Design of Finite-Length Generalized LDPC Codes

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Electrical Engineering)

by

Tingjun Xie

May 2013

APPROVAL SHEET

The dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

AUTHOR

The dissertation has been read and approved by the examining committee:

Stephen G. Wilson

Advisor

Maite Brandt-Pearce

Toby Berger

Randy Cogill

Zongli Lin

Accepted for the School of Engineering and Applied Science:

James H. Ay

Dean, School of Engineering and Applied Science

May
2013

Dedicated to my parents and my wife, for their boundless love, support and encouragement

Acknowledgments

I would like to express my deepest gratitude and esteem to my advisor, Professor Stephen G. Wilson. I am very grateful for his guidance, support and understanding throughout my study and research in the University of Virginia. I am unable to describe how much in these years I have benefited and learned from Professor Wilson's broad knowledge, engineer's intuition, lucid formulation of problems, insights amid research complications, and conscientious dedication to work. I am truly fortunate to have him as my advisor in my Ph.D study; this work would be impossible without his sustaining support and encouragement.

I want to thank my committee members: Professor Toby Berger, Professor Randy Cogill, Professor Zongli Lin, Professor Maïté Brandt-Pearce and Professor Stephen G. Wilson. Their academic insights and technical discussions on the research were important for me to finish this work. Their suggestions and comments significantly improve the quality of the dissertation. And special thanks to Professor Maïté Brandt-Pearce for the financial support at times during the research.

The labmates during my PhD study have left my life with a pleasant memory. Finally, I want to give my regards to Oguz Dogan, Jakob Harmon, Ryan Hinton, Mohammad Noshad, John Peng, Muhammad Qureshi, Taylor Robertson, Anup Shrinivasan, Houbing Song, Yi Tang, Xu Wang, Jackie Xing and Chenguang Xu (surnames listed alphabetically), for their discussions, suggestions, weekly lab votes and brain teasers.

Abstract

Finding coding schemes that approach the Shannon capacity limit has been the central theme during the development of error-control coding (ECC). In recent years low-density parity-check (LDPC) codes, due to their remarkable performance under iterative decoding, have received extensive study and have been adopted by many modern advanced communication systems.

LDPC codes can be further extended to generalized LDPC (GLDPC) codes and doubly-generalized LDPC (DGLDPC) codes. GLDPC and DGLDPC codes are supersets of LDPC codes that possess more generic code structures. GLDPC and DGLDPC codes are more flexible than LDPC codes, since a richer variety of check nodes (CNs) or variable nodes (VNs) offers additional options and flexibility, which enable many advantages over LDPC codes in numerous scenarios.

In this dissertation we focus on improved design methods for finite-length (FL) GLDPC and DGLDPC codes. The first research topic is the error floor behavior of FL-GLDPC and FL-DGLDPC codes. In GLDPC and DGLDPC codes, the existence of generalized CNs and VNs requires more careful management of edge connections for achieving low error floors. In this work the approximate cycle extrinsic message degree (ACE), a cycle parameter used in LDPC codes that reflects the resilience of the cycle to the noise, is generalized to extended ACE (eACE) for GLDPC and DGLDPC codes. By integrating eACE into the progressive-edge-growth (PEG) algorithm and adding more constraints on the connectivity of nodes, the error floor performance of GLDPC and DGLDPC codes can be greatly improved.

In practice, designing codes capable of fast encoding is an important issue. Efficiently-encodable (EE) GLDPC and DGLDPC codes are studied and a self-consistent unified framework is proposed for constructing EE iteratively-decodable block codes, including EE-LDPC, EE-GLDPC and EE-DGLDPC codes. The competing nature of the fast encoding speed and the low error floor performance is analyzed and verified via various simulation results.

EE codes are systematic and suitable for puncturing; therefore, they are good candidates for rate-compatible (RC) data transmissions. Based on the characteristics of GLDPC and DGLDPC codes, we devise several systematic puncturing algorithms to provide good puncturing patterns for them. It is shown that we can significantly improve the code performance compared with random puncturing. Over a wide range of code rates, RC EE-GLDPC and EE-DGLDPC codes show advantages over EE-LDPC codes, especially at high signal-to-noise ratios (SNRs).

Contents

List of Figures

List of Tables

List of Abbreviations

ACE	approximate cycle extrinsic message degree
APP	*a-posteriori* probability
AWGN	additive white Gaussian noise
BEC	binary erasure channel
BER	bit error rate
BIAWGN	binary-input additive white Gaussian noise
BP	belief propagation
BPSK	binary phase-shift keying
CC	convolutional code
CN	check node
DE	differential evolution
DGLDPC	doubly-generalized Low-Density Parity-Check
eACE	extended approximate cycle extrinsic message degree
ECC	error control coding
EE	efficiently-encodable
EERC	efficiently-encodable rate-compatible
EMD	extrinsic message degree
EXIT	extrinsic information transfer
FER	frame error rate
FL	finite length

GLDPC	generalized Low-Density Parity-Check
hGLDPC	hybrid GLDPC
i.i.d.	independent, identically distributed
IRA	irregular repeat-accumulate
L-**SR**	L-step recoverable
LDPC	Low-Density Parity-Check
LLR	log likelihood ratio
MAP	maximum-*a-posteriori*
ML	maximum likelihood
PDF	probability density function
PEG	progressive edge growth
REP	repetition
RC	rate compatible
RV	random variable
SCN	super check node
sGLDPC	strict-sense GLDPC
SNR	signal-to-noise ratio
SOMAP	soft-output maximum-*a-posteriori*
SOML	soft-output maximum likelihood
SPA	sum-product algorithm
SPC	single parity-check

SVN	super variable node
TS	trapping set
VN	variable node

Chapter 1

Introduction

In this chapter, we first give a brief introduction to digital communication and the history of channel coding techniques. After reviewing the existing research literature, the motivations of this work are presented. The outline of the dissertation is summarized at the end of this chapter.

1.1 Digital communication and channel coding

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point." [1]. The classical end-to-end communication system model involves three essential constituent units: source, channel and sink, where source produces information (data, speech, video, image, etc.), and channel can be referred to as any medium (coaxial cable, optical link, wireless, storage system, etc.), which carries and transports information; during the transmission the information is also corrupted by the noise. The sink receives information and tries to reconstruct the transmitted information with as little distortion as possible.

In his landmark paper [1], Claude E. Shannon discovered an important parameter for the noisy channel: channel capacity C, and proved the channel capacity theorem, that as long as the transmitted information rate R (in bits per channel use) is below C ($R < C$), then reliable communication is achievable. Conversely if $R > C$ it is impossible to achieve error-free transmission. Shannon also demonstrated that when $R < C$, one has to properly encode the information (or in other words, add memory and redundancies), in order to attain error-free transmission.

The channel capacity theorem, together with source coding theorem [1], constitutes the so-called "the source-channel separation theorem", which is commonly adopted as the design concept by most modern digital communication systems, as shown in Fig. 1.1. In Fig. 1.1, the source data is first sampled, digitalized and losslessly compressed to reduce the information rate (source coding), then in the second step redundancies are added (channel coding, or equivalently, error control coding (ECC)) to combat noise. At the receiver, channel decoding and source decoding are consecutively implemented to try to recover original data with best effort.

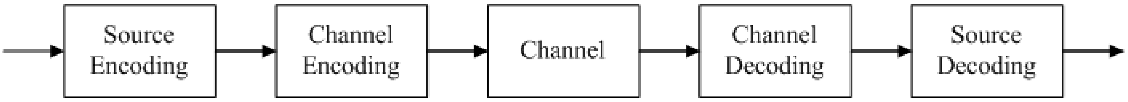

Figure 1.1: An end-to-end digital communication model.

Shannon's channel capacity theorem, however, is essentially a non-constructive existence proof: in order to achieve channel capacity C, one has to use randomly-selected codes of infinite length, and adopt maximum likelihood (ML) decoding or "typical-set decoding", all of which are extremely difficult, if not impossible, to implement in practice. It remains a challenging task for people to design good practical channel codes and at the same time efficiently decode them, and the main theme in the history of coding theory is essentially finding capacity-approaching ECC techniques with affordable decoding complexity.

Channel codes can be roughly categorized into *block codes* and *convolutional codes (CCs)*. For block codes, the encoded codeword only depends on the current information block, whereas in CCs the encoded output depends on not only the current information input, but also previous ν inputs, where ν is called "memory length" or "constraint length" of CCs. A block code $C(n, k)$ has a blocklength of n bits, among which k bits are *information bits*, and the remaining $n - k$ bits are *parity bits*. The values of k information bits uniquely determine the values of n bits. *The code rate* is defined as $R = k/n$, denoting the number of information bits carried per channel code bit.

A block code is a *linear* code if and only if (iff) any linear combination [1] of arbitrary number of codewords is another valid codeword. In the binary case, *the weight of a codeword* is defined as the number of "1"s in the codeword. The *Hamming distance between two codewords* is defined as the number of bit positions two codewords differ from each other, and the *minimum Hamming distance*

[1] over the field of the code symbols

d_{min} *of a code* is defined as the minimum of Hamming distances among all possible codeword pairs. It can be shown that for linear codes, d_{min} equals the smallest codeword weight in the code [8].

Channel codes designed in the 1950s and 1960s include Hamming codes [2], Reed-Muller codes [3], Bose-Chaudhuri-Hocquenghem (BCH) codes [4, 5] and Reed-Solomon (RS) codes [6]. The invention of Viterbi decoder [7] in 1970s enabled ML decoding of CCs. The concatenation of CC and RS codes was used 1980s. All the above codes evolved towards reducing the gap from channel capacity, as shown in Fig. 1.2.

Fig. 1.2 shows the ECC codes used from 1950s to 1990s by NASA in deep-space communications. From Fig. 1.2 it can also be observed that, even for the concatenated codes, there is still about 3 dB gap from channel capacity. This is due to the decoding complexity constraint: ML decoding has a decoding complexity growing as $2^{\min((n-k),k)}$ (for block codes) or 2^{ν} (for CCs) and at that time a near-ML decoding algorithm is unknown. Before the appearance of Turbo codes it was widely believed that a 2-3 dB gap from capacity was essentially inevitable.

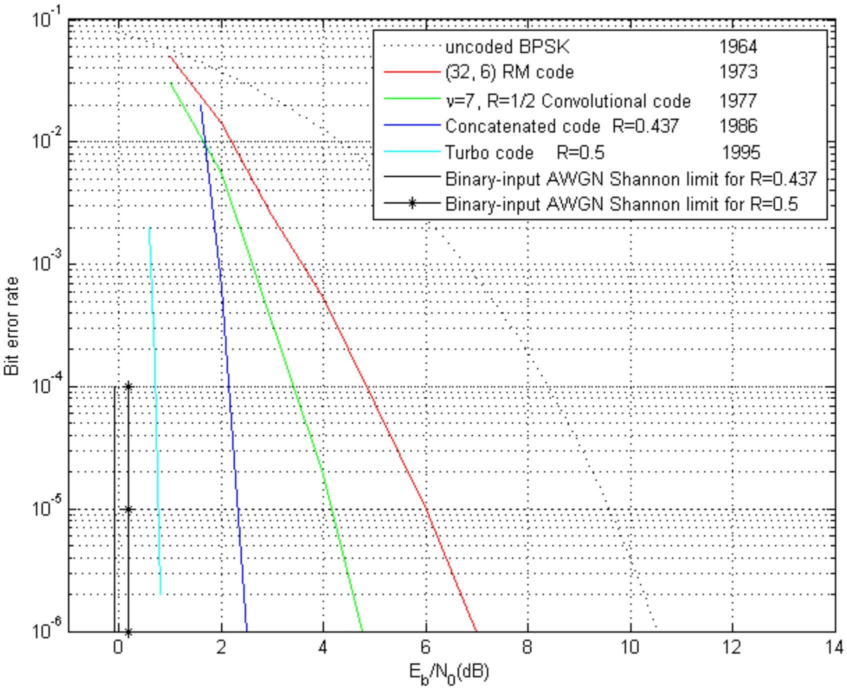

Figure 1.2: The development of ECC techniques.

The invention of Turbo codes [9] in 1993 is a landmark event in coding history. Turbo codes are composed by two or more constituent codes, which are concatenated by interleaver(s), either in parallel or serial fashion. During decoding, constituent soft-in soft-out (SISO) decoders passes *extrinsic* soft values to each other, in an alternating manner. It turns out that such low-complexity *iterative decoding* shows near-ML performance in cases of long blocklengths, as shown in Fig. 1.2. A length-65536 rate-0.5 Turbo code has the performance only about 0.7 dB away from capacity, at bit error rate (BER) around 10^{-5}.

Low-density parity-check (LDPC) codes, which were first discovered by Gallager in the 1960s, are another class of capacity-approaching codes. Before the emergence of iterative decoding, they were ignored by most researchers. A carefully-designed long-blocklength LDPC code can achieve performance within a few tenths or even hundredths of decibels of capacity under iterative decoding [12]. Compared with Turbo codes, LDPC codes have the advantages such as low error floor, parallel processing and simple early-termination criterion. LDPC codes have been adopted as ECC solutions in many modern communication standards such as IEEE 802.3an [18], DVB-S2 [19], WiMAX [20] and IEEE 802.11n [21].

1.2 Motivation of dissertation

LDPC codes can be further extended to generalized LDPC (GLDPC) codes [31–41]. GLDPC codes usually have good minimum distance properties [32, 33], compared with LDPC codes, and they generally have faster convergence speed, and have better error floor behavior. GLDPC codes can be further extended to doubly-generalized LDPC (DGLDPC) codes [42–50], and it is shown in [44, 46] that DGLDPC codes have better performance than LDPC codes in both waterfall region and error floor region. Waterfall region refers to the region in which the error curve decreases quickly with the increase of signal-to-noise ratio (SNR), and error floor region is where error curve falls more slowly with the increase of SNRs, which usually happens when SNR is high.

GLDPC and DGLDPC codes are more flexible relative to LDPC codes since they are supersets of LDPC codes, and can exhibit many advantages over LDPC codes in numerous scenarios. In addition, GLDPC and DGLDPC codes constitute a bridge between classical algebraic codes and modern iteratively-decodable codes. Study of GLDPC and DGLDPC codes helps understand better the inherent mechanism and nature of the iterative decoding, which will greatly contribute to the modern coding theory.

In existing research literature, people usually focus on the asymptotic behavior of DGLDPC codes and build very long blocklength codes [44, 46], or moderate-length DGLDPC codes with the same check node (CN) types [50]. For general *short-to-moderate length* GLDPC and DGLDPC codes, the research on designing those codes with low error floor is lacking.

In practice, finding efficiently-encodable (EE) codes is an important issue. The brute-force encoding method has the complexity $O(N^2)$, where N is the code length, whereas some carefully-designed codes enjoy linear or even sub-linear encoding complexity. There are a few papers on constructing EE-GLDPC codes with specific structures [37–39], however, the systematic design of EE-GLDPC codes and EE-DGLDPC codes are still open issues.

Another topic related to EE codes is the design of rate-compatible (RC) codes. Using RC codes as ECC schemes is a popular method for time-varying channel (e.g. in wireless communications). In addition, RC codes can also conveniently provide a range of "fixed" code rates, which could be an important feature in ECC systems. In existing studies, RC-LDPC codes are widely studied in [54–66], nevertheless there are few studies on RC-GLDPC and RC-DGLDPC codes, making it an unexplored research area.

The above three inadequacies of existing research are the main research topics of this work. In this work we first propose a new concept, namely eACE, to help improve the error floor behavior of finite-length (FL) GLDPC and DGLDPC codes. We also propose systematic design principles and algorithms for designing EE-GLDPC and EE-DGLDPC codes, providing an across-the-board solution for EE block codes. Finally, systematic puncturing algorithms are devised for EE-GLDPC and EE-DGLDPC codes to improve performance of variable rate transmission.

1.3 Summary of dissertation

Chapter 2 provides the preliminary background on this work. It first gives the description of LDPC, GLDPC and DGLDPC codes, then the iterative decoding algorithms are presented. Next, density evolution, extrinsic information transfer (EXIT) charts and differential evolution (DE), which are useful tools for determining and optimizing the *decoding thresholds* of code ensembles, are discussed. Finally we present the progressive-edge-growth (PEG) algorithm and the notion of approximate cycle extrinsic message degree (ACE) for designing FL-LDPC codes.

Chapter 3 is on designing short-length GLDPC and DGLDPC codes with good error floors. For GLDPC and DGLDPC codes, ACE is generalized to extended ACE (eACE). We propose two modified PEG algorithms incorporating eACE, and it is shown that by doing so better error floor behavior can be obtained. In the end, we offer the evidence that for shorter DGLDPC codes, eACE plays a more important role than the traditional girth property.

Chapter 4, 5 and 6 focus upon designing EE-GLDPC and EE-DGLDPC codes. By generalizing the idea of irregular-repeat accumulate (IRA) and efficiently-encodable rate-compatible (EERC) LDPC codes, principles and algorithms for EE-GLDPC codes are devised and discussed in detail in Chapter 4. We show that the well-known IRA-LDPC and EERC-LDPC codes are two special cases of EE-LDPC codes when some specific recovery vector **n**'s are applied. We also give some examples of simple EE-GLDPC codes and illustrate the encoding processes in detail. Simulations show that the additional EE property does not degrade the code performance: EE-GLDPC codes have similar performance with ordinary GLDPC codes.

Chapter 5 further extends EE-GLDPC to EE-DGLDPC codes. The existence of super variable nodes (SVNs) makes the design of EE-DGLDPC codes more involved: each SVN contains a certain number of *constrained* and *free* edges; these should be treated differently when building the EE-DGLDPC codes. Due to the existence of constrained and free edges, if the same decoder structure is used for decoding, we need to permute the edges connected to SVNs and super check nodes (SCNs) to ensure accurate decoding. The encoding processes of toy codes are also provided for better understanding.

The encoding speed and error floor behavior of EE codes are discussed in Chapter 6. Theoretically we can design EE codes with any encoding time complexity, such as $O(\log_2 N)$ or even $O(1)$; nonetheless, the incurred expense of faster encoding speed is the poorer error floor performance. The tradeoff between encoding speed and error floor is verified by both theoretical analysis and extensive simulation results, for EE-LDPC, EE-GLDPC and EE-DGLDPC codes.

Compared with LDPC codes, the SCNs and SVNs in GLDPC and DGLDPC codes exhibit more complicated decoding behaviors. The systematic puncturing algorithms for EE-GLDPC and EE-DGLDPC codes are topics of Chapter 7. A good puncturing algorithm can to a large degree resolve the rate constraint issue of traditional sGLDPC codes, i.e. sGLDPC codes are not suitable for designing high-rate codes. Simulation results show that our systematic puncturing has considerable performance improvement over random puncturing. In addition, the punctured codes, like their mother codes, maintain the advantage of good error floor performance, which can be a very important feature in practical applications. Simulation results indicate that EE-GLDPC and EE-DGLDPC codes show competitive advantages over EERC-LDPC and IRA-LDPC codes, especially at high SNRs, over a wide range of code rates.

Chapter 8 is the conclusion of dissertation and some open topics are brought up for future research.

Chapter 2

Preliminaries on LDPC, GLDPC and DGLDPC Codes

2.1 Code structures and representations

2.1.1 LDPC codes

LDPC codes, first introduced by Gallager in [10], are linear block codes which can be specified by a sparse *parity-check matrix* $H_{M\times N}[i][j]$, where H has M rows and N columns. In this work, we only consider binary LDPC codes. Let x be a binary vector of length N. Then x is a valid LDPC codeword iff it satisfies all the parity-check constraints imposed by H, i.e. $xH^T = \mathbf{0}$ (in modulo-2 sense). In other words, a LDPC code is the collection of vectors in the null space of H.

If H has full row rank, we have $K = N - M$ independent bits, and the remaining M bits are completely and uniquely determined by these K bits. In this case such LDPC code has K information bits and M parity bits, and the corresponding code rate $R = \frac{(N-M)}{N}$. We define the *design code rate* as $R_d = \frac{(N-M)}{N}$, in certain cases the rows of H are not necessarily linearly independent therefore we have $R > R_d$. In practice, if the PEG algorithm is used to generate an LDPC code, we usually have $R = R_d$ or a rate very close to R_d, while for some special subclasses of LDPC codes such as geometric LDPC codes [22], the actual code rate is usually much higher than the design rate.

The *row weight* of a certain row of H matrix is the number of “1”s in this row, similarly we have the *column weight* of a column. If an H matrix has a constant column weight dv and a row weight dc, it is called a *regular LDPC code*, denoted by "a (dv, dc)-regular LDPC code", the corresponding code rate $R \geq (1 - \frac{dv}{dc})$. If row weight or column weight are not constants, then it is called an *irregular LDPC code*.

In (2.1) we give a simple $(2, 4)$ regular LDPC code with $N = 10$ and $M = 5$.

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{2.1}$$

The LDPC code defined in (2.1) has $K = 4$, which is smaller than $(N - M) = 5$. This is because the rows are not independent: if we sum all the rows we obtain an all-zero vector. In addition, the H matrix has 40% "1"s and 60% "0"s. Strictly speaking, it cannot be called a "LDPC code", nonetheless we show this small matrix for illustration's sake. Practical codes have much lower density of "1"s in H matrix. In Fig. 2.1 the H matrix of a $(N = 1008, K = 504)(3, 6)$-regular LDPC code is shown, the dots in the figure are "1"s. The density of "1"s is $3/504 \approx 0.595\%$. The sparsity of LDPC codes enables low-complexity iterative message-passing decoders, which makes near-ML decoding possible.

LDPC codes can also be represented by a *bipartite graph*, or *Tanner graph* [31]. A Tanner graph contains two disjoint classes of nodes: variable nodes (VNs) and check nodes (CNs). An $H_{M\times N}[i][j]$ matrix corresponds to a Tanner graph with N VNs and M CNs. There is an *edge* connecting the jth VN and ith CN iff $H[i][j] = 1$. In the Tanner graph, each VN denotes a bit in the codeword, and each CN imposes a single parity-check (SPC) constraint on the set of VNs incident to it. The Tanner graph and parity-check matrix H are essentially equivalent to each other. The corresponding Tanner graph of H matrix in (2.1) is shown in Fig. 2.2.

For LDPC codes, given the VN degree vector $\mathbf{dv} = (2, 3..., dvmax)$ and CN degree vector $\mathbf{dc} = (2, 3..., dcmax)$, where $dvmax$ and $dcmax$ are the maximum degrees of VNs and CNs, the degree distribution $(\boldsymbol{\lambda} \doteq (\lambda_2, \lambda_3, ..., \lambda_{dvmax}), \boldsymbol{\rho} \doteq (\rho_2, \rho_3, ..., \rho_{dcmax}))$ prescribes the way edges connect

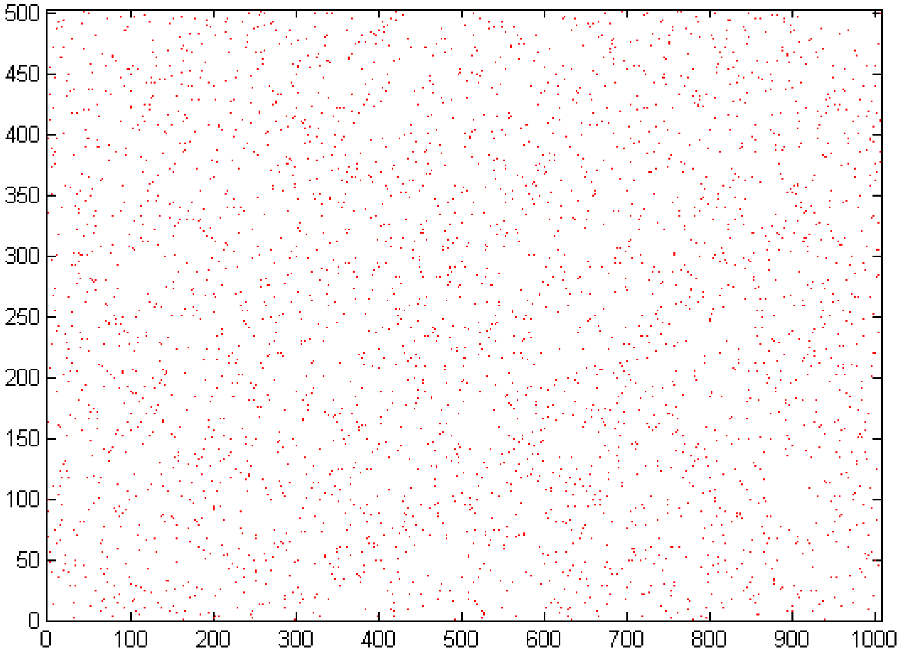

Figure 2.1: The structure of H matrix of a $(1008, 504)(3, 6)$-regular LDPC code.

VNs and CNs, namely, in the Tanner graph a λ_i fraction of edges are incident to VNs with degree $dv = i$, where $i = 2, 3...dvmax$, and a ρ_j fraction of edges incident to CNs with degree $dc = j$, where $j = 2, 3...dcmax$. The *degree distribution generating polynomials* $\lambda(x)$ and $\rho(x)$ are commonly used to describe the LDPC code. These are defined as

$$\begin{aligned}\lambda(x) &= \sum_{i=2}^{i=dvmax} \lambda_i x^{i-1} \\ \rho(x) &= \sum_{j=2}^{j=dcmax} \rho_j x^{j-1}.\end{aligned} \tag{2.2}$$

It can be directly obtained that [14]

$$\begin{aligned}&\lambda(1) = \rho(1) = 1 \\ &R_d = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx} = 1 - \frac{\sum_{j=2}^{j=dcmax} \frac{\rho_j}{j}}{\sum_{i=2}^{i=dvmax} \frac{\lambda_i}{i}}\end{aligned} \tag{2.3}$$

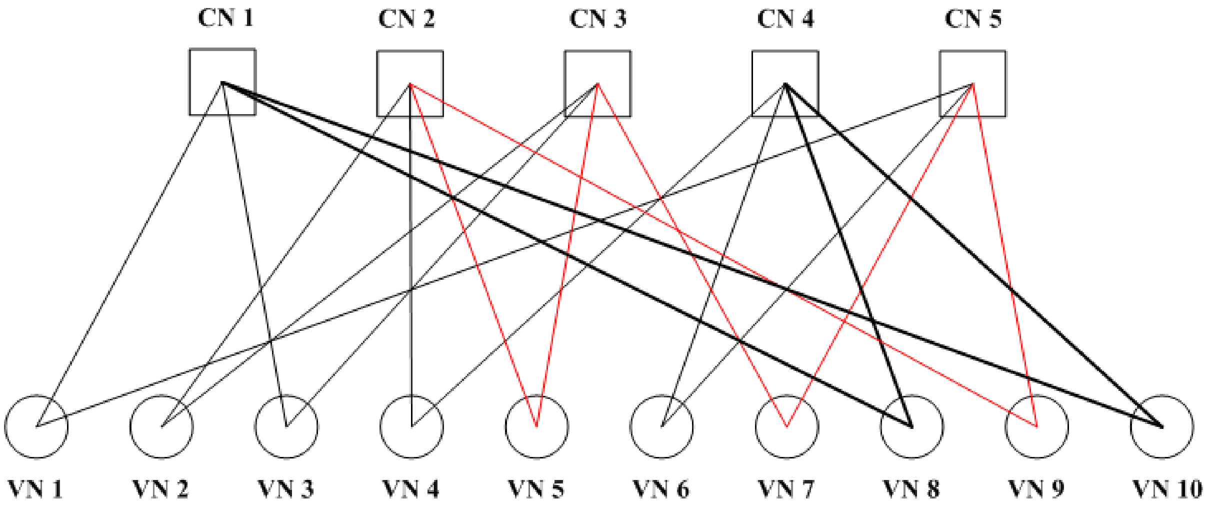

Figure 2.2: The Tanner graph of a (10,5) LDPC code.

2.1.2 Important graph properties

The iterative decoding of LDPC codes is essentially performed by message passing on the Tanner graph, therefore the decoding performance is considerably affected by the graph properties. Several important graph parameters are listed below.

Cycle: A length-l cycle in Tanner graph can be defined as a length l path, starting from a certain node s, traversing $(l-1)$ unique nodes, and ending in the same node s. In Fig. 2.2, the bold black lines represent a length-4 cycle: the path starts from CN 1, after traversing VN 8, CN 4 and VN 10, ends in CN 1 again. Similarly, the red lines in Fig. 2.2 form a length-6 cycle. It is straightforward to see that for Tanner graph the lengths of cycles are always even; furthermore, if no parallel edges between two nodes are allowed (which is a basic principle in iteratively-decodable codes), the shortest cycle length is 4.

Girth of a Tanner graph: The shortest cycle length in the Tanner graph.

The neighbors of a node: The neighbor set of a node s, $\mathcal{N}(s)$, is defined as $\mathcal{N}(s) \doteq \{t|$ there is an edge directly connecting s and $t\}$

Local tree: For a certain node s, a *level-l local tree rooted in s* can obtained recursively as follows: take s as the root, unfold once the Tanner graph in a breadth-first fashion, so that all the nodes in

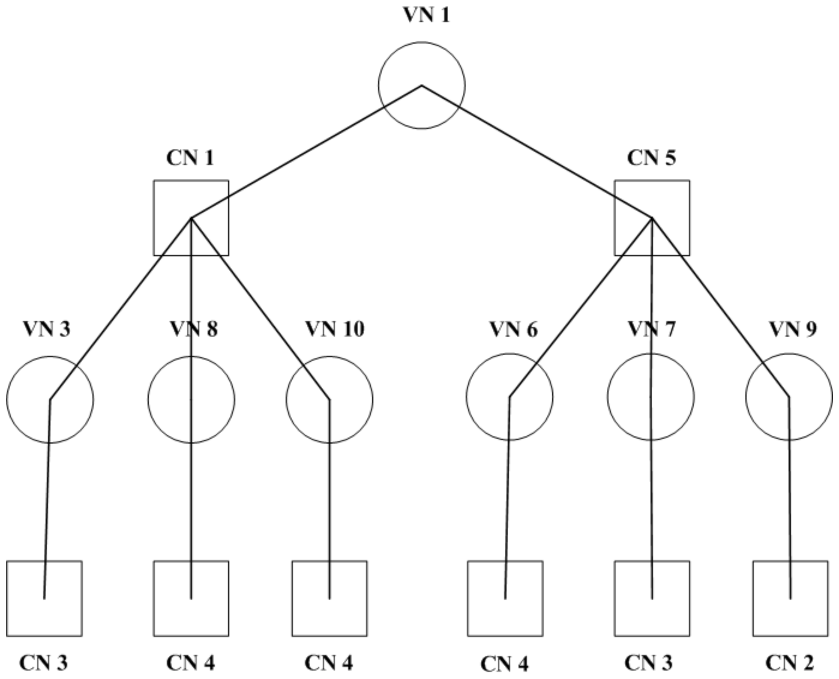

Figure 2.3: The level-3 local tree rooted in VN 1.

$\mathcal{N}(s)$ are traversed; this is *level*-1 *local tree rooted in* s, and s is called the *parent node* of the nodes in the 1st-layer. For each node in $\mathcal{N}(s)$, unfold the Tanner graph breadth-first again so that all the neighbors of nodes in $\mathcal{N}(s)$, excluding the parent node s, are traversed, then we obtain *level*-2 *local tree rooted in* s; the nodes in $\mathcal{N}(s)$ are parent nodes of 2nd-layer nodes. The procedure is done recursively until the level-l local tree is established. For LDPC code in (2.1), Fig. 2.3 shows a level-3 local tree rooted in VN 1.

Practical codes will inevitably contain cycles [74]. The existence of cycles in the Tanner graph will cause a dependence of messages during the belief-propagation (BP) decoding, which is the root of the suboptimality of BP algorithm. Generally speaking, short cycles will cause a poorer error floor, and should be avoided when designing an iteratively-decodable code. Nevertheless, short cycles do not necessarily always degrade the decoding; other factors can also possibly affect the code performance. A code with larger girth could possibly have a poorer error floor behavior than its counterpart with a smaller girth. Such phenomenon will be discussed in detail in Chapter 3.

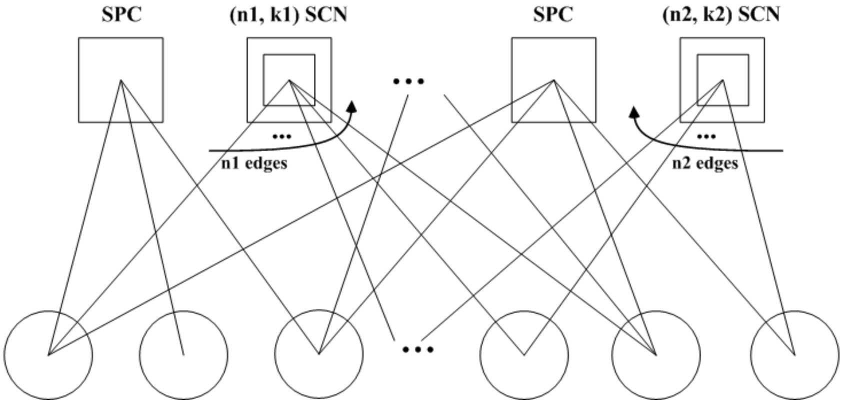

Figure 2.4: The Tanner graph of GLDPC codes.

2.1.3 GLDPC codes

In the Tanner graph of a LDPC code, each CN puts an SPC constraint on the edges incident to it, in other words, a CN with degree n requires that the incident edge variables be a valid codeword of SPC$(n, n-1)$. If a CN imposes the constraint that the incident edge variables be a valid codeword of some linear code $C(n, k)$ other than SPC$(n, n-1)$, such CN is referred to as super check node (SCN). A GLDPC code can be obtained by replacing *some or all* SPC-CNs by SCNs in LDPC Tanner graph. Fig. 2.4 illustrates the structure of the Tanner graph of GLDPC codes.

The H matrix equivalent to the Tanner graph of a GLDPC code is commonly known as the *adjacency matrix*, denoted by H_{adj}. An edge connects the jth VN and ith CN iff $H_{adj}[i][j] = 1$. A row in H_{adj} corresponding to a $C(n, k)$ SCN (a $C(n, k)$-SCN row), imposes $n-k$ parity checks on the codeword, and $C(n, k)$ is usually referred to as the *component code*. Finally, the parity-check matrix of $C(n, k)$, is denoted by H_{comp} throughout this dissertation; a GLDPC code is fully described by its adjacency matrix and H_{comp}'s of all the SCNs.

The parity-check matrix of a GLDPC code, H_{SPC}, can be attained through *row expansion* [44]: for a $C(n, k)$-SCN row, the ith "1" in the row is replaced by the ith column of H_{comp}, for $i = 1, 2..., n$, and each "0" in the row is replaced by an all-zero column of length $(n-k)$. H_{SPC} is obtained after all SCN rows are expanded. As an example, (2.4) gives a H_{adj} where the second row is a SCN row, which is shown in bold font.

$$H_{adj} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Suppose H_{comp} is

$$H_{comp} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{2.5}$$

After row expansion, H_{SPC} is

$$H_{SPC} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

In this work, we call a GLDPC code “strict-sense” if all the CNs use the *same* linear code as their component codes, denoted by ”sGLDPC code” [32–37], otherwise it is called a hybrid GLDPC code, denoted by ”hGLDPC code” [40, 41].

The code rate of GLDPC codes can be obtained by calculating the rank of H_{SPC}, or using (2.16). For an sGLDPC code using $C(n, k)$ as its component code, let $\overline{dv}$ be the average degree of VNs. The design code rate can be concisely written as

$$R_d = 1 - \overline{dv} \cdot \frac{(n-k)}{n} \tag{2.7}$$

For GLDPC codes, the existence of SCNs can improve the code minimum distance [33], relative to LDPC codes. In addition, SCNs have more powerful error correcting capability than SPC codes, therefore GLDPC codes generally show better error floor behaviors than LDPC codes. One drawback of GLDPC codes is the so-called “*rate loss under iterative decoding*”, which refers to the fact that GLDPC codes suffer from a certain degree of *decoding threshold degradation* under iterative decoding [15, 16, 36]. The rate loss issue is more prominent on sGLDPC codes, which can

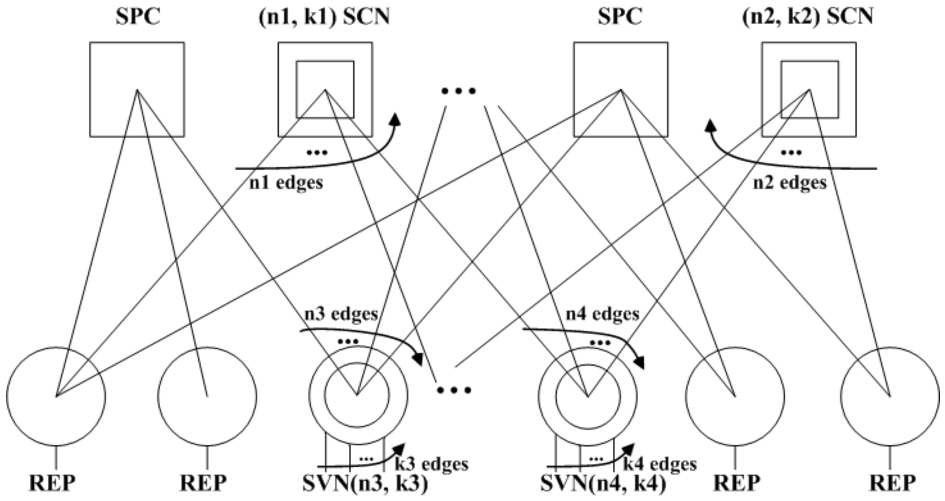

Figure 2.5: The Tanner graph of DGLDPC codes.

be addressed by (1) using hGLDPC codes, (2) generalizing GLDPC codes to DGLDPC codes, or (3) efficient puncturing, which will be discussed in Chapter 7.

2.1.4 DGLDPC codes

In the Tanner graph of LDPC and GLDPC codes, each VN puts a repetition (REP) constraint on the edge variables incident to it; equivalently, a VN with degree n requires that the incident edge variables be a valid codeword of REP$(n, 1)$. If a VN imposes the constraint that the incident edge variables be a valid codeword of some linear code $C(n, k)$ other than REP$(n, 1)$, such VN is referred to as super variable node (SVN). A DGLDPC code can be obtained by replacing some or all REP-VNs by SVNs in the LDPC or GLDPC Tanner graph. Fig. 2.5 illustrates the structure of the Tanner graph of DGLDPC codes.

In the Tanner graph of DGLDPC codes, for a SVN using $C(n, k)$ as its component code, the edge variables incident to this SVN should be a valid codeword of $C(n, k)$, and it has k edges connected to the communication channel. Note that for REP it has one edge coupled with the channel, as shown in Fig. 2.5, however such edges are commonly omitted in the Tanner graphs of LDPC (Fig. 2.1) and GLDPC codes (Fig. 2.4). The H matrix equivalent to the Tanner graph of a DGLDPC code is also referred to as *adjacency matrix*, denoted by H_{adj}. The generator matrix of $C(n, k)$, G_{comp}, is necessary to describe the specific structure of SVN.

For LDPC and GLDPC codes, a column with weight n corresponds to one codeword bit, and the values of n bits in the column are the same (REP constraint); whereas for DGLDPC codes a column in H_{adj} corresponding to a SVN using $C(n,k)$ as its component code (a $C(n,k)$-SVN column), has weight-n and represents k codeword bits, and the values of n bits in the column are usually different, since they are determined by G_{comp} using these k codeword bits ($C(n,k)$ constraint).

The parity-check matrix of a DGLDPC code H_{SPC}, can be obtained by first a *row expansion* then a *column expansion* [44]: The row expansion is done first exactly the way done in GLDPC codes. Then the column expansion is carried out as follows: for a $C(n,k)$-SVN column, the ith "1" in the column is replaced by the ith row of G_{comp}^T, the transposed matrix of G_{comp}, for $i = 1, 2...n$, and each "0" in the column is replaced by an all-zero row of length k. Finally H_{SPC} is obtained when all the SVN columns are expanded. As an example, (2.8) gives a H_{adj} where the second row is a SCN row, shown in bold font, and the fifth column is a SVN column, shown in red color.

$$H_{adj} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.8}$$

Suppose H_{comp} is defined by a $(4,2)$ code:

$$H_{comp} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{2.9}$$

After row expansion, we have

$$\widetilde{H} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.10}$$

Suppose G_{comp} is a $(3, 2)$ code:

$$G_{comp} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.11}$$

After column expansion we have

$$H_{SPC} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.12}$$

producing an $(8, 3)$ code.

It should be noted that a DGLDPC code is fully described by its H_{adj}, the G_{comp}'s of all the SVNs and the H_{comp}'s of all the SCNs. If G_{comp} assumes a different form, it may change the DGLDPC code. For example, if we use another G_{comp} as

$$G_{comp} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{2.13}$$

we will have another DGLDPC code, which contains a weight-6 codeword, while the code defined in (2.12) has a maximum codeword weight 5. Similarly if H_{comp} is changed, the code could also be possibly changed. On the other hand, in the viewpoint of asymptotic analysis, the permutation of columns of H_{comp} will not affect the decoding threshold of a code ensemble, but the permutation of columns of G_{comp} will, which will be discussed in detail in subsection 2.3.2.

In the following, for the sake of clarity and simplicity, the term "REP-VN" denotes a REP VN and "VN" can refer to either SVN or REP-VN, similarly "SPC-CN" denotes a SPC CN and "CN" can refer to either SCN or SPC-CN. Lastly we want to point out that the decoding of GLDPC and DGLDPC codes, are commonly implemented on H_{adj} instead of H_{SPC}; this is because H_{comp} and G_{comp} usually contain many length-4 cycles, which degrade the performance of iterative decoding. Graph properties such as cycles, girth and local trees, have exactly the same definitions on H_{adj}.

A comprehensive way to characterize iteratively-decodable block codes

The VN and CN types are greatly enriched in GLDPC and DGLDPC codes. In order to have a concise description of the properties of iteratively-decodable codes (LDPC,GLDPC and DGLDPC codes), the following notations are introduced: denote by $\mathcal{T}_V$ and $\mathcal{T}_C$ the sets of VN and CN types, respectively. Let $\mathcal{I}_V$ and $\mathcal{I}_C$ be the sets of indices for the VN and CN types, respectively. For $i \in \mathcal{I}_V$ ($j \in \mathcal{I}_C$), let n_{vi} (n_{cj}), k_{vi} (k_{cj}), and r_{vi} (r_{cj}) denote the code length, the information length and the rate of type-i (type-j) component code on the VN (CN) side. Define the degree profile $(\boldsymbol{\lambda}, \boldsymbol{\rho}) \doteq ((\lambda_1, \lambda_2, ..., \lambda_i, ..., \lambda_{|\mathcal{T}_V|}), (\rho_1, \rho_2, ..., \rho_j, ..., \rho_{|\mathcal{T}_C|}))$, where λ_i (ρ_j) denotes the fraction of edges connected to the type-i (type-j) component code on the VN (CN) side. Finally define:

$$\begin{aligned} \lambda(x) &\doteq \sum_{i \in \mathcal{I}_V} \lambda_i x^{n_{vi}-1} \\ \rho(x) &\doteq \sum_{j \in \mathcal{I}_C} \rho_j x^{n_{cj}-1} \end{aligned} \tag{2.14}$$

Let C be a bipartite-graph based code and H_{adj} have dimension $N_C \times N_V$, i.e. there are N_C CNs and N_V VNs in the bipartite graph. The code length N of code C, and the number of parity-check equations M can be calculated as follows [45]:

$$\begin{aligned} N &= \frac{N_V}{\int_0^1 \lambda(x)dx} \cdot \sum_{i \in \mathcal{I}_V} \frac{\lambda_i k_{vi}}{n_{vi}} \\ M &= \frac{N_C}{\int_0^1 \rho(x)dx} \cdot \sum_{j \in \mathcal{I}_C} \frac{\rho_j (n_{cj} - k_{cj})}{n_{cj}} \end{aligned} \tag{2.15}$$

The *design code rate* R_d, which is a generalization of (2.3), can be calculated as:

$$R_d = 1 - \frac{\sum_{j \in \mathcal{I}_C} \rho_j (1 - r_{cj})}{\sum_{i \in \mathcal{I}_V} \lambda_i r_{vi}} \tag{2.16}$$

2.2 Decoding algorithms

In iterative decoding, VNs and CNs exchange *extrinsic information* in an alternating manner hoping to find a valid codeword. In this work we focus on BP decoding on the additive white Gaussian noise (AWGN) channel, nonetheless the algorithms described here can be readily applied to other channels which yield multi-level quantized outputs or real outputs. Decoding GLDPC/DGLDPC codes on binary erasure channels (BEC) can be found in [36, 45–47].

2.2.1 SPA for LDPC codes

The BP decoding of LDPC codes can be implemented in either probabilistic domain [11], or log-likelihood ratio (LLR) domain, and both are essentially identical in principle. LLR is a characterization of the random variable (RV) assuming two values, defined as the logarithm of Likelihood Ratio (LR), where LR is the ratio between the probability of a RV is 0 and the probability is 1. For example, if a RV assumes "0" or "1" equiprobably, the corresponding LLR is $\log(0.5/0.5) = 0$.

In this work we only describe sum-product algorithm (SPA), which is a special form of LLR-BP decoding. Consider a (N, K) LDPC code, a codeword $\boldsymbol{c} = (c_1, c_2, ..., c_N)$, where $(c_i \in \{0, 1\})$ is modulated using unit-energy binary phase-shift keying (BPSK) into $\boldsymbol{x} = (x_1, x_2, ..., x_N)$, where $x_i \in \{-1, +1\}$. The mapping rule is $x_i = 1 - 2 \cdot c_i$, so the Boolean 1 is mapped to -1 and 0 to $+1$. The codeword is transmitted on AWGN channel. At the receiver, let $\boldsymbol{y} = (y_1, y_2, ..., y_N)$ be the received real-valued sequence, and $y_i = x_i + n_i$, $i = 1, 2, ..., N$. n_i's are independently, identically distributed (i.i.d.) Gaussian RV with zero mean and variance $\mathcal{N}_0/2$.

The "channel LLR" for x_i (in LLR form), provided by the channel data, is calculated as

$$Y_i = \log(\frac{P(x_i = 0|y_i}{P(x_i = 1|y_i}) = \frac{4y_i}{\mathcal{N}_0} \tag{2.17}$$

Let $R_{ji}^{(l)}$ be the so-called extrinsic message (in LLR form) from CN j to VN i in the lth iteration, similarly define $Q_{ij}^{(l)}$ the extrinsic LLR message from VN i to CN j in the lth iteration. For the lth iteration, the update of VN is

$$Q_{ij}^{(l)} = \sum_{k \in \mathcal{N}(i) \backslash j} R_{ki}^{(l-l)} + Y_i \tag{2.18}$$

where $\mathcal{N}(i) \backslash j$ denotes the set of CNs which are neighbors of VN i, excluding CN j. Note that the soft-output maximum-likelihood (SOML) decoding of a REP-VN i given $R_{ij}^{(l-1)}$ and Y_i is given by

$$W_i^{(l-1)} = \sum_{k \in \mathcal{N}(i)} R_{ki}^{(l-1)} + Y_i \tag{2.19}$$

The VN update rule (2.18) and (2.19) indicate that the *extrinsic message* Q_{ij} can be obtained from SOML decoding result W_i, which is the sum of all incoming LLR messages and channel LLR, after subtracting the *intrinsic message* (or *a-priori message*) R_{ji}.

At the beginning of decoding, since $R_{ki} = 0$, (2.18) is initialized as

$$Q_{ij}^{(1)} = Y_i \tag{2.20}$$

For CN update in the lth iteration, it can be written as

$$\tanh\,(\frac{R_{ji}^{(l)}}{2}) = \prod_{k \in \mathcal{N}(j) \setminus i} \tanh(\frac{Q_{kj}^{(l)}}{2}) \tag{2.21}$$

where $\tanh(\cdot)$ is hyperbolic tangent function. It is worth noting that the extrinsic message R_{ji} obtained by (2.21) is derived from the SOML decoding of the SPC-CN. (2.18) has the form of "sum" and (2.21) has the form of "product", therefore it is called "sum-product algorithm" (SPA).

SPA is a special form of BP algorithm when SOML decoding is used on both REP-VNs and SPC-CNs. (2.21) involves nonlinear operations, therefore in practice many simplified BP algorithms such as "min-sum" [23], "normalized min-sum" [24] and "off-set min-sum" [24] are proposed. Compared with SPA, they reduce the iterative decoding complexity to certain degree at the expense of a slightly degraded decoding performance.

Finally, we give the procedures of SPA below. The maximum number of iterations is denoted by L. In practice L is usually chosen in the range 20-200. After L iterations if $\widehat{\boldsymbol{c}} \cdot H^T \neq \mathbf{0}$, we say that SPA fails to achieve successful decoding and take $\widehat{\boldsymbol{c}}$ as the decoding output.

SPA procedure

1. **Initialization**: $l = 1$, initialize $Q_{ij}^{(l)} = Y_i$, for $j \in \mathcal{N}(i), i = 1, 2..., N$.
2. **Horizontal Step**: update $R_{ji}^{(l)}$ by $\tanh(\frac{R_{ji}^{(l)}}{2}) = \prod_{k \in \mathcal{N}(j) \setminus i} \tanh(\frac{Q_{kj}^{(l)}}{2})$
3. **Vertical Step**: update $Q_{ij}^{(l+1)}$ by $Q_{ij}^{(l+1)} = \sum_{k \in \mathcal{N}(i) \setminus j} R_{ki}^{(l)} + Y_i$
4. **Codeword Check**: calculate *a-posteriori* probability (APP) $W_i^{(l)} = \sum_{k \in \mathcal{N}(i)} R_{ki}^{(l)} + Y_i$.

Decide $\widehat{c}_i = 0$ if $W_i \geq 0$, else $\widehat{c}_i = 1$. If $\widehat{\boldsymbol{c}} = (\widehat{c}_1, \widehat{c}_2, ..., \widehat{c}_N)$ does not satisfy $\widehat{\boldsymbol{c}} \cdot H^T = \mathbf{0}$ and $l < L$, then $l = l + 1$ and goto step 2, otherwise stop and output $\widehat{\boldsymbol{c}}$ as the decoding result.

2.2.2 Iterative decoding of GLDPC and DGLDPC codes

For GLDPC and DGLDPC codes, the extrinsic messages sent from SCNs and SVNs are needed to be calculated based on the incoming messages. The SOML decoding of SCN/SVN involves more complicated procedures than SPC-CNs/REP-VNs. For the sake of clarity, we adopt the following notations. Let N_V and N_C be the number of VNs and CNs in the Tanner graph, the jth CN uses $C(n_{cj}, k_{cj})$ as its component code, similarly the ith CN uses $C(n_{vi}, k_{vi})$ as its component code.

Let codeword $\boldsymbol{b} = (\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_{N_V})$, where $\boldsymbol{b}_i = \{b_{i,1}, b_{i,2}, ...b_{i,k_{vi}}\}$ ($b_{i,j} \in \{0, 1\}$), is modulated by BPSK into $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_{N_V})$, where $\boldsymbol{c} = 1 - 2 \cdot \boldsymbol{b}$. At the receiver, $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_{N_V})$ is the received vector, $\boldsymbol{Y} = (\boldsymbol{Y}_1, \boldsymbol{Y}_2, ..., \boldsymbol{Y}_{N_V})$ is the channel LLR vector. For the ith VN, it receives channel LLR $\boldsymbol{Y}_i = \{Y_{i,1}, Y_{i,2}...Y_{i,k_{vi}}\}$.

Enumerating edges from VNs' side, define

- $Q_{i,j}^{(l)}$ is the extrinsic message (from VN to CN) on the jth edge of the ith VN in lth iteration.
- $P_{i,j}^{(l)}$ is the incoming message (from CN to VN) the jth edge of the ith VN in lth iteration.

Similarly enumerating edges from CNs' side,

- $R_{i,j}^{(l)}$ is the extrinsic message (from CN to VN) on the jth edge of the ith CN in lth iteration.
- $S_{i,j}^{(l)}$ the incoming message (from VN to CN) on the jth edge of the ith CN in lth iteration.

The vector forms can be written as

$$\begin{aligned}
\boldsymbol{Q}^{(l)} &= \{Q_{1,1}^{(l)}, Q_{1,2}^{(l)}, ..., Q_{1,n_{v1}}^{(l)}, Q_{2,1}^{(l)}, Q_{2,2}^{(l)}, ..., Q_{2,n_{v2}}^{(l)}, ..., ..., Q_{N_V,1}^{(l)}, Q_{N_V,2}^{(l)}, ..., Q_{N_V,n_{N_V}}^{(l)}\}. \\
\boldsymbol{P}^{(l)} &= \{P_{1,1}^{(l)}, P_{1,2}^{(l)}, ..., P_{1,n_{v1}}^{(l)}, P_{2,1}^{(l)}, Q_{2,2}^{(l)}, ..., P_{2,n_{v2}}^{(l)}, ..., ..., P_{N_V,1}^{(l)}, P_{N_V,2}^{(l)}, ..., P_{N_V,n_{N_V}}^{(l)}\}. \\
\boldsymbol{R}^{(l)} &= \{R_{1,1}^{(l)}, R_{1,2}^{(l)}, ..., R_{1,n_{c1}}^{(l)}, R_{2,1}^{(l)}, R_{2,2}^{(l)}, ..., R_{2,n_{c2}}^{(l)}, ..., ..., R_{N_C,1}^{(l)}, R_{N_C,2}^{(l)}, ..., R_{N_C,n_{N_C}}^{(l)}\}. \\
\boldsymbol{S}^{(l)} &= \{S_{1,1}^{(l)}, S_{1,2}^{(l)}, ..., S_{1,n_{c1}}^{(l)}, S_{2,1}^{(l)}, S_{2,2}^{(l)}, ..., S_{2,n_{c2}}^{(l)}, ..., ..., S_{N_C,1}^{(l)}, S_{N_C,2}^{(l)}, ..., S_{N_C,n_{N_C}}^{(l)}\}.
\end{aligned} \tag{2.22}$$

Denote by $\boldsymbol{\Pi}$ the permutation between edges from VNs and CNs. Without loss of generality, we assume

$$\begin{aligned}
\boldsymbol{S}^{(l)} &= \boldsymbol{\Pi}(\boldsymbol{Q}^{(l)}) \\
\boldsymbol{R}^{(l)} &= \boldsymbol{\Pi}(\boldsymbol{P}^{(l)})
\end{aligned} \tag{2.23}$$

The relationship between $\boldsymbol{Q}$, $\boldsymbol{P}$, $\boldsymbol{S}$ and $\boldsymbol{T}$ is depicted in Fig. 2.6.

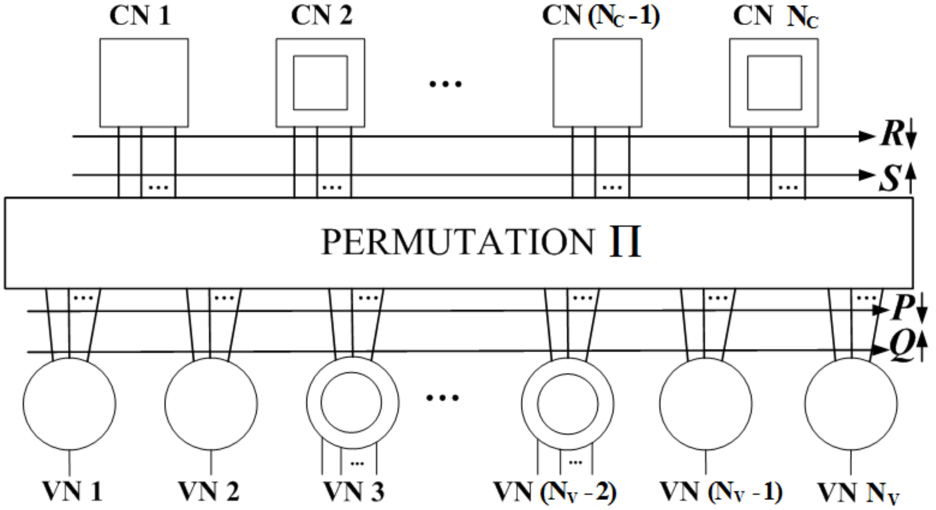

Figure 2.6: The relationship between $\boldsymbol{Q}$, $\boldsymbol{P}$, $\boldsymbol{S}$ and $\boldsymbol{T}$ during the DGLDPC decoding.

Let $\{\boldsymbol{x}_i\}$ be the codeword of the component code used by the ith CN, and $x_{i,j}$ the jth bit in the codeword. The ith CN update using SOML decoding can be written as

$$R_{i,j}^{(l)} = \log \frac{\sum\limits_{\boldsymbol{x}_i : x_{i,j}=0} \prod\limits_{p=1, p \neq j}^{n_{ci}} e^{-S_{i,p}^{(l)} \cdot x_{i,p}}}{\sum\limits_{\boldsymbol{x}_i : x_{i,j}=1} \prod\limits_{p=1, p \neq j}^{n_{ci}} e^{-S_{i,p}^{(l)} \cdot x_{i,p}}} \tag{2.24}$$

Let $\{\boldsymbol{z}_i\}$ be the codeword of the component code used by the ith VN, and $z_{i,j}$ the jth bit in the codeword. The ith CN update using SOML decoding can be written as

$$Q_{i,j}^{(l+1)} = \log \frac{\sum\limits_{\boldsymbol{b}_i : z_{i,j}=0} \left(\prod\limits_{p=1, p\neq j}^{n_{vi}} e^{-P_{i,p}^{(l)} \cdot z_{i,p}} \prod\limits_{p=1}^{k_{vi}} e^{-Y_{i,p} \cdot b_{i,p}} \right)}{\sum\limits_{\boldsymbol{b}_i : z_{i,j}=1} \left(\prod\limits_{p=1, p\neq j}^{n_{vi}} e^{-P_{i,p}^{(l)} \cdot z_{i,p}} \prod\limits_{p=1}^{k_{vi}} e^{-Y_{i,p} \cdot b_{i,p}} \right)} \tag{2.25}$$

At the beginning of iteration, $\boldsymbol{P} = \boldsymbol{0}$, therefore the iterative decoding is initialized as

$$Q_{i,j}^{(1)} = \log \frac{\sum\limits_{\boldsymbol{b}_i : z_{i,j}=0} \left(\prod\limits_{p=1}^{k_{vi}} e^{-Y_{i,p} \cdot b_{i,p}} \right)}{\sum\limits_{\boldsymbol{b}_i : z_{i,j}=1} \left(\prod\limits_{p=1}^{k_{vi}} e^{-Y_{i,p} \cdot b_{i,p}} \right)} \tag{2.26}$$

The SOML estimation $\boldsymbol{W}^{(l)} = \{\boldsymbol{W}_1^{(l)}, \boldsymbol{W}_2^{(l)}, ...\boldsymbol{W}_{N_V}^{(l)}\}$ for $\boldsymbol{b}$, where $\boldsymbol{W}_i^{(l)} = \{W_{i,1}^{(l)}, W_{i,2}^{(l)}, ...W_{i,k_{vi}}^{(l)}\}$. $W_{i,j}^{(l)}$, after lth iteration, is

$$W_{i,j}^{(l)} = \log \frac{\sum\limits_{\boldsymbol{b}_i : b_{i,j}=0} \left(\prod\limits_{p=1}^{n_{vi}} e^{-P_{i,p}^{(l)} \cdot z_{i,p}} \prod\limits_{p=1}^{k_{vi}} e^{-Y_{i,p} \cdot b_{i,p}} \right)}{\sum\limits_{\boldsymbol{b}_i : b_{i,j}=1} \left(\prod\limits_{p=1}^{n_{vi}} e^{-P_{i,p}^{(l)} \cdot z_{i,p}} \prod\limits_{p=1}^{k_{vi}} e^{-Y_{i,p} \cdot b_{i,p}} \right)} \tag{2.27}$$

$\boldsymbol{W}^{(l)}$ can be obtained by arranging for $W_{i,j}^{(l)}$, where $i = 1, 2, ...N_V$, $j = 1, 2...k_{vi}$.

Based on (2.24), (2.25), (2.26) and (2.27), the iterative decoding of GLDPC/DGLDPC codes can be formulated as follows. The maximum number of iteration is L.

Iterative decoding of GLDPC/DGLDPC codes

1. **Initialization**: $l = 1$, initialize $Q_{i,j}^{(l)}$ using (2.26). Form $\boldsymbol{Q}^{(l)}$ based on $Q_{i,j}^{(l)}$, where $i = 1, 2..., N_V,\ j = 1, 2, ..., n_{vi}$. And find $\boldsymbol{S}^{(l)} = \mathbf{\Pi}(\boldsymbol{Q}^{(l)})$.

2. **Horizontal Step**: update $R_{i,j}^{(l)}$ using (2.24). Form $\boldsymbol{R}^{(l)}$ based on $R_{i,j}^{(l)}$, where $i = 1, 2..., N_C,\ j = 1, 2, ..., n_{ci}$. And find $\boldsymbol{P}^{(l)} = \mathbf{\Pi}^{-1}(\boldsymbol{R}^{(l)})$.

3. **Vertical Step**: update $Q_{i,j}^{(l+1)}$ using (2.25). Form $\boldsymbol{Q}^{(l+1)}$ based on $Q_{i,j}^{(l+1)}$, where $i = 1, 2..., N_V,\ j = 1, 2, ..., n_{vi}$. And find $\boldsymbol{S}^{(l+1)} = \mathbf{\Pi}(\boldsymbol{Q}^{(l+1)})$.

4. **Decision and output**: calculate APP $W_{i,j}^{(l)}$ using (2.27). Form $\boldsymbol{W}^{(l)}$ based on $W_{i,j}^{(l)}$, where $i = 1, 2..., N_V,\ j = 1, 2, ..., k_{vi}$. Decide $\widehat{b}_{i,j} = 0$ if $W_{i,j}^{(l)} \geq 0$, else $\widehat{b}_{i,j} = 1$. If $\widehat{\boldsymbol{b}}$ formed by $\widehat{b}_{i,j}$ does not satisfy $\widehat{\boldsymbol{b}} \cdot H_{SPC}^{T} = \mathbf{0}$ and $l < L$, then $l = l + 1$ and go to step 2, otherwise stop and output $\widehat{\boldsymbol{b}}$ as the decoding result.

It should be noted that (2.17), (2.18), (2.21) and (2.19) are special forms of (2.26), (2.25), (2.24) and (2.27), respectively. Therefore for REP-VNs and SPC-CNs in GLDPC/DGLDPC codes, the update rule can be simplified using (2.17), (2.18), (2.21) and (2.19). The trellis-based SOML decoding algorithms such as BCJR [75] and one-sweep algorithm [76] can be directly used on solving (2.24), the decoding complexity is $O(\min(2^k, 2^{(n-k)}))$ for a $C(n, k)$ SCN, whereas for some special codes such as Hamming codes and 1st-order Reed-Muller codes, fast SOML decoding algorithms exist [77]. In practice, some low-complexity (of course inferior to SOML) soft-output decoding algorithms such as Chase decoding [78] and generalized minimum distance (GMD) decoding [79] can also be used. In addition it is worth noting that, if the SVN adopts a systematic form of G_{comp}, the decoding complexity of SVN can be reduced to that of SCN.

The decoding complexities in SVNs and SCNs imply that small component codes or high-rate codes are preferred. In many cases such as hGLDPC and DGLDPC codes, the complexity increase is modest since there is only a small percentage of edges connected to SCNs and SVNs. With the development of technology, the decoding complexity of GLDPC and DGLDPC codes will become more and more affordable for most practical uses.

2.3 Analysis and optimizing tools for code ensembles

Given blocklength N and a certain $(\boldsymbol{\lambda}, \boldsymbol{\rho})$, a collection of codes (LDPC, GLDPC and DGLDPC) can be obtained from randomly permuting edges connecting VNs and CNs under the constraint of $(\boldsymbol{\lambda}, \boldsymbol{\rho})$. This collection of codes is called an *ensemble*, denoted by $\mathbf{C}^N(\boldsymbol{\lambda}, \boldsymbol{\rho})$. When $N \to \infty$, the notation $\mathbf{C}^\infty(\boldsymbol{\lambda}, \boldsymbol{\rho})$ is used.

The ensemble $\mathbf{C}^\infty(\boldsymbol{\lambda}, \boldsymbol{\rho})$, or *profile*, is usually studied instead of individual LDPC codes. This is because the following properties of the ensembles: the *concentration property* [13] states that with $N \to \infty$, the individual codes converge in probability exponentially to the average performance of $\mathbf{C}^N(\boldsymbol{\lambda}, \boldsymbol{\rho})$. In addition the *convergence property* [13] states that with $N \to \infty$ the average performance of $\mathbf{C}^N(\boldsymbol{\lambda}, \boldsymbol{\rho})$ converges in probability to the performance of cycle-free $\mathbf{C}^\infty(\boldsymbol{\lambda}, \boldsymbol{\rho})$ case.

Let $P_e^\infty(l)$ be the average BER of $\mathbf{C}^\infty(\boldsymbol{\lambda}, \boldsymbol{\rho})$ after lth iterations. It is proved in [13] that given a certain $(\boldsymbol{\lambda}, \boldsymbol{\rho})$, there exists a decoding threshold $\sigma^\star = \sigma^\star(\mathbf{C}^\infty(\boldsymbol{\lambda}, \boldsymbol{\rho}))$, such that:

$$\begin{aligned} \lim_{l\to\infty} P_e^\infty(l) &= 0 \quad \text{if } \sigma < \sigma^\star \\ \lim_{l\to\infty} P_e^\infty(l) &= \eta(\sigma) > 0 \quad \text{if } \sigma > \sigma^\star \end{aligned} \tag{2.28}$$

where $\eta(\cdot)$ is a positive real number, which is a function of σ. In general, σ and $\sigma^\star$ can be regarded as some noise level in the channel, such as the additive white Gaussian noise in AWGN channels or erasure probability ϵ in BEC.

(2.28) states that if channel condition is better than $\sigma^\star$, a code in $\mathbf{C}^\infty(\boldsymbol{\lambda}, \boldsymbol{\rho})$ can achieve error-free decoding performance, otherwise the BER is always some positive value greater than 0. The decoding threshold of a profile important in the sense that it is not only a critical parameter in infinite blocklength case, but also a good indicator on the waterfall performance of a code in FL ensemble $\mathbf{C}^N(\boldsymbol{\lambda}, \boldsymbol{\rho})$. In the following, we introduce several tools for determining and optimizing code ensembles.

2.3.1 Density evolution

Density evolution [13] is an effective and accurate tool for determining the threshold $\sigma^\star$ of a given LDPC profile with the degree distribution $(\boldsymbol{\lambda} \doteq (\lambda_2, \lambda_3, ..., \lambda_{dvmax}), \boldsymbol{\rho} \doteq (\rho_2, \rho_3, ..., \rho_{dcmax}))$. For GLDPC and DGLDPC ensembles, density evolution on GLDPC and DGLDPC is too complicated to implement since the efficient density evolution on SVNs and SCNs is still lacking. In practice EXIT charts are used instead to evaluate the $\sigma^\star$ of GLDPC/DGLDPC ensembles, which will be discussed in the subsection 2.3.2.

Suppose the channel satisfies *the symmetric condition* [14], i.e. $P(y|1) = P(-y|-1)$, then during the iteration, the probability density function (PDF) of RVs sent from VNs and CNs also satisfies the symmetric condition under the BP algorithm. Based on above properties we can hold the *all-one codeword assumption* that the conditioned BER is independent of the specific codeword transmitted so an all-one codeword can be used [14].

Let P_0 be the PDF of channel LLR, and $P_i^{(l)}$ be the PDF of extrinsic message RV sent from VN with degree $dv = i$ in the lth iteration. Also define $Q_j^{(l)}$ the PDF of extrinsic message RV sent from CN with degree $dc = j$ in the lth iteration. The *average PDF* sent from VNs $P^{(l)}$, and from CNs $Q^{(l)}$, are calculated as

$$\begin{aligned} P^{(l)} &= \sum_{i=2}^{dvmax} \lambda_i \cdot P_i^{(l)} \\ Q^{(l)} &= \sum_{j=2}^{dcmax} \rho_j \cdot Q_j^{(l)} \end{aligned} \tag{2.29}$$

The incoming messages are assumed i.i.d. with PDF $Q^{(l)}$, since the blocklength is infinite. According to (2.18), $P_i^{(l)}$ can be calculated as

$$P_i^{(l)} = P_0 \circledast \overbrace{Q^{(l-1)} \circledast Q^{(l-1)} \circledast ... \circledast Q^{(l-1)}}^{(i-1)} \tag{2.30}$$

where $\circledast$ denotes the ordinary convolution, and $P_i^{(1)} = P_0$.

For calculating $Q_i^{(l)}$, one has to convert $P^{(l)}$ into a function $\Gamma(\cdot)$ on $GF(2) \times [0, +\infty]$ [14], where $GF(2)$ denotes the sign part and $[0, +\infty]$ is the magnitude part, then apply convolution $(i-1)$

times on $GF(2) \times [0, +\infty]$, then use $\Gamma^{-1}(\cdot)$ to obtain $Q_i^{(l)}$. For more details, readers are referred to [14]; on the other hand, the calculation of $P_i^{(l)}$ and $Q_i^{(l)}$ can also be done by look-up table (LUT) instead of convolution operations.

The density evolution updates $P^{(l)}$ and $Q^{(l)}$ alternatingly, and after a sufficiently large number of iterations, say L, the evolved PDF will indicate if $\mathbf{C}^{\infty}(\boldsymbol{\lambda}, \boldsymbol{\rho})$ can converge or not under a given σ: if $P^{(L)}$ has a distribution that no probability mass can be found at $X \leq 0$, $\mathbf{C}^{\infty}(\boldsymbol{\lambda}, \boldsymbol{\rho})$ can converge under σ, otherwise it cannot.

As an example, Table 2.1 [14] lists the code rate, $\sigma^{\star}$'s of several regular LDPC profiles, and the corresponding maximum σ_C prescribed by the capacity of binary-input AWGN (BIAWGN) channel, the equivalent SNRs (in dB) of thresholds and capacity are also given. It can be seen that all the thresholds have some gaps from the corresponding capacity, and they all have fairly good decoding thresholds. Among the rate-0.5 regular profiles, the $(3, 6)$ profile has the best threshold, which is 0.923 dB away from channel capacity.

Table 2.1: The code rates and $\sigma^{\star}$'s of regular LDPC profiles, and the corresponding maximum σ_C prescribed by the capacity of BIAWGN channel.

Regular LDPC profile	code rate	$\sigma^{\star}$	$(E_b/N_0)_{\star}$ (dB)	σ_C	$(E_b/N_0)_C$ (dB)
(3,6)	0.5	0.88	1.110	0.979	0.187
(4,8)	0.5	0.83	1.618	0.979	0.187
(5,10)	0.5	0.79	2.047	0.979	0.187
(3,5)	0.4	1.0	0.969	1.148	-0.236
(4,6)	0.333	1.01	1.674	1.295	-0.507
(3,4)	0.25	1.26	1.003	1.549	-0.793

2.3.2 EXIT charts

The EXIT chart is another efficient tool for evaluating $\sigma^{\star}$ of a given $\mathbf{C}^{\infty}(\boldsymbol{\lambda}, \boldsymbol{\rho})$. In density evolution the system calculates the averaged PDF of extrinsic messages, which can be very complicated in terms of both calculation and storage. As an accurate and robust statistic, *mutual information* is used and tracked in EXIT chart during the decoding iterations. Compared with density evolution, EXIT charts [67–71] have the advantages of low computational complexity and straightforward visual effect. For a lucid illustration, we give the decoding model with two encoders below, which is based on [69]:

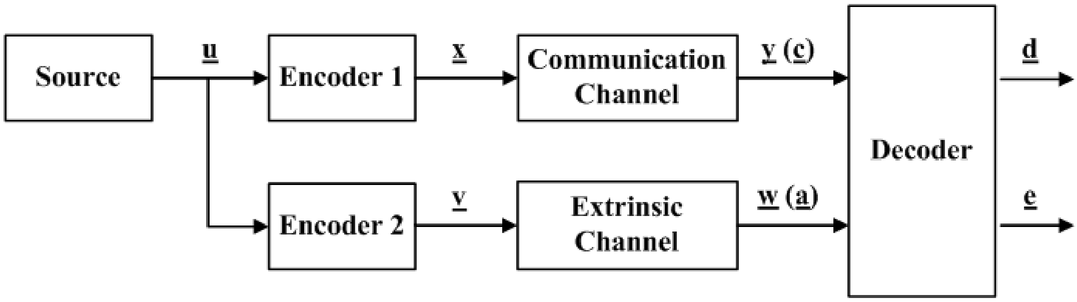

Figure 2.7: A decoding model with two encoders for EXIT analysis.

In Fig. 2.7, the information sequence $\underline{\mathbf{u}}$ of length k is encoded into $\underline{\mathbf{x}}$ of length m and $\underline{\mathbf{v}}$ of length n, therefore it is a $(m+n,k)$ code. $\underline{\mathbf{x}}$ and $\underline{\mathbf{v}}$ are transmitted over the communication and extrinsic channel, respectively; the decoder receives $\underline{\mathbf{y}}$ ($\underline{\mathbf{c}}$ in LLR form) and $\underline{\mathbf{w}}$ ($\underline{\mathbf{a}}$ in LLR form). The decoder implements soft-output maximum-*a-posteriori* (SOMAP) based on $\underline{\mathbf{y}}$ and $\underline{\mathbf{w}}$, and gives soft output $\underline{\mathbf{d}}$ and $\underline{\mathbf{e}}$, where $\underline{\mathbf{d}}$ is the APP of $\underline{\mathbf{v}}$.

We now demonstrate that Fig. 2.7 is a generic model for the components codes of many iterative-decodable codes. In the following, an uppercase letter denotes the RV, and lowercase letter the realization of such RV. Let w_i be the observed value in ith position in $\underline{\mathbf{w}}$, and similarly define y_i, a_i, d_i and c_i. $\underline{\mathbf{v}}_{[i]}$ denotes the vector $\underline{\mathbf{v}}$ with the ith entry removed.

The SOMAP decoder computes d_i (in LLR form)

$$d_i = \log \frac{P(V_i = 0|\underline{\mathbf{y}}, \underline{\mathbf{w}})}{P(V_i = 1|\underline{\mathbf{y}}, \underline{\mathbf{w}})} \tag{2.31}$$

where

$$\begin{aligned}
P(V_i = 0|\underline{\mathbf{y}}, \underline{\mathbf{w}}) &= \sum_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=0} P(\underline{\mathbf{u}}|\underline{\mathbf{y}}, \underline{\mathbf{w}}) \\
&= \sum_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=0} \frac{P(\underline{\mathbf{u}})P(\underline{\mathbf{w}}|\underline{\mathbf{u}})P(\underline{\mathbf{y}}|\underline{\mathbf{u}}, \underline{\mathbf{w}})}{P(\underline{\mathbf{y}}, \underline{\mathbf{w}})} \\
&= \sum_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=0} \frac{P(\underline{\mathbf{u}})P(\underline{\mathbf{w}}|\underline{\mathbf{v}}(\underline{\mathbf{u}}))P(\underline{\mathbf{y}}|\underline{\mathbf{x}}(\underline{\mathbf{u}}))}{P(\underline{\mathbf{y}}, \underline{\mathbf{w}})} \\
&= \frac{P(w_i|V_i = 0)}{P(\underline{\mathbf{y}}, \underline{\mathbf{w}})} \left(\sum_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=0} P(\underline{\mathbf{u}})P(\underline{\mathbf{w}}_{[i]}|\underline{\mathbf{v}}_{[i]}(\underline{\mathbf{u}}))P(\underline{\mathbf{y}}|\underline{\mathbf{x}}(\underline{\mathbf{u}})) \right)
\end{aligned} \tag{2.32}$$

The expression for $P(V_i = 1|\underline{\mathbf{y}}, \underline{\mathbf{w}})$ can be similarly obtained as

$$P(V_i = 1|\underline{\mathbf{y}}, \underline{\mathbf{w}}) = \frac{P(w_i|V_i = 1)}{P(\underline{\mathbf{y}}, \underline{\mathbf{w}})} \left(\sum_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=1} P(\underline{\mathbf{u}}) P(\underline{\mathbf{w}}_{[i]}|\underline{\mathbf{v}}_{[i]}(\underline{\mathbf{u}})) P(\underline{\mathbf{y}}|\underline{\mathbf{x}}(\underline{\mathbf{u}})) \right) \tag{2.33}$$

Substitute (2.32) and (2.33) into (2.31), notice that

$$a_i = \log \frac{P(w_i|V_i = 0)}{P(w_i|V_i = 1)} \tag{2.34}$$

and

$$\begin{aligned} e_i &= \log \frac{P(V_i = 0|\underline{\mathbf{y}}, \underline{\mathbf{w}}_{[i]})}{P(V_i = 1|\underline{\mathbf{y}}, \underline{\mathbf{w}}_{[i]})} \\ &= \log \frac{\sum\limits_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=0} P(\underline{\mathbf{u}}) P(\underline{\mathbf{w}}_{[i]}|\underline{\mathbf{v}}_{[i]}(\underline{\mathbf{u}})) P(\underline{\mathbf{y}}|\underline{\mathbf{x}}(\underline{\mathbf{u}}))}{\sum\limits_{\underline{\mathbf{u}}:v_i(\underline{\mathbf{u}})=1} P(\underline{\mathbf{u}}) P(\underline{\mathbf{w}}_{[i]}|\underline{\mathbf{v}}_{[i]}(\underline{\mathbf{u}})) P(\underline{\mathbf{y}}|\underline{\mathbf{x}}(\underline{\mathbf{u}}))} \end{aligned} \tag{2.35}$$

Finally the d_i can be written as the sum of *extrinsic* message e_i and *a-priori* message a_i.

$$d_i = a_i + e_i \tag{2.36}$$

We give several examples for the model in Fig. 2.7. Consider Turbo codes using parallel-concatenated-convolutional-code (PCCC) structure; then both component codes have extrinsic and communication channels, and $\underline{\mathbf{u}}$=$\underline{\mathbf{x}}$. For LDPC, GLDPC, and DGLDPC codes, there is no communication channel for CNs, and a CN with degree dc receives $\underline{\mathbf{w}}$ of length dc from the extrinsic channel. For REP-VN, $\underline{\mathbf{u}}$=$\underline{\mathbf{x}}$ with $k = 1$, while for a $C(n_v, k_v)$-SVN, $\underline{\mathbf{u}}$=$\underline{\mathbf{x}}$ with length k_v, and it receives $\underline{\mathbf{w}}$ of length n_v from the extrinsic channel. The final remark is that (2.35) is the generic expression for calculating extrinsic messages, which encompasses all previous node update rules such as (2.18), (2.21), (2.25), (2.24).

The averaged *a-priori* mutual information I_A and the averaged *extrinsic* mutual information I_E are defined as:

$$\begin{aligned} I_A &\doteq \frac{1}{n} \sum_{i=1}^{n} I(V_i; A_i) \\ I_E &\doteq \frac{1}{n} \sum_{i=1}^{n} I(V_i; E_i) \end{aligned} \tag{2.37}$$

A *EXIT curve* can be obtained by plotting I_E as a function of I_A, note that in binary case $0 \leq I_E \leq 1$ and $0 \leq I_A \leq 1$.

EXIT curves for LDPC codes

For a BEC with erasure probability q, suppose a REP-VN with degree dv receives the extrinsic information $I_A = 1 - p$. Then I_E is

$$I_E^{(dv)} = 1 - q(1 - I_A)^{dv-1} \tag{2.38}$$

whereas for a SPC code with degree dc receiving $I_A = 1 - p$, since there is no communication channel, I_E is

$$I_E^{(dc)} = (I_A)^{dc-1} \tag{2.39}$$

For the AWGN channel, the EXIT curve is calculated based on the fact that the PDF of extrinsic message (in LLR form) can be approximated by a Gaussian RV which satisfies the consistency condition [72]. Let $X \in \{\pm 1\}$ be the modulated random bit, and Υ be the PDF of extrinsic message, then the mutual information $I(X; \Upsilon)$ can be approximated by a Gaussian RV $\mathcal{N}(0, \sigma^2)$ such that σ satisfies

$$I(X; \Upsilon) = J(\sigma) = 1 - \int_{-\infty}^{\infty} \frac{e^{-\frac{\left(\xi - \sigma^2/2\right)^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \cdot \log_2\left(1 + e^{-\xi}\right) d\xi \tag{2.40}$$

Based on the "sum" update rule of REP-VN, the $I_E^{(dv)}$ of a REP-VN with degree dv can be calculated as

$$I_E^{(dv)} = J\left(\sqrt{(dv-1) \cdot \left[J^{-1}\left(I_A^{(dv)}\right)\right]^2 + 8R \cdot \frac{E_b}{N_0}}\right) \tag{2.41}$$

where $8R \cdot \frac{E_b}{N_0}$ is sometimes also denoted by σ_{ch}^2, which is the variance of channel LLR. Based on the duality property [70], the SPC-CN with degree dc, $I_E^{(dc)}$ is

$$I_E^{(dc)} \approx 1 - J\left(\sqrt{(dc-1)} \cdot J^{-1}\left(1 - I_A\right)\right) \tag{2.42}$$

In LDPC codes, the I_A and I_E of VNs are commonly written as $I_{A,V}$ and $I_{E,V}$, and for CNs they are $I_{A,C}$ and $I_{E,C}$. For a VN with degree dv, we write $I_{E,V}^{(dv)}$ as $I_{E,V}^{(dv)} = I_{E,V}^{(dv)}(dv, \frac{E_b}{N_0}, R, I_{A,V})$.

Similarly for a CN with degree dc, $I_{E,C}^{(dc)} = I_{E,C}^{(dc)}(dc, I_{A,C})$. Finally the averaged EXIT curves of VNs and CNs can be obtained as

$$\begin{aligned} I_{E,V} &= \sum_{i=2}^{dvmax} \lambda_i \cdot I_{E,V}^{(i)}(i, \frac{E_b}{N_0}, R, I_{A,V}) \\ I_{E,C} &= \sum_{j=2}^{dcmax} \rho_j \cdot I_{E,C}^{(j)}(j, I_{A,C}) \end{aligned} \tag{2.43}$$

EXIT charts

In bipartite-graph based codes, we have $I_{A,V} = I_{E,C}$ and $I_{E,V} = I_{A,C}$. The *EXIT chart* can be obtained by plotting $I_{E,V}(I_{A,V})$ and $I_{E,C}(I_{A,C})$ in the same graph. We usually plot the inverse function of $I_{E,C}(I_{A,C})$ (i.e. $I_{A,C}(I_{E,C})$) to make EXIT chart more informative: if under a certain σ (equivalently, SNR), $I_{E,V}(I_{A,V}) > I_{A,C}(I_{E,C}) \quad \forall I_{A,V} \in [0,1)$, the EXIT chart is said to have an "open tunnel", which means for an infinite blocklength code we can achieve successful decoding under σ, since $I_{E,V}$ can achieve $I_{E,V} = 1$ after a sufficient number of iterations; otherwise, if $I_{E,V}(I_{A,V})$ and $I_{A,C}(I_{E,C})$ first intersect at $x \in [0,1)$, it means the decoder will get stuck at x and fail in decoding. Given a degree profile $(\boldsymbol{\lambda}, \boldsymbol{\rho})$, an EXIT chart can efficiently and accurately determine if it can converge or not under a certain σ (or SNR). In Fig. 2.8 the EXIT chart of a $(3,6)$ regular LDPC ensemble on AWGN channel is shown: at $\frac{E_b}{N_0} = 1.5$ dB, the VN curve is always above the CN one so the code ensemble can converge at $\frac{E_b}{N_0} = 1.5$ dB; whereas at $\frac{E_b}{N_0} = 0.5$ dB two curves first intersect at the point around (0.11,0.61), which means the decoding will fail at $\frac{E_b}{N_0} = 0.5$ dB.

EXIT curves for SCNs and SVNs

EXIT curves on BEC: Fig. 2.7 is still a valid model for analysis of SVNs/SCNs in GLDPC and DGLDPC codes. The EXIT chart under BEC is analyzed first: for SVN using $C(n,k)$ as its component code and using G_{comp} as the generator matrix of $C(n,k)$, let I_k be the $(k \times k)$ identity matrix, the $\widetilde{e}_{g,h}$ is called the (g,h)*-th un-normalized split information function*, which can be calculated via the summation over the rank of all the possible submatrices obtained by selecting g $(0 \le g \le n)$ columns in G_{comp} and h $(0 \le h \le k)$ columns in I_k. Suppose the channel has the erasure probability q and $I_{A,V} = 1 - p$, then $I_{E,V}$ can be calculated as [69]:

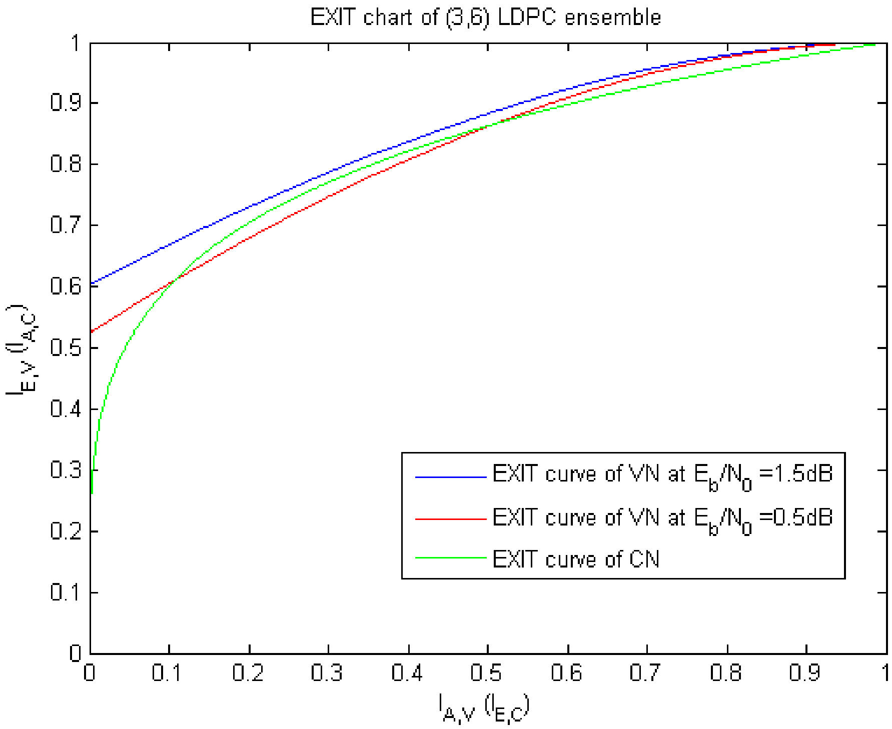

Figure 2.8: The EXIT chart of the $(3, 6)$ LDPC ensemble under different SNR's.

$$I_{E,V}(p,q) = 1 - \frac{1}{n}\sum_{h=0}^{k}(1-q)^h q^{k-h}\sum_{g=1}^{n}(1-p)^{g-1}p^{n-g}\cdot[g\cdot\widetilde{e}_{g,h} - (n-g+1)\cdot\widetilde{e}_{g-1,h}] \tag{2.44}$$

whereas for SCN of $C(n, k)$, we denote by $\widetilde{e}_g$ of $C(n, k)$ the *g-th un-normalized information function*, which can be calculated via the summation over the rank of all the possible submatrices obtained by taking $g, 0 \leq g \leq n$ columns in G_{comp}. Let $I_{A,C} = 1 - p$, $I_{E,C}$ can be calculated as:

$$I_{E,C}(p) = 1 - \frac{1}{n}\sum_{g=1}^{n}(1-p)^{g-1}p^{n-g}\cdot[g\cdot\widetilde{e}_g - (n-g+1)\cdot\widetilde{e}_{g-1}] \tag{2.45}$$

(2.45) can also be obtained by letting $q = 1$ in (2.44). In order to calculate $\widetilde{e}_{g,h}$ and $\widetilde{e}_h$ of a certain code C, a brute-force method is commonly used. For instance, the $\{\widetilde{e}_g\}$ for SCN using a Hamming$(15, 11)$ code is

$$\{\widetilde{e}_g\} = \{0, 15, 210, 1365, 5460, 15015, 30030, 45045, 51465, 44940, 29715, 14490, 4970, 1155, 165, 11\}$$

As an example for SVN node, Table 2.2 shows the $\{\widetilde{e}_{g,h}\}$ of a SPC$(6,5)$ SVN node using systematic G_{comp}

$$G_{comp} = \begin{bmatrix} 1&0&0&0&0&1 \\ 0&1&0&0&0&1 \\ 0&0&1&0&0&1 \\ 0&0&0&1&0&1 \\ 0&0&0&0&1&1 \end{bmatrix} \tag{2.46}$$

Table 2.2: The $\{\widetilde{e}_{g,h}\}$ of a SPC$(6,5)$ SVN node using systematic G_{comp}.

$\widetilde{e}_{g,h}$	$h=0$	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$
$g=0$	0	5	20	30	20	5
$g=1$	6	55	160	210	130	30
$g=2$	30	200	500	600	345	75
$g=3$	60	350	800	890	480	100
$g=4$	60	325	690	720	370	75
$g=5$	30	150	300	300	150	30
$g=6$	5	25	50	50	25	5

It should be noted that if G_{comp} adopts another form, $\{\widetilde{e}_{g,h}\}$ will be different, resulting in *a different EXIT curve*, which is not the case with SCN (i.e. $\{\widetilde{e}_g\}$ is not changing with different G_{comp}). If the SPC$(6,5)$ uses the so-called "cyclic" G_{comp} where

$$G_{comp} = \begin{bmatrix} 1&1&0&0&0&0 \\ 0&1&1&0&0&0 \\ 0&0&1&1&0&0 \\ 0&0&0&1&1&0 \\ 0&0&0&0&1&1 \end{bmatrix} \tag{2.47}$$

and Table 2.3 shows the $\{\widetilde{e}_{g,h}\}$ of a SPC$(6,5)$ SVN node using cyclic G_{comp}. The examples above indicate that for DGLDPC codes, G_{comp} constitutes another degree of freedom in optimizing the ensemble. When describing a DGLDPC profile, it is not enough to give the SVN type, but G_{comp} is also needed.

Table 2.3: The $\{\widetilde{e}_{g,h}\}$ of a SPC$(6,5)$ SVN node using cyclic G_{comp}.

$\widetilde{e}_{g,h}$	$h=0$	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$
$g=0$	0	5	20	30	20	5
$g=1$	6	58	168	216	130	30
$g=2$	30	213	531	618	345	75
$g=3$	60	370	838	908	480	100
$g=4$	60	335	705	726	370	75
$g=5$	30	150	300	300	150	30
$g=6$	5	25	50	50	25	5

In [46], Paolini *et al* proposed a recursive algorithm for calculating the *averaged EXIT function* of an expurgated ensemble $\mathcal{G}^{\star}(n,k)$ for SVN and SCN, where $\mathcal{G}^{\star}(n,k) = \{C(n,k) |\ C(n,k)$ has a $d_{min} \geq$

2}. The EXIT curves of $\mathcal{G}^\star(n,k)$ adds more flexibility when designing the profile: we can use the EXIT charts analyzing for the ensemble $\mathcal{E}$ using component codes randomly picked from $\mathcal{G}^\star(n,k)$, which is an expanded ensemble from the traditional ensemble using fixed node types. As an example, the $\{\widetilde{e}_{g,h}\}$ of $\mathcal{G}^\star(6,5)$ is shown in Table 2.4.

Table 2.4: The $\{\widetilde{e}_{g,h}\}$ of $\mathcal{G}^\star_{comp}(6,5)$.

$\widetilde{e}_{g,h}$	$h=0$	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$
$g=0$	0	5	20	30	20	5
$g=1$	6	59.0323	174.1935	226.4516	135.4839	30
$g=2$	30	217.7419	557.4194	655.1613	358.0645	75
$g=3$	60	377.4194	873.5484	943.2258	490.3226	100
$g=4$	60	338.7097	716.1290	735.4839	372.5806	75
$g=5$	30	150	300	300	150	30
$g=6$	5	25	50	50	25	5

Fig. 2.9 gives three SVN EXIT curves at channel erasure probability $\epsilon = 0.65$: $(6,5)$ SPC code using systematic G_{comp}, $(6,5)$ SPC code using cyclic G_{comp} and $\mathcal{G}^\star(6,5)$. Note the obvious dependence of EXIT curves on the specific forms of G_{comp}'s.

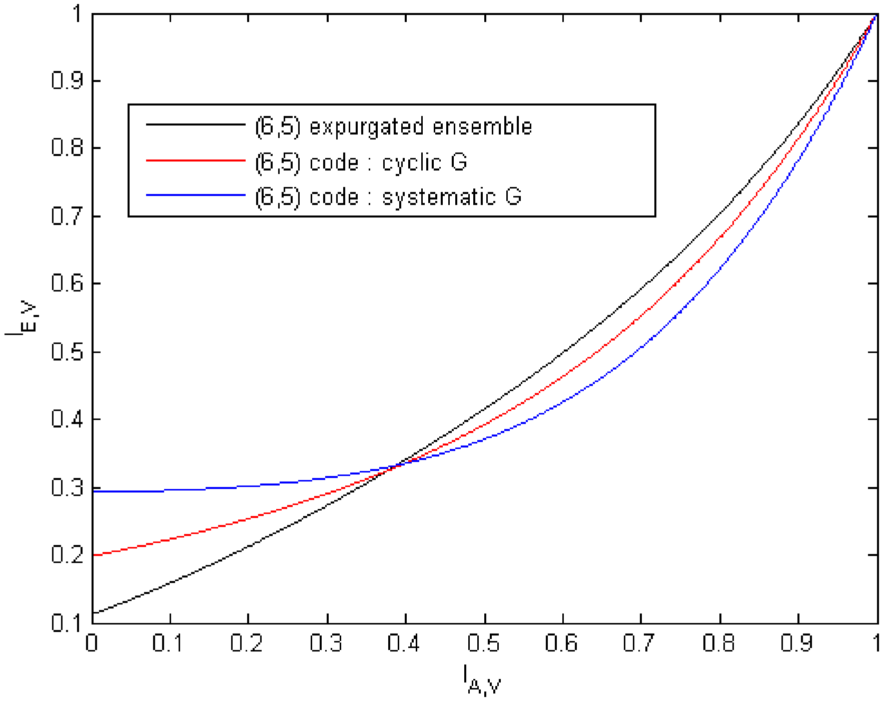

Figure 2.9: SVN EXIT curves of (6,5) SPC codes of various forms at channel erasure probability $\epsilon = 0.65$.

EXIT curves on AWGN channels: For AWGN channel, it is generally impossible to have a closed-form expression for $I_{E,C}$ and $I_{E,V}$ of code $C(n,k)$, except for some very simple codes [44], so Monte-Carlo simulations are frequently used. Fig. 2.10 depicts the SCN EXIT curves of several linear codes on AWGN channel obtained by Monte-Carlo simulation. Also in Fig. 2.11 the SVN EXIT curves of several linear codes on AWGN channel at channel-symbol-to-noise ratio $\frac{E_s}{N_0} = 0$ dB are drawn.

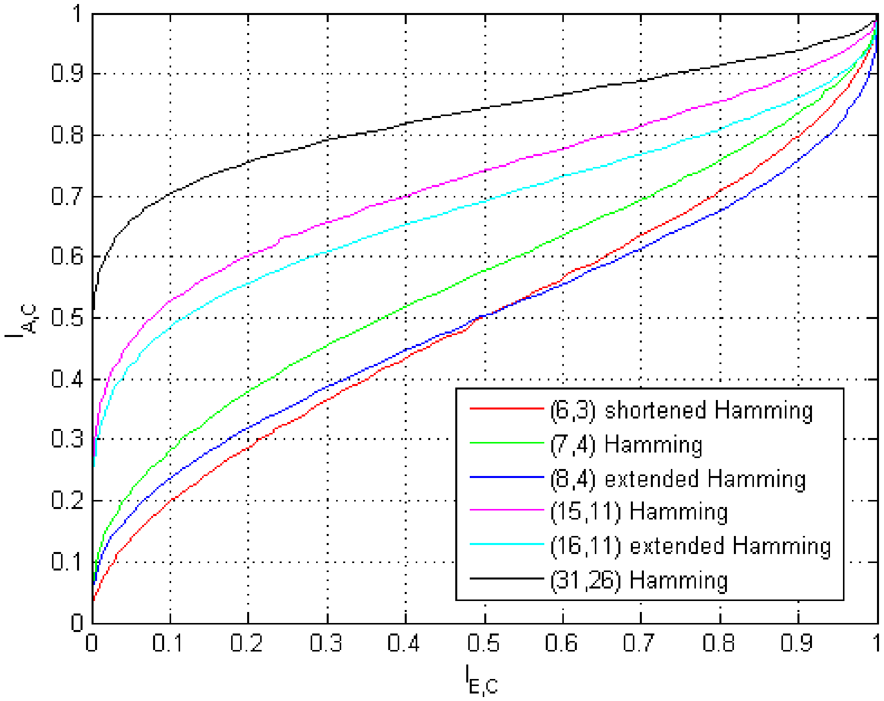

Figure 2.10: EXIT curves of several SCNs on AWGN channel.

Besides the Monte-Carlo approach, the EXIT function of SCN and SVN on AWGN channel can also be approximated by an infinite series, where each term depends on the EXIT function for a BEC [71]. For SCNs employing a *high-rate* $C(n,k)$,

$$I_E^{AWGN}\left(\frac{E_b}{N_0}\right) \cong \frac{1}{\ln 2}\sum_{i=1}^{\infty}\frac{1}{(2i-1)(2i)} I_E^{BEC}(\varepsilon_i) \tag{2.48}$$

where

$$\varepsilon_i = 1 - \Phi_i\left(\frac{\left[J^{-1}(I_A)\right]^2}{2}\right)$$

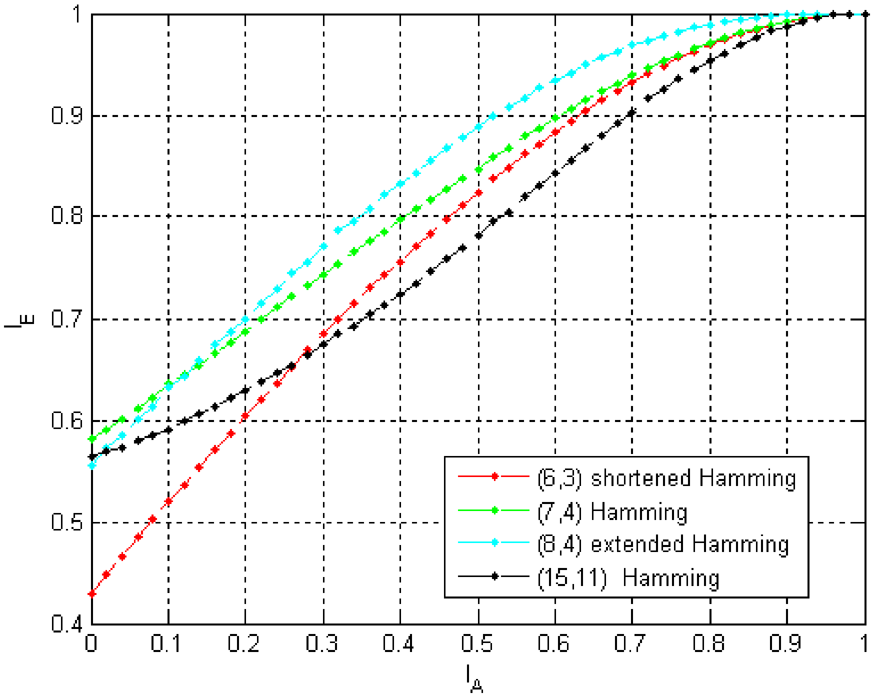

Figure 2.11: SVN EXIT curves of several linear codes on AWGN channel at $\frac{E_s}{N_0} = 0$ dB.

and

$$\Phi_i(m) = \int_{-1}^{+1} \frac{2t^{2i}}{(1-t^2)\sqrt{4\pi m}} e^{-\frac{\left((\ln\frac{1+t}{1-t})-m\right)^2}{4m}} dt$$

For SCNs employing a *low-rate* code, according to the duality property, it can be approximated by [71]

$$I_E^{AWGN}\left(\frac{E_b}{N_0}\right) \cong 1 - \frac{1}{\ln 2}\sum_{i=1}^{\infty} \frac{1}{(2i-1)(2i)} I_E^{\perp,BEC}(\varepsilon_i) \tag{2.49}$$

where

$$\varepsilon_i = 1 - \Phi_i\left(\frac{\left[J^{-1}(1 - J\left(\sqrt{8R\cdot\frac{E_b}{N_0}}\right)\right)\right]^2}{2}\right)$$

For SVNs, the EXIT curve can be approximated as [50]

$$I_E^{AWGN}\left(I_A, \frac{E_b}{N_0}, R\right) \cong 1 - \frac{1}{\ln 2}\sum_{i=1}^{\infty} \frac{1}{(2i-1)(2i)} I_E^{\perp,BEC}(\varepsilon_i, \eta_i) \tag{2.50}$$

where $\varepsilon_i = 1 - \Phi_i\left(\frac{1}{2}\left[J^{-1}(1-I_A)\right]^2\right)$ and $\eta_i = 1 - \Phi_i\left(\frac{1}{2}\left[J^{-1}\left(1 - J\left(\sqrt{8RE_b/N_0}\right)\right)\right]^2\right)$.

It should be noted that for moderate rate component codes, the infinite series approximation ((2.48), (2.49) and (2.50)) is not as accurate as their high-rate/low-rate counterparts. In this case Monte-Carlo simulation seems to be the best solution.

EXIT charts for GLDPC/DGLDPC codes: For GLDPC/DGLDPC codes, the averaged EXIT curves of VN and CN can then be obtained after we have EXIT curves for each component code:

$$\begin{aligned} I_{E,V} &= \sum_{i \in \mathcal{I}_V} \lambda_i \cdot I_{E,V}^{(i)} \\ I_{E,C} &= \sum_{j \in \mathcal{I}_C} \rho_j \cdot I_{E,C}^{(j)} \end{aligned} \tag{2.51}$$

where $I_{E,V}^{(i)}(I_{E,C}^{(j)})$ is the EXIT function of type-i(type-j) component code in VN(CN). Note that the rationale for doing the averaging is the random permutation of edges between VNs and CNs. Therefore, if under a certain SNR, $I_{E,V}(I_{A,V}) > I_{A,C}(I_{E,C}) \quad \forall I_{A,V} \in [0,1)$, then for an infinite blocklength code we can achieve successful decoding with channel quality σ.

2.3.3 Differential evolution for optimizing profiles

Density evolution and EXIT charts are used to determine the threshold of a *given* profile $(\boldsymbol{\lambda}, \boldsymbol{\rho})$. If node types are the only parameters given, there will be a set of candidate profiles $\Phi = \{(\boldsymbol{\lambda}, \boldsymbol{\rho}) \mid (\boldsymbol{\lambda}, \boldsymbol{\rho}) \text{ satisfies } (2.16)\}$. Φ has approximately the dimension of $\Re^D[0,1]$, $D = |\mathcal{T}_V| + |\mathcal{T}_C| - 3$, therefore it is rather difficult to implement a direct optimization on Φ to identify $(\boldsymbol{\lambda}, \boldsymbol{\rho})_{best} \in \Phi$, such that $\sigma^\star(\boldsymbol{\lambda}, \boldsymbol{\rho})_{best} > \sigma^\star(\boldsymbol{\lambda}, \boldsymbol{\rho})$, $\forall (\boldsymbol{\lambda}, \boldsymbol{\rho}) \in \Phi$, $(\boldsymbol{\lambda}, \boldsymbol{\rho}) \neq (\boldsymbol{\lambda}, \boldsymbol{\rho})_{best}$. On the contrary, heuristic algorithms such as DE are still adequate tools for determining $\sigma^\star(\mathcal{T}_V, \mathcal{T}_C, R)$, the decoding threshold of $(\boldsymbol{\lambda}, \boldsymbol{\rho})_{best}$.

DE [80] belongs to the class of heuristic methods that try to find the *optimal solution* to the problem by *iteratively* generating improved candidates, under some carefully designed *measures*; therefore they are frequently used to perform optimizations on complicated combinatorial problems where direct optimization methods are unsuitable or inapplicable (e.g. many NP-hard problems). Heuristic methods include many algorithms such as hill climbing, genetic algorithm (GA) [82], simulated annealing (SA) [83], etc. As a powerful tool in the family, DE is widely used in the optimization of the degree profiles of LDPC codes [14, 84, 85]

DE first generates N random candidates $(\boldsymbol{\lambda}, \boldsymbol{\rho})_i,\ i = 1, 2..., N$ in Φ, then updates them over iterations to make gradual improvements in order to achieve $(\boldsymbol{\lambda}, \boldsymbol{\rho})_{best}$. We first describe below the rules for updating candidates, which is the core part occurring during one iteration in DE.

- **Mutation**

```
FOR $i = 1, 2...N$
1. Randomly choose three numbers $j, k$ and $l$ from $\{1, 2, ..., N\}$, such as $i \neq j \neq k \neq l$
2. $(\boldsymbol{\lambda}, \boldsymbol{\rho})_i^{new} = (\boldsymbol{\lambda}, \boldsymbol{\rho})_j + F \cdot ((\boldsymbol{\lambda}, \boldsymbol{\rho})_k - (\boldsymbol{\lambda}, \boldsymbol{\rho})_l)$
END
```

- **Selection**

```
FOR $i = 1, 2...N$
1. if $(\boldsymbol{\lambda}, \boldsymbol{\rho})_i^{new}$ is "better" than $(\boldsymbol{\lambda}, \boldsymbol{\rho})_i$, $(\boldsymbol{\lambda}, \boldsymbol{\rho})_i = (\boldsymbol{\lambda}, \boldsymbol{\rho})_i^{new}$;
2. else $(\boldsymbol{\lambda}, \boldsymbol{\rho})_i = (\boldsymbol{\lambda}, \boldsymbol{\rho})_i$;
END
```

In the mutation step, $F \in (0, 1)$ is a parameter controlling the degree of mutation, making F too large will make the algorithm have difficulty finding the global optimum, whereas F too small will cause slow convergence or trapping at a local optimum. In practice it usually assumes a value between 0.2 and 0.8. In the selection step, a measure is needed to judge the goodness of $(\boldsymbol{\lambda}, \boldsymbol{\rho})$, which can be chosen as the minimum difference between VN and CN EXIT curves, i.e. "the width of the narrowest part of the tunnel". It should be noted that the update algorithm described above is only one of many possible update rules of DE. We can as well choose other forms of mutation rules described in [81] and a crossover step can also be integrated into the update [80].

Under a given σ, the updating iterations are performed many times until one of the N candidates can converge, or the maximum number of iterations is reached. In the former case we claim that $\sigma < \sigma^\star(\mathcal{T}_V, \mathcal{T}_C, R)$ and $\sigma > \sigma^\star(\mathcal{T}_V, \mathcal{T}_C, R)$ otherwise. Finally, in order to find $\sigma^\star(\mathcal{T}_V, \mathcal{T}_C, R)$, an outer loop over σ is needed, where we start from a relatively small σ, and increase it gradually until $\sigma = \sigma^\star(\mathcal{T}_V, \mathcal{T}_C, R)$.

As an example of DE, Table 2.5 gives the optimized $(\boldsymbol{\lambda}, \boldsymbol{\rho})$'s of rate-0.5 LDPC profiles, and their thresholds on the BEC derived by DE, under the constraints of different $dvmax$'s. The Shannon limit for rate-0.5 codes is $\epsilon^\star = 0.5$, and with the increase of $dvmax$ the threshold is approaching this limit.

Table 2.5: Optimized distributions of some rate-0.5 LDPC profiles and their thresholds on BEC.

$dvmax$	$\lambda(x)$	$\rho(x)$	Threshold (ϵ)
3	$0.3836x + 0.6164x^2$	$0.9590x^4 + 0.0410x^5$	0.4488
4	$0.4788x + 0.5212x^3$	$0.5457x^4 + 0.4543x^5$	0.4638
6	$0.4175x + 0.1684x^2 + 0.4141x^5$	$0.0095x^4 + 0.9892x^5 + 0.0013x^6$	0.4773
8	$0.3625x + 0.2518x^2 + 0.3857x^7$	$0.5862x^5 + 0.4075x^6 + 0.0063x^7$	0.4844

Another DE example for DGLDPC codes uses $\mathcal{T}_V$ ={$(8,1)$REP, $(6,5)$SPC (cyclic G_{comp})} and $\mathcal{T}_C$={SPC$(6,5)$, Hamming$(15,11)$}, Fig. 2.12 gives a $R = 0.5$ $(\boldsymbol{\lambda}, \boldsymbol{\rho})$ optimized by DE with decoding threshold $\epsilon^\star = 0.4840$ on BEC channel, where

$$\boldsymbol{\lambda} = (0.4825, 0.5175)$$

$$\boldsymbol{\rho} = (0.2087, 0.7913)$$

It can be observed that VN EXIT curve is always above CN curve but with a very narrow open tunnel exists between them.

2.4 Tools for constructing finite-length codes

2.4.1 PEG (progressive edge growth) algorithm

PEG [25] is a semi-random graph algorithm for constructing FL codes. It begins with an empty Tanner graph, and appends the edges sequentially. During the appending, it tries to connect the current VN with the best CN candidate so that the girth of the partial Tanner graph is maximized; this procedure continues until all the edges have been appended.

When building a blocklength N LDPC code, one needs to specify the VN degree vector $\{dv_1, dv_2, ..., dv_N\}$ before implementing PEG, whereas the CN degree vector is left open. PEG tends to generate a concentrated CN-degree profile (i.e. only two CN degrees in the profile), due to its connecting feature during the process, which we will see in the following.

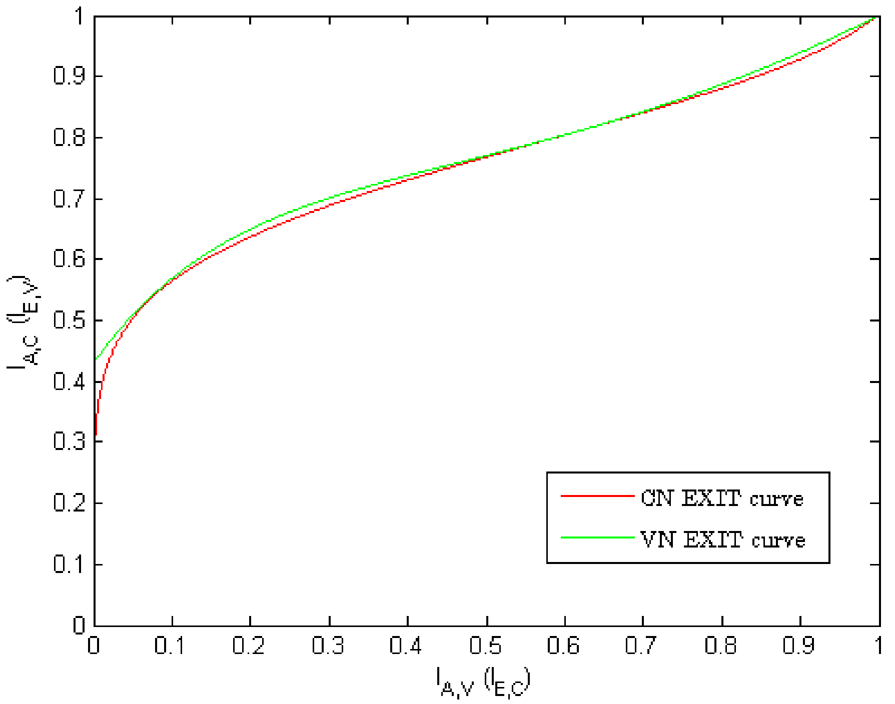

Figure 2.12: The EXIT chart of an optimized DGLDPC profile $(\boldsymbol{\lambda}, \boldsymbol{\rho})$ with decoding threshold $\epsilon^\star = 0.4840$.

The PEG algorithm is as follows

- FOR VN 1, 2,...,N, do the following

1. For the first edge of the VN i, connect it to the CN whose current degree is the smallest, randomly choose one if there are many of them.

2. For the remaining edges of the VN i, according to the current partial Tanner graph, expand the local tree rooted in i until the case 2.1 or 2.2 is met. Note that 2.1 and 2.2 are *mutually-exclusive*.

Case 2.1. The local tree cannot cover all the CNs in the Tanner graph, i.e. some CNs, denoted by $\mathcal{CN}(\infty)$, are never reachable by expanding the local tree, or

Case 2.2. The local tree can cover all the CNs in the Tanner graph. Let CNs, denoted by $\mathcal{CN}(2l)$, be those not reachable by the local tree down to $(2l-3)$th level, but appearing in the $(2l-1)$th level.

3. If case 2.1, connect the edge to the CN with the smallest degree in $\mathcal{CN}(\infty)$, randomly choose one if there are many of them. If case 2.2, connect the edge to the CN with the smallest degree in $\mathcal{CN}(2l)$, randomly choose one if there are many of them.

Case 2.1 usually happens at the early stage of PEG, when the Tanner graph is not fully populated, therefore by appending the edge to $\mathcal{CN}(\infty)$ will not incur any cycles. Case 2.2 usually happens at the later stage of PEG, when the Tanner graph is almost fully connected, therefore appending the edge will inevitably introduce cycles. PEG tries to connect the VN to the farthest CN in the local tree so as to maximize the cycle length; by appending the edge to $\mathcal{CN}(2l)$ will incur cycles of length-$2l$.

As a simple example, the PEG procedures for a small LDPC code with $N = 4$ and $M = 3$ is shown below. The VN degree vector is $\{2, 2, 2, 2\}$.

1) The 1st edge of VN 1 can be connected to any CNs since the Tanner graph is empty, choose a CN at random, say CN 1.

2) For the 2nd edge, expand the local tree, and it cannot reach CN 2 or 3; choose a CN at random, say CN 2, append the edge to CN 2. VN 1 is done.

3) The 1st edge of VN 2 can be connected to any CNs; choose the CN with the small degree, which is CN 3.

4) For the 2nd edge, expand the local tree, and it cannot reach CN 1 or 2, whose degree are both 1; choose a CN at random, say CN 1, append the edge to CN 1. VN 2 is done.

5) The 1st edge of VN 3 can be connected to any CNs. Both CN 2 and 3 have the smallest degree 1, choose one at random, say CN 2.

6) For the 2nd edge, expand the local tree. Since it can cover all the CNs, the "farthest" CN is CN 3 in level-5, append the edge to CN 3, a length-6 cycle is formed. VN 3 is done.

7) The 1st edge of VN 4 can be connected to any CNs. Since all the CNs have degree 2, choose one at random, say CN 1.

8) For the 2nd edge, expand the local tree. Since it can cover all the CNs, the farthest CNs are CN 2 and 3 with the same degree in level-3. Choose one at random, say CN 2, append the edge to CN 2, a length-4 cycle is formed. VN 4 is done.

After the PEG procedure, a girth-4 Tanner graph is constructed, shown in Fig. 2.13. Notice that the CN degree vector is $\{3, 3, 2\}$, which is a concentrated CN degree profile.

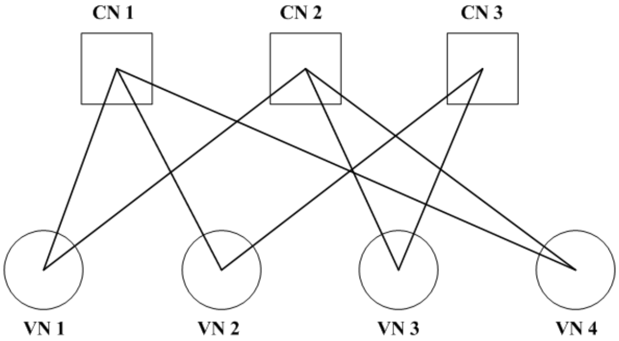

Figure 2.13: The Tanner graph of a $N = 4$, $M = 3$ LDPC code constructed by PEG.

2.4.2 EMD and ACE

PEG tries to maximize the girth of Tanner graph, which reduces the dependence of messages during the iteration. A further study shows that cycles, though of the same lengths, can have various contributions to the error events. The introduction of *extrinsic message degree (EMD)* and the *approximate cycle EMD (ACE)* [26] to LDPC codes enables a qualitative description on this phenomenon. EMD of a REP-VN subset refers to the *number* of SPC-CNs which are singly-connected to this subset. The larger the value of EMD is, the more information from outside subset will be received; on the contrary, a subset with a small EMD will receive less information and will more likely produce an error event.

In practice, it is more convenient to use ACE instead of EMD for a cycle: the ACE of a cycle is defined as $\sum_i (dv_i - 2)$, where dv_i is the degree of ith REP-VN, and the summation is over the REP-VNs in the cycle. The ACE is an upper bound on EMD and ACE=EMD iff no VNs in the cycle connect to the same SPC-CN outside the cycle. A length-4 cycle with ACE $= 4$ is shown in Fig. 2.14.

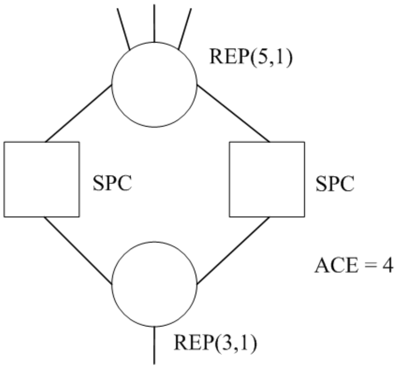

Figure 2.14: A length-4 cycle with ACE = 4.

The ACE of a REP-VN with degree dv, is defined as $(dv - 2)$ and the ACE of a SPC-CN is always 0. A cycle with smaller ACE is more susceptible to noise since it takes in less information from outside the cycle. When designing FL-LDPC codes, short cycles with small ACE should be avoided. The ACE and PEG can be combined to offer a further improvement on the error floor [27].

Chapter 3

Finite-Length GLDPC and DGLDPC Codes with Low Error Floors

The term “error floor”, refers to the phenomenon happening in the high SNR region during the iterative decoding, where the error curve decreases more slowly than it does in the waterfall region. The error floor is mainly caused by poor local structures in the Tanner graph. When PEG is employed to construct FL codes, there are many factors such as VN orderings and ACE which will greatly affect the code error floor performance. In addition, for GLDPC and DGLDPC codes which contain more node types relative to LDPC codes, the error floor is more sensitive to the specific connectivity of CNs and VNs. In this chapter, a new concept, eACE, is proposed to measure the susceptibility of cycles to the channel noise, and two modified PEG algorithms are devised based on eACE to improve the error floor performance of GLDPC and DGLDPC codes.

3.1 Variable node ordering in the PEG

When using PEG to create a FL code, the cycle lengths in the Tanner graph during the PEG process evolve as follows: in the early stage of PEG, there is no cycle (i.e. girth is ∞) since the graph is sparse, whereas in the later stage when the graph is nearly fully connected, cycles emerge and their lengths decrease as edges are appended gradually. As an example, we build a $(504, 252)(3, 6)$ regular LDPC code using PEG. Fig. 3.1 shows the development of shortest cycle length, as PEG progresses.

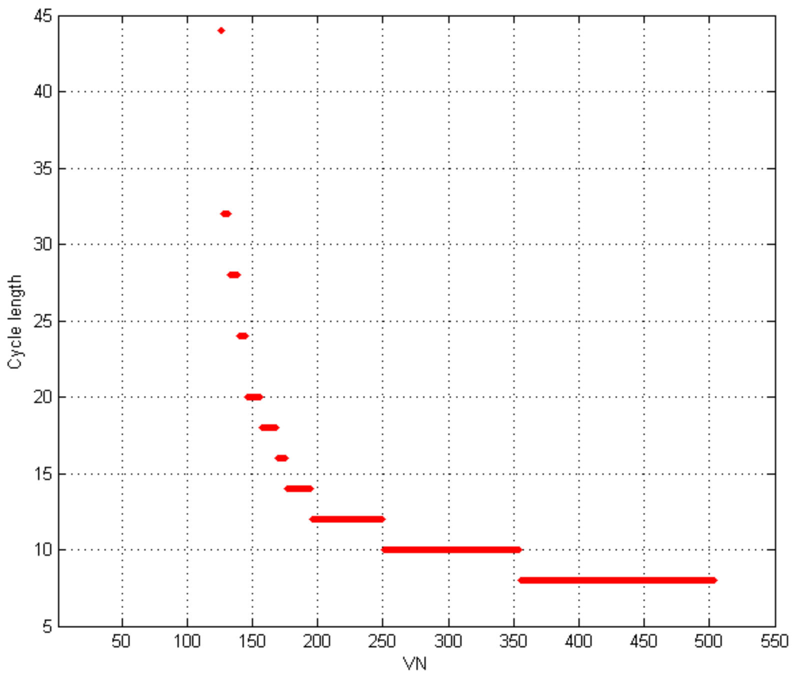

Figure 3.1: The shortest length of cycles as a function of VN i during PEG.

In Fig. 3.1, no cycle is generated when PEG traverses the first 125 VNs, and at 126th VN a length-44 cycle is formed. The shortest length of cycles keeps decreasing until it finally reaches 8, and this is the girth of the completed Tanner graph. Based on this observation and the notion of ACE, it is preferable to sort the VNs in a way such that they satisfies $dv_1 \leq dv_2 \leq dv_3 \leq ... \leq dv_N$, in other words, rank the VNs from the lowest to highest degree. By doing so the shorter cycles inevitably generated at the end of PEG will contain the highest degree VNs, which increases the ACE of short cycles, so the error floor behavior can be improved.

Fig. 3.2 shows the performance of PEG-constructed length-1008 rate-0.5 irregular LDPC codes. The degree profile is

$$\begin{aligned} \lambda(x) &= 0.3332x + 0.2404x^2 + 0.4264x^5 \\ \rho(x) &= 0.6734x^5 + 0.3266x^6 \end{aligned} \tag{3.1}$$

which is optimized by EXIT charts and DE, and it has a decoding threshold of 0.597 dB on the

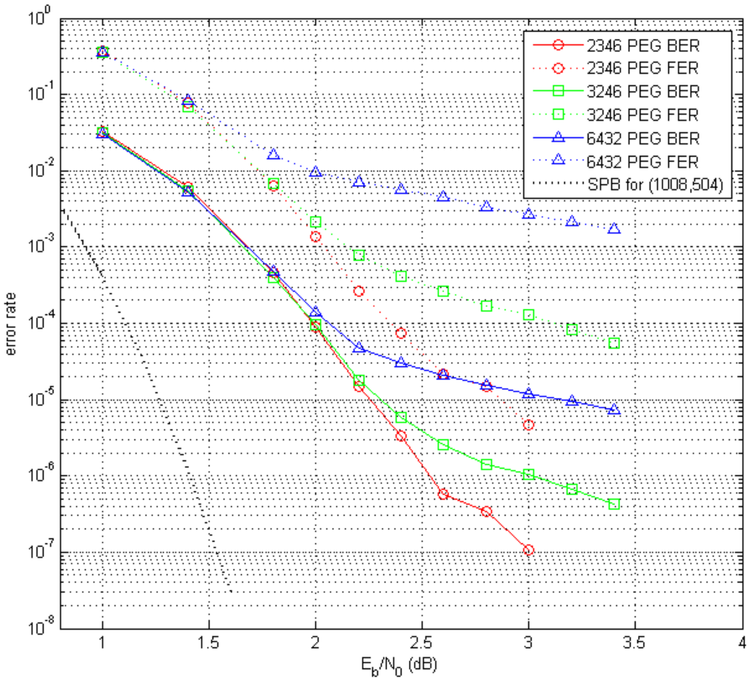

Figure 3.2: The effect of VN ordering in the PEG for irregular LDPC codes.

BIAWGN channel. In Fig. 3.2, we adopt different VN orders to verify the claim above. We use SPA for decoding, and the maximum number of iterations I_{max} is set to 200. The notation “2346” in the figure means that the PEG starts from deg-2 VNs, then deg-3, deg-4 and deg-6 VNs. “3246” and ”6432” are defined similarly. It can be observed that LDPC code with the order “2346” has the best error floor performance, due to the higher ACE value of short cycles, whereas the code with order ”6432” has the worst error floor. Fig. 3.2 also shows the sphere-packing bound (SPB) [28] for AWGN channels for $N = 1008$, $K = 504$ FL codes. The SPB is the ultimate bound on the error performance of *FL codes*; it approaches and becomes the capacity limit as the code length goes to infinity. It can be seen that there is about 1.25 dB degradation of “2346” code from SPB at frame error rate (FER) around 10^{-4}, which is caused mainly by the suboptimality of iterative decoding compared with ML decoding.

For a linear code $C(n, k)$, let $\mathcal{A}_{C(n,k)} = \int_0^1 I_E(I_A)dI_A$ be the area below the EXIT curve of $C(n, k)$. We say a code C_1 is “stronger” than C_2 if $\mathcal{A}_{C_1} > \mathcal{A}_{C_2}$, otherwise C_1 is “weaker”. Based on these terms the principle of VN ordering in the LDPC-PEG can also be phrased as: *start from the weakest VNs, then second weakest VNs and so on, finally the strongest VNs.*

For DGLDPC codes with no SCNs in it, the principle above can be readily applied, i.e. order the VNs in DGLDPC codes, according to $\mathcal{A}$, from weakest to strongest. Such ordering of VNs offers the best error floor behavior. Fig. 3.3 gives the performance of a PEG-constructed length-1200 rate-0.5 DGLDPC codes using SPC(3, 2) as its SVNs, with G_{comp} of systematic form. The degree profile is

$$\lambda(x) = 0.0017x + 0.3782x^2 + 0.4156x^5 + 0.2045x^2(SPC(3,2))$$
$$\rho(x) = 0.9825x^5 + 0.0175x^6,$$

which is optimized by EXIT charts and DE, and it has a decoding threshold of 0.55 dB, about 0.36 dB from capacity. MAP decoding is used on SPC(3, 2) and $I_{max} = 200$. In Fig. 3.2, the notation "SVN236" means that the PEG starts from SPC(2,3), then deg-2, deg-3 and deg-6 VNs. "23SVN6" and "632SVN" are defined similarly. It can be seen that DGLDPC code following the above principle (i.e. "SVN236") has the best error floor performance, whereas the code with "6432SVN" has the worst error floor, and it becomes almost flat in high SNR region.

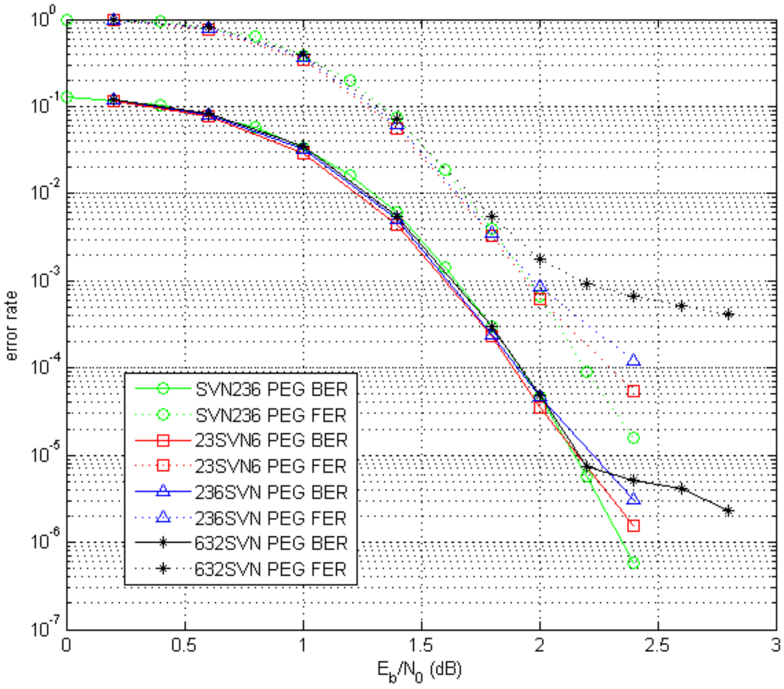

Figure 3.3: The effect of VN ordering in PEG for DGLDPC codes with no SCNs.

3.2 Extending ACE to eACE and the modified PEG

In this section we discuss the treatment of CNs in the PEG. First we define a new concept, the eACE of a cycle, defined as

$$\sum_{node\, i \in cycle} (d_{min,i} - 2) \tag{3.2}$$

where $d_{min,i}$ is the minimum Hamming distance of the ith node (VN or CN), and the summation is over both *VNs* and *CNs* in the cycle. It can be observed that ACE is a special case of eACE, since a deg-dv REP-VN has $d_{min} = dv$ and a SPC-CN has $d_{min} = 2$, so the case of LDPC codes eACE degenerates to ACE. In Fig. 3.4 we give an illustration of ACE and eACE.

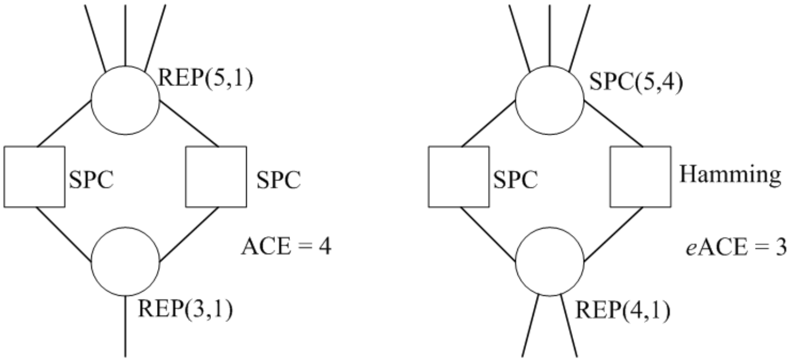

Figure 3.4: ACE in LDPC codes and eACE in DGLDPC codes for some length-4 cycles.

The rationale for introducing the notion of eACE is the following: in GLDPC/DGLDPC codes, SCNs have better error correcting capabilities than SPC-CNs, and when studying the susceptibility of a cycle to noise, the error correcting capabilities of CNs should be considered. It is verified by simulations that eACE is an effective qualitative property of cycle, which will be shown in the following simulations.

We propose a modified PEG algorithm incorporating eACE and it is suitable for building hGLDPC and DGLDPC codes. For simplicity, we assume that there is only one SCN type in the CNs. i.e. CNs contain SPCs and SCNs using the same component code, typically the Hamming $(15, 11)$ code. In addition, if a SCN has degree n, we say it has n sockets; if there are l $(l < n)$ edges connected to it, we say this SCN has $(n - l)$ *empty sockets* and l sockets have been filled up.

The modified PEG algorithm is as follows

- Order VNs from weakest to strongest. For VN 1, 2,...,N, do the following

1. For the first edge of the VN i, connect it to the SCN which has empty sockets and the current degree is the smallest; randomly choose one if there are many of them. If there is no such SCN, connect it to SPC with the smallest degree; randomly choose one if there are many of them.

2. For the remaining edges of the VN i, according to the current partial Tanner graph, expand the local tree rooted at i until case 2.1 or 2.2 is satisfied.

Case 2.1. The local tree cannot cover all the CNs in the Tanner graph, i.e. some CNs, denoted by $\mathcal{CN}(\infty)$, are never reachable by expanding the local tree.

Case 2.2. The local tree can cover all the CNs in the Tanner graph. Let CNs, denoted by $\mathcal{CN}(2l)$, be those not reachable by the local tree down to $(2l-3)$th level, but appearing in the $(2l-1)$th level.

3. If case 2.1, connect the edge to the SCN which has empty sockets and the smallest degree in $\mathcal{CN}(\infty)$. If there is no such SCN, connect it to SPC with the smallest degree. If case 2.2, connect the edge to the SCN which has empty sockets and has smallest degree in $\mathcal{CN}(2l)$. If there is no such SCN, connect to SPC which has the smallest degree in $\mathcal{CN}(2l)$.

The modified PEG is referred to as "SCNfirst-PEG" in this work. Similarly we consider "SPCfirst-PEG" if we exchange "SPC" with "SCN" in the algorithm above. Notice that eACE is naturally integrated into SCNfirst-PEG, since the connecting rules tend to generate the edges connecting weak VNs and strong SCNs; in case 2.2, when the cycles of length-$2l$ are formed, they all contain SCNs instead of SPCs, therefore increasing the eACE of cycles. In addition, notice that a SCN has a *fixed* number of sockets whereas a SPC has a *flexible* number of sockets. An ordinary PEG may leave SCNs with some empty sockets in the final stage of PEG, hence the last several edges have to connect to SCNs to fill up all the sockets of SCN, which possibly incurs some very short cycles. In contrast, SCNfirst-PEG tends to fill up the sockets of SCNs in an earlier stage, and at the end of PEG, by simply adjusting the degree of SPCs when necessary, the girth property of the Tanner graph can remain uncompromised.

In Fig. 3.5, we give the simulation results of several length-1000 rate-0.5 hGLDPC codes, built by SCNfirst-PEG and SPCfirst-PEG with different VN orderings. The degree profile of hGLDPC code is

$$\lambda(x) = 0.3881x + 0.2681x^2 + 0.0027x^3 + 0.0034x^4 + 0.3377x^5$$

$$\rho(x) = 0.6303x^6 + 0.1010x^7 + 0.0061x^8 + 0.0171x^9 + 0.2456x^{14}\ (Hamming(15, 11))$$

which is optimized by EXIT charts and DE, and it has a decoding threshold of 0.50 dB. In the experiment, we set $I_{max} = 200$, and use one-sweep MAP decoding on SCNs [76] and SPA on SPC-CNs and REP-VNs. For the sake of fairness we also require that no modifications on SPC-CNs are allowed, therefore all the codes have the same degree distribution after PEG.

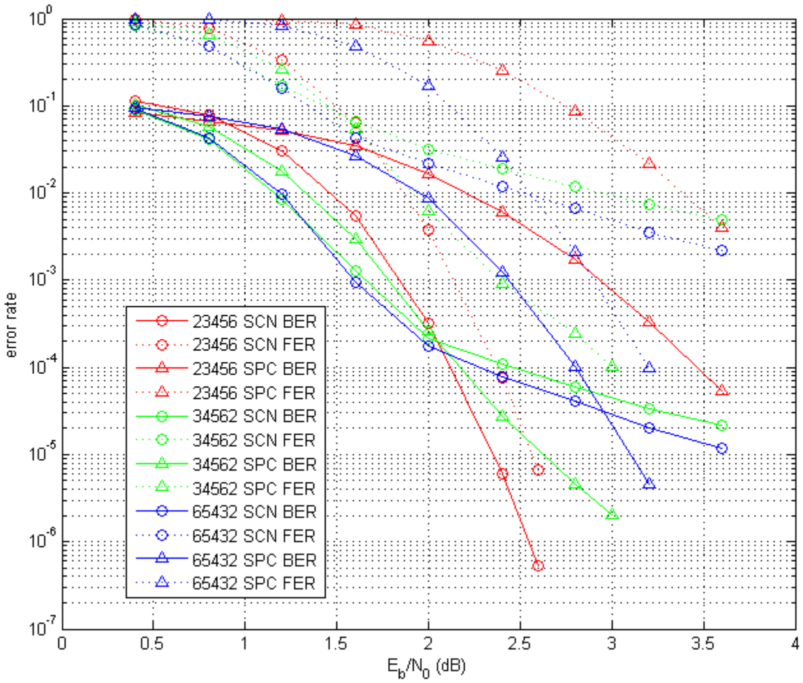

Figure 3.5: Simulation results of length-1000 rate-0.5 hGLDPC codes, built by SCNfirst-PEG and SPCfirst-PEG with different VN orderings.

In Fig. 3.5, the notation “23456 SCN” means that the VN ordering is “deg-2, deg-3,...,deg-6 REP-VNs”, and SCNfirst-PEG is used. The remaining notations are defined similarly. It can be seen that code “23456 SCN” has the best error floor behavior, which agrees with the analysis above; whereas “34562 SCN” and ”65432 SCN” have high error floors, due to the fact that deg-2 REP-VNs have to connect to SPC-CNs at the end of PEG, generating short cycles with poor eACE values.

3.3 Relaxation on the girth constraint to lower the error floor of DGLDPC codes

We call a cycle with length ξ and ACE (or eACE) η a "(ξ, η)-cycle". For two cycles with (ξ_1, η_1) and (ξ_2, η_2) respectively, if $\xi_1 > \xi_2$ and $\eta_1 > \eta_2$, it is reasonable to conclude that the (ξ_1, η_1)-cycle is less susceptible to channel noise than the (ξ_2, η_2)-cycle. However, if $\xi_1 < \xi_2$ and $\eta_1 > \eta_2$, it is not clear which cycle is more noise-resilient, since ξ and η are merely qualitative measurements on cycles.

All the PEG algorithms above attempt to maximize the girth of partial Tanner graph during the edge-appending, hence they tend to generate the cycles with large ξ's, but not necessarily large η's. If we study the case 2.2 in the SCNfirst-PEG, it is very likely that there is no SCN in $\mathcal{CN}(2l)$, so the generated cycles have poor eACE values. For DGLDPC codes, if SPC codes are used as SVNs (they are even weaker than deg-2 REP-VNs), simulation shows that a cycle containing many SPC-SVNs with a small η tends to cause decoder "trapping" even when ξ is large.

In view of this phenomenon, we further extend SCNfirst-PEG to "Girth-relaxed SCNfirst PEG" (GR-SCNfirst-PEG). In GR-SCNfirst-PEG it is furthermore required that SPC-SVN *must* be connected to SCNs. Adding such constraint generally decreases the girth of codes, but it will increase the η of cycles containing SPC-SVNs. The GR-SCNfirst-PEG is largely the same with SCNfirst-PEG, the only difference is in case 2.2: in GR-SCNfirst-PEG, if there is no candidate SCN in $\mathcal{CN}(2l)$ and VN i is a SPC-SVN, connect the edge to *the farthest SCN* in the local tree which has empty sockets and the smallest degree.

Fig. 3.6 depicts simulation results of length-1200, rate-0.5 DGLDPC codes, using SCNfirst-PEG, SPCfirst-PEG and GR-SCNfirst-PEG with different VN orderings, and no modifications on SPC-CNs is allowed. We choose SPC$(4, 3)$ (systematic G_{comp}) as SVNs and Hamming$(15, 11)$ as SCNs. The degree profile is

$$\lambda(x) = 0.1509x + 0.0147x^2 + 0.1834x^3 + 0.2478x^5 + 0.4032x^3 \ (SPC(4,3))$$
$$\rho(x) = 0.2323x^5 + 0.0685x^6 + 0.6992x^{14} \ (Hamming(15,11))$$

which is optimized by EXIT charts and DE, and it has a decoding threshold of 0.434 dB. In these DGLDPC codes, CN 1 to CN 119 are SCNs and VN 1 to VN 257 are SPC-SVNs. I_{max} is set to 200.

It can be seen in Fig. 3.6 that GR-SCNfirst-PEG code, which contains 16 length-4 cycles, has the best error floor performance, whereas the “SVN2346 SCNfirst” code, which has no length-4 cycles and supposedly should have good error floor behavior, shows a very shallow error floor.

The trapping sets (TS’s) of “SVN2346 SCNfirst” code were then studied. If a TS contains l VNs we call it a size-l TS. At $\frac{E_b}{N_0} = 3.6$ dB, we transmitted and decoded 300000 frames; after decoding, 956 erroneous frames are collected, all are invalid codewords, and there are 5930 erroneous bits in them. Table 3.1 lists the number and the size of these 956 TS’s. It is worth noting that, among 5930 erroneous bits, only 6 are REP-VNs and the remaining 5924 bits are attached to SPC-SVNs.

Table 3.1: Distribution of 956 TS’s in the “SVN2346 SCNfirst” code at $\frac{E_b}{N_0} = 3.6$ dB.

TS size	1	2	3	4	5	6	7	8	9	10	11	12	13
Number of TS’s	0	2	5	7	338	302	163	69	35	21	6	7	1

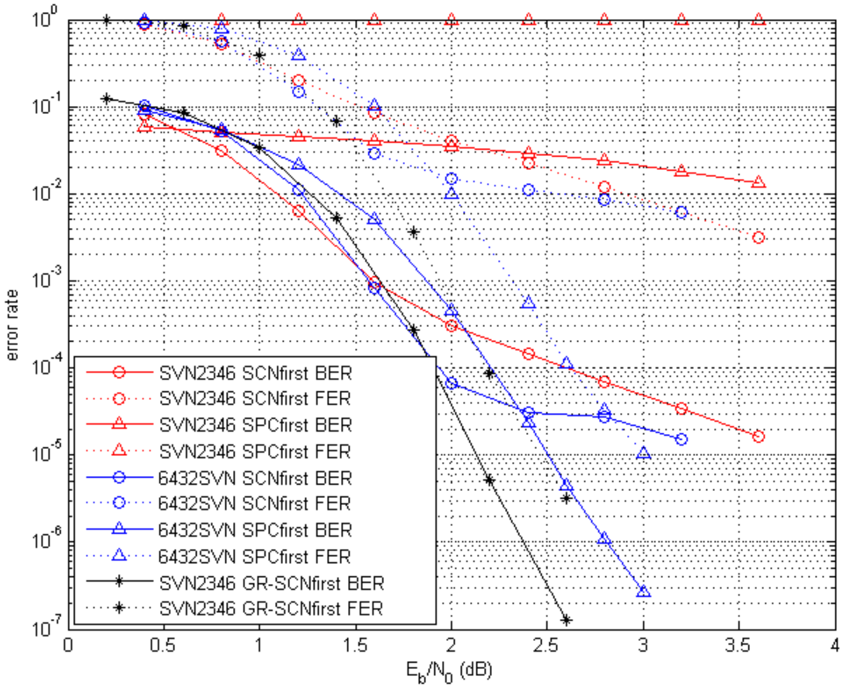

Figure 3.6: Simulation results of length-1200 rate-0.5 DGLDPC codes, built by SCNfirst-PEG, SPCfirst-PEG and GR-SCNfirst-PEG with different VN orderings.

From Table 3.1 we can see that TS’s containing 5, 6 and 7 SPC-SVNs are dominant error events. Of these 956 TS’s there are 146 different TS configurations. In Fig. 3.7, a typical size-5 TS which has shown up 94 times in the 956 TS's is drawn. Notice that this cycle has $\xi = 10$ and $\eta = 0$, and this TS is introduced by the *girth-maximizing* SCNfirst-PEG.

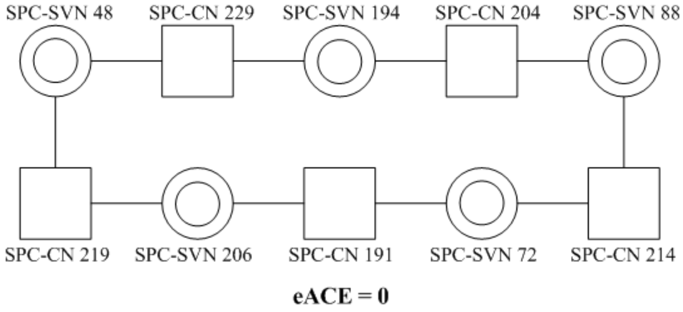

Figure 3.7: A length-10 cycle as a dominant TS in the “SVN2346 SCNfirst” code.

For GR-SCNfirst-PEG code, we find out that the smallest eACE of length-4 cycles is $\eta = 6$, such cycle contains two SCNs, a SPC-SVN and a deg-6 REP-VN, which is shown in Fig. 3.8. In addition, there are two additional edges of SPC-SVN connected to other SCNs outside the cycle. Such connectivity enables the enhanced protection of SPC-SVNs, therefore improving the error floor behavior.

The final remark on Fig. 3.6 is that “6432SVN SPCfirst” code also has a good error floor behavior. Due to the fixed number of sockets of SCNs, many length-4 cycles are generated in the final stage of PEG. It turns out that it contains 68 length-4 cycles, nonetheless every cycle contains two SCNs, which increases η and lowers the error floor.

GR-SCNfirst-PEG essentially imposes a changeover point between SPC-SVNs and REP-VNs during the PEG; for SPC-SVNs before the changeover a larger eACE is obtained at the expense of a (possible) decrease in the girth, and SCNfirst-PEG is used on REP-VNs after the changeover, with maximizing the girth as the primary objective. In practice, the switch point can be flexibly adjusted and placed among VNs, for example it can be place between deg-2 and deg-3 REP-VNs. The tradeoff between the waterfall and error floor performance as a function of this threshold is an interesting topic for future study.

3.4 Summary

In this chapter we focus on the designing FL-GLDPC and DGLDPC codes with good error floors. The main points are listed as follows:

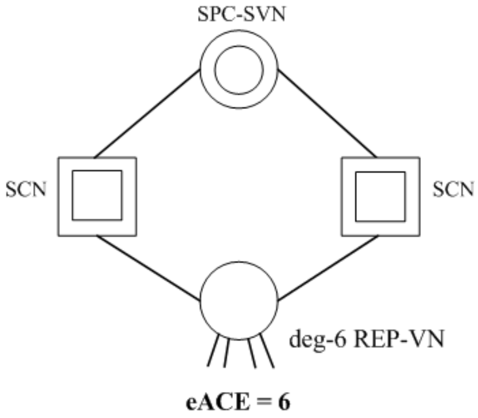

Figure 3.8: A length-4 cycle with eACE=6.

- The VN ordering is important in PEG, it is advisable processing VNs from the weakest to the strongest.
- eACE is a generalization of ACE, which can be naturally fit into GLDPC and DGLDPC codes.
- SCNfirst-PEG is an effective PEG algorithm for building FL-hGLDPC codes.
- Shorter DGLDPC codes are more likely to have poor error floors. GR-SCNfirst-PEG is an appropriate way to construct short DGLDPC codes with good error floors.

Chapter 4

Design of EE-GLDPC Codes

In practice, encoding complexity is an important issue when designing channel codes. The brute-force encoding method has the complexity $O(N^2)$, where N is the code length, while some carefully designed codes enjoy linear-time or even sublinear-time encoding complexity. After the rediscovery of LDPC codes under iterative decoding, one of the criticisms on LDPC codes is their higher encoding complexity compared with Turbo codes [13]. Later on, the algorithm proposed in [17] and the invention of IRA-LDPC [53] and EERC-LDPC codes [54] have largely solved the encoding complexity issue of conventional LDPC codes. In this chapter and next two chapters, we propose systematic design algorithms for EE-GLDPC and EE-DGLDPC codes, providing an across-the-board solution for designing EE block codes.

4.1 Review of IRA-LDPC and EERC-LDPC codes

IRA-LDPC codes: the *repeat-accumulate* code [51] is the prototype of IRA-LDPC codes. In such codes, the information bits are repeated q times, randomly-permuted, combined and accumulated to generate parity check bits. All these procedures are easy to implement and they have linear-time encoding complexity due to the accumulator. Repeat-accumulate codes are extended to irregular-repeat-accumulate codes in [52] by allowing irregular degree of repetition on different information bits, and can be further generalized to *extended irregular-repeat-accumulate codes* by allowing irregular degree of combining. Throughout this work, we use the term "IRA-LDPC" to denote extended irregular-repeat-accumulate codes.

In IRA-LDPC codes, the parity-check matrix H of dimension $M \times N$ can be written as $H = [H_1\ H_2]$, where the *random* submatrix H_1 is $M \times (N - M)$ and the *structured* submatrix H_2 is $M \times M$. The H_2 matrix has a bi-diagonal form

$$H_2 = \begin{bmatrix} 1 & & & & \\ 1 & 1 & & & \\ & 1 & \cdots & & \\ & & 1 & 1 & \\ & & & 1 & 1 \end{bmatrix} \tag{4.1}$$

with

$$H_2^{-T} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ & 1 & 1 & \cdots & 1 \\ & & 1 & \cdots & 1 \\ & & & \ddots & 1 \end{bmatrix} \tag{4.2}$$

All the VNs in H_2 except for the last one are deg-2 REP-VNs. Furthermore, no cycles exist in H_2, and H_2^{-T} is an upper-triangular matrix. Notice that the generator matrix G can be written as $G = [I\ H_1^T H_2^{-T}]$, so the parity bits can be obtained by first passing information bits through the irregular repeater and combiner (H_1^T) then the accumulator (H_2^{-T}). Letting $\boldsymbol{u}$ be the information sequence, the encoding process of IRA-LDPC code is depicted by the block diagram in Fig. 4.1. After encoding, the codeword can written as $\boldsymbol{c} = [\boldsymbol{u}\ \boldsymbol{p}]$.

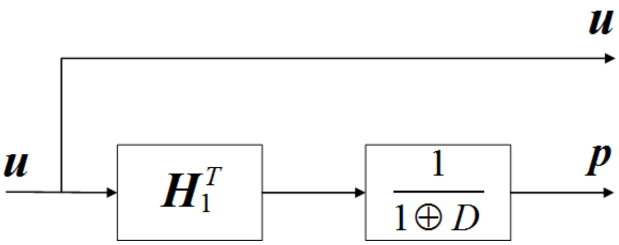

Figure 4.1: The encoding of an IRA-LDPC code.

Another way to *encode* IRA-LDPC codes is using an *iterative erasure decoder*. After inserting $\boldsymbol{u}$ into the iterative erasure decoder, since in the first parity check equation only the value of the first parity bit is unknown, therefore its value is found after the first iteration, in other words, the first parity bit is *recovered*. In the second iteration, only the value of the second parity bit is unknown in the second parity check equation, so its value is found after the second iteration. This procedure is done recursively until all the values of parity bits j, $j = 1, 2, ..., M$ are determined, at this point the codeword $\boldsymbol{c} = [\boldsymbol{u}\ \boldsymbol{p}]$ is found. It takes M iterations for the iterative erasure decoder to encode, still linear-time encoding.

EERC-LDPC codes: EERC-LDPC codes are another class of EE-LDPC codes with faster encoding. The VNs in H_2 of EERC-LDPC codes are still all deg-2 REP-VNs (excluding the last one, which is a deg-1 VN), and no cycles exist in H_2. In addition, the CN degree profile in H_2 of EERC-LDPC codes is more irregular than IRA-LDPC codes. An example of H_2 when $M = 8$ is depicted in (4.3).

$$H_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

EERC-LDPC codes can be encoded by a sliding-window encoder [54], and also an iterative erasure decoder. Since EERC-LDPC codes impose more constraints on the possible positions of "1"s in H_2 matrix (4.3), so it can be encoded in fewer iterations than IRA-LDPC codes. During the encoding, about half of the parity bits are recovered after the first iteration, about $\frac{1}{4}$ of the parity bits are recovered after the second iteration, and $\frac{1}{8}$ after the third iteration, and so on. Since the numbers of parity bits recovered in each iteration forms a geometric progression with common ratio $\frac{1}{2}$, It can be shown that the encoding time of EERC-LDPC codes is $O(\log_2 N)$.

Fig. 4.2 shows the H_2 matrix structure of a $(1024, 512)$ EERC-LDPC code. From Fig. 4.2 we can see that the structure of H_2 matrix ensures the encoding process finished in $\log_2 512 + 1 = 10$

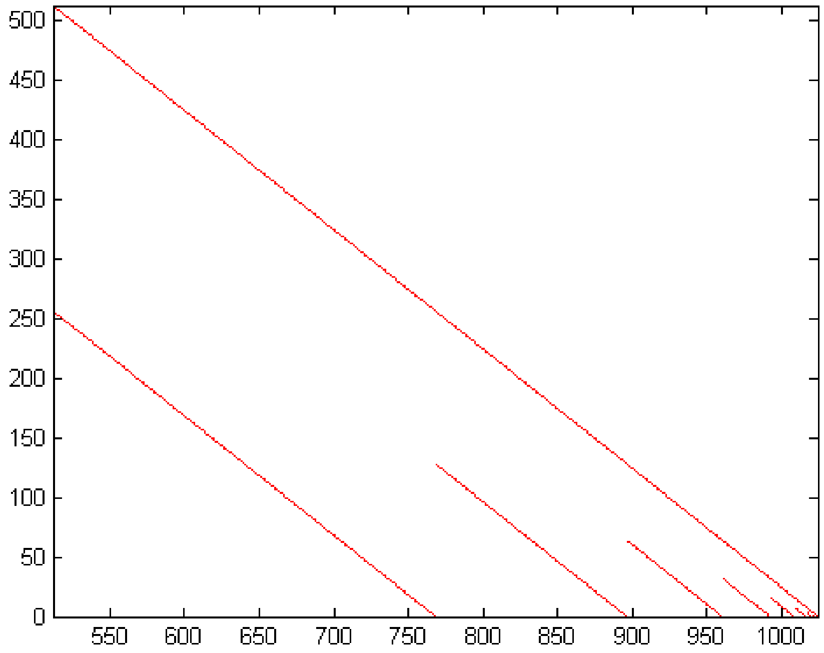

Figure 4.2: The structure of H_2 matrix of a $(1024, 512)$ EERC-LDPC code.

iterations, the number of parity bits recovered during each iteration is 256, 128, 64, 32, 16, 8, 4, 2, 1, and 1.

Extensions on IRA and EERC-LDPC codes IRA-LDPC and EERC-LDPC codes can be regarded as special classes of LDPC codes with $H = [H_1 \ H_2]$, where H_2 has a general form shown in Fig. 4.4:

$$H_2 = \begin{bmatrix} I_{n_1} & 0 & 0 & \cdots & 0 \\ & I_{n_2} & 0 & \cdots & 0 \\ & & I_{n_3} & \cdots & 0 \\ 0's & and & 1's & \ddots & 0 \\ & & & & I_{n_D} \end{bmatrix} \tag{4.4}$$

In (4.4), H_2 is a lower triangular matrix with the first $(M-1)$ VNs having degree 2 and the last VN degree 1. I_{n_i} denotes an $n_i \times n_i$ identity matrix. From (4.4) we can see that $\sum_{i=1}^{D} n_i = M$ and $n_D = 1$. Define $S_d = \sum_{i=1}^{d} n_i$, and let $S_0 = 0$. For parity bit $v_j, j = 1, 2 ... M$, let $S_{d-1} < j \leq S_d$. It is straightforward to obtain that the candidate CNs for connecting the second edge with parity bit v_j are from the CN set $\mathcal{C}_j$ with indexes $\{S_d + 1, S_d + 2, ..., M\}$.

We use Γ to denote the set of LDPC codes with such structure, and Γ to denote one certain LDPC code in Π . We can show that IRA-LDPC codes and EERC-LDPC codes belong to Π. For example, IRA-LDPC code has $\mathbf{n} = [n_1, n_2, ...n_D] = [1, 1, ...1]$, and $D = M$; EERC-LDPC code has $\mathbf{n} = [n_1, n_2, n_3, ...]$, and n_i is determined by (4.9).

For the codes in Π, we have the following properties.

1. An iterative decoder can encode the codeword of Γ in D steps, where D is the length of $\mathbf{n}$, since an iterative decoder can recover n_1 parity bits in the first step, n_2 parity bits second step, and so on.

2. The parity bit v_j, $S_{d-1} < j \leq S_d$ is a *d-step recoverable* (d-SR) node [62]; i.e. the bit v_j is recovered after dth iteration during the encoding. This is the direct result from property 1.

3. No cycles exist in H_2 for any code Γ.

The special LDPC code class Π, provides helpful insight on designing EE-GLDPC and EE-DGLDPC codes. In the next section we describe how to extend IRA-LDPC codes to IRA-sGLDPC codes.

4.2 From IRA-LDPC to IRA-sGLDPC codes: a heuristic exploration

If an erasure decoder is used to encode an IRA-LDPC code, the parity bits are recovered one by one in H_2, which means that, when recovering the jth bit in H_2, the jth bit is regarded as "parity bit" in current parity check equation. For sGLDPC codes, suppose $C(n, k)$ is used as component code, for each row of H_2, we need to expand the last bit to an all-1 row vector of length $(n - k)$, as shown in (4.5). Notice that by doing this the last $(n - k)$ bits are regarded as parity bits in each SCN, so the last $(n - k)$ columns of H_{comp} should be linearly independent.

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \cdots & \\ & & & 1 \end{bmatrix} \rightarrow \begin{bmatrix} \underbrace{1 \ldots 1}_{n-k} & & & \\ & \underbrace{1 \ldots 1}_{n-k} & & \\ & & \cdots & \\ & & & \underbrace{1 \ldots 1}_{n-k} \end{bmatrix} \tag{4.5}$$

Now consider the treatment of 1's on the secondary diagonal of IRA-LDPC codes. Two extreme schemes exist: scheme 1 puts only one "1" on the secondary diagonal, while scheme 2 replaces the single one by a row of $(n-k)$ ones. Both schemes have obvious flaws: the first scheme does not form cycles in H_2, however it generates many deg-1 parity bits that worsen the code performance. While the second scheme results in many length-4 cycles that worsen the code performance. In order to verify the above claims, we build two $(N = 1095, K = 511)$ EE-sGLDPC codes by PEG, with Hamming$(15, 11)$ as component codes. Fig. 4.3 shows the decoding performance of codes based on these schemes, and both of them show very serious error floors.

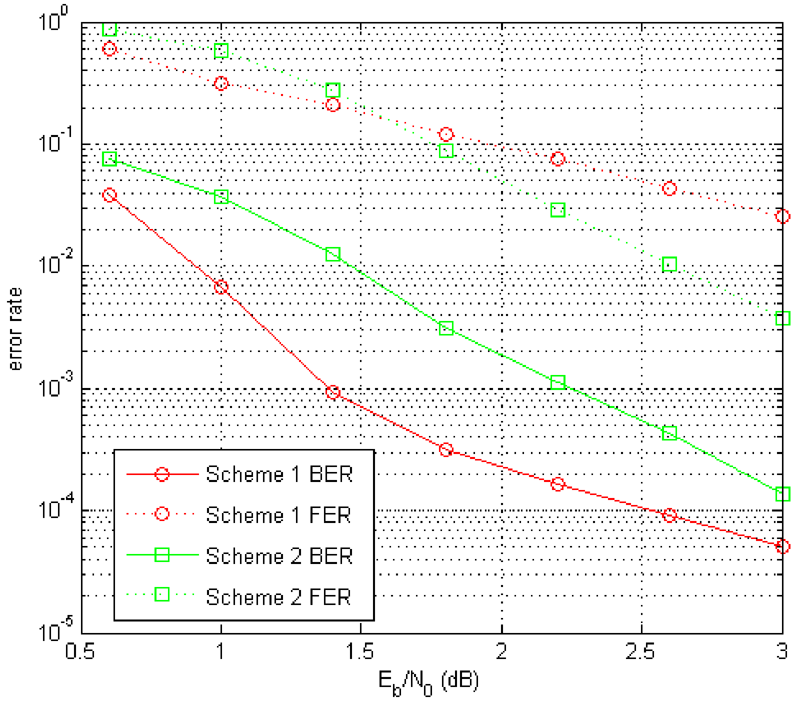

Figure 4.3: Performance of IRA-sGLDPC codes with poor H_2 structures.

Now we discuss how to overcome these two types of flaws. Let H_{adj} be a $M_C \times N$ matrix. After the expansion of the last bit to a all-1 row vector of length $(n-k)$ for each row, for the jth parity bit v_j, let $l = \lceil(\frac{j-1}{n-k})\rceil$, and define CN set $\mathcal{C}_j$ with indexes $\{l+1, l+2, ...M_C\}$, we argue that if the second edge of parity bit v_j is connected to one of the CNs from the set $\mathcal{C}_j$, then the constructed sGLDPC code has the EE property: when an iterative erasure decoder is used to encode, after the first iteration parity bit 1, 2,...,$(n-k)$ are guaranteed to be recovered; after the second iteration parity bit $(n-k+1)$, $(n-k+2)$,...,$2(n-k)$ are guaranteed to be recovered, and so on. All $M_C(n-k)$ parity bits can be recovered in at most M_C iterations, therefore the encoding time is $O(M_C) = O\left(N(1-R)/(n-k)\right) = O(N)$.

In practice, in order to generate as many deg-2 VNs as possible while keeping girth relatively large, for $j = 1, 2..., M_C(n-k)$, we try to make every VN v_j have degree 2 by connecting it to some CN selected from $\mathcal{C}_j$, and PEG is used to help judge the goodness of CNs in $\mathcal{C}_j$.

When j is small, the cardinality of $\mathcal{C}_j$ is large, and it decreases as j increases. At the end of a codeword, some VNs are left with degree 1 in order to avoid length-4 cycles. A toy example with $(n-k) = 3$ is shown as follows:

$$H_2 = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \ddots & 000 & 000 & 000 & 000 \\ \cdots & 111 & 000 & 000 & 000 \\ \cdots & 010 & 111 & 000 & 000 \\ \cdots & 100 & 010 & 111 & 000 \\ \cdots & 001 & 001 & 010 & 111 \end{bmatrix} \tag{4.6}$$

In this example, the last four rows of the H_2 matrix are shown. There are six deg-1 VNs in total in the last three rows. For $(M_C - j)$th row of IRA-sGLDPC codes, $j = 0, 1 \cdots (n-k-1)$, there are $(n-k-j)$ deg-1 bits. The total number of deg-1 VNs in H_2 can be calculated as $\sum_1^{n-k} j = \frac{1}{2}(n-k+1)(n-k)$, while the other VNs are deg-2 nodes. Due to the error correcting capability of SCNs, a small portion of deg-1 parity bits basically does not degrade error floor performance, as simulation results show.

4.3 Construction of EE-GLDPC codes with recovery vector n

In this section we extend the IRA-sGLDPC codes to other EE-GLDPC (including EE-sGLDPC and EE-hGLDPC) codes with flexible encoding speeds, providing a more general framework for EE-GLDPC codes. These EE-GLDPC codes can be seen as GLDPC versions of code class Π. In the following, we assume that the adjacency matrix $H_{adj} = [H_1\ H_2]$ has a dimension $M_C \times N$. We introduce the recovery vector $\mathbf{n} = [n_1, n_2, ...n_D]$, where $\sum_{i=1}^{D} n_i = M_C$; the recovery vector is a design parameter controlling the encoding speed. If we design EE-GLDPC codes according to $\mathbf{n}$ (the design algorithm is shown in the following), it is guaranteed that using an iterative erasure decoder the encoding process can be finished after at most D iterations.

Define $S_d = \sum_{i=1}^{d} n_i$ and $S_0 = 0$. Let c_i be the ith CN in H_{adj}, $i = 1, 2..., M_C$, and v_j the jth VN in H_{adj}, $j = 1, 2..., N$. For c_i the linear code $C(p_i, q_i)$ is used as the component code, and $H_{c,i}$ be the corresponding H_{comp}. Based on $\mathbf{n}$ we define $\mathcal{C}_i$ as the set of CNs with indices $(S_{i-1}+1), (S_{i-1}+2), ..., S_i$, so the cardinality of $\mathcal{C}_i$ is $|\mathcal{C}_i| = n_i$, $i = 1, 2..., D$.

Let $M = \sum_{i=1}^{M_C} (p_i - q_i)$, we first give the definition of *the envelope of the H_2 matrix.* It is a set of "1"s which assume special positions in H_{adj} matrix, and is defined as:

$$\{H(i,j) = 1 \mid (i,j) \text{ satisfies } N - M + \sum_{l=1}^{i-1} (p_l - q_l) < j \leq N - M + \sum_{l=1}^{i} (p_l - q_l)\} \tag{4.7}$$

Since the edges in the bipartite graph have a one-to-one mapping with the "1"s in H_{adj} matrix, in the following we use the term "edge" and $H(i,j)$ interchangeably. We call an edge "on the envelope" iff its corresponding $H(i,j) = 1$ is an element of the envelope. An edge is "below the envelope" iff the corresponding $H(i,j)$ has the property that $i = i_0$, $j > j_0$, and "above the envelope" iff $i = i_0$, $j < j_0$, where $H(i_0, j_0)$ is the edge on the envelope. We also define $\mathcal{V}_i$ as the set of v_j's such that

$$\{j \mid H(k,j) \text{ is on the envelope, } \forall\, k \in \mathcal{C}_i\} \tag{4.8}$$

We also refer to the set of on-the-envelope edges with row indices i as "the envelope section of c_i", and the number of edges in the envelope section of c_i is $(p_i - q_i)$. Now we consider the set of codes $\mathbf{\Pi}$ satisfying the following constraining rules

1. An envelope exists in the H_2 matrix.

2. $\forall\, v \in \mathcal{V}_i$, $i = 1, 2..., D$ the remaining edges of v can be connected to any CNs except those in $\mathcal{C}_l, 1 \leq l \leq i$.

3. For each $H_{c,i}$, the columns corresponding to the edges of the envelope section of c_i are linearly independent on $GF(2)$.

Let Γ be an arbitrary code in $\mathbf{\Pi}$. It can be straightforwardly argued that Γ can be efficiently-encoded by an iterative erasure decoder using maximum-*a-posteriori* (MAP) decoding: based on the values of information bits, the decoder can guarantee recovering all the unknown values of v's in $\mathcal{V}_1$ after the first iteration since all remaining edges of v's are connected to $\mathcal{C}_l, l = 2, 3...D$; then the second iteration recovers all the values of v's in $\mathcal{V}_2$, and so on, until all the parity bits are recovered.

In practice, it is possible that after the construction the actual recovery rate is slightly faster than D iterations, for the reason that, according to constraining rule 2, the length of the actual $\mathbf{n}$ could possibly be slightly decreased during the construction. So D becomes an upper bound on the actual recovery time.

For constraining rule 3, the $(p_i - q_i)$ independent columns ensure that the last $(p_i - q_i)$ nodes are parity bits in each super check equation during the iterations, therefore the MAP decoding can be used to recover them [29]; furthermore, if these nodes do not form a stopping set, a belief-propagation (BP) decoder can also be used; finally, if $H_{c,i}$ is systematic and such columns form a $(p_i - q_i) \times (p_i - q_i)$ identity matrix, then an erasure-filling decoder can be used, which is the simplest of the three decoders.

Based on the above discussions, it can be concluded that the codes in $\mathbf{\Pi}$ have the following properties: an iterative MAP decoder can encode the codeword of Γ in *at most* D steps. and the parity bits in $\mathcal{V}_i$ are *at worst* i-SR nodes. Finally, it is straightforward to obtain that the code set Π defined in previous section is a subset of $\mathbf{\Pi}$, since besides EE-LDPC codes, $\mathbf{\Pi}$ also includes EE-sGLDPC and EE-hGLDPC codes.

Application of recovery vector n: EERC-sGLDPC codes

$\mathbf{n}$ is a design parameter on EE codes, and in practice $\mathbf{n}$ can be flexibly adjusted to control the encoding speed. EERC-sGLDPC codes, which have a logarithmic encoding speed as a function of code length (i.e. $\log_2 N$), can be obtained by choosing a special $\mathbf{n}$. For a given M_C, n_i in $\mathbf{n}$ is determined recursively using the following formula [54]:

$$n_i = \lfloor M - \frac{1}{2}\sum_{j=0}^{i-1} n_j \rfloor \quad 1 \leq i \leq D-1, \text{and } n_0 = M \tag{4.9}$$

After determining $\mathbf{n} = [n_1, n_2, ..., n_D]$, we use the above method to construct H_2. The formula in (4.9) indicates that $\mathbf{n} = [n_1, n_2, ..., n_D]$ resembles a geometric progression with common ratio 0.5. Due to this property, a sequence of fractal triangles of 0's can be found along the diagonal of H_2 of EERC-sGLDPC codes. In addition, the $\mathbf{n}$ in EERC-sGLDPC codes imposes a more strict restriction on the CNs a parity bit v_j can connect to. Compared with IRA-sGLDPC codes, fewer CNs are available for a $v_j \in \mathcal{C}_i$ to connect to, especially for the values of j that is slightly larger than $(n-k) \cdot S_d$.

For EERC-sGLDPC codes, $\frac{1}{2}(n-k+1)(n-k)$ becomes a lower bound on the number of deg-1 VNs; this is because we usually do not have $n_{D-(n-k)+1} = n_{D-(n-k)+2} = \cdots = n_D = 1$ in $\mathbf{n}$. For example, when $n-k=3$ and $\mathbf{n} = [\cdots, 2, 1, 1]$, one possible H_2 structure is

$$H_2 = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \ddots & 111 & 000 & 000 & 000 \\ \cdots & 000 & 111 & 000 & 000 \\ \cdots & 100 & 010 & 111 & 000 \\ \cdots & 010 & 100 & 010 & 111 \end{bmatrix} \tag{4.10}$$

Now there are seven deg-1 VNs in the H_2 matrix, greater than $\frac{1}{2}(n-k+1)(n-k) = 6$. We can decrease this number by one by splitting $n_{D-2} = 2$ into the vector $[1, 1]$, however the length of $\mathbf{n}$ will increase by one, which means it take one more step for the iterative decoder to finish encoding. Generally, for a EE-sGLDPC code with $\mathbf{n}$, the number of deg-1 VN, N_1 can be calculated as

$$N_1 = (n-k)n_D + \sum_{i=t}^{D-1}\left(\left(n-k-\sum_{j=i+1}^{D} n_j\right) \cdot n_i\right) \tag{4.11}$$

where t satisfies

$$\begin{cases} (n-k) \leq \sum_{j=t}^{D} n_j \\ (n-k) > \sum_{j=t+1}^{D} n_j \end{cases}$$

In practical EE-sGLDPC code design, the number of deg-1 VNs usually only constitutes a very small portion of VNs, so the tradeoff between the number of deg-1 VN and encoding time appears to be a trivial issue. For hGLDPC and DGLDPC codes, usually SCNs corresponds to rows with small indexes and SPC-CNs with large indexes, so the number of deg-1 VN is only n_D.

The encoding time of EERC-sGLDPC codes is the length of $\mathbf{n}$, also notice the fact that $\mathbf{n} = [n_1, n_2, ..., n_D]$ resembles a geometric progression and $\sum_{i=1}^{D} n_i = M$, so the encoding time is on the order of $O(\log_2 M)$, or $O(\log_2 N)$. It is worth noting that for the same code length and rate, the number of rows of the adjacency matrix H_{adj} in GLDPC codes is $(n-k)$ times smaller than its counterpart in LDPC codes, which results in an even faster encoding time for EE-GLDPC codes.

Building IRA-LDPC and EERC-LDPC codes under generalized EE principle

IRA-LDPC and EERC-LDPC codes are also members of $\mathbf{\Pi}$. Both codes have $(p_i - q_i) = 1, \forall\, i$, since all CNs are SPC codes. In the following we propose an extra rule on H_2 of EE-LDPC codes, which simplifies the H_2 construction, and show that IRA-LDPC and EERC-LDPC codes are obtained by applying this rule under some specific $\mathbf{n}$. For the sake of simplicity we only consider the case that all parity bits in H_2, except the last n_D positions, are deg-2 REP-VNs.

Without loss of generality, it is assumed that when constructing the H matrix of EE-LDPC codes, the envelope of the H_2 matrix is first generated and we proceed to append the second edges of parity bits from left to right in H_2 matrix. The extra rule is as follows: the second edge is appended to the candidate CN with the smallest number of edges; if ties exist, choose the CN with the smallest number of index (this guarantees the uniqueness of H_2 matrix; if not done by so, many possible variations of H_2 exist in EERC-LDPC case). Another merit of this extra rule is it tries to generate a CN degree distribution as uniform as possible, which helps lower the error floor at high SNRs.

As an example, we give a 8×8 H_2 matrix under constraint $\mathbf{n} = [3, 3, 2]$: the red '1's form the envelope of H_2, and the blue '1's are the second edges which are all below the envelope satisfying the extra rule above.

$$H_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.12}$$

As special cases, it is readily seen that if $\mathbf{n} = [1, 1..., 1]$, by applying such rule we obtain IRA-LDPC codes, and EERC-LDPC codes if the elements in $\mathbf{n}$ form a geometric progression with common ratio 0.5.

The shape of H_2 of EE-GLDPC codes

For EE-GLDPC codes, H_2 itself must contain cycles, and the H_2 matrix of EE-GLDPC codes have more irregular structure compared with EE-LDPC codes. In Fig. 4.4, a blowup of the lower right corner of the H_2 matrix of an EE-hGLDPC code using $\mathbf{n} = [..., 16, 8, 8, 8, 4, 4, 4]$ is shown: the red dots correspond to "1"s in H_2 matrix. It can be observed that the distribution of red dots is fairly irregular. Also note that $\mathbf{n}$ imposes some "triangular forbidden areas" below the envelope with the dimension of the triangle determined by the elements of $\mathbf{n}$.

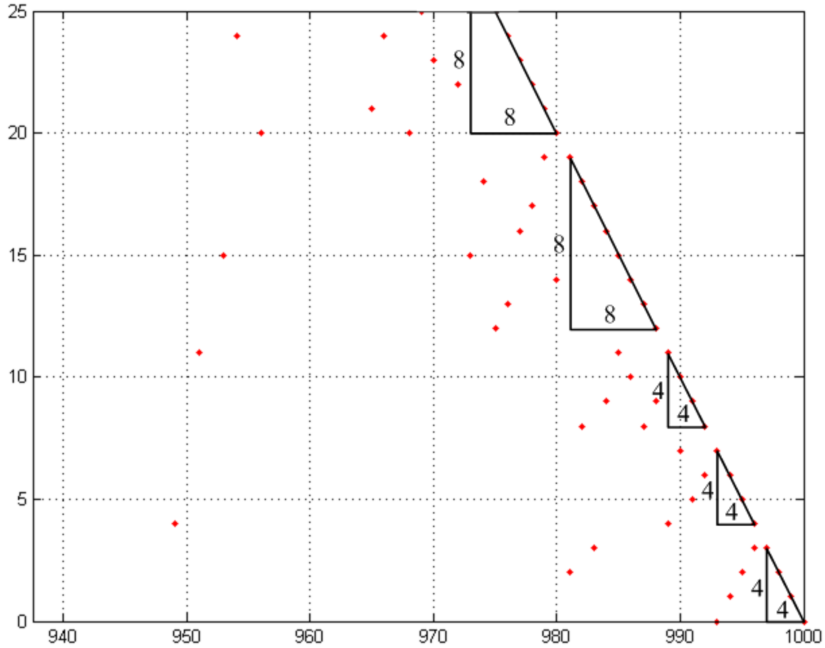

Figure 4.4: Blowup of the lower right corner of H_2 matrix of a length-1000 EE-hGLDPC code using $\mathbf{n} = [..., 16, 8, 8, 8, 4, 4, 4]$.

4.4 Encoding procedures and simple examples

We give a simple IRA-sGLDPC code using $(7, 4)$ Hamming code as its component code to illustrate the encoding process using an iterative erasure decoder. The IRA-sGLDPC code has a H_{adj} as

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & \vdots & 111 & 000 & 000 & 000 \\ 1 & 1 & 1 & 0 & 0 & \vdots & 010 & 111 & 000 & 000 \\ 0 & 1 & 0 & 1 & 0 & \vdots & 100 & 010 & 111 & 000 \\ 0 & 0 & 0 & 0 & 1 & \vdots & 001 & 100 & 010 & 111 \end{bmatrix}, \tag{4.13}$$

and is a $(17, 5)$ code. Suppose H_{comp} has the form

$$H_{comp} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

and the information sequence is $\boldsymbol{u} = 11101$.

⋆ 1st iteration: 11101 → 11101000. The red bits are regarded as information bits in H_{comp} to generate parity bits 000, so the first three parity bits of IRA-sGLDPC code are 000.

⋆ 2nd iteration: 11101000 → 11101000010. Similarly, the red bits are regarded as information bits in H_{comp} to generate parity bits 010, so the fourth, fifth and sixth parity bits of IRA-sGLDPC code are 010.

⋆ 3rd iteration: 11101000010 → 11101000010011. Following the procedures above, the values of 7th, 8th and 9th parity bits of IRA-sGLDPC code are found as 011.

⋆ Final iteration: 11101000010011 → 11101000010011011. The values of 10th, 11th and 12th parity bits of IRA-sGLDPC code are also found as 011.

⋆ Output: $\boldsymbol{u} = 11101$ encoded as $\boldsymbol{c} = 11101000010011011$, after four iterations.

It can be observed that this $(17, 5)$ IRA-sGLDPC code is a systematic linear code: $\boldsymbol{u} = 11101$ appears in the first five bits in $\boldsymbol{c} = 11101000010011011$. Similarly, it is straightforward to obtain that all the EE-GLDPC codes are systematic, and the information bits correspond to the columns of H_1 matrix.

As another example, we list below the H_2 matrix of a simple EE-hGLDPC code.

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{4.15}$$

The envelope is formed by red "1"s. c_1 and c_2 are SCNs, they use some linear code $C(n, n-3)$ as component codes, the remaining CNs are SPC codes. All VNs are deg-2 nodes except the last one. We adopt $\mathbf{n} = [2, 1, 1, 1]$. When encoding, 6 parity bits are recovered after the 1st iteration, then the 7th, 8th and 9th parity bits after 2nd, 3rd and 4th iterations, respectively.

4.5 Simulation results

The performance of EE-GLDPC codes is studied in this section. In the first experiment, we construct a $(1024, 640, 0.625)$ IRA-sGLDPC code π_1 using Reed-Muller $(3, 5)$ code as the component code. The component code is a $(32, 26)$ code so the adjacency matrix H_{adj} is 64×1024. The degree distributions of π_1 are:

$$\lambda(x) = 0.0103 + 0.9590x + 0.0308x^2,$$
$$\rho(x) = x^{31}$$

The PEG algorithm is used to build π_1, and it operates with decreasing indices of parity bit j, since the cardinality of available CNs increases if j decreases. We remark that the PEG algorithm does not specify the CN degree, whereas the H_{adj} matrix of sGLDPC codes should have a fixed row weight. It could happen that in the final stages a very few shorter cycles may be formed due to the lack of available sockets in SCNs, which slightly affects the cycle spectrum of code. In this experiment, PEG can avoid length-4 cycles in H_{adj} so the girth of π_1 is 6.

Fig. 4.5 shows the structure of H_{adj} matrix of π_1. It can be seen that H_1 has a random structure and H_2 is a lower-triangular matrix, which enables the EE property.

Fig. 4.6 shows the performance of π_1 on the BIAWGN channel. For comparison, we designed two length-1200, rate-0.5 EE-LDPC codes; one is an IRA-LDPC and the other an EERC-LDPC code; for these codes we set $dvmax = 6$ and the profile is optimized by EXIT charts and DE as

$$\lambda(x) = 0.3332x + 0.2404x^2 + 0.4264x^5$$
$$\rho(x) = 0.6734x^5 + 0.3266x^6$$

Due to existence of deg-1 VNs and the fixed structure of H_2 matrix, the actual degree profile (especially $\rho(x)$) obtained by PEG is slightly different from above. We puncture the first 200 parity bits in the H_2 matrix of EERC-LDPC code, which is optimal puncturing argued by [54], and 200 parity bits in the H_2 matrix of IRA-LDPC code are also punctured using the method proposed in [66]; therefore two punctured LDPC codes are $(1000, 600)$ rate-0.6 codes. The maximum number of iterations is set to $I_{max} = 200$ for three codes.

In Fig. 4.6, we can see that IRA-LDPC and EERC-LDPC punctured codes show slightly better

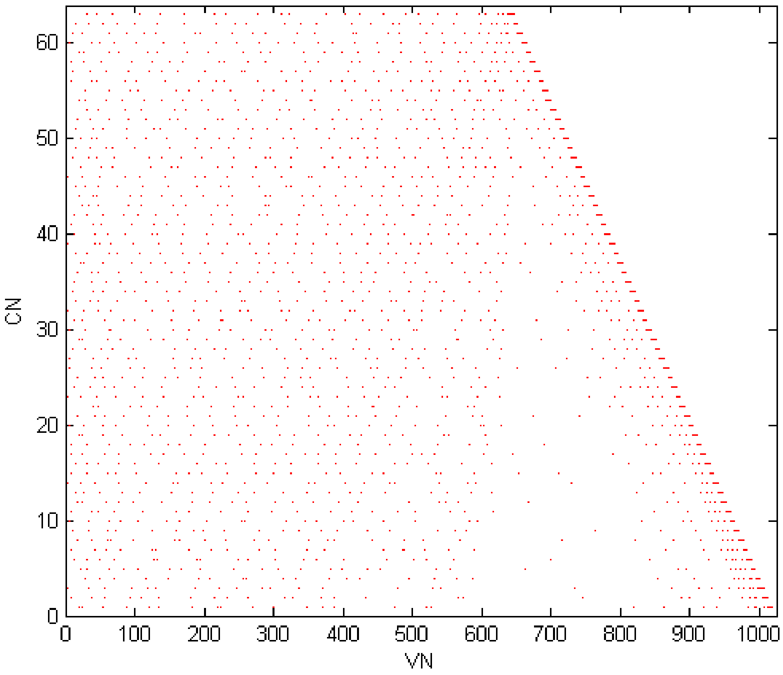

Figure 4.5: The structure of H_{adj} matrix of π_1.

waterfall performance over π_1, by about 0.1 dB at BER$= 10^{-3}$. This is due to the inherent rate loss of sGLDPC codes. However, IRA-LDPC and EERC-LDPC punctured codes show a high error floor compared with π_1 at high SNR. π_1 outperforms LDPC codes about 0.6 dB at BER$= 10^{-7}$, with a similar code length and a higher code rate. The low error floor of π_1 is attributed to the powerful error correcting capabilities of Reed-Muller $(5, 3)$ codes at high SNR, and such low error floor is achieved at the expense of a slightly degraded waterfall performance.

The EERC-sGLDPC code is studied in the second experiment: a $(645, 301, 0.467)$ code π_2 is devised using a $(15, 11)$ Hamming code as component code. The H_{adj} of π_2 is 86×645 and contains no length-4 cycles. The recovery vector $\mathbf{n}$ is chosen as $[43, 21, 11, 5, 2, 2, 1, 1]$, satisfying a geometric progression. The degree profile of π_2 are

$$\lambda(x) = 0.0085 + 0.9659x + 0.0256x^2$$

$$\rho(x) = x^{14}$$

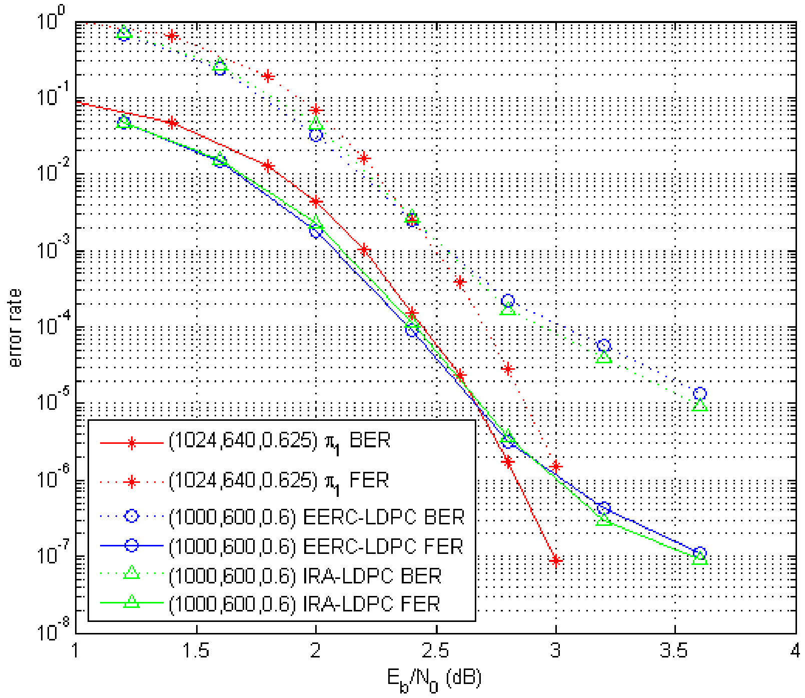

Figure 4.6: Comparison of the $(1024, 640, 0.625)$ IRA-sGLDPC code with IRA-LDPC and EERC-LDPC punctured codes.

Fig. 4.7 shows the structure of H_{adj} of π_2. It can be seen that the fractal triangles (forbidden areas) along the diagonal of H_2 make the logarithmic encoding speed feasible.

Fig. 4.8 compares the performance of π_2 with ordinary GLDPC codes [33]. π_2 and these ordinary GLDPC codes are of slightly different lengths but use the same component code. From Fig. 4.8, after taking the code-length difference into account, we conclude that, as a special class of GLDPC codes, π_2 offers a similar or slightly better performance over the ordinary sGLDPC codes. Moreover, π_2 exhibits a good error floor behavior at high SNRs, for both BER and FER.

Performance comparison of π_2 with irregular LDPC code is done in the next experiment: we puncture 42 parity bits from π_2 to obtain π_3, a $(603, 301, 0.499)$ code. These 42 parity bits are punctured in a way such that each SCN contains at most one punctured parity bits (i.e. uniform puncturing). The reason for such puncturing pattern is explained in Chapter 7. In order to illustrate the principle of uniform puncturing, we give the indices of first ten punctured bits: $[2, 8, 11, 14, 17, 21, 27, 29, 33, 39]$. Note that the punctured bits are uniformly distributed among the SCNs.

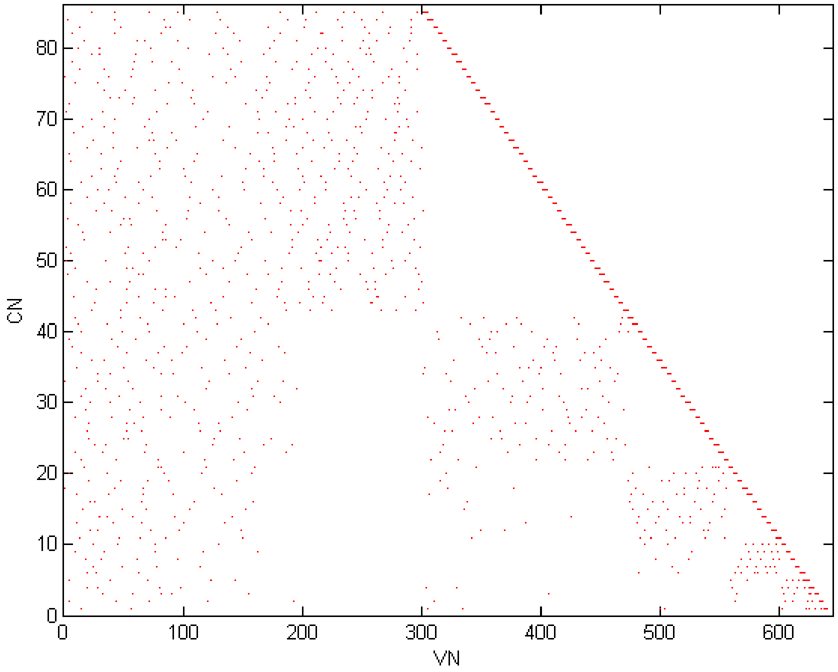

Figure 4.7: The structure of H_{adj} matrix of a $(645, 301)$ EERC-sGLDPC code.

The comparison of π_3 with the irregular LDPC code of same length and rate is made in Fig. 4.9. The irregular LDPC code devised in [30] has regular check degree 9 and the highest VN degree is 20. At low SNR, the irregular LDPC code outperforms π_3 a little because it enjoys a lower decoding threshold. However, π_3 behaves better after SNR passes through 2.2 dB. At BER $= 10^{-6}$, π_3 has an improvement of 0.4-0.5 dB over irregular LDPC codes, and about 0.8 dB at BER $= 10^{-7}$, gains are likely to be even better for even lower error rates.

The last experiment involves the design of IRA-hGLDPC codes. EXIT charts and DE are used to derive a good ensemble profile. The optimized profile has a decoding threshold $E_b/N_0 = 0.56$ dB, and has the degree distribution

$$\lambda(x) = 0.3740x + 0.3050x^2 + 0.0038x^3 + 0.0035x^4 + 0.3137x^5$$

$$\rho(x) = 0.0741x^5 + 0.5780x^6 + 0.0317x^7 + 0.0765x^8 + 0.2397x^{14}(Hamming(15, 11))$$

With the help of SCNfirst-PEG, we designed a $(N, K, R) = (1024, 512, 0.5)$ IRA-hGLDPC mother code $\mathcal{C}_h$ using the above optimized profile. Due to the randomness of PEG and the structure

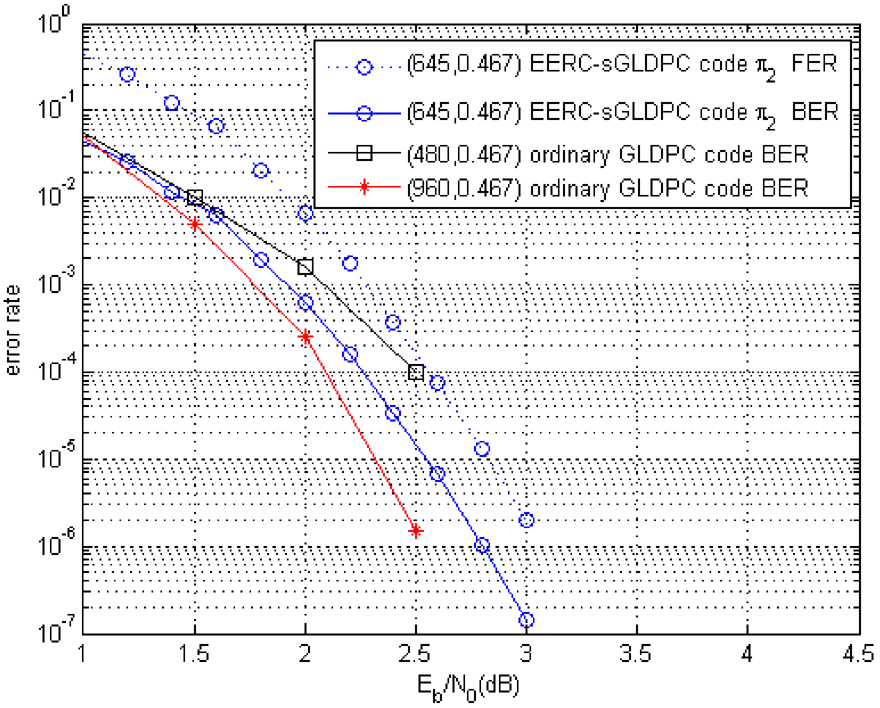

Figure 4.8: Performance comparison of EERC-sGLDPC code π_2 with ordinary GLDPC codes.

of IRA-hGLDPC code, the actual degree distribution is slightly different. H_{adj} is 368×1024 and contains 48 SCNs, and there is no length-4 cycle in H_{adj}.

Fig. 4.10 shows the structure of H_{adj} of $\mathcal{C}_h$. Notice that the envelope in H_2 matrix has two parts: the first part has a mild slope, which corresponds to envelope sections of SCNs, and the second part has a steeper slope, which is composed of envelope sections of SPC-CNs.

The performance of $\mathcal{C}_h$ is compared with EERC-LDPC and IRA-LDPC codes. We construct a EERC-LDPC code and a IRA-LDPC code of the same length and rate, with the same maximum VN degree $dvmax = 6$ using PEG. The profiles of LDPC codes are also optimized by EXIT charts and DE. After the PEG procedure, the actual degree distribution of the EERC-LDPC code is

$$\lambda(x) = 0.0003 + 0.3328x + 0.2468x^2 + 0.0012x^3 + 0.4188x^5$$

$$\rho(x) = 0.7367x^5 + 0.2356x^6 + 0.0075x^7 + 0.0140x^8 + 0.0062x^9$$

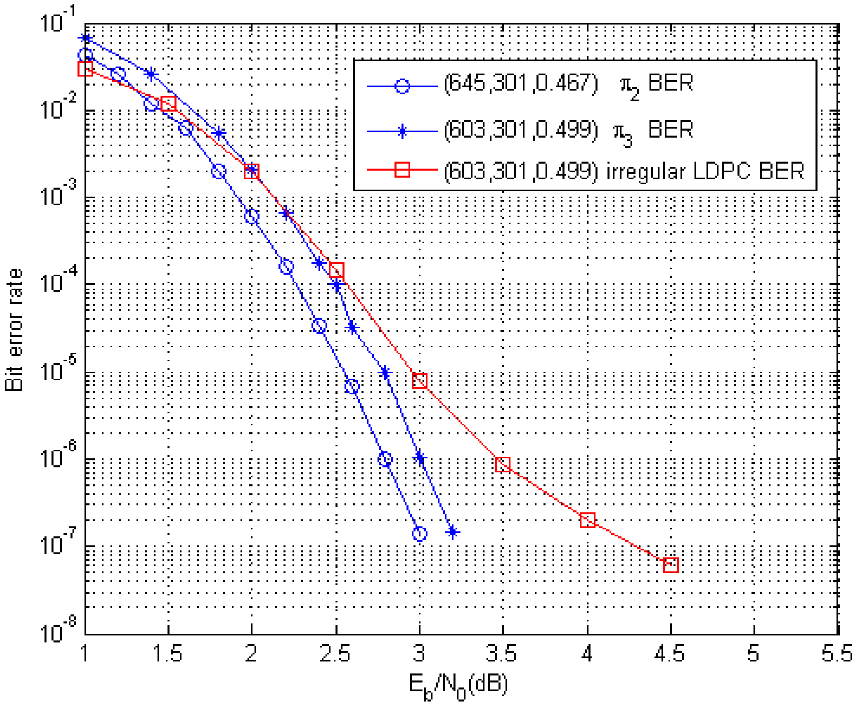

Figure 4.9: Performance comparison of EERC-sGLDPC code π_3 with irregular LDPC code.

and the actual degree distribution of the IRA-LDPC code is

$$\lambda(x) = 0.0003 + 0.3328x + 0.2468x^2 + 0.0012x^3 + 0.4188x^5$$

$$\rho(x) = 0.7012x^5 + 0.2988x^6$$

For LDPC codes also no length-4 cycle exist in H matrix. Fig. 4.11 shows the performance comparison of $\mathcal{C}_h$ and LDPC codes, with $I_{max} = 200$ for three codes. The SCNfirst-PEG causes slight inferior waterfall performance of $\mathcal{C}_h$ compared with LDPC codes, nonetheless in the error floor region the existence of SCNs in hGLDPC codes makes $\mathcal{C}_h$ have a good error floor: $\mathcal{C}_h$ has a 0.7 dB gain over EERC-LDPC codes at BER around 10^{-7}. The low error floor of $\mathcal{C}_h$ is even better reflected in FER, where FERs of IRA-LDPC and EERC-LDPC codes begin to flare out when SNR is greater than 2.2 dB.

At the end of this chapter, we like to point out that more simulation results of EE-LDPC and EE-GLDPC codes using various **n**'s will be presented in Chapter 6, where the tradeoff between **n** and error floor performance is studied.

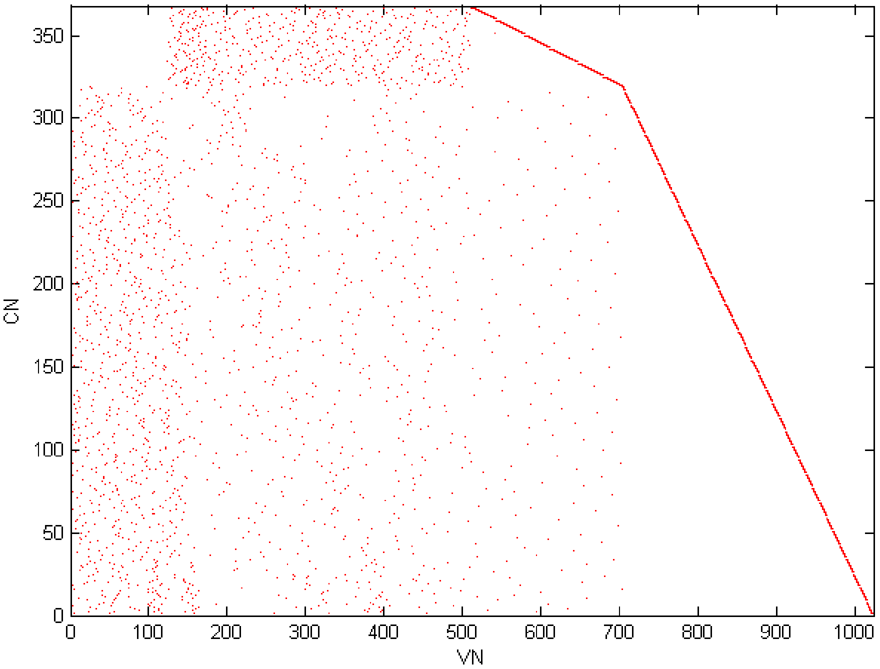

Figure 4.10: The structure of H_{adj} matrix of a $(1024, 512)$ IRA-hGLDPC code.

4.6 Summary

The principles for designing EE-GLDPC codes are discussed in this chapter. The main points are listed below

- EE-LDPC codes can be extended to EE-GLDPC codes, using the generalized EE principle (i.e. the three constraining rules).
- The H_2 matrix in EE-GLDPC codes contains cycles and is more irregular than it is in IRA-LDPC and EERC-LDPC codes.
- EE-GLDPC codes have very similar decoding performance with ordinary GLDPC codes.
- IRA-LDPC and EERC-LDPC codes are special EE codes which are obtainable by adding an extra rule to the generalized EE principle.

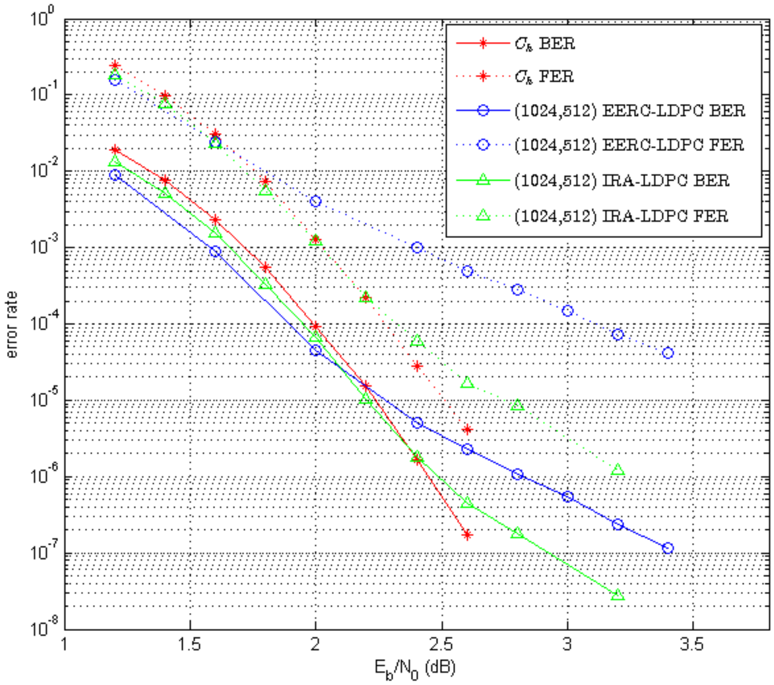

Figure 4.11: Performance comparison of the $(1024, 512)$ hGLDPC code with IRA-LDPC and EERC-LDPC codes.

Chapter 5

Design of EE-DGLDPC Codes

The EE property described in Chapter 4 offers a basis for extending EE-GLDPC codes to EE-DGLDPC codes. However the existence of SVNs requires that the H_2 matrix should be further modified to ensure EE property of DGLDPC codes. In this chapter, we present design principles for EE-DGLDPC codes. The proposed algorithm encompasses a much wider scope of iteratively-decodable block codes, providing an across-the-board solution for designing EE block codes. Simulation results also indicate that, in addition to their EE property, EE-DGLDPC codes also have performance nearly as good as ordinary DGLDPC codes.

5.1 Free edges vs. constrained edges

In previous chapter we have discussed the EE principle for sGLDPC and hGLDPC codes. We now show how it can be extended from hGLDPC to the DGLDPC case. Consider a REP-VN with degree n in GLDPC codes, when it is replaced by a SVN using $C(n, k)$ as a component code. This is equivalent to adding an extra $k - 1$ information bits to the code. If the original REP-VN was in H_1, then after substitution, all the k bits of SVN are information bits. On the other hand, if the original REP-VN was in H_2, then $k - 1$ bits are information bits and one bit is parity bit. In view of this, for EE-DGLDPC codes designed in this work, *information bits are generally not grouped together* as in the cases of EE-LDPC and EE-GLDPC codes.

Now the task is how to appropriately arrange the edges of SVNs in H_2 matrix so that H_{adj} has the EE property. In the following, we call a SVN in H_2 “recovered” iff all the values of k information bits of $C(n,k)$ are known, otherwise it is "not recovered". Let G_{comp} of dimension $k \times n$ be the generator matrix used by the SVN. The SVN has n edges connected to CNs, notice that the message carried by each edge is some linear combination of k information bits of $C(n,k)$.

If only $k-1$ information bits of the component code are known, some edges can still have the messages carried by them recovered, since these messages can be linearly represented by these $k-1$ bits, while others can not. Based on such observation we have the following definitions for *constrained edges* and *free edges*:

Definition: for a SVN in H_2, once $k-1$ information bits of the SVN are chosen, we call an edge e a *free edge* if the message carried by e can be recovered by $(k-1)$ information bits with known values; otherwise it is called a *constrained edge.*

It should be noted that the number of constrained and free edges depends on the specific choice of $k-1$ information bits. For example, consider Hamming $(7,4)$ code with generator matrix

$$G_{comp} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

If we choose the first three information bits of the component code as *known* bits, then the 1st, 3rd and 7th edges are constrained edges and the remaining four edges are free edges. On the other hand, if the 1st, 2nd and 4th information bits are chosen as known bits, we will have four constrained edges and three free edges.

Once constrained and free edges are defined, we can construct EE-DGLDPC codes as follows: the envelope is defined the same way as it is done in EE-hGLDPC codes. For REP-VNs in H_2, the constraining rule 2 in section 4.3 still applies. For SVNs in H_2, one of the constrained edges is one of the elements forming the envelope and the remaining constrained edges need to satisfy constraining rule 2, whereas for free edges they *can be placed anywhere* (above or below the envelope) as long as no parallel edges are formed.

The efficient encodability of EE-DGLDPC codes can be argued as follows: the free edges carry known values during all iterations so they can be placed in any row, since they do not add extra unknowns in parity-check or super check equations. The iterative decoder can still recover all the VNs and SVNs in $\mathcal{V}_i$ after the ith iteration, as it did in hGLDPC codes.

It is worth noting that, the EE principle for DGLDPC codes is a generalized form of EE-hGLDPC codes. In EE-hGLDPC codes, all the VNs are REP-VNs, and notice that the generator matrix of a deg-n REP-VN has the form

$$G_{comp} = \overbrace{[1, 1, ..., 1]}^{n}$$

so all edges are constrained edges and needed to be placed below the envelope. Therefore there is no edge above the envelope for EE-LDPC, EE-sGLDPC and EE-hGLDPC codes, as shown in Fig. (4.2), (4.4), (4.5), (4.7) and (4.10).

5.2 Permutations of edges of SVNs and SCNs

The more intricate structure of SVN, namely free edges and constrained edges, entails additional handling in the EE-DGLDPC iterative decoder design, which is reflected in the requirement for permuting columns for both G_{comp} of SVNs and H_{comp} of SCNs. In this section an external-permutation solution is proposed to tackle this issue.

Permutation of edges of SVN

The requirement to permute the columns of the generator matrix of the SVN is caused by nature of the free edges. Suppose there are l constrained edges and $n - l$ free edges in the SVN. When building the code, due to the additional constraints on graphical properties, such as maximizing the girth, there could be an arbitrary number (0 to $n - l$) of free edges below the envelope. Generally speaking, we can not guarantee finding a fixed $G_{comp} = [G_1, G_2, ..., G_n]$ and cleverly choosing $k - 1$ information bits such that the last $l + f$ columns of G_{comp} *always* contain l constrained edges and f free edges, for all $f = 0, 1, ...n - l$.

Let us assume there are f ($0 \leq f \leq (n - l)$) free edges under the envelope. A permutation σ is mapping $(1, 2, ..., n) \rightarrow (p_1, p_2, ...p_n)$ such that $G' = [G_{p_1}, G_{p_2}, ...G_{p_n}]$ satisfies that the last $(f + l)$ columns correspond to f free edges and l constrained edges. There are two ways to implement the SVN decoder: the first one simply uses G' as the generator matrix of SVN, however by doing so in most cases we can not have a uniform decoder over SVNs. The second way is adding a fixed permutation of order n between the $C(n,k)$-SVN and CNs: when messages arrive from CNs to the SVN, the edges are permuted using σ, and from SVN to CNs it is σ^{-1}, as shown in Fig. 5.1. The external permutation mechanism provides a practical solution for using a uniform G_{comp} over SVNs. In practice, the permutation is simply a one-time rewiring to each SVN's sockets.

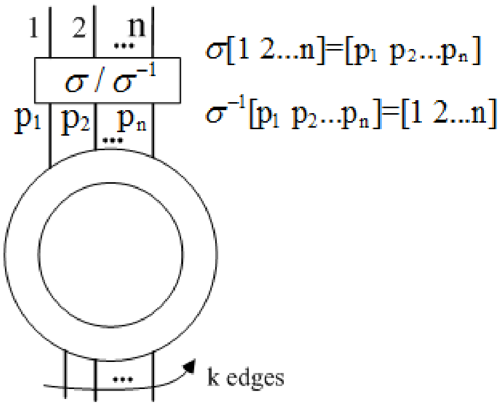
1
2
n
...
σ / σ^{-1}
$\sigma[1\ 2 ... n]=[p_1\ p_2 ... p_n]$
$\sigma^{-1}[p_1\ p_2 ... p_n]=[1\ 2 ... n]$
p_1
p_2
p_n
...
...
k edges

Figure 5.1: The external permutation mechanism in SVN.

As an example of the claim above, consider the G_{comp} in (5.1). Suppose the seven edges of SVN 1 in H_2 matrix has the form $[...1...1..1...1...1...1...1...]^T$, and SVN 2 is $[...1...1..1...1...1...1...1...]^T$, where the red "1" means that this "1" is on the envelope. After the examination we know that no matter how we carefully pick $k-1=3$ information bits in SVN 1 and 2, there exists no such G_{comp} which at the same time satisfies that the last three columns of G_{comp} correspond to three constrained edges, and last four columns correspond to three constrained edges and one free edge. In other words, we either need to use G_{comp} and $G^{'}_{comp}$, where one is the column permutation of the other, to decode SVN 1 and SVN 2 respectively, or one of the SVNs needs the external permutation mechanism to have a uniform decoder.

Suppose SVN 1 chooses the first three information bits of the component code as known bits, and uses G_{comp} as

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

Notice that the last three columns of G_{comp} correspond to three constrained edges. For SVN 2, if the first three information bits of the component code are chosen as known bits, a possible $G^{'}_{comp}$ has the form

$$G^{'}_{comp} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Now the last four columns of $G^{'}_{comp}$ correspond to three constrained edges and one free edge. Now if we want to use G_{comp} to decode SVN 2, based on the relationship between G_{comp} and $G^{'}_{comp}$, then σ is $\sigma[1,2,3,4,5,6,7] = [1,2,3,5,6,7,4]$ and $\sigma^{-1}[1,2,3,4,5,6,7] = [1,2,3,7,4,5,6]$.

Permutation of edges of SCN

The necessity to permute the edges of SCNs is also caused by the free edges of SVNs. In hGLDPC codes, column permutation of $H_{c,i}$ is not necessary since the envelope section of c_i always corresponds to the last $(p_i - q_i)$ columns in $H_{c,i}$. As long as the last $(p_i - q_i)$ columns in $H_{c,i}$ are linearly independent, a uniform H_c is good for SCNs with the same component code. For EE-DGLDPC codes, the free edges can cause that the envelope section of c_i does not corresponds to the linearly-independent columns in $H_{c,i}$; in this case the EE-DGLDPC codes can become no longer efficiently-encodable. In view of this, a permutation of columns is possibly needed to make sure the envelope section has independent columns, this is especially the case when erasure-filling decoding is used: the permutation should be done in such a way that the columns in $H_{c,i}$ corresponding to the envelope section of c_i form an identity matrix.

Similar with SVNs, in order to correctly decode SCNs, we can either use different forms of H_{comp}'s for SCNs, which could possibly increase the overall complexity of the SCN decoders; or have a uniform decoder using the same H_{comp} for all the SCNs as long as the external permutation method is employed. The permutation process is essentially the same with it in SVNs.

On the other hand, if a MAP decoder on SCN is applied, it is possible to permute H_{comp} in a way such that any $p_i - q_i$ consecutive columns are linearly independent, and in such case a uniform H_{comp} can be used without any external permutation. For a Hamming $(15, 11)$ code, such H_{comp}

can be found as

$$H_{comp} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.2}$$

Notice that any consecutive four columns are linearly independent.

Finally, for SPC-CNs, no permutation of H_{comp} is needed since the H_{comp} of a deg-n SPC-CN has the form

$$H_{comp} = \overbrace{[1, 1, ..., 1]}^{n}$$

5.3 Encoding procedures and examples

We first give a simple DGLDPC code example where no SCNs exist in the H_{adj} matrix. Let H_{adj} be

$$H_{adj} = \begin{bmatrix} 1 & 0 & 1 & 0 & \vdots & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & \vdots & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \vdots & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & \vdots & 0 & 0 & 1 & 1 \end{bmatrix} \tag{5.3}$$

where the red column corresponds to a SVN using SPC$(3, 2)$ as its component code, and choose G_{comp} as

$$G_{comp} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{5.4}$$

It is also assumed that the first information bit in the SVN is the information bit of DGLDPC code. After column expansion we have

$$H_{SPC} = \begin{bmatrix} 1 & 0 & 1 & 0 & \vdots & 1 & 0 & \mathbf{10} & 0 \\ 0 & 1 & 1 & 1 & \vdots & 1 & 1 & \mathbf{00} & 0 \\ 0 & 1 & 0 & 1 & \vdots & 0 & 1 & \mathbf{11} & 0 \\ 1 & 1 & 1 & 0 & \vdots & 0 & 0 & \mathbf{01} & 1 \end{bmatrix} \tag{5.5}$$

Notice that the values of “1”s in the red column with bold fonts are known (since these are information bits of the DGLDPC code), and we can see that H_{SPC} is essentially a $(9,5)$ IRA-LDPC code with one information bit in H_2 matrix. An iterative erasure decoder can encode this DGLDPC code in four iterations.

In the next example, we present a simple $(14,7)$ rate-0.5 EE-DGLDPC code with one SCN in it. Let H_{adj} be

$$H_{adj} = \begin{bmatrix} {\color{red}\mathbf{1}} & 0 & 1 & 1 & \vdots & 1 & 0 & 0 & 0 & 0 & {\color{red}\mathbf{1}} & 0 \\ {\color{red}\mathbf{0}} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \vdots & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & {\color{red}\mathbf{1}} & \mathbf{0} \\ {\color{red}\mathbf{1}} & 0 & 1 & 0 & \vdots & 0 & 1 & 0 & 0 & 1 & {\color{red}\mathbf{0}} & 0 \\ {\color{red}\mathbf{0}} & 1 & 1 & 1 & \vdots & 0 & 0 & 1 & 0 & 1 & {\color{red}\mathbf{1}} & 0 \\ {\color{red}\mathbf{1}} & 1 & 0 & 0 & \vdots & 0 & 0 & 0 & 1 & 0 & {\color{red}\mathbf{1}} & 1 \end{bmatrix} \tag{5.6}$$

where the first and the tenth columns are SVNs, denoted by SVN 1 and SVN 2 respectively, and they use $G_{comp,1} = [1\,1\,0; 0\,1\,1]$ and $G_{comp,2} = [1\,1\,0\,0; 0\,1\,1\,0; 0\,0\,1\,1]$, respectively. Also assume the first two information bits in the SVN 2 are information bits of the DGLDPC code. The second row is a SCN using $(6,3)$ shortened Hamming code with H_{comp} as

$$H_{comp} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Let $\boldsymbol{u} = [u_1, u_2, ..., u_7]$ be the information sequence. u_1 and u_2 belong to SVN 1 and u_6 and u_7 belong to SVN 2. Let $\boldsymbol{p} = [p_1, p_2, ..., p_7]$ be the parity-bit sequence, and p_6 belongs to SVN 2. After encoding, the codeword $\boldsymbol{c}$ is a length-14 sequence with

$$\boldsymbol{c} = [u_1, u_2, u_3, u_4, u_5, p_1, p_2, p_3, p_4, p_5, u_6, u_7, p_6, p_7]$$

For the sake of clarity, we only do the column expansions on H_{adj}. After column expansions, $\tilde{H}$ is

$$\tilde{H} = \left[\begin{array}{ccccccccccccc} u_1u_2 & u_3 & u_4 & u_5 & \vdots & p_1 & p_2 & p_3 & p_4 & p_5 & u_6u_7p_6 & p_7 \\ 10 & 0 & 1 & 1 & \vdots & 1 & 0 & 0 & 0 & 0 & 100 & 0 \\ \mathbf{00} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \vdots & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{110} & \mathbf{0} \\ 11 & 0 & 1 & 0 & \vdots & 0 & 1 & 0 & 0 & 1 & 000 & 0 \\ 00 & 1 & 1 & 1 & \vdots & 0 & 0 & 1 & 0 & 1 & 011 & 0 \\ 01 & 1 & 0 & 0 & \vdots & 0 & 0 & 0 & 1 & 0 & 001 & 1 \end{array}\right] \tag{5.7}$$

Suppose $\boldsymbol{u} = 0110101$. The encoding process is as follows

- $\star$ 1st iteration: $u_1 = 0$, $u_4 = 0$, $u_5 = 1$ and $u_6 = 0$ are used in the first SPC equation to determine the value of p_1. We have $p_1 = 1$.

- $\star$ 2nd iteration: $u_3 = 1$, $p_1 = 1$ and $u_6 \oplus u_7 = 1$ are regarded as known bits in H_{comp} (they correspond to the 1st, 2nd and 6th columns of H_{comp}, respectively) to calculate p_2, p_3 and p_4. We obtain $p_2 = 0$, $p_3 = 1$ and $p_4 = 0$.

- $\star$ 3rd iteration: $u_1 \oplus u_2 = 1$, $u_4 = 0$ and $p_2 = 0$ are involved in the third SPC equation to find p_5. $p_5 = 1$.

- $\star$ 4th iteration: $u_3 = 1$, u_4=0, $u_5 = 1$, $p_3 = 1$ and $p_5 = 1$, therefore $u_7 \oplus p_6 = 0$, and $p_6 = 1$.

- $\star$ 5th iteration: $u_2 = 1$, $u_3 = 1$, $p_4 = 0$ and $p_6 = 1$, so $p_7 = 1$. After the 5th iteration, all the 7 parity bits are recovered.

- $\star$ Output: $\boldsymbol{c} = [u_1, u_2, u_3, u_4, u_5, p_1, p_2, p_3, p_4, p_5, u_6, u_7, p_6, p_7] = [01101101010111]$;

Note that this EE-DGLDPC code is still systematic, nonetheless the information bits are not clustered together but rather dispersed among the codeword bits. In addition, the H_{adj} structure makes the p_2, p_3 and p_4 not correspond to the last three columns in H_{comp}, so it is necessary to make sure that the 3rd, 4th and 5th columns of H_{comp} are linearly independent; otherwise, if we use H_{comp} as follows

$$H_{comp} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

then in the second iteration the values of p_2, p_3 and p_4 can not be determined, even if a MAP decoder is used; in such case this DGLDPC code no longer has the EE property. Therefore, this simple EE-DGLDPC example also verifies the necessity of permutations of SCN edges.

5.4 Simulation results

We next present simulation results of EE-DGLDPC codes and their comparisons with ordinary DGLDPC codes and irregular LDPC codes. All the optimized profiles are obtained by EXIT charts and DE. It should also be noted that the EXIT chart provides an optimistic estimation of the threshold [73]; the actual threshold is slightly higher.

Table 5.1 lists the profiles used in the simulation. DGLDPC$_1$, LDPC$_1$ and DGLDPC$_3$ are rate-0.5 profiles and DGLDPC$_2$ and LDPC$_2$ are rate-0.75 profiles. The generator matrix G_{comp} of SPC-SVNs assume the systematic form, and $\mathbf{n}$ is chosen as the all-one vector. When decoding, the sum-product algorithm (SPA) is used when updating REP-VNs and SPC-CNs, and the MAP decoder based on the component code trellis [76] is adopted for the SCN decoding. As for SPC-type SVNs, a simple update rule exists [50]. The maximum number of iteration I_{max} is set to 200, and BER is calculated based on the information bits.

We first consider moderate-blocklength, rate 0.5 codes. Length $N = 1200$ EE-DGLDPC and irregular LDPC codes are designed by PEG using profile DGLDPC$_1$ and LDPC$_1$, respectively. For DGLDPC code, the number of SPC$(3, 2)$ SVNs is 246 and the adjacency matrix H_{adj} is 600×954. The last column of H_{adj} of EE-DGLDPC code corresponds to a deg-1 VN in order to fulfill the EE property, and all the SVNs are placed in the H_2 part. Fig. 5.2 shows the structure of H_{adj}; it can be observed that there are many free edges above the envelope.

Fig. 5.3 gives the performance comparisons of EE-DGLDPC, ordinary DGLDPC and irregular LDPC codes. The waterfall performance of EE-DGLDPC code is slightly better than irregular LDPC codes, which agrees with the threshold difference. The ordinary DGLDPC code does not

Table 5.1: Profiles and thresholds of LDPC and DGLDPC codes used in the simulations.

	DGLDPC$_1$	LDPC$_1$	DGLDPC$_2$	LDPC$_2$	DGLDPC$_3$	DGLDPC$_4$
Variable Nodes						
REP(2,1)	0.00180	0.33311	0.00132	0.27957	0.10005	0.15046
REP(3,1)	0.37785	0.24636	0.07021	0.00054	0.00085	0.00118
REP(4,1)	0.00010	0.00119	0.77462	0.71989	0.35247	0.03180
REP(6,1)	0.41592	0.41934			0.15612	0.33579
SPC(3,2)	0.20433					0.48077
SPC(4,3)			0.15385			
SPC(5,4)					0.39051	
Check Nodes						
SPC(5,4)	0.00058	0.00005			0.00025	
SPC(6,5)	0.98065	0.70142			0.00244	0.02274
SPC(7,6)	0.01749	0.29240			0.22403	0.27757
SPC(8,7)	0.00069	0.00468			0.00047	
SPC(9,8)	0.00059	0.00145			0.00112	
SPC(10,9)			0.00044	0.00124		
SPC(11,10)			0.00244	0.00644		
SPC(12,11)			0.98696	0.46560		
SPC(13,12)			0.00748	0.52171		
SPC(14,13)			0.00214	0.00202		
SPC(15,14)			0.00054	0.00299		
Ham.(15,11)					0.77169	0.69969
Threshold (by EXIT)						
	0.552 dB	0.567 dB	2.042 dB	2.059 dB	0.417 dB	0.429 dB

perform as well as EE-DGLDPC code in the waterfall region, but it has a slightly a steeper error curve at higher SNRs. All three codes do not show error floors down to BER= 10^{-7}.

Next we show the performance of $N = 1200$ and $R = 0.75$ high-rate codes built by PEG using profile DGLDPC$_2$ and LDPC$_2$. The number of SPC$(4, 3)$ SVNs is 139 and the dimension of H_{adj} is 300×922. Likewise, all the SVNs are placed in H_2. The performance of EE-DGLDPC, ordinary DGLDPC and irregular LDPC codes are shown in Fig. 5.4. All three codes have nearly the same waterfall curves (which also agrees with threshold analysis) and show good error floor performance. Nonetheless under the close examination it can be observed that EE-DGLDPC code actually exhibits a slightly better waterfall curve and a slightly worse BER at highest SNR.

In the next experiment we demonstrate the advantages of EE-DGLDPC codes over LDPC codes at longer block lengths. We designed EE-DGLDPC codes with profile DGLDPC$_3$ and irregular LDPC codes with LDPC$_1$, of lengths 10^4 and 5×10^4. All codes are randomly-constructed except that parallel edges and length-4 cycles are avoided. The threshold of LDPC$_1$ is 0.15 dB worse than that of DGLDPC$_3$, and such gap is also reflected in Fig. 5.5: EE-DGLDPC codes outperform LDPC codes over all SNRs; at BER= 3×10^{-5}, length 5×10^4 EE-DGLDPC code outperforms about 0.09

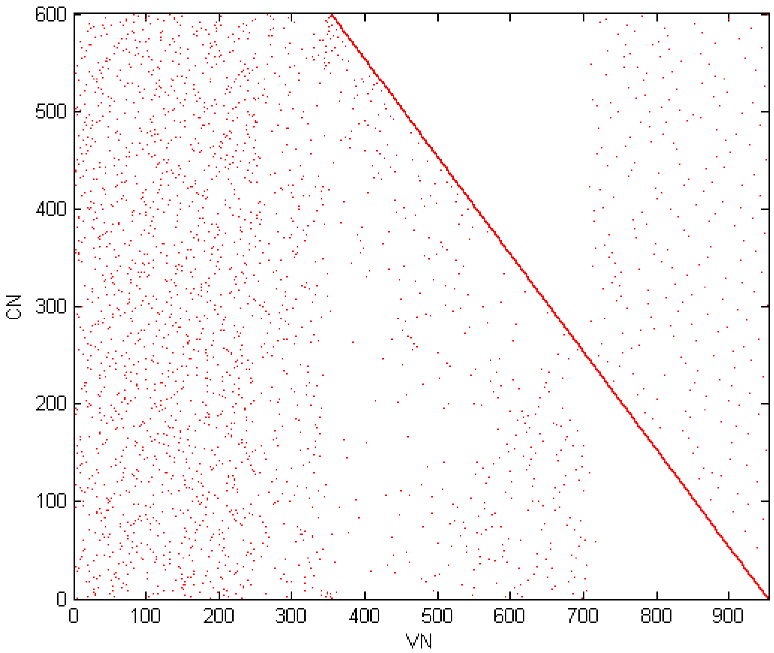

Figure 5.2: The structure of H_{adj} of a length-1200, rate-0.5 EE-DGLDPC code with no SCNs.

dB over the LDPC code; for length 10^4 and 5×10^4 code, LDPC codes show serious error floors, whereas EE-DGLDPC codes have much better error floor performance compared with LDPC codes. The error floor advantages of EE-DGLDPC codes over irregular LDPC codes becomes more obvious when FER is taken into account.

As a last experiment, we compare the EE-DGLDPC code with LDPC code at a short blocklength. A length-1200 rate-0.5 IRA-LDPC code and a EERC-LDPC code of the same length and rate are built by PEG according to profile LDPC$_1$. The EE-DGLDPC code is built by GR-SCNfirst-PEG using profile DGLDPC$_4$, and the dimension of H_{adj} is 234×782. There are 418 SVNs and 122 SCNs in H_{adj}. Fig. 5.6 shows the structure of H_{adj}, in Fig. 5.6 we can see the free edges above the envelope, envelope sections of SCNs and SVNs, and the effect of GR-SCNfirst-PEG: all the SPC-SVNs are connected to SCNs. Finally, no length-4 cycles exist in these three codes.

Fig. 5.7 shows the performance comparison of three codes: though the EE-DGLDPC code has a profile with better decoding threshold than the LDPC profiles, the connectivity constraint enforced by GR-SCNfirst-PEG causes some degradation in the waterfall region, in exchange for a lower error floor. At high SNR, EE-DGLDPC achieves about 0.5 dB gain over IRA-LDPC codes and 0.8 dB

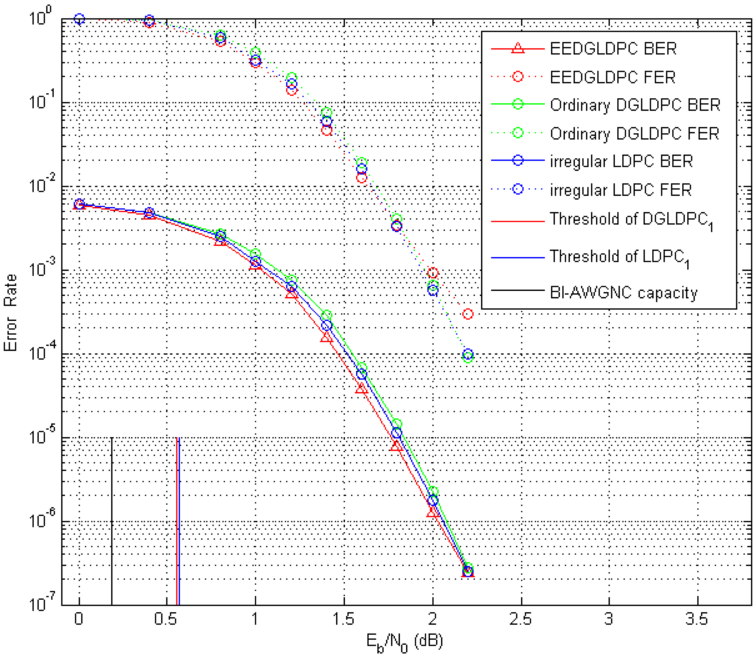

Figure 5.3: Performance comparison of length-1200, rate-0.5 irregular LDPC, EE-DGLDPC with no SCNs, and ordinary DGLDPC codes.

gain over EERC-LDPC codes, at BER=2×10^{-7}. The low error floor of EE-DGLDPC code is more visible on FER curve: at $\frac{E_b}{N_0} = 2.5$ dB, the FER of EE-DGLDPC code is more than two orders of magnitude lower than EERC-LDPC code.

5.5 Summary

In this chapter designing EE-DGLDPC codes are discussed. The main points are listed below:

- For a $C(n, k)$-SVN in the H_2 matrix, it corresponds to $k - 1$ information bits and one parity bit of the DGLDPC code. Of n edges in the Tanner graph, some are free edges and the others are constrained edges.

- When building an EE-DGLDPC code, constrained edges must be placed below the envelope whereas free edges can be placed anywhere.

- Due to the existence of free edges, the permutation of edges of SVNs and SCNs are sometimes necessary to ensure the EE property.

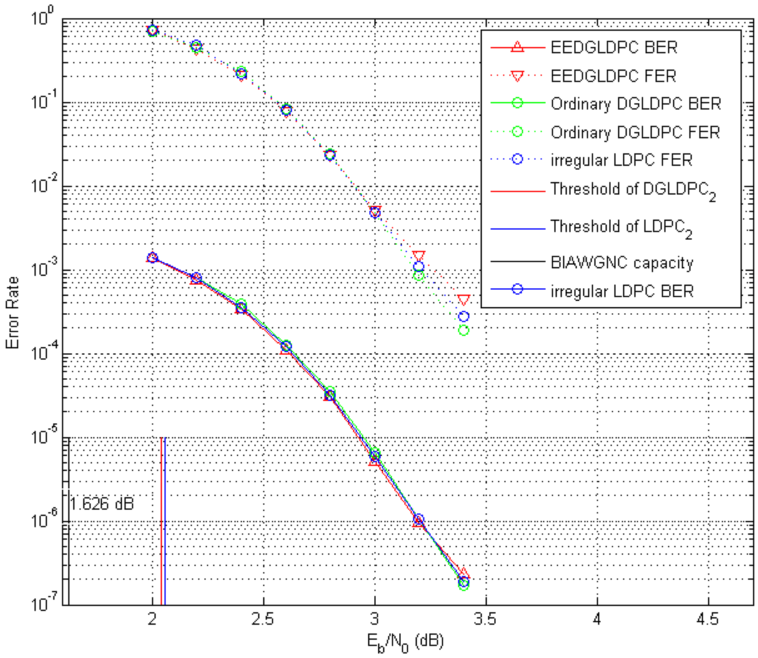

Figure 5.4: Performance comparison of length-1200, rate-0.75 irregular LDPC, EE-DGLDPC with no SCNs, and ordinary DGLDPC codes.

- EE-DGLDPC codes have a very similar decoding performance with ordinary LDPC codes.

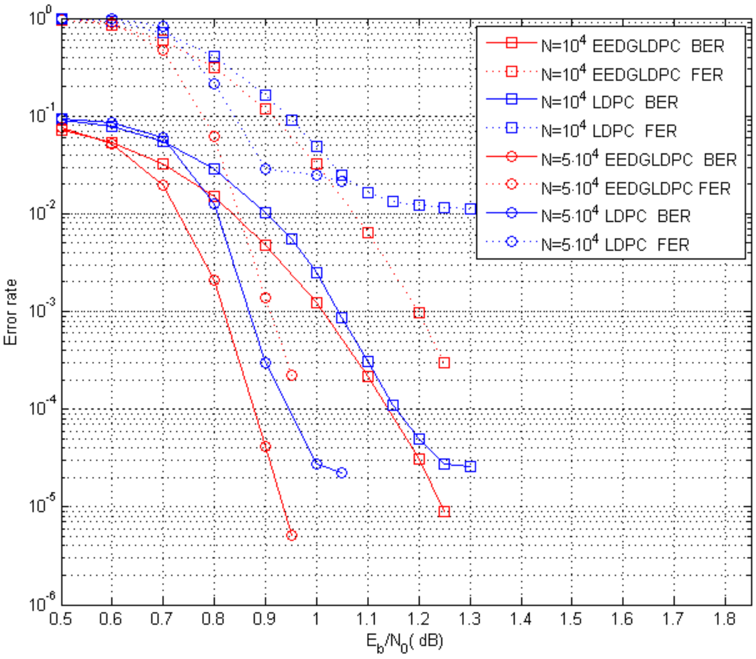

Figure 5.5: Performance comparison of rate-0.5, length 10^4 and 5×10^4 EE-DGLDPC and irregular LDPC codes.

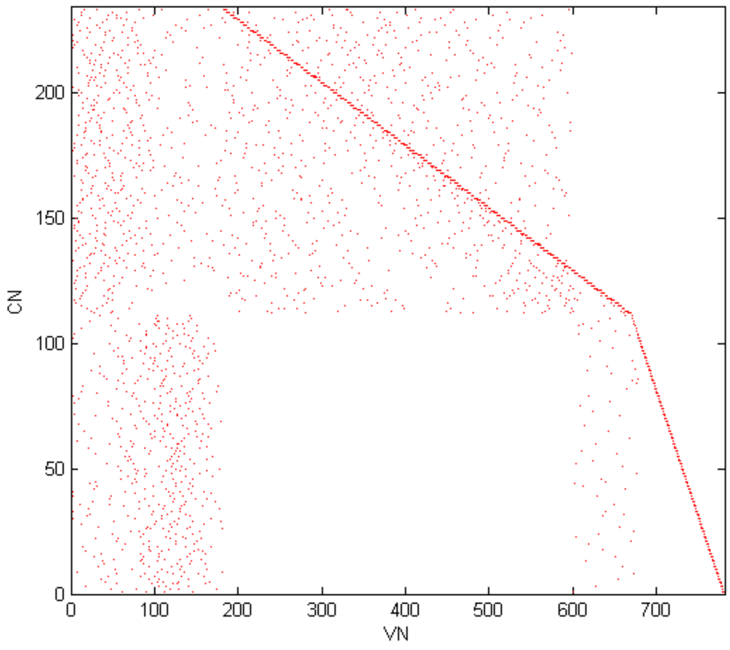

Figure 5.6: The structure of H_{adj} of a length-1200 rate-0.5 EE-DGLDPC code with SCNs.

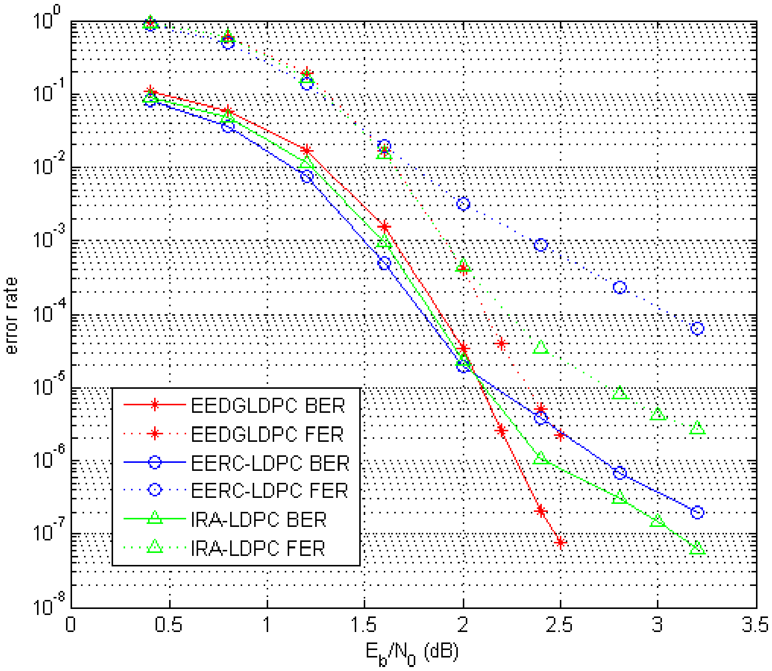

Figure 5.7: Performance comparison of the length-1200 rate-0.5 EE-DGLDPC code with SCNs, with IRA-LDPC and EERC-LDPC codes.

Chapter 6

Encoding Speed and Error Floor

Theoretically, we can have a $\mathbf{n}$ of any length as long as $\sum_{i=1}^{D} n_i = M_C$, so designing EE codes with any encoding time-complexity, even with $O(1)$ is always possible. Nonetheless the shorter $\mathbf{n}$ is, the more restrictions on the available positions where "1"s can be placed will be, which implies the potential poorer error floor performance. In this chapter, we present both theoretical analysis and simulation results to verify this claim. It is also demonstrated that the error floor performance of EE-GLDPC codes are relatively more robust to the the length change of $\mathbf{n}$.

6.1 The tradeoff between encoding speed and error floor

There are two major factors making EE codes subject to poor error floor effects: deg-1 VNs and a dispersive CN distribution. The number of deg-1 VNs can be calculated using (4.11), given $\mathbf{n} = [n_1, n_2, ...n_D]$; whereas the dispersive CN distribution is produced by a fast-diminishing sequence $\mathbf{n}$ (e.g. $\mathbf{n}$ for EERC codes) [60]. Note that after appending all the constrained edges of all VNs in $\mathcal{V}_i$ $(i = 1, 2, ..., D)$, there is an average number of $\sum_{j=1}^{j=|\mathcal{V}_i|} (r_j - 1)/\sum_{j=i+1}^{j=D} n_j$ edges connected to each CN in $\mathcal{C}_l,\ l = i+1, i+2, ...D$, where r_j is the number of constrained edges of jth VN in $\mathcal{V}_i$. Therefore the last several CNs tend to have very large degree if the elements in $\mathbf{n}$ diminish quickly, resulting in a dispersive CN distribution. Such phenomenon can be alleviated by increasing n_D, however doing so will inevitably increase the number of deg-1 VNs.

Based on the analysis above, we see the competing effects of fast encoding (a short recovery vector) and error performance in the error floor region. Generally speaking, a shorter length of

n will imply either an increased number of deg-1 VNs or a more dispersive CN distribution, or both. It is difficult to give a quantitative analysis for the dispersiveness, especially for GLDPC and DGLDPC codes, due to the random nature of H_2 matrix. As a qualitative illustration of the tradeoff, we list below in Table 6.1 several **n**'s and their corresponding parameters, for EE-LDPC codes using the extra rule (described in subsection 4.3) for the H_2 matrix, when $M_C = 1024$. It can be observed in Table 6.1 that a smaller value of D involves either an increased value of n_D, or a larger degree of last CN, both of which are unfavorable for the error floor performance.

Table 6.1: Recovery vectors and their corresponding parameters for LDPC codes.

$\mathbf{n}$	D, or $\lvert\mathbf{n}\rvert$	n_D	last CN's deg(to VNs in $\mathbf{H_2}$)
$[1023, 1]$	2	1	1024
$[1000, 20, 3, 1]$	4	1	50
EERC:$[512, 256, ..., 4, 2, 1, 1]$	11	1	11
$[\overbrace{32, 32, ...32}^{31}, 16, 8, 4, 2, 1, 1]$	37	1	7
$[\overbrace{64, 64, ..., 64}^{16}]$	16	64	2
$[\overbrace{8, 8, ..., 8}^{128}]$	128	8	2
IRA:$[1, 1, ..., 1]$	1024	1	2

6.2 Simulation results

In this section, experiments are carried out to study the effect of **n**, for LDPC, GLDPC and DGLDPC codes. In all experiments, the MAP decoder is used for VNs and CNs (i.e. SPA for LDPC codes). The maximum number of decoding iterations is set at $I_{max} = 200$. BER is calculated taking into account both the erroneous information and parity bits, whereas FER counts errors anywhere in the codeword.

6.2.1 EE-LDPC codes

First, we study standard LDPC codes with maximum VN degree $dvmax = 6$. The optimized degree distribution is

$$\begin{aligned} \lambda(x) &= 0.33311x + 0.24636x^2 + 0.00120x^3 + 0.41933x^5 \\ \rho(x) &= 0.00005x^4 + 0.70142x^5 + 0.29240x^6 + 0.00468x^7 + 0.00145x^8 \end{aligned} \tag{6.1}$$

It should be noted that the CN degree randomness caused by PEG and the number of deg-1 VNs (prescribed by $\mathbf{n}$'s) will slightly modify the degree distribution of actual EE codes. In the first experiment length-1000, rate-0.5 codes with *constant* elements in $\mathbf{n}$ are built. Let $L_{a1}, L_{a4}, L_{a8}, L_{a16}, L_{a64}$ and L_{a128} be the EE-LDPC codes built by $\mathbf{n} = [1, 1, ...], [4, 4, ...], ...[64, 64, ...]$ and $[128, 128, ...]$, respectively. Due to the issue of divisibility in some cases n_D could be different from $n_i, i = 1, 2, ..., (D-1)$. In Fig. 6.1 the decoding performance of these codes is shown. It can be observed that with the increase of elements in $\mathbf{n}$ (equivalently, shortening of $\mathbf{n}$), error floors appear to be more serious; this is primarily attributed to the increased number of deg-1 VNs.

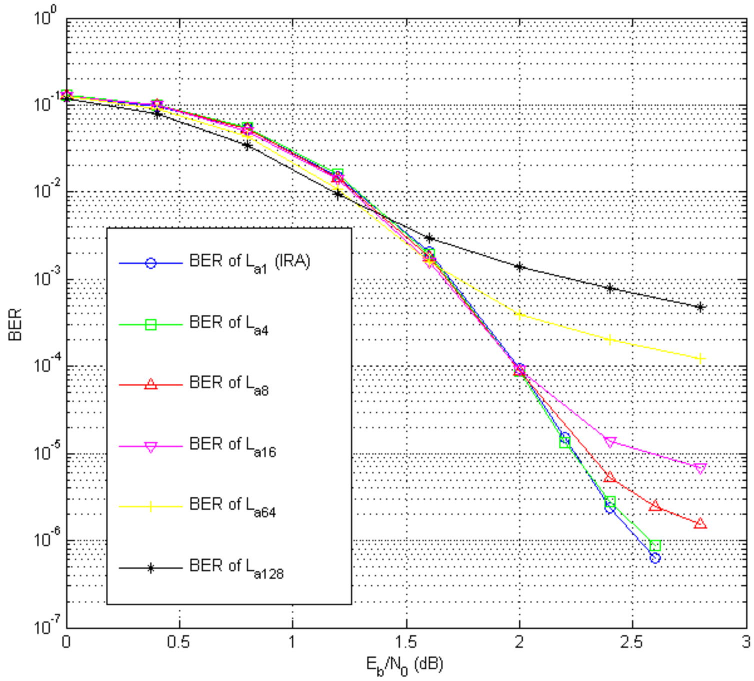

Figure 6.1: Performance comparison of length-1000 EE-LDPC codes using constant elements in $\mathbf{n}$.

In the second experiment we study length-1024 rate-0.5 LDPC codes with irregular elements in $\mathbf{n}$. We designed 8 codes $L_i, i = 0, 1, ..., 7$ with different length $\mathbf{n}$'s, based on the profile in (6.1). The parameters of these codes are listed in Table 6.2.

Table 6.2: EE-LDPC codes using different length $\mathbf{n}$'s and their parameters.

	$\mathbf{n}$	$D(\|\mathbf{n}\|)$	n_D
L_0	ordinary LDPC	—	—
L_1	$[128, 128, 128, \overbrace{1, 1..., 1}^{128}]$	131	1
L_2	$[128, 128, 128, \overbrace{2, 2..., 2}^{63}, 1, 1]$	68	1
L_3	$[128, 128, 128, 64, 32, 16, 8, 4, 2, 1, 1]$	11	1
L_4	EERC:$[256, 128, 64, ..., 4, 2, 1, 1]$	10	1
L_5	$[128, 128, 64, 64, 32, 32, 32, 16, 8, 8]$	10	8
L_6	$[51, 51, 51, 51, 51, 51, 51, 51, 51, 53]$	10	53
L_7	$[256, 128, 64, 32, 32, 32]$	6	32

The performance of these 8 codes are shown in Fig. 6.2. It can be seen that when D is relatively long (as in L_1 and L_2), there is little performance loss at high SNR, compared with ordinary LDPC codes. However, when D is small, it is likely for codes to exhibit serious error floors, as in L_4, L_6 and L_7. For the L_4 (EERC) code it is due to the dispersion of CN distribution and for L_6 and L_7 it is the deg-1 VNs.

Once D is fixed, some improvements on error floor performance can still be achieved when the code is carefully designed: L_5 has the same D with L_4 and L_6, and it shows a much better error floor performance than L_4 and L_6 codes. In practice, empirical study may be needed to identify the code with the best error floor behavior. It is also worth noting that, though some EE codes do not show obvious error floors down to a certain BER (e.g. L_1 and L_2 at BER$= 10^{-6}$), they are more likely to show error floors than ordinary codes in the region beyond simulation capability, due to the position restrictions imposed by $\mathbf{n}$ on them.

6.2.2 EE-GLDPC codes

EE-sGLDPC codes: We first study the effect of $\mathbf{n}$ on EE-sGLDPC codes. We choose the profile as

$$\begin{aligned} &\lambda(x) = 0.0667 + 0.7333x + 0.2000x^2 \\ &\rho(x) = x^{14} \ (Hamming(15, 11)) \end{aligned} \tag{6.2}$$

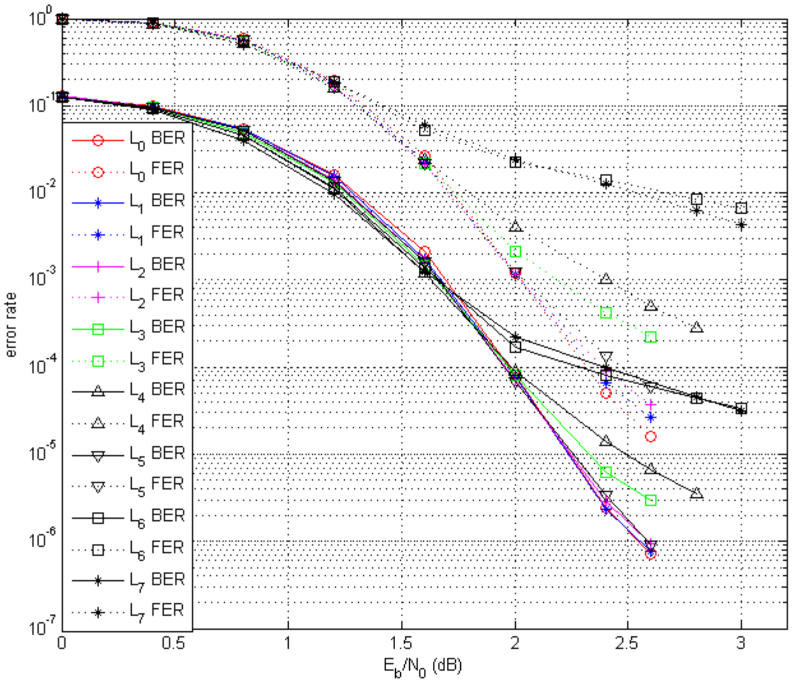

Figure 6.2: Performance comparison of length-1024 EE-LDPC codes using irregular elements in $\mathbf{n}$.

The average VN degree $\overline{d_v} = 2$, and the code rate is $R = 0.4667$. We designed $(960, 448)$ EE-sGLDPC codes with various $\mathbf{n}$'s. The dimension of H_{adj} is 128×960, and $\mathbf{n}$'s are $[1, 1, ..., 1]$, $[2, 2, ..., 2][4, 4, ..., 4]$, $[8, 8, ..., 8]$, $[16, 16, ..., 16]$, $[64, 32, 16, 8, 4, 2, 1, 1]$ and $[32, 32, 16, 16, 16, 8, 4, 4]$. The maximum number of iterations is set to $I_{max} = 50$.

Fig. 6.3 shows the simulation results of these EE-sGLDPC codes. It can be observed that, except for $\mathbf{n} = [16, 16, ..., 16]$, all the EE-sGLDPC codes have almost the same decoding performance, though the lengths of $\mathbf{n}$'s are quite different from each other. This indicates that the error floor performance of EE-sGLDPC codes can remain good for a wide range of $\mathbf{n}$ lengths, this is due to the error correcting capabilities of SCNs. On the other hand, if the length of $\mathbf{n}$ is too short, EE-sGLDPC codes also shows serious error floors: for the code using $\mathbf{n} = [16, 16, ..., 16]$, it is due to the 64 deg-1 VNs in it.

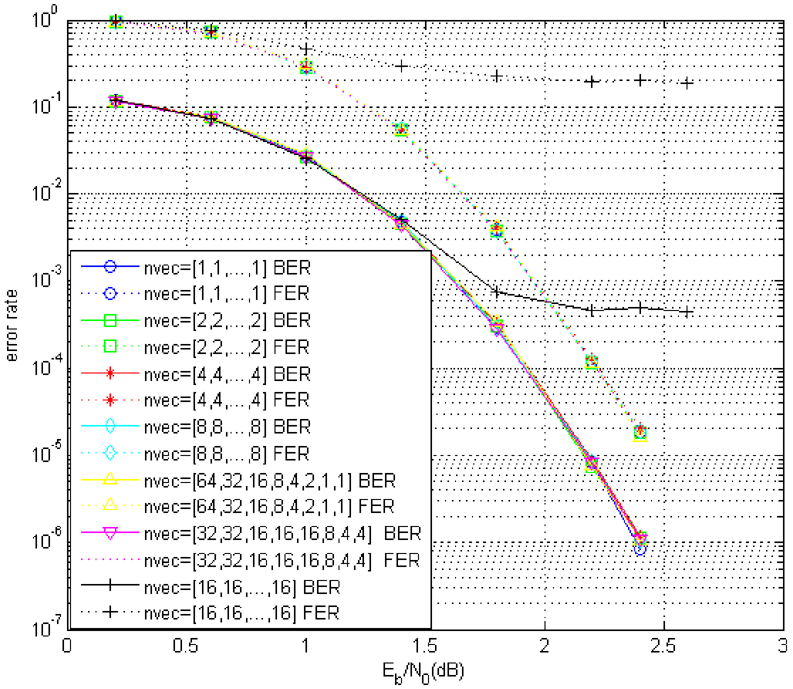

Figure 6.3: Performance of $(960, 448)$ EE-sGLDPC codes with various $\mathbf{n}$'s.

EE-hGLDPC codes: We also study the performance of EE-hGLDPC codes under various $\mathbf{n}$'s. The optimized profile is

$$\lambda(x) = 0.3893x + 0.2682x^2 + 0.0025x^3 + 0.0026x^4 + 0.3374x^5$$

$$\rho(x) = 0.0014x^5 + 0.6311x^6 + 0.1011x^7 + 0.0037x^8 + 0.0166x^9 + 0.2460x^{14} \ (Hamming(15, 11))$$

We first constructed a set of 4 codes: GL_{250}, GL_{500}, GL_{1000} and GL_{2000}, of length $250, 500, 1000$ and 2000 respectively. The $\mathbf{n}$'s are chosen as $[1, 1, ..., 1]$, $[2, 2, ..., 2]$, $[4, 4, ..., 4]$ and $[8, 8, ..., 8]$, respectively, so each code has encoding speed $D = 89$ cycles. (Note this provides a family with $\mathcal{O}(1)$ encoding time, i.e. independent of code length N.) Fig. 6.4 gives the performance of these codes. It can be seen that all codes show good error floor performance.

To further increase the encoding speed, another set of 4 codes: GL'_{250}, GL'_{500}, GL'_{1000} and GL'_{2000} are designed, the only difference with their previous counterparts is $\mathbf{n}$'s are $[4, 4, ..., 4, 5]$, $[8, 8, ..., 8, 10]$, $[16, 16, ..., 16, 20]$ and $[32, 32, ..., 32, 40]$, respectively, thus $D = 22$ for each code. From Fig. 6.5 we see that the new codes also have very good error floor performance, and in fact performance barely

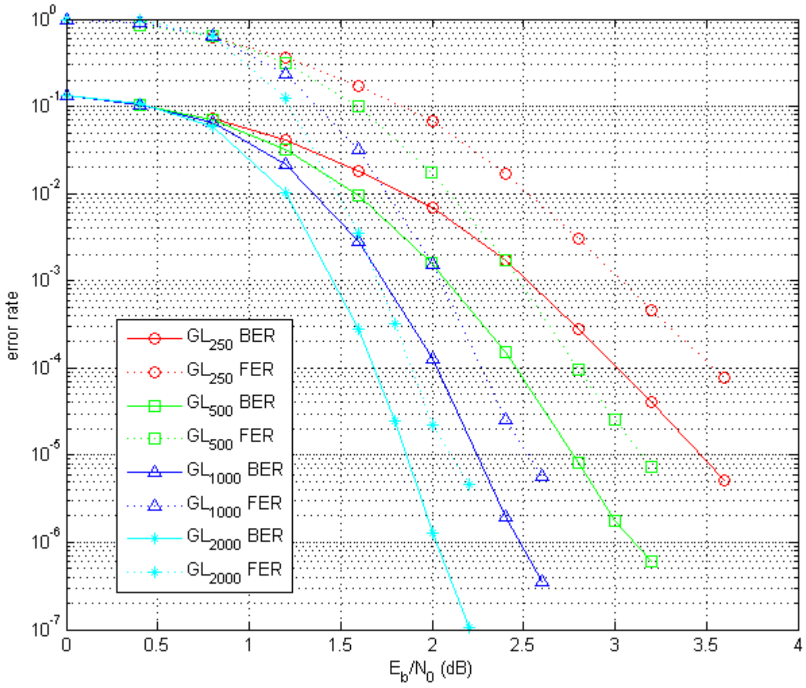

Figure 6.4: Performance of four length-1000 EE-hGLDPC codes with fixed encoding speed $D = 89$.

changes from those with longer encoding time, due to the error correcting capability of Hamming codes.

6.2.3 EE-DGLDPC codes

In this experiment, rate-0.5 EE-DGLDPC codes are built according to the following optimized profiles, where SPC(5,4) codes (G_{comp} matrix in systematic form) are used as SVNs and Hamming(15,11) as SCNs.

$$\lambda(x) = 0.1000x + 0.0009x^2 + 0.3525x^3 + 0.1561x^5 + 0.3905x^4 \ (SPC(5,4))$$

$$\rho(x) = 0.0003x^4 + 0.0024x^5 + 0.2240x^6 + 0.0005x^7 + 0.0011x^8 + 0.7717x^{14} \ (Hamming(15,11))$$

Four length-10000 EE-DGLDPC codes DGL_1, DGL_2, DGL_3 and DGL_4 are designed using the above distribution. The codes are randomly-constructed except that all length-4 cycles are removed. The dimension of adjacency matrix H_{adj} is 1763×5086. The recovery vector $\mathbf{n}$'s are chosen as $[1, 1, ..., 1]$, $[2, 2, ..., 2, 1, 1, 1]$, $[4, 4, ..., 4, 3, 2, 2]$ and $[16, 16, ..., 16, 19]$, respectively. Fig. 6.6 gives the

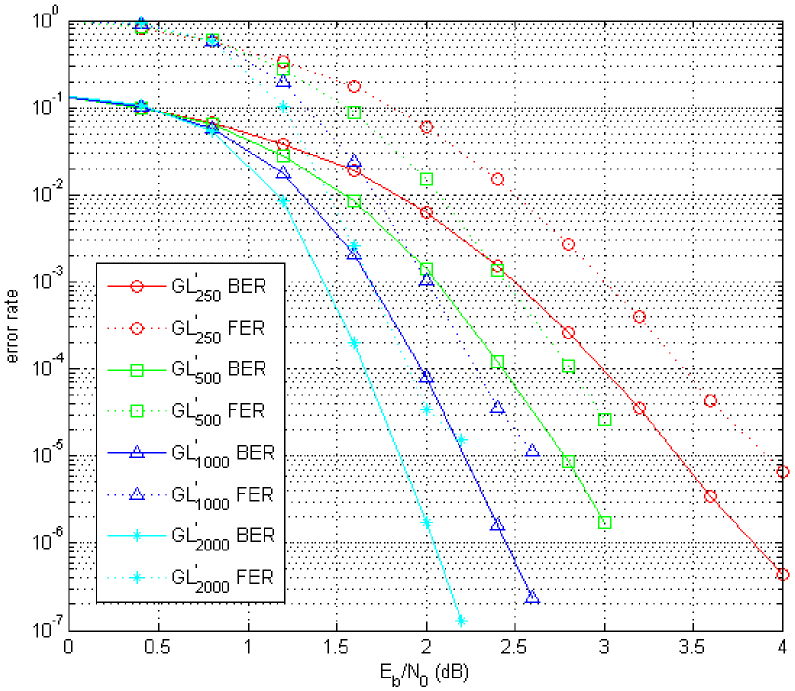

Figure 6.5: Performance of four length-1000 EE-hGLDPC codes with fixed encoding speed $D = 22$.

performance of four EE-DGLDPC codes.

It is interesting to observe in Fig. 6.6 that with the decrease of D, the waterfall performance of EE-DGLDPC codes degrades slightly; this is probably because DGLDPC codes contains more types of nodes, and the threshold of the ensemble is sensitive to the types of connectivity of edges (as in multi-edge type codes). Again, a too-small D will significantly degrade the code performance.

6.3 Summary

The main points of this chapter are listed below:

- The recovery vector $\mathbf{n}$ can be of any length to make the encoding with any speed.
- A faster encoding speed tends to cause a poorer the error floor performance.
- The error floor performance of EE-GLDPC codes is relatively robust to the length change of $\mathbf{n}$.

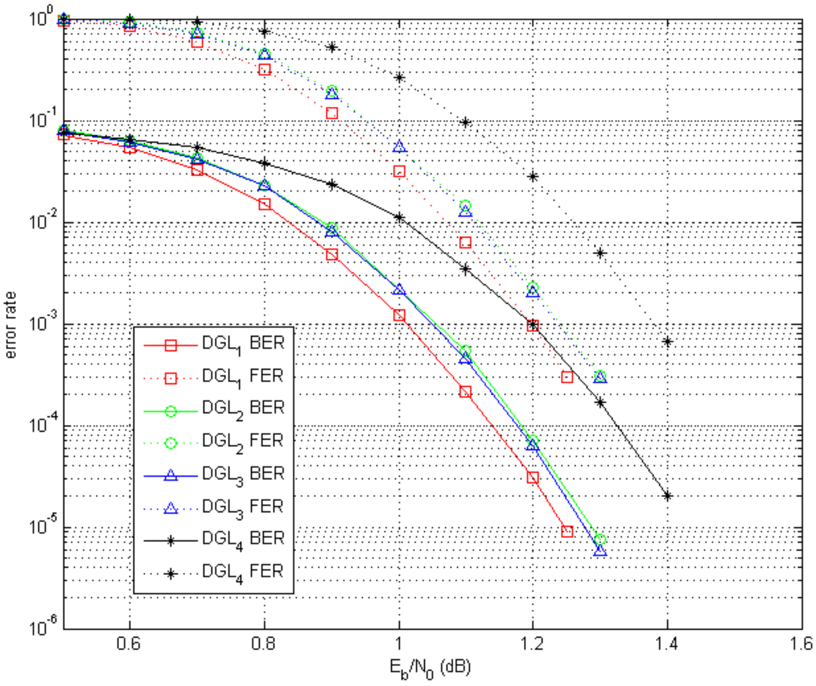

Figure 6.6: Performance of four length-10^4 EE-DGLDPC codes with various $\mathbf{n}$'s.

Chapter 7

Efficient Puncturing Algorithms for Finite-Length EE Generalized Codes

Rate-compatible (RC) codes are commonly used in time-varying channels such as mobile and wireless communications due to their flexibility and efficiency. In RC codes, codes of various rates are nested together and the higher-rate codes can be obtained by puncturing parity bits in lower-rate codes, i.e. the higher-rate codes are "embedded" in lower-rate codes. During the transmission, once the decoder fails, the additional redundant bits are sent incrementally to the receiver to perform decoding repeatedly until the codeword can be successfully decoded, therefore RC codes are suitable for Type-II hybrid automatic repeat request (HARQ). In addition, RC codes can also conveniently provide a range of "fixed" code rates obtainable from a common mother code.

EE-GLDPC and EE-DGLDPC codes are systematic codes, therefore they can be punctured all the way up to rate $R = 1$ without causing any stopping sets in the punctured set. EE-GLDPC codes are naturally suitable for puncturing and are able to address the rate constraint issue of ordinary sGLDPC codes, i.e. the dedicated sGLDPC codes are constrained to relatively lower rate.

Compared with RC-LDPC codes, which have received wide-spread study, there are few studies on RC-GLDPC and RC-DGLDPC codes in existing literature. Because GLDPC and DGLDPC codes use more complicated component codes as SCNs and SVNs, it is not easy to design a straightforward, yet efficient, puncturing algorithm for them. In this Chapter, we demonstrate that for EE-GLDPC and EE-DGLDPC codes, we can design good puncturing patterns to improve the performance of punctured codes rather than randomly puncturing them. Simulations show that the punctured EE-GLDPC and EE-DGLDPC codes can show a similar or even better performance than IRA-LDPC and EERC-LDPC codes over a range of code rates and SNRs.

7.1 Backgrounds on the puncturing of LDPC codes

Studies on RC-LDPC codes can be found in [54–66]. In [59] and [61], RC-LDPC codes are constructed from higher rates to lower rates by extending. In [57], the authors investigate the optimal puncturing distribution based on the asymptotic analysis and show that it will result in little loss in decoding threshold over a range of code rates. Such analysis nonetheless does not describe in detail how the specific bits should be punctured. In [56] it is proved that there is a cutoff rate for the punctured profile such that if the punctured rate is lower than the cutoff rate, the error free of decoding can be achieved via a high-enough SNR, otherwise the error rate is always positive unless the channel is noiseless. Finally, the performance of infinite-length punctured LDPC codes under ML decoding is studied in [58].

When the code length is finite, the notion of l-step recoverable (l-SR) nodes is important. A node is called a l-SR node if it is recovered after the lth iteration when the punctured codeword is encoded by an erasure decoder. Note that in this case the known values of information bits and unpunctured parity bits are provided to the erasure decoder. The l-SR node can also be defined as follows: if the codeword is transmitted on AWGN channel, such node receives no extrinsic information in the first $l-1$ decoding iterations but receives the non-zero extrinsic information from at least one of neighbor CNs in the lth iteration, and the non-zero extrinsic information can be either correct or wrong on the true value of this l-SR node. The CNs providing extrinsic information in the lth iteration are referred to as *survived CNs in the lth iteration*.

In [63], the number of survived CNs is considered when puncturing a l-SR node. In [62], Ha *et al.* argues that, a punctured node j has a higher recovery error probability if more unpunctured bits in the recovery tree are used to restore the value of j. In view of this, the authors propose an greedy “group-and-sort” algorithm which tries to minimize the number of iterations required to recover the whole codeword for a given rate, by sequentially maximizing the size of l-SR node group for $l = 1, 2...$; inside each l-SR node group, the VNs are ranked according to the number of survived

CNs they connect to, and the VNs with larger number of survived CNs are punctured earlier to improve the puncturing performance. A modified version of [62] appears in [64], where the cost functions of VNs and CNs are defined to further enhance puncturing performance. A puncturing method for protograph-based LDPC codes is proposed in [65], where the notion of "puncturing score" is introduced to speed up convergence, which is in accordance with the principle adopted in [62].

IRA-LDPC and EERC-LDPC codes are systematic codes so they can also be punctured up to $R = 1$. The optimized puncturing for EERC-LDPC codes is proposed in [54]: since the first $\lfloor \frac{M}{2} \rfloor$ parity bits are 1-SR nodes, the following $\lfloor \frac{M}{4} \rfloor$ parity bits are 2-SR nodes, and so on, the optimized puncturing for EERC-LDPC codes is simply puncturing parity bits sequentially from left to right in H_2 matrix to achieve the desired punctured code rate.

For IRA-LDPC codes, an optimized puncturing scheme is devised by Yue *et al.* [66]. The authors propose a "reverse unpuncturing" algorithm to obtain good punctured IRA-LDPC codes. The *dedicated puncturing* for a certain puncturing rate R is first defined: it is realized by *evenly* placing the unpunctured parity bits of rate R code among the H_2 matrix, in this way the L, which is the number of iterations required to recover the whole codeword, is minimized.

The "reverse unpuncturing" algorithm is implemented as follows [66]

Step 1. Puncture the IRA-LDPC code to the highest desirable code rate R_h using dedicated puncturing.

Step 2. The punctured parity bits, based on the distance with the nearest unpunctured bits, are categorized into $\boldsymbol{G} = [G_1, G_2, ..., G_L]$, where G_i is the group of i-SR nodes. Any two consecutive parity bits in G_L are assigned with label-1, and other parity bits in G_L are assigned with label-0.

Step 3. Randomly *unpuncture* a parity bit with label-1, update the recoverable steps for the VNs affected by the unpuncturing. Repeat this procedure until there is no parity bit with label-1. Then we begin to randomly unpuncture a parity bit with label-0, repeat until G_L is empty.

Step 4. update $\boldsymbol{G}$. If there are still punctured parity bits, repeat step 2 and 3 until we reach $R = R_m$, where R_m is mother code rate.

The point of the unpuncturing process is trying to fix the weakest punctured VNs in the current punctured code. After the "reverse unpuncturing", we have a set of RC codes with any puncturing rates from $R = R_m$ to R_h.

7.2 The complicated behavior of SCNs

Previous studies on the sequential puncturing of FL LDPC codes can be found in [54, 62–66]. By sequential puncturing we mean puncturing the bits one by one from mother code rate R_m to the highest rate R_h, or unpuncturing from R_h to R_m [66]. In the following, let G_l denote the set of l-SR nodes, and G_0 be the unpunctured nodes. When designing a puncturing algorithm for LDPC codes, it is desirable that the puncturing pattern have a smaller maximum value of l, i.e. the number of iterations required to recover the whole codeword. We denote this number by L. Simulation shows that this criterion is also applicable for EE-GLDPC codes. Before we present the puncturing algorithm for EE-GLDPC codes, we will discuss several issues unique to GLDPC codes.

Assume a linear code $C(n, k)$ with minimum Hamming distance d_{min} is used as the component code of GLDPC code. Let $\mathcal{E}$ be the set of erased bits in the $C(n, k)$, and $|\mathcal{E}|$ be the cardinality of $\mathcal{E}$. If the maximum likelihood (ML) decoder is used, then we have the following results:

1. $|\mathcal{E}| < d_{min}$: *all* $|\mathcal{E}|$ erasures can be fully recovered.

2. $d_{min} \le |\mathcal{E}| \le (n - k)$: *all* $|\mathcal{E}|$ erasures *may or may not* be fully recovered.

3. $|\mathcal{E}| > (n - k)$: *a portion* of erasures could possibly be recovered, but *not all* of the erasures.

A justification of these results is given below. Let $\overline{\mathcal{E}}$ be the complementary set of $\mathcal{E}$, and $col(\mathcal{E})$ be the columns of parity-check matrix corresponding to $\mathcal{E}$.

Fact 1 [29]: an erased bit v in $\mathcal{E}$ receives no extrinsic information (i.e. can not be recovered), if and only if $col(v)$ is in the vector space spanned by $col(\{\mathcal{E} \backslash \{v\}\})$.

Fact 2: an unerased bit v in $\overline{\mathcal{E}}$ receives no extrinsic information if and only if $col(v)$ is in the vector space spanned by $col(\{\mathcal{E}\})$.

Since a SCN $C(n, k)$ can recover up to $n - k$ erasures, on AWGN channel another issue with GLDPC codes is the reliability of extrinsic messages from the SCN. We can show that the reliability

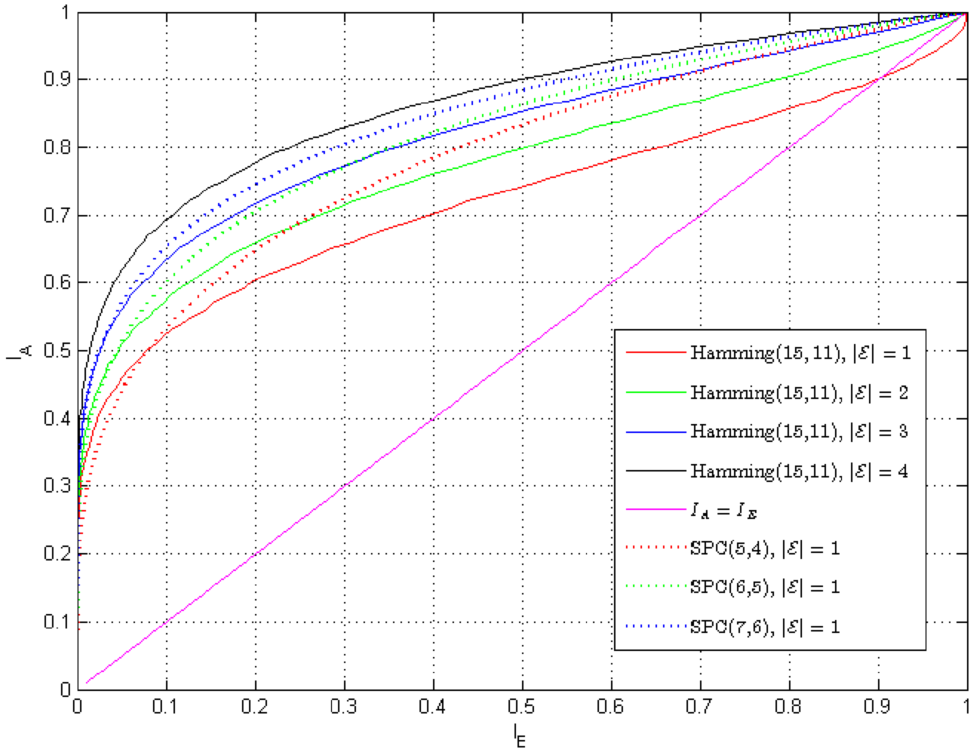

Figure 7.1: EXIT curves for various SPC codes, and $|\mathcal{E}| = 1, 2, 3, 4$ in Hamming $(15, 11)$ code.

of the extrinsic messages decreases as the number of erasures increases. As an example, assume Hamming $(15, 11)$ code is used as the component node, the EXIT curves of erased information bit(s) for $|\mathcal{E}| = 1, 2, 3, 4$ are drawn in Fig. 7.1. The EXIT curves of one erased bit for SPC$(5, 4)$, SPC$(6, 5)$ and SPC$(7, 6)$ are also shown in Fig. 7.1.

Based on this result, it is advisable to puncture in the way that bits in G_l be uniformly-connected to as many SCNs as possible rather than being clustered in a small group of SCNs, although in both cases all the bits in G_l can be recovered after l iterations.

In view of this, for GLDPC codes, we define the "deficiency of a CN" as the number of punctured nodes connected to it, denoted by τ. A CN is called *deficient* iff $\tau > 0$. Let $\mathcal{T} = (\tau_1, \tau_2, ...\tau_M)$ be the deficiency vector of a punctured codeword, where M is the number of CNs in the adjacency matrix H_{adj} and τ_i is the deficiency of CN i. In next subsection we will see that $\mathcal{T}$ serves as an approximate and efficient auxiliary measure on the goodness of puncturing pattern.

Finally, for GLDPC codes we no longer have the result for LDPC codes on the relationship between the reliability of a recovered punctured node and the number of unpunctured VNs in its recovery tree [62]. Again this is due to the decoding behavior difference of SPC and $C(n,k)$: for LDPC codes the "tanh" update rule on SPC guarantees that the extrinsic message emitted from SPC is always less reliable than *a-priori* messages, but this is not the case with GLDPC codes. For example in Fig. 7.1, when $|\mathcal{E}| = 1$ it is possible that $I_{E,C} > I_{A,C}$ as $I_{A,C} \rightarrow 1$. Though for GLDPC codes there are cases that the extrinsic message is more reliable than the *a-priori* messages, in practice due to the number of erasures in each CN and the actual range of $I_{A,C}$, together with the message dependence caused by the finite girth of the code, it is still important to minimize L during the sequential puncturing.

7.3 Puncturing algorithms for EE-GLDPC codes

7.3.1 EE-sGLDPC codes

Compared with the SPC code used in LDPC codes, the component codes in GLDPC codes exhibit a more complicated decoding behavior. We have found it difficult to find a simple algorithm such as in [62], so a greedy algorithm is adopted to do the sequential puncturing for EE-sGLDPC codes. This greedy algorithm uses a depth-first search strategy: at each puncturing step, the "best" node is punctured, and this procedure is performed progressively until all parity bits are punctured.

Suppose we want to decide if a candidate VN v should be punctured or not. Define $\boldsymbol{G} = [G_1, G_2, G_3, ...G_L]$ to be the recovery vector *after* v is punctured. Let $|G_i|$ the number of VNs in G_i and $|\boldsymbol{G}| = \sum |G_i|$, and $\tilde{\mathcal{T}}$ be the deficiency vector obtained after $(L-1)$th iterations, and $|\tilde{\mathcal{T}}|$ be the sum of elements in $\tilde{\mathcal{T}}$. The following factors are first considered, in order of descending priority:

1. the length of vector $\boldsymbol{G}$ after v is punctured.
2. the distribution of elements in $\boldsymbol{G}$ after v is punctured.
3. $\frac{|\tilde{\mathcal{T}}|}{|G_L|}$, the mean value of elements in $\tilde{\mathcal{T}}$.
4. $Var(\tilde{\mathcal{T}})$, the variance of elements in $\tilde{\mathcal{T}}$.

The first criterion tries to minimize L, the maximum number of iterations to recover the whole codeword. When there is more than one VN that results in puncturing patterns of the same L, it is desirable that more bits be recovered at earlier iterations, so based on the idea of exponentially-weighted average we define the cost function $W = \sum 2^{(i-1)} |G_i|$ to provide a quantitative assessment on the goodness of each pattern, and the VN with the smallest W will be selected (criterion 2). If there are still ties among the VN candidates, the VN yielding the smallest mean value of elements in $\tilde{\mathcal{T}}$ is selected (criterion 3), since more reliable messages will be returning to the punctured bits in G_L.

We also adopt an additional auxiliary criterion 4 based on $\tilde{\mathcal{T}}$, if there are still ties after the sieve of above-mentioned criteria. We use an example to illustrate criterion 4: Let $\boldsymbol{G} = [G_1, G_2, G_3, ...G_L]$ and $\boldsymbol{G'} = [G_1^{'}, G_2^{'}, G_3^{'}, ...G_L^{'}]$ correspond to two different puncturing patterns $\mathcal{P}$ and $\mathcal{P}^{'}$, and $\tilde{\mathcal{T}}$ and $\tilde{\mathcal{T}}^{'}$ be the deficiency vectors obtained after $(L-1)$th iterations. If $Var(\tilde{\mathcal{T}}) < Var(\tilde{\mathcal{T}}^{'})$ then $\mathcal{P}$ is presumed to be better than $\mathcal{P}^{'}$, where $Var(\cdot)$ means the variance of vector elements. That is to say, the elements in $\tilde{\mathcal{T}}$ are desired to be as uniform as possible; by doing so we can increase the reliability of VNs in G_L, which are the most vulnerable nodes in the graph. Criterion 4 is very useful at early stages of puncturing, since there are many ties even after the filters of first three criteria.

Let H_{adj} be the adjacency matrix of the EE-sGLDPC code, and H_{comp} the component matrix (H_{comp} can be of different structures for each CN). During the puncturing, let $PunctSet$ be the set of punctured parity bits and $VNodes$ be the pool of remaining parity bits. The pseudocode for our systematic puncturing algorithm is given as follows

The systematic puncturing algorithm for EE-sGLDPC codes

```
Input: $H_{adj}$, $H_{comp}$ Output: $PunctSet$ Initialization: $VNodes = \{\text{all parity bits}\}$.
$BestNodes = \emptyset$. $PunctSet = \emptyset$.
Process:
FOR {ii=1; ii<$NumofParityBits$; ii+1}
    FOR {each parity bit $v$ in $VNodes$}
        $PunctSetTemp = PunctSet \cup v$.
        puncture all the bits in $PunctSetTemp$, use the erasure
        decoder to calculate $L$, $W$, $\frac{|\tilde{\mathcal{T}}|}{|G_L|}$, and $Var(\tilde{\mathcal{T}})$.
    END
    $BestNodes$={best bits after the filter of four criteria}.
    if $|BestNodes| > 1$, randomly choose a bit $v_{best}$ from $BestNodes$
    $PunctSet$= $PunctSet \cup \{v_{best}\}$. $VNodes$= $VNodes \setminus \{v_{best}\}$. $BestNodes = \emptyset$.
END
RETURN $PunctSet$
```

As an example, we compare this systematic puncturing with random puncturing for a short IRA-sGLDPC code $(N, K) = (150, 70)$ code using Hamming $(15, 11)$ as the component code. Fig. 7.2 plots L (the number of iterations required to recover the whole codeword) versus the number of punctured parity bits for these two cases. Due to the tie-breaking of systematic puncturing, we give two sample patterns for systematic puncturing. It can be observed that the L of random puncturing is always greater than or equal to that of systematic puncturing, for any puncturing rate.

7.3.2 EE-hGLDPC codes

For hGLDPC codes, i.e the CN constraints contain both SCNs and SPCs, we will not have a good puncturing pattern if we directly adhere to the above puncturing algorithm. This is due to the fact that SCNs and SPCs have different error correcting capabilities. The EXIT curve is useful in determining the error correcting capability of a given SCN or SPC-CN with deficiency τ. We define $\mathcal{A} = \int_0^1 I_E(I_A, \tau) dI_A$, which corresponds to the area below I_E curve as a function of I_A. A larger $\mathcal{A}$ implies a larger error correcting capability and vice versa. For example, a Hamming $(15, 11)$ code

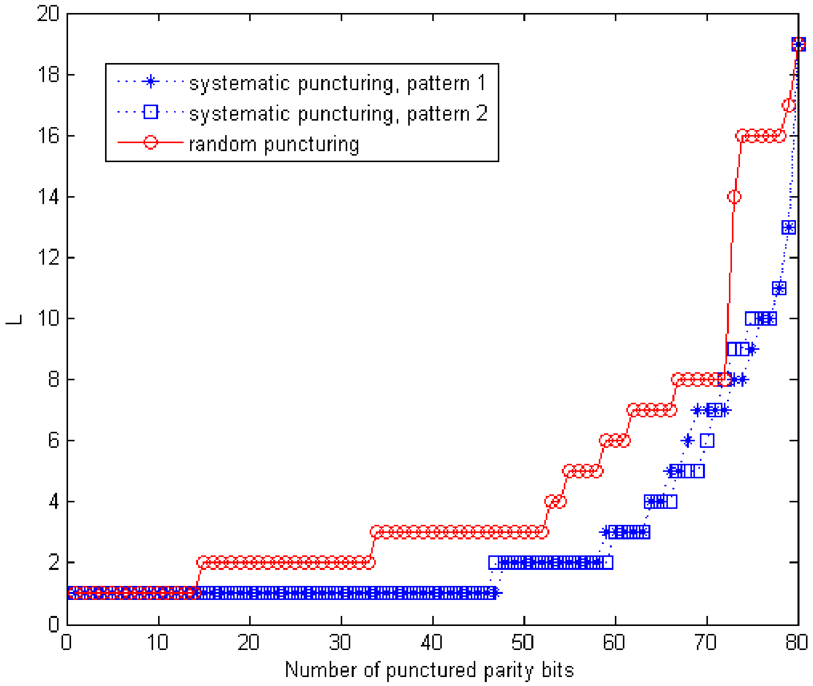

Figure 7.2: Recovery speed comparison of systematic and random puncturing of a $(150, 70)$ IRA-sGLDPC code.

of deficiency 1 is considered providing more reliable messages than a SPC$(7, 6)$ code of the same deficiency degree, as shown in Fig. 7.1. Therefore, when puncturing, it is desirable to puncture the CNs with larger $\mathcal{A}(\tau)$ first.

The above claim is supported by the following simulation. We construct a length-1000, rate-0.5 hGLDPC code using the profile

$$\lambda(x) = 0.3893x + 0.2682x^2 + 0.0025x^3 + 0.0026x^4 + 0.3374x^5$$

$$\rho(x) = 0.0014x^5 + 0.6311x^6 + 0.1011x^7 + 0.0037x^8 + 0.0166x^9 + 0.2460x^{14} \ (Hamming(15, 11))$$

and we adopt two puncturing schemes: “SPCpreferred” and “SCNpreferred”. Both schemes try to first minimize L, then W, as defined above. After the two criteria when there are still ties among the candidate VNs, the SPCpreferred scheme will puncture the VN which has more edges connected to SPC-CNs and at the same time keep $Var(\tilde{\mathcal{T}})$ as small as possible, whereas SCNpreferred will puncture the VN which has more edges connected to SCNs and at the same time keep $Var(\tilde{\mathcal{T}})$ as small as possible.

Fig. 7.3 shows the BER performance comparison of SPCpreferred, SCNpreferred and random puncturing. It can be seen that SPCpreferred scheme suffers from a noticeable performance loss compared with SCNpreferred scheme at lower rates, and at $R = 0.6$ it is even worse than the random puncturing. This is due to the intentional puncturing on the weaker SPC-CNs first. Fig. 7.4 shows the corresponding FER performance, and similar results can be observed.

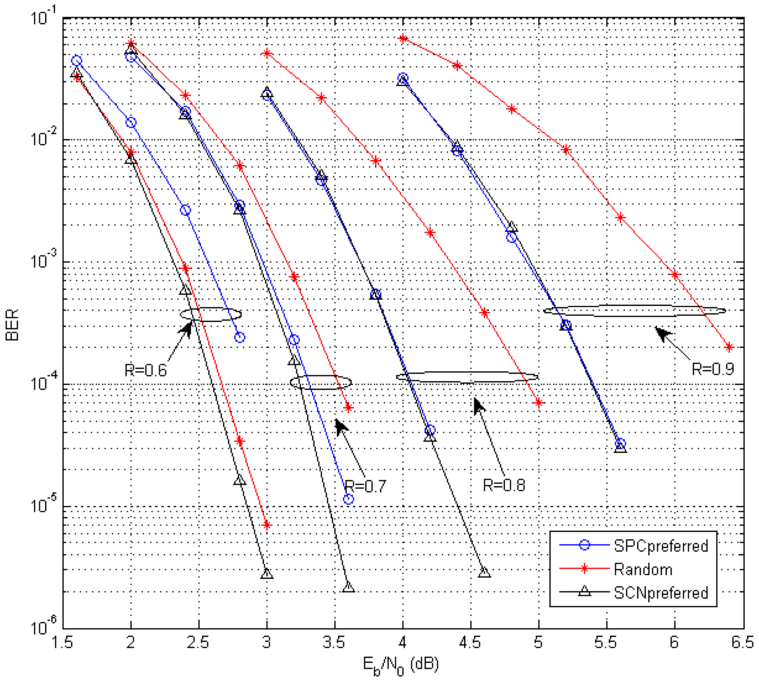

Figure 7.3: BER performance comparison of SPCpreferred, SCNpreferred and random puncturing for the hGLDPC code.

In order to give a quantitative measure on the quality of messages allowing recovery of the punctured nodes, a numeric scoring system for $(C(n, k), \tau)$ and SPC$(n, n-1)$ is still needed. Due to the dynamics of iterative decoding and the intersecting EXIT curves of various CNs, we find it very difficult to determine the optimal scoring system, hence the heuristic approach is adopted instead: for punctured bit $p \in G_L$ assuming binary value $X_p = \{0, 1\}$, let Q_p be the RV representing the APP of p after the Lth iteration. We define $\mathcal{I}_p = I(X_p, Q_p)$, which is the mutual information (MI) between X_p and Q_p, and $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_{|G_L|}\}$ be the vector of MIs of the punctured bits in G_L. The following four criteria, in order of descending priority, are used for determining the best candidate VN v.

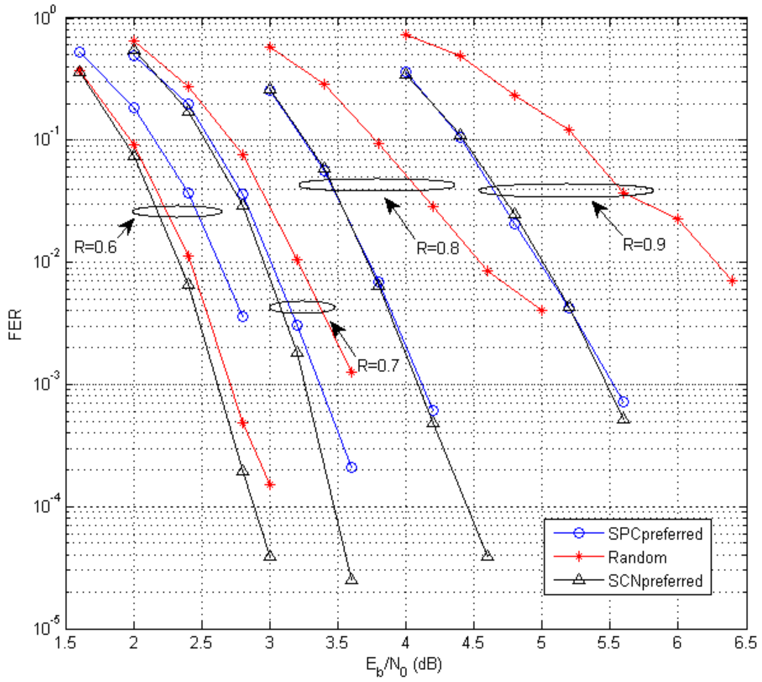

Figure 7.4: FER performance comparison of SPCpreferred, SCNpreferred and random puncturing for the hGLDPC code.

1. minimize L after v is punctured.
2. minimize W after v is punctured.
3. maximize the mean of $\boldsymbol{\mathcal{I}}$ after v is punctured.
4. minimize the variance of $\boldsymbol{\mathcal{I}}$ after v is punctured.

In practice, $\boldsymbol{\mathcal{I}}$ is still difficult to calculate, so we adopt an additional assumption that, during the Lth iteration, CNs take the messages of some specific quality I_A as their inputs and output the recovery messages with quality I_E to the nodes in G_L. Or, assume that a uniformly-distributed I_A in some interval and calculate the *average* I_E to the nodes in G_L. The calculation of $\mathcal{I}_p = I(X_p, Q_p)$ can also simplified by the ordinary "sum" of all I_E's to the VN instead of the nonlinear operation depicted in (2.41). By such simplifications $\boldsymbol{\mathcal{I}}$ can be calculated easily. The puncturing process for hGLDPC codes is essentially the same with sGLDPC codes, except that the rules are modified to accommodate the structure of hGLDPC codes. The puncturing algorithm stops when all parity bits are punctured, where we obtain a set of RC codes from $R = R_m$ to $R_h = 1$. The third criterion is very useful at the beginning of puncturing: criterion 3 will prefer SCNs over SPCs because they are more powerful, resulting in a better puncturing performance, as verified by the simulation above.

7.3.3 Simulation results and discussions

In all simulations, we adopt one-sweep MAP decoding for decoding SCNs. The maximum number of decoder iterations is set as $I_{max} = 200$. When calculating BER, only information bits are considered. For EE-GLDPC codes, Hamming $(15, 11)$ is used as the component code. Theoretically H_{comp} can have different structure for each SCN, but for the sake of simplicity we assume that H_{comp} is the same for all SCNs:

$$H_{comp} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

EE-sGLDPC puncturing: in the first experiment, we study the performance of RC EE-sGLDPC codes. We designed a $(N, K, R) = (1095, 511, 0.467)$ IRA-sGLDPC code $\mathcal{C}_s$ of average VN degree $\overline{d_v} = 2$, using the PEG algorithm. The degree distribution of $\mathcal{C}_s$ is

$$\lambda(x) = 0.0667 + 0.7333x + 0.2000x^2$$
$$\rho(x) = x^{14}$$

In the simulation $\mathcal{C}_s$ is punctured to $R = 0.5$, 0.6, 0.7, 0.8 and 0.9, and this requires that $73, 243, 365, 456$ and 527 bits are punctured, respectively. Table 7.1 list the number of punctured bits and code length for this code. We use the algorithm described above to generate one systematic puncturing pattern, or randomly create a puncturing pattern as a realization of random puncturing. Due to the randomness of random puncturing, three random realizations are generated and simulated, but it turns out that these random patterns have performance very close to each other. For the sake of clarity we only show one random puncturing in the following. Table 7.2 lists the distribution of $\boldsymbol{G}$ for random and systematic puncturing methods at $R = 0.5, 0.6, 0.7$ and 0.8. When $R = 0.9$, L for systematic puncturing is 7 and random puncturing is 20.

Table 7.1: Code lengths and number of punctured bits for the $(1095, 511)$ IRA-sGLDPC code at various puncturing rates.

	$R = 0.5$	$R = 0.6$	$R = 0.7$	$R = 0.8$	$R = 0.9$
Code length(Num. punctured bits)	1022(73)	852(243)	730(365)	639(456)	568(527)

Table 7.2: $\boldsymbol{G}$'s of systematic and random puncturing for the $(1095, 511)$ IRA-sGLDPC code at various puncturing rates.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
systematic $R = 0.5$	73	0	0	0	0	0	0	0
random $R = 0.5$	73	0	0	0	0	0	0	0
systematic $R = 0.6$	243	0	0	0	0	0	0	0
random $R = 0.6$	213	29	1	0	0	0	0	0
systematic $R = 0.7$	365	0	0	0	0	0	0	0
random $R = 0.7$	223	98	35	9	0	0	0	0
systematic $R = 0.8$	279	168	9	0	0	0	0	0
random $R = 0.8$	149	105	70	54	40	29	7	2

Fig. 7.5 gives the performance of random and systematic puncturing patterns. It can be observed that systematic puncturing always outperforms random puncturing; at BER= 10^{-5}, systematic puncturing has a gain of about 0.35 dB and 0.8 dB over random puncturing, for $R = 0.7$ and $R = 0.8$, respectively. In addition, it is worth noting that at $R = 0.5$, both random and systematic puncturing can recover all the punctured bits after one iteration, however simulations show that systematic puncturing is always slightly better than random puncturing at all SNRs, this verifies the effectiveness of criteria 3 and 4 used in puncturing algorithm.

We also constructed three *dedicated* sGLDPC codes with $K = 511$, of designed rates 0.5, 0.6 and 0.7, using Hamming $(15, 11)$ code as its component code. Due to the constraints imposed by SCNs, the closest code length N is 1023, 851 and 731 for dedicated codes, achieving the rates 0.4995, 0.6005 and 0.6990, respectively. The $\overline{d_v} = 1.8768$ for $R = 0.4995$, 1.4982 for $R = 0.6005$ and 1.1286 when $R = 0.6990$. We can not build dedicated codes of rate 0.8 and 0.9, since in such cases $\overline{d_v} < 1$. In Fig. 7.5, rate 0.6 and 0.7 dedicated codes show a nearly flat performance curve, due to the existence of a large portion of degree-1 VNs; this is the well-known rate constraint issue of sGLDPC codes, i.e. sGLDPC codes are not suitable for designing high-rate codes. However our proposed systematic puncturing algorithm achieves competitive performance for sGLDPC codes even in the high-rate region.

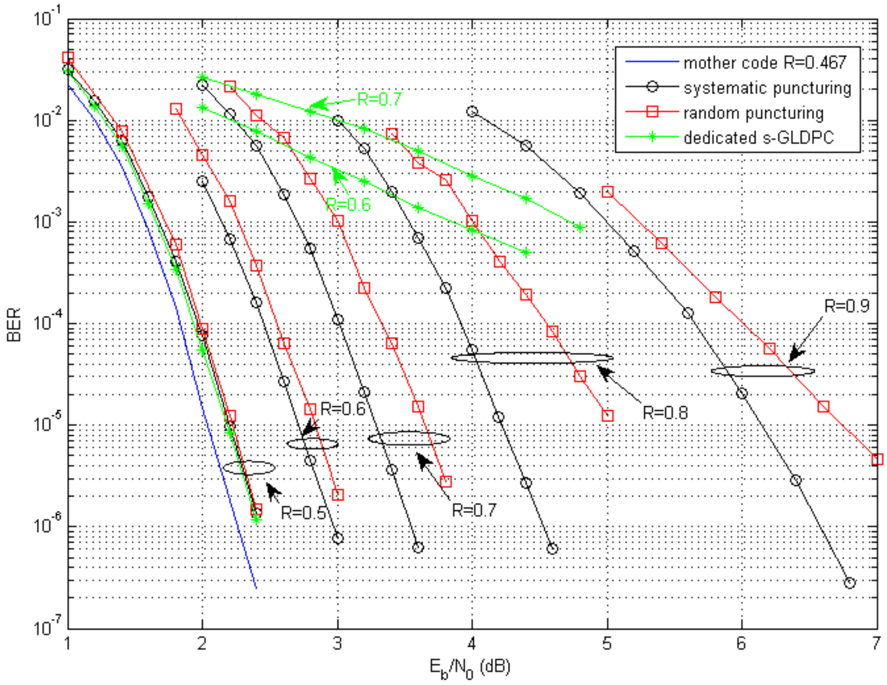

Figure 7.5: Performance comparison of systematic and random puncturing of (1095, 511) IRA-sGLDPC code.

In the second experiment, we build using PEG a shorter (645, 301) IRA-sGLDPC code using the profile above. In the simulation the code is punctured to $R = 0.5$, 0.6, 0.7, 0.8 and 0.9. Table 7.3 list the number of punctured bits and code length for this code. We compare the performance of random and systematic puncturing methods. Table 7.4 lists the distribution of $\boldsymbol{G}$ for two methods at $R = 0.5, 0.6, 0.7$ and 0.8. When $R = 0.9$, L for systematic puncturing is 7 and random puncturing is 16. Fig. 7.6 gives the performance comparison of random puncturing and systematic puncturing. We can see that systematic puncturing is better than random puncturing at all SNRs and rates. At lower rate $R = 0.5$, the performance gap between the two is narrow (< 0.1 dB) and it widens as R increases. At $R = 0.7$ and $R = 0.8$, the gap is about 0.6 dB and 0.9 dB, respectively, at BER= 10^{-5}.

Table 7.3: Code lengths and number of punctured bits for the (645, 301) IRA-sGLDPC code at various puncturing rates.

	$R = 0.5$	$R = 0.6$	$R = 0.7$	$R = 0.8$	$R = 0.9$
Code length(Num. punctured bits)	602(43)	502(143)	430(215)	376(269)	334(311)

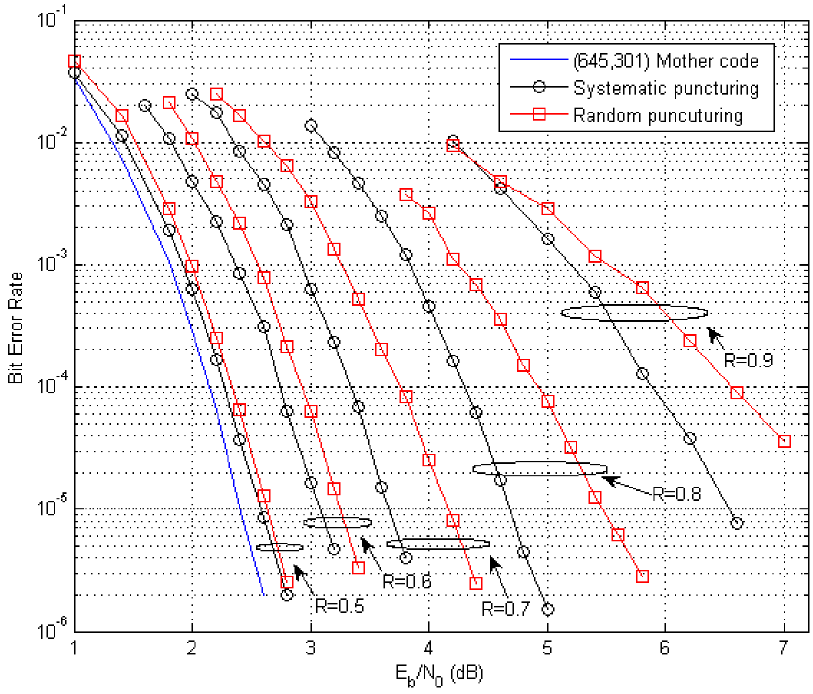

Figure 7.6: Performance comparison of systematic and random puncturing of $(645, 301)$ IRA-sGLDPC code.

Table 7.4: $\boldsymbol{G}$'s of systematic and random puncturing methods for the $(645, 301)$ IRA-sGLDPC code at various puncturing rates.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
intent $R = 0.5$	43	0	0	0	0	0	0	0
random $R = 0.5$	43	0	0	0	0	0	0	0
intent $R = 0.6$	143	0	0	0	0	0	0	0
random $R = 0.6$	118	23	2	0	0	0	0	0
intent $R = 0.7$	215	0	0	0	0	0	0	0
random $R = 0.7$	112	68	28	7	0	0	0	0
intent $R = 0.8$	162	101	6	0	0	0	0	0
random $R = 0.8$	86	62	52	36	26	7	0	0

EE-hGLDPC puncturing: Next, the puncturing performance of EE-hGLDPC codes is studied. The optimized profile we use has a decoding threshold $E_b/N_0 = 0.56$ dB, and has the degree distribution

$$\lambda(x) = 0.3740x + 0.3050x^2 + 0.0038x^3 + 0.0035x^4 + 0.3137x^5$$
$$\rho(x) = 0.0741x^5 + 0.5780x^6 + 0.0317x^7 + 0.0765x^8 + 0.2397x^{14}(Hamming)$$

With the help of PEG, we designed a $(N, K, R) = (1024, 512, 0.5)$ IRA-hGLDPC mother code $\mathcal{C}_h$ using the above optimized profile. Due to the randomness of PEG and the structure of IRA-hGLDPC code, the actual degree distribution is slightly different. We assume I_A is uniform on the interval $[0.5, 1]$. $\mathcal{C}_h$ is punctured to $R = 0.6$, 0.7, 0.8 and 0.9, corresponding to a total of $171, 293, 384$ and 455 punctured bits, respectively. Similar with sGLDPC case we generate three random puncturing patterns and their performance are very similar, so we only show one random case. Table 7.5 lists the distribution of $\boldsymbol{G}$ for random and systematic puncturing methods at $R = 0.5$, 0.6, 0.7 and 0.8. When $R = 0.9$, L for systematic puncturing is 5 and random puncturing is 16.

Table 7.5: $\boldsymbol{G}$'s of systematic and random puncturing for the $(1024, 512)$ IRA-hGLDPC code at various puncturing rates.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}
systematic $R = 0.6$	171	0	0	0	0	0	0	0	0	0
random $R = 0.6$	149	20	2	0	0	0	0	0	0	0
systematic $R = 0.7$	293	0	0	0	0	0	0	0	0	0
random $R = 0.7$	197	63	18	11	4	0	0	0	0	0
systematic $R = 0.8$	285	99	0	0	0	0	0	0	0	0
random $R = 0.8$	176	81	47	34	23	8	6	5	2	2

Fig. 7.7 compares the performance of random and systematic puncturing patterns. The simulation results are similar to the results above. At BER$= 10^{-5}$, systematic puncturing has a gain of about 0.3 dB and 0.8 dB over random puncturing, for $R = 0.7$ and $R = 0.8$, respectively, whereas at $R = 0.9$ it is about 1.1 dB.

Our next experiment involves the comparison of systematic punctured EE-GLDPC codes with IRA-LDPC and EERC-LDPC codes. We construct by PEG a $(1024, 512, 0.5)$ EERC-LDPC code and an IRA-LDPC code of the same length and rate, under maximum VN degree constraint $dvmax = 6$. The actual degree distribution of the EERC-LDPC code is

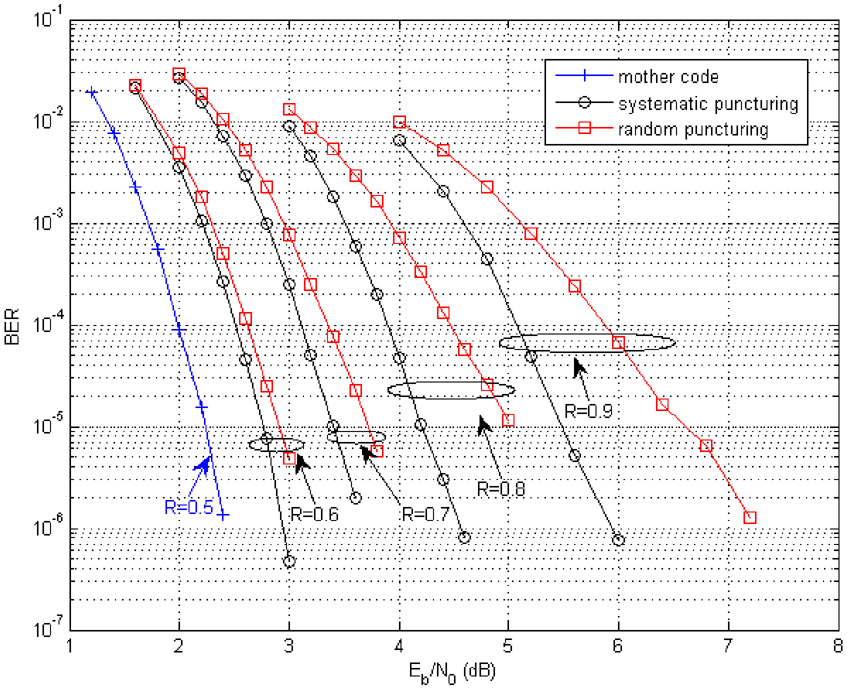

Figure 7.7: Performance comparison of systematic and random puncturing of $(1024, 512)$ IRA-hGLDPC code.

$$\lambda(x) = 0.0003 + 0.3328x + 0.2468x^2 + 0.0012x^3 + 0.4188x^5$$

$$\rho(x) = 0.7367x^5 + 0.2356x^6 + 0.0075x^7 + 0.0140x^8 + 0.0062x^9$$

and the actual degree distribution of the IRA-LDPC code is

$$\lambda(x) = 0.0003 + 0.3328x + 0.2468x^2 + 0.0012x^3 + 0.4188x^5$$

$$\rho(x) = 0.7012x^5 + 0.2988x^6$$

Fig. 7.8 gives the BER performance comparison of systematic punctured $\mathcal{C}_s$, $\mathcal{C}_h$, IRA-LDPC and EERC-LDPC codes, at $R = 0.5$, 0.6 and 0.7. The four codes have virtually the same waterfall performance, whereas $\mathcal{C}_s$ and $\mathcal{C}_h$ show significantly better error floor performance. At $R = 0.5$ it can be reasonably estimated that $\mathcal{C}_s$ and $\mathcal{C}_h$ have a gain of 0.4 dB over EERC-LDPC code at BER=10^{-6}, and 0.8 dB at BER=10^{-7}. At $R = 0.7$, $\mathcal{C}_h$ exhibits a slightly worse waterfall performance compared with $\mathcal{C}_s$ and EERC-LDPC codes, but its BER is about $\frac{1}{3}$ of EERC-LDPC code at high SNR (3.8 dB), and gains are likely to be even better when BER is still lower. The low error floor of EE-sGLDPC

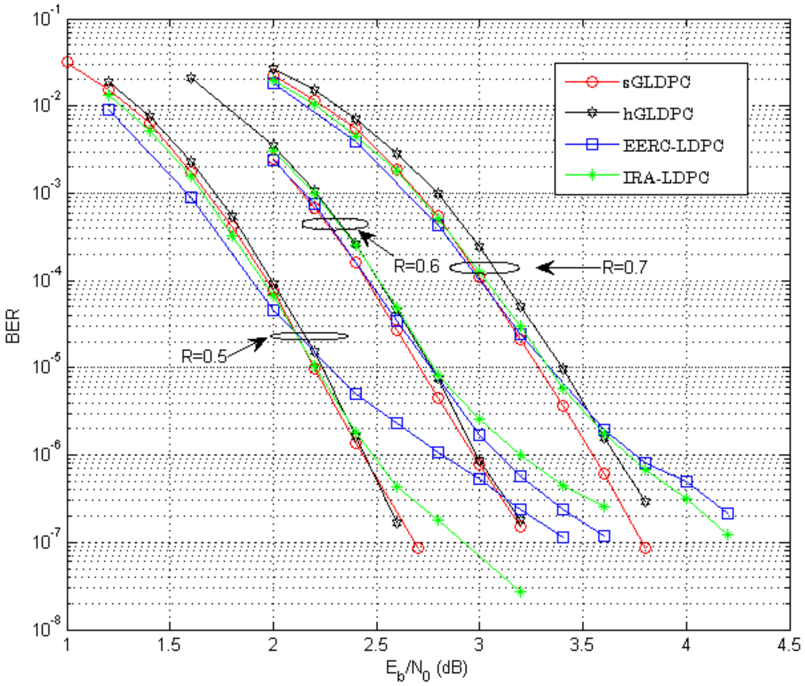

Figure 7.8: BER performance comparison of EE-GLDPC codes, IRA-LDPC and EERC-LDPC codes, $R = 0.5$, 0.6 and 0.7.

and EE-hGLDPC codes are more clearly reflected in terms of FER, as Fig. 7.9 shows.

At $R = 0.8$ and 0.9, $\mathcal{C}_s$ and $\mathcal{C}_h$ still preserve the good error floor performance, and outperform IRA-LDPC and EERC-LDPC codes at high SNRs, as shown in Fig. 7.10. Fig. 7.11 depicts the FER performance comparison, where advantages of GLDPC codes are more obvious at high SNRs. However, it can be seen that $\mathcal{C}_s$ begins to show serious waterfall performance degradation, especially at $R = 0.9$; there is about 0.7 dB loss compared with EERC-LDPC code at BER=10^{-5}. We argue this is due to the inherent weakness of sGLDPC codes at very high code rates.

Assume a sGLDPC code and a LDPC code have the same recovery time L at very high rates $(R \geq 0.9)$. Since most of parity bits have been punctured, by some simple analysis we know that on average each SCN recovers $n - k$ parity bits in each iteration. A $C(n, k)$ SCN with deficiency $\tau = n - k$ suffers greatly from performance degradation; for example a Hamming $(15, 11)$ component code with $\tau = 4$ has an EXIT curve nearly the same with SPC$(8, 7)$, such a sGLDPC code at $R \geq 0.9$ is essentially equivalent to a heavily-punctured LDPC code with CN degree 8 and $\overline{d_v}$ around 2, which implies a poor decoding threshold; on the contrary. The EERC-LDPC code has

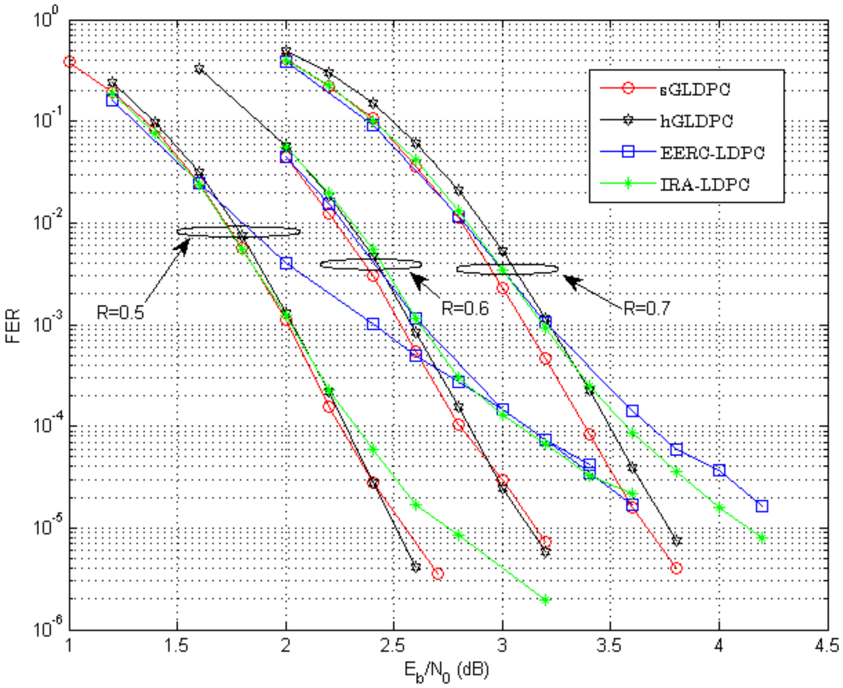

Figure 7.9: FER performance comparison of EE-GLDPC codes, IRA-LDPC and EERC-LDPC codes, $R = 0.5$, 0.6 and 0.7.

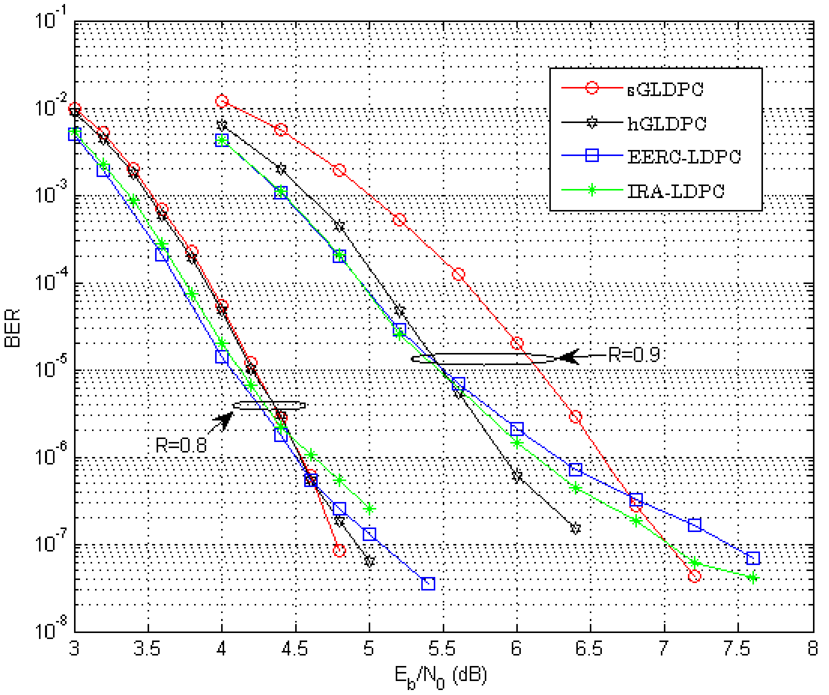

Figure 7.10: BER performance comparison of EE-GLDPC codes, IRA-LDPC and EERC-LDPC codes, $R = 0.8$ and 0.9.

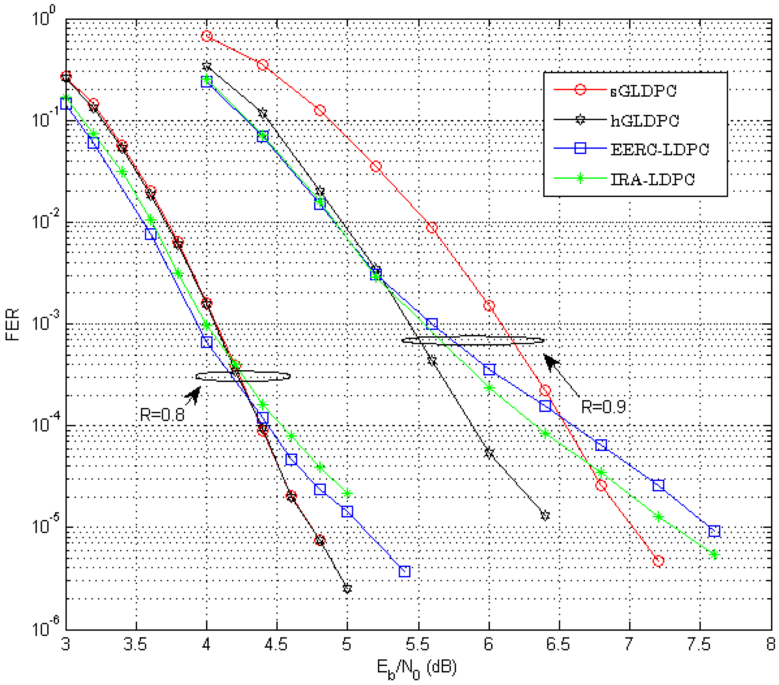

Figure 7.11: FER performance comparison of EE-GLDPC codes, IRA-LDPC and EERC-LDPC codes, $R = 0.8$ and 0.9.

$dvmax = 6$ and a large portion of SPC$(6, 5)$ checks, both of which greatly help improving the decoding threshold. As shown in Fig. 7.12, we designed a $R = 0.467$ IRA and EERC-LDPC codes with $dvmax = 6$; IRA-LDPC codes are *unpunctured* from $R_h = 1$ using the method in [66]; at puncturing rate $R = 0.9$, the average APP quality of parity bits in G_L, measured in terms of MI, are drawn for EERC-LDPC, IRA-LDPC code and $\mathcal{C}_s$. We can see the quality of parity bits of $\mathcal{C}_s$ are significantly inferior to LDPC codes, confirming the discussion above.

Finally, hGLDPC codes can be regarded as SCN-doped LDPC codes. The allowed higher degree VNs in hGLDPC codes can overcome the drawback of sGLDPC codes at high code rates, as shown in Fig. 7.10. In practice, the doping degree, i.e. the number of SCNs in hGLDPC codes, can be easily and smoothly adjusted to encompass both advantages of sGLDPC and LDPC codes.

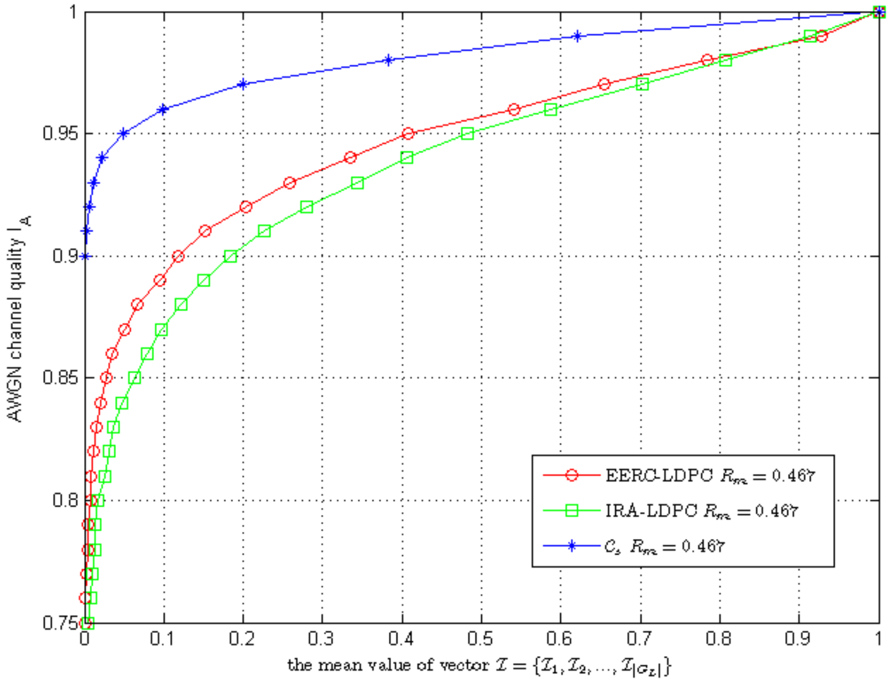

Figure 7.12: The average APP quality of parity bits in G_L, measured in terms of MI, for EERC-LDPC, IRA-LDPC and $\mathcal{C}_s$ at $R = 0.9$.

7.4 Puncturing algorithms for EE-DGLDPC codes

7.4.1 EE-DGLDPC with no SCNs

For EE-DGLDPC codes with no SCNs, they can be regarded as EE-LDPC codes with information bits existing in H_2 matrix. Therefore, the systematic puncturing for such EE-DGLDPC codes can be largely reduced to the systematic puncturing of EE-LDPC codes, the only difference is that in EE-LDPC codes all parity bits are REP-VNs whereas in EE-DGLDPC codes, parity bits are either REP-VNs or among the SPC-SVNs.

As an example, the length $N = 1200$, rate-0.5 IRA-DGLDPC code in subsection 5.4 is adopted as a mother code for puncturing. The adjacency matrix H_{adj} is 600×954 and the number of SPC(3, 2) SVNs is 246, so in H_2 matrix parity bits are either deg-2 REP-VNs or in the SPC(3, 2) SVNs. The IRA-DGLDPC code uses the degree profiles

$$\lambda(x) = 0.00180x + 0.37785x^2 + 0.00010x^3 + 0.41592x^5 + 0.20433x^2\ (SPC(3,2))$$

$$\rho(x) = 0.00058x^4 + 0.98065x^5 + 0.001749x^6 + 0.00069x^7 + 0.00059x^8$$

Notice the H_2 matrix for this code is essentially a double-diagonal matrix, as in IRA-LDPC codes. So the reverse unpuncturing algorithm for IRA-LDPC codes can be directly applied. We begin with the code from $R_h = 1$, and progressively unpuncture the parity bits until we reach $R_m = 0.5$. The only extra rule we add here is: when there are both REP-VNs and SPC-SVNs in label-1 (or label-0) nodes (see section 7.1) that can be unpunctured, we first unpuncture REP-VNs then SPC-SVNs. We choose $I_{max} = 200$ for the decoding, and BER is calculated in information bits only. Fig. 7.13 shows the BER performance comparison of systematic puncturing algorithm, specifically the reverse unpuncturing algorithm, with three random puncturing patterns.

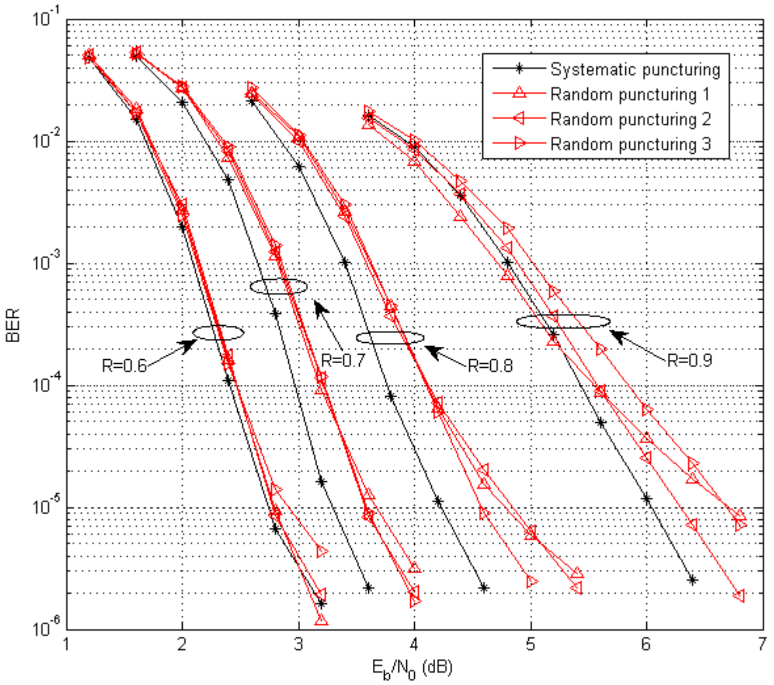

Figure 7.13: BER performance comparison of systematic puncturing and random puncturing for DGLDPC code with no SCNs.

It can be observed that in 7.13 systematic puncturing shows significant performance gain over random puncturing patterns, the gain is more noticeable for FER curves, which is shown in Fig. 7.14. It can also be seen that the punctured EE-DGLDPC code with no SCNs tend to exhibit a

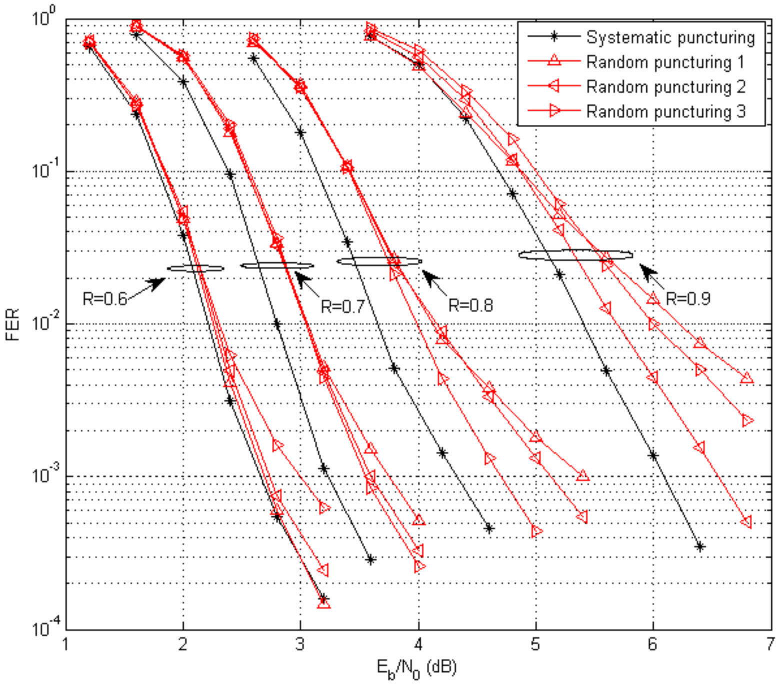

Figure 7.14: FER performance comparison of systematic puncturing and random puncturing for DGLDPC code with no SCNs.

higher error floor compared with those GLDPC/DGLDPC codes with SCNs, this is because in such EE-DGLDPC codes there are no component codes having powerful error correcting capabilities. The final comment is at $R = 0.6$ and $\frac{E_b}{N_0} = 3.2$ dB, the systematic puncturing BER is worse than one of the random puncturings; we believe this is because at this rate the punctured bits in systematic puncturing are mostly SPC-SVNs, which are more prone to the noises.

7.4.2 EE-DGLDPC with SCNs

For EE-DGLDPC codes with SCNs, the parity bits usually contain REP-VNs and SVNs; the corresponding VNs therefore have different error correcting capabilities. In view of this, we introduce a scaling vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_{|G_L|}]$, where α_i is the scaling factor reflecting the error correcting capability of the ith VN in G_L.

Based on $\boldsymbol{\alpha}$, we define $\boldsymbol{\mathcal{I}}' = \boldsymbol{\mathcal{I}} \cdot \boldsymbol{\alpha} = [\alpha_1 \mathcal{I}_1, \alpha_2 \mathcal{I}_2, ..., \alpha_{|G_L|} \mathcal{I}_{|G_L|}]$. The following four criteria, in order of descending priority, are used for determining the best candidate VN v.

1. minimize L after v is punctured.

2. minimize W after v is punctured.

3. maximize the mean of $\boldsymbol{\mathcal{I}}'$ after v is punctured.

4. minimize the variance of $\boldsymbol{\mathcal{I}}'$ after v is punctured.

The above four criteria used for EE-DGLDPC with SCNs are essentially the same with those for EE-hGLDPC codes, the only difference is that we use $\boldsymbol{\mathcal{I}}'$ instead of $\boldsymbol{\mathcal{I}}$ in criterion 3 and 4.

Simulation results

A length-1200 rate-0.5 IRA-DGLDPC code is built by GR-SCNfirst-PEG. The degree profile used by the code is

$$\lambda(x) = 0.15046x + 0.00118x^2 + 0.03180x^3 + 0.33579x^5 + 0.48077x^2 \ (SPC(3,2), \text{systematic } G_{comp})$$

$$\rho(x) = 0.02274x^5 + 0.27757x^6 + 0.69969x^{14} \ (Hamming(15,11))$$

which has a decoding threshold $E_b/N_0 = 0.429$ dB. When puncturing the code, we choose $\alpha = 2$ for SPC-SVNs and $\alpha = 1$ for REP-VNs, compared with the case that $\alpha = 0.5$ for SPC-SVNs and $\alpha = 1$ for REP-VNs, simulation results show that it can achieve relatively better decoding performance at high puncturing rates, without causing degradation at low and moderate puncturing rates. In addition, we assume I_A is uniform on the interval $[0.5, 1]$ to calculate $\boldsymbol{\mathcal{I}}'$. The IRA-DGLDPC code is punctured to $R = 0.6$, 0.7, 0.8 and 0.9, corresponding to a total of $200, 343, 450$ and 533 punctured bits, respectively. We also generate three random puncturing patterns and obtain their performance.

Fig. 7.15 shows the BER performance comparison of systematic puncturing and three random puncturing patterns. It can be seen that systematic puncturing outperforms random puncturing in all puncturing rates, except that at $R = 0.8$, systematic puncturing has almost the same BER performance with the best random puncturing. The advantage of systematic puncturing over random puncturing is more visible in the FER comparison, which is shown in Fig. 7.16.

In the next experiment, we compare the systematic punctured IRA-DGLDPC code with EERC-LDPC and IRA-LDPC codes. A $(1200, 600, 0.5)$ EERC-LDPC code and an IRA-LDPC code of the

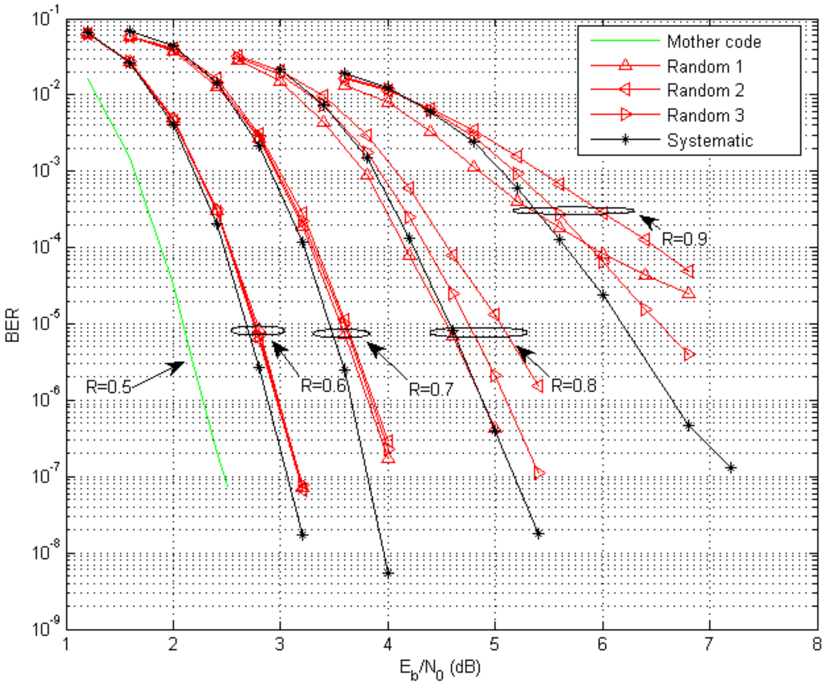

Figure 7.15: BER performance comparison of systematic puncturing and random puncturing for DGLDPC code with SCNs.

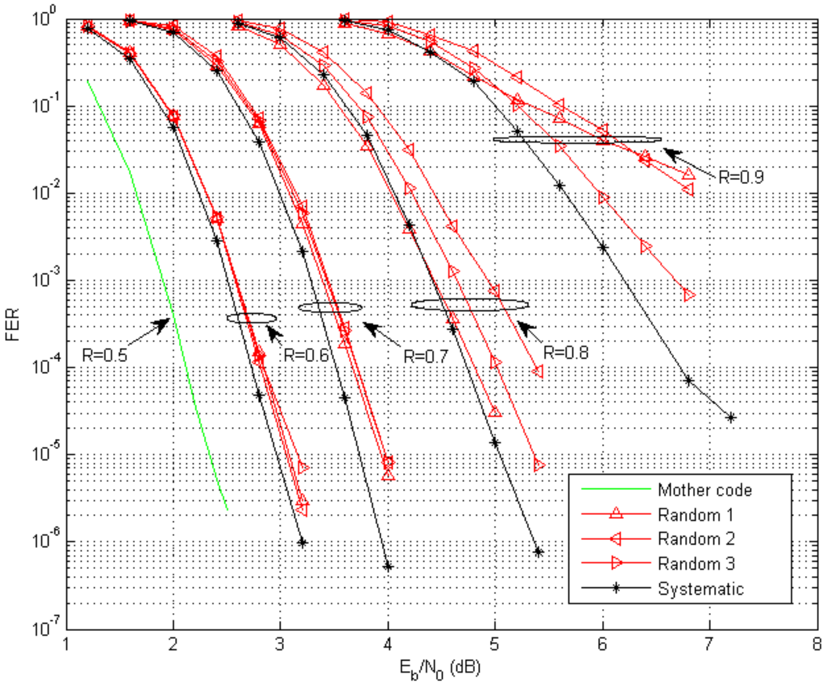

Figure 7.16: FER performance comparison of systematic puncturing and random puncturing for DGLDPC code with SCNs.

same length and rate are built by PEG. The actual degree distribution of the EERC-LDPC code is

$$\lambda(x) = 0.00027 + 0.33280x + 0.24641x^2 + 0.00106x^3 + 0.41946x^5$$
$$\rho(x) = 0.72568x^5 + 0.24934x^6 + 0.01489x^7 + 0.00717x^8 + 0.00292x^{10}$$

and the actual degree distribution of the IRA-LDPC code is

$$\lambda(x) = 0.00027 + 0.33280x + 0.24641x^2 + 0.00106x^3 + 0.41946x^5$$
$$\rho(x) = 0.69856x^5 + 0.30144x^6$$

In the simulation, MAP decoding is used for VNs and CNs, and $I_{max} = 200$. Fig. 7.17 gives the BER performance comparison of systematic punctured IRA-DGLDPC, IRA-LDPC and EERC-LDPC codes, at $R = 0.5$, 0.6 and 0.7. At $R = 0.5$ and 0.6, these codes have almost the same waterfall performance, and IRA-DGLDPC code shows better error floor behavior. For instance, at $R = 0.6$ and BER$= 10^{-7}$, IRA-DGLDPC has about 0.45 dB gain over EERC-LDPC code and a 0.7 dB gain over IRA-LDPC code. At $R = 0.7$ and BER$= 10^{-4}$, IRA-DGLDPC code has waterfall performance about 0.25 dB inferior to IRA-LDPC and EERC-LDPC codes, nonetheless it shows better error floor than LDPC codes: at $E_b/N_0 = 4$ dB, the BER of IRA-DGLDPC code is about one order lower than those of LDPC codes. The low error floor of IRA-DGLDPC codes is more clearly reflected in terms of FER, as Fig. 7.18 shows.

At $R = 0.8$ and 0.9, as Fig. 7.19 and Fig. 7.20 depict, IRA-DGLDPC code still shows good error floor behavior, however it suffers from a serious waterfall performance degradation. For instance at BER$= 10^{-4}$, IRA-DGLDPC code has a gap about 0.6 dB and 0.8 dB from EE-LDPC codes for $R = 0.8$ and 0.9, respectively. Though it is likely that the IRA-DGLDPC code can outperform EE-LDPC codes at even lower error rates, such waterfall performance degradation remains an interesting topic for future research.

We give several tentative explanations here: the first one is, for this specific IRA-DGLDPC code, there are about 70% edges connected to SCNs, therefore at very high code rates the inherent weakness of SCNs will cause the waterfall degradation, as EE-sGLDPC codes show in Fig. 7.10 and Fig. 7.11. Another possible reason is that the IRA-DGLDPC code is built by GR-SCNfirst-PEG, the constraints imposed by GR-SCNfirst-PEG make it more like a "multi-edge type" code, it is

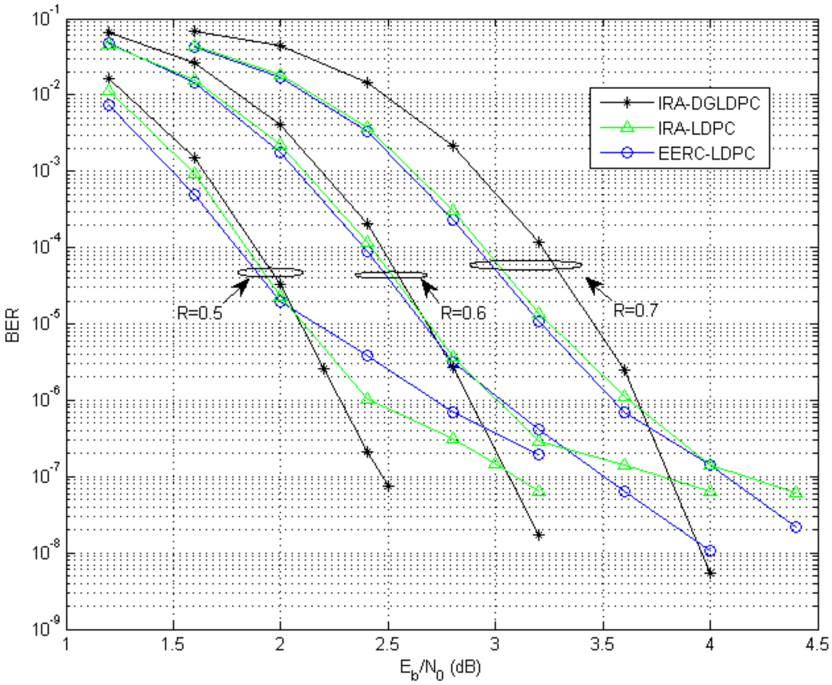

Figure 7.17: BER performance comparison of IRA-DGLDPC code, IRA-LDPC and EERC-LDPC codes, $R = 0.5$, 0.6 and 0.7.

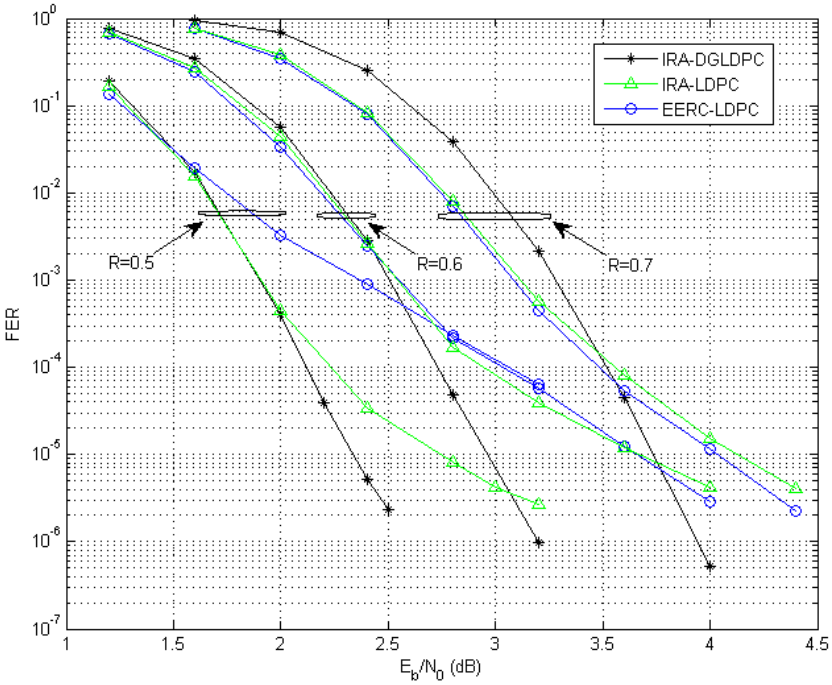

Figure 7.18: FER performance comparison of IRA-DGLDPC code, IRA-LDPC and EERC-LDPC codes, $R = 0.5$, 0.6 and 0.7.

possible that it has a poor cutoff puncturing rate [56] which results in poor performance at high rates. The last explanation is that the proposed systematic puncturing algorithm still has room for improving, this speculation is also supported by the small gap between systematic puncturing and random puncturing curves in Fig. 7.15 and Fig. 7.16.

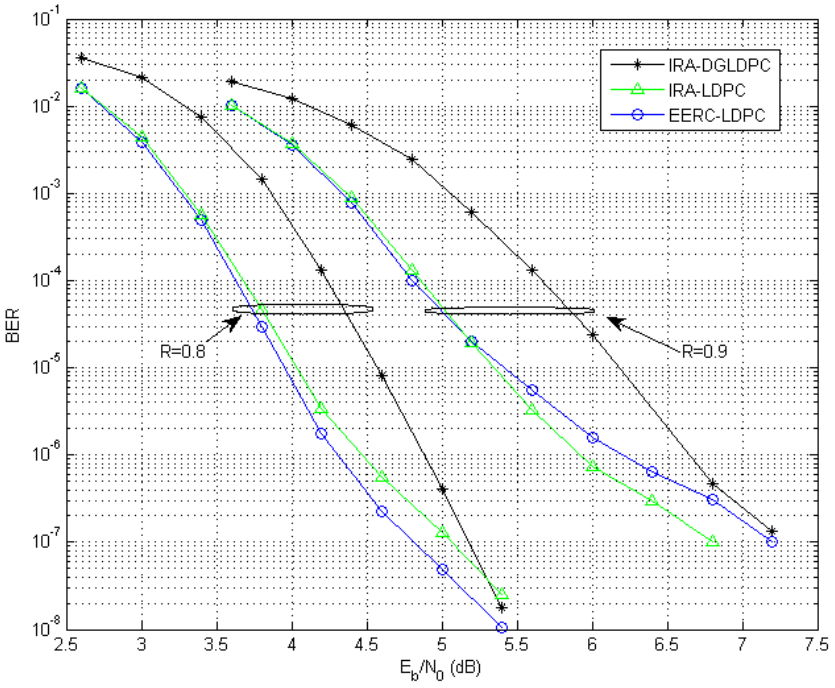

Figure 7.19: BER performance comparison of IRA-DGLDPC code, IRA-LDPC and EERC-LDPC codes, $R = 0.8$ and 0.9.

7.5 Summary

In this chapter we study the systematic puncturing for RC-GLDPC and RC-DGLDPC codes. The main points are listed below:

- SCNs and SVNs in GLDPC and DGLDPC codes show more complicated puncturing behaviors than SPC-CNs and REP-VNs in LDPC codes.

- Systematic puncturing algorithms are proposed for EE-GLDPC and EE-DGLDPC codes, which considerably improve the decoding performance compared with random puncturing.

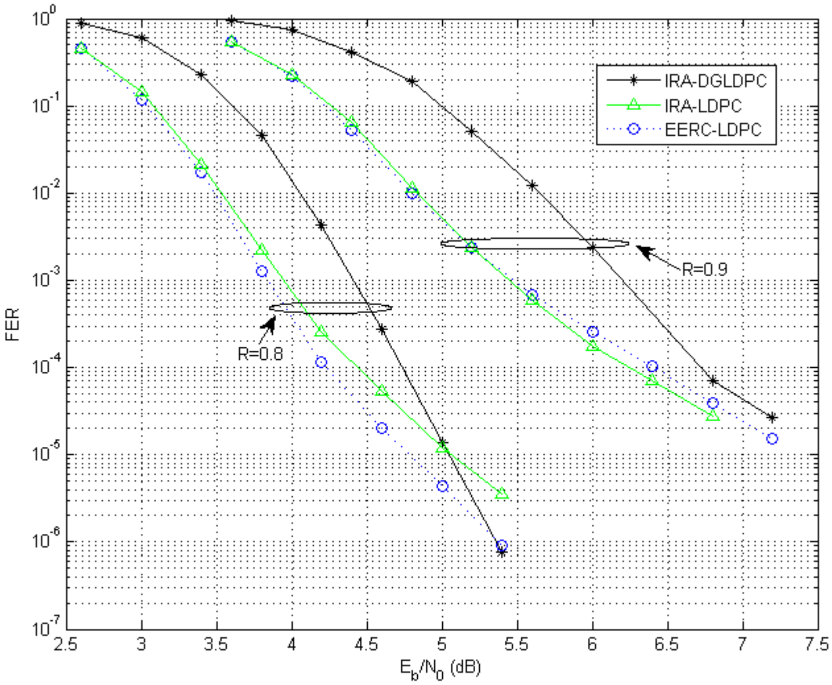

Figure 7.20: FER performance comparison of IRA-DGLDPC code, IRA-LDPC and EERC-LDPC codes, $R = 0.8$ and 0.9.

- The punctured EE-GLDPC and EE-DGLDPC codes, like their mother codes, have good error floor performance. They show competitive advantages relative to EERC-LDPC and IRA-LDPC codes, especially at high SNRs, over a wide range of code rates.

- EE-sGLDPC codes suffer from a noticeable waterfall degradation at high puncturing rates, which can be addressed by using hGLDPC or DGLDPC codes.

Chapter 8

Conclusions

In this work, we focus on designing short-to-moderate-length generalized LDPC codes, including GLDPC and DGLDPC codes.

For short GLDPC and DGLDPC codes, we extend the traditional cycle parameter ACE to a more general form eACE in Chapter 3, which is a more general form measuring the robustness of a cycle to the noise. The eACE can be naturally integrated into a modified PEG, which improves the error floor performance of GLDPC codes. On the other hand, DGLDPC codes contain more node types so the error floor is more sensitive to the specific connectivity of CNs and VNs, a girth-relaxed version of PEG is proposed to enforce additional connectivity constraints so that the weakest VNs in the cycle can be effectively protected against noise by a larger eACE. The girth-relaxed PEG achieves a significantly better error floor performance than ordinary PEG for DGLDPC codes.

Designing GLDPC and DGLDPC codes with EE property is the second topic in this work, which is discussed in-depth in Chapter 4 and 5. Based on existing IRA-LDPC and EERC-LDPC codes, we generalize the EE principle to GLDPC and DGLDPC codes, and systematic algorithms for designing EE-GLDPC and EE-DGLDPC codes are devised, providing an across-the-board solution for EE iteratively-decodable block codes. When properly designed, these EE codes have nearly the same performance with ordinary codes, in addition to their EE property. The free edges in EE-DGLDPC codes adds more intricacies for the decoding, the simple one-time permutation can effectively solve this issue.

The tradeoff between the encoding speed and error floor performance is also investigated in Chapter 6 under both analysis and various simulation results. In practice this is an interesting and important issue in real applications: the encoding speed can be maximized given the required error rate under a certain SNR, or the error floor performance can be optimized once the encoding speed fixed. It is also shown that GLDPC codes are more resilient to the change of the encoding speed.

EE codes are systematic and suitable for puncturing, therefore they are good candidates for variable rate designs. The final research topic dealt in Chapter 7 is designing the good puncturing patterns for EE-GLDPC and EE-DGLDPC codes: several systematic puncturing algorithms are devised to provide good puncturing patterns for EE-GLDPC and EE-DGLDPC codes, and it is shown that it can significantly improve the code performance compared with random puncturing. Moreover, RC EE-GLDPC and EE-DGLDPC codes show advantages over EERC-LDPC and IRA-LDPC codes, especially at high SNRs, over a wide range of code rates.

At the end of this dissertation, we propose several interesting future research topics.

1. The tradeoff between waterfall and error floor performance of DGLDPC codes under various PEGs and their parameters.

2. A unified theory on the finite-length puncturing, based on such theory we can generate optimized puncturing patterns on all the EE iteratively-decodable block codes.

3. Finding good VN and CN type combinations to further improve the decoding threshold of DGLDPC codes.

4. Design of EE convolutional GLDPC and DGLDPC codes.

Bibliography

[1] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, 1948.

[2] R. W. Hamming, “Error detecting and error correcting codes,” *Bell Syst. Tech. J.*, vol. 29, pp. 147-160, Apr. 1950.

[3] D. E. Muller, “Application of boolean algebra to switching circuit design and to error detection,” *IRE Trans. Electron. Comput.*, vol. 3, pp. 6-12, Sept. 1954.

[4] A. Hocquenghem, “Codes correcteurs d’erreurs,” *Chiffres*, vol. 2, pp. 147-156, Sept. 1959.

[5] R. C. Bose and D. X. Ray-Chaudhuri, “On a class of error-correcting binary group codes,” *Inform, and Control*, vol. 3, pp. 68-79, Mar. 1960.

[6] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *J. Soc. Indust. Appl. Math.*, vol. 8, pp. 300-304, 1960.

[7] G. D. Forney, Jr.,“The Viterbi algorithm,” *Proc. IEEE*, vol. 61, pp. 267-278, 1973.

[8] R. G. Gallager, *Information Theory and Reliable Communication.* John Wiley and Sons, 1968.

[9] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: turbo codes,” *Proc. IEEE Int. Conf. Commun.,* Geneva, Switzerland, pp. 1064-1070, May 1993.

[10] R. G. Gallager, “Low-density parity-check codes,” *IRE Trans. Inf. Theory*, vol. 39, no. 1, pp. 37-45, Jan. 1962.

[11] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457-458, Mar. 1997.

[12] S. Y. Chung, G. D. Forney, T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 1, pp. 58-60, Jan.2001.

[13] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory,* vol. 47, pp. 599-618, Feb. 2001.

[14] T. J. Richardson, A. Shokrollahi, and R. Urbanke,"Design of capacity-approaching low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619-637, Feb. 2001.

[15] C. Di, R. Urbanke, and T. Richardson, "Weight distribution of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 52, no. 11, pp. 4839-4855, Nov. 2006.

[16] I. Sason, "On universal properties of capacity-approaching LDPC code ensembles," *IEEE Trans. Inform. Theory*, vol. 55, no. 7, pp. 2956-2990, July 2009.

[17] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638-656, Feb. 2001.

[18] *IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, Sep. 2006, IEEE Std. 802.3an.

[19] *ETSI Standard TR 102 376 V1.1.1: Digital Video Broadcasting (DVB) User Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications (DVB-S2)* , Feb. 2005, ETSI Std. TR 102 376.

[20] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1* , Feb. 2006, IEEE Std. 802.16e.

[21] *IEEE Draft Standard for Information Technology-Telecommunications and information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment : Enhancements for Higher Throughput* , Feb. 2007, IEEE Std. 802.11n/D2.00.

[22] Y. Kou, S. Lin and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, Vol. 47, pp. 2711-2736, Nov. 2001.

[23] M. Fossorier, M. Mihaljević and H. Imai, "Reduced complexity iterative decoding of low density parity check codes based on belief propagation," *IEEE Trans. Commun.*, vol. 47, pp. 673-680, May 1999.

[24] J. Zhang, M. Fossorier, D. Gu, and J. Zhang, "Two-dimensional correction for min-sum decoding of irregular LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 3, Mar. 2006.

[25] X. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth Tanner graphs," in *Proc. IEEE GLOBECOM*, San Antonio, TX, Nov. 2001, pp. 995-1001.

[26] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242-1247, Aug. 2004.

[27] D. Vukobratoviè and V. Šenk, "Generalized ACE constrained progressive edge-growth LDPC code design," *IEEE commu. Letters*, vol. 12, no. 1, pp. 32-34. Jan. 2008.

[28] C. E. Shannon, "Probability of error for optimal codes in a Gaussian channel," *Bell Syst. Tech. J.*, vol. 38, pp. 611-656, 1959.

[29] C. Di, D. Proietti, E. Teletar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, no. 6, pp. 533-547, June 2002.

[30] A. Ramamoorthy and R. D. Wesel, "Construction of short block length irregular low-density parity-check codes," *in Proc. IEEE Int. Conf. on Comm. (ICC)*, pp. 410-414, Paris, June 2004.

[31] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533-547, Sep. 1981.

[32] J. Boutros, O. Pothier, and G. Zemor, "Generalized low density (Tanner) codes." in *Proc. IEEE Intl. Conf. Commun.(ICC)*, pp. 441-445, June 1999.

[33] M. Lentmaier and K. S. Zigangirov, "On generalized low-density paritycheck codes based on Hamming component codes," *IEEE Commun. Lett.*, vol. 3, pp. 248-250, Aug. 1999.

[34] I. Djordjevic, O. Milenkovic, and B. Vasic, "Generalized low-density parity-check codes for optical communication systems," *J. Lightw. Technol.*, vol. 23, no. 5, pp. 1939-1946, May 2005.

[35] N. Miladinovic and M. Fossorier, "Generalized LDPC codes with Reed-Solomon and BCH codes as component codes for binary channels," *Proc. IEEE Global Telecommun. Conf*, St. Louis, MO, pp. 1239-1244, Nov. 2005.

[36] N. Miladinovic and M. Fossorier, "Generalized LDPC codes and generalized stopping sets," *IEEE Trans. Commun.*, vol. 56, no. 2, pp. 201-212, Feb. 2008.

[37] G. Liva, W. Ryan, and M. Chiani, "Quasi-cyclic generalized LDPC codes with low error floors," *IEEE Trans. Commun.*, vol. 56, no. 1, pp. 49-57, Jan. 2008.

[38] T. Zhang and K. K. Parhi, "A class of efficient-encoding generalized low-density parity check codes," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP 2001)*, vol. 4, pp. 2477-2480, 2001.

[39] G. Yue, L. Ping, and X. Wang, "Generalized low-density parity-check codes based on Hadamard constraints," *IEEE Trans. Inform. Theory*, vol. 53, no. 3, pp. 1058-1079, Mar. 2007.

[40] S. Abu-Surra, G. Liva, and W. Ryan, "Low-floor Tanner codes via Hamming-node or RSCC-node doping," in *Proc. of Int. Symp. Applied Algebra, Algebraic Algoritmhs, and Error Correcting Codes*, M. Fossorier, H. Imai, S. Lin, and A. Poli, Eds., Berlin, Germany, Lecture Notes in Computer Science, pp. 245-254, 2006.

[41] J. Chen and R. M. Tanner, "A hybrid coding scheme for the Gilbert-Elliott channel," *IEEE Trans. Commun.*, vol. 54, no. 10, pp. 1787-1796, Oct. 2006.

[42] S. Dolinar, "Design and iterative decoding of networks of many small codes," in *Proc. 2003 IEEE Int. Symp. Inform. Theory*, Yokohama, Japan, Jun./Jul. pp. 346, 2003.

[43] Y. Wang and M. Fossorier, “EXIT chart analysis for doubly generalized LDPC codes,” in *Proc. 2006 IEEE Global Telecommun. Conf.*, San Francisco, CA, pp. 1-6, Nov. 2006.

[44] Y. Wang and M. Fossorier, “Doubly generalized LDPC codes over the AWGN channel,” *IEEE Trans. Commun.*, vol. 57, no. 5, pp. 1312-1319, May 2009.

[45] E. Paolini, M. Fossorier, and M. Chiani, “Doubly-generalized LDPC codes: stability bound over the BEC,” *IEEE Trans. Inform. Theory*, vol. 55, no. 3, pp. 1027-1046, Mar. 2009.

[46] E. Paolini, M. Fossorier, and M. Chiani, “Generalized and doubly generalized LDPC codes with random component codes for the binary erasure channel,” *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1651-1672, April 2010.

[47] E. Paolini, “Iterative decoding methods based on low-density graphs,” Ph.D. dissertation, Univ. Bologna, Bologna, Italy, May 2007.

[48] M. F. Flanagan, E. Paolini, M. Chiani, and M. Fossorier, “On the growth rate of the weight distribution of irregular doubly-generalized LDPC codes,” *46th Proc. Allerton Conf.*, Sept. 2008.

[49] C. L. Wang and M. P. C. Fossorier, “On asymptotic ensemble weight enumerators of LDPC-like codes,” *IEEE J. on Sel. Areas in Commu*, vol. 27, no. 6, pp. 899-907. Aug. 2009.

[50] Y. Wang, “Generalized constructions, decoding and implementation of LDPC codes,” Ph.D. dissertation, Dept. Elect. Eng., Univ. of Hawaii at Manoa, Honolulu, HI, 2008.

[51] D. Divsalar,H. Jin, and R. McEliece, “Coding theorems for ‘turbo-like’ code,” *in Proc. 36th Allerton Conf. Commun., Control, Comput.*, Sep. 1998, pp. 201-210.

[52] H. Jin, A. Khandekar, and R. McEliece, “Irregular repeat-accumulate codes,” *Proc. 2nd Int. Symp. Turbo Codes Related Topics*, Brest, France, Sept. 2000, pp. 1-8.

[53] M. Yang, W. E. Ryan, and Y. Li, “Design of efficiently encodable moderate-length high-rate irregular LDPC codes,” *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 564-571, Apr. 2004.

[54] J. Kim, A. Ramamoorthy and S. McLaughlin, “The design of efficiently-encodable rate-compatible LDPC codes,” *IEEE Trans.Commun.*, vol. 57, no. 2, pp.365-375, Feb. 2009.

[55] C. Shi and A. Ramamoorthy, "Design and analysis of E2RC codes," *IEEE J. Sel. Areas. Commun.*, vol. 27, no. 6, pp. 889-898, Aug. 2009.

[56] H. Pishro-Nik and F. Fekri, "Results on punctured low-density parity-check codes and improved iterative decoding techniques," *IEEE Trans. Inform. Theory.*, Vol. 53, No. 2, pp. 599-614, Feb. 2007.

[57] J. Ha, J. Kim, and S. W. McLaughlin, "Rate-compatible puncturing of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 50, no. 11, pp. 2824-2836, Nov. 2004.

[58] C.-H. Hsu and A. Anastasopoulos, "Capacity achieving LDPC codes through puncturing," *IEEE Trans. Inf. Theory*, vol. 54, no. 10, pp. 4698-4706, Oct. 2008.

[59] J. Li and K. R. Narayanan, "Rate-compatible low density parity check (RC-LDPC) codes for capacity-approaching ARQ schemes in packet data communications," in *Proceeding of Intl. Conf. on Communications, Internet and Information Technology (CIIT)*, US Virgin Islands, pp. 201-206, Nov. 2002.

[60] S. Song, D. Hwang, S. Seo, and J. Ha, "Linear-time encodable rate-compatible punctured LDPC codes with low error floors," *IEEE VTC*, pp. 749-753, May. 2008.

[61] M. R. Yazdani and A. H. Banihashemi, "On construction of rate compatible low-density parity check codes," *IEEE Commun. Lett.*, vol. 8, no. 3, pp. 159-161, March 2004.

[62] J. Ha, J. Kim, D. Klinc, and S. W. McLaughlin, "Rate-compatible punctured low-density parity-check codes with short block lengths," *IEEE Trans. Inform. Theory*, vol. 52, no. 2, pp. 728-738, Feb. 2006.

[63] E. Choi, S. Suh, and J. Kim, "Rate-compatible puncturing for low-density parity-check codes with dual-diagonal parity structure," *in Proc. IEEE Symp. Person. Indoor Mobile Radio Commun.*, vol. 4, pp. 2642-2646, Sept. 2005.

[64] 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 Trans. Wireless Commun.*, Vol. 6, no. 11, pp. 3914-3919, Nov. 2007.

[65] M. E. Khamy, J. Hou and N. Bhushan, "Design of rate-compatible structured LDPC codes for hybrid ARQ applications," *IEEE Trans. sel. commun.*, vol. 27, no. 6, pp. 965-973, Aug. 2009.

[66] G. Yue, X. Wang, and M. Madihian, "Design of rate-compatible irregular repeat accumulate codes," *IEEE Trans. Commun.*, vol. 55, no. 6, pp. 1153-1163, June. 2007.

[67] S. ten Brink, "Convergence of iterative decoding," *Electron Lett.*, vol. 35, no. 10, pp. 806-808, May 1999.

[68] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727-1737, Oct. 2001.

[69] A. Ashikmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: Model and erasure channel property," *IEEE Trans. Inform. Theory*, vol. 50, no. 11, pp. 2657-2673, Nov. 2004.

[70] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, pp. 670-678, Apr. 2004.

[71] E. Sharon, A. Ashikhmin, and S. Litsyn, "EXIT functions for binary input memoryless symmetric channels," *IEEE Trans. Commun.*, vol. 54, pp. 1207-1214, Jul. 2006.

[72] T. Richardson, A. Shokrollahi and R. Urbanke, "Design of provably good low-density parity-check codes," *Proc. IEEE Int. Symp. Inform. Theory*, Sorrento, Italy, p. 199, Jun. 2000.

[73] S. R. Kollu and H. Jafarkhani, "On the EXIT chart analysis of low-density parity-check codes," *in Proc. IEEE GLOBECOM 05*, pp.1131-1136, Dec. 2005.

[74] T. Etzion, A. Trachtenberg and A. Vardy, "Which codes have cycle-free Tanner graphs ?," *IEEE Trans. on Inform. Theory*, vol. 45, no. 6, pp. 2173-2181, September 1999.

[75] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory,* vol. 20, pp. 284-287, Mar. 1974.

[76] T. Johansson and K. Zigangirov, "A simple one sweep algorithm for optimal APP symbol decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. 44, pp. 3124-3129, Nov. 1998.

[77] A. Ashikhmin and S. Lytsin, “Simple MAP decoding of first-order Reed-Muller and Hamming codes,” *IEEE Trans. Inform. Theory*, vol. 50, pp. 1812-1818, Aug. 2004.

[78] D. Chase, “A class of algorithms for decoding block codes with channel measurement information,” *IEEE Trans. Inform. Theory*, vol. 18, pp. 170-182, Jan. 1972.

[79] G. D. Forney Jr., “Generalized minimum distance decoding,” *IEEE Trans. Inform. Theory*, pp. 125-131, Apr. 1966.

[80] R. Storn and K. Price, “Differential evolution - a simple and efficient heuristic for global optimization over continous spaces,” *J. Global Optimization*, vol. 11, pp. 341-359, Dec. 1997.

[81] K. P. Wong and Z. Y. Dong, “Differential Evolution, an alternative approach to evolutionary algorithm, ” *Proc of the 13th Int. Conf. on Intelligent Systems Application to Power Systems*, pp. 73-83. 2005.

[82] A. S. Fraser, “Simulation of genetic systems by automatic digital computers” *Aust. J. Biol. Sci.*, vol. 10, pp. 484-491. 1957.

[83] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, pp. 671-680. 1983.

[84] A. Shokrollahi and R. Storn, “Design of efficient erasure codes with differential evolution,” *Proc. 2000 IEEE Int. Symp. Inform. Theory*, Sorrento, Italy, pp. 5, Jun. 2000.

[85] J. Hou, P. H. Siegel, and L. B. Milstein, “Performance analysis and code optimization of low-density parity-check codes on Rayleigh fading channels,” *IEEE J. on Sel. Areas in Commun.*, vol. 19, no. 5, pp. 924-934, May 2001.