The Design of Rate-Compatible Structured Low-Density Parity-Check Codes

A Thesis
Presented to
The Academic Faculty

by

Jaehong Kim

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2006

The Design of Rate-Compatible Structured Low-Density Parity-Check Codes

Approved by:

Date Approved: October 25, 2006

Acknowledgements

First of all, I am grateful to my research advisor, Dr. Steven W. McLaughlin, for his thoughtful advice. His breadth of knowledge and depth of commitment to this area of research impressed me greatly and were critical to the completion of this project.

I also thank Dr. John R. Barry, Dr. Ye (Geoffrey) Li, Dr. Mark A. Clements, and Dr. Alexandra O. Boldyreva for serving on my defense committee. Their insightful comments have been valuable that made me this dissertation improved.

This work is supported by Samsung Advanced Institute of Technology and Samsung Electronics Co., Ltd. My special thanks go to vice president Seung-yong Park for his encouragement and support.

I would like to thank Dr. Jeongseok Ha for his help in research as well as living when I first came to Georgia Tech. Without his initial framework, I couldn't get much progress in the project. I am also deeply grateful to Dr. Aditya Ramamoorthy for his valuable comments and discussion. His great insight and experience made this research nicer.

I also thank to my colleagues, Demijan Klinc, Woonhaing Hur, Dr. Sunghwan Kim for their contributions to this project. Thanks also go to Jinsung Park, Janghyuk Cho, Franklin Bien, Taejoong Song, Dr. Youngsik Hur and many other Korean friends for their friendly conversations and help.

Finally, I would like to thank to my wife, SangEun Oh, for her endless love and sacrifice. My little kids, Sunwoo and Jiwoo, always bring me smile. I am really sorry that I couldn't have much time to be with them. Most of all, I thank my parents and my parents in law for their love and support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACK	Acknowledgement
APP	A Posteriori Probability
ARQ	Automatic Repeat reQuest
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BER	Bit Error Rate
BP	Belief Propagation
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
CRC	Cyclic Redundancy Check
CSI	Channel State Information
E^2RC	Efficiently-Encodable Rate-Compatible
eIRA	extended IRA
FEC	Forward Error Correction
FER	Frame Error Rate
HARQ	Hybrid Automatic Repeat reQuest
IC	Integrated Circuit
IR	Incremental Redundancy
IRA	Irregular Repeat Accumulate
LDPC	Low-Density Parity-Check
LLR	Log Likelihood Ratio
MAP	Maximum A Posteriori
MMSE	Minimum Mean Square Error
NACK	Negative Acknowledgement
PEG	Progressive Edge Growth
QC	Quasi-Cyclic
QPSK	Quadrature Phase Shift Keying
RCPC	Rate-Compatible Punctured Codes
V-BLAST	Vertical Bell Labs Layered Space-Time
VLSI	Very Large Scale Integration
WER	Word Error Rate

Summary

The main objective of our research is to design practical low-density parity-check (LDPC) codes which provide a wide range of code rates in a rate-compatible fashion. To this end, we first propose a rate-compatible puncturing algorithm for LDPC codes at short block lengths (up to several thousand symbols). The proposed algorithm is based on the claim that a punctured LDPC code with a smaller level of recoverability has better performance. The proposed algorithm is verified by comparing performance of intentionally punctured LDPC codes (using the proposed algorithm) with randomly punctured LDPC codes. The intentionally punctured LDPC codes show better bit error rate (BER) performances at practically short block lengths. From simulations, in the case of the regular code with block length of 1024, the intentionally punctured LDPC code has 3dB better E_b/N_o performance than that of the randomly punctured one for a BER of 10^{-5} at code rate 0.8. In case of irregular LDPC codes, the performance improvements of the intentionally punctured LDPC codes are 1.25dB over randomly punctured LDPC codes at code rate 0.8 for a BER of 10^{-5}.

Even though the proposed puncturing algorithm shows excellent performance, several problems are still remained for our research objective. First, how to design an LDPC code of which structure is well suited for the puncturing algorithm. Second, how to provide a wide range of rates since there is a puncturing limitation with the proposed puncturing algorithm. To attack these problems, we propose a new class of LDPC codes in which the proposed puncturing algorithm concept is imbedded. We call this class of codes efficiently-encodable rate-compatible (E^2RC) codes, which has several strong points. First, the codes can be efficiently encoded. We present low-complexity encoder implementation with shift-register circuits. In addition, we show that a simple erasure decoder can also be used for the linear-time encoding of these codes. Thus, we can share a message-passing decoder for both encoding and decoding in transceiver systems that require an encoder/decoder pair. Second, we show that the non-systematic parts of the parity-check matrix are cycle-free, which ensures good code characteristics. From simulations, the performance of the E^2RC codes (mother codes) is as good as that of extended irregular repeat-accumulate (eIRA) codes and other irregular LDPC codes. Finally, the E^2RC codes having a systematic rate-compatible puncturing structure show better puncturing performance than other irregular LDPC codes and eIRA codes in all ranges of code rates. From simulations, the puncturing of the E^2RC codes outperforms 2.8dB of E_b/N_o than random puncturing of irregular LDPC codes at a BER of 10^{-5} with code rate 0.8. Even the best effort intentional puncturing algorithm is applied to the irregular LDPC codes and eIRA codes, E^2RC codes show 1.2dB and 1.1dB better than irregular LDPC codes and eIRA codes, respectively, at a BER of 10^{-5} with code rate of 0.9.

The throughput performance of incremental redundancy (IR) hybrid automatic repeat request (HARQ) systems highly depends on the performance of high-rate codes. Since the E^2RC codes show excellent puncturing performance in all ranges of code rates, especially at high puncturing rate, we apply them to IR-HARQ systems. From simulations we observe that E^2RC codes outperform eIRA codes and general irregular LDPC codes by 2dB and 2.2dB, respectively, at the throughput of 0.8.

CHAPTER I

INTRODUCTION

Low-density parity-check (LDPC) codes by Gallager [1] had been forgotten for several decades in spite of their excellent properties, since the implementation of these codes seemed to be impossible at that time. These codes were rediscovered in the middle of the 1990s [2] and were shown to achieve Shannon limit within 0.0045dB [3]. LDPC codes are now considered good candidates for the next-generation forward error correction (FEC) technique in high throughput wireless and recording applications. Their excellent performance and iterative decoder make them appropriate for technologies such as DVB-S2, IEEE 802.16e [4], and IEEE 802.11n [5], [6].

While semiconductor technology has progressed to an extent where the implementation of LDPC codes has become possible, many practical issues still remain. First and foremost, there is a need to reduce complexity without sacrificing performance. Second, for applications such as wireless LANs, the system throughput depends upon the channel conditions and hence the code needs to have the ability to operate at different rates. Third, while the LDPC decoder can operate in linear time, it may be hard to perform low-complexity encoding of these codes. In particular, the class of irregular LDPC codes introduced by Richardson *et al.* [7] may have high memory and processing requirements, especially at short block lengths. While the encoding time can be reduced substantially using the techniques presented in [8] at long block lengths, their techniques may be hard to apply at short block lengths. The other option is to resort quasi-cyclic

(QC) LDPC or algebraic constructions that can be encoded by shift registers [9].

Irregular repeat-accumulate (IRA) codes were introduced by Jin *et al*. [10]. These codes have a linear-time encoder and their performance is almost as good as irregular LDPC codes. This class of codes was extended, called extended IRA (eIRA) codes, by Yang *et al*. [11], where they demonstrated high-rate codes with very low error floors.

A popular technique for achieving rate adaptation in a system is through the use of rate-compatible puncturing. A rate-compatible punctured code (RCPC) is suitable for applying to incremental redundancy (IR) hybrid automatic repeat request (HARQ) systems, since the parity bit set of a higher rate code is a subset of the parity bit set of a lower rate code [12]. The RCPC scheme has another advantage in that it has the same encoder and decoder while operating at different rates. The number of parity bits that the transmitter sends depends on the rate requirement. At the decoder end, parity bits that are not transmitted are treated as erasures. Thus, puncturing provides a low-complexity solution to the rate-adaptation problem.

Motivated by these observations, this dissertation first proposes the puncturing algorithm for LDPC codes with short block lengths. Based on the puncturing algorithm, a new class of codes is proposed that can be efficiently encoded as well as can be punctured in a rate-compatible fashion. The proposed LDPC codes will be shown to have a linear-time encoder and have good performance under puncturing for a wide range of rates. Finally, we verify that the proposed codes show good throughput performance when they are applied to IR-HARQ systems over time-varying channels.

This dissertation is organized as follows:

In Chapter 1, a brief outline of the dissertation and organization of each chapter are described. Chapter 2 introduces fundamentals of channel coding, LDPC codes and their iterative decoding algorithm, and efficient encoding methods of LDPC codes. Chapter 2 also presents IRA codes which have most popular structure, i.e., bi-diagonal structure, in the international standards recently. Chapter 3 covers the proposed puncturing algorithm which finds the best puncturing locations for a given parity-check matrix. The proposed puncturing algorithm consist of two steps, that is, grouping and sorting. These two-step puncturing algorithm is verified through the simulations. Based on the proposed puncturing algorithm, Chapter 4 introduces design of a new class of codes, called E^2RC codes. The code construction algorithm and efficient encoding structure with a simple shift-register circuit are dealt with in this chapter. In the simulations, we compare the puncturing performance of E^2RC codes with that of other irregular LDPC codes including eIRA codes. The proposed E^2RC codes are applied to IR-HARQ systems in Chapter 5. In this chapter, we verify that the proposed E^2RC codes have better throughput performance than other LDPC codes from the simulations. Finally, in Chapter 6, summary of our contributions and future work are discussed.

CHAPTER II

BACKGROUND

Channel coding is an essential technique to cope with errors occurring in channels of communication systems and storage systems. Channel coding has flourished in two branches. Channel errors can be corrected with forward error correction (FEC) codes. On the other hand, a receiver may request retransmission of the previous data if it fails to recover them, which is called automatic repeat request (ARQ). FEC codes can be classified into block codes, such as cyclic codes and LDPC codes, and tree codes, such as convolutional codes and Turbo codes. In this chapter, we briefly explain the block codes where LDPC codes are specified.

Let us consider linear block codes over the binary field $F_2 \triangleq \left(\{0,1\}, +, \times\right)$. Let F_2^N be the *N*-dimensional vector space over F_2. Then, an (*N*, *K*) linear block code C is defined as *K*-dimensional subspace of F_2^N, where *K* is a data word length and *N* is a codeword length. Since C is a subspace of dimension *K*, there are *K* linearly independent vectors $\boldsymbol{g}_0, \boldsymbol{g}_1, \cdots, \boldsymbol{g}_{K\text{-}1}$ which span C. Let $\boldsymbol{m} = \left[m_0, m_1, \cdots, m_{K-1}\right]$ be the data word and $\boldsymbol{c} = \left[c_0, c_1, \cdots, c_{N-1}\right]$ be the corresponding codeword in the code C. The mapping $\boldsymbol{m} \rightarrow \boldsymbol{c}$ is thus naturally written as $\boldsymbol{c} = m_0\boldsymbol{g}_0 + m_1\boldsymbol{g}_1 + \cdots + m_{K-1}\boldsymbol{g}_{K-1}$. This relationship can be represented in the matrix form $\boldsymbol{c} = \boldsymbol{m}\,G$, where G is a $K \times N$ matrix;

$$G = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{K-1} \end{bmatrix}.$$

We call the matrix G the generator matrix for C. In fact, C is the row space of G. The encoding process can be viewed as an injective mapping that maps vectors from the K-dimensional vector space into vectors from the N-dimensional vector space. The ratio

$$R = \frac{K}{N}$$

is called *code rate*.

On the other hand, the null space $C^{\perp}$ of C has dimension $N-K$ and is spanned by $N-K$ linearly independent vectors $\mathbf{h}_0, \mathbf{h}_1, \cdots, \mathbf{h}_{N-K-1}$. Since each $\mathbf{h}_i \in C^{\perp}$, we should have for any $\mathbf{c} \in C$ that

$$\mathbf{h}_i \cdot \mathbf{c}^T = 0, \quad \forall\, i.$$

This relationship can be represented in the matrix form as $H \cdot \mathbf{c}^T = \mathbf{0}$, where the matrix H is the so-called *parity-check matrix* defined as

$$H = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{N-K-1} \end{bmatrix}.$$

A low-density parity-check code is so called because the parity-check matrix H has a low density of 1s. We address the details of LDPC codes in the following chapter.

2.1 Low-Density Parity-Check Codes

Every LDPC code is uniquely specified by its parity-check matrix H or**,** equivalently, by means of the *Tanner graph* [13], as illustrated in Figure 2.1. The Tanner graph consists of two types of nodes: *variable nodes* and *check nodes*, which are connected by edges. Since there can be no direct connection between any two nodes of the same type, the Tanner graph is said to be *bipartite*. Consider an LDPC code defined by its corresponding Tanner graph. Each variable node, depicted by a circle, represents one bit of a codeword**,** and every check node, depicted by a square, represents one parity-check equation.

$$H = \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \leftarrow A \\ \leftarrow B \\ \leftarrow C \\ \leftarrow D \end{array}$$

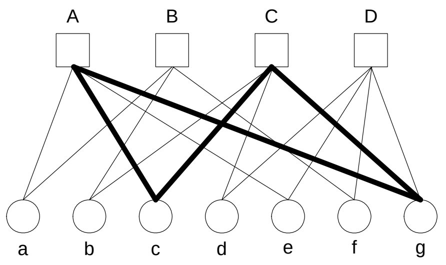


Figure 2.1 A parity-check matrix and its Tanner graph**;** Thick lines in the graph implies cycle 4**.**

Since we are considering N codeword length and K data word length, the Tanner graph contains N variable nodes and M check nodes, where $M = N - K$. Let us denote the parity-check matrix $H = \left(h_{ij}\right)_{1 \le i \le M, 1 \le j \le N}$. Then, the i-th check node is connected to the j-th variable node if and only if $h_{ij} = 1$. For example, 1 in column f and row D in the parity-check matrix in Figure 2.1 corresponds to an edge connection between variable node f and check node D in the Tanner graph. If there are *d* edges emanating from a node, variable or check node, we say that node has degree *d*. In Figure 2.1, variable node f has degree 2 and check node D has degree 4. Tanner graphs can also serve as a nice visualization tool for a variety of issues concerning LDPC codes.

Definition 2.1: A *cycle* of length *l* in a Tanner graph is a path comprised of *l* edges that begins and ends at the same node, whereby every edge has been traversed only once.

The *length* of a cycle is the number of edges in that path. Usually, LDPC codes contain many cycles of different lengths in their Tanner graph.

Definition 2.2: The *girth* in a Tanner graph is the minimum cycle length of the graph.

The girth has a great importance for the code's performance. Since Tanner graphs are bipartite, the smallest girth has length 4, as shown by the thick line in Figure 2.1. However, it is desirable to avoid short cycles in designing LDPC codes since such cycles can cause poor performance.

An *ensemble* of LDPC codes is defined by two generating polynomials of the degree distributions, called a *degree distribution pair*, for the variable and check nodes. That is,

$$\lambda(x) = \sum_{i=2}^{d_c} \lambda_i x^{i-1},$$

$$\rho(x) = \sum_{i=2}^{d_v} \rho_i x^{i-1},$$

where λ_i is the fraction of edges emanating from variable nodes of degree i, ρ_i is the fraction of edges emanating from check nodes of degree i, and d_v and d_c denote the maximum variable node and check node degrees, respectively.

As a special case when each of $\lambda(x)$ and $\rho(x)$ is monomial, an LDPC code is said to be *regular*. In fact, an LDPC code defined with a parity-check matrix that contains the same number of 1s in each column (d_v) and the same number of 1s in each row (d_c) is said to be (d_v, d_c) regular LDPC codes. The number of all 1s in H is equal to Md_c, and also to Nd_v. Hence, the code rate R of a regular LDPC code can be expressed as

$$\begin{aligned} R &= \frac{N-M}{N} \\ &= 1-\frac{M}{N} \\ &= 1-\frac{d_v}{d_c}. \end{aligned}$$

It is shown in [1] that the regular LDPC codes with the best performance have $d_v = 3$.

In general cases, where the number of 1s per column or row is not constant in the parity-check matrix H, an LDPC code is said to be *irregular*. Assume that there are N variable nodes and M check nodes. Then, the number of variable nodes of degree i is

$$\begin{aligned} N_v(i) &= N\frac{\lambda_i / i}{\sum_{\mathrm{j=2}}^{\mathrm{d_v}} \lambda_j / j} \\ &= N\frac{\lambda_i / i}{\int_0^1 \lambda(\mathrm{x})dx}, \end{aligned}$$

and the total number of edges in the Tanner graph from the variable node point is

$$E = N\sum_{i=2}^{d_v}\frac{\lambda_i}{\int_0^1 \lambda(x)\,dx}$$

$$= \frac{N}{\int_0^1 \lambda(x)\,dx}.$$

Likewise, the total number of edges from check node point is

$$E = \frac{M}{\int_0^1 \rho(x)\,dx}.$$

Thus, the code rate of an irregular LDPC code can be obtained as

$$R = 1 - \frac{M}{N}$$

$$= 1 - \frac{\int_0^1 \rho(x)\,dx}{\int_0^1 \lambda(x)\,dx}.$$

Sometimes it is convenient to have variable and check node distributions from the node perspective. That is, the fractions of variable and check nodes of each degree are

$$\lambda_i^{'} = \frac{\lambda_i / i}{\sum_{j=2}^{d_v} \lambda_j / j},$$

$$\rho_i^{'} = \frac{\rho_i / i}{\sum_{j=2}^{d_c} \rho_j / j},$$

where $\lambda_i^{'}$ and $\rho_i^{'}$ are fractions of variable and check nodes with degree *i*, respectively. Irregular LDPC codes are more flexible in their design because of the relaxed constraints and have proved to perform much better then the regular LDPC codes [7].

2.2 ITERATIVE DECODING ALGORITHM

This section summarizes the iterative decoding of LDPC codes based on the belief propagation (BP) algorithm. The decoding problem consists of finding the most likely vector $\mathbf{x}$ such that $\mathbf{H} \cdot \mathbf{x} = 0$, where $\mathbf{H}$ is a parity-check matrix defining an LDPC code. We consider binary phase shift keying (BPSK) modulated input data over the additive white Gaussian noise (AWGN) channel. Let N -tuple vector $\mathbf{x} = [x_1, \cdots, x_N]$ be a BPSK modulated codeword at the transmitter, where $x_i \in \{-1, 1\}$. The codeword bits are modulated according to

$$\begin{aligned} x_i &= (-1)^{c_i} \\ &= 1 - 2c_i , \end{aligned}$$

where c_i is the i-th bit of the codeword $\mathbf{c} = [c_1, \cdots, c_N]$. Then, the received vector at the receiver can be expressed as

$$\mathbf{y} = \mathbf{x} + \mathbf{n} ,$$

where $\mathbf{y} = [y_1, \cdots, y_N]$, $y_i \in R$, and the noise vector $\mathbf{n} = [n_1, \cdots, n_N]$ is composed of N independent additive noise n_i chosen from zero-mean Gaussian distribution with the standard deviation σ

$$P_n(a) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{a^2}{2\sigma^2} \right).$$

Much like the optimal maximum a posteriori (MAP) symbol-by-symbol decoding of trellis codes, we try to compute the a posteriori probability (APP) that c_i equals 1, given the received sequence $\mathbf{y}$ and the fact that $\mathbf{c}$ must satisfy some constraints. Without loss of generality, we focus on decoding the i-th bit of the codeword. First, let us think about the following APP ratio:

$$\frac{\Pr\left[c_i = 0 \mid \mathbf{y}, S_i\right]}{\Pr\left[c_i = 1 \mid \mathbf{y}, S_i\right]},$$

where S_i is the event that the bits in $\mathbf{c}$ satisfy the d_c parity-check constraints involving c_i. If c_i is 0, the remaining (d_c - 1) bits in a given parity-check equation involving c_i must contain an even number of 1s for S_i to occur. On the other hand, if c_i is 1, each parity-check constraint involving c_i must contain an odd number of 1s. The following Lemma will be helpful for further analysis.

Lemma 2.1 [1]: Consider a sequence of m independent bits $\mathbf{a} = \left[a_1, \cdots, a_m\right]$ with $\Pr\left[a_i = 1\right] = P_i$. The probability that $\mathbf{a}$ contains an even number of 1s is

$$\frac{1}{2} + \frac{1}{2}\prod_{i=1}^{m}\left(1 - 2P_i\right).$$

Proof: We prove this by induction. If a sequence of m independent bits $\mathbf{a} = \left[a_1, \cdots, a_m\right]$ contains an even number of 1s, the modulo-2 sum of all bits in $\mathbf{a}$, designated as A_m, is 0. For m = 2, we can have

$$\begin{aligned}\Pr\left[A_2 = 0\right] &= \Pr\left[a_1 + a_2 = 0\right] \\ &= P_1 P_2 + \left(1 - P_1\right)\left(1 - P_2\right) \\ &= \frac{1}{2} + \frac{1}{2}\left(1 - 2P_1\right)\left(1 - 2P_2\right) \\ &= \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{2}\left(1 - 2P_i\right).\end{aligned}$$

Assume that the equation holds for $m = L - 1$:

$$\Pr\left[A_{L-1}=0\right]=\frac{1}{2}+\frac{1}{2}\prod_{i=1}^{L-1}\left(1-2P_i\right).$$

Then, for $m=L$, we get

$$\begin{aligned}\Pr\left[A_L=0\right]&=\Pr\left[A_{L-1}+a_L=0\right]\\&=\frac{1}{2}+\frac{1}{2}\left(1-2\Pr\left[A_{L-1}=1\right]\right)\left(1-2P_L\right)\\&=\frac{1}{2}+\frac{1}{2}\left(1-2\left(1-\Pr\left[A_{L-1}=0\right]\right)\right)\left(1-2P_L\right)\\&=\frac{1}{2}+\frac{1}{2}\prod_{i=1}^{L}\left(1-2P_i\right).\end{aligned}$$

■

From the Lemma 2.1, we are ready to get the APP ratio for c_i.

Theorem 2.1 [1]: Assume that the received samples in the received vector $\mathbf{y}$ are statistically independent. Let S_i be the event that the bits in $\mathbf{c}$ satisfy the d_c parity-check constraints involving c_i. Then, the APP ratio for c_i given $\mathbf{y}$ and S_i is

$$\frac{\Pr\left[c_i=0\mid\mathbf{y},S_i\right]}{\Pr\left[c_i=1\mid\mathbf{y},S_i\right]}=\frac{\Pr\left[c_i=0\mid y_i\right]}{\Pr\left[c_i=1\mid y_i\right]}\cdot\frac{\prod_{j\in C_i}\left(1+\prod_{k\in R_j\backslash\{i\}}\left(1-2\Pr\left[c_{kj}=1\mid y_{kj}\right]\right)\right)}{\prod_{j\in C_i}\left(1-\prod_{k\in R_j\backslash\{i\}}\left(1-2\Pr\left[c_{kj}=1\mid y_{kj}\right]\right)\right)},$$

where c_{kj} and y_{kj} are the k-th bit in the j-th parity-check equation involving c_i and the received sample corresponding c_{kj}, respectively.

Proof: By applying Bayes' rule, we have

$$\frac{\Pr[c_i=0\,|\,\mathbf{y},S_i]}{\Pr[c_i=1\,|\,\mathbf{y},S_i]}=\frac{\Pr[c_i=0\,|\,y_i]}{\Pr[c_i=1\,|\,y_i]}\cdot\frac{\Pr[S_i\,|\,c_i=0,\mathbf{y}]/\Pr[S_i]}{\Pr[S_i\,|\,c_i=1,\mathbf{y}]/\Pr[S_i]}$$

$$=\frac{\Pr[c_i=0\,|\,y_i]}{\Pr[c_i=1\,|\,y_i]}\cdot\frac{\Pr[S_i\,|\,c_i=0,\mathbf{y}]}{\Pr[S_i\,|\,c_i=1,\mathbf{y}]}.$$

From Lemma 2.1, the probability of an odd number of 1s in the other d_c-1 bits of the *j*-th parity-check equation is

$$\frac{1}{2}-\frac{1}{2}\prod_{k\in R_j\backslash i}\left(1-2\Pr\left[c_{kj}=1\,|\,y_{kj}\right]\right).$$

Since y_i is assumed to be statistically independent, the probability that all d_c parity-check constraints are satisfied is the product of all such probabilities:

$$\frac{\Pr[S_i\,|\,c_i=0,\mathbf{y}]}{\Pr[S_i\,|\,c_i=1,\mathbf{y}]}=\frac{\prod_{j\in C_i}\left(1+\prod_{k\in R_j\backslash\{i\}}\left(1-2\Pr\left[c_{kj}=1\,|\,y_{kj}\right]\right)\right)}{\prod_{j\in C_i}\left(1-\prod_{k\in R_j\backslash\{i\}}\left(1-2\Pr\left[c_{kj}=1\,|\,y_{kj}\right]\right)\right)}.$$

■

The computation of the APP ratio as given by the formula in the above Theorem 2.1 is complex. Gallager instead provided an iterative algorithm that is exactly the BP based decoding approach nowadays. Before we give the iterative decoding algorithm, we will need the following result.

Lemma 2.2: Suppose $y_i=x_i+n_i$, where $n_i\sim\mathcal{N}\left(0,\sigma^2\right)$ and $\Pr[x_i=+1]=\Pr[x_i=-1]$ $=1/2$, then

$$\Pr[x_i=x\,|\,y]=\frac{1}{1+e^{-2yx/\sigma^2}}.$$

Proof:

$$\begin{aligned}
\Pr\left[x_i = x \mid y\right] &= \frac{p\left(y \mid x_i = x\right)\Pr\left[x_i = x\right]}{p\left(y\right)} \\
&= \frac{\frac{1}{2}e^{-(y-x)^2/2\sigma^2}}{\frac{1}{2}e^{-(y-1)^2/2\sigma^2} + \frac{1}{2}e^{-(y+1)^2/2\sigma^2}} \\
&= \frac{e^{xy/\sigma^2}}{e^{y/\sigma^2} + e^{-y/\sigma^2}} \\
&= \frac{1}{e^{y(1-x)/\sigma^2} + e^{-y(1+x)/\sigma^2}} \\
&= \frac{1}{1+e^{-2yx/\sigma^2}}.
\end{aligned}$$

■

With these results, we formulate an iterative decoding algorithm for LDPC codes, which is known as the *message passing algorithm*. The information is iteratively exchanged between the neighboring nodes in the Tanner graph by passing messages along the edges. Each message can be associated to the codeword bit corresponding to the variable node incident to the edge carrying the message. A message sent from either check or variable node along an adjacent edge should not depend on the message previously received along that edge.

A message from the variable node *i* to the check node *j* in the *l*-th iteration, carrying the probability that the value of the *i*-th bit is *k*, is denoted by $q_{ij}^{(l)}(k)$. On the other hand, a message from the check node *j* to the variable node *i* in the *l*-th iteration, carrying the probability that the value of the *i*-th bit is *k*, is $r_{ji}^{(l)}(k)$. Initially, the variable nodes only have information about the channel output values of their corresponding bits. Since no additional information from the neighboring check nodes is available, they send the message along adjacent edges:

$$q_{ij}^{(0)}(0) = \frac{1}{1+e^{-2y_i/\sigma^2}},$$

$$q_{ij}^{(0)}(1) = \frac{1}{1+e^{2y_i/\sigma^2}}.$$

Subsequently, the messages are iteratively exchanged between check nodes and variable nodes. In Figure 2.2, we see a check node connected to d_c variable nodes. In each iteration, the check node will receive messages from its neighboring variable nodes, process that information, and pass the updated message back to the neighboring variable nodes:

$$r_{ji}^{(l)}(0) = \frac{1}{2} + \frac{1}{2}\prod_{k\in R_j\setminus i}\left(1-2q_{kj}^{(l-1)}(1)\right),$$

$$r_{ji}^{(l)}(1) = \frac{1}{2} - \frac{1}{2}\prod_{k\in R_j\setminus i}\left(1-2q_{kj}^{(l-1)}(1)\right).$$

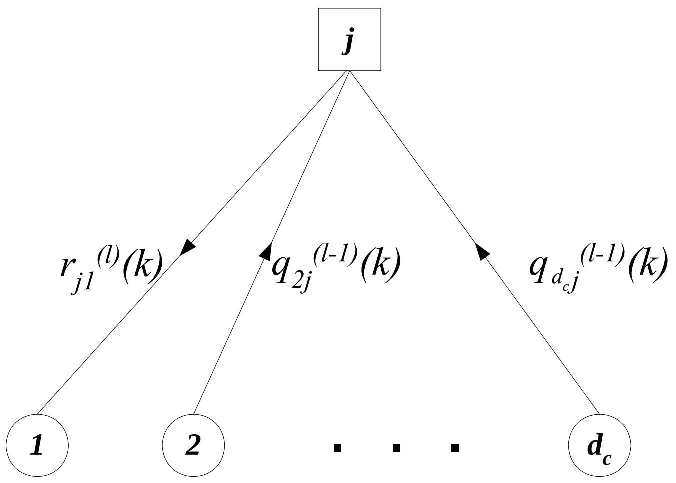


Figure 2.2 The Check node message update.

Furthermore, the message passing from the variable node is as shown in Figure 2.3. Similarly, each variable node collects messages from its neighboring check nodes, calculates the probability that its corresponding bit is 1 and sends it to the neighboring check nodes:

$$\frac{1-q_{ij}^{(l)}(1)}{q_{ij}^{(l)}(1)} = \frac{1-q_{ij}^{(0)}(1)}{q_{ij}^{(0)}(1)} \cdot \frac{\prod_{k \in C_i \setminus j} \left(1-r_{ki}^{(l)}(1)\right)}{\prod_{k \in C_i \setminus j} r_{ki}^{(l)}(1)},$$

whereby the message received from the check node j was left out, since the updated message has to depend solely on extrinsic information. Then, we easily get

$$q_{ij}^{(l)}(0) = q_{ij}^{(0)}(0) \cdot \prod_{k \in C_i \setminus j} r_{ki}^{(l)}(0),$$

$$q_{ij}^{(l)}(1) = q_{ij}^{(0)}(1) \cdot \prod_{k \in C_i \setminus j} r_{ki}^{(l)}(1).$$

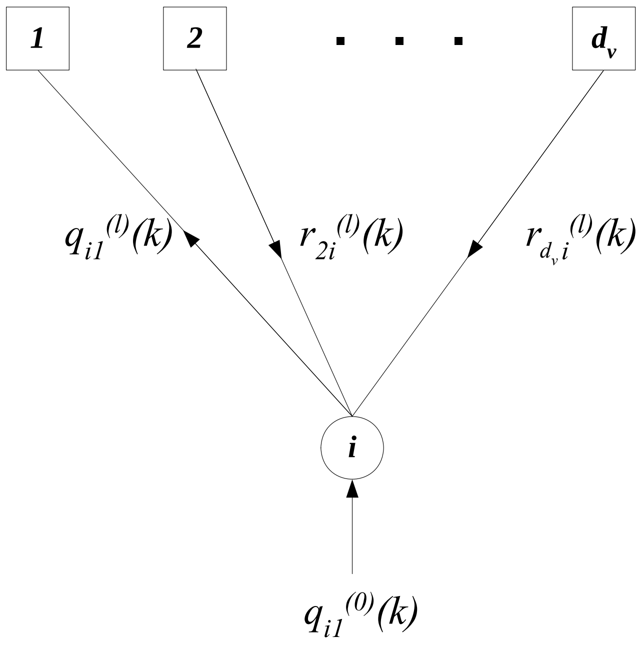


Figure 2.3 Variable node message update.

After receiving the check node messages, a variable node i calculates the probability, $\Pr\left[c_i = 1 \mid \mathbf{y}, S_i\right]$ by taking into consideration all incoming check node messages. If $H \cdot \hat{c} = 0$, where

$$\hat{c} = \begin{cases} 1, & \text{if } \Pr\left[c_i = 1 \mid \mathbf{y}, S_i\right] > \dfrac{1}{2} \\ 0, & \text{otherwise,} \end{cases}$$

or if the maximum number of iterations has been reached, the algorithm stops; otherwise, a new iteration is started.

This algorithm will converge to the true maximum APP with the growing number of iterations only if the messages are statistically independent, which is the case only if the graph corresponding to H is cycle-free. However, the graphs of practical codes will never be completely cycle-free. Therefore, the algorithm will give us an approximate solution for the APP, which fortunately still yields a remarkable performance.

Up to this point, the decoder analysis has been treated in the probability domain. In the algorithm, we can notice a substantial number of multiplications, which tend to become numerically unstable and are harder to implement in hardware compared to the additions. To simplify those equations, we introduce the following notation:

$$L\left(c_i\right) = \log \frac{\Pr\left[c_i = 0 \mid y_i\right]}{\Pr\left[c_i = 1 \mid y_i\right]},$$

called the *log-likelihood ratio (LLR)*. The probability distribution for a binary random variable is uniquely specified by $L\left(c_i\right)$. Its sign indicates the most likely value for c_i, while its magnitude is a measure of certainty for that decision. Also, let us define

$$L\left(r_{ji}\right) \triangleq \log \frac{r_{ji}(0)}{r_{ji}(1)},$$

$$\text{and } L\left(q_{ij}\right) \triangleq \log \frac{q_{ij}(0)}{q_{ij}(1)}.$$

Then, the initialization step becomes

$$\begin{aligned} L\left(q_{ij}\right) &= L\left(c_i\right) \\ &= \log \frac{\left(1+e^{-2y_i/\sigma^2}\right)^{-1}}{\left(1+e^{+2y_i/\sigma^2}\right)^{-1}} \\ &= \frac{2y_i}{\sigma^2}, \end{aligned}$$

where σ denotes the standard deviation of the zero-mean white Gaussian noise. The constant of proportionality $2/\sigma^2$ is called the *channel reliability*. Let us consider the following relationship:

$$\begin{aligned} \tanh\left(\frac{1}{2}\log\frac{p_0}{p_1}\right) &= p_0 - p_1 \\ &= 1 - 2p_1. \end{aligned}$$

Using this equation, we have

$$\begin{aligned} \tanh\left(\frac{1}{2}L\left(r_{ji}\right)\right) &= \tanh\left(\frac{1}{2}\log\frac{r_{ji}(0)}{r_{ji}(1)}\right) \\ &= 1 - 2r_{ji}(1) \\ &= \prod_{k\in R_j \backslash i}\left(1 - 2q_{kj}(1)\right) \\ &= \prod_{k\in R_j \backslash i}\tanh\left(\frac{1}{2}L\left(q_{kj}\right)\right). \end{aligned}$$

Thus, the check node update equation can be

$$L\left(r_{ji}\right)=2\tanh^{-1}\left\{\prod_{k\in R_j\backslash i}\tanh\left(\frac{1}{2}L\left(q_{kj}\right)\right)\right\}.$$

The problem with this expression is that we are still left with a product. We can remedy this by considering $L\left(q_{ij}\right)$ as sign and magnitude separately. Let us rewrite $L\left(q_{ij}\right)$ as

$$L\left(q_{ij}\right)=\alpha_{ij}\beta_{ij},$$

where $\alpha_{ij}\triangleq sign\left(L\left(q_{ij}\right)\right)$ and $\beta_{ij}\triangleq\left|L\left(q_{ij}\right)\right|$.

Then, the previous check node update results can be rewritten as

$$\tanh\left(\frac{1}{2}L\left(r_{ji}\right)\right)=\prod_{k\in R_j\backslash i}\alpha_{ij}\cdot\prod_{k\in R_j\backslash i}\tanh\left(\frac{1}{2}\beta_{kj}\right).$$

Then, we have

$$\begin{aligned}L\left(r_{ji}\right)&=\left(\prod_{k\in R_j\backslash i}\alpha_{kj}\right)\cdot 2\tanh^{-1}\prod_{k\in R_j\backslash i}\tanh\left(\frac{1}{2}\beta_{kj}\right)\\&=\left(\prod_{k\in R_j\backslash i}\alpha_{kj}\right)\cdot 2\tanh^{-1}\log^{-1}\sum_{k\in R_j\backslash i}\log\tanh\left(\frac{1}{2}\beta_{kj}\right)\\&=\left(\prod_{k\in R_j\backslash i}\alpha_{kj}\right)\cdot\Phi\left(\sum_{k\in R_j\backslash i}\Phi\left(\beta_{kj}\right)\right),\end{aligned}$$

where we have defined

$$\Phi\left(x\right)\triangleq-\log\tanh\left(\frac{1}{2}x\right)=\log\frac{e^x+1}{e^x-1}.$$

We have shown how the updated check node message can be calculated in the log domain. On the other hand, the formula for a variable node message update in the log domain can be easily derived as

$$\begin{aligned} L\left(q_{ij}\right) &= \log\frac{q_{ij}(0)}{q_{ij}(1)} \\ &= \log\frac{\Pr\left[c_i = 0 \mid y_i\right]}{\Pr\left[c_i = 1 \mid y_i\right]} + \log\frac{\prod_{k\in C_i\setminus j} r_{ki}(0)}{\prod_{k\in C_i\setminus j} r_{ki}(1)} \\ &= L\left(c_i\right) + \sum_{k\in C_i\setminus j} L\left(r_{ki}\right). \end{aligned}$$

The first term on the right-hand side represents the contribution from the *i*-th channel output, while the second term represents messages received from the neighboring check nodes. Here, all incoming check node messages are taken into account. After each iteration, the decoder has to evaluate the LLR values for each variable node and check if all the parity-check constraints are fulfilled by verifying if $H \cdot \hat{c} = 0$, where

$$\hat{c} = \begin{cases} 1, & \text{if } L\left(c_i\right) < 0, \\ 0, & \text{otherwise.} \end{cases}$$

Again, if all parity-check constraints are fulfilled or if the maximum number of iterations has been reached, the decoder stops; otherwise, a new iteration is started.

2.3 Efficient Encoding Method

In general, encoder for LDPC codes can be difficult to implement efficiently. Implementing an LDPC encoder with a conventional way using a generator matrix G has a complexity quadratic in block length. If the parity-check matrix H is sparse, usually G is dense, meaning that it contains a significant number of 1s and that it requires more XOR operators to implement. To attack this problem, Richardson *et al.* propose an approach in [8] where they show how LDPC codes can be encoded with linear complexity if H is brought to an approximate lower triangular form. Alternatively, encoding can be simplified via algebraic and combinatorial code construction methods. Such "structured" codes can be realized with simple encoders based on shift-register circuits. This section briefly introduces the efficient encoding method in [8].

Suppose a given parity-check matrix H is $M \times N$, and the associated codeword $\mathbf{c}$ such that $H \cdot \mathbf{c}^T = \mathbf{0}$. Let us denote the size of message symbols $K = N - M$. The straightforward way of constructing an encoder for such a code is to change H into an equivalent lower triangular form with Gaussian elimination, as shown in Figure 2.4.

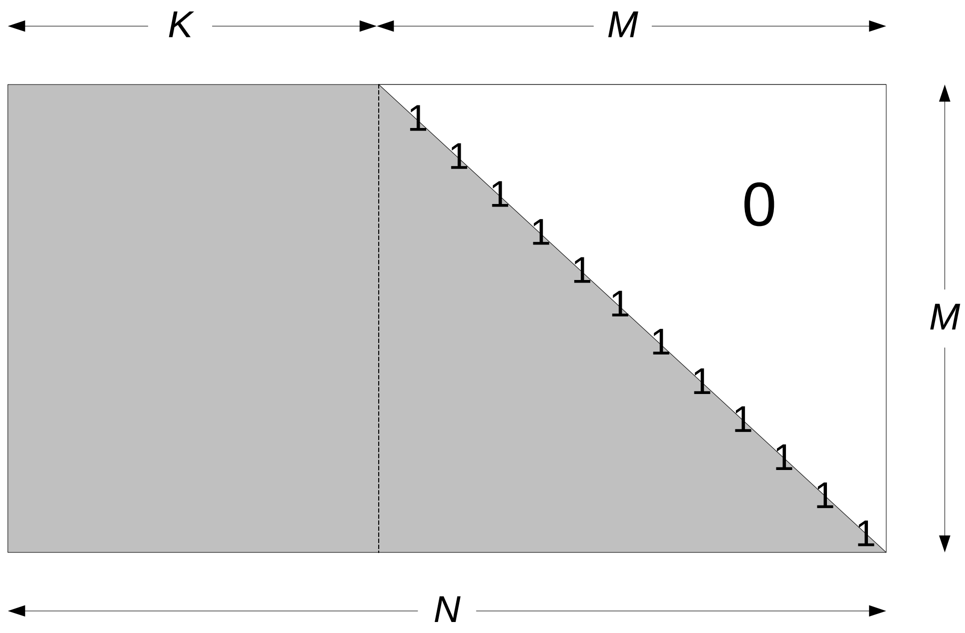


Figure 2.4 Lower triangular form.

Let us denote the systematic part of the codeword $\mathbf{c}$ as $\mathbf{m}$ and the non-systematic part of the codeword as $\mathbf{p}$ such that $\mathbf{c} = (\mathbf{m} \mid \mathbf{p})$. For K desired message symbols, M parity symbols can be determined using backward substitution. That is,

$$p_i = \sum_{j=1}^{K} H_{i,j} s_j + \sum_{j=1}^{i-1} H_{i,j+K} p_j \text{ , where } 0 \le i < M \text{ .}$$

However, the complexity of such an encoding scheme is huge. That is, converting the matrix H into the desired form requires $O(N^3)$ operations, and the actual encoding requires $O(N^2)$ operations. Richardson and Urbanke proposed the low-complexity encoding method, which requires $O(N)$ operations [8]. We can convert the given parity-check matrix to the form shown in Figure 2.5 by performing row and column permutations.

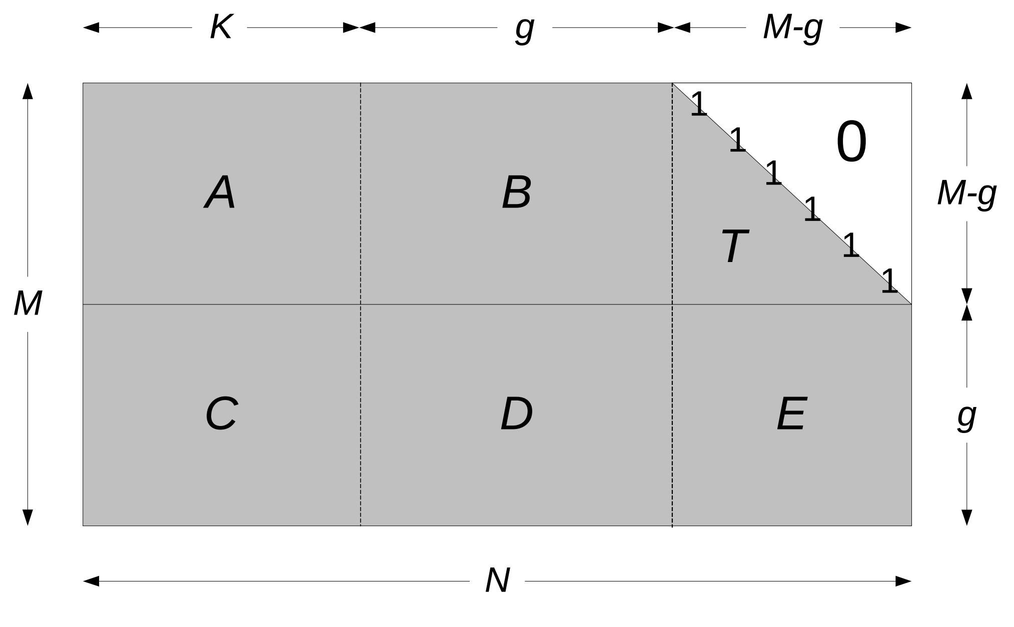


Figure 2.5 Approximate lower triangular form.

Suppose we bring the matrix in the form

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix},$$

where A is $(M-g)\times K$, B is $(M-g)\times g$, T is $(M-g)\times(M-g)$, C is $g\times K$, D is $g\times g$, and E is $g\times(M-g)$. As in Figure 2.5, T is lower triangular with 1s along the diagonal, and these matrices are sparse. Let us consider the following matrix:

$$\begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix},$$

and multiply this matrix to the left of H. Then, we have

$$\begin{pmatrix} A & B & T \\ -ET^{-1}A+C & -ET^{-1}B+D & 0 \end{pmatrix}.$$

Let $c = (m, p_1, p_2)$, where m denotes the systematic part of a codeword c, and the parity part splits into two parts, namely, p_1 of length g and p_2 of length $(M-g)$. From the equation $H\cdot c^T = 0$, we have the following two equations:

$$\begin{cases} Am^T + Bp_1^T + Tp_2^T = 0 \\ \left(-ET^{-1}A+C\right)m^T + \left(-ET^{-1}B+D\right)p_1^T = 0. \end{cases}$$

Then, we can obtain p_1 and p_2 as follows:

$$\begin{cases} p_1^T = -\phi^{-1}\left(-ET^{-1}A+C\right)m^T \\ p_2^T = -T^{-1}\left(Am^T + Bp_1^T\right), \end{cases}$$

where we define $\phi = -ET^{-1}B+D$ and assume for the moment that ϕ is nonsingular. Rather than precomputing $-\phi^{-1}\left(-ET^{-1}A+C\right)$ and then multiplying with m^T, we can reduce the complexity more by breaking the computation into several smaller steps. By doing so, we can accomplish the encoding step in complexity $O(N)$.

2.4 IRREGULAR REPEAT ACCUMULATE CODES

Jin *et al.* introduced a promising class of codes, called Irregular Repeat Accumulate (IRA) codes, which has several strong points [10]. First, IRA codes can be encoded in linear time like Turbo codes. Second, their performance is superior to turbo codes of comparable complexity and as good as best-known irregular LDPC codes. In addition, they have a simple structure, that is, the parity part of IRA codes has a bi-diagonal structure, illustrated in the Figure 2.6, and their message part adopts arbitrary permutation to maintain the check node degree concentrated.

The eIRA codes by Yang *et al.* extended the IRA codes [11]. The eIRA codes achieve good performance by assigning degree-2 nodes to nonsystematic bits and ensuring that the degree-2 nodes do not form a cycle amongst themselves. Furthermore, they avoid cycles of length-4 and make the systematic bits correspond to variable nodes of degree higher than two. They ensure efficient encoding by forming the parity in the bi-diagonal structure like IRA codes as shown in Figure 2.6.

$$H_2 = \begin{bmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & & \ddots & & \\ & & & 1 & 1 & \\ & & & & 1 & 1 \end{bmatrix}$$

Figure 2.6 Bi-diagonal structure in IRA (or eIRA) codes.

Let H_1 and H_2 be the systematic part and nonsystematic part of the parity-check matrix

$$H = [H_1 \mid H_2]$$

of eIRA codes. The systematic generator matrix G is given by

$$G = [I_k \mid P],$$

where I_k denotes an $k \times k$ identity matrix and P denotes parity part. Since

$$\begin{aligned} G \cdot H^T &= [I_k \mid P] \cdot \begin{bmatrix} H_1^T \\ H_2^T \end{bmatrix} \\ &= H_1^T + P \cdot H_2^T = 0, \end{aligned}$$

we can get $P = H_1^T \cdot H_2^{-T}$. Then, the systematic codeword is represented by $c = m \cdot G = m \cdot [I_k \mid P] = \left[m \mid m \cdot H_1^T \cdot H_2^{-T} \right]$. We can implement a simple encoder as shown in Figure 2.7. The encoding complexity can be made low if the multiplication with H_2^{-T} can be implemented efficiently. For eIRA codes, the multiplication with H_2^{-T} can be implemented with a differential encoder whose transfer function is $\frac{1}{1 \oplus D}$.

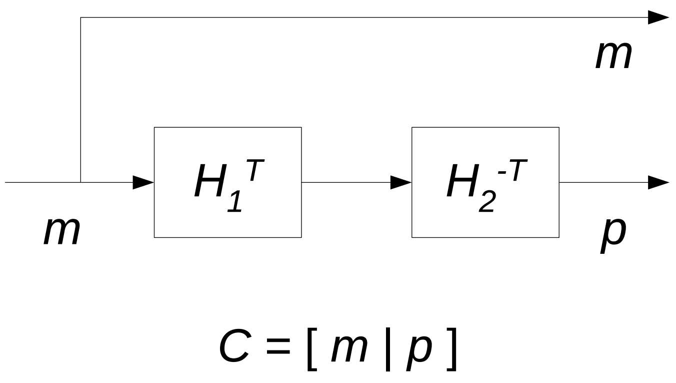


Figure 2.7 Encoder example of eIRA codes.

CHAPTER III

RATE-COMPATIBLE PUNCTURING ALGORITHM

Over time-varying channels such as wireless channels, a communication system needs to be operated adaptively at different rates. One possible solution to this problem is to use several dedicated codes for different rates. However, this requires several different encoder/decoder pairs, which increases the complexity. On the other hand, we can also use codeword symbol puncturing to obtain a channel coding scheme that provides a family of codes with different coding rates according to the channel state information (CSI). In terms of complexity, this would be much more efficient than providing several dedicated codes for different rates since it requires only one encoder/decoder pair.

Rate-compatible punctured codes (RCPC) were introduced by Hagenauer [12]. In RCPC codes, the parity bits of a higher-rate code are a subset of the parity bits of the lower-rate code. This subset property is suitable for applying RCPC to incremental redundancy (IR) automatic repeat request (ARQ) systems.

The rate-compatible puncturing of LDPC codes based on degree distributions was introduced by Ha *et al.* [14]. They proposed a design rule for good puncturing distributions with a simplified equation, called a steady-state equation. However, this method assumes infinitely long block lengths, and extending this to short block lengths is a significant challenge. In this thesis, we propose efficient puncturing algorithm for short block length (up to several thousand symbols) LDPC codes.

3.1 PRELIMINARY ANALYSIS

Suppose that the Tanner graph of the mother code is denoted by $G = (V \bigcup C, E)$, where *V* denotes the set of variable nodes, *C* denotes the set of check nodes, and *E* denotes the set of edges. Let $S \subseteq V$ be a subset of the variable nodes. Then, the set of check node neighbors of *S* will be denoted by *N(S)*. Similar notation will be used to denote the set of variable node neighbors of a subset of the check nodes. The set of unpunctured nodes is denoted by V_0; then, the set of punctured variable nodes is denoted by $V \setminus V_0$.

Definition 3.1: [*1-step recoverable node*] A punctured variable node $p \in V \setminus V_0$ is called a *1*-step recoverable (*1*-SR) node if there exists $c \in N(\{p\})$ such that $N(\{c\}) \setminus \{p\} \subseteq V_0$.

1-step recoverable nodes are so named because, in the absence of any channel errors, these nodes can be decoded in one step of iterative decoding. This definition can be generalized to *k*-step recoverable (*k*-SR) nodes (see Figure 3.1). Let V_1 be the set of *1*-SR nodes among the punctured variable nodes. Similarly, let V_k be the set of *k*-step recoverable nodes, which are defined as follows:

Definition 3.2: [*k-step recoverable node*] A punctured variable node $p \in V \setminus V_0$ is called *k*-step recoverable (*k*-SR) node if there exists $c \in N(\{p\})$ such that $N(\{c\}) \setminus \{p\} \subseteq \bigcup_{i=0}^{k-1} V_i$ and that there exists $q \in N(\{c\}) \setminus \{p\}$, where $q \in V_{k-1}$.

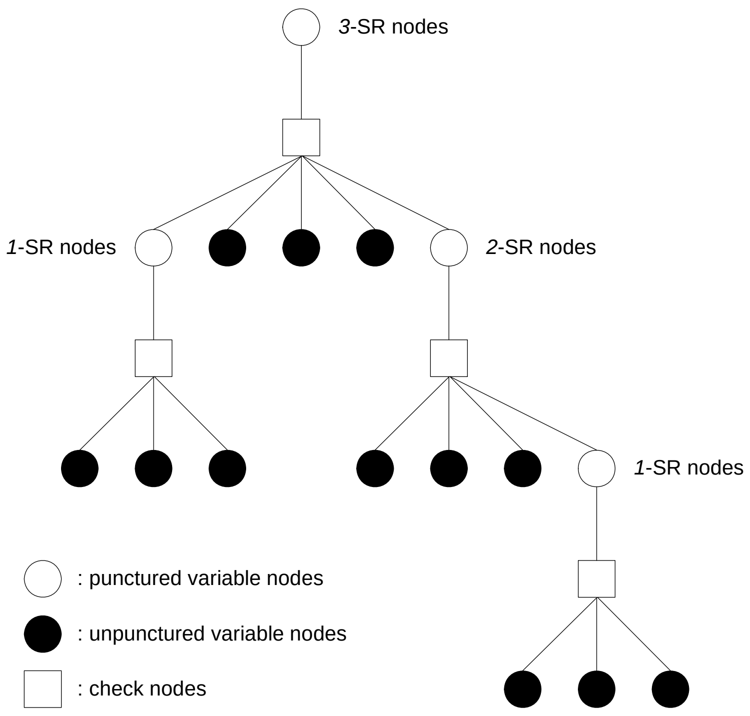


Figure 3.1 *k*-SR node in the recovery tree.

Note that the *k*-SR node will be recovered after exactly k iterations of iterative decoding assuming that the channel does not cause any errors. So, a large number of low-SR nodes are intuitively likely to reduce the overall number of iterations, which results in good puncturing performance.

Let us consider building a tree originating from a *k*-SR node v. First, we link v with its guaranteed surviving check node c and subsequently link c with all variable nodes from the set $N(\{c\})\backslash\{v\}$. In the next step, this process is repeated on every new punctured variable node in the tree until every branch terminates with an unpunctured variable node. The resulting tree is called the *recovery tree* of v. We show an example of a recovery tree in Figure 3.1. The number of unpunctured nodes in the recovery tree of v will be important, so we designate it as $S(v)$ and $S(v)=12$ in Figure 3.1. Assuming that v is recovered with the message-passing decoding algorithm on the recovery tree of v, we define the recovery-error probability of v, $P_e(v)$ as follows.

Definition 3.3: [*Recovery-error probability* $P_e(v)$ of a *k*-SR node v] For $v \in V_k$ and $k \geq 1$, $P_e(v)$ is the probability of v being recovered with a wrong symbol in the *k*-th iteration by the message through the surviving check node from unpunctured nodes in the recovery tree of v.

When we transmit a punctured LDPC code over a binary erasure channel (BEC) with an erasure probability of ζ, the probability that a variable node v in V_k is recovered in its recovery tree is expressed in a recursive form as shown in the following definition 3.4.

Definition 3.4:

$$\Psi(v,\zeta) \triangleq (1-\zeta)^{S(v)} = \begin{cases} (1-\zeta), & \text{for } v \in V_0 \\ \prod_{j=1}^{d_c-1} \Psi(\gamma_j,\zeta) & \text{for } v \in V_k \text{ and } k > 0, \end{cases}$$

where V_0 and V_k are sets of unpunctured nodes and *k*-SR nodes, respectively, and d_c is a degree of the survived check node of v, γ_j's are the neighbors of the survived check node except for v, and $\gamma_j \in V_h$ for $0 \leq h \leq k-1$.

The following Theorems 3.1-3 tell that the *k*-SR node with a smaller $S(v)$ will have a smaller recovery-error probability. The recovery-error probability of a variable node $v \in V_{k>0}$ over a BEC with ζ can be obtained as follows.

Theorem 3.1: The $P_e(v)$ of $v \in V_{k>0}$ over a BEC with ζ is

$$P_e(v) = \frac{1}{2}\left(1 - \Psi(v, \zeta)\right),$$

and the probability that the variable node v is recovered over a BEC with ζ is $\Psi(v, \zeta)$.

Proof: We will prove this fact by induction. For $k = 1$, all variable nodes in the recovery tree of v are in V_0. Thus,

$$\begin{aligned} P_e(v) &= \frac{1}{2}\left(1 - (1-\zeta)^{d_c - 1}\right) \\ &= \frac{1}{2}\left(1 - (1-\zeta)^{S(v)}\right) \\ &= \frac{1}{2}\left(1 - \Psi(v, \zeta)\right), \end{aligned}$$

where d_c is a degree of a survived check node of v. Now, assume that for a variable node $\gamma_j \in V_k$,

$$\begin{aligned} P_e(\gamma_j) &= \frac{1}{2}\left(1 - \Psi(\gamma_j, \zeta)\right) \\ &= \frac{1}{2}\left(1 - (1-\zeta)^{S(\gamma_j)}\right). \end{aligned}$$

Then for $v \in V_{k+1}$,

$$\begin{aligned}
P_e(\gamma_j) &= \frac{1}{2}\left(1-\prod_{j=1}^{d_c-1}\Psi(\gamma_j,\zeta)\right) \\
&= \frac{1}{2}\left(1-(1-\zeta)^{\sum_{j=1}^{d_c-1}S(\gamma_j)}\right) \\
&= \frac{1}{2}\left(1-(1-\zeta)^{S(v)}\right) \\
&= \frac{1}{2}\left(1-\Psi(v,\zeta)\right),
\end{aligned}$$

where d_c is a degree of the survived check node of v, γ_j's are the neighbors of the survived check node except for v, and the number of unpunctured nodes in the recovery tree of v is $S(v)=\sum_{j=1}^{d_c-1}S(\gamma_j)$.

■

In Theorem 3.2, we consider the recovery-error probability of a variable node $v \in V_{k>0}$ over a binary symmetric channel (BSC) with an erasure probability of ζ.

Theorem 3.2: The $P_e(v)$ of $v \in V_{k>0}$ over a BSC with ζ is

$$P_e(v)=\frac{1}{2}\left(1-\Psi(v,2\zeta)\right).$$

Proof: We will prove this by induction. A recovery-error probability of a variable node $v \in V_1$ will be

$$\begin{aligned} P_e(v) &= \frac{1}{2}\left(1-(1-2\zeta)^{d_c-1}\right) \\ &= \frac{1}{2}\left(1-(1-2\zeta)^{S(v)}\right) \\ &= \frac{1}{2}\left(1-\Psi(v,2\zeta)\right), \end{aligned}$$

where d_c is a degree of a survived check node of v. Now, assume that

$$\begin{aligned} P_e(\gamma_j) &= \frac{1}{2}\left(1-\Psi(\gamma_j,2\zeta)\right) \\ &= \frac{1}{2}\left(1-(1-2\zeta)^{S(\gamma_j)}\right), \end{aligned}$$

for a variable node $\gamma_j \in V_k$. Then for $v \in V_{k+1}$,

$$\begin{aligned} P_e(\gamma_j) &= \frac{1}{2}\left(1-\prod_{j=1}^{d_c-1}\Psi(\gamma_j,2\zeta)\right) \\ &= \frac{1}{2}\left(1-(1-2\zeta)^{\sum_{j=1}^{d_c-1}S(\gamma_j)}\right) \\ &= \frac{1}{2}\left(1-(1-2\zeta)^{S(v)}\right) \\ &= \frac{1}{2}\left(1-\Psi(v,2\zeta)\right), \end{aligned}$$

where d_c is a degree of the survived check node of v, γ_j's are the neighbors of the survived check node except for v, and the number of unpunctured nodes in the recovery tree of v is $S(v)=\sum_{j=1}^{d_c-1}S(\gamma_j)$.

■

To obtain the recovery-error probability over AWGN channel, we need to define the function $\phi(x)$ in [15].

Definition 3.5 [15]:

$$\phi(x) = \begin{cases} 1 - 1/\sqrt{4\pi x}\int_{\mathbb{R}} \tanh\frac{u}{2} e^{-(u-x)^2/4x}\, du, & x > 0 \\ 1, & x = 0. \end{cases}$$

Theorem 3.3: The recovery-error probability of a variable node $v \in V_k$ over an AWGN channel with Gaussian Approximation in [15] is $P_e(v) = Q\left(\sqrt{m_u(v)/2}\right)$, where $Q(\cdot)$ is the Q-function, $m_u(v) = \phi^{-1}\left(1 - \phi\left(1 - \Psi\left(v, \phi\left(m_{u_0}\right)\right)\right)\right)$ for $m_{u_0} = 2/\sigma^2$ and noise variance σ^2.

Proof: An updated mean of a variable node $v \in V_1$ will be $\phi^{-1}\left(1 - \left(1 - \phi\left(m_{u_0}\right)\right)^{d_c - 1}\right)$ $= \phi^{-1}\left(1 - \left(1 - \phi\left(m_{u_0}\right)\right)^{S(v)}\right) = \phi^{-1}\left(1 - \Psi\left(v, \phi\left(m_{u_0}\right)\right)\right)$. Assume that for a variable node $\gamma \in V_k$, $m_u(\gamma) = \phi^{-1}\left(1 - \Psi\left(\gamma, \phi\left(m_{u_0}\right)\right)\right)$. Then, for a variable node $v \in V_{k+1}$,

$$\begin{aligned} m_u(v) &= \phi^{-1}\left(1 - \prod_{j=1}^{d_c-1}\left(1 - \phi\left(m_{u_0}\right)\right)^{S(\gamma_j)}\right) \\ &= \phi^{-1}\left(1 - \left(1 - \phi\left(m_{u_0}\right)\right)^{\sum_{j=1}^{d_c-1} S(\gamma_j)}\right) \\ &= \phi^{-1}\left(1 - \left(1 - \phi\left(m_{u_0}\right)\right)^{S(v)}\right) \\ &= \phi^{-1}\left(1 - \Psi\left(v, \phi\left(m_{u_0}\right)\right)\right), \end{aligned}$$

where, m_{u_0} is the mean of a log-likelihood ratio message from the channel to unpunctured variable nodes, d_c is a degree of a surviving check node of v, γ_j's are the neighbors of the surviving check node except for v, $\sum_{j=1}^{d_c-1} S(\gamma_j) = S(v)$, and $\gamma_j \in V_h$ for $1 \le h \le k$.

■

3.2 Algorithm For Finding Puncturing Locations

In this section, we present a two-step search algorithm for finding the puncturing pattern and order, that is, grouping and sorting. In the grouping step, we separate all variable nodes into groups $V_0, V_1, \ldots, V_k$. In the sorting step, we determine the order of puncturing nodes within each group.

The recovery-error probability is a probability for *k*-SR node to be recovered in the *k*-th iteration with a wrong message. Based on the Theorems 3.1-3, the search algorithm chooses a new puncturing node with the smallest recovery-error probability out of several candidates. Before describing the search algorithm, we address two definitions below.

Definition 3.6:[*Effective column weight*] For a parity-check matrix $H = \{h_{j,k}\}_{1\le j\le M,\, 1\le k\le N}$, let $\Lambda^{\gamma} = \{\rho : h_{\rho,j} = 1 \text{ and } 1 \le \rho \le M\}$ be a subset of row indices R. The effective column weight $cw_{eff}(c, R)$ is defined as $|\Lambda^c \cap R|$, where $|\cdot|$ is a cardinality of a set.

Definition 3.7:[*Effective row weight*] For a parity-check matrix $H = \{h_{j,k}\}_{1\le j\le M,\, 1\le k\le N}$, let $\Gamma^{\rho} = \left\{ \gamma : h_{\rho,\gamma} = 1 \text{ and } 1 \le \gamma \le N \right\}$ be a subset of column indices C. The effective row weight $rw_{eff}\left(r, C\right)$ is defined as $\left|\Gamma^{r} \cap C\right|$, where $\left| \cdot \right|$ is a cardinality of a set.

Proposed Grouping Algorithm

STEP 0 [**Initialization**] For a given $M \times N$ parity-check matrix H, $k = 1$, R_0 and R_1 are empty sets, $R_{\infty} = \left\{1, 2, ..., M\right\}$, $\Gamma^{\rho} = \left\{ \gamma : h_{\rho,\gamma} = 1, 1 \le \gamma \le N \right\}$, $\Lambda^{\gamma} = \left\{ \rho : h_{\rho,\gamma} = 1, 1 \le \rho \le M \right\}$, V_0 and V_1 are empty sets, $V_{\infty} = \left\{1, 2, ..., N\right\}$, $S\left(j\right) = 0$ for all $1 \le j \le n$.

STEP 1 [**Group Column Indices**] Form a set $\mathcal{G}_{\infty}^{\rho}$ such that for each $\rho \in R_{\infty}$.

STEP 2 [**Find Candidate Rows**] Make a subset of R_{∞} (call it Ω) such that $\forall \omega \in \Omega$, $\left|\mathcal{G}_{\infty}^{\omega}\right| = \mathrm{rw}_{\mathrm{eff}}^{\min} \le \left|\mathcal{G}_{\infty}^{\rho}\right| = \mathrm{rw}_{\mathrm{eff}}\left(\rho, V_{\infty}\right)$ for any $\rho \in R_{\infty}$.

STEP 3 [**Group Row Indices**] Make a set $\mathcal{C}_{\infty}^{\gamma}$ such that $\mathcal{C}_{\infty}^{\gamma} = \Lambda^{\gamma} \cap R_{\infty}$, for all $\gamma \in \mathcal{G}_{\infty}^{\omega}$, and $\omega \in \Omega$.

STEP 4 [**Find the Best Rows**] Find a subset of Ω (call it Ω^{*}) such that $\forall \omega^{*} \in \Omega^{*}$, $\exists c \in \mathcal{G}_{\infty}^{\omega^{*}}$ such that $\left|\mathcal{C}_{\infty}^{c}\right| = \mathrm{cw}_{\mathrm{eff}}^{\min} \le \left|\mathcal{C}_{\infty}^{\gamma}\right| = \mathrm{cw}_{\mathrm{eff}}\left(\gamma, R_{\infty}\right)$ for any $\omega \in \Omega$ and $\gamma \in \mathcal{C}_{\infty}^{\omega}$.

STEP 5 [**Make a Set of Ordered Pairs**] Pick a column index $c^{*} \in \mathcal{G}_{\infty}^{\omega^{*}}$ with $\mathrm{cw}_{\mathrm{eff}}^{\min}$ randomly, when there is more than one column index with $\mathrm{cw}_{\mathrm{eff}}^{\min}$. Then, we will have an ordered pair $\mathcal{O} = \left\{ \left(\omega_1^{*}, c_1^{*}\right), \left(\omega_2^{*}, c_2^{*}\right), \ldots, \left(\omega_p^{*}, c_p^{*}\right) \right\}$, where ω_j^{*}'s and c_j^{*}'s are row and column indices with $\mathrm{cw}_{\mathrm{eff}}^{\min}$ and $\mathrm{rw}_{\mathrm{eff}}^{\min}$, respectively, and $1 \le j \le \left|\mathcal{O}\right| = p$.

STEP 6 [**Find the Best Pair**] Pick a pair $\left(\omega^{*}, c^{*}\right)$ from $\mathcal{O}$ such that $\mathcal{W}\left(\omega^{*}\right) \le \mathcal{W}\left(\omega_j^{*}\right)$, where $1 \le j \le p$, $\mathcal{W}\left(\omega_j^*\right) = \sum_{\gamma \in \Gamma^{\omega_j^*}} S(\gamma)$. If there is more than one pair satisfying the inequality, pick one randomly.

STEP 7 [**Update**] $V_k = V_k \cup \{c^*\}$, $V_0 = V_0 \cup \left(\mathcal{G}_\infty^{\omega^*} \setminus \{c^*\}\right)$, $V_\infty = V_\infty \setminus \mathcal{G}_\infty^{\varpi^*}$, $R_k = R_k \cup \{\omega^*\}$, $R_0 = R_0 \cup \left(\mathcal{C}_\infty^{\omega^*} \setminus \{\omega^*\}\right)$, and $R_\infty = R_\infty \setminus \mathcal{C}_\infty^{\varpi^*}$, $S(\gamma) = 1$ for $\gamma \in \mathcal{G}_\infty^{\omega^*} \setminus c^*$, $S\left(c^*\right) = \sum_{\gamma \in \left\{\Gamma^{\omega^*} \setminus c^*\right\}} S(\gamma)$.

STEP 8 [**Check Stop Condition**] If V_∞ is an empty set, then STOP.

STEP 9 [**Decision**] If R_∞ is not an empty set, then go to STEP 1.

STEP 10 [**No More Undetermined Rows**] $R_\infty = \left\{\rho : \rho \in R_0 \text{ and } \mathrm{rw}_{\mathrm{eff}}\left(\rho, V_\infty\right) > 0\right\}$.

STEP 11 $k = k + 1$, and go to STEP 1.

In STEP 0, Γ^ρ (Λ^γ) is a set of non-zero column (row) indices in the row ρ (column γ), and V_0, V_1, and V_∞ are sets of unpunctured, *1*-SR, and undetermined column indices, respectively. When there are no more undetermined columns, the algorithm will stop. R_k and R_∞ are sets of row indices that are surviving check nodes of *k*-SR nodes and undetermined check nodes, respectively. If a row contains a *k*-SR node, the *k*-SR node is also on the other $d_v - 1$ rows, where d_v is a degree of the *k*-SR node. The indices of the other $d_v - 1$ rows of all *k*-SR nodes are assigned to R_0, and the rows in R_0 are excluded from the candidate rows for a new surviving check node of a *k*-SR node. In STEP 1, $\mathcal{G}_\infty^\rho$ will have all the column indices both in Γ^ρ and V_∞. Thus, the cardinality of $\mathcal{G}_\infty^\rho$ is $\mathrm{rw}_{\mathrm{eff}}\left(\rho, V_\infty\right)$. In STEP 2, we look for rows in R_∞ with a minimum of $\mathrm{rw}_{\mathrm{eff}}\left(\rho, V_\infty\right)$, which is simply denoted as $\mathrm{rw}_{\mathrm{eff}}^{\min}$. Since the size of $|V_\infty|$ decreases by $|\mathcal{G}_\infty^\rho|$ in STEP 7, the rows with $\mathrm{rw}_{\mathrm{eff}}^{\min}$ will give us more *k*-SR nodes. In general, there may be more than one row with $\mathrm{rw}_{\mathrm{eff}}^{\min}$. The set Ω contains the row indices with $\mathrm{rw}_{\mathrm{eff}}^{\min}$. In STEP 3, $\mathcal{C}_\infty^\gamma$ will have row indices belonging to both Λ^γ and R_∞. Similar to $\mathcal{G}_\infty^\omega$, the cardinality of $\mathcal{C}_\infty^\gamma$ is $\mathrm{cw}_{\mathrm{eff}}(\gamma, R_\infty)$. We look for rows in which there is at least one column with a minimum of $\mathrm{cw}_{\mathrm{eff}}(\gamma, R_\infty)$, which is simply denoted as $\mathrm{cw}_{\mathrm{eff}}^{\min}$. Again, we will have more *k*-SR nodes with $\mathrm{cw}_{\mathrm{eff}}^{\min}$ since in STEP 7, $|R_\infty|$ decreases by $|C_\infty^{\omega^*}|$. In STEP 6, we make a set $\mathcal{O}$ of ordered pairs, each of which has a row and a column of the row with $\mathrm{rw}_{\mathrm{eff}}^{\min}$ and $\mathrm{cw}_{\mathrm{eff}}^{\min}$, respectively. The set of ordered pairs is not unique since a row may have several columns with $\mathrm{cw}_{\mathrm{eff}}^{\min}$. In this case, we randomly choose a column from them. In terms of maximizing V_k, each ordered pair gives us statistically the same result. Among the ordered pairs, we will choose the one with the highest probability (smallest recovery error) to be recovered in the *k*-th iteration.

As in Theorems 3.1-3, we pick up a pair with the smallest $S(c)$, which is equivalently evaluated with a measure $\mathcal{W}$ for computational efficiency. In STEP 7, we update the sets with the pair chosen in STEP 6. In STEP 9, the cardinality of R_∞ is checked. If it is not zero, STEP 1 will be visited again. Otherwise, R_∞ is updated in STEP 10, where R_∞ takes rows ρ's with non-zero $\mathrm{rw}_{\mathrm{eff}}^{\min}(\rho, G_\infty)$ in R_0. In STEP 11, we increase k by 1 and start looking (*k+1*)-SR nodes.

In the parity-check matrix, the puncturing algorithm assigns each column to a group among $V_0, V_1, \ldots, V_k$. If we permute rows and columns in the parity-check matrix such that columns in the same group are gathered and that elements corresponding to k-SR nodes are formed diagonal, then the parity-check matrix can be reconstructed as shown in Figure 3.3.

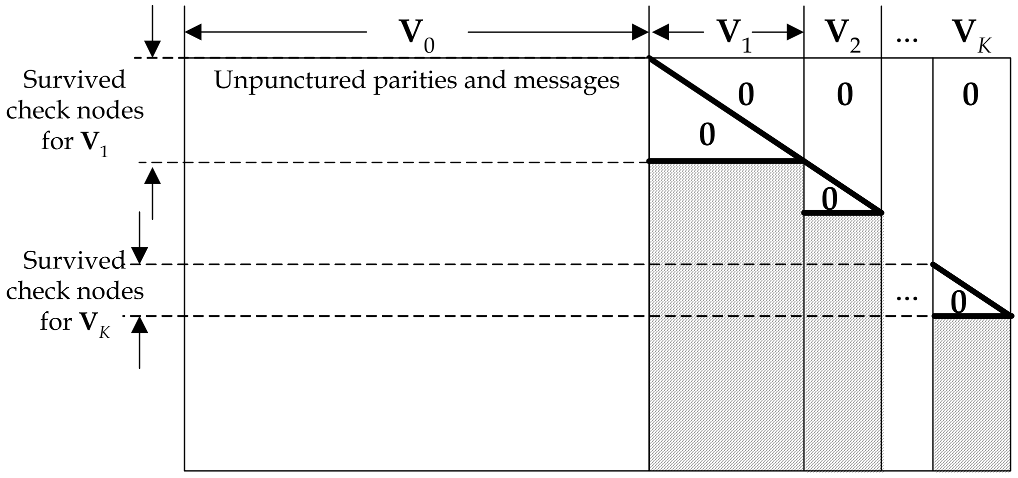

$\mathbf{V}_0$
$\mathbf{V}_1$
$\mathbf{V}_2$
...
$\mathbf{V}_K$
Survived check nodes for $\mathbf{V}_1$
Unpunctured parities and messages
0
0
0
0
0
Survived check nodes for $\mathbf{V}_K$
...
0

Figure 3.2 Logical structure of a parity-check matrix.

By puncturing symbols in V_k, the maximum achievable code rate $r_{\max}$ can be expressed as

$$r_{\max} = \frac{r_0}{1 - \sum_{j=1}^{K} |V_j| / N},$$

where r_0 is the mother code rate, N is the block length of the mother code. For a sequence of designed rates, $r_0 \le r_1 \le \ldots \le r_M \le r_{\max}$, we can compute the required number of punctured nodes Np_j as

$$Np_j = \left\lfloor \frac{N\left(r_j - r_0\right)}{r} \right\rfloor,$$

for $0 \le j \le M$. We will take Np_j nodes from V_k's in terms of minimizing performance loss resulting from the puncturing. Now, we discuss the sorting step, where we determine the order of puncturing within each group $V_0, V_1, \dots, V_k$.

Proposed Sorting Algorithm

STEP 0 [**Initialization**] For a given $M \times N$ parity-check matrix, $j = 1$, $k = 1$, Λ^c is a set of non-zero row indices in the column c, P_0 is an empty sets, and $\mathbf{R} = \{1, 2, ..., M\}$.

STEP 1 If $j > M$, STOP.

STEP 2 $\mathbf{P}_j = \mathbf{P}_{j-1}$.

STEP 3 $\delta Np_j = Np_j - \left|\mathbf{P}_j\right|$.

STEP 4 If δNp_j is zero, $j = j+1$ and go to STEP 1.

STEP 5 Make a set of column indices $\{c_1, c_2, ..., c_p\}$, for $1 \le p \le |V_k|$ from V_k such that $\forall c_j \in \{c_1, c_2, ..., c_p\}$, $\mathrm{cw}_{\mathrm{eff}}\left(c_j, \mathbf{R}\right) = \mathrm{cw}_{\mathrm{eff}}^{\max} \ge \mathrm{cw}_{\mathrm{eff}}\left(c, \mathbf{R}\right)$, for any $c \in V_k$.

STEP 6 If $p > 1$, we take nodes c_j^* such that $\forall c \in \{c_1, c_2, ..., c_p\}$, $\deg\left(c_j^*\right) \le \deg(c)$. If there are multiple such nodes, we pick one from them arbitrary and call it c^*.

STEP 7 $\mathbf{P}_j = \mathbf{P}_j \bigcup \{c^*\}$, $V_k = V_k \setminus \{c^*\}$, and $\mathbf{R} = \mathbf{R} \setminus \Lambda^{c*}$, $\delta Np_j = \delta Np_j - 1$.

STEP 8 If V_k is an empty set, $k = k+1$, and $\mathbf{R} = \{1, 2, ..., M\}$.

STEP 9 Go to STEP 4.

In the proposed sorting algorithm, $\mathbf{P}_j$ will have column indices that are punctured to achieve rate r_j. Obviously, for r_0, $\mathbf{P}_0$ is an empty set in the initialization. In the algorithm, it is assumed that we will design rates from r_0 to r_M, which is less than or equal to $r_{\max}$. Thus, in STEP 1, if j is larger than M, the algorithm will stop. In STEP 2, $\mathbf{P}_j$ takes the column indices in $\mathbf{P}_{j-1}$, which makes the punctured LDPC codes rate-compatible since all the punctured nodes for r_{j-1} will be punctured again for r_j. In STEP 3, δNp_j accounts for how many additional nodes are needed to make $\mathbf{P}_j$ besides the ones in $\mathbf{P}_{j-1}$. The loop between STEP 4 and STEP 9 continues until δNp_j becomes zero. In STEP 5, we look for nodes with the largest number of surviving check nodes, which is equivalent to nodes with $\mathrm{cw}_{\mathrm{eff}}^{\max}$.

We compute the additional number of punctured nodes to the ones in the previous rate r_{j-1}. If we need additional nodes, the algorithm looks for nodes with a maximum effective column weight $\mathrm{cw}_{\mathrm{eff}}^{\max}$, which means the node with the maximum surviving check nodes. We exclude rows with *k*-SR nodes from $\mathbf{R}$ in STEP 7. Thus, $\mathrm{cw}_{\mathrm{eff}}^{\max}$ counts only surviving check nodes of a variable node. In STEP 6, we choose nodes with the smallest column degree from $\{c_1, c_2, ..., c_p\}$. This selection will give us nodes with the smallest number of dead check nodes in $\{c_1, c_2, ..., c_p\}$, which is d_c - $\mathrm{cw}_{\mathrm{eff}}^{\max}$ for the smallest column degree of d_c. By puncturing a node with dead check nodes, the punctured node not only has a reliable message from the dead check nodes but also makes dead check nodes to the *k*-SR nodes connect to the dead check nodes. Thus, we look for nodes with the largest number of surviving check nodes and the smallest number of dead check nodes.

3.3 SIMULATION RESULTS

The proposed algorithms are based on the claim that a puncturing distribution that will be recovered within the smallest number of iterations guarantees better performance. We verify the claim with computer simulations, where we show that punctured LDPC codes with the proposed algorithms have better performances than ones with the conventional random puncturing in terms of bit error rate (BER) and word error rate (WER). The BER and WER performances are measured after observing at least 50 erroneous code words at each E_b/N_o value to guarantee statistical confidence. The punctured LDPC codes are also compared with dedicated LDPC codes that are designed at the rates of the punctured LDPC codes.

First, we implement half rate mother LDPC codes with a regular structure ($\lambda(x) = x^2$ and $\rho(x) = x^5$) at block lengths of 1024 and 4096. We deliberately avoid cycle-4 loops in parity-check matrices of the mother codes to get the better minimum distance property [16]. By puncturing the mother codes, we implement punctured LDPC codes at rate 0.5, 0.6, 0.7, and 0.8. The block lengths of the punctured LDPC codes and the number of punctured parity bits for the rates are listed in Table 3.1. Note that the block lengths of the punctured LDPC codes are shorter at higher rates. Although punctured LDPC codes have shorter block lengths, a received LDPC code is decoded on the Tanner graph of its mother code. For fair comparisons, dedicated LDPC codes are designed at the block lengths of the corresponding punctured LDPC codes for the rates.

Table 3.1 Block lengths of punctured LDPC codes; The lengths in parentheses are the number of punctured symbols at the rates.

Block lengths	Code rates			
	0.5	0.6	0.7	0.8
1024	1024 (0)	853 (171)	731 (293)	640 (384)
4096	4096 (0)	3413 (683)	2926 (1170)	2560 (1536)

For (3,6) regular LDPC codes with block length 1024, the puncturing group distributions with the proposed grouping algorithm (denoted as *Intentional*) and the three different trials (denoted as *Random 1*, *2*, and *3*) of random selections are in Table 3.2. The distributions with the random puncturing are implemented three times with different random seeds to see the performance variations with the different levels of recoverability. In Table 3.2, the distribution with the intentional puncturing requires 3 iterations to recover all the punctured parity bits, meaning that the level of recoverability is 3. However, the distributions with the random puncturing require 9 iterations, which results in higher recovery-error probabilities of the symbols in the groups with higher indices.

Table 3.2 Group distributions of the intentional puncturing and the random puncturing of a regular LDPC code with $\lambda(x) = x^2$ and $\rho(x) = x^5$ at a block length of 1024; The largest code rate is 0.8.

	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}	V_{14}
Intentional	640	294	78	12	0	0	0	0	0	0	0	0	0	0	0
Random 1	640	108	60	44	45	37	42	33	11	4	0	0	0	0	0
Random 2	640	100	50	38	36	26	28	27	30	28	19	2	0	0	0
Random 3	640	100	46	32	26	18	19	17	20	23	25	25	26	15	3

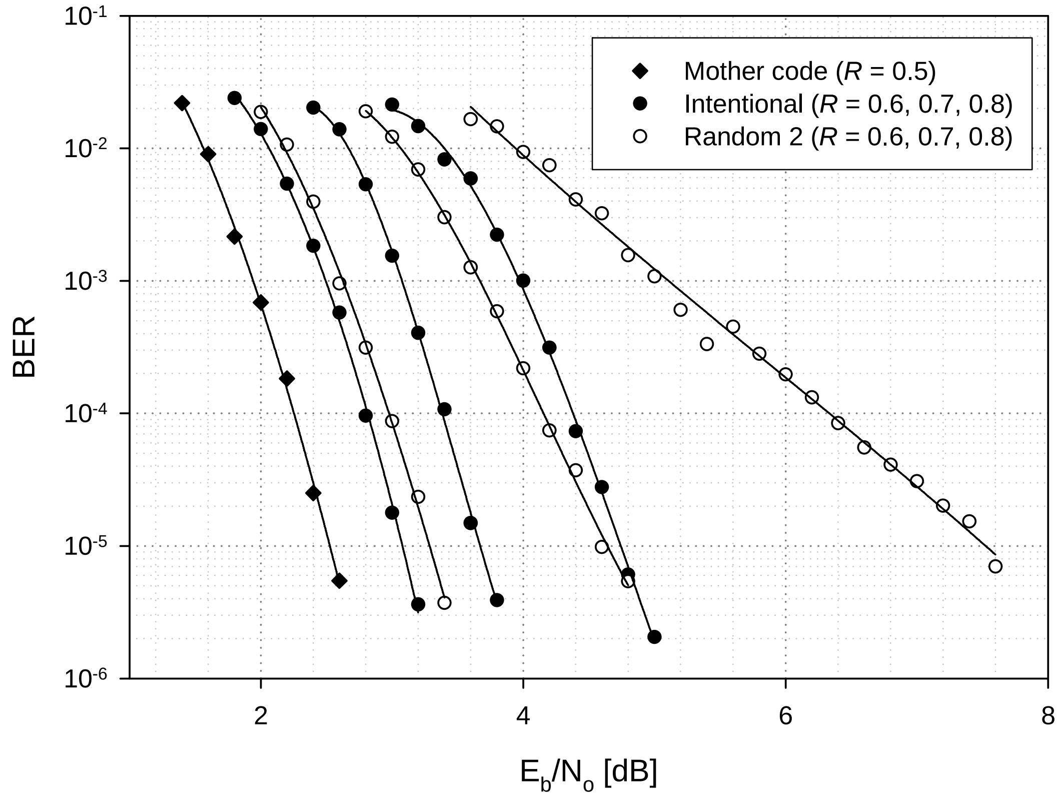


Figure 3.3 Comparison between the intentional (filled) and random (unfilled) puncturing of a regular LDPC code at block length 1024; code rates are 0.5, 0.6, 0.7 and 0.8 from the left to the right, the puncturing distributions are from Intentional and Random in Table 3.2 and the BERs of the half rate mother are represented with the diamonds.

The intentionally punctured LDPC codes are compared with the randomly punctured ones in Figure 3.2. In the comparison, the intentional puncturing outperforms at all the rates. The performance improvement with the proposed algorithm becomes more distinctive at higher rates. At rate 0.8, to achieve a BER of 10^{-5}, the intentionally punctured LDPC code has 3dB better E_b/N_o performance than that of the randomly punctured one.

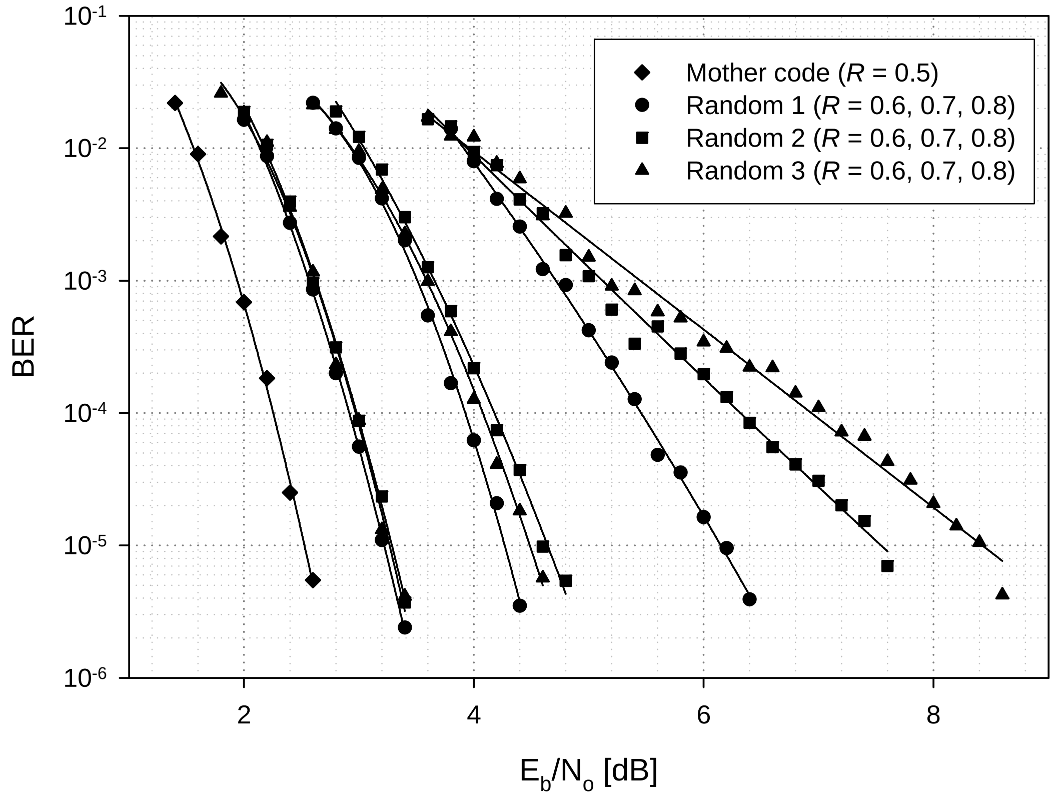


Figure 3.4 Randomly punctured LDPC codes with three different random seeds; the circles, squares and triangles correspond to BERs of Random 1, 2, and 3 in Table 3.2, respectively, and the BERs of the half rate mother code at block length 1024 are represented with the diamonds.

In Figure 3.4, we compared BER performance of the randomly punctured LDPC codes with three different random seeds at block length 1024, where the ones with the smallest and largest level of recoverability (denoted as Ramdom 1 and 3 in Table 3.2, respectively) show the best and the worst BER performances at rate 0.8, respectively. Thus, the simulation results justify our design rule which looks for selections of punctured parity bits with a smaller level of recoverability for the highest rate, 0.8. However the performances at the intermediate rates (0.6 and 0.7) in Figure 3.4 do not seem to depend on the levels of recoverability since the punctured LDPC code with the puncturing distribution Random 3 has better performance than the one with Random 2.

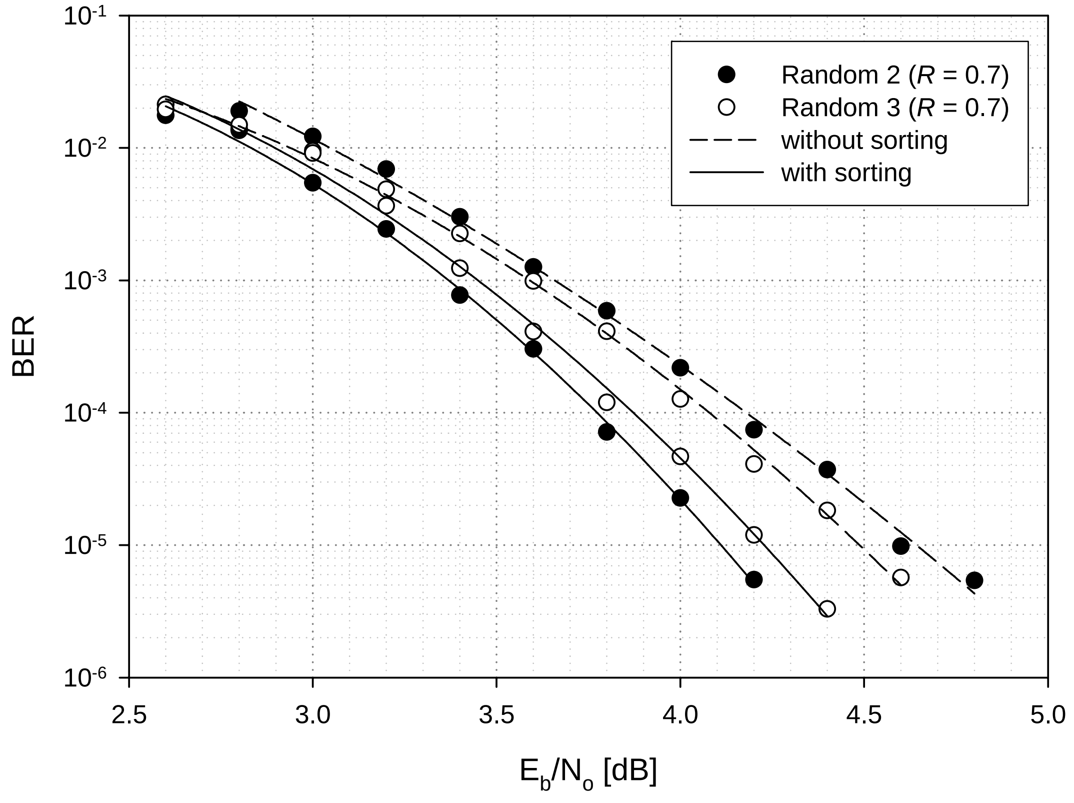


Figure 3.5 Randomly punctured LDPC codes at rate 0.7 with (solid lines)/without (dashed lines) the sorting algorithm; the filled and unfilled circles are performances of the punctured LDPC codes with the puncturing distributions Random 2 and 3, respectively and the mother code has a block length of 1024.

The distributions in Table 3.2 describe levels of recoverability of the punctured LDPC codes at the highest rate. Thus, better group distribution of a punctured LDPC code does not guarantee better performance at intermediate rates if we do not carefully choose the order of puncturing. For intentional puncturing, we apply the sorting algorithm to determine the order of puncturing but in the case of random puncturing, we puncture parities with smaller node indices first. The performances of the randomly punctured LDPC codes at the intermediate rates can also be improved by applying the sorting to the random puncturing.

In Figure 3.5, we compare the performances of the punctured LDPC codes with Random 2 and Random 3 at rate 0.7 with/without the sorting algorithm. The sorting algorithm improves the E_b/N_o performances of the punctured LDPC codes at a BER of 10^{-5} by 0.5dB for Random 2 and 0.3dB for Random 3. After applying the sorting algorithm, the punctured LDPC code with Random 2 has better performance than that of Random 3, which means the punctured LDPC with the smaller level of recoverability at the highest rate has better performance.

One more thing to be noticed in Figure 3.4 is the BER performance variations with the different seeds. The required E_b/N_o to achieve a BER of 10^{-5} at rate 0.8 has a difference of 2.2dB between the best (Random 1) and the worst (Random 3) cases. The random puncturing may delete significant amount of parity bits in a stopping set, which results in severe performance degradation especially at higher rates. It is hard to know whether a random puncturing results in catastrophic selections of punctured parity bits without time-consuming computer simulations. However, by analyzing the level of recoverability under the framework of the proposed grouping algorithm, we can predict the performance of randomly punctured LDPC codes and rule out the catastrophic selections. Thus, the proposed idea is also useful for designing randomly punctured LDPC codes.

To compare performances of a dedicated and the punctured LDPC codes, we design a regular LDPC code ($\lambda(x) = x^2$ and $\rho(x) = x^9$) for rate 0.7 at block length 731. The block length is chosen for the block lengths of the punctured and dedicated LDPC codes to be the same as shown in Table 3.1. Since dedicated LDPC codes can have smaller minimum distances due to shorter block lengths, it is possible that dedicated LDPC codes show poorer performances at high E_b/N_o regions.

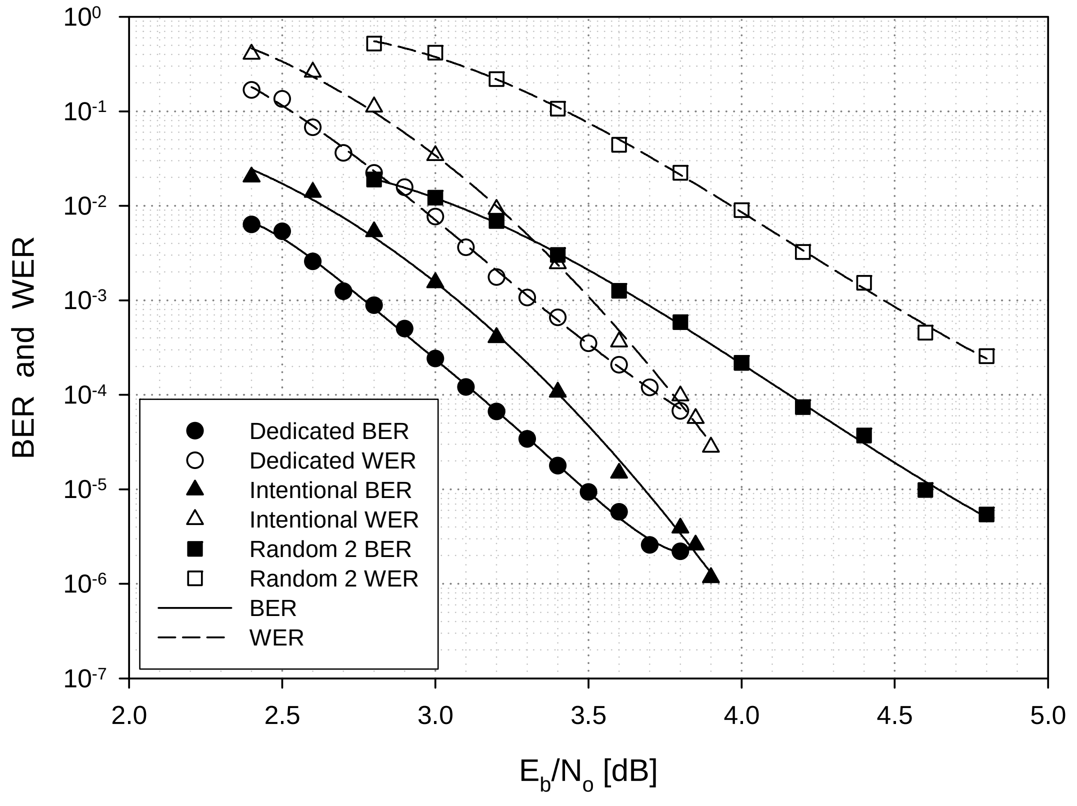


Figure 3.6 BERs (filled) and WERs (unfilled) of a dedicated LDPC code, a proposed punctured LDPC code and a randomly punctured LDPC code with rate 0.7 from the left to the right, respectively; the block length of the base LDPC code is 1024.

In Figure 3.6, we compare the BER and WER performances of the dedicated, randomly punctured and intentionally punctured LDPC codes with code rate 0.7 and block length 731. In the comparisons, the BER/WER performances are in the order of the dedicated, intentionally punctured, and the randomly punctured LDPC codes from the best to the worst. However, at high E_b/N_o regions, there are crossover points in BER and WER curves of the dedicated and intentionally punctured LDPC codes because of the smaller block length of the dedicated LDPC code. Thus, depending on a required BER/WER performance, intentionally punctured LDPC codes are more favorable than dedicated LDPC codes besides the structural advantage of the rate compatibility and the lower complexities of encoders and decoders.

The performance variation of random puncturing becomes smaller as the block length increases. To see the performance variation with increasing block lengths, we do the same simulations at a block length of 4096. The group distributions of the intentional and random puncturing are listed in Table 3.3, where the levels of recoverability are 3 and at least 10 for intentional and random puncturing, respectively.

Table 3.3 Group distributions of the intentional puncturing and the random puncturing of a regular LDPC code with $\lambda(x)=x^2$ and $\rho(x)=x^5$ at a block length of 4096; The largest code rate is 0.8.

	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}	V_{13}
Intentional	2560	1155	323	58	0	0	0	0	0	0	0	0	0	0
Random 1	2560	436	236	188	168	140	133	118	82	29	6	0	0	0
Random 2	2560	398	227	178	128	111	102	99	103	84	69	35	2	0
Random 3	2560	363	207	146	124	110	102	104	115	113	85	52	13	2

We evaluate the BER performances of the three randomly punctured LDPC codes in

Figure 3.7, where the performance variations among the different random puncturing distributions are noticeably smaller. At rate 0.8, the required E_b/N_o for a BER of 10^{-5} has a variation of less than 0.9dB as compared to 2.2dB in the case of block length 1024. The smaller performance variation can also be predicted by the group distribution in Table 3.3. Thus, locations of punctured parity bits are more important at shorter block lengths.

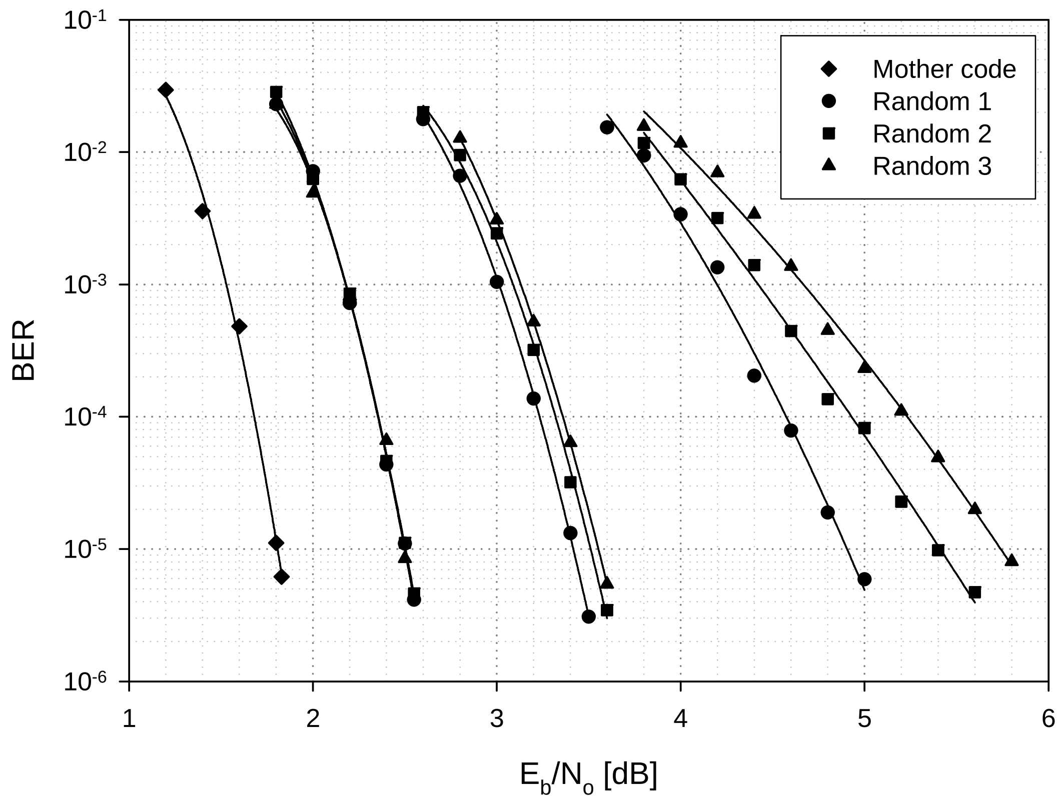


Figure 3.7 Randomly punctured LDPC codes with three different random seeds; the circles, squares and triangles correspond to BERs of Random 1, 2, and 3 in Table 3.3, respectively, and the BERs of the half rate mother code at block length 4096 are represented with the diamonds.

To see the level of recoverability with increasing block lengths, we design 10,000 different random puncturing distributions for the regular ($\lambda(x)=x^2$ and $\rho(x)=x^5$) LDPC codes at the block lengths of 1024, 4096, 65536. The levels of recoverability are observed by analyzing the group distributions of the puncturing distributions. Histograms of the levels of recoverability with the three different block lengths are compared in Figure 3.8, where most of time, the level of recoverability is bigger than 10. However, the variations of the level of recoverability become smaller at the longer block lengths. In the extreme case with the block length of 65536, the variation of the level of recoverability is significantly smaller, where 11 and 12 account for over 99% of the occurrence. Thus, we confirm that careful selections of punctured bits are more important at smaller block lengths from the difference perspective.

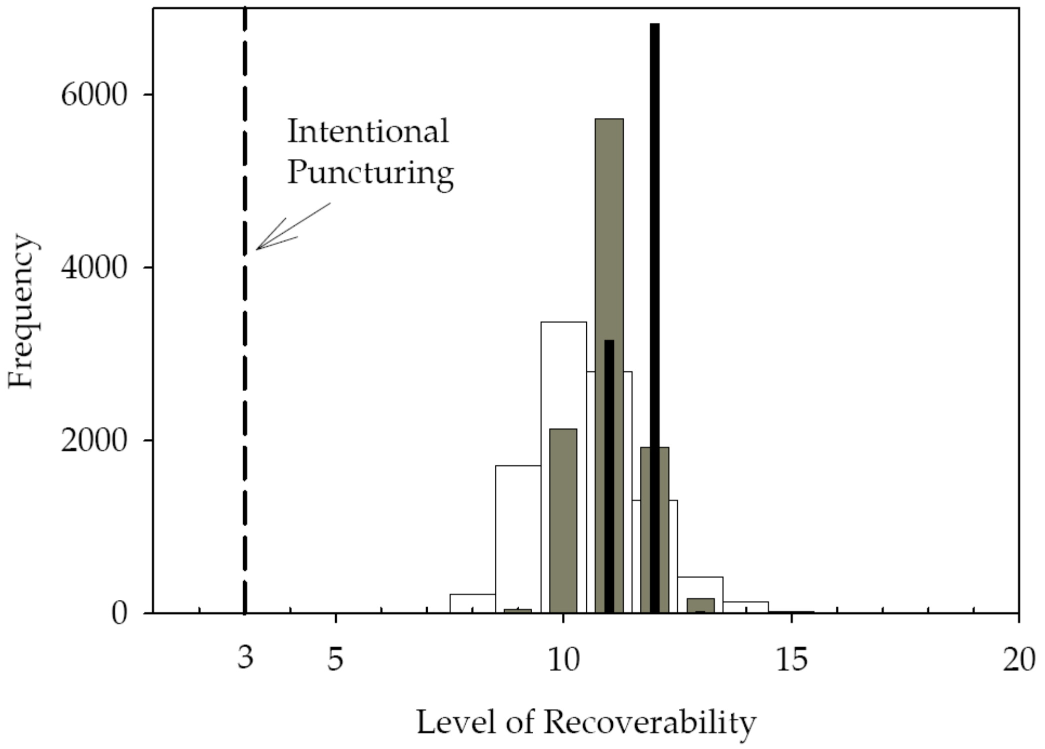


Figure 3.8 Histograms of the levels of recoverability; results from 10,000 trials with the regular LDPC codes at the block lengths 1024 (unfilled), 4096 (shaded), and 65536 (filled).

The randomly puncture LDPC codes (Random 2 in Table 3.3) are compared with the intentionally punctured ones (Intentional in Table 3.3) in Figure 3.9, where at rate 0.8, to achieve a BER of 10^{-5}, the intentionally punctured LDPC code requires 1.6dB smaller E_b/N_o value than that of the randomly punctured one.

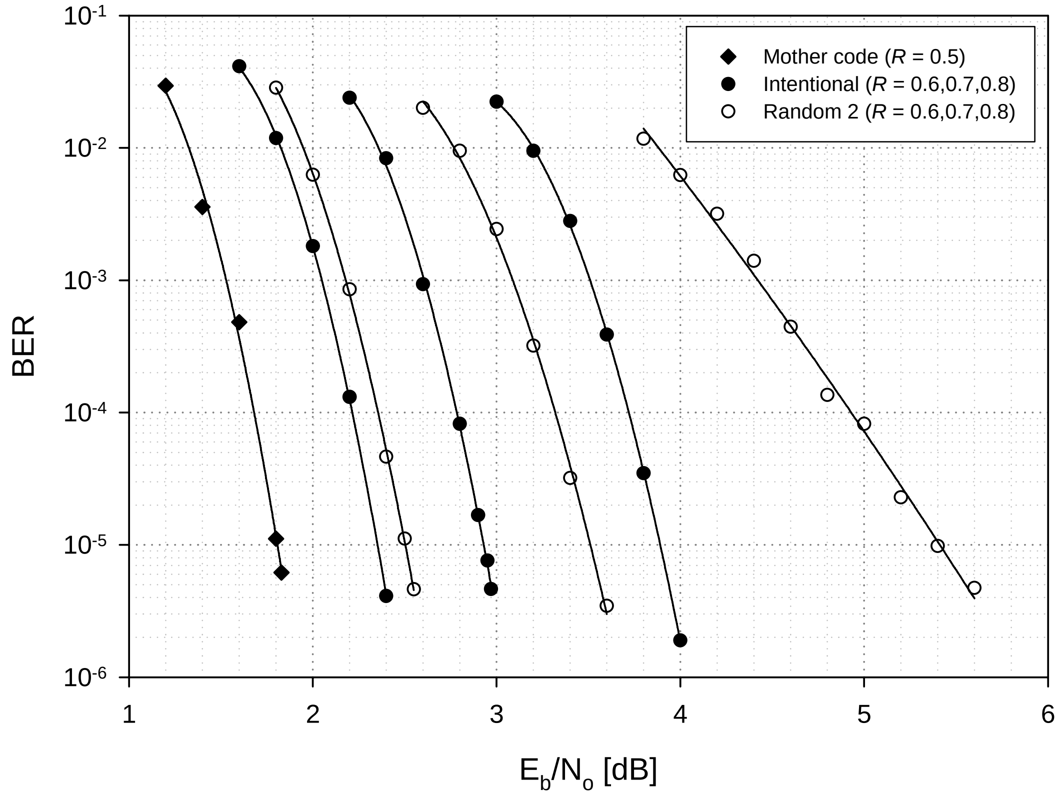


Figure 3.9 Comparison between the intentional (filled) and random (unfilled) puncturing of a regular LDPC code at block length 4096; code rates are 0.5, 0.6, 0.7 and 0.8 from the left to the right, and the puncturing distributions are from Intentional and Random 2 in Table 3.3.

The proposed algorithms can be applied to irregular LDPC codes as well as regular LDPC codes. To demonstrate the performance of punctured LDPC, we design a half rate irregular LDPC code as a mother code whose degree distribution pair is

$$\lambda(x) = 0.28286x + 0.39943x^2 + 0.31771x^7$$
$$\rho(x) = 0.6x^5 + 0.4x^6.$$

A parity-check matrix for the irregular LDPC codes at block lengths of 1024 is designed with the Progressive Edge Growth (PEG) algorithm [16] to get a better girth distribution. The mother code is punctured either randomly or intentionally based the proposed algorithms. The group distributions are listed in Table 3.4, where Random (Intentional) 1024 and 4096 indicates the group distributions of random (intentional) puncturing at block lengths of 1024 and 4096, respectively.

Table 3.4 Group distributions of the intentional puncturing and the random puncturing of an irregular LDPC code with $\lambda(x) = 0.28286x + 0.39943x^2 + 0.31771x^7$ and $\rho(x) = 0.6x^5 + 0.4x^6$ at a block length of 1024 and 4096; The largest code rate is 0.8.

	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}
Intentional 1024	640	333	46	5	0	0	0	0	0	0	0	0
Random 1024	640	77	63	41	41	48	38	26	23	15	10	2
Intentional 4096	2560	1311	206	19	0	0	0	0	0	0	0	0
Random 4096	2560	361	261	214	182	163	132	112	63	37	8	3

For fair comparisons, we simulate random puncturing with three different random seeds, and then pick one that has middle performance among them, as we did in the regular case. The performances of randomly and intentionally punctured LDPC codes at block lengths 1024 and 4096 are evaluated and compared in Figure 3.10 and Figure 3.11, respectively.

Again, the intentionally punctured LDPC codes outperform the randomly punctured LDPC codes at all rates. At rate 0.8, for a BER of 10^{-5}, the intentionally punctured LDPC codes at block lengths 1024 and 4096 have 1.25dB and 0.8dB of E_b/N_o improvements over those of the randomly punctured ones, respectively.

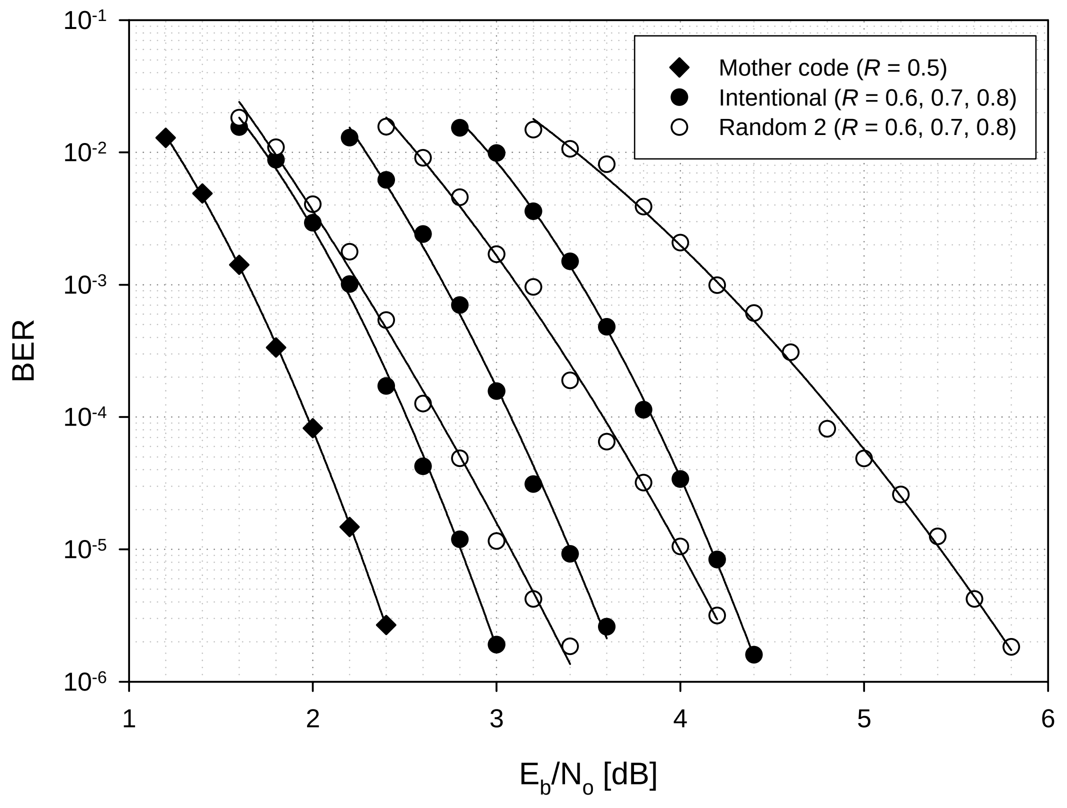


Figure 3.10 Comparison between the proposed puncturing (filled dots) and random puncturing (unfilled dots); the half rate irregular mother code (leftmost) has a block length of 1024, and the punctured LDPC codes have rates of 0.6, 0.7, and 0.8 from the left to the right.

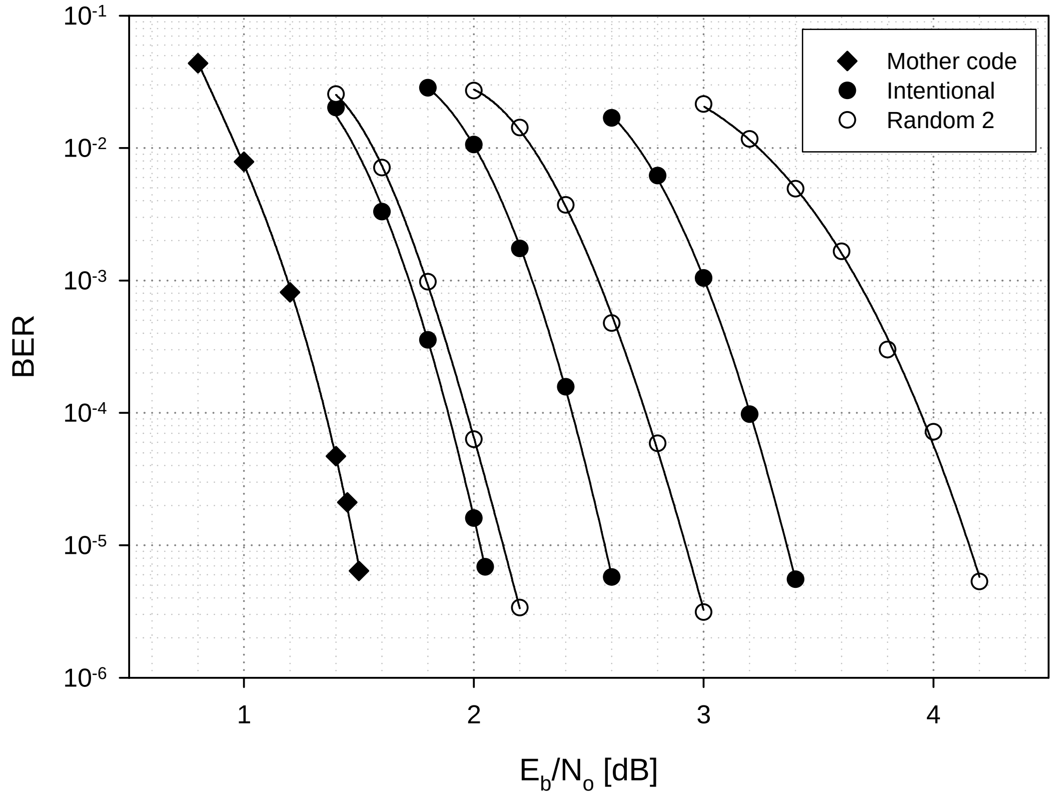


Figure 3.11 Comparison between the proposed puncturing (filled dots) and random puncturing (unfilled dots); the half rate irregular mother block length of 4096, and the punctured LDPC codes have rates of 0.6, 0.7, and 0.8 from the left to the right.

3.4 CONCLUSION

We propose the grouping and sorting algorithms to design rate-compatible punctured LDPC codes at short block lengths. The algorithms are based on the claim that a punctured LDPC code with a smaller level of recoverability has better performance. We mathematically explain why the proposed algorithms provide us with better punctured LDPC codes by introducing the concepts of recovery tree and recovery error probability.

The proposed algorithms are verified by comparing performance of punctured LDPC codes based on the algorithm (called intentionally punctured LDPC codes) with randomly punctured LDPC codes. The intentionally punctured LDPC codes show better BER performances at relatively small block lengths (1024 and 4096), and the performance improvement is more distinctive at smaller block lengths. In our simulations, in the case of the regular code with block length 1024, the intentionally punctured LDPC has 3dB better E_b/N_o performance than that of the randomly punctured one for a BER of 10^{-5} at code rate 0.8. For the longer block length 4096, the intentionally punctured LDPC code outperforms the randomly punctured LDPC code by 1.6dB at rate 0.8 for a BER of 10^{-5}. That is, the improvement becomes smaller but still significant.

More important observation is the performance variation of randomly punctured LDPC codes. Especially, at small block lengths, the variation becomes unacceptable. In our simulations, we observed 2.2dB performance difference between the best and the worst randomly punctured LDPC codes. It is possible to puncture significant amount of parities in a stopping set, which results in poor performance. In the conventional design rule of randomly punctured LDPC codes, the performance variation can be evaluated with time-consuming computer simulations. However, by analyzing group distributions of random puncturing distributions, we predict their BER performances in a much faster way.

The performance variations become smaller at larger block lengths, which are verified by evaluating histograms of levels of recoverability at three different block lengths, 1024, 4096, and 65536. In the case that we have to use random puncturing, the analysis under the framework of the grouping algorithm gives us a good random puncturing distribution.

We also show that the sorting algorithm can be applied for random puncturing. A random puncturing distribution tells the locations of parities to be punctured but the distribution does not say an order to puncture the parities. Although performance at the highest code rate is determined by the level of recoverability, performances of punctured LDPC codes with intermediate code rates from that of the mother code to the highest code rate depend on the order to puncture the parities.

Finally, we apply the proposed algorithm to irregular LDPC codes at block lengths 1024 and 4096. The performance improvements of the intentionally punctured LDPC codes are 1.25dB for block length 1024 and 0.8dB for block length 4096 over randomly punctured LDPC codes at code rate 0.8 for a BER of 10^{-5}.

CHAPTER IV

EFFICIENTLY-ENCODABLE RATE-COMPATIBLE CODES

The proposed puncturing algorithm in the previous section (from now on, we call this intentional puncturing) works for any given mother code. However, the maximum puncturing rate is often limited when this algorithm is applied, so that high puncturing rates are difficult to achieve. From here, we are interested in the problem of mother code design for high puncturing capacity and good puncturing performance. In other words, we focus on a technique for code design in which the parity-check matrix of a mother code has a large number of variable nodes that are k-step recoverable with low values of k.

The eIRA codes of Yang *et al*. achieve good performance by assigning degree-2 nodes to nonsystematic bits and ensuring that the degree-2 nodes do not form a cycle among themselves. Furthermore, they avoid cycles of length-4 and make the systematic bits correspond to variable nodes of degree higher than two. They ensure efficient encoding by forming the parity in the bi-diagonal structure illustrated in Figure 2.6.

It is interesting to see whether there exist other ways of placing the degree-2 nodes so that the above conditions are satisfied. We present below an example of such a placement in Figure 4.1. Observe that the column degree of each node is 2 and that there does not exist any cycle in this matrix. We shall see later that this construction can be generalized and the resulting matrices can be used to construct LDPC codes that can be efficiently encoded and have good puncturing performance across a wide range of rates.

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

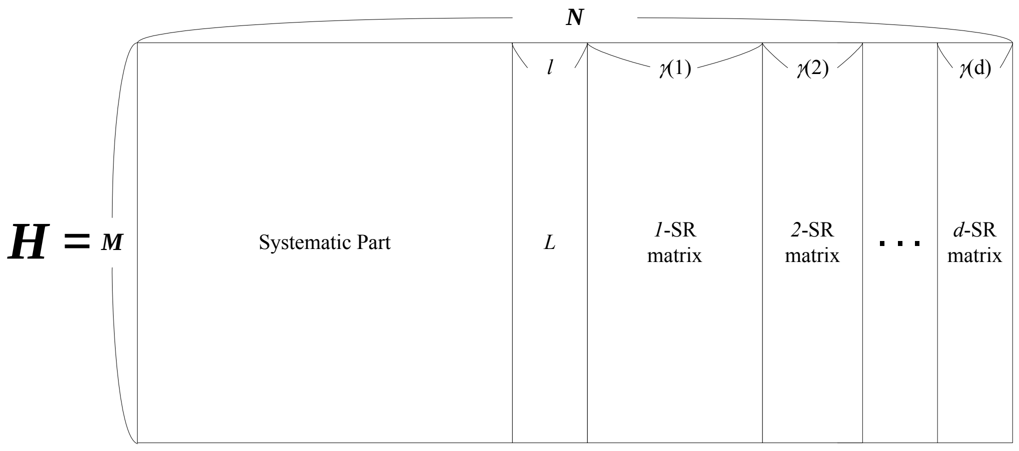
Figure 4.1 Another cycle-free structure with weight-2 nodes.

4.1 NEW CLASS OF IRREGULAR LDPC CODES

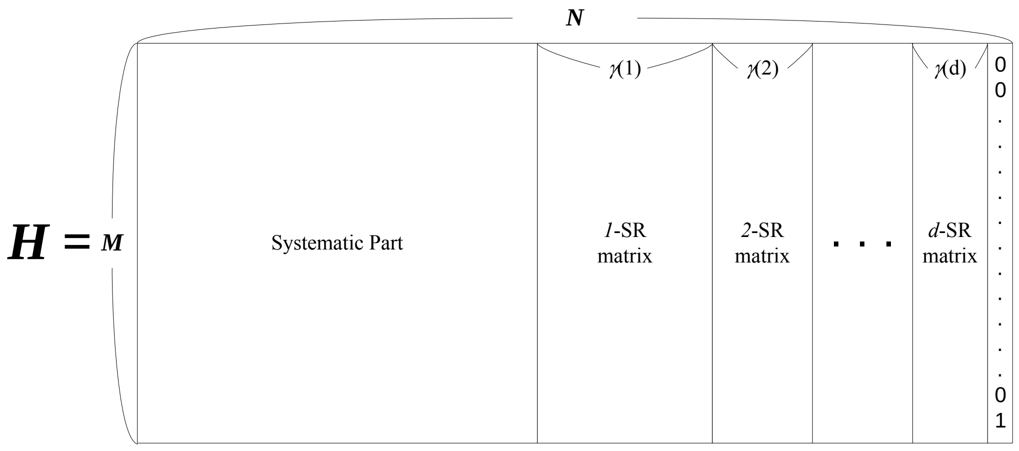
In this work we are interested in designing rate-compatible punctured codes that exhibit good performance across a wide range of coding rates. To ensure good performance over the different coding rates we attempt to design the mother code matrix to have a large number of *k*-SR nodes with low values of *k*. From a practical perspective the requirement of low-complexity encoding is also important. Like punctured RA, IRA and eIRA codes, these codes are designed to recover all the punctured bits when the channel is error-free even when they achieve the maximum puncturing rate by running sufficient iterations of iterative decoding. Before describing our design algorithm, we define a *k*-SR matrix.

Let h_i denote the columns of the parity-check matrix H, where $0 \le i < N$. Let $T(i)$ denote the variable node corresponding to the *i*-th column in the Tanner graph of H.

Definition 4.1: [*k-SR matrix*] The matrix $P = (h_s)_{s \in S}$ is called a *k*-SR matrix, if $T(s) \in V_k$ for all $s \in S$, where $S \subseteq \{0,1,\cdots,N-1\}$.



(a) $N_v(2) < M - 1$



(b) $N_v(2) = M - 1$

Figure 4.2 Construction of the parity-check matrix of the proposed codes.

In the proposed E^{2}RC codes, we construct the parity-check matrix laying several *k*-SR matrices as shown in Figure 4.2. We assign all the degree-2 nodes to the nonsystematic part, and nodes having degree higher than two are elements of *0*-SR matrix. Consider the submatrix of *0*-SR matrix formed by the high degree nodes in the nonsystematic part. We denote such submatrix of *0*-SR matrix as L, and the number of columns in L as l as depicted in Figure 4.2(a). Except for the *0*-SR matrix, we define the number of *k*-SR matrices in the parity-check matrix H and the column size of each *k*-SR matrix as following definitions. In our construction the non-systematic part of the mother code parity-check matrix consists of *k*-SR matrices that can be punctured efficiently.

Definition 4.2: The depth *d* is the number of *k*-SR matrices except the *0*-SR matrix in a parity-check matrix.

Definition 4.3: The function $\gamma(k)$ is the number of columns in the *k*-SR matrix in a parity-check matrix, $\gamma(k) = |V_k|$, where $k > 0$.

From Definition 4.3, note that the size of the *k*-SR matrix is $M \times \gamma(k)$. As defined in chapter II, $N_v(i)$ represent the number of nodes of degree *i*. Figure 4.2 (a) shows the case when $N_v(2) < M-1$, and we will explain the design of such case at the latter part of this section. Other than that, we assume that $N_v(2) = M-1$ throughout the thesis. One can consider to design when $N_v(2) > M-1$, but it is hard to find a good degree distributions with huge portion of degree-2 nodes. Furthermore, we cannot guarantee cycle-free among the degree-2 nodes, which is an important design rule that will be explained later. When $N_v(2) = M-1$, there will be no *0*-SR nodes in the nonsystematic part, i.e., $l = 0$. In this case, we insert a degree-1 node in the last column of nonsystematic part, and assign all the variable nodes of nonsystematic part to degree-2 nodes except the last degree-1 node as shown in Figure 4.2(b).

Example 4.1: For $M = 8$ and $N_v(2) = 7$, we can construct the nonsystematic part H_2 as

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

In the matrix H_2, the first four columns form the *1*-SR matrix, the next two columns form the *2*-SR matrix, and the next one column forms the *3*-SR matrix. Thus, depth $d = 3$, $\gamma(1) = 4$, $\gamma(2) = 2$, and $\gamma(3) = 1$. We can also regard the last degree-1 column as *4*-SR matrix. However, our convention in this thesis is to only consider degree-2 columns to calculate the depth *d*. From now on, we refer to the last degree-1 column in H_2 as (*d+1*)-SR matrix since the connections with other *k*-SR matrices makes it (*d+1*)-SR node.

■

We shall represent the position of the ones in a column belonging to a *k*-SR matrix by the powers of a polynomial in *D*. Let $S_k = \sum_{j=1}^{k} \gamma(j)$. Thus, S_k represents the sum of the size of the submatrix formed by the *1*-SR, *2*-SR,… and *k*-SR matrices. The *j*-th column of *k*-SR matrix has the following sequence:

$$h_{k,j} = D^{j+S_{k-1}}\left(1+D^{\gamma(k)}\right), \quad \text{where} \quad 1 \le k \le d, 0 \le j \le \gamma(k)-1$$

$$h_{d+1} = D^{M-1}.$$

In the sequence, D^i represents the position of nonzero element in a column, i.e., i-th element of the column is nonzero, where $0 \le i \le M-1$. For Example 4.1, we can notice that the depth can be obtained by $d = \log_2 M = \log_2 8 = 3$ and $\gamma(k) = \frac{M}{2^k}$ for $1 \le k \le d$, $\gamma(d+1) = 1$. In general, M need not be a power of two. We present the algorithm for constructing H_2 for general M below.

Proposed Code Construction Algorithm

STEP 1 [**Finding Optimal Degree Distribution**] Find an optimal degree distribution for the desired code rate.

STEP 2 [**Parameter Setting**] For a given design parameter, M (number of parity symbols), obtain the depth d and $\gamma(k)$ as Set the size of k-SR matrix as $M \times \gamma(k)$.

STEP 3 [**Generating *k*-SR matrix**] The j-th column of k-SR matrix has the following sequence:

$$h_{k,j} = \begin{cases} D^{j+S_{k-1}}\left(1+D^{\gamma(k)}\right), & \mathit{for} \quad 1 \le k \le d \\ D^{M-1}, & \mathit{for} \quad k = d+1 \end{cases}, \quad \mathit{where} \quad 0 \le j \le \gamma(k)-1.$$

STEP 4 [**Constructing matrix *T***] Construct the matrix T as follows:

$$T = \left[\mathit{1}\text{-SR matrix} \mid \mathit{2}\text{-SR matrix} \mid \cdots \mid d\text{-SR matrix}\right].$$

STEP 5 [**Forming matrix *H₂***] Add a degree-1 node to T and form

$$H_2 = \left[T \mid (d+1)\text{-SR matrix}\right].$$

STEP 6 [**Edge Construction for *H₁***] Construct the matrix H_1 by matching the degree distribution (STEP 1) as closely as possible.

STEP 7 [**Constructing matrix *H***] Assign H_1 as systematic parts and H_2 as nonsystematic parts:

$$H = [H_1 \mid H_2].$$

In STEP 1, we first find an optimal degree distribution for the desired mother code rate, say R_L, using the density evolution [7]. When we determine the degree distribution, the number of degree-2 nodes, $N_v(2)$, is an important factor. The E^{2}RC codes are designed so that all the degree-2 nodes in the nonsystematic part can be punctured. This will give us the achievable highest puncturing rate, say R_H. Then, $R_H = K/(N - N_v(2))$. Thus, the E^{2}RC codes can provide an ensemble of rate-compatible codes of rate R_L~R_H. Since we now consider a design when $N_v(2) = M - 1$ so that all the parities can be punctured, $R_H = 1.0$. In STEP 2, we set the design parameters. We try to maximize the number of low-SR nodes while the increasing the row degree is restrained. In fact, we design the function $\gamma(k)$ such that it assign the half of the parities as *1*-SR nodes, and the half of the remaining parities as *2*-SR nodes, and so on. We can set the depth d as $d = \lceil \log_2 M \rceil$, and $\gamma(k)$ as

$$\gamma(k) = \left\lfloor M - \frac{1}{2}\sum_{i=0}^{k-1}\gamma(i) \right\rfloor \quad \text{for } 1 \le k \le d,$$

$$\gamma(d+1) = 1, \text{ and } \gamma(0) \triangleq M,$$

where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are the ceiling function and the floor function, respectively. We observe that the function $\gamma(k)$ has several interesting facts as follows:

Fact 4.1: S_d is equal to $N_v(2)$ in the nonsystematic part, where $d = \lceil \log_2 M \rceil$.

Proof: From the definition, M should be $2^{d-1} < M \leq 2^d$. By definition, M can be represented by $M = 2 \cdot \gamma(1) + R_1$, where R_1 is the remainder when M is divided by 2, i.e., R_1 = 0, or 1. Then, we have

$$\begin{aligned} &M - \gamma(1) = \gamma(1) + R_1 = 2 \cdot \gamma(2) + R_2 \\ &M - \gamma(1) - \gamma(2) = \gamma(2) + R_2 = 2 \cdot \gamma(3) + R_3 \\ &\cdots \\ &M - \gamma(1) - \gamma(2) - \cdots - \gamma(d-1) = \gamma(d-1) + R_{d-1} = 2 \cdot \gamma(d) + R_d \end{aligned} \tag{a}$$

In the above equations, the remainders can be $R_1, R_2, ..., R_d$ = 0, or 1. From (a), we can also have

$$M + R_1 = 2 \cdot (\gamma(1) + R_1) = 2 \cdot (2 \cdot \gamma(2) + R_2)$$

$$M + R_1 + 2 \cdot R_2 = 2^2 \cdot (\gamma(2) + R_2) = 2^2 \cdot (2 \cdot \gamma(3) + R_3)$$

$$\cdots$$

$$M + R_1 + 2 \cdot R_2 + \cdots 2^{d-2} \cdot R_{d-1} = 2^{d-1} \cdot (\gamma(d-1) + R_{d-1}) = 2^d \cdot \gamma(d) + 2^{d-1} \cdot R_d \tag{b}$$

$$M + R_1 + 2 \cdot R_2 + \cdots 2^{d-1} \cdot R_d = 2^d \cdot (\gamma(d) + R_d) \tag{c}$$

From (b), LHS is greater than 2^{d-1} from the range of M. So, $\gamma(d) \geq 1$ in RHS since R_d = 0 or 1. On the other hand, $\gamma(d) + R_d$ in (c) should be 1 since the sum of LHS is less than 2^{d+1}. Thus, we conclude that $\gamma(d)$ is 1 and R_d is 0. Then, from (a), we have $\gamma(1) + \gamma(2) + \cdots + \gamma(d) = M - \gamma(d) = M - 1$.

■

Fact 4.2: $\gamma(k) \geq 1$ for $1 \leq k \leq d$.

Proof: It is obvious that $\gamma(1) \geq \gamma(2) \geq \cdots \geq \gamma(d)$ and $\gamma(d) = 1$ from the proof of Fact 4.1.

■

From the generation sequence in STEP 3, we can notice that *k*-SR matrix is composed of only degree-2 variable nodes except for the last (*d+1*)-SR matrix.

Observation 4.1: Every column in *k*-SR matrix has degree two. In particular, when $N_v(2) = M - 1$, all the columns of the nonsystematic part have degree two except the last column which has degree one.

After generating *k*-SR matrix, we put together *k*-SR matrices in the matrix *T* in STEP 4. Then in STEP 5, we construct H_2 matrix, nonsystematic part of *H* matrix, $H_2 = \left[T \mid (d+1)\text{-SR matrix}\right]$ by adding a degree-1 column at the end of H_2. The following Example 4.2 is to help understand the construction of H_2 matrix of the proposed algorithm.

Example 4.2: For $M = 10$ and $N_v(2) = 9$, the depth d = 4, and $\gamma(1)$ = 5, $\gamma(2)$ = 2, $\gamma(3)$ = 1, $\gamma(4)$ = 1.

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

■

In the next STEP 6, we construct edges for the matrix H_1 trying to keep the degree distribution obtained from STEP 1. Finally, combining H_1 and H_2 make the whole parity-check matrix in STEP 7. Note that the degree distribution of the nonsystematic part is already fixed by the construction algorithm (see Observation 4.1 and Corollary 4.1). From the above Example 4.1 and 4.2, we can observe the right degree distributions of H_2. Except for the last row degree, we notice that the number of degree-k rows is the number of columns in k-SR matrix, which is exactly $\gamma(k)$. Lemma 4.1 explains the details.

Lemma 4.1: In the matrix H_2, any column in k-SR matrix is connected to at least one row of degree-k. Furthermore, this row has exactly one connection to a column from each l-SR matrix, where $1 \le l < k \le d$.

Proof: Consider the j_k-th column in the k-SR matrix. Its sequence is given by

$$\begin{aligned} h_{k,j_k} &= D^{j_k+S_{k-1}}\left(1+D^{\gamma(k)}\right) \\ &= D^{j_k+S_{k-1}} + D^{j_k+S_k}, \text{ where } 0 \le j_k \le \gamma(k)-1. \end{aligned}$$

We shall demonstrate that the first entry of h_{k,j_k} is connected to a column in the l-SR matrix for $1 \le l < k$. First, note that H_2 is lower-triangular with ones in the diagonal. An immediate consequence of this fact is that h_{k,j_k} can only be connected to the second entry of h_{l,j_l}, the j_l-th column in the l-SR matrix. Suppose that the second entry of the j_l-th column in the l-SR matrix is connected to the first entry of the j_k-th column in the k-SR matrix. This implies that $j_l + S_l = j_k + S_{k-1}$. Clearly $j_l = j_k + S_{k-1} - S_l \ge 0$ since $k > l$ and $0 \le j_k \le \gamma(k) - 1$. We shall now show that $j_l = j_k + S_{k-1} - S_l \le \gamma(l) - 1$. This means that for a given j_k, it is possible to find a unique column j_l belonging to the l-SR matrix to which it is connected. From the proof of Fact 4.1, we have $S_i = M - \gamma(i) - R_i$, where $R_i = 0$ or 1. Then, we have

$$\begin{aligned} j_l = j_k + S_{k-1} - S_l &\le \gamma(k) - 1 + S_{k-1} - S_l \\ &= S_k - S_l - 1 \\ &\le S_d - S_l - 1 \\ &= M - \gamma(d) - R_d - \left(M - \gamma(l) - R_l\right) - 1 \\ &= \gamma(l) - 1 - R_l - 1 \\ &\le \gamma(l) - 1. \end{aligned}$$

Therefore, for a given j_k, we can find a corresponding j_l in the l-SR matrix for $1 \le l < k$. Note that the first entry of j_k is connected to the corresponding j_l. Since the matrix is lower-triangular, this entry cannot have any connection with a m-SR matrix where $m > k$. Therefore this particular row has degree exactly k. This concludes the proof.

■

From Lemma 4.1, we can obtain the exact number of rows with degree-k except the last row. To find out the entire row degree distributions, let's define ζ as the row degree of the last row.

Observation 4.2: The row degree ζ of the last row in the matrix H_2 can be obtained as $\zeta = \sum_{i=1}^{d} [\gamma(i) + S_i - S_d] + 1$.

Proof: Let's consider the connections of the last row with each k-SR matrix. It is easy to see that if $M = 2\cdot\gamma(1)$, there is a connection between the *1*-SR matrix and the last row, otherwise, there is no connection. In the same way, if $M = \gamma(1)+2\cdot\gamma(2)$, there is a connection between the *2*-SR matrix and the last row, and so on. Thus, we can get ζ as

$$\begin{aligned}\zeta &= \underbrace{\big(1-(M-2\gamma(1))\big)}_{\textit{1}\text{-SR matrix}} + \underbrace{\big(1-(M-\gamma(1)-2\gamma(2))\big)}_{\text{2-SR matrix}} + \cdots \\ &\quad + \underbrace{\big(1-(M-\gamma(1)-\gamma(2)-\cdots-2\gamma(d))\big)}_{d\text{-SR matrix}} + \underbrace{1}_{(d+\textit{1})\text{-SR matrix}} \\ &= \sum_{i=1}^{d} [\gamma(i) + S_i - (M-1)] + 1 \\ &= \sum_{i=1}^{d} [\gamma(i) + S_i - S_d] + 1,\end{aligned}$$

since we have $S_d = M - 1$ from Fact 4.1.

■

From Observation 4.2, we can obtain $\zeta = \sum_{i=1}^{4} [\gamma(i) + S_i - 9] + 1 = 3$ for Example 4.2. Since we know ζ, we are ready to get the whole right degree distributions for H_2. The following Observation 4.3 and Corollary 4.1 give the whole right degree distributions.

Observation 4.3: The number of degree-k rows in the matrix H_2 is $\gamma(k) + \delta(k-\zeta)$ for

$1 \le k \le d$, where $\delta(i) = \begin{cases} 1 & \textit{if } i = 0, \\ 0 & \textit{otherwise} \end{cases}$.

Corollary 4.1: The right degree distribution (node perspective) of the matrix H_2 is as follows:

$$\rho(x) = \sum_{i=1}^{d+1} \hat{\rho}_i x^{i-1} \text{, where } \hat{\rho}_i = \frac{\gamma(i) + \delta(i-\zeta)}{M} \text{ for } 1 \le k \le d \text{ and } \hat{\rho}_{d+1} = \frac{\delta(i-\zeta)}{M}.$$

Proof: First, consider the k-SR matrix when $1 \le k \le d$. From the Lemma 4.1, if we pick a column in the k-SR matrix, the first element of the column is included in a row of degree k, and the second element of the column has the row degree greater than k. The number of columns in the k-SR matrix is $\gamma(k)$ and each column is connected to one degree-k row. Thus, the number of rows having degree k is at least $\gamma(k)$ except the last row. For a $(d+1)$-SR matrix, there is only one degree-ζ row. From Fact 4.1, summing the number of rows having degree-k results in $\gamma(1) + \gamma(2) + \cdots + \gamma(d) + 1 = M$. Therefore, the number of rows of degree k except the last row is exactly $\gamma(k)$.

■

From Observation 4.1 and Corollary 4.1, we can determine the exact degree distributions for the nonsystematic parts, namely the H_2 matrix. For a desired code rate, we can find the optimal degree distributions for the whole code while fixing these degree distributions for H_2. Then, we can get the degree distributions for the H_1 matrix. For the systematic part, namely the H_1 matrix, we choose variable nodes of higher degree greater than two. Besides finding the optimal degree distributions, there are three additional design rules for finite-length LDPC codes proposed in [7]:

(a) assign degree-2 variable nodes to nonsystematic bits;

(b) avoid short cycles involving only degree-2 variable nodes;

(c) cycle-4 free in the code graph.

The proposed E^2RC codes meet the design rule (a) as stated above. For design rule (b), we will show that there is no cycles involving only degree-2 variable nodes in Lemma 4.2 and 4.3.

Lemma 4.2: Suppose there exists a length-$2s$ cycle in a matrix which consists of only weight two columns. Consider the submatrix formed by the subset of columns that participates in the cycle. Then, all the participating rows in the cycle must have degree two in that submatrix.

Proof: To have a length-$2s$ cycle, the number of columns participating in the cycle needs to be s and the number of rows participating in the cycle needs to be s. Let us denote the submatrix formed by the columns participating in the cycle by U. Then, the number of edges in U is $2s$ since each of the columns has degree two. Each row participating in the cycle must have a degree greater than or equal to two in U since each row has to link at least two different columns in U. Suppose there is a row having degree strictly greater than two in U. Then, there should be a row having a degree less than two in U, i.e., equal to one, since the average row weight in U is two (the number of edges / the number of rows = $2s$ / s = 2), which is a contradiction. This is because a row that has degree-one in U cannot participate in a cycle with the columns in U. Thus, every participating row must have degree two in U.

■

Armed with Lemma 4.2, we will prove that the proposed matrix H_2 is cycle free.

Lemma 4.3: The matrix H_2 constructed by the E^{2}RC construction algorithm is cycle free.

Proof: Suppose that there exist s columns $v_1, v_2, \ldots, v_s$ that form a cycle of length $2s$. We form the $M \times s$ submatrix formed by the columns. Let us denote this submatrix H_s. Suppose that column v_i belongs to the k_i-SR matrix in H_2. Find the minimum value of k_i. Let us call it k_{min}. Applying Lemma 4.1, we have that $v_{k_{\min}}$ has exactly one connection to each l-SR matrix, where $1 \le l < k_{min}$, and no connection to m-SR matrices where $m > k_{min}$, i.e., there is a check node connected to $v_{k_{\min}}$ that is singly-connected in the submatrix H_s. Applying Lemma 4.2, we realize that a cycle cannot exist amongst the s columns.

■

Since all of the nodes (except one) are degree 2 in H_2, the fraction of degree-2 nodes in degree distributions is very high. For a finite length code, the higher portion of degree-2 nodes cause better threshold performance, but a big fraction of degree-2 nodes can result in a small minimum distance, causing a greater probability of decoding errors and higher error floors. To reduce these effects, we can use methods such as those presented in [16] [17] [18] [19] when we construct the H_1 matrix. By doing so, the E^{2}RC codes can meet the design rule (c).

4.2 LOW-RATE CODE DESIGN

Since the E^{2}RC codes have strong point in puncturing, considering mother code design for low rate ($R < 0.5$) is a necessary step. As stated earlier, all the degree-2 nodes in the nonsystematic part of the parity-check matrix can be punctured in our codes. Thus, to get the maximum puncturing characteristic for low rate codes, $(1-R)$ portion of the nodes should be filled with degree-2 nodes. However, it is hard to find a good degree distribution including huge portion of degree-2 nodes. In other words, we should consider the code design which allow some portion of nodes in the nonsystematic part have degree greater than two. This is the reason why we consider the case when $N_v(2) < M-1$. We will briefly explain the difference of the construction algorithm for this case comparing with the case stated earlier. In STEP 1, we find optimal degree distributions for the desired code rate as before, but the target code rate is mainly low rate. For STEP 2, we first determine the size of parities that are not to be punctured, which is *l* in Figure 4.2(a). Since the E^{2}RC codes regard all the degree-2 nodes in the nonsystematic part to be punctured, the size of unpunctured nodes in the nonsystematic part can be $l = M - N_v(2)$. Then, the size of the matrix L is $M \times l$. We set the depth *d* as $d = \lfloor \log_2 M - \log_2 l \rfloor$, and obtain *γ*(*k*) the same as the previous settings for $1 \le k < d$. However, the previous settings for *γ*(*k*)'s are designed to match $S_d = N_v(2) = M-1$. When $N_v(2) < M-1$, we set $\gamma(d) = N_v(2) - S_{d-1}$ so that they can satisfy $S_d = N_v(2)$. To generate the sequence of *d*-SR matrix, we set

$$\delta = \left\lfloor M - \frac{1}{2}\sum_{i=0}^{d-1}\gamma(i) \right\rfloor.$$

Then, the the *j*-th column of *k*-SR matrix of STEP 3 has the following sequence:

$$h_{k,j} = \begin{cases} D^{j+S_{k-1}}\left(1+D^{\gamma(k)}\right), & for \quad 1 \le k < d \\ D^{j+S_{k-1}}\left(1+D^{\delta}\right), & for \quad k = d \end{cases}, \quad where \quad 0 \le j \le \gamma(k)-1.$$

We formulate T the same as in STEP 4, then we set $H_2 = [L \mid T]$ in STEP 5, where variable nodes in the matrix L have degree higher than two. Note that we do not put the degree-1 node in H_2. In STEP 6, we only need to construct edges for the matrix L and H_1 trying to keep the degree distribution obtained from STEP 1. Finally in STEP 7, we accomplish the parity-check matrix by putting together H_1 and H_2 the same as before. In the case when $N_v(2) < M - 1$, we can say that the submatrix formed by the columns of $N_v(2)$ is cycle free since we generate the sequence same as before.

For the proposed codes, rate-compatibility can be easily obtained by puncturing nodes of degree two from left to right in the H_2 matrix. For a desired code rate R_p obtained from puncturing the mother code of rate R_L, the number of puncturing symbols $p = N\left(1 - R_L / R_p\right)$, where N is the code length and $R_L \leq R_p \leq R_H$. Equivalently, we can achieve any desired code rates by puncturing first p nodes from the first node in *1*-SR matrix. This can be a good advantage when it is applied to IR Hybrid-ARQ systems, which will be discussed in the next chapter.

Another big advantage for the proposed codes (when $N_v(2) = M - 1$) is that even if all the parity bits are punctured, they can be recovered completely after (d+*1*) iterations using a erasure decoder or a LDPC decoder when the channel has no errors. This is because the k-SR nodes in k-SR matrix can be recovered after k iterations with the help of other unpunctured nodes and lower-SR nodes. This property can be used to encode. The proposed codes not only have simple rate-compatible puncturing scheme but also an efficient encoding structure. We describe the encoding structure in the following section.

4.3 EFFICIENT ENCODER IMPLEMENTATION

In this work, we propose a new encoding method which can be applied to other block codes as well as E^{2}RC codes. First, we will explain the case when $N_v(2) = M-1$. For the parity-check matrix $H = \left[H_1 \,|\, H_2\right]$ of an E^{2}RC code obtained from the proposed construction algorithm, let a codeword $c = \left[m \,|\, p\right]$, where m is the systematic symbols, and p is nonsystematic symbols. Let the systematic generator matrix G is given by $G = \left[I_k \,|\, P\right]$. As we stated in the section 2.4, the systematic codeword can represented by $c = m \cdot G = m \cdot \left[I_k \,|\, P\right] = \left[m \,|\, m \cdot H_1^T \cdot H_2^{-T}\right]$. From the construction sequence of E^{2}RC codes, H_2 is the lower triangular matrix. Let L_M be the $M \times M$ lower triangular matrix of H_2. For M is power of 2, we have the following results by inspection:

$$\begin{aligned} H_2 &= L_M \\ &= \begin{bmatrix} I & O \\ I & L_{M/2} \end{bmatrix}, \end{aligned}$$

and

$$\begin{aligned} H_2^{-T} &= L_M{}^{-T} \\ &= \begin{bmatrix} I & L_{M/2}{}^{-T} \\ O & L_{M/2}{}^{-T} \end{bmatrix}. \end{aligned}$$

We can easily check that

$$\begin{aligned} H_2^T H_2^{-T} &= \begin{bmatrix} I & L_{M/2}{}^{-T} + L_{M/2}{}^{-T} \\ O & L_{M/2}{}^{T} \cdot L_{M/2}{}^{-T} \end{bmatrix} \\ &= \begin{bmatrix} I & O \\ O & I \end{bmatrix} \\ &= I. \end{aligned}$$

Here, we can obtain L_M recursively;

$$L_1 = 1,$$
$$L_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix},$$
$$L_4 = \begin{bmatrix} I & O \\ I & L_2 \end{bmatrix},$$
$$\cdots$$

Likewise, we can get L_M^{-T} recursively;

$$\begin{aligned}
H_2^{-T} = L_M^{-T} &= \begin{bmatrix} I & L_{M/2}^{-T} \\ O & L_{M/2}^{-T} \end{bmatrix} \\
&= \left[\begin{array}{ccc} & I & L_{M/4}^{-T} \\ I & O & L_{M/4}^{-T} \\ & I & L_{M/4}^{-T} \\ O & O & L_{M/4}^{-T} \end{array}\right] \\
&\cdots \\
&= \left[\begin{array}{ccccc} & & I & \cdots & L_1^{-T} \\ & I & O & \cdots & L_1^{-T} \\ I & & I & \cdots & \cdot \\ & O & O & \cdots & \cdot \\ & & I & \cdots & \cdot \\ & I & O & \cdots & \cdot \\ O & & I & \cdots & \cdot \\ & O & O & \cdots & L_1^{-T} \end{array}\right] \\
&= \left[\begin{array}{ccccc} & & I & \cdots & 1 \\ & I & O & \cdots & 1 \\ I & & I & \cdots & \cdot \\ & O & O & \cdots & \cdot \\ & & I & \cdots & \cdot \\ & I & O & \cdots & \cdot \\ O & & I & \cdots & \cdot \\ & O & O & \cdots & 1 \end{array}\right].
\end{aligned}$$

It is possible to show that the multiplication with H_2^{-T} can be implemented simply with a shift-register circuit. Thus, an example of encoder is shown in Figure 4.3.

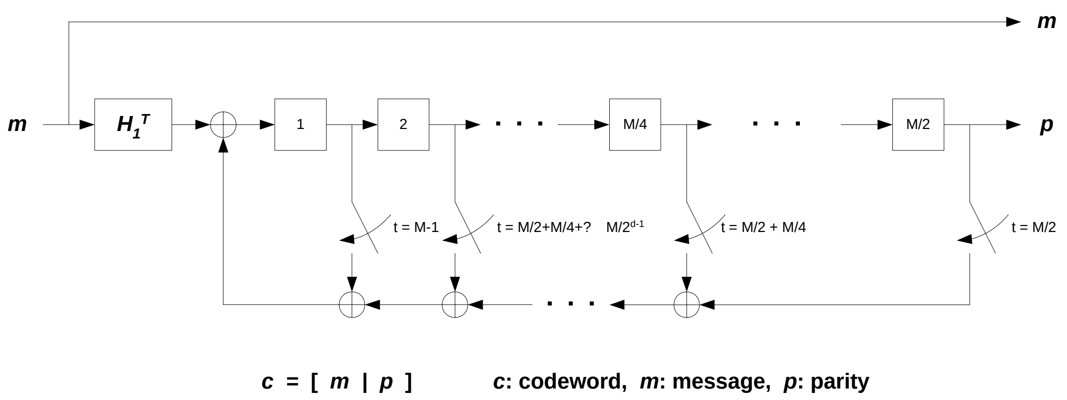

m
m
H_1^T
1
2
M/4
M/2
p
t = M-1
t = M/2+M/4+?   M/2^{d-1}
t = M/2 + M/4
t = M/2
c = [m | p]
c: codeword, m: message, p: parity

Figure 4.3 An example of shift-register implementation of E^{2}RC codes when $M = 2^d$ case.

So far, we have explained the case when M is power of 2. To derive an efficient encoder for general case, we set $H \cdot c^T = [H_1 \mid H_2] \cdot [m \mid p]^T = H_1 m^T + H_2 p^T = 0$. Let $s^T = H_1 m^T$, then we have $H_2 p^T = H_1 m^T = s^T$. Let $H_2 = \left(h_{i,j}\right)_{1 \le i,j \le M}$, then $s_i = \sum_{j=1}^{M} h_{ij} p_j = \sum_{j=1}^{i-1} h_{ij} p_j + p_i$ since $h_{ij} = 1$ for $i = j$ and $h_{ij} = 0$ for $i < j$ (since H_2 is lower triangular) in the construction of the E^{2}RC codes.

$$H_2 \cdot p^T = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1M} \\ h_{21} & h_{22} & \cdots & h_{2M} \\ h_{31} & h_{32} & \cdots & h_{3M} \\ \vdots & \vdots & \ddots & \vdots \\ h_{M1} & h_{M2} & \cdots & h_{MM} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_M \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_M \end{bmatrix}.$$

By observing the sequence for *1*-SR matrix construction, we can notice that the elements between the two entries of the sequence and below the second entry of the sequence are 0, that is,

$$h_{ij} = \begin{cases} 0 & , 1 \le i \le \gamma(1),\ i < j \\ 1 & , \gamma(1)+1 \le i \le M,\ j = i-\gamma(1) \\ 0 & , \gamma(1)+1 \le i \le M,\ j < i-\gamma(1) \end{cases}.$$

Then we have

$$p_i = \begin{cases} s_i, & for\ 1 \le i \le \gamma(1) \\ s_i + \sum\limits_{j=i-\gamma(1)}^{i-1} h_{ij} p_j, & for\ \gamma(1)+1 \le i \le M \end{cases}.$$

The above results tell us that we can get p_i with using previous $\gamma(1)$ p_i's, which enables us to implement the E^{2}RC encoder by using $\gamma(1)$ shift registers. The following example explains encoding method more detail.

Example 4.3: For M=7, we can construct H_2 matrix as follows:

$$H_2 \cdot p^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix}.$$

By doing some matrix operations, we get the following equations: $p_1 = s_1$, $p_2 = s_2$, $p_3 = s_3$, $p_1 + p_4 = s_4$, $p_2 + p_5 = s_5$, $p_3 + p_4 + p_6 = s_6$, and $p_5 + p_6 + p_7 = s_7$. Then, we can obtain p_i's by using p_j's, where $j < i$: $p_1 = s_1$, $p_2 = s_2$, $p_3 = s_3$, $p_4 = p_1 + s_4$,

$p_5 = p_2 + s_5$, $p_6 = p_3 + p_4 + s_6$, and $p_7 = p_5 + p_6 + s_7$. Then, we only need $\gamma(1) = 3$ number of registers for the encoder in Figure 4.4. The coefficients for multiplication in Figure 4.4 can be obtained from the above sliding windows of the rectangular in the matrix equation. For this reason, we will refer to this encoding method as *sliding window method*. The coefficient g_i's are time varying according to the rectangular windows. Assuming that the window starts from the first row at initial time t=0, g_0 will be on at t=3-5, g_1 will be on at t=5-6, and g_2 will be on at t=6.

■

From the Example 4.3, we can generalize the shift-register encoder implementation of E^{2}RC codes. The encoder can be represented as division circuit as shown in Figure 4.4. We can represent the division circuit in Figure 4.4 as a generator polynomial as $g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{\gamma(1)-1} x^{\gamma(1)-1} + x^{\gamma(1)}$.

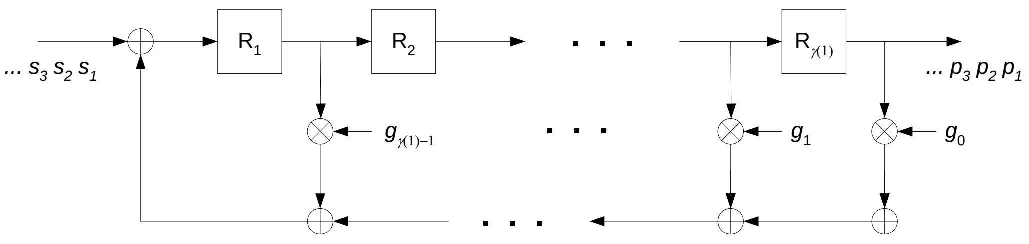


Figure 4.4 An example of shift-register implementation of E^{2}RC codes.

By observing the matrix H_2, we can obtain the coefficients of the polynomial. As in Figure 4.5, let us think about the window of size w. As we slide down the window from the first row to the last row, we can get a parity-check equation one by one. The coefficients in the window will change or stay between 0 and 1 for each row. If we trace the time-varying coefficients, then we can implement the shift-register encoder of Figure

4.4. We set the window size w as $\gamma(1)$ since the largest distance between nonzero elements in a row of H_2 is $\gamma(1)$ from the E^{2}RC code construction algorithm. We can set the window size differently for other codes. In the sliding window, the first entry corresponds to g_0, and the last entry to $g_{\gamma(1)-1}$.

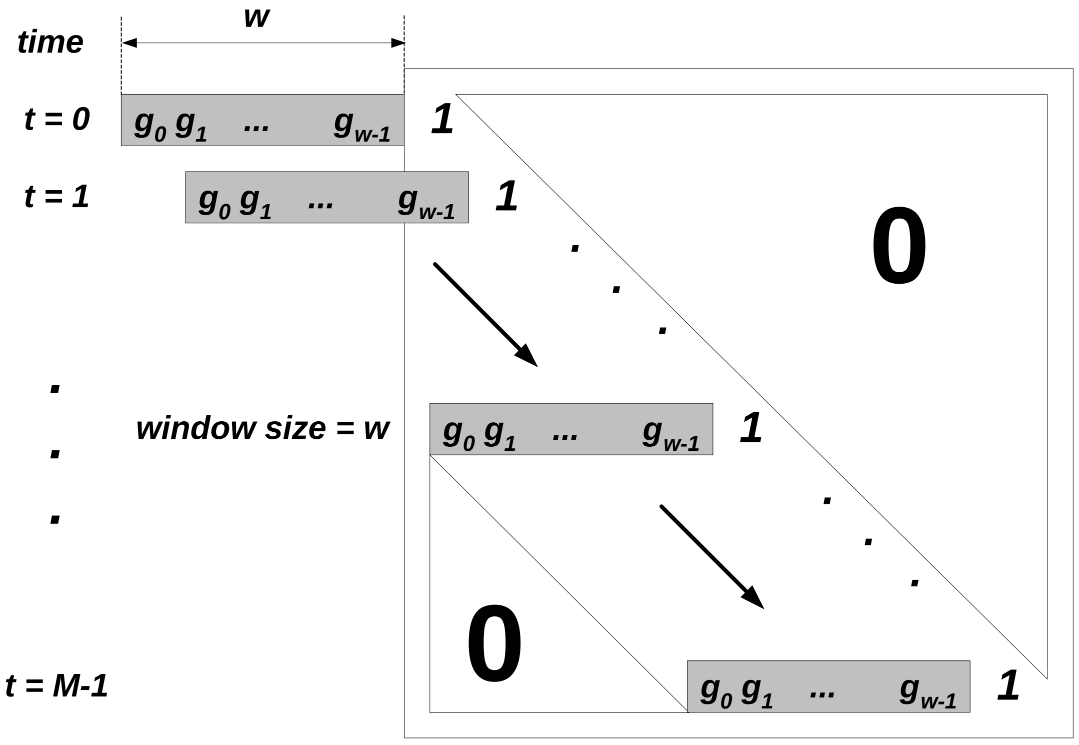


Figure 4.5 Nonsystematic part of a parity-check matrix for applying sliding window encoding method.

Let us define the time is zero when the window starts from the first row. The initial ($t = 0$) status of coefficients is 0. In our code construction, note that g_i can exist only if $i = \gamma(1) - \gamma(k)$ for $1 \le k \le d$. In other words, we only have to consider d coefficients and other than those are all zero. For a such coefficient g_i, it is on at time $t = S_k$, and will last until the window reach the last row ($t = S_d$) if there is a connection for *k*-SR matrix in the last row. Otherwise, it will be off at the last row. Figure 4.6 shows the timing diagram of coefficients.

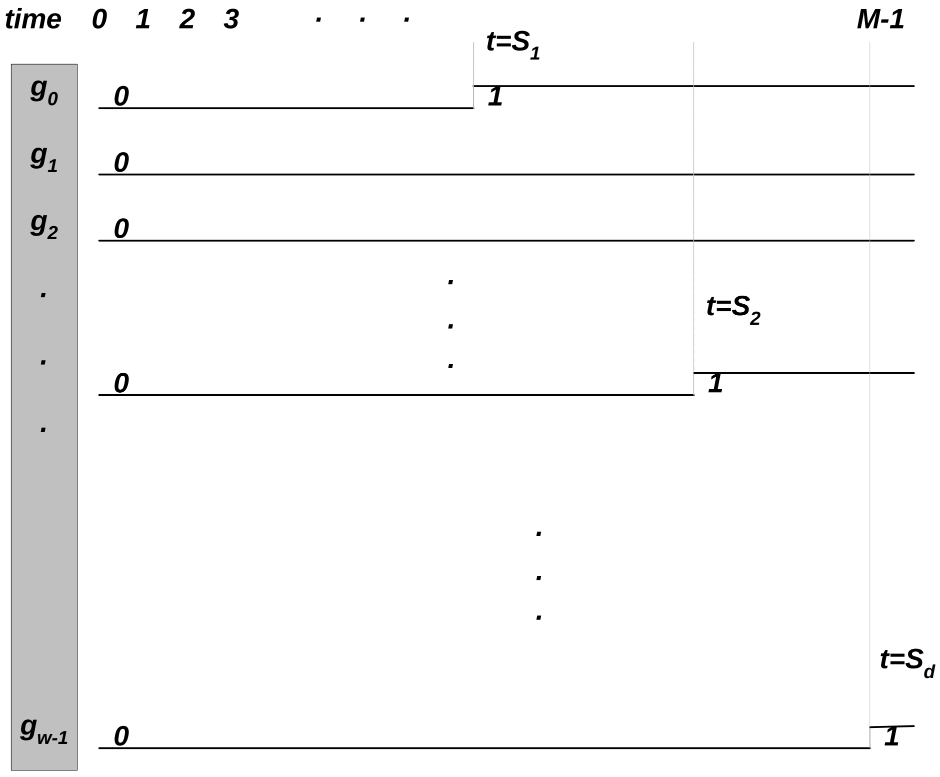


Figure 4.6 Timimg diagram of coefficients of sliding window encoder.

From Observation 4.2, note that there is a connection for *k*-SR matrix in the last row if $\gamma(k) + S_k - S_d = 1$ and no connection if the value is 0. Then, the coefficients of the generator polynomial $g(x)$ can be represented as

$$g_i = \sum_{k=1}^{d} \delta(i-\gamma(1)+\gamma(k))\{u(t-S_k)-\delta(\gamma(k)+S_k-S_d)\cdot u(t-S_d)\},$$

where we define the unit step function as follows:

$$u(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0. \end{cases}$$

The E^{2}RC codes have similarity with cyclic codes in the sense that they can be represented as a generator polynomial. The only difference is that the coefficients of the generator polynomial are time-varying. For the above Example 4.3 when M =7, $g_0 = u(t-3)-u(t-6)$, $g_1 = u(t-5)$, $g_1 = u(t-6)$. As mentioned earlier, the proposed sliding window encoding method can be applied any other block codes if the nonsystematic part of their parity-check matrix has lower-triangular form as shown in Figure 4.5. In fact, the window size can be lowered if the lower-triangular form in Figure 4.5 has lower-triangular 0's in it, which can be achieved by column and row permutation for a given parity-check matrix.

Another way to implement the encoder of the proposed E^{2}RC codes is by using a simple iterative erasure decoder. Recall that all the nodes in *k*-SR matrix can be recovered in *k* iterations with erasure decoder since they are all *k*-SR nodes. For the proposed codes, even if all the parity bits are erased, we can obtain the exact parity bits within (d+1) iterations using a simple erasure decoder or general LDPC decoder of message-passing algorithm as long as the systematic bits are known exactly (this is the case at the encoder). In a transceiver system, this can be a big advantage in terms of complexity. We only need to provide an LDPC decoder for both encoding and decoding, and do not need any extra encoder.

Even though we may not use the shift-register implementation of sliding-window method for the encoder when $N_v(2) < M-1$, we can easily apply the efficient encoding method proposed in [8]. To match notations in [8], let the parity-check matrix H represent as $H = \begin{bmatrix} A & B & C \\ D & E & F \end{bmatrix}$. Then, $\begin{bmatrix} A \\ D \end{bmatrix}$ is the systematic part of E^{2}RC codes, $\begin{bmatrix} B \\ E \end{bmatrix} = L$, and $\begin{bmatrix} C \\ F \end{bmatrix} = T$ is *k*-SR matrices. For E^{2}RC codes, we know the exact sequence of the matrix *T*. Furthermore, the matrix *C* is a lower triangular with ones in the diagonal, which does not require preprocess. These make us easy to apply the efficient encoding method in [8] to E^{2}RC codes.

4.4 SIMULATION RESULTS

We consider rate-1/2 mother codes with block length of 1024. To compare the puncturing performance of the E^{2}RC codes with that of other LDPC codes, we generate eIRA codes and general irregular LDPC codes of which degree distributions are optimized in AWGN channel as those in [11] for rate-1/2 codes:

$$\lambda(x) = 0.30780x + 0.27287x^2 + 0.41933x^6$$

$$\rho(x) = 0.4x^5 + 0.6x^6.$$

For E^{2}RC codes, however, the actual degree distributions are slightly different to compensate for the right degree of H_2:

$$\lambda(x) = 0.00030 + 0.30210x + 0.27136x^2 + 0.42625x^6$$

$$\rho(x) = 0.41147x^5 + 0.54626x^6 + 0.01892x^7 + 0.01064x^8 + 0.00592x^9 + 0.00325x^{10} + 0.00354x^{11}.$$

We apply the algorithm proposed in [16] [17] [18] [19] to H_1 design for having the better girth characteristics. We design eIRA codes of length 1026 to compare the performance between the proposed E^2RC codes and the eIRA codes. We also use the algorithm of [16] [17] [18] [19] to design the systematic part of eIRA codes. As shown in Figure 4.7, the mother code performance of two codes shows almost the same over the AWGN channel. However, E^2RC codes outperform eIRA codes at every puncturing rate. In this simulation, we adopt the random puncturing strategy for puncturing eIRA codes.

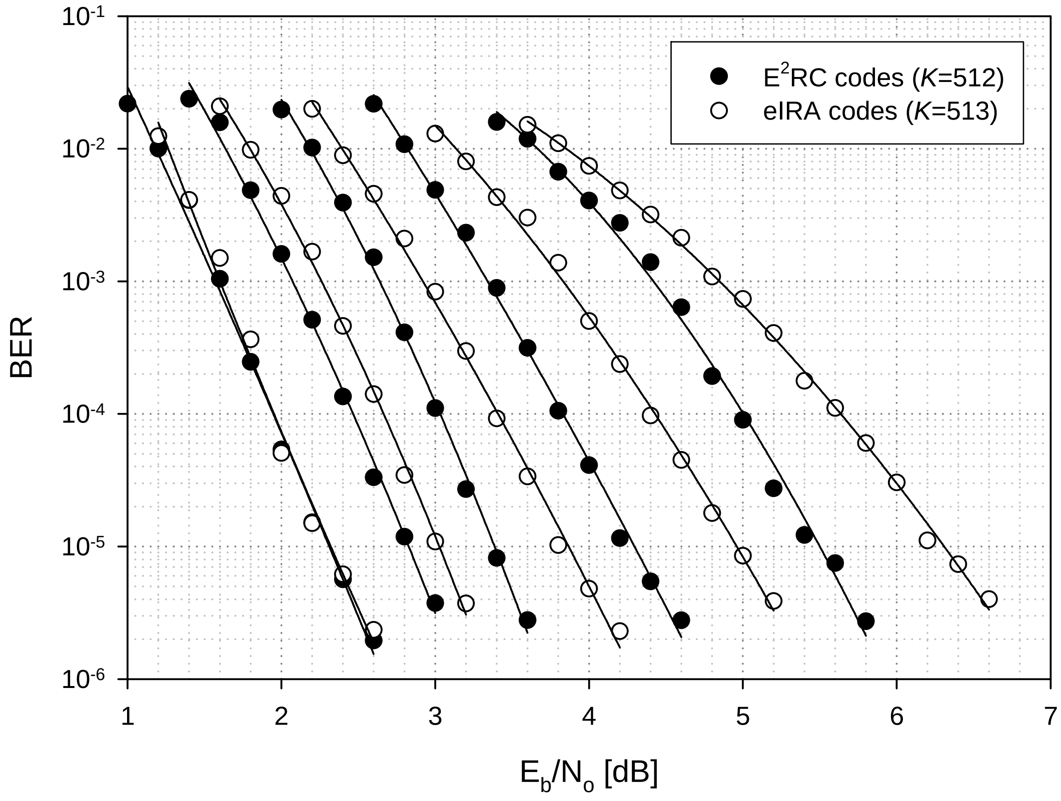


Figure 4.7 Puncturing performance comparison between the proposed E^2RC codes (filled circle) of length=1024 and the eIRA codes (unfilled circle) of length=1026 with random puncturing. Curves are for rate=0.5 (mother code), 0.6, 0.7, 0.8 and 0.9 from left to right.

Next, we apply the intentional puncturing algorithm to the eIRA codes, but in this case we face puncturing limitations. In fact, the intentional puncturing assigns 256 nodes as *1*-SR nodes and cannot find further *k*-SR nodes ($k \geq 2$) if we try to maximize the number of *1*-SR nodes. To get a high rate (R = 0.7, 0.8, 0.9) in eIRA codes, we puncture randomly after the puncturing limitation (256 *1*-SR nodes), which destroys the previous tree structure of *1*-SR nodes, resulting in poor performance. As shown in Figure 4.8, the puncturing performance of the E^{2}RC codes is better than that of eIRA codes as the code rates are increased. For a code rate of 0.9, the E^{2}RC codes show 1.1dB of E_b/N_o better than that of eIRA codes at a BER of 10^{-5}.

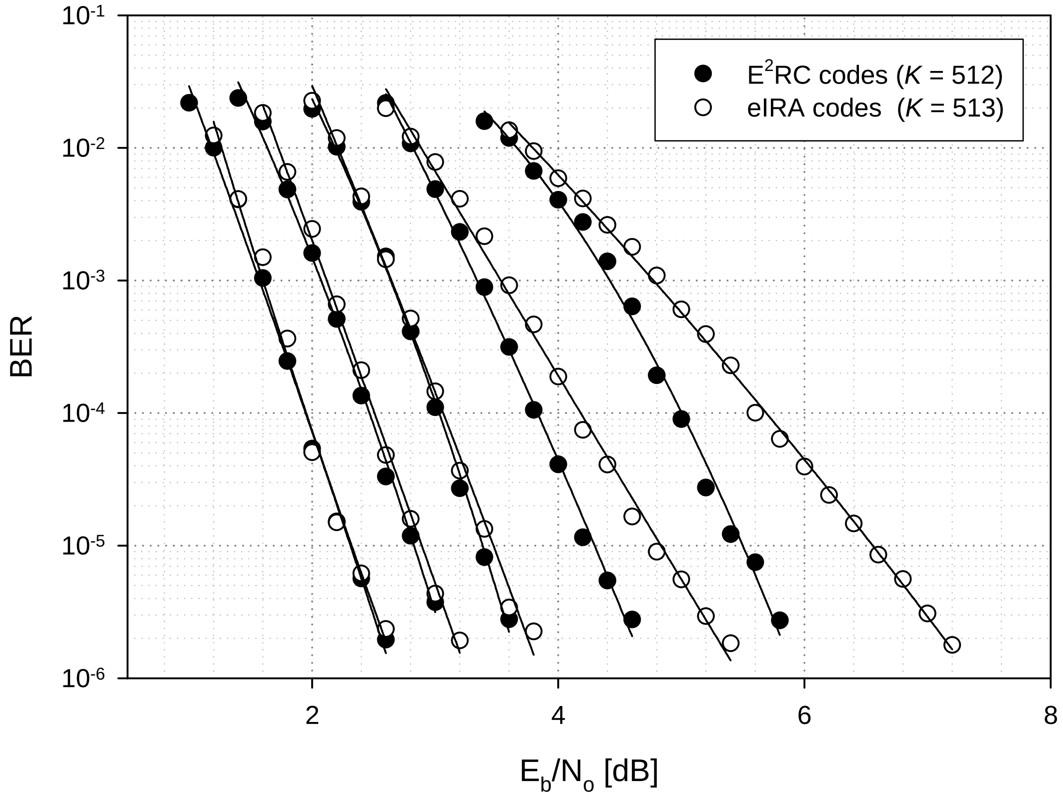


Figure 4.8 Puncturing performance comparison between the proposed E^{2}RC codes (filled circle) of length=1024 and the eIRA codes (unfilled circle) of length=1026 with the intentional puncturing. Curves are for rate=0.5 (mother code), 0.6, 0.7, 0.8 and 0.9 from left to right.

To compare the puncturing performance with general irregular LDPC codes, we generate an irregular LDPC code having the same degree distribution as in [11]. The code length of this code is 1026, and we also apply the algorithm in [16] [17] [18] [19] to generate the code. From the rate-1/2 mother codes, we generate punctured codes of rate 0.6, 0.7, 0.8, and 0.9 using random puncturing and the intentional puncturing algorithm. For the random puncturing case as in Figure 4.9, the performance gaps are large. For code rate of 0.8, the E^2RC codes show 2.8dB of E_b/N_o better than that of the general irregular LDPC codes at a BER of 10^{-5}.

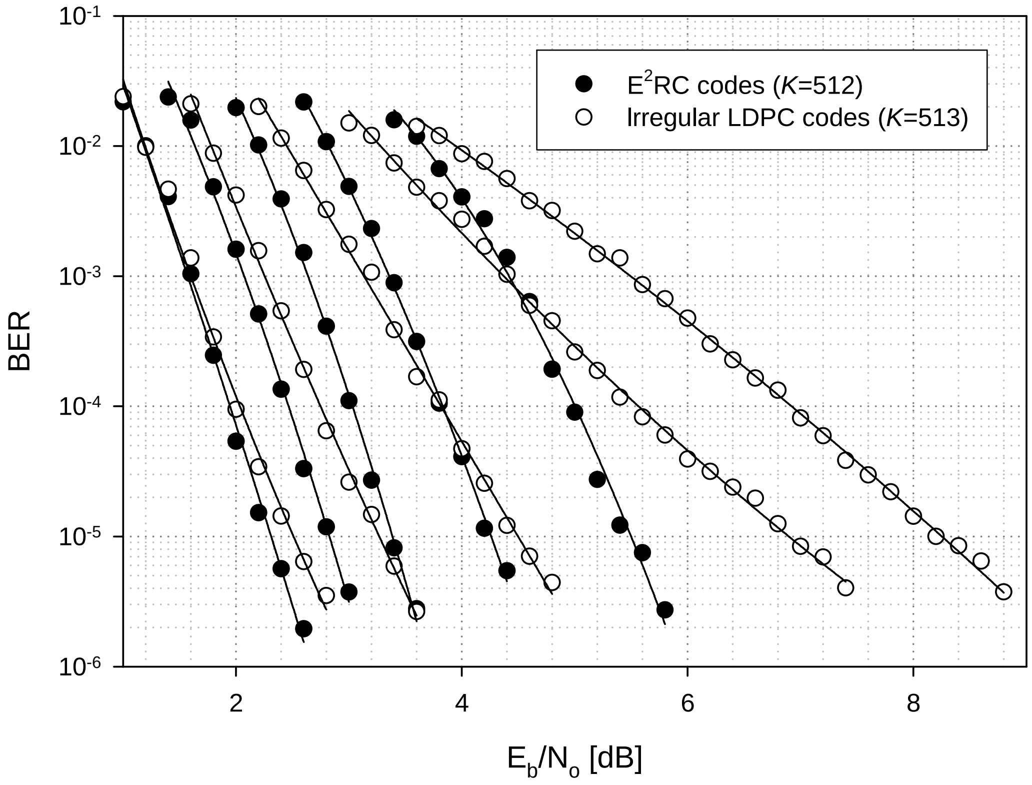


Figure 4.9 Puncturing performance comparison between the proposed E^2RC codes (filled circle) of length=1024 and the irregular LDPC codes (unfilled circle) of length=1026 with random puncturing. Curves are for rate=0.5 (mother code), 0.6, 0.7, 0.8 and 0.9 from left to right.

The intentional puncturing case is shown in Figure 4.10, the proposed E^2RC codes show better performance in all ranges of rates over the AWGN channel. For the rate 0.7 case, the puncturing of proposed codes is 0.2dB better than that of the general irregular LDPC codes at a BER of 10^{-5} and for the rate of 0.9 the gain increases to 1.2dB.

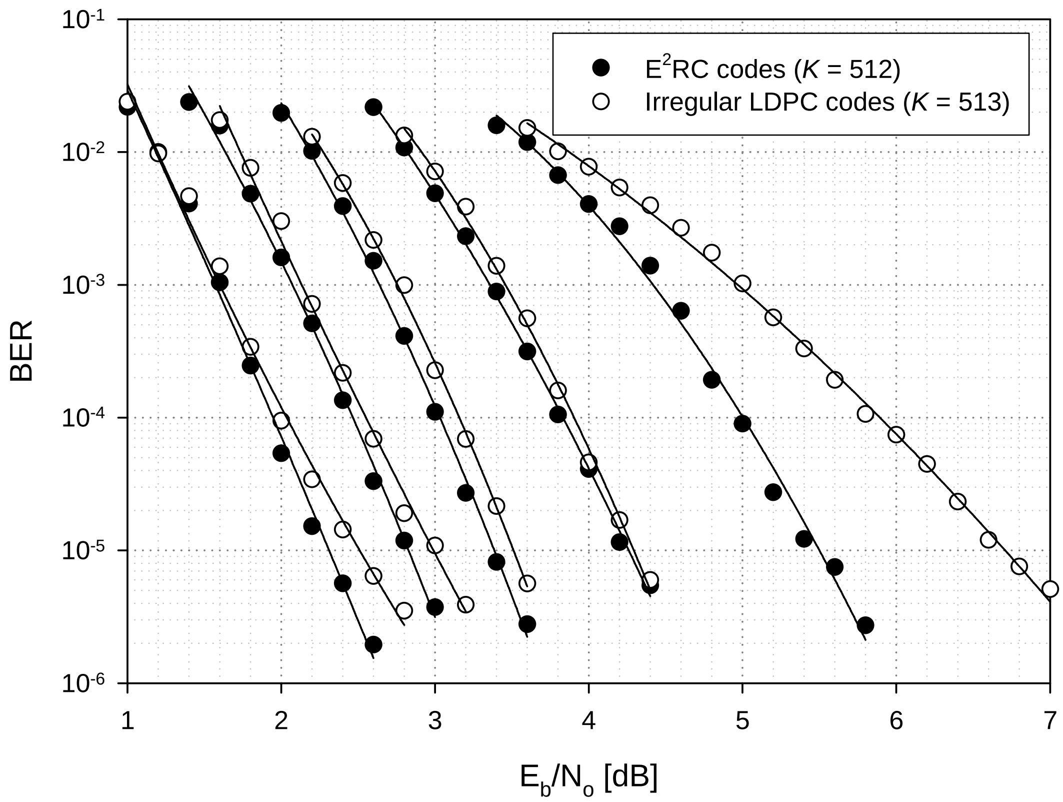


Figure 4.10 Puncturing performance comparison between the proposed E^2RC codes (filled circle) of length=1024 and the irregular LDPC codes (unfilled circle) of length=1026 with the intentional puncturing. Curves are for rate=0.5 (mother code), 0.6, 0.7, 0.8 and 0.9 from left to right.

For practical purpose, designing a low rate E^2RC code and providing a wide range of rates by puncturing are useful. There are other methods to lower the rates such as extending and shortening. However, these methods often increase hardware complexity or the performance of lower rate code has not been proved analytically good. On the other hand, puncturing from the low rate mother codes has limitation, which is, general LDPC codes severely degrade their performance as they are punctured. The E^2RC codes show no such performance degradation when punctured as other LDPC codes. For E^2RC codes, all the degree-2 nodes in the parities can be punctured.

As an example, we consider a rate-0.4 mother code of which degree distributions are optimized in AWGN channel:

$$\lambda(x) = 0.29472x + 0.25667x^2 + 0.44861x^9$$

$$\rho(x) = x^5.$$

In this case, 88.4% of the parities are degree-2 nodes and the remaining 11.6% of the parities are degree-3 nodes. Thus, the structure of E^2RC codes is changed from the original one, and the E^2RC codes can achieve rate of 0.85 since all the degree-2 nodes can be punctured. For rate-0.4 mother code with $N = 2000$, $K = 800$, and $N_v(2) = 1061$, we have the depth $d = 4$, and $\gamma(1) = 600$, $\gamma(2) = 300$, $\gamma(3) = 150$, $\gamma(4) = 11$. In addition, the E^2RC codes can have perfect right degree concentration at degree 6. We apply the PEG algorithm to generate matrix other than degree-2 parities.

To compare the puncturing performance, the general irregular LDPC codes with the same degree distributions as above are generated by using the PEG algorithm. The best-effort intentional puncturing algorithm is applied to the general irregular LDPC codes. The maximum achievable rate of this general irregular LDPC code is 0.69 with intentional puncturing. So, after the limit we apply random puncturing. The puncturing performance comparison between E^2RC codes and general irregular LDPC codes is depicted in Figure 4.11 and Figure 4.12.

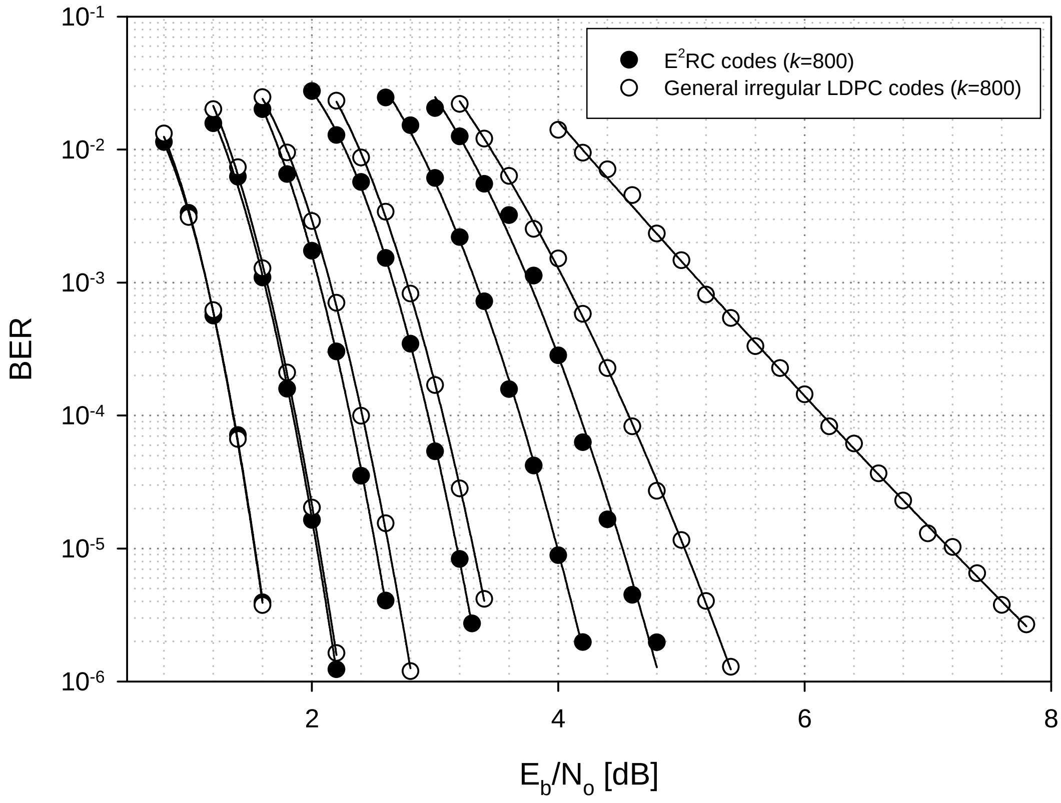


Figure 4.11 The puncturing BER performance comparison between E^{2}RC codes (filled circles) and general irregular LDPC codes (unfilled circles) with intentional puncturing. Rates are 0.4 (mother codes), 0.5, 0.6, 0.7, 0.8, and 0.85 from left to right.

In Figure 4.11 and Figure 4.12, the E^{2}RC codes show good performance over a wide range of rates 0.4~0.85. At a BER of 10^{-5} in Figure 4.11, the E^{2}RC codes outperform 1.0dB and 2.7dB of E_b/N_o than the general irregular LDPC codes at rate 0.8 and 0.85, respectively. The same tendency can be observed in FER performance in Figure 4.12. At a FER of 10^{-3} in Figure 4.12, the E^{2}RC codes outperform 1.0dB and 2.8dB of E_b/N_o than the general irregular LDPC codes at rate 0.8 and 0.85, respectively.

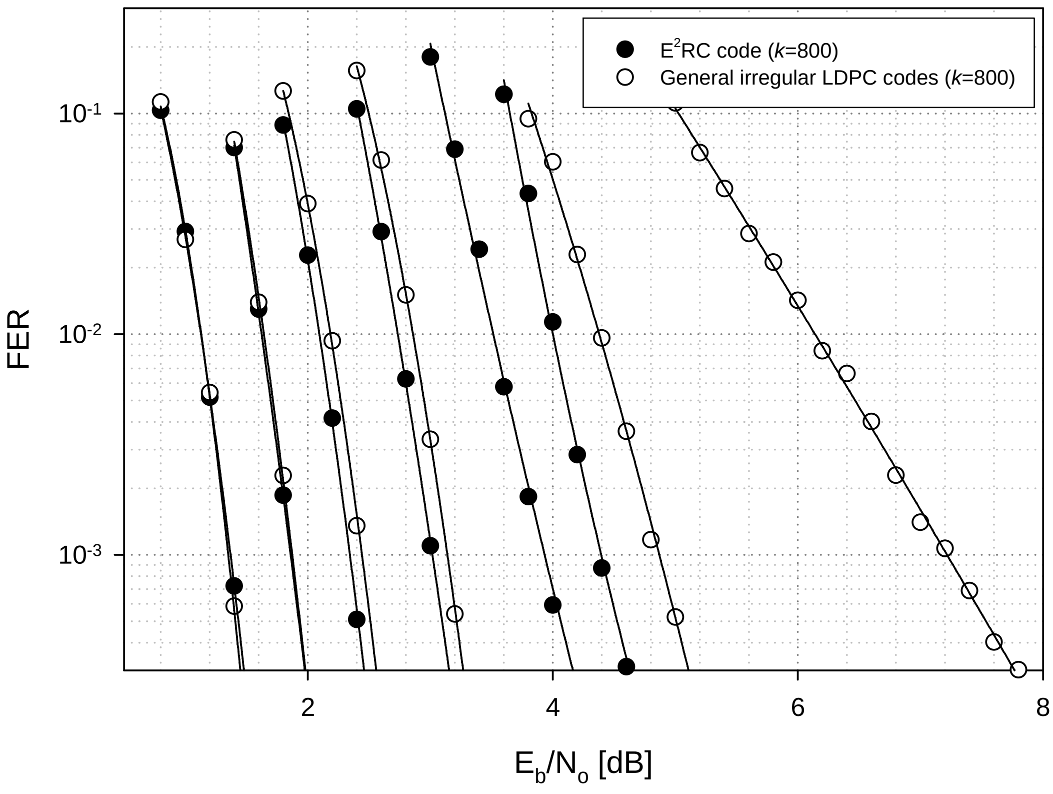


Figure 4.12 The puncturing FER performance comparison between E^2RC codes (filled circles) and general irregular LDPC codes (unfilled circles) with intentional puncturing. Rates are 0.4 (mother codes), 0.5, 0.6, 0.7, 0.8, and 0.85 from left to right.

We also compared the puncturing performance of E^2RC codes with the dedicated codes in Figure 4.13. The dedicated codes for each rate are generated with the degree distributions obtained from ones in the mother codes of rate 0.4. We assume that only the number of degree-2 nodes is reduced for higher rates. The dedicated codes are generated with PEG algorithm to increase their girth characteristics. From the simulations, the performance gaps between E^2RC codes and the dedicated codes for each rate at BER of 10^{-5} are less than 0.3dB.

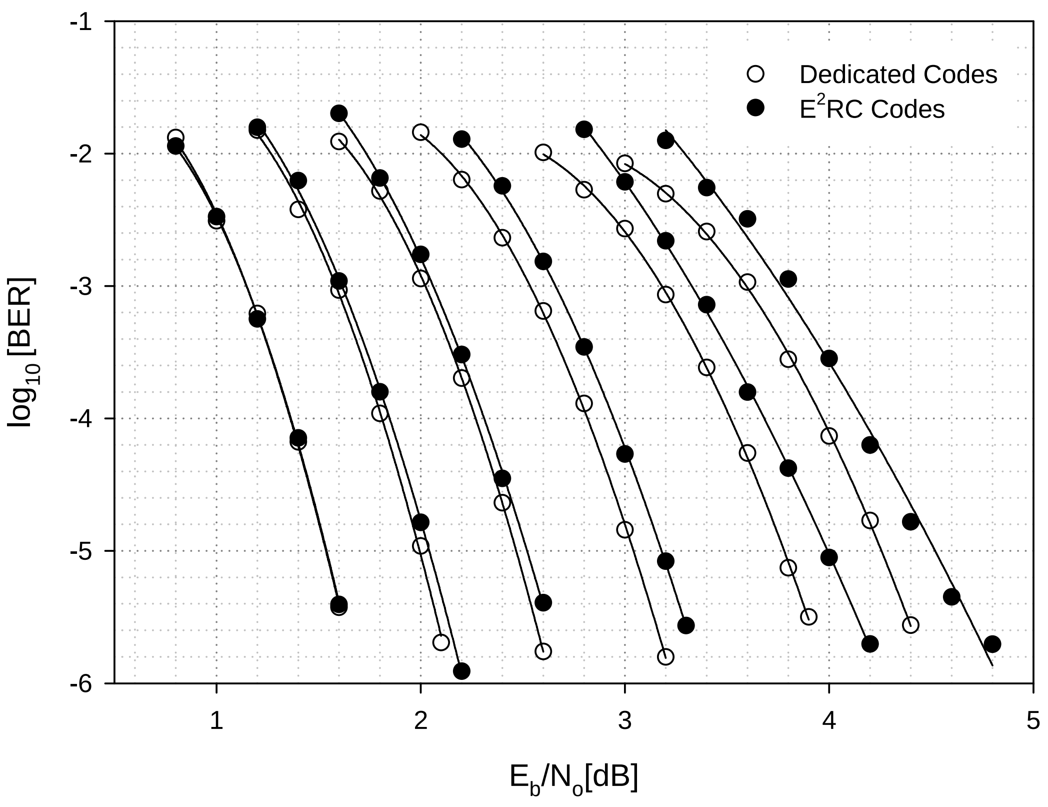


Figure 4.13 The BER performance comparison between puncturing of E^2RC codes (filled circles) and the dedicated LDPC codes (unfilled circles). Rates are 0.4 (mother codes), 0.5, 0.6, 0.7, 0.8, and 0.85 from left to right.

4.5 CONCLUSIONS

We have proposed a new class of codes, E^2RC codes, which has several strong points. First, the codes are efficiently encodable. We have presented shift-register implementation of encoder which has low-complexity. We also showed that a simple erasure decoder can also be used for the linear-time encoding of these codes. Thus, we can share a message-passing decoder for both encoding and decoding if it is applied to the transceiver systems which require an encoder/decoder pair. Second, we have shown that the nonsystematic parts of the parity-check matrix are cycle-free, which ensures good code characteristics. From simulations, the performance of the E^2RC codes (mother codes) is as good as that of eIRA codes and other irregular LDPC codes. Third, the E^2RC codes having systematic rate-compatible puncturing structure show better puncturing performance than other irregular LDPC codes and eIRA codes in all ranges of code rates. From simulations, the E^2RC codes show better puncturing performance as code rates increased. At rate of 0.8, the E^2RC codes outperform over 0.8dB of E_b/N_o than both eIRA codes and general irregular LDPC codes. Even when the best effort puncturing algorithm is applied to both eIRA codes and general irregular LDPC codes, the E^2RC codes show 1.5dB and 0.7dB of E_b/N_o than the best effort puncturing of the irregular LDPC codes and eIRA codes, respectively, at a BER of 10^{-5}. Finally, the E^2RC codes can provide good performance over a wide range of rates when they are designed low rate. We believe that these characteristics of E^2RC codes are more valuable when they are applied to IR Hybrid-ARQ systems. From the simulation, the E^2RC codes show good performance over a wide range of rates 0.4~0.85.

CHAPTER V

RATE-COMPATIBLE LDPC CODES FOR INCREMENTAL REDUNDANCY HYBRID ARQ SYSTEMS

Many wireless broadband systems require flexible and adaptive transmission techniques since they operate in the presence of time-varying channels. For these systems, incremental redundancy hybrid automatic repeat request (IR-HARQ) schemes are often used, whereby parity bits are sent in an incremental fashion depending on the quality of the time-varying channel [20]. Careful design of an adaptive forward error correction (FEC) code can improve data throughput in such systems. The incremental redundancy systems require the use of rate-compatible punctured codes (RCPC) [12]. These codes can be operated at different rates by using the same encoder-decoder pair. Depending on the rate requirement, an appropriate number of parity bits are sent by the transmitter. The receiver decodes by treating the parity bits that are not transmitted (called punctured bits) as erasures. In addition, the set of parity bits of a higher rate code forms a subset of the parity bits of a lower rate code. Thus, in an IR-HARQ system if the receiver fails to decode at a particular rate, it only needs to request additional parity bits from the transmitter.

IR-HARQ systems require good frame error rate (FER) performance, especially at high rate region to get good throughput performance. Since the proposed E^2RC codes show excellent puncturing performance at high rate region, we apply these codes to IR-HARQ systems.

5.1 INCREMENTAL REDUNDANCY HYBRID ARQ SYSTEMS

Previous work in IR-HARQ systems includes [21], [22], where the design of an ensemble of FEC codes is considered. The objective of IR-HARQ scheme is to improve the throughput by retransmitting the required fractional part of the parity bits rather than the whole information and parity bits when the previous transmission fails. The code combining process of our IR-HARQ scheme follows the Chase's rule [23], and details of the steps are as follows:

Code Combining Process for IR-HARQ Scheme

STEP 1: Making a frame with cyclic redundancy check (CRC)

STEP 2: LDPC encoding

STEP 3: Ordering and grouping the parity bits

STEP 4: Transmit the message and/or the required parity group

At the receiver end, the frame is reconstructed with the message and parity groups of the previous frame after receiving the parity group of the current frame. Then, the frame is decoded with LDPC decoder. We detect errors with the help of CRC detection. If errors occur in the current frame, send the negative acknowledgement (NACK) signal to the transmitter, and the transmitter sends the next required parity group. Otherwise, sends the acknowledgement (ACK) signal to the transmitter. If the transmitter receives an ACK signal, it stops sending the current frame and prepares the next frame.

An important performance measure of an IR-HARQ scheme is the throughput, which is defined as the ratio of the number of information bits k to the total number of bits that need to be transmitted for acceptance by the receiver. The throughput, η, is given by

$$\eta = \frac{k}{(1-F(1))(k+p_1)+\sum_{i=2}^{\infty}\left((1-F(i))\prod_{j=1}^{i-1}F(j)\right)\left(k+\sum_{j=1}^{i}p_j\right)},$$

where $F(i)$ is the probability of frame error at the *i*-th transmission, and p_j is the length of parity group at the *j*-th transmission. In the simulation, we consider *k*=1024, and p_j's are used as in Table 5.1.

5.2 System Model

As a system model with the IR-HARQ scheme, we consider an LDPC coded Vertical Bell Labs Layered Space-Time (V-BLAST) system [24], [25] in time-varying multiple antenna environments as depicted in Figure 5.1. The throughput and spectrum efficiency of this system can be improved by using LDPC codes, which are powerful capacity-approaching codes with feasible decoding complexity.

The original V-BLAST scheme [24] uses different channel codes at different layers. In this work, we only consider the single LDPC code as a channel code, and separate the output in parallel for each layer. We consider a 2×2 MIMO system, which has 2 transmit antennas and 2 receive antennas over a frequency flat Rayleigh fading channel. At the transmitter, the source data bits are encoded with an LDPC encoder, separated into two substreams, and mapped onto quadrature phase shift keying (QPSK) constellation points for each substream.

At the receiver side, the received signal can be expressed mathematically as $\boldsymbol{Y} = \boldsymbol{HX} + \boldsymbol{W}$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are complex input and output vectors, respectively, and $\boldsymbol{W}$ is a complex Gaussian noise with a variance σ^2. The complex 2×2 channel matrix is $\boldsymbol{H}$, which consists of channel coefficients of MIMO frequency-flat fading channels. At the receiver, perfect channel estimation is assumed, and the minimum mean square error (MMSE) detector is used for making a soft decision on the channel inputs. Then, each received soft bit stream is multiplexed into one stream and converted into a stream of log-likelihood ratio (LLR) values. These are used for soft decoding of a log-domain LDPC decoder.

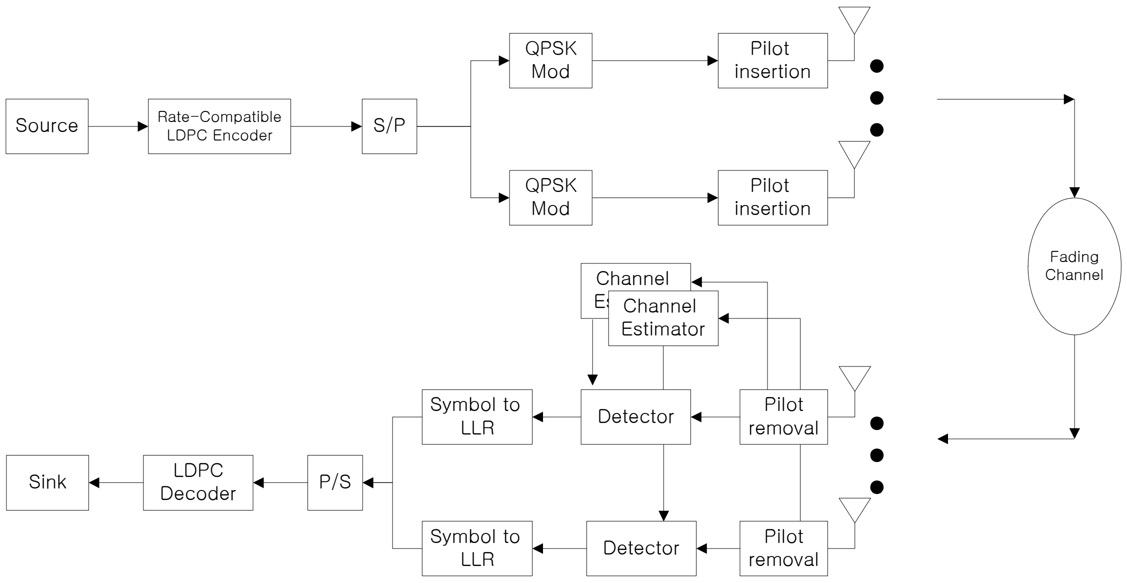


Figure 5.1 An LDPC coded V-BLAST MIMO system.

5.3 SIMULATION RESULTS

We consider a rate-1/2 LDPC code with code length of 2048. For IR-HARQ systems, IR parity bits are assigned as in Table 5.1, which are used as subset codes of an ensemble. We assume that the first transmission starts from rate of 0.94. We compare the FER and throughput performance of E^{2}RC codes with those of eIRA codes and general irregular LDPC codes.

Table 5.1 Ensemble of LDPC codes in the IR-HARQ simulation.

i	1	2	3	4	5	6	7	8	9
p_i	64	64	128	128	128	128	128	128	128
rate	0.94	0.89	0.80	0.73	0.67	0.62	0.57	0.53	0.50

When we generate eIRA codes and general LDPC codes, we try to keep the same degree distributions as those in [11] for rate-1/2 codes, which are optimized in additive white Gaussian noise (AWGN) channel:

$$\lambda(x) = 0.00015 + 0.30235x + 0.27132x^2 + 0.42618x^6$$
$$\rho(x) = 0.35555x^5 + 0.64445x^6.$$

For E^{2}RC codes, however, the actual degree distributions are slightly different to compensate the right degree of H_2.

$$\begin{aligned}\lambda(x) &= 0.00015 + 0.30235x + 0.27132x^2 + 0.42618x^6\\ \rho(x) &= 0.41140x^5 + 0.54617x^6 + 0.01892x^7 + 0.01064x^8\\ &\quad + 0.00592x^9 + 0.00325x^{10} + 0.00178x^{11} + 0.00193x^{12}.\end{aligned}$$

We apply the progressive edge growth (PEG) algorithm proposed in [16] to H_1 design of eIRA codes and E^{2}RC codes for having the better girth characteristics. First, we compare the puncturing performance between the proposed E^{2}RC codes and the eIRA codes. We apply the intentional puncturing algorithm proposed in [26], [27] to the eIRA codes, and compare the FER performance with E^2RC codes (see Figure 5.2). In this case, we face puncturing limitations. In fact, the puncturing algorithm in [26], [27] assigns 512 nodes as *1*-SR nodes and cannot find any more *k*-SR nodes ($k \geq 2$) if we try to maximize the number of *1*-SR nodes. To get a high rate in eIRA codes we puncture randomly after the puncturing limitation (512 *1*-SR nodes). This destroys the previous tree structure of *1*-SR nodes resulting in poor performance. The puncturing performance of the E^2RC codes is better than that of eIRA codes as the code rates are increased even though we apply the best effort puncturing algorithm to eIRA codes.

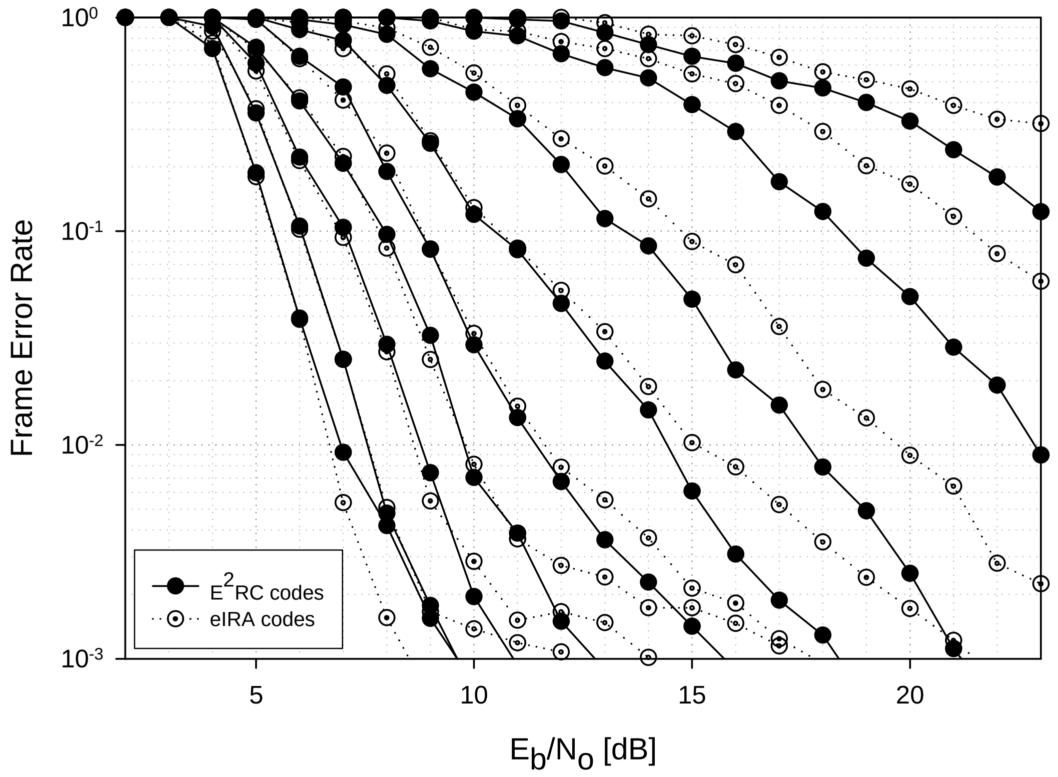


Figure 5.2 Performance comparison of rate-1/2 E^2RC codes (filled circle) and eIRA codes (unfilled circle). The message size is 1024 bits and curves are for rate=0.5, 0.53, 0.57, 0.62, 0.67, 0.73, 0.80, 0.89, 0.94 from left to right.

For throughput simulations, we consider FER of 10^{-3}, and simulate codes over the IR-HARQ scheme presented in section 5.2. We present the throughput performance comparison between E^2RC and eIRA codes in Figure 5.3. At the throughput of 0.8 in Figure 5.3, the E^2RC codes have 2dB gain over eIRA codes. This is because as mentioned earlier the throughput performance highly depends on the high puncturing rate.

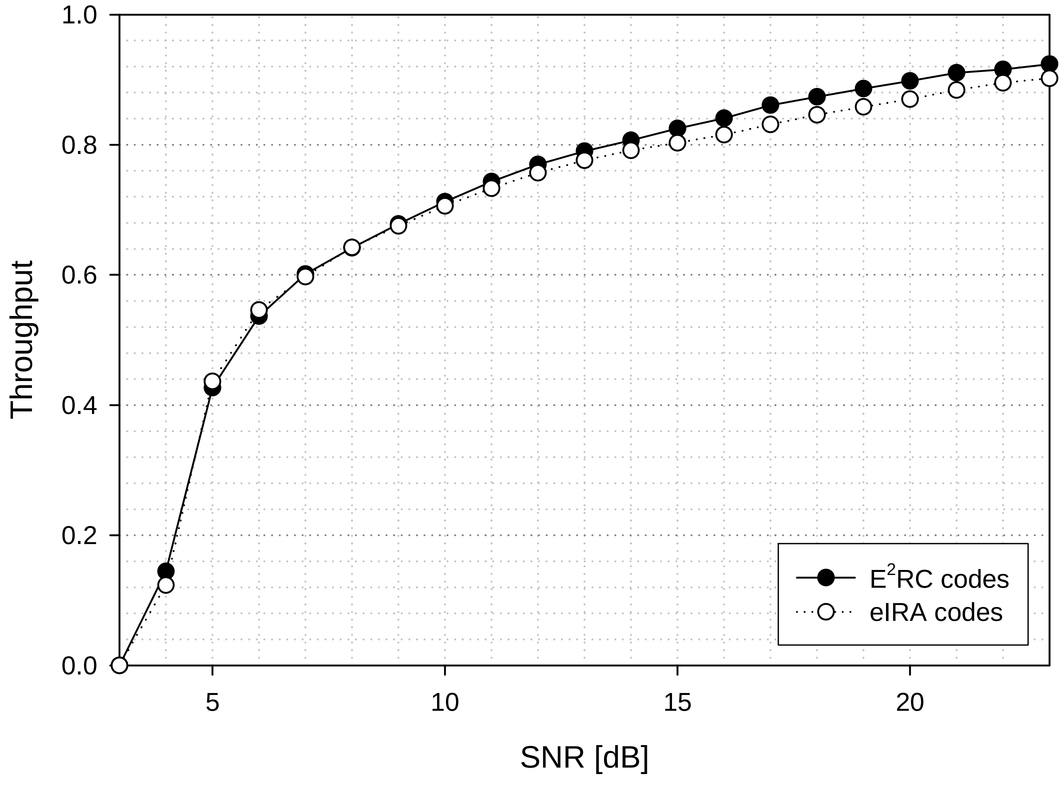


Figure 5.3 Throughput performance comparison of E^2RC codes (filled circle) and eIRA codes (unfilled circle). The message size is 1024 bits for both codes.

To compare the performance with the general irregular LDPC codes, we also apply the

PEG algorithm in [16] to generate the code. From a rate-1/2 mother code, we provide punctured codes of rate as following the Table 5.1 using the puncturing algorithm in [26], [27]. Through the simulation, we observe that the FER performance of E^{2}RC codes is slightly worse than or equal to general LDPC codes at lower code rate (rates 0.5~0.62), but outperforms them at higher code rate (rates 0.67~0.94). For this reason, the E^{2}RC codes show better throughput performance than the general irregular LDPC codes as shown in Figure 5.4. The E^{2}RC codes have a gain of about 2.2dB at the throughput of 0.8.

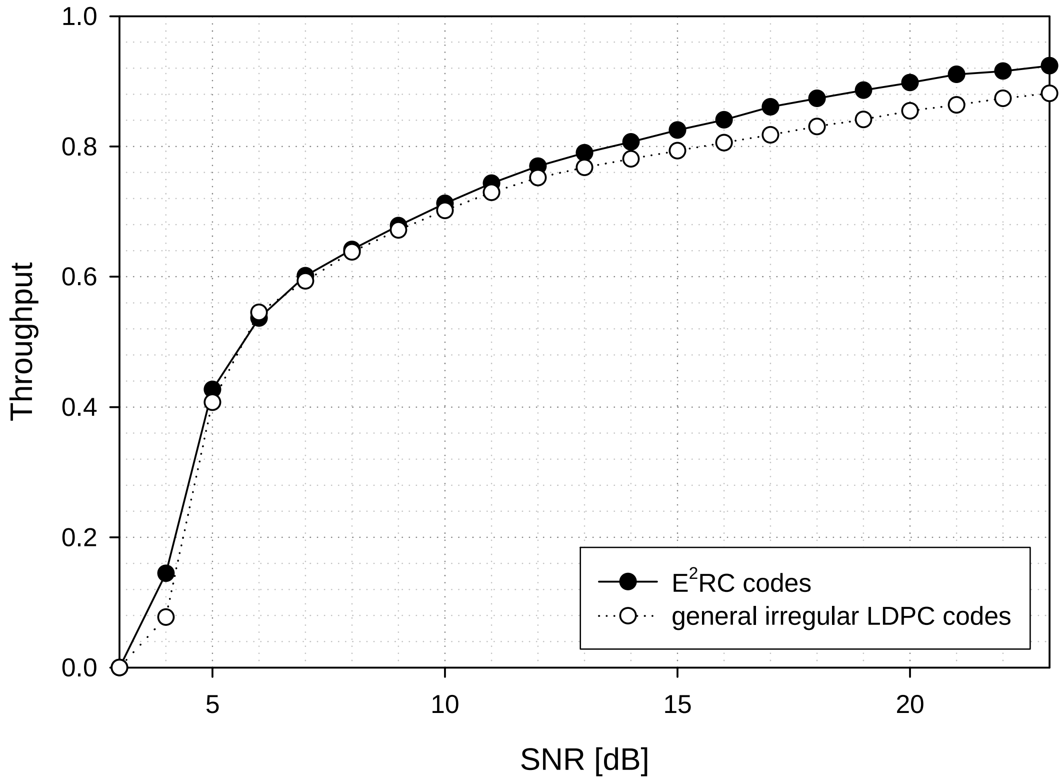


Figure 5.4 Throughput performance comparison of E^{2}RC codes (filled circle) and general irregular LDPC codes (unfilled circle). The message size is 1024 bits for both codes.

5.4 CONCLUSIONS

The E^2RC codes show better puncturing performance than other irregular LDPC codes and eIRA codes in all ranges of code rates, especially in high puncturing rate. These characteristics result in good threshold performance over time-varying channel in IR-HARQ systems. From simulations we observe that E^2RC codes outperform eIRA codes and the general irregular LDPC codes by 2dB and 2.2dB, respectively, at the throughput of 0.8.

CHAPTER VI

REMARKS

6.1 CONTRIBUTIONS

The focus of our research is designing practical rate-compatible LDPC codes which provide a wide range of rates. In this dissertation, we first propose a new rate-compatible puncturing method for LDPC codes at short block lengths. Based on the puncturing algorithm, we propose a design algorithm of rate-compatible LDPC codes, which also can be encoded efficiently. We summarize our contributions below:

1. We propose a new rate-compatible puncturing algorithm which consists of grouping and sorting algorithms. By introducing the concepts of recovery tree and recovery error probability, we mathematically show that the proposed algorithm has better puncturing performance. We also verify the better puncturing performance of the proposed algorithm by simulations.

2. We provide a tool for predicting puncturing performance of a random puncturing distribution by analyzing group distributions and the level of recoverability.

3. Based on the puncturing algorithm, we propose a new class of codes, called E^2RC codes, which can be efficiently encoded as well as can be punctured in a rate-compatible fashion. We provide a generalized construction algorithm of these codes. The proposed E^2RC codes show better puncturing performance than any other irregular LDPC codes over a wide range of code rates.

4. We develop an efficient encoding method, called sliding window encoding method,

with simple a shift-register circuit. Much like an encoder of cyclic codes, the sliding window encoder can be implemented with a division circuit. The only difference is coefficients of the division circuit are time-varying.

5. We provide a good channel coding scheme to IR-HARQ systems. In IR-HARQ systems, RCPC scheme is required, and good high-rate performance is needed to improve throughput. Since the proposed E^{2}RC codes show excellent puncturing performance especially at high rate region, we verify by simulation that the proposed E^{2}RC codes show better throughput performance in IR-HARQ systems over time-varying channels.

6.2 FUTURE WORK

We have concentrated on designing rate-compatible LDPC codes for short block length. For the E^{2}RC codes to have more practical meaning, a few interesting and challenging topics are remained.

1. We have proposed a construction algorithm of low-rate E^{2}RC codes and shown their design example. However, more research on low-rate code design is needed so that we can have a practically operating rate range. Implementing an encoder with shift-register circuits for low-rate design should be developed as well.

2. We have used degree distributions optimized only for mother codes over AWGN channels. However, we can find a degree distribution which is globally optimized for a wide range by considering puncturing rates. The basic assumption of E^{2}RC codes is puncturing degree-2 nodes in the parity parts. While we simulate the E^{2}RC codes over the AWGN channel, the punctured nodes can be considered as transmitting over the mixed channel, i.e., BEC plus AWGN channel. By applying this restriction to the density

evolution, we can find an optimal degree distribution for E^{2}RC codes. We believe that the optimal degree distributions will improve the overall performance of E^{2}RC codes.

3. QC LDPC codes have gained popularity since they have many advantages. Especially in integrated circuit (IC) decoder implementations, QC LDPC codes can have simple regular wiring and modular structure because of their cyclic symmetry. We should consider QC-type E^{2}RC codes to reduce the complexity in implementing decoder. Finding an optimal core matrix and parameters for QC-type E^{2}RC codes will be a good topic.

References

[1] R. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

[2] D. Mackay, and R. Neal, “Near Shannon limit performance of low-density parity-check codes,” *Electron. Lett.*, vol. 32, pp. 1645-1646, Aug. 1996.

[3] S. Chung, G. Forney Jr., T. Richardson, and R. L. Urbanke, “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit,” *Electron. Lett.,* vol. 5, pp. 58-60, Feb. 2001.

[4] IEEE P802.16e/D6, February 2005.

[5] 11-04-0889-03-000n-tgnsync-proposal-technical-specification.doc.

[6] 11-04-0886-06-000n-wwise-proposal-ht-spec.doc.

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

[8] 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.

[9] S.Lin, L. Chen, J. Xu and I. Djurdjevic, “Near Shannon limit quasi-cyclic low-density parity-check codes,” in *Proc. 2003 IEEE GLOBECOM Conf.* San Francisco, CA, Dec. 2003.

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

[11] M. Yang, W. E. Ryan, and Y. Li, “Design of Efficiently Encodable Moderate-Length High-Rate Irregular LDPC Codes,” *IEEE Trans. Comm.*, vol. 52, no. 4, pp. 564-571, Apr. 2004.

[12] J. Hagenauer, “Rate-compatible punctured convolutional codes (RCPC codes) and their applications,” *IEEE Trans. Comm.*, vol. 36, pp. 389-400, Apr. 1988.

[13] R. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inform. Theory*, vol. IT—27, pp. 533-547, Sep. 1981.

[14] J. Ha, J. Kim, S. W. McLaughlin, “Rate-Compatible Puncturing of Low-Density Parity-Check Codes,” *IEEE Trans. Inform Theory*, vol.50, no. 11, Nov. 2004.

[15] S. Chung, J. Richardson, and R. Urbanke, “Analysis of sum-product decoding of Low-density parity-check codes using a Gaussian approximation,” *IEEE Trans. Inform. Theory*, vol. IT-47, pp. 657-670, Feb. 2001.

[16] X. Hu, E. Elefitheriou, and D. M. Arnold, “Progressive edge-growth Tanner graphs,” in *Proc. IEEE GLOBECOM*, San Antonio, Texas, pp. 995-1001, Nov. 2001.

[17] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, “Selective Avoidance of Cycles in Irregular LDPC Code Construction,” *IEEE Trans. On Comm.*, vol. 52, no. 8, pp. 1242-1247, 2004.

[18] A. Ramamoorthy and R. D. Wesel, “Construction of Short Block Length Irregular Low-Density Parity-Check Codes,” in Proc. IEEE Int. Conf. on Comm., Paris, June 2004.

[19] W. Weng, A. Ramamoorthy, and R. D. Wesel, “Lowering the Error Floors of High-Rate LDPC Codes by Graph Conditioning,” VTC 2004, Los Angeles, California.

[20] R. H. Deng and H. Zhou, “An adaptive coding scheme with code combining for mobile radio systems,” *IEEE Trans. Vehicular Tech.*, vol. 42, no. 4, 1993.

[21] N. Varnica, E. Soljanin, and P. Whiting, “LDPC Code Ensembles for Incremental Redundancy Hybrid ARQ,” *Proc. Int. Symp. Inform. Theory,* pp. 995-999, Sept., 2005.

[22] S. Sesia, G. Caire, and G. Vivier, “Incremental Redundancy hybrid ARQ schemes based on low-density parity check codes,” *IEEE Transactions on Communications*, vol. 52, No. 8. , pp. 1311-1321, Aug. 2004.

[23] D. Chase, “ Code Combining- A Maximum-Likelihood Decoding Approach for Combining an Arbitrary Number of Noisy Packets,” *IEEE Transactions on Communications*, vol. 1, No. 5. , pp. 385-393, May. 1985.

[24] G. J. Foschini, “Layered space-time architecture for wireless communication in a fading environment when using multiple antennas,” *Bell Lab. Tech. Journ.*, vol.1, pp.41-59, 1996.

[25] J. Kim, G. L. Stuber, and Ye Li, “Robust V-BLAST MIMO-OFDM Channel Estimation in Time-Varying Channels Using Iterative Wiener Filters,” *IEEE Global Telecommunications conference* (*Globecom 2005*), St. Louis, 2005.

[26] J. Ha, J. Kim, and S. W. McLaughlin, “Puncturing for Finite Length Low-Density Parity-Check Codes,” in *Proc. Int. Symp. Inform. Theory,* Chicago, 2004.

[27] 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, Feb. 2006.

Vita

Jaehong Kim received his B.S. and M.S. degree in electronic communication engineering from Hanyang University, Korea, in 1995 and 1997 respectively.

Since 1997, he has been on the Technical Staff of Samsung Electronics, working on the Very Large Scale Integration (VLSI) implementation of channel coding for wireless channels. He is currently working toward the Ph.D. degree in electrical and computer engineering at Georgia Institute of Technology under supervision of Dr. Steven W. McLaughlin. His research interests include channel coding, digital communication systems and VLSI design.