Figure C.7: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/8$.

Puncturing pattern of previous lower rate code: $[01111; 01010; 00010]$						
A_3^8		$C_\infty^{\mathbf{a}_i}$	$\boldsymbol{E}_c^{\mathbf{a}_i}$	$w = 10, 12, 14$		
i	$\mathbf{a}_i$			$\boldsymbol{E}_\tau^{\mathbf{a}_i}$	$\boldsymbol{E}_b^{\mathbf{a}_i}$	$\sum \boldsymbol{E}_b^{\mathbf{a}_i}$
130	$[11111; 01010; 00010]$	8	$[0]$	$[0\ 2\ 11]$	$[164\ 1387\ 10412]$	11963
131	$[01111; 11010; 00010]$	6	$[1\ 1]$	$[0\ 1\ 13]$	$[164\ 1387\ 10412]$	11963
132	$[01111; 01110; 00010]$	8	$[0]$	$[1\ 5\ 8]$	$[161\ 1373\ 10317]$	11851
133	$[01111; 01011; 00010]$	5	$[1\ 1\ 1]$	$[2\ 7\ 26]$	$[162\ 1387\ 10415]$	11964
134	$[01111; 01010; 10010]$	7	$[1]$	$[3\ 1\ 10]$	$[164\ 1387\ 10412]$	11963
135	$[01111; 01010; 01010]$	8	$[0]$	$[0\ 4\ 8]$	$[164\ 1376\ 10320]$	11860
136	$*[01111; 01010; 00110]$	2	$[1\ 1\ 1\ 1\ 2]$	$[2\ 7\ 36]$	$[161\ 1373\ 10317]$	11851
137	$[01111; 01010; 00011]$	8	$[0]$	$[0\ 4\ 11]$	$[162\ 1387\ 10415]$	11964

Table C.8: Enumerators for the puncturing patterns of the $(57, 3, 5)$ Tanner LDPC-CC with rate $R' = 6/7$ obtained using cycles of length $w = 10$, 12, and 14: the enumerator of ∞-SR nodes $C_\infty^{\mathbf{a}_i}$, the enumerator of m-SR nodes $\boldsymbol{E}_c^{\mathbf{a}_i}$, the enumerator of CPCTSs $\boldsymbol{E}_\tau^{\mathbf{a}_i}$, and the enumerator of punctured variable nodes involved in short cycles $\boldsymbol{E}_b^{\mathbf{a}_i}$.

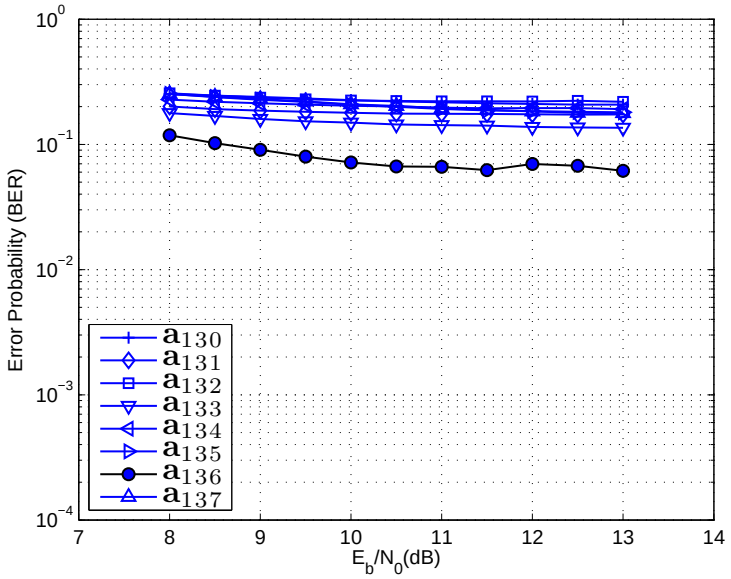


Figure C.8: BERs of the punctured LDPC-CCs of (5.6) with rate $R' = 6/7$.

References

[1] A. J. Felström and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999. iv, 4, 7, 15, 22, 28, 54, 69, 73, 81

[2] D. J. Costello, Jr., A. E. Pusane, S. Bates, and K. S. Zigangirov, "A comparison between LDPC block and convolutional codes," in *Proc. IEEE Information Theory and Applications Workshop*, San Diego, USA, Jan. 2006. iv, 4, 28, 54, 69

[3] S. Bates, D. G. Elliot, and R. Swamy, "Termination sequence generation circuits for low-density parity-check convolutional codes," *IEEE Transactions on Circuits Systems*, vol. 53, no. 9, pp. 1909–1917, Sep. 2006. iv, 4, 28, 54

[4] S. Lin and D. J. Costello, Jr., *Error control coding*, 2nd ed. Pearson Prentice Hall, 2004. ix, 2, 9

[5] 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*, vol. 50, no. 12, pp. 2966 – 2984, Dec. 2004. x, 9, 10, 11, 39, 41, 50, 51, 54, 57, 62, 63, 65, 66, 70, 71

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *IEEE International Conference on Communications*, vol. 2, May 1993, pp. 1064 –1070. 3

[7] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. IT-8, pp. 21–28, Jan. 1962. 3

[8] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K. S. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 5274 –5289, Oct. 2010. 4, 69

[9] D. G. M. Mitchell, A. E. Pusane, and D. J. Costello, Jr., "Minimum distance and trapping set analysis of protograph-based LDPC convolutional codes," *IEEE Transactions on Information Theory*, vol. PP, no. 99, pp. 1–27, Aug. 2012. 4, 55

[10] R. Smarandache, A. Pusane, P. Vontobel, and D. Costello, "Pseudo-codewords in LDPC convolutional codes," in *IEEE International Symposium on Information Theory*, Jul. 2006, pp. 1364 –1368. 4

[11] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389 –400, Apr. 1988. 4, 69, 75

[12] S. J. Johnson, "Introducing low-density parity-check codes." 4, 19, 72

[13] I. E. Bocharova, M. Handlery, and R. Johannesson, "A BEAST for prowling in trees," *IEEE Transactions on Information Theory*, vol. 50, no. 6, pp. 1295–1302, Jun. 2004. 5, 54, 67, 69, 96

[14] H. Zhou and N. Goertz, "Cycle analysis of time-variant LDPC convolutional codes," in *Proc. IEEE 6th International Symposium on Turbo Codes and Iterative Information Processing*, Brest, France, Sep. 2010. 7, 28, 38, 39

[15] A. Sridharan, "Design and analysis of LDPC convolutional codes," *Phd Dissertation, University of Notre Dame*, 2005. 15, 41, 51

[16] A. E. Pusane, A. J. Felström, A. Sridharan, M. Lentmaier, K. Zigangirov, and D. J. Costello, Jr., "Implementation aspects of LDPC convolutional codes," *IEEE Transactions on Communications*, vol. 56, no. 7, pp. 1060 –1069, Jul. 2008. 15

[17] S. Kim, G. Sobelman, and H. Lee, "A reduced-complexity architecture for LDPC layered decoding schemes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 1099 –1103, Jun. 2011. 19

[18] J. Ha, D. Klinc, J. Kwon, and S. W. McLaughlin, "Layered bp decoding for rate-compatible punctured LDPC codes," *IEEE Communications Letters*, vol. 11, no. 5, pp. 440 –442, May 2007. 19, 69

[19] A. I. Vila Casado, M. Griot, and R. D. Wesel, "LDPC decoders with informed dynamic scheduling," *IEEE Transactions on Communications*, vol. 58, no. 12, pp. 3470 –3479, Dec. 2010. 19

[20] A. I. V. Casado, M. Griot, and R. D. Wesel, "Improving LDPC decoders via informed dynamic scheduling," in *IEEE Information Theory Workshop*, Sep. 2007, pp. 208 –213. 19

[21] A. I. Vila Casado, M. Griot, and R. D. Wesel, "Informed dynamic scheduling for belief-propagation decoding of LDPC codes," in *IEEE International Conference on Communications*, Jun. 2007, pp. 932 –937. 19

[22] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004. 27

[23] O. Milenkovic, N. Kashyap, and D. Leyba, "Shortened array codes of large girth," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3707 –3722, Aug. 2006. 27

[24] M. O'Sullivan, "Algebraic construction of sparse matrices with large girth," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 718 –727, Feb. 2006. 27

[25] S. Kim, J.-S. No, H. Chung, and D.-J. Shin, "Quasi-cyclic low-density parity-check codes with girth larger than 12," *IEEE Transactions on Information Theory*, vol. 53, no. 8, pp. 2885 –2891, Aug. 2007. 27, 46

[26] J. Huang, L. Liu, W. Zhou, and S. Zhou, "Large-girth nonbinary QC-LDPC codes of various lengths," *IEEE Transactions on Communications*, vol. 58, no. 12, pp. 3436 –3447, Dec. 2010. 27

[27] I. Bocharova, F. Hug, R. Johannesson, B. Kudryashov, and R. Satyukov, "Searching for voltage graph-based LDPC tailbiting codes with large girth," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2265 –2279, Apr. 2012. 27

[28] M. Esmaeili and M. Gholami, "Geometrically-structured maximum-girth LDPC block and convolutional codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 831 –845, Aug. 2009. 27

[29] B. Vasic and O. Milenkovic, "Combinatorial constructions of low-density parity-check codes for iterative decoding," *IEEE Transactions on Information Theory*, vol. 50, no. 6, pp. 1156 – 1176, june 2004. 27

[30] B. Ammar, B. Honary, Y. Kou, J. Xu, and S. Lin, "Construction of low-density parity-check codes based on balanced incomplete block designs," *IEEE Transactions on Information Theory*, vol. 50, no. 6, pp. 1257 – 1269, Jue 2004. 27

[31] M. Karimi and A. Banihashemi, "Counting short cycles of quasi cyclic protograph LDPC codes," *IEEE Communications Letters*, vol. 16, no. 3, pp. 400 –403, Mar. 2012. 27

[32] ——, "Message-passing algorithms for counting short cycles in a graph," *IEEE Transactions on Communications*, vol. PP, no. 99, pp. 1 –11, 2012. 28

[33] H. Zhou and N. Goertz, "Cycle analysis of time-invariant LDPC convolutional codes," in *Telecommunications (ICT), 2010 IEEE 17th International Conference on*, Apr. 2010, pp. 23 –28. 28, 39, 45

[34] ——, "Unavoidable cycles in polynomial-based time-invariant LDPC convolutional codes," *11th European Wireless*, pp. 1 –6, Apr. 2011. 28, 46

[35] ——, "Girth analysis of polynomial-based time-invariant LDPC convolutional codes," in *19th International Conference on Systems Signals and Image Processing (IWSSIP)*, Apr. 2012, pp. 104 –108. 28, 46

[36] H. Park, S. Hong, J. S. No, and D. J. Shin, "Design of multiple-edge protographs for QC LDPC codes avoiding short inevitable cycles," *submitted to IEEE Transactions on Information Theory*, 2011. 46

[37] A. E. Pusane, M. Lentmaier, K. S. Zigangirov, and D. J. Costello, Jr., "Reduced complexity decoding strategies for LDPC convolutional codes," in *International Symposium on Information Theory*, Jun. 2004, p. 490. 51, 81

[38] L. R. Bahl, C. D. Cullum, W. D. Frazer, and F. Jelinek, "An efficient algorithm for computing free distance," *IEEE Transactions on Information Theory*, vol. IT-18, pp. 437–439, May 1972. 54

[39] M. Rouanne and D. J. Costello, Jr., "An algorithm for computing the distance spectrum of trellis codes," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 6, pp. 929–940, Aug. 1989. 54

[40] A. Sridharan, D. Truhachev, M. Lentmaier, D. Costello, and K. Zigangirov, "Distance bounds for an ensemble of LDPC convolutional codes," *IEEE Transactions on Information Theory Transactions on*, vol. 53, no. 12, pp. 4537 –4555, Dec. 2007. 55

[41] D. Truhachev, K. Zigangirov, and D. Costello, "Distance bounds for periodically time-varying and tail-biting LDPC convolutional codes," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4301 –4308, Sep. 2010. 55

[42] H. Zhou, D. Mitchell, N. Goertz, and D. J. Costello, Jr., "Distance spectrum estimation of LDPC convolutional codes," in *Proceedings IEEE International Symposium on Information Theory*, Jul. 2012, pp. 473 –477. 55

[43] M. Hirotomo and M. Morii, "Improvement of the method for computing the weight spectrum of LDPC convolutional codes," *IEICE Technical Report*, vol. 111, no. 220, pp. 15–22, Sep. 20011. 63

[44] D. J. C. MacKay and M. C. Davey, "Evaluation of gallager codes for short block length and high rate applications," *IMA volumes in Mathematics and its Applications*, vol. 123, no. 5, pp. 113–130, 2001. 65

[45] D. G. M. Mitchell, R. Smarandache, and D. J. Costello, Jr., "Quasi-cyclic LDPC codes based on pre-lifted protographs," in *Proc. IEEE Information Theory Workshop*, Paraty, Brazil, Oct. 2011. 66

[46] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system," *IEEE Transactions on Information Theory*, vol. 24, pp. 235–265, 1997. 66

[47] T. Tian, C. Jones, and J. Villasenor, "Rate-compatible low-density parity-check codes," in *Proceedings International Symposium on Information Theory*, Jun. 2004, p. 152. 68

[48] H. Saeedi, H. Pishro-Nik, and A. H. Banihashemi, "Successive maximization for systematic design of universally capacity approaching rate-compatible sequences of LDPC code ensembles over binary-input output-symmetric memoryless channels," *IEEE Transactions on Communications*, vol. 59, no. 7, pp. 1807 –1819, Jul. 2011. 68, 69

[49] M. R. Yazdani and A. H. Banihashemi, "On construction of rate-compatible low-density parity-check codes," *IEEE Communications Letters*, vol. 8, no. 3, pp. 159 – 161, Mar. 2004. 68

[50] Z. Si, M. Andersson, R. Thobaben, and M. Skoglund, "Rate-compatible LDPC convolutional codes for capacity-approaching hybrid ARQ," in *IEEE Information Theory Workshop*, Oct. 2011, pp. 513 –517. 68

[51] J. Ha, J. Kim, and S. McLaughlin, "Rate-compatible puncturing of low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2824 – 2836, Nov. 2004. 68

[52] J. Ha, J. Kim, D. Klinc, and S. W. McLaughlin, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 728 –738, Feb. 2006. 68, 70, 73, 74

[53] J. Kim, A. Ramamoorthy, and S. W. Mclaughlin, "The design of efficiently-encodable rate-compatible LDPC codes," *IEEE Transactions on Communications*, vol. 57, no. 2, pp. 365 –375, Feb. 2009. 68

[54] H. Y. Park, J. W. Kang, K. S. Kim, and K. C. Whang, "Efficient puncturing method for rate-compatible low-density parity-check codes," *IEEE Transactions on Wireless Communications*, vol. 6, no. 11, pp. 3914 –3919, Nov. 2007. 68

[55] B. Vellambi and F. Fekri, "Finite-length rate-compatible LDPC codes: a novel puncturing scheme," *IEEE Transactions on Communications*, vol. 57, no. 2, pp. 297 –301, Feb. 2009. 68

[56] M. El-Khamy, J. Hou, and N. Bhushan, "Design of rate-compatible structured LDPC codes for hybrid ARQ applications," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 965 –973, Aug. 2009. 68

[57] A. I. V. Casado, W. Weng, S. Valle, and R. Wesel, "Multiple-rate low-density parity-check codes with constant blocklength," *IEEE Transactions on Communications*, vol. 57, no. 1, pp. 75 –83, Jan. 2009. 68

[58] H. Lou and J. Garcia-Frias, "Rate-compatible low-density generator matrix codes," *IEEE Transactions on Communications*, vol. 56, no. 3, pp. 321 –324, Mar. 2008. 68

[59] S. Kudekar, T. Richardson, and R. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," in *Proceedings IEEE International Symposium on Information Theory*, Jul. 2012, pp. 453 –457. 69

[60] T. Richardson, "Error-floors of LDPC codes," *Proc. IEEE 41th Annual Allerton Conference*, Sep. 2003. 69, 71

[61] T. Nguyen, A. Nosratinia, and D. Divsalar, "The design of rate-compatible protograph LDPC codes," *IEEE Transactions on Communications*, vol. 60, no. 10, pp. 2841 –2850, Oct. 2012. 69

[62] Z. Si, R. Thobaben, and M. Skoglund, "Rate-compatible LDPC convolutional codes achieving the capacity of the BEC," *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 4021 –4029, Jun. 2012. 69

[63] H. Zhou, D. G. M. Mitchell, N. Goertz, and J. Costello, D.J., "A puncturing algorithm for rate-compatible LDPC convolutional codes," in *International Symposium onTurbo Codes and Iterative Information Processing (ISTC)*, Aug. 2012, pp. 255 –259. 70

[64] ——, "Robust rate-compatible punctured LDPC convolutional codes," *submitted to IEEE Transactions on Communications*. 70

[65] M. Ivkovic, S. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using tanner graph covers," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3763 –3768, Aug. 2008. 72

[66] J. Chen and M. Fossorier, "Density evolution for two improved BP-based decoding algorithms of LDPC codes," *IEEE Communications Letters*, vol. 6, no. 5, pp. 208 –210, May. 2002. 73

[67] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533 – 547, Sep. 1981. 73

[68] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008. 76

[69] R. Smarandache and P. O. Vontobel, "Quasi-Cyclic LDPC codes: Influence of proto- and Tanner-graph structure on minimum Hamming distance upper bounds," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 585 –607, Feb. 2012. 92