Channel Coding for Highly Efficient Transmission in Wireless Local Area Network

Zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

von der Fakultät für Elektrotechnik und Informationstechnik
der
Technischen Universität Dortmund
genehmigte

Dissertation

von

Deny Hamdani

aus Bandung, Indonesien

Tag der mündlichen Prüfung: 21.03.2012

Hauptreferent: Prof. Dr.-Ing. Rüdiger Kays
Koreferent: Prof. Dr.-Ing. Jürgen Götze

Für meine Eltern und Deniras

ACKNOWLEDGMENT

First of all, I would like to express my sincerest gratitude to my *Doktorvater*, Prof. Dr.-Ing. Rüdiger Kays, for his inspiring guidance revealed during my doctoral education at the Technische Universitaet Dortmund. I am really grateful to him for all his fruitful help and for his inspiring insights that supported this work very much. I would like also to gratefully thank Prof. Dr.-Ing. Jürgen Götze for co-supervisory of this thesis and his recommendation for my DAAD scholarships. I thank also sincerely to examiners: Prof. Dr.-Ing. Christian Rehtanz and Prof. Dr.-Ing. Peter Krummrich, for their advisory opinion.

I wish to thank my roommate Dr.-Ing. Wolfgang Endemann for his charming friendship and many fruitful discussions as well as his support in finishing-touch of this dissertation. Many thanks also to my labmates for their help and friendship: Yasser Al-Nahlaoui, Christian Schilling, Beatriz Aznar, Dominik Lubeley, Thomas Jäger, Oliver Hundt, Dirk Seebeck, Harald Gebhard, Klaus Jotschulte, Heidrun Schettke, Helga Scheffler, Jürgen Marsch and also, *last but not least*, Stefan Nowak. They made Lehrstuhl fuer Kommunikationstechnik to become an incredible, pleasant environment for research and study.

I am grateful to all my friends in Ruhrgebiet, Aachen and Karlsruhe for making my stay in Germany a very pleasant one, especially, Supriyanto, Nurrakhman Yulianto, and Aulia Rahman, who have cordially helped me in finishing my study. I also wish to thank gratefully Mrs. Barbara Schwarz-Bergmann and DAAD for their cordial assistance and scholarships during my study in Germany. I am also grateful to Prof. Dr. Suwarno, Dr. Djoko Darwanto Gitokarsono and Prof. Dr. Ngapuli Irmea Sinisuka for their support, guidance and encouragement in my professional life. Special thanks also to Dr. Sony Suhandono and Dr. Endra Susila for valuable discussion and their grammatical review on technical English writing of this dissertation.

Finally, I would like to express special thanks to my family. I sincerely thank my parents, Mamah and Bapa, for their everlasting love, pray, encouragement, and support which led me to possibly earn such high educational degree. No words are sufficient to show my appreciation and respect for them. My wife, Dr.-med Ira Safitri, has always supported and encouraged me during my study. Her true love and cordial support have encouraged me to finish my study. I am especially grateful to her for taking such a loving care of our beloved sons Farhan, Fauzan and Fathur Denira. I also wish to thank Amah and Apah for their cordial support, and also my sister, all sisters- and brother-in-laws who helped us in many ways. I greatly appreciate their thoughtful support. My gratitude goes also to my late teacher Pak Muslim for teaching me about the blessed life.

ABSTRACT

Since their rediscovery, Low Density Parity Check (LDPC) codes sparked high interests due to their capacity-approaching performance achieved through their low decoding complexity. Therefore, they are considered as promising scheme for channel coding in future wireless application. However, they still constitute disadvantage in their high encoding complexity. The research on practical LDPC codes with good performance is quite challenging. In this direction their potential characteristics are explored with respect to the technical requirement of wireless local area network (WLAN).

This thesis is focused on three topics, which correspond to three major issues in the research of LDPC codes: code characterization with girth degree distribution, low encoding complexity with structured construction, and higher decoding convergence with two-stage decoding scheme.

In the first part of the thesis, a novel concept of girth degree is introduced. This concept combines the idea of the classical concept of girth with node degree. The proposed concept is used to characterize the codes and measure their performance. A simple tree-based search algorithm is applied to detect and count the girth degree. The proposed concept is more effective than the classical concept of girth in measuring the performance. It shows that the girth degree plays more significant role than the girth it self, in determining the code performance. Furthermore, the existence of short-four-cycles to some extent is not harmful to degrade the code performances.

The second part deals with a simple method for constructing a class of LDPC codes, which pose relative low encoding complexity but show good performance. The combination of the stair structure and the permutation matrices, which are constructed based on the proposed method, yields very simple implementation in encoding process within encoder. The resulting encoder can be implemented using relatively simple shift-register circuits. Their performance is comparable with that of irregular MacKay codes. In short code length, they outperform some well-established structured codes. The performance of the proposed codes is comparable with the optional LDPC codes for WLAN at higher code rates. However, the proposed codes are relatively suboptimal at lower code rate. Such performance is achieved by the proposed codes in lower encoding complexity

In the third part, a method for enhancing the decoding convergence for high coded modulation system is introduced. The two-stage decoding scheme is proposed to improve bit reliabilities in decoding process leading to reduced decoding iteration without performance losses. This is achieved by making use of the output from the first decoding stage as the additional input for the second decoding stage. The optimal combination of the maximal iteration of both decoding stages is capable of reducing the average iteration. This method shows its efficiency at the waterfall region of signal-to-noise-ratio.

KURZFASSUNG

Seit ihrer Wiederentdeckung haben die Low Density Parity Check (LDPC) Codes ein hohes Interesse erfahren, da sie mit niedrigem Aufwand für die Dekodierung fast die Kanalkapazität erreichen. Daher sind sie ein vielversprechendes Kanalcodierungsschema für zukünftige drahtlose Anwendungen. Sie weisen allerdings noch den Nachteil eines hohen Enkodierungsaufwandes auf. Die Einwicklung eines mit geringem Aufwand implementierbaren LDPC Codes mit guten Leistungen stellt noch eine große Herausforderung dar. Die Nutzbarkeit der potenziellen Eigenschaften von LDPC-Codes im Bezug auf die technischen Randbedingungen gerade bei drahtlosen lokalen Netzwerken (*Wireess Local Area Network* - WLAN) wirft dabei besonders interessante Fragestellungen auf.

Die vorliegende Dissertation konzentriert sich auf drei große Themen bezüglich der Erforschung von LDPC Codes, nämlich die Charakterisierung des Codes mittels Umfangsmaßverteilung (Girth Degree Distribution), den niedrigen Enkodierungsaufwand mittels strukturierter Codekonstruktion sowie die verbesserte Decodierungskonvergenz mittels eines Zwei-Phasen Dekodierungsverfahrens.

Im ersten Teil der Dissertation wird ein neues Konzept zur Beurteilung von Codes eingeführt. Es basiert auf der Umfangsmaßverteilung. Dieses Konzept kombiniert die Ideen des klassischen Konzeptes - basierend auf dem Umfang (G*irth*) - mit denen des Knotenmaßes (N*ode Degree*) und wird zur Charakterisierung und zur Abschätzung der Leistungsfähigkeit des Codes eingesetzt. Zur Erkennung und Berechnung des Umfangs wird ein einfacher, baumbasierter Suchalgorithmus eingeführt. Dieses Konzept ermöglicht eine effizientere Leistungsabschätzung als das der alleinigen Verwendung des Umfangs. Es wird gezeigt, dass das Umfangsmaß bei der Ermittlung der Leistung des Codes eine wesentlich größere Rolle spielt als der Umfang. Im Rahmen dieser Untersuchungen fällt als weiteres Ergebnis an, dass die Existenz von kurzen Schleifen der Länge 4 die Leistungsfähigkeit des Codes nicht beeinträchtigt.

Der zweite Teil der Dissertation beschäftigt sich mit einem einfachen Verfahren für die Konstruktion einer Gruppe von LDPC Codes, die bei einem relativ niedrigen Enkodierungsaufwand dennoch eine gute Leistung aufweist. Die Kombination einer Treppestruktur in Verbindung mit Permutationsmatrizen führt zu einer sehr einfachen Implementierung, ohne dass ein erheblicher Leistungsverlust auftritt. Der resultierende Enkodierer kann ausschließlich mit einer sehr einfachen Schaltung aus Schieberegistern implementiert werden. Die Leistungsfähigkeit des entstehenden Codes ist mit der des unregelmäßigen MacKay-Codes vergleichbar. In kurzer Kodelänge übertreffen sie sogar einige bekannte strukturierte Codes. Allerdings sind die vorgeschlagenen Codes suboptimal im Vergleich mit den optionalen LDPC Codes für WLAN, sofern niedrige Coderaten betrachtet werden. Sie erweisen sich aber als ebenbürtig bei höheren Coderaten. Diese Leistungsfähigkeit wird von den hier vorgeschlagenen Codes mit relativ niedrigem Enkodierungsaufwand erreicht.

Letztendlich wird im dritten Teil der Dissertation ist ein Verfahren zur Steigerung der Decodierungskonvergenz beim Einsatz von LDPC Codes in Kombination mit Modulationsverfahren hoher Wertigkeit vorgestellt. Das Zwei-Phasen Dekodierverfahren wird zur Verbesserung der Bit-Zuverlässigkeit im Dekodierungsprozess eingeführt. Dieses bewirkt eine Reduktion der benötigten Dekodierungsschritte ohne Leistungsverlust. Erreicht wird dies durch die Verwendung der Ergebnisse einer ersten Dekodierungsphase als erneute Eingabe für eine zweite Dekodierungsphase. Die optimale Kombination der durchzuführenden Iterationen beider Dekodierungsphasen kann die Anzahl der insgesamt benötigten Iteration im Durchschnitt reduzieren. Dieses Verfahren zeigt seine Wirksamkeit im Wasserfallbereich des Signal-Rausch-Verhältnisses.

CONTENTS

LIST OF FIGURES

Chapter 5

LIST OF TABLES

LIST OF PUBLICATIONS

1. Deny Hamdani, Wolfgang Endemann, Ruediger Kays. *A Class of LDPC Codes with Very Efficient Encoder*. International Conference on Electrical Engineering and Informatics (ICEEI2007), 17-19 June 2007, Bandung, Indonesia.

2. Deny Hamdani, Wolfgang Endemann, Ruediger Kays. *Measuring Performance of LDPC Codes with Girth Degree*. International Conference on Electrical Engineering and Informatics (ICEEI2007), 17-19 June 2007, Bandung, Indonesia.

3. Deny Hamdani, Wolfgang Endemann, Ruediger Kays. *Enhancing Performance of High-Order Modulation with LDPC Codes using Feedback Mechanism*. International Conference on Electrical Engineering and Informatics (ICEEI2007), 17-19 June 2007, Bandung, Indonesia.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Mobile computing has been of great interest within information and communication communities due to its tremendous growing resulting in a continuous influx of mobile devices supporting modern human life-style in the last decades. The indispensable growth in mobile computing is boosted by rapid development in device technology, affordable prices and increasing mobility requirement. Wireless local area networks (WLAN) in form of ad-hoc networks became the key technology in wireless networking of mobile computing. The high-speed WLAN standardized in IEEE 802.11 was introduced to cope with the development in area of mobile networking.

At the beginning of their development, the IEEE 802.11 wireless standards incorporated a well-known convolutional code and the Viterbi algorithm as error correcting channel code. At the time of the development of the standards this was the most practical solution considering cost, complexity, power consumption, and decoding latency. Unfortunately, convolutional codes and the Viterbi algorithm show inferior performance with respect to the theoretical capacity limits.

To cope with these problems, it would have been possible to introduce Turbo codes to the WLAN scheme. Although being capable of coming reasonably close to theoretical capacity, Turbo were not applied to WLAN due to their performance degradation for short blocks as well as their high decoding complexity. This fact led to the introduction of another family of forward-error-correcting codes known as Low Density Parity Check (LDPC) codes. By utilizing their advantageous very sparse matrix characteristics, a significant improvement over the current coding system can be reasonably realized. LDPC codes have been proven to outperform Turbo codes [RU03] and make reliable communication in vicinity of the Shannon limit possible [Chun01]. Although a scheme of LDPC codes has been officially incorporated in the recent WLAN Standard IEEE 802.11n [IEEE09], it is always of challenging interest to elaborate and explore this rediscovered coding technique for improving the performance of WLAN system.

This thesis deals with the investigation of LDPC codes as an efficient and reliable channel coding technique, which best fit to the wireless channel within WLAN systems. Some constraints concerning the environment and the application in WLAN shall be taken into account, such as limited processing capability, limited battery power, small physical form, demanded Quality of Service (QoS), and harmonious coexistence with other systems working in the neighbored bands. A significant drawback of LDPC codes in their encoding complexity has been the focus of this research to cope with aforementioned requirement.

1.2 Historical Review

1.2.1 Channel Coding

The history of channel coding could be dated back to 1948 as Claude Shannon published his seminal paper, *A Mathematical Theory of Communication* [Shan48]. He proved that reliable communication can be established over a noisy channel using channel codes provided that the information rate is not exceeding the so-called capacity of the channel. He derived the limit of the information rate over a noisy channel and presented channel codes with very long block length and optimal decoding that can achieve reliable communication. However, his approach is practically prohibitive due to expensive realization of both encoding and decoding.

The introduction of Shannon's noisy coding theorem has been sparking the research in field of coding theory. Practical capacity-achieving schemes found large interest within the coding community. Over four decades after Shannon's publication, none of a large number of proposed coding systems could approach Shannon's theoretical limit. The breakthrough just came in the early of 90's with the discovery of turbo codes [BGT93]. Thanks to their pseudorandom interleaver and iterative decoding algorithm, turbo codes could operate near Shannon's capacity limit with reasonable decoding complexity. The discovery of turbo codes sparked the research interest in the field of codes on graphs and iterative decoding. This fact led to the next breakthrough in 1995, that is, the rediscovery of LDPC codes [KN95], which was firstly invented in 1962 by Gallager [Gall62] and unfortunately, ignored thereafter. The main reason was that the state-of-the-art hardware technology considered the realization of LDPC codes as impractical at that time.

In principal, the construction techniques of channel codes are based on algebraic approach and convolutional approach. Further category of channel codes is derived from the combination of both approaches. Therefore, based on their construction technique, the channel codes can be classified into three categories [Li02]. The first category of channel codes is block codes based on algebraic approach. This codes includes Hamming codes introduced by Hamming in 1950 [Hamm50], BCH codes by Hocquenhem in 1959 [Hocq59] and independently by Bose and Ray-Chaudhuri in 1960 [BC60] and RS codes by Reed and Solomon in 1960 [RS60]. Those codes are practical in hardware implementation. However, there was no comfortable soft decoding algorithm. Furthermore, those codes are not flexible in code lengths.

The second category of channel codes is convolutional codes which were firstly introduced by Elias in 1955 [Elia55]. These codes are decoded by trellis decoding. They operate on serial data and are usually described by their code rate and constraint length. More powerful codes can be produced by longer constraint length, however, at cost of exponentially increased maximum likelihood decoding complexity. Unlike block codes, it is convenient to change code lengths of convolutional codes. The technique of puncturing keeps the flexibility of convolutional codes in code rate without extra complexity. They have efficient soft decoding algorithms, such as soft-output Viterbi algorithm [HH89] and a posteriori probability algorithm [BCJR74], which are of great advantages on fading channels. As an alternative, convolutional codes can be decoded by trellis decoding introduced by Ungerboeck [Unge82].

The third category of channel codes is the compound codes which combine block and convolutional codes. They use iterative decoding. The discovery of this code is dated back to 1966 as Forney [Forn66] concatenated an inner code and an outer code. This construction makes the codes to have error probability that decreases exponentially at rate less than capacity, while decoding complexity increases only algebraically. The development of these codes was accelerated by the discovery of turbo codes by Berrou, Glavieux, and Thitimajshima in 1993 [BGT93] and the rediscovery of LDPC codes by MacKay and Neal in 1995 [KN95]. Their impressive near-capacity performance lead to the introduction of other concatenated codes providing similar coding gains, such as parallel concatenated convolutional codes [BM96], serial concatenated convolutional codes [BMDP96] and hybrid concatenated convolutional codes [DP97]. Further codes includes turbo product codes [Elia54, CT94], regular/irregular repeat-accumulate codes [DJE98], and product accumulate codes [LNG01a]. These codes have common features including the application of a (random) interleaver and decoding techniques.

The milestones of history of coding theory can be summarized as follows

1948	Shannon's channel coding theorem [Shan48]
1950	Hamming codes [Hamm50]
1955	Convolutional codes [Elia55]
1959/60	BCH codes [Hocq59, BC60], RS codes [RS60]
1962	LDPC codes [Gall62], rediscovered in 1995 [KN95]
1982	Trellis-coded modulation [Unge82]
1993	Turbo codes [BGT93]

1.2.2 LDPC Coding

The history of LDPC coding began in the early 1960s when Gallager [Gall62, Gall63] introduced a channel coding scheme providing the structural basis for codes with near-shannon-limit performance. Gallager's codes applied iterative decoding based on message-passing decoding algorithms. However, such decoding demanded very intensive computation, which could not be supported by the existing hardware technology at that time. Hence, Gallager's codes were considered as impractical to be implemented as channel codes. It was about four decades later when LDPC codes were rediscovered in the mid 1990s as MacKay [KN95] realized the immense potential of the codes due to their near-Shannon limit performance.

Before the rediscovery of LDPC codes took place, a small number of researchers worked with Gallager's codes during the few decades after the first publication of Gallager's codes, including Zyablov and Pinsker [ZP75] and Margulis [Marg82]. In the early 1980s, Tanner [Tann81] provided a graphical representation of LDPC codes and other coding schemes. Tanner suggested the employment of a recursive approach for the construction of LDPC codes and presented a graph representation of the parity check matrix of LDPC codes. In the same work, Tanner also introduced the min-sum decoding algorithm.

In the early 1990s, channel coding research was sparked by the introduction of Turbo codes which have shown the impressive near-shannon limit performance with relative lower complexity. The application of iterative decoding in Turbo codes's scheme has accelerated the research of iterative decoding techniques leading to revisiting the work of Gallager. However, in their initial development, Turbo codes were not considered in connection with graphical representations. The iterative decoding techniques used for decoding turbo codes have been linked by McEliece *et al*. [EKC98] to the principles of belief propagation described by Pearl [Pear88], which was the basis of the decoding techniques proposed by Gallager.

The work of Tanner and its developments by Wiberg et al. [Wibe94, WKL95, Wibe96] provided the basis for the factor graph representation of codes in common use recently. These graphs have been further highlighted by Kschischang et al [KFL98]. The further usage of the graphs led Sipser and Spielman [SS96] to introduce LDPC codes whose parity check matrix is based on expander graphs. The common interest in the algorithms used for iterative decoding in the artificial intelligence community has led MacKay and Neal [KN95] in the mid 1990s to the rediscovery of LDPC codes.

Since then, the research of LDPC codes has been focusing on the improvements of the code performance. The general approach has been to modify the graph describing the code. The performance of LDPC codes in the case of very long block sizes (around 10^5 to 10^7 bits) has outperformed Turbo codes and approached the Shannon bound to within hundredths of decibel. Such techniques and their performance gains have been demonstrated by Sipser and Spielman [SS96], Richardson *et al*. [RSU00, RSU01] and Luby *et al*. [LMSSS97, LMSS98, LSMS01]. Chung *et al*. [CFRU01] showed that the threshold for a code rate 1/2 on the AWGN channel is within 0.0045 dB of the Shannon limit. Their simulation resulted within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} using a block length of 10^7 bits.

In coding research community, LDPC codes have been studied extensively in many aspects. Richardson *et al*. [RU01a] proposed the density evolution algorithm to calculate the asymptotic performance of a given LDPC code over AWGN channel. Chung *et al.* [Chun00, CRU01] simplified the complex density evolution algorithm using Gaussian Approximation. The bounds of code rate and performance of LDPC codes were studied by Burshtein *et al*. in [BKLM02, Burs02, MiB01]. MacKay *et al*. [KN97, KN99] simulated LDPC codes at high block length and illustrated that LDPC codes are capable of outperforming the turbo codes when communicating over AWGN channel.

LDPC codes using higher decoding Galois field were also proposed by Davey *et al.* [DK98, DK99, Dave99], Song *et al.* [SC03] and Nakamura *et al.* [NKS01]. Lentmaier [Lent97] proposed the generalized LDPC codes by replacing the rows in the parity check matrix of LDPC codes with a Hamming code and this also attracted the interest from Zhang *et al*. [ZP01a, ZP01b], Hirst *et al*. [HH02a, HH02b] and Boutros *et al*. [BPZ99]. LDPC codes using higher decoding Galois field were also proposed by Davey *et al.* [DK98, DK99, Dave99], Song *et al.* [SC03] and Nakamura *et al.* [NKS01]. Lentmaier [Lent97] proposed the generalized LDPC codes by replacing the rows in the parity check

matrix of LDPC codes with a Hamming code and this also attracted the interest from Zhang *et al*. [ZP01a, ZP01b], Hirst *et al*. [HH02a, HH02b] and Boutros *et al*. [BPZ99].

Motivated by their outstanding performance, LDPC codes have been studied in many other channels and employed with various modulation scheme. The performance of LDPC codes over Rayleigh fading channel [HSM01, LWN02], Rician channel [LZW04], Nakagami channel [MMØ02], Gilbert-Elliot channel [EKP03], Lorentzian channel [STC00], partial-response channel [LNG01b] and binary erasure channel [NF04] are evaluated. LDPC codes are elaborated for some communication scheme, such as OFDM scheme [FO01, SNG03], MIMO scheme [BKA04], CDMA scheme [SKP00]. The elaboration of LDPC codes within bandwidth efficient coded modulation schemes are studied in [EO01, HSMP03]. In addition to exploring the performance issues, some researchers are trying to reduce the complexity of encoding and decoding of LDPC codes such as Richardson *et al*. [RU01a, RU01b], Kou *et al*. [KLF00, KLF01], Spielman [Spiel96] and Pothier *et al*. [PBB99].

The research of LDPC codes leads to the design of their derivatives as Tornado [BLMR98], LT [Luby02], and Raptor [Shok03], which were protected by patents.

The milestones of LDPC coding research can be summarized as follows

1948	Shannon Limit [Shan48]
1962	Invention of LDPC codes [Gall62]
1975	Quantification of LDPC codes complexity [ZP75]
1981	Graph representation on LDPC codes parity check matrix [Tann81]
1995	Rediscovery of near Shannon-limit performance of LDPC codes [KN95]
1998	Irregular LDPC codes [LMSS98]
2001	World record-breaking LDPC codes performance [CFRU01]

1.3. Thesis Organization

The thesis is structured as follows.

In Chapter 1 the introduction to this work, including motivation of the research, historical review of LDPC codes, and the organization of the thesis, is given.

In Chapter 2 some basic concepts for theoretical basis of the research, including transmission system, channel coding, LDPC codes, coded modulation and wireless LAN, are discussed.

In Chapter 3 the thesis introduces the concept of girth degree that can be considered an extension of the concept of girth in the theory of LDPC codes. The concept is used to characterize LDPC codes and also to explore the role of girth in determining the performance of LDPC codes. Girth detection and girth degree counting algorithms are introduced. Some LDPC codes are characterized and evaluated using this concept. It is of great interest that the existence of short-four-cycles is realized within some optional LDPC codes in Standard IEEE 802.11n, which in fact suffer from no performance degradation.

In Chapter 4 the thesis discusses a simple method for code construction using the stair structure to reduce encoding complexity, which is one of the drawbacks of LDPC codes.

The proposed codes are based on permutation submatrices in cascade and lattice forms concatenated with the identity submatrix to provide lower encoding complexity. The permutation submatrices are determined by shift parameters, which are randomly generated by a permutation process. The shift parameters are examined by two proposed methods, i.e. slope method and girth degree method. These methods will realize the possible existence of short-four-cycles that may degrade the code performance. Their performance is investigated under different parameter of LDPC codes, such as weight column, code rate, code length, etc. For benchmarking, their BER performance is put into comparison with some well-established, such as MacKay codes and optional LDPC codes in the IEEE 802.11n Standard. In the hardware implementation, their encoder/decoder derived from parity-check equation can be implemented in simple shift-registers circuits. One of the most interesting results is that the girth degree plays more significant role in code performance than the girth does.

In Chapter 5 the thesis is focused on the simple method for improving decoding process. A two-stage decoding using feedback mechanism is proposed. The method is investigated within a high-order coded modulation system with different parameters including types of LDPC codes, QAM bit-constellation level, and code length under different maximal number of decoding iteration.

In Chapter 6 the result of this work is summarized. The contribution of this work is cited. Some interesting topics are mentioned for advancing this work in the future.

CHAPTER 2

BASIC THEORY

2.1 Communication System

In principal, a typical communication system consists of three major components that are transmitter, receiver, and channel. The transmitter translates the information bits into the signals that can be effectively transmitted over the channel. The channel, which is mainly the physical medium over which the communication process takes place, passes the signals to the receiver. The signals are then translated by the receiver to retrieve the information.

Conceptually, the basic elements of a communication system are illustrated with the general block diagram shown in Figure 2.1 [LC04]. As the input of the communication system, the source information in the transmitter may be either analogue (time-continuous) or discrete (symbol sequences). Symbol sequences can be produced from analogue signals via sampling and the analogue-to-digital conversion process. Prior to transmission, the source encoder removes redundancy from the source information. The symbol sequences are assumed as a stream of statistically independent, equally likely discrete symbols (binary digits) with a constant rate of R_s bits per second.

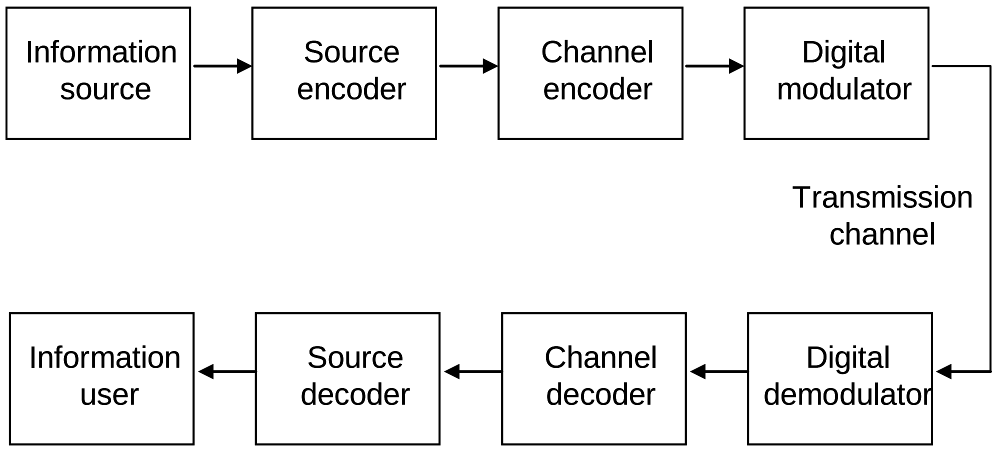


Figure 2.1 General block diagram of coded systems for digital communications

Furthermore, the symbol sequences are encoded by the channel encoder for error correcting purpose before modulation. The encoder adds redundancy to information bits and produces data at a higher rate R_c. In case of encoding of a block code, the encoder accepts information in successive k-bit blocks and for each k bits generates a block of n bits, called a codeword, where $n \geq k$. Thus the encoder outputs bits at a rate $R_c = R_s \cdot (n/k)$.

The modulator matches the encoder output to the transmission channel. The modulator modulates binary or M-ary encoded symbols in form of waveforms appropriate physically to the transmission channel. The modulator simply converts a binary digit or the M possible encoded symbols to two or M possible waveform of equal duration.

The transmission channel passes the modulated waveform to the point just prior to demodulation. During the transmission process noise is added to the transmitted waveform. In received signals noise constitutes the most significant factor constraining the performance of a communication system. Noise limits the ability of the demodulator to reliably distinguish one modulated waveform from another, thereby producing errors in the demodulator output. Thermal noise is always present in electrical circuitry. This noise is broadband and steady in its power level, and has Gaussian amplitude statistics. The resulting errors tend to occur independently from one signaling interval to the next. Other impairments are impulsive noise and multipath interference.

At the receiver, the demodulator receives the noisy waveform. It computes and delivers estimates of the transmitted data in each separate transmission symbol interval and produces a number or a set of numbers that represent an estimate of a transmitted symbol. Since the received waveforms are noisy, the symbol decisions are subject to be erroneous.

The channel decoder receives the demodulated outputs and converts them into symbol decisions that reproduce, as accurately as possible, the data that was encoded by the channel encoder. For block coding, the decoder accepts consecutive blocks of n demodulator outputs and produces k decoded symbols for each block. The decoder attempts to make definite symbol decisions that operate on hard-decision or soft-decision. Error probability at the output of the decoder provides an important measure of the overall performance of the communication system. The symbol-error-rate, which is the average rate of occurrence of symbol errors, taken as a fraction of the total number of symbols received over a long period of time become the measure of the quality of a communication system.

Finally, the source decoder accepts the sequence of symbols from the channel decoder. In accordance with the encoding method used in the transmitter it reconstructs the information originally generated by the analogue source.

2.2 Channel Coding

The basic idea of forward error correcting in channel coding theory is to add redundancy to the transmitted information in order to cope with channel errors. For a binary block code the channel encoder divides the information sequence into message blocks of k information bits each [LC04]. A message block is represented by the binary k-tuple $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$. A total of 2^k different possible messages are available. Each message $\mathbf{u}$ is transformed independently into n-tuple $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ of codeword. The set of 2^k codewords of length n is called an (n, k) block code and the ratio $R = k/n$ is called code rate. The encoder is memoryless because each message is encoded independently, which means the n-bit codeword depends only on the corresponding k-bit message. Hence, the encoder is implemented with a combinational logic circuit.

For a convolutional code, the n-k redundant bits are also added to each message to form a codeword. The channel encoder also transforms k-bit blocks of the information sequence **u** into a code sequence **v** of n-symbol blocks. This transformation proceeds with code rate $R = k/n$. In contrast to block code, the encoder of convolutional code is not memoryless because each encoded block depends not only on the corresponding k-bit message block at the same time unit but also on m previous message blocks. Hence, the encoder is said to have a memory order of m. The encoder is implemented with a sequential logic circuit.

In order to describe the mechanism of channel coding, a simplified model of a channel coding system is depicted in Figure 2.2. The encoder transforms the information sequences **U** into a discrete encoded sequence **X** called as codeword [LC04] and transmits it over the noisy channel. The decoder receives the noisy sequence **Y**. The sequence **Y** is a non-deterministic function of the channel input **X**, and the decoder tries to reconstruct the input string $\bar{\mathbf{U}}$ based on the knowledge of **Y**.

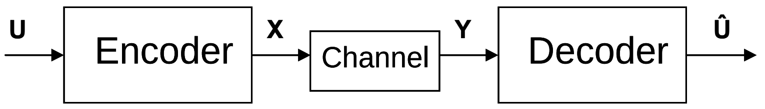


Figure 2.2 A simplified model of channel coding system

A model for the channel is necessary to analyze such a system. In this model, it is assumed that the output sequence **Y** of the channel has the same length as the input sequence **X**, and depends on **X** via a conditional probability density function (pdf) $p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$. For special cases, further consideration must be made.

In case of a memoryless channel, the channel output at any time instant depends only on the input at that time instant, i.e., if $\mathbf{y} = y_1 y_2 \ldots y_n$ and $\mathbf{x} = x_1 x_2 \ldots x_n$, then $p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} p_{Y|X}(y_i \mid x_i)$. In this case, the channel is completely described by its input and output bits, and the conditional pdf $p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$ for one time instant.

In this kind of channel, the output of the channel will be independent and identically distributed copies of some random variable Y as the input to the channel is generated by independent and identically distributed copies of a random variable X. Hence, the information between random variable X and Y, $I(X;Y)$, is a function of the pdf of X.

For analyzing the system, the additive white Gaussian noise (AWGN) channel is commonly applied as the channel model, which is parameterized by a non-negative real number σ. The channel output **Y** is given by **X** + **N**, where **X** is the channel input and **N** is a Gaussian random variable with mean 0 and variance σ^2. The conditional pdf $p_{Y|X}(y|x)$ is therefore a Gaussian pdf with mean μ and variance σ^2.

In case of a binary symmetric input channel, the channel is parameterized with a parameter p (i.e. the crossover probability of the channel) and output binary alphabet {0,1}.

By supposing that the input distribution of a binary-input channel conditioned on the knowledge of the received value y, that is, a-posteriori probabilities $\Pr(X = 0|Y = y)$ and $\Pr(X = 1|Y = y)$, their ratio can be computed using Bayes' rule

$$\frac{\Pr(X=0\,|\,Y=y)}{\Pr(X=1\,|\,Y=y)} = \frac{p_{Y|X}(y\,|\,0)}{p_{Y|X}(y\,|\,1)} \frac{\Pr(X=0)}{\Pr(X=1)} \tag{2.1}$$

Hence, the ratio $\frac{p_{Y|X}(y\,|\,0)}{p_{Y|X}(y\,|\,1)}$ is sufficient for the estimation of the input to the channel. This quantity corresponding to the output y of a binary-input channel is called likelihood ratio. Its logarithm $\log\frac{p_{Y|X}(y\,|\,0)}{p_{Y|X}(y\,|\,1)}$ is called the log-likelihood ratio (LLR).

2.3 LDPC Codes

2.3.1 Description

The ensembles of LDPC codes are defined by the set of parity-check matrices in a non-systematic form with a small number of ones in each column and in each row (see Figure 2.3). Since each row of the parity check matrix **H** is a single parity check, an LDPC code can be viewed as the concatenation of (n-k) single parity check codes in parallel, where n is the codeword length and k is the information bit length [Li02].

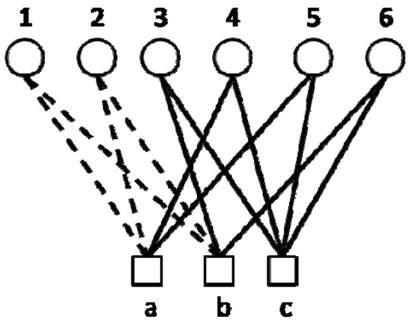
$$\begin{array}{c|c|}
 & n \\
\hline
 & 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0 \\
\hline
 & 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1 \\
\hline
 & 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\
\hline
n\text{-}k & \begin{array}{c}\cdots\cdots\cdots\cdots\\ \cdots\cdots\cdots\cdots\\ \cdots\cdots\cdots\cdots\end{array} \\
\hline
 & 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\
\hline
 & 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0 \\
\hline
\end{array}$$

Figure 2.3 Parity check matrix **H** of an LDPC code

Alternatively, LDPC codes can be represented by a Tanner graph [Tann81]. This bipartite graph utilizes variable (or bit) nodes (illustrated as circle) and check nodes (box) to represent the columns and rows of the parity check matrix **H** and uses inter-connecting edges to represent the relations between variable and check nodes (see Figure 2.4). The variable nodes represent elements of the codeword as variables, while the check nodes represent constraints among these variables. For binary linear codes, the variable nodes represent binary variables, and a check node assures that the binary sum of all its neighbors is zero.

$$\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
a & 1 & 1 & 0 & 1 & 1 & 0 \\
b & 1 & 1 & 1 & 0 & 0 & 1 \\
c & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
\end{array}$$

(a) Parity check matrix



(b) Bipartite graph

Figure 2.4 Representation of LDPC codes

Based on the fraction of non-zero elements in the columns of their parity-check matrix, LDPC codes are categorized as regular and irregular. The regularity of LDPC codes corresponds to the uniformity of their column weights and row weight as well. The parameter of the regularity includes the column weight (corresponds to the variable node degree in the bipartite graph) γ and the row weight (the check node degree) ρ . An LDPC code is considered to be (γ,ρ)-regular if all column have weight γ and all rows have weightρ. Meanwhile, irregular LDPC codes are not constrained to uniform row or column weights. Their node degree profiles, $\gamma(i)$ and $\rho(i)$, are usually used to describe the distributions of row weights and column weights, respectively. Another important parameter of LDPC codes is the girth. It is defined as the length of the shortest cycle in the bipartite graph. A cycle of length of four as the girth is highlighted by a dashed line in Figure 2.4.b.

The regular LDPC codes have a minimum distance (averaged over the ensembles of the code) which increases linearly with the block size, provided the column weight is at least 3 [Gall63]. This implies that the codes have excellent asymptotic performance in the code length. With carefully designed row and column degree profiles, irregular LDPC codes can outperform regular LDPC codes [LMSS98, RSU01, CRU01].

The decoding of an LDPC code takes place using an iterative message-passing algorithm. The algorithm is working based on belief propagation operating on the graph representation of the code. The message-passing algorithm is also known as the sum-product algorithm, where (soft) messages are passed from bit nodes to check nodes and check nodes to bit nodes, vice-versa, in an iterative way until the message is correctly decoded before the maximal number of iterations is exceeded or failed otherwise. The girth plays an important role in determining the performance of LDPC codes because the performance of the message-passing algorithm is adversely affected by the existence of short cycles. The convergence of a message-passing decoder is optimal in the ideal case where the bipartite graph is tree-like or cycle-free. However, in practice, the existence of short cycles within LDPC codes is difficult to be avoided in constructing the codes. Nevertheless, the suboptimal decoder performs quite satisfactorily.

2.3.2 Code Construction

Based on their construction, LDPC codes can be divided into two categories: random codes and structured codes. Random LDPC codes are constructed by computer search based on certain design rules or graph structures, such as girth and node degree distribution, while structured LDPC codes are constructed based on algebraic and combinatorial methods.

In general, random LDPC codes with long code length perform closer to the Shannon limit than their equivalent structured LDPC codes. However, the random LDPC codes do not have sufficient structure to provide simple encoding and hence, is highly complex to implement in hardware. The wiring structure of the encoder is very difficult. On the other hand, structured codes are advantageous in encoding over random codes. LDPC codes in cyclic or quasi-cyclic structure can be encoded with simple shift registers. Their encoding complexity is linearly proportional to the number of parity-check bits for serial coding and to the code length for parallel encoding. For practical lengths, in fact, LDPC codes with well-designed structure are capable of equally performing well, even outperforming LDPC codes with equivalent random structure.

The construction of random codes is mostly based upon random edge connections. A common construction technique is to build the parity-check matrix using random permutation matrices as described in [KWD99]. The random construction prohibits the creation of cycles of length four by maintaining column weight and row weight to be uniform. Gallager [Gall62] showed that the ensemble of such codes poses excellent distance properties provided column weight of at least three and row weight is greater than column weight. MacKay *et al*. [KN95] provided algorithms to generate semi-randomly codes with sparse parity check matrices.

Richardson *et al*. [RSU01, RU01] and Luby *et al*. [LMSSS97, LMSS98, LSMS01] define ensembles of irregular LDPC codes, in which column-weight and row-weight are not uniform. These codes are parameterized by the polynomial of node degree distribution, which can be optimized via density evolution algorithm. By relaxing the regularity code constraint and allowing irregular degree sequences the performance of Gallager's original LDPC construction could be improved. Several years before, MacKay *et al*. [KWD99] have explored some examples of irregular construction. For finding good irregular distributions for LDPC codes Feldman and Karger [FK02] applied linear programming techniques to develop heuristics. Chung *et al*. [CFRU01] developed algorithms to compute the capacity of randomly constructed LDPC codes and used these algorithms to provide optimized irregular codes, which can achieve asymptotically 0.0045 dB of the Shannon limit and simulated performance within 0.04dB of the Shannon bound for a block length of 10^7 at a bit error rate of 10^{-6}.

On the other hand, there have been some works to create some classes of LDPC codes in structured form based on the algebraic approach. Margulis [Marg82] applied explicit graph constructions to Gallager codes. Rosenthal *et al*. [RV00] extended this technique using Ramanujan graphs to build LDPC codes. Bond *et al*. [BHS00] proposed circulant matrices to build LDPC codes. Some types of LDPC codes employing the circulant matrices are the following: array codes [Fan00], repeat accumulate codes [DJE98] and their extension, irregular repeat accumulate codes [JKM00] and extended irregular repeat accumulate codes [YLR02], then combinatorial codes [KD00, VM04, JW01], finite geometry codes [KLF01], RS-based LDPC codes [DXGL03], convolutional LDPC codes [FZ99, BPZ99], product accumulate codes [LRG04]. Ping *et al*. [PLP99] have shown how the parity-check matrix can be built from separate deterministic and random components with only a small loss in code performance. Tang *et al*. [TXKLG04] proposed hybrid method to construct LDPC codes by combining several base constructions, i.e. Gallager codes, finite geometries codes, and circulant codes.

In constructing LDPC codes there have been some issues that present the requirement as follows

- Computation: the amount of computation in check nodes and variable nodes also increases, if column weight and row weight increase, respectively. The critical intensive computation is in the check nodes.
- Efficiency: the memory size is determined by the size of parity check matrix. The small parity check matrix can save the required chip area necessary for storing the memory.
- Flexibility: the rate adjustment is required to support many different code rates, and also support H-ARQ mechanism. A single parity check matrix capable of performing rate scalability can save memory size

In practice, the recent code design is targeting LDPC codes having excellent performance and providing flexibility and low encoding/decoding complexity. Both parameters are always trade-off in design. Their features can be summarized as follows

- Structured block: the matrix is composed of the same style of permutation subblocks, which allows structured decoding leading to the reduction of decoder implementation
- Low-complexity encoding: the encoding is performed in a structured, recursive manner, without degrading the performance with multiple weight-1 columns
- Designed to match the OFDMA scheme: the matrices are provided in different base matrices for exact code rate for different block sizes.
- Compact representation: The shift values for each block size are derived from the largest block size of the code rate.
- Simplified decoder architecture: each base matrix has 24 columns, which perform better, and provide a larger parallelization. The same number of columns between code rates minimizes the number of different expansion factors that must be supported.

Some companies have proposed their practical schemes of LDPC codes for broadband wireless access, such as Intel, Motorola, Nortel, etc. Intel [XJ04] proposed efficient encoding technique using the triangular matrix structure and adapted the parity-check matrix of extended irregular repeat accumulate. With this structure, the parity bits in codewords are generated by a differential encoder. Motorola [BCB04] proposed the modification of Intel's LDPC codes by introducing the modified triangle matrix so that the recursive encoding is enabled. Samsung [Kim04] applied Richardson's efficient encoding technique and adapted the permutation matrix for its parity-check matrix. Its idea is to make use of the lower-triangular submatrix and make dimension of a submatrix as small as possible. Nortel [PSJ04] adopted the encoding technique of π-rotation LDPC codes [EC01]. The parity-check matrix is composed of several sub-matrices, which slopes are rotated at certain degree.

An informal LDPC group has been working on the goal of achieving consensus on an optional advanced LDPC code for the OFDMA PHY [IEEE05]. A downselection process was conducted to get best codes among eight candidates: Intel, LG, Motorola, Nokia, Nortel, Runcom, Samsung, and TI. The candidates shared many desirable features. After redesigning process, Motorola with its enhanced proposal came out as the winner [IEEE05].

2.3.3 Encoding

It is well-known that the significant drawback of LDPC codes is their high encoding complexity, which scales as a quadratic of the code block length. This fact has inspired research into the area of structuring the codes in order to reduce encoding complexity, which is practically of importance.

Basically, the encoding of LDPC codes proceeds using two methods, i.e. the encoding with generator matrix derived from its corresponding parity check matrix and the encoding with reduced encoding complexity. The latter method is proposed to reduce the complexity of the first method, which is originally applicable for encoding linear block code.

In encoding with a generator matrix, the process of encoding an LDPC codes requires the generator matrix **G**, which corresponds to the parity check matrix **H** by the expression $\mathbf{GH}^T = 0$. The parity check matrix **H** is transformed via Gauss-Jordan elimination and columns reordering into a systematic form $\mathbf{H}_S = [\mathbf{P}^T\ \mathbf{I}]$ where $\mathbf{P}^T$ is a transposed parity-check submatrix and **I** is an identity matrix. From this, a generator matrix $\mathbf{G} = [\mathbf{I}\ \mathbf{P}]$ is produced. The encoding is performed by multiplying **G** and information vector **u** via the relation $\mathbf{x} = \mathbf{G}^T\mathbf{u}$. The number of operations required to calculate each element of the transmitted **x** is k multiplications and k-1 additions. Hence, the total number of operations required by this process is $n \times k$ multiplications and $n \times (k\text{-}1)$ additions. This method reveals the drawback of LDPC codes, in which the encoding complexity scales as a quadratic of the block length. The other drawback is the parity-check submatrix **P** is generally not sparse. These lead to long latency and high storage requirement.

To overcome such drawbacks, the encoding complexity is reduced to a linearity-approaching level. The modification of parity-check matrices takes place to introduce more rapid encoding and smaller memory requirement in the encoder. The principal of encoding LDPC codes using a parity check matrix having a structure which approaches lower-triangular form is discussed by MacKay et al. [Kay99]. This method allows most of the parity bits to be calculated in linear-time via back-substitution using sparse operations. Their simulation shows that such codes have a performance which approximates the regular random LDPC codes.

Some researchers published their LDPC codes with reduced encoding complexity. Sipser and Spielman proposed a class of linear-time encodable and decidable expander codes in [SS96]. Their approach involves using cascaded graphs to recursively build irregular codes based upon simple subcodes at each stage of the graph. Error correcting codes are built by recursively combining weaker error-reducing codes. Luby et al. built codes related to these structures in [LMSSS97], exhibiting performance very close to that of turbo codes.

MacKay and Neal [KN95] and Richardson and Urbanke [RU01b] proposed the triangular parity-check matrices with reduced encoding complexity as depicted in Figure 2.5. The complexity of their parity-check matrices can be reduced by keeping the dimension of submatrix $g \times g$ as low as possible. They define the *gap g* to be the difference between the number of rows in the approximate upper-triangular form and the number of rows in **H**. MacKay's parity-check matrix requires the static memory proportional to be m^2 for encoding. This method allows the complexity to be lower compared to m $(n\text{-}m)$ for storing the generator matrix. The reduction of encoding complexity is greater as the ratio g/m decreases.

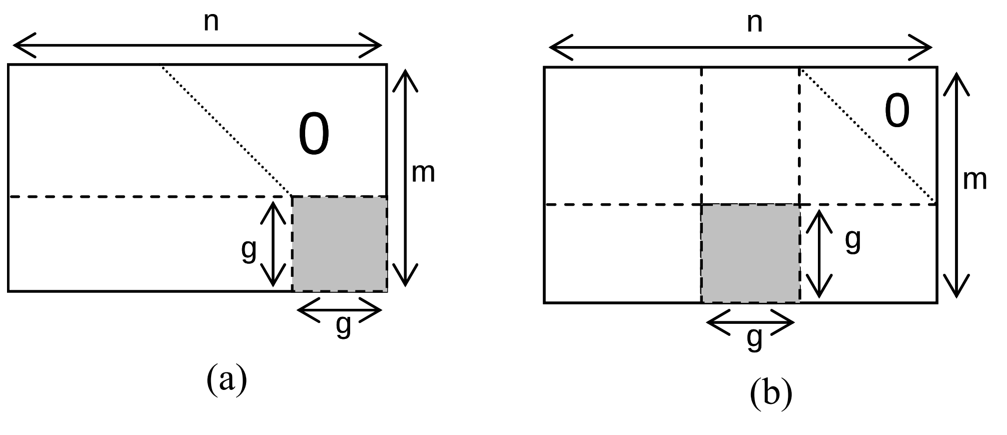


Figure 2.5 Triangular parity-check matrix of MacKay (a) and Richardson (b)

Meanwhile, Richardson and Urbanke have shown that the parity-check matrix for most LDPC codes can be manipulated such that the coefficient of the quadratic component in the encoding complexity can be made very small. Their method involves using a pre-processing stage, which rearranges **H** so that it is in approximate triangular form. Encoding complexity is then shown to be proportional to $n + g^2$. The idea of Richardson resulted in the reduction of the encoding complexity to $0.017^2\ n^2$ for (n,3,6) code. The algorithms assumed that the rows of **H** are linearly independent. The codes could be linear-timely encoded if dimension g is equal to their root square of m.

This recursive encoding technique using a triangular structure is widely recognized and found many applications. In wireless local area networks, Intel [XJ04] is among the first to introduce this technique, which is modified by Motorola [BCB04] to enable recursive encoding. Meanwhile, Samsung [Kim04] makes use of the parity-check structure of Richardson, which outperforms those of Intel and of Motorola. The idea of Richardson also became the basis for the structure of the optional LDPC codes for the existing WLAN IEEE 802.11n standard [IEEE09].

Some authors proposed a concatenated form of the parity check matrix to ease the encoding process. Vasic *et al*. [VKK02] and Echard *et al*. [EC01] constructed their parity check matrices in serial form. Oenning *et al*. [OM01] proposed them in parallel form by making use of recursive characteristics of its concatenated submatrices. Haley *et al*. [HGB02] proposed iterative encoding techniques using the Jacobi method, which adopted the principle of message passing in decoding.

2.3.4 Decoding

The decoding of LDPC codes begins with calculating the most probable transmitted codeword based on the received message from the transmission channel. The decoding proceeds iteratively using belief-propagation based decoding schemes. This iterative decoding is of the success keys that allow the performance of LDPC codes to approach the Shannon-limit.

The first iterative decoding approach for decoding of LDPC codes was introduced by the inventor of LDPC codes, Gallager [Gall62]. He proposed some decoding algorithms. Among them are the bit flipping algorithm and the algorithm using message passing of conditional probabilities. The message-passing algorithm belongs to the class of sum-product algorithms and is widely known as belief-propagation. However, the powerful performance of this iterative decoding algorithm in approaching Shannon-limit was realized in [KN97] after over three decades due to the hardware constraints.

In the last fifteen years, a lot of papers proposing improvements of the decoding schemes of LDPC codes or their alternatives were published. Some of them are worth to be mentioned. Fossorier and Lin [FL95] introduced an ordered statistics soft decoder for linear block codes. The complexity reduction of LDPC decoding algorithm was introduced in [EMD01]. Belief-propagation algorithms were improved and/or optimized in [YHB04]. Several variations of bit flipping decoding algorithms can be found in [MF02]. A linear programming decoding approach for LDPC codes was proposed in [FWK05].

In addition to the sum-product algorithm [Gall63] and its modified version, i.e. min-sum algorithm [Wibe96], several approaches to LDPC decoding are proposed, such as bit-flipping [SS96], maximum-likelihood [SM02], linear-programming [FK02], Turbo code [MS03], soft bit [HGS04] and hybrid [ZB04].

The decoding algorithm for LDPC codes is based on the idea of belief propagation [Pear88]. Upon this idea, each node acts as an independent entity and communicates with other nodes via a belief message passed through the edges. The message sent by a variable node to a check node is its estimate of its own value. The message sent by a check node to a variable node is its estimate of the value of the variable nodes. The update rules at the nodes are essentially maximum a-posteriori estimators, given that the incoming messages along the different edges are independent. Again, in order not to form short cycles in the computation tree, the output along any edge is based only on the input from the other edges.

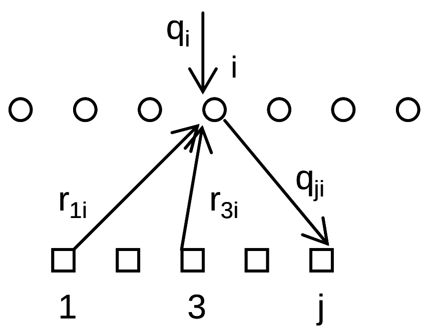
For each edge of the underlying bipartite graph, the decoding algorithm iteratively updates two types of message *q* and *r* as log-likelihood ratio of posteriori probability [Kay99, RSU01]. The message *q* is sent from the variable node to the check node along a connecting edge *e*. It is expressed as

$$q = \log \frac{p(x=0\,|\,y)}{p(x=1\,|\,y)}, \tag{2.2}$$

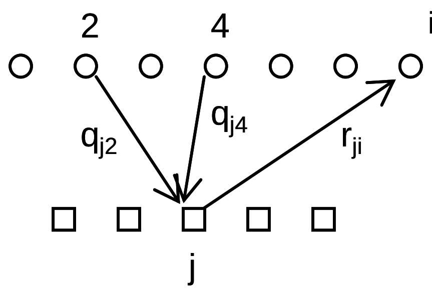
where *x* denotes the value of the bit node, and *y* denotes the message coming from the corresponding channel. The quantity *r* is sent from the check node to the variable node along an edge *e*, which is expressed as

$$r = \log \frac{p(x=0\,|\,v)}{p(x=1\,|\,v)} \tag{2.3}$$

where *v* denotes the messages coming from the edges connected to the check node, other than edge *e*. During the message updating, the incoming message via edge *e* is excluded in determining the outgoing message via edge *e*. This message-passing algorithm is illustrated in Figure 2.6.



(a) Message passing from variable node *i* to check node *j*



(b) Message passing from check node *j* to variable node *i*

Figure 2.6 Message passing decoder of LDPC codes

The condition of the parity-check matrix plays a significant role in supporting the efficacy of the message-passing decoding algorithm. The exact LLR of all bits could be produced after l iterations, if the bipartite graph defined by the parity-check matrix contains no loops of length up to $2l$ [Kay99]. If the graph is assumed to be loop-free, the decoding algorithm can be directly analyzed because the incoming messages to every node are independent. Also, the decoder performance will converge to that of a corresponding loop-free graph as the codeword length approaches infinity for almost all the graphs in a code ensemble (λ,ρ) and almost all inputs according to the general concentration theorem of [RU01a].

To figure out the iterative decoding algorithm analytically, as an example, a regular (j, k) LDPC code is considered. Based on the facts that a) LDPC codes are linear block codes, b) both the channel and the decoding algorithm are symmetric [RU01a], it is assumed that the all-zero codeword is sent. Using BPSK modulation, the fraction of incorrect messages that is passed is equal to the fraction of messages with negative signs. The fraction of incorrect messages is averaged over all the bits of a codeword and passed during iteration of the decoding algorithm.

Also, the message passed from the variable node to the check node is

$$q = q_0 + \sum_{i=1, i\neq j}^{j} r_j \tag{2.4}$$

where q_0 is the initial message conditioned on the channel output, and r_i , $i = 1,\ldots, j$, are the incoming LLR messages from all the incident edges, other than edge from variable node j.

The message r passed from the check node to the bit node is

$$\tanh\frac{r}{2} = \prod_{i=1, i\neq k}^{k} \tanh\frac{q_i}{2} \tag{2.5}$$

where q_i , $i = 1, \ldots , k$ are the incoming LLR messages from the neighbor edges, other than edge from check node k. Logarithmic operations on both sides in the equation (2.5) changes the product into a pair of summations.

$$\operatorname{sgn}(r_i) = \sum_{i=1, i\neq k}^{k} \operatorname{sgn}(q_i) \tag{2.6}$$

and

$$\log\left|\tanh\frac{r}{2}\right| = \sum_{i=1, i\neq k}^{k} \log\left|\tanh\frac{q_i}{2}\right| \tag{2.7}$$

where the sign function *sgn(x)* = 1 if $x \geq 0$, and *sgn(x)* = -1 otherwise.

If the code symbols mapped into the signal point *w* = (1-2*x*), the sampled matched filter output *y* has the conditional probability density function (pdf)

$$p(y \mid w) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y-w)^2}{2\sigma^2}) \tag{2.8}$$

where $\sigma^2 = \frac{1}{2R(E_b / N_0)}$ is the variance of the noise, and *R* is the code rate.

Due to the symmetric characteristics, *Pr*(*x* = 0) = *Pr*(*x* = 1) = 1/2, the message observed from the channel can be expressed as

$$q_0 = \log \frac{P(x=0 \mid y)}{P(x=1 \mid y)} = \frac{2}{\sigma^2} y \tag{2.9}$$

In case of multilevel coding, the mapping device maps a binary vector $\boldsymbol{c} = (c_0, \ldots, c_{t-1})$ onto a complex symbol $a \in A$, where *A* is the signal set and $|A| = 2^t$, where *t* is number of symbol bits. At the receiver side, the channel output ***y*** will be taken into account to produce LLR of symbol bits as decoder input. Figure 2.7 depicts Gray-mapped 16-QAM signal constellation.

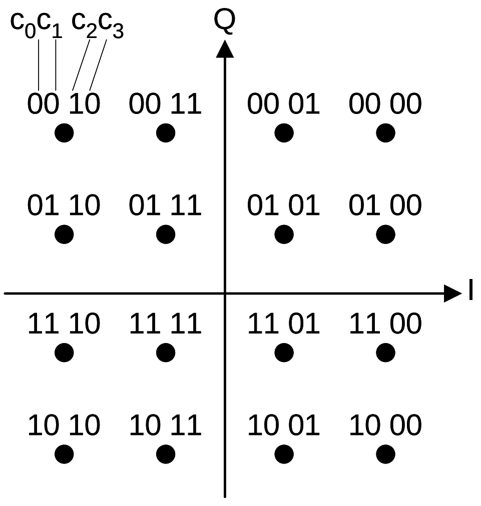


Figure 2.7 Signal constellation for 16-QAM with Gray labeling

The LLR of bits c_i, i = 1, …, t of the received symbol, $L(c_i)$, can be defined as [RMC03]:

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid y)}{\Pr(c_i = 1 \mid y)}\right) \tag{2.10}$$

The optimum decision rule is to decide, $c_i = 1$ if $L(c_i) \leq 0$, and 0 otherwise. Define two set partitions, $S_i^{(0)}$ and $S_i^{(1)}$, such that $S_i^{(0)}$ comprises symbols with $c_i = 0$ and $S_i^{(1)}$ comprises symbols with $c_i = 1$ in the signal constellation. Then, the equation (2.10) can be derived to

$$L(c_i) = \log\left(\frac{\sum_{\alpha \in S_i^{(0)}} \Pr(a = \alpha \mid y)}{\sum_{\gamma \in S_i^{(1)}} \Pr(a = \gamma \mid y)}\right) \tag{2.11}$$

Assume that all the symbols are equally likely and that fading is independent of the transmitted symbols. Using Bayes' rule, the LLR of symbol bit c_i is

$$L(c_i) = \log\left(\frac{\sum_{\alpha \in S_i^{(0)}} f_{y \mid a}\,(y \mid a = \alpha)}{\sum_{\gamma \in S_i^{(1)}} f_{y \mid a}\,(y \mid a = \gamma)}\right) \tag{2.12}$$

with

$$f_{y \mid a}\,(y \mid a = x) = \frac{1}{2\pi\sigma^2} \cdot \exp(-\frac{\left\|y - x\right\|^2}{2\sigma^2}); \quad x = \alpha\,,\gamma \tag{2.13}$$

where $f_{y \mid a}\,(y \mid a = x)$ is the Gaussian conditional probability density function for y as complex-valued and the variance σ^2 is assumed known.

2.4 Wireless LAN

Wireless Local Area Network (WLAN) is widely used as networking technology that provides wideband wireless connectivity between devices as well as access to the core network or the wider internet. It is currently applicable in corporate, public, and home environments. It also supports the mobility for users to move within a local coverage area. Thanks to its wireless technology, WLAN offers an easy way to configure computer networks without the need for cable installation. WLAN is also considered as a potential high-speed extension to cellular radio access networks.

Currently, WLAN technology operating in the 2.4 GHz industrial, scientific, and medical (ISM) band is widely used. WLAN, which is initially based on the first official "legacy" standard IEEE 802.11, provided an internationally accepted standard for WLAN with data rates up to 2 Mb/s. A higher-rate extension to this standard, 802.11b, achieves data rates of up to 11 Mb/s, operating within the same band. However, the relative high interference level in the operating band and the increasing demand for higher bit rates have pushed the standard to seize the new dedicated spectrum at 5 GHz band with the capability to support multiple transmission modes with raw data rates of up to 54 Mb/s. In North America, the FCC has allocated 300 MHz of spectrum U-NII band with the extended standard 802.11a developed by IEEE [IEEE99]. In Europe, the ERC have designated a total of 455 MHz of spectrum with the HIPERLAN/2 standard developed by ETSI [ETSI99]. Probably most widespread is IEEE 802.11g, which more or less shifted the 802.11a-Version into the 2.4 GHz ISM band.

In order to cope with frequency selective fading respectively typical indoor echo situations, OFDM is applied as reliable transmission scheme in the PHY layers. As shown in Table 2.1 the PHY layer modes with different coding and modulation schemes are selected by a link adaptation scheme [Jush99].

Table 2.1 Mode-dependent parameters

Mode	Modula-tion	Coding Rate R	Nominal bit rate (Mbps)	Coded bits per sub-carrier	Coded bits per OFDM symbol	Data bits per OFDM symbol
1	BPSK	1/2	6	1	48	24
2	BPSK	3/4	9	1	48	36
3	QPSK	1/2	12	2	96	48
4	QPSK	3/4	18	2	96	72
5	16-QAM	1/2	24	4	192	96
6	16-QAM	3/4	36	4	192	144
7	64-QAM	3/4	54	6	288	216
8	64-QAM	2/3	48	6	288	192

Referring to the diagram in Figure 2.8, the transmission mechanism in PHY layer of the IEEE 802.11a at transmitter side can be highlighted as follows. The input data in the form of packet data unit (PDU) frame come into a scrambler preventing long runs of 1s and 0s with a pseudorandom sequence of length 127. The scrambled data is then processed by a convolutional encoder. The encoder consists of a 1/2-rate mother code and subsequent puncturing. The puncturing schemes support code rates 1/2, 2/3, and 3/4. In case of 16-QAM, the coded data is interleaved in order to prevent error bursts from being input to the convolutional decoding process in the receiver. The interleaved data is subsequently mapped to data symbols according to BPSK, QPSK, 16-QAM, or 64-QAM constellation. IFFT technique is applied for OFDM modulation, whose numerical value is presented in Table 2.2. The transmitter output including 48 data symbols and 4 pilots is transmitted in parallel in the form of one OFDM symbol.

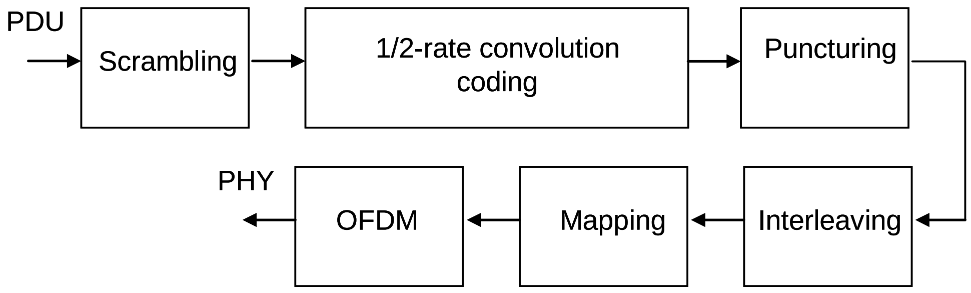


Figure 2.8 IEEE 802.11a transmitter PHY layer

Table 2.2 OFDM parameters

Parameter	Value
Sampling rate (f_s)	20 MHz
Useful symbol duration (T_u)	3,2 μs
Guard interval duration (T_g)	0,8 μs
Total symbol duration (T_{Total})	4.0 μs
Number of data subcarriers (N_D)	48
Number of pilot subcarriers (N_P)	4
FFT size	64
Subcarrier spacing (Δ_f)	0.3125 MHz
Total bandwidth (B)	16.875 MHz

The 802.11a use different training sequences in the preamble. The training symbols used for channel estimation are the same, but the sequences provided for time and frequency synchronization are different. Decoding of the convolutional code is typically implemented by means of a Viterbi decoder.

In the IEEE 802.11a, different channel models have been produced to represent these different environments [MS98]. Table 2.3 summarizes the channel models, which are wideband, with Rayleigh or Rician modeled tapped delay lines. Each tap suffers independent Rayleigh or Rician (in the case of channel model D) fading with a mean corresponding to an exponentially decaying average power delay profile.

Table 2.3 Channel models

Name	RMS delay spread	Characteristic	Environment
A	50 ns	Rayleigh	Office NLOS
B	100 ns	Rayleigh	NLOS
C	150 ns	Rayleigh	NLOS
D	140 ns	Rice	LOS
E	250 ns	Rayleigh	NLOS

In order to enhance network throughput over the widely used standard IEEE 802.11a and 802.11g, in 2009, the IEEE 802.11n [IEEE09] was introduced. In this standard, LDPC codes are considered as optional high-performance error-correcting code and the complementary for the so far widely used convolutional code. LDPC codes shall support the WLAN application with three different codeword lengths: 648, 1296 and 1944 bits, transmitted at four coding rates: 1/2, 2/3, 3/4 and 5/6. The rate-dependent parameters to be supported by LDPC codes are presented in Table 2.4.

Table 2.4 Rate-dependent parameter of LDPC codes in the IEEE 802.11n Standard

Coding rate (R)	Information block length (K in bits)	Codeword block length (N in bits)
1/2	972	1944
1/2	648	1296
1/2	324	648
2/3	1296	1944
2/3	864	1296
2/3	432	648
3/4	1458	1944
3/4	972	1296
3/4	486	648
5/6	1620	1944
5/6	1080	1296
5/6	540	648

The IEEE 802.11n standard accommodates the significant increase in the maximum raw data rate from 54 Mbps to 600 Mbps. This can be achieved by employing multiple-input multiple-output (MIMO) and 40-MHz channels to the PHY (physical layer) and frame aggregation to the MAC layer.

Various modulation schemes and coding rates are defined with a Modulation and Coding Scheme (MCS) index value as listed in Table 2.4. This index constitutes the high-throughput physical layer parameters that consists of modulation order: BPSK, QPSK, 16-QAM and 64-QAM) and forward error correction coding rate: 1/2, 2/3, 3/4, and 5/6.

The maximum raw data rate of 600 Mbps is achieved with the maximum of four spatial streams using a 40 MHz-wide channel with 400ns Guard-Interval and 64-QAM modulation at code rate 5/6 The relationships between the variable allowing the maximum data rate is presented in Table 2.5.

The encoding of LDPC codes proceeds systematically. The encoder encodes an information block of size k into a codeword of size n by adding n-k parity bits. The block length of a codeword is selected via the encoding process of packet data unit.

The parity-check matrix of LDPC codes is composed of square subblocks or submatrices. The submatrices can be either cyclic-permutations of the identity matrix (so-called permutation submatrice) or null submatrix. Such structures of parity-check matrices make the complexity of the encoding low. Some prototypes of parity-check matrices of all optional WLAN codes for all coding rates and code rates are defined in the IEEE 802.11n Standard as presented in Appendix.

Table 2.5 MCS parameters in the IEEE 802.11n Standard

MCS Index	Spatial Streams	Modulation Type	Coding Rate	Data Rate Mbps			
				20-MHz channel		40-MHz channel	
				800ns GI	400ns GI	800ns GI	400ns GI
0	1	BPSK	1/2	6.50	7.20	13.50	15.00
1	1	QPSK	1/2	13.00	14.40	27.00	30.00
2	1	QPSK	3/4	19.50	21.70	40.50	45.00
3	1	16-QAM	1/2	26.00	28.90	54.00	60.00
4	1	16-QAM	3/4	39.00	43.30	81.00	90.00
5	1	64-QAM	2/3	52.00	57.80	108.00	120.00
6	1	64-QAM	3/4	58.50	65.00	121.50	135.00
7	1	64-QAM	5/6	65.00	72.20	135.00	150.00
8	2	BPSK	1/2	13.00	14.40	27.00	30.00
9	2	QPSK	1/2	26.00	28.90	54.00	60.00
10	2	QPSK	3/4	39.00	43.30	81.00	90.00
11	2	16-QAM	1/2	52.00	57.80	108.00	120.00
12	2	16-QAM	3/4	78.00	86.70	162.00	180.00
13	2	64-QAM	2/3	104.00	115.60	216.00	240.00
14	2	64-QAM	3/4	117.00	130.00	243.00	270.00
15	2	64-QAM	5/6	130.00	144.40	270.00	300.00
16	3	BPSK	1/2	19.50	21.70	40.50	45.00
17	3	QPSK	1/2	39.00	43.30	81.00	90.00
18	3	QPSK	3/4	58.50	65.00	121.50	135.00
19	3	16-QAM	1/2	78.00	86.70	162.00	180.00
20	3	16-QAM	3/4	117.00	130.70	243.00	270.00
21	3	64-QAM	2/3	156.00	173.30	324.00	360.00
22	3	64-QAM	3/4	175.50	195.00	364.50	405.00
23	3	64-QAM	5/6	195.00	216.70	405.00	450.00
24	4	BPSK	1/2	26.00	28.90	54.00	60.00
25	4	QPSK	1/2	52.00	57.80	108.00	120.00
26	4	QPSK	3/4	78.00	86.70	162.00	180.00
27	4	16-QAM	1/2	104.00	115.60	216.00	240.00
28	4	16-QAM	3/4	156.00	173.30	324.00	360.00
29	4	64-QAM	2/3	208.00	231.10	432.00	480.00
30	4	64-QAM	3/4	234.00	260.00	486.00	540.00
31	4	64-QAM	5/6	260.00	288.90	540.00	600.00

CHAPTER 3

GIRTH DEGREE FOR CODE EVALUATION

3.1 Background

In the theory of LDPC codes, girth is an important parameter for LDPC codes, which is strongly related to code performance. Girth can be defined as the size of the smallest cycle of the bipartite graph. This parameter is considered as a determining factor in the decoding process and therefore, used as one of the criteria in code construction. It has to be kept as large as possible to hold the message-independence assumption, on which the iterative message passing on a code graph is based. This assumption is valid as long as the number of decoding iterations is smaller than half of the girth. Unfortunately, graphs of good finite-length LDPC codes contain a large number of cycles, which typically have a small girth [ETV99].

Cycles in the Tanner graphs prevent the sum-product algorithm from converging [EMC98, ETV99]. Further, cycles, especially short cycles, degrade the performance of LDPC decoders because they affect the independence of the extrinsic information exchanged in the iterative decoding. Hence, LDPC codes with large girth are desired. Some researchers proposed methods for constructing LDPC codes with large girth [HEA01, ZM03]

This chapter introduces a novel concept of girth degree, which can be considered as the extension of the classical concept of girth. The proposed concept combines the idea of girth and node degree distribution. Girth degree as a tool for characterizing LDPC codes and a measure of their performance is introduced in [HEK07a]. A tree-based algorithm for detecting girth and counting girth degree is also discussed. The results show that the concept of girth degree can be used as a tool to explore the characteristics and the performance of LDPC codes. In some cases, the concept of girth degree is working more effectively than the classic concept of girth.

3.2 Concept of Girth Degree

In graph theory, girth g is referred to as the length of the shortest cycle in the graph. Mao and Banihashemi [MB01] extend the definition of girth for a variable node v in the graph to be the length of the shortest cycle passing through that variable node. This girth is referred to as local girth g_v and its set $\{g_v\}$ is referred to as girth histogram. It follows that the girth of the graph introduced beforehand is referred to as global girth $g = \min\{g_v\}$. Because the graph is bipartite, all cycles must have even length and the minimum cycle is four. Hereafter, the term girth is used in the same meaning with local girth if it is not referred to other meaning.

Based on the definition of the local girth above, each variable node has a local girth. It is interesting to characterize a code by considering the distribution of the local girth among all variable nodes. If all variable nodes of the code have the same local girth, then the code is referred to as having *homogeneous local girth* as shown in Figure 3.1, in which the homogeneity is represented by two variable nodes having the same local girth of four (g-4).

On the other side, the variable nodes may also have different local girth, for example, some variable nodes have local girth of four and the others have local girth of six. Figure 3.2 depicts a simple representation, in which two variable nodes 1 and 2 have different local girths: girth of four (g-4) and girth of six (g-6), respectively. If the variable nodes of code have different local girth, then the code is referred to having *heterogeneous local girth.*

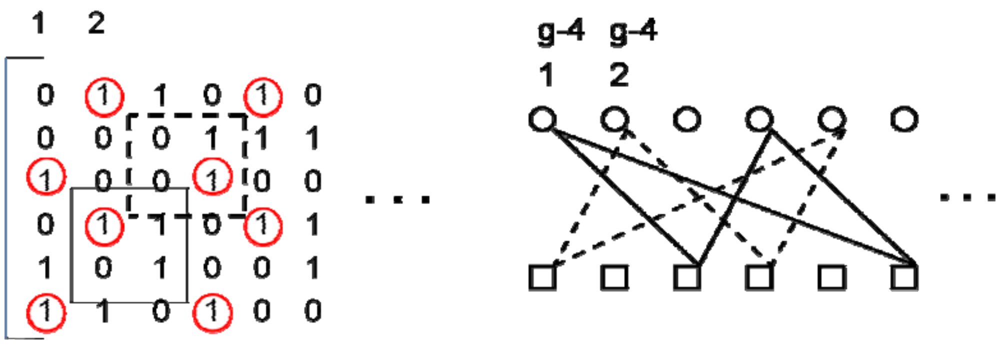


Figure 3.1 Example for homogeneous local girth

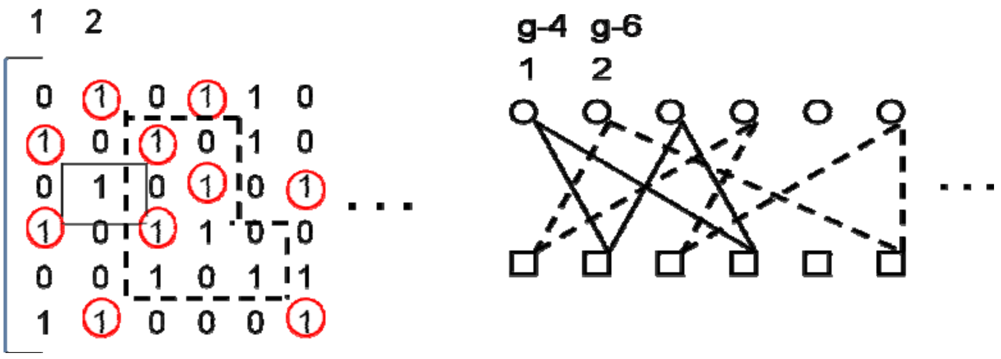


Figure 3.2 Example for heterogeneous local girth

Instead of using the girth histogram [MB01], a polynomial expression for representing the distribution of (local) girth is here introduced. The girth distribution indicates the fraction number of variable nodes with certain number of local girth. The girth distribution $\varphi(y)$ can be described by two parameters: local girth gv and the fraction of its corresponding variable node φ_{gv}.

$$\varphi(y) = \Sigma\, \varphi_{gv} \cdot y^{gv} \qquad (3.1)$$

with $\varphi = n_{gv}/n$, where n_{gv} denotes number of variable nodes with local girth gv (gv is *even* and $gv \geq 4$) and n denotes total number of variable nodes.

A quantity, average girth Φ_{gv} referred to as the average number of local girth per variable node, can be derived from (3.1) and is defined as follows

$$\Phi_{gv} = \Sigma\, \varphi_{gv} \cdot gv \qquad (3.2)$$

In order to understand the concept, an example is given. For instance, an LDPC code with heterogeneous local girth has half the number of variable nodes with local girth of six and the rest variable nodes with local girth of eight. Hence, the girth distribution of the code can be expressed as

$$\varphi(y) = 0.5\, y^6 + 0.5\, y^8$$

Hence, the code has average girth $\Phi_{gv} = 7$.

In [MB01, ZP01c] the girth distribution and the average girth are used as effective criteria for searching good LDPC code over one code ensemble. Based on hypothesis of independency in the iterative decoding, the larger average girth Φ_{gv} guarantees more independency in decoding iterations, and therefore, better code performance can be expected.

In addition to the concept of girth distribution above, a novel concept of (local) girth degree distribution is introduced. This concept can be considered as an extension of the theory of girth. This concept is meant to become a tool for characterizing an LDPC code in terms of its girth condition. The concept includes an algorithm for girth detection that can be used to assure that no short-four-cycles are included in the constructed LDPC code.

The idea of girth degree is dated back to the idea of node degree. Instead of vertices emanating from a variable node in case of node degree, the cycles of local girth passing a variable node are considered in case of girth degree. A terminology of girth degree is here coined to refer to as the number of cycle of local girth passing a variable node.

In the graph a variable node may have a number of cycles of local girth. The cycles are passing the node through different edges. As illustration, Figure 3.3 depicts two local girth of four (g-4) passing through the first non-zero entry in the first column in the parity check matrix. Correspondingly, in the corresponding bipartite-graph, the variable node 1 is passed through by two local girths of four. Actually, there are still more cycles of local girth of four passing the non-zero entries in the first column or variable node 1, which are here not illustrated for clarity. Hence, the variable node 1 can be said to have girth degree of six because it is passed through by six cycles of girth of four.

The distribution of girth degree is of interest for further characterizing the code. Girth degree distribution is referred to as the fraction number of variable nodes with a certain number of cycles of local girth. The girth degree distribution $\varphi(x,y)$ can be described by a polynomial with three parameters: local girth gv, girth degree gd and fraction of its corresponding variable node φ_{gd}.

$$\varphi(x,y) = \Sigma\varphi_{gv}\,(\Sigma\varphi_{gd}\,x^{gd}\,)\,y^{gv} \tag{3.3}$$

with $\varphi_{gd} = n_{gd}/n_{gv}$, where n_{gd} denotes the number of variable nodes with girth degree gd in the corresponding local girth gv, n_{gv} denotes number of variable nodes with local girth gv and n denotes total number of variable nodes. It is to note that gv is *even* and $gv \geq 4$.

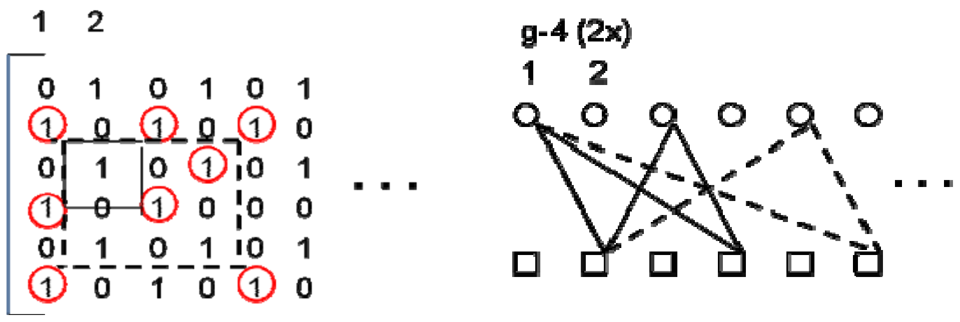


Figure 3.3 Two of six cycles of girth of four passing through the variable node 1

A quantity, average girth degree in polynomial form $\Phi_{gd}\,(y)$, referred to as the average number of cycles of local girth at a variable node, can be derived from (3.3) and defined as follows

$$\Phi_{gd}(y) = \Sigma\;\varphi_{gd}\cdot gd\cdot y^{gv} \tag{3.4}$$

To get the picture of the average girth degree distribution, the equation (3.3) can be simplified by taking the average girth degree into account instead of all girth degrees for describing the girth degree condition of each girth in more compact form as follows

$$\underline{\varphi}(x,y) = \Sigma\;\varphi_{gv}\,\underline{x}^{\Phi gd(y)}\,y^{gv} \tag{3.5}$$

where $\Phi_{gd}(y)$ denotes the average girth degree of girth gv.

In addition to that, the equation (3.3) can be also derived to get the whole picture on how the girths with their girth degrees are distributed within all variable nodes. The polynomial $\boldsymbol{\varphi}(x,y)$ shows the proportionality of variable nodes with girth degree of each girth within all variable nodes.

$$\boldsymbol{\varphi}(x,y) = \;\Sigma\,(\Sigma\varphi_{gv}\,\varphi_{gd}\,x^{gd}\,)\,y^{gv} \tag{3.6}$$

In order to understand the concept, an example is given. It is to note that the number applied here is simply made for clarity and do not reflect any real LDPC code. For instance, an LDPC code has heterogeneous local girth consisting of girth-6 at a half number of variable nodes with girth degree 5 and 10 (in fifty-fifty) and girth-8 at a quarter of variable nodes with girth degree 10 and 20 (in fifty-fifty) and girth-10 at the rest of variable nodes with girth degree 40, each. Hence, according to the equation (3.5) the girth degree distribution of the code is

$$\varphi(x,y) = 0.5(0.5x^{5}+0.5x^{10})\, y^{6} + 0.25(0.5x^{10} + 0.5x^{20})\, y^{8} + 0.25x^{40}\, y^{10}$$

Hence, the polynomial of the average girth degree is

$$\Phi_{gd}(y) = 7.5\, y^{6} + 15\, y^{8} + 40\, y^{10}$$

This states that average girth degree of girth-6, girth-8 and girth-10 is 7.5, 15 and 40, respectively.

In more solid form, the average girth degree distribution of the code can be described according to the equation (3.5) as follows

$$\underline{\varphi}(x,y) = 0.5\, \underline{x}^{7.5} y^{6} + 0.25\, \underline{x}^{15} y^{8} + 0.25\, \underline{x}^{40} y^{10}$$

This polynomial can be interpreted that the code has a half of variable nodes with girth-6 with 7.5 girth-cycles in average, a quarter of variable nodes with girth of 8 with 15 girth-cycles in average and a quarter of variable nodes with girth of 10 with 40 girth-cycles in average.

To get the picture of girth degree of each girth within all variable nodes, according to the equation (3.6) the corresponding polynomial can be expressed as follows

$$\boldsymbol{\varphi}(x,y) = (0.25\, x^{5}+0.25\, x^{10})\, y^{6} + (0.125\, x^{10} + 0.125\, x^{20})\, y^{8} + 0.25x^{40}\, y^{10}$$

This polynomial shows that girth-6 passes a quarter of variable nodes with 5 cycles, and other quarter with 10 cycles, girth-8 passes one-eighth of variable nodes with 10 cycles and other one-eighth with 20 cycles and girth 10 passes a quarter of variable nodes with 40 cycles.

In a code with heterogeneous local girth, the smallest local girth plays the most important role in determining the code performance. Its girth degree influences strongly the decoding performance. Their impact on the code performance is presumably stronger than that of the local girth in average or that of global girth, which is theoretically considered as determinant factor for the goodness of an LDPC code

Hypothetically, the code performance gets better as the average girth degree decreases because the lower the average girth degree is, the higher is the independency of variable nodes in the iterative coding, upon which the message-passing based decoding algorithm is working for Shannon-limit-near performance.

3.2.1 Girth Degree Detection Method

Here, an algorithm for detecting girth degree is proposed. The work of the algorithm consists of two parts, namely, girth detection and girth degree counting. The algorithm begins with the detection of the girth of all variable nodes, and than, the cycles of the girth are counted. From the work of the algorithm, the girth distribution as well as the girth degree distribution can be obtained.

In the first part of the algorithm, in principal, the algorithm checks the shortest cycles connecting the even number of non-zero elements in the parity-check matrix. In the bipartite graph, this corresponds to an even number of edges, in which the first and the last edge emanates from and incident onto the same variable node, respectively, in a closing form.

The process of girth detection takes place at all variable nodes. The girth detection begins with checking the smallest girth, i.e. girth 4, at each variable node. If the girth 4 is not found, the girth detection continues with the next higher girth. Soon a girth is realized at any column in the parity-check matrix or the corresponding variable node in the bipartite graph, the girth detection takes place in the next column or its corresponding variable node.

In the parity-check matrix, the girth is detected by tracing the non-zero elements in column-wise and row-wise vice-versa. The algorithm of the girth detection process can be described as follows. First, non-zero elements in the first column of the parity check matrix or correspondingly the first edge emanating from the first variable node is checked vertically as shown in an example depicted in Figure 3.4(a). There, three non-zero elements are recognized. Then, the non-zero elements in the corresponding rows are checked horizontally. From tracing in the first row, three non-zero elements, including the first traced non-zero element, are recognized. This tracing process continues until the final non-zero elements, which are in the same column with the first traced non-zero elements, are found.

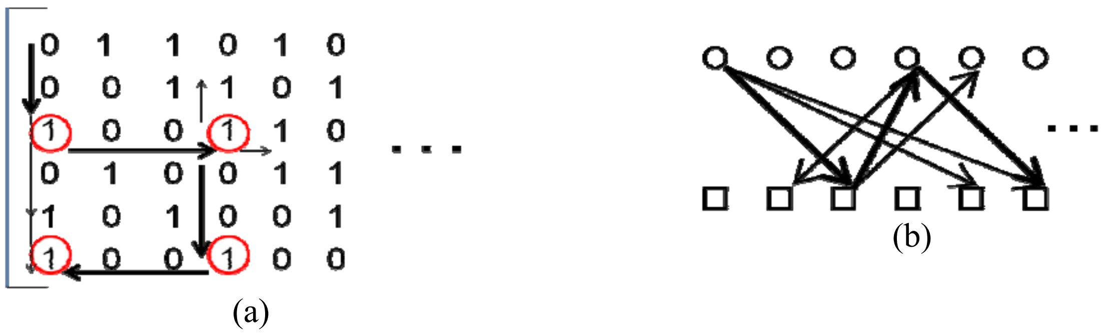


Figure 3.4 An example of girth detection process in a parity check matrix (a) and a bipartite graph (b)

Referring to Figure 3.4(a), the girth detection process can be described as follows
1) Trace the first column to detect non-zero elements e_1 and others
2) Trace row of non-zero element e_1 to detect non-zero element e_2 and others
3) Trace column of non-zero element e_2, detect non-zero element e_3 and others
4) Trace row of non-zero element e_3, and check non-zero element

5) If a non-zero is detected in the first column, than a girth of four is found. Otherwise go back to (2) to find other non-zero elements for a higher girth

Correspondingly, in the bipartite graph, the vertical tracing of non-zero elements within the first column is represented by three edges emanating from the first variable node as depicted in Figure 3.4(b). The following horizontal tracing of non-zero elements within the third row is represented by two edges emanating from the third check code. The tracing process continues until a girth is detected, or correspondingly, the edge reverts to the first variable node, from which the first edge emanates, to form a closed cycle.

In the second part of the algorithm, soon after a girth is found, its cycles passing through its associated variable nodes are counted. As the cycles of girth (or girth degree) of all variable nodes are counted, the distribution of girth degree can be determined.

The algorithm of girth degree counting starts with the girth detection. As a girth is detected, the number of its cycles is then counted. The algorithm counts the number of girths emanating from a variable node or its corresponding column. Figure 3.5 shows an example of the girth counting process for the first three columns. Each column has a girth of 4 in the depicted part of the parity-check matrix. The number of girth cycles may increase if the whole part of the parity-check matrix is considered. After the cycles of girth at all columns are counted, the girth degree distribution can be determined

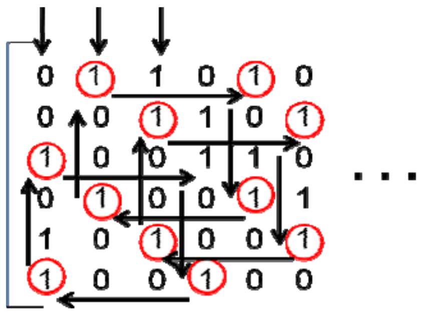


Figure 3.5 An example of girth degree counting process for girth of four

The girth degree counting process can be summarized as follows

1) Detect a girth of a variable nodes, starting with the first nodes
2) Count the number of cycles of the detected girth
3) Repeat the step 1 and 2 for the next variable nodes
4) After the girth degree counting for all variable nodes is done, present the girth degree distribution.

In fact, counting the girth degree needs more computation than only detecting girth. Counting the girth is a time-consuming process, especially if the dimension of the parity check matrix is large. This algorithm can be also used to count the number of cycles that are higher than the girth. However, more extensive computation is necessary. Referring to the idea that a small cycle degrades the code performance significantly, only girth is considered and hence, the other higher cycles can be ignored.

3.3 Simulation Results

Here, the concept of girth degree is applied to characterize LDPC codes so that the behavior of the codes is better understood. The codes are characterized by investigating their girth distribution and its correlation with some code parameters, i.e. code rate and code length. Moreover, the codes are characterized by their girth degree distribution.

Impact of code length and code rate on girth condition

First, the concept is applied to a well-known LDPC benchmark code, i.e. random MacKay code [MN95]. It is interesting to investigate the girth distribution of the code with column weight of three in different code lengths. Figure 3.6 shows girth distribution of the MacKay code at code rate 3/4. In general, the girth distribution is heterogeneous for all code lengths. The exception is applicable for code lengths between 200 and 800 bits, where the girth distribution is homogenous with girth-6. The fraction of girths is changing with the code length. As the code length increases, the fraction of larger girth increases and the fraction of smaller girth decreases.

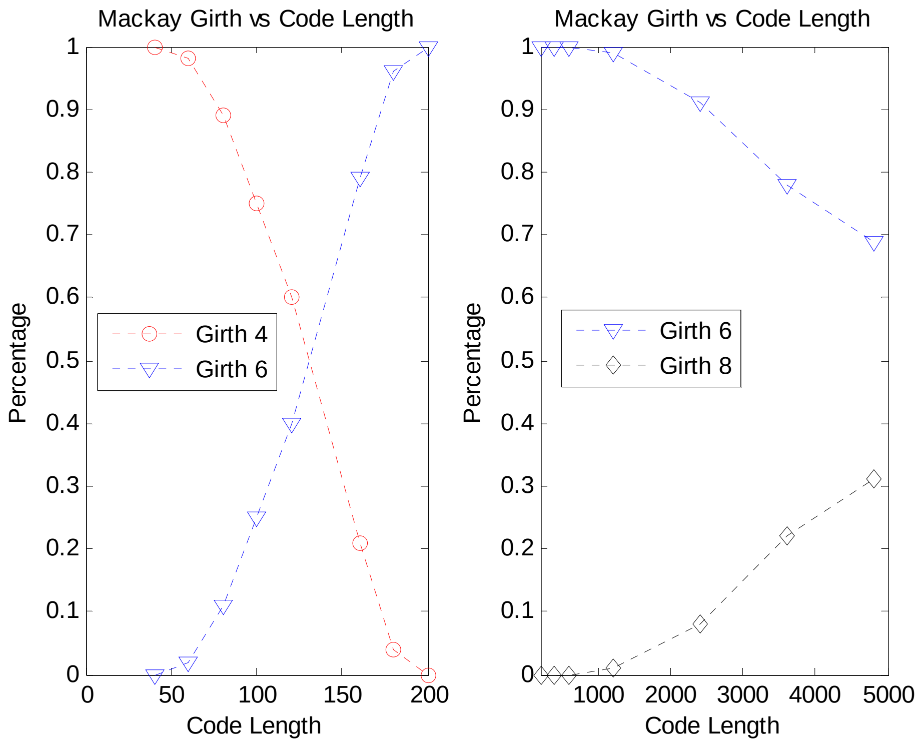


Figure 3.6 Girth distribution of MacKay code at code rate 3/4 as function of code length

In very short codes (length up to 200 bits), the existence of harmful girth-4 (short four cycles) is unavoidable. At this code length, the code has a heterogeneous girth distribution and contains a fraction of girth of four (girth-4) and girth of six (girth-6). Here, the fraction of girth-6 increases and the fraction of girth-4 decreases as the code length increases. In code lengths between 200 bits and 800 bits, the code has a homogeneous girth distribution with all girth-6. The girth distribution begins to become heterogeneous, where the fraction of girth-6 decreases and the fraction of girth-8 increases as code length become longer.

This fact proves that larger fraction of larger girth (or larger girth distribution) corresponding to longer code length contributes to better code performance. As theoretically proved, larger girth guarantees more independence of the extrinsic information exchanged in the iterative decoding leading to better decoding convergence.

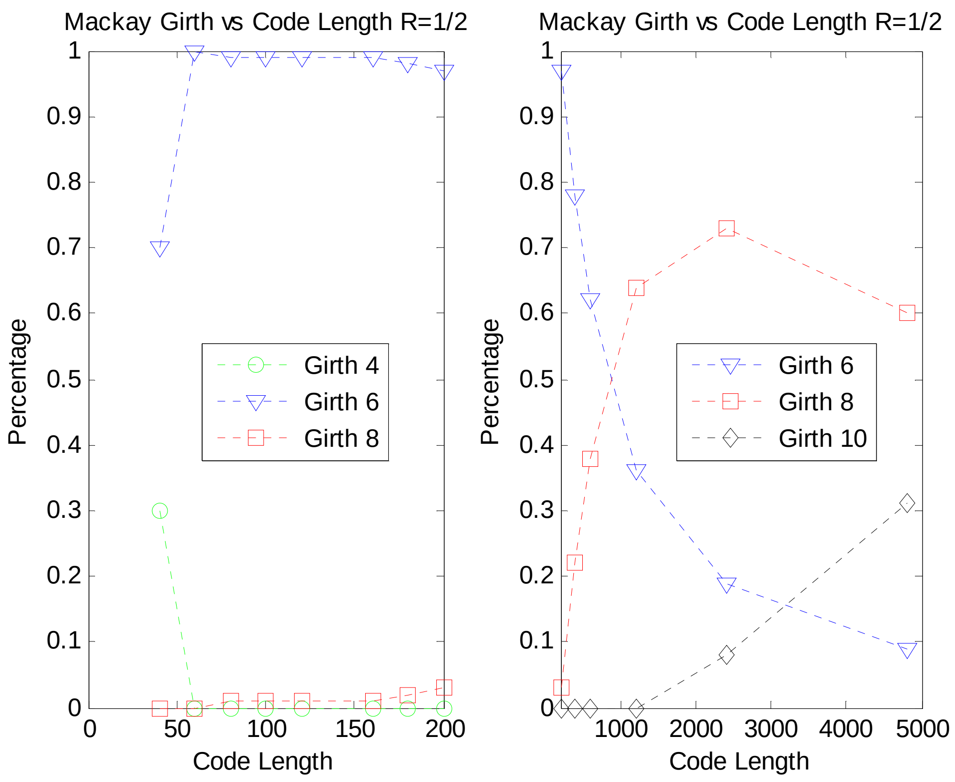


Figure 3.7 Girth distribution of MacKay code at code rate 1/2 as function of code length

Furthermore, the impact of the code rate on the girth distribution of the code is investigated. In Figure 3.7 the girth distribution of the code as a function of the code length at code rate 1/2 can be observed. In comparison with the code at code rate 3/4, the girth distribution of the code at code rate 1/2 is relative better. It means that the fraction of larger girth is relative higher at the same code length. At very short code lengths (less than 200 bits) the code has quasi homogeneous girth distribution, in which girth-6 appears strongly dominant over girth-8. At this code length, no harmful girth-4 exists, except at code length of tens bits. As the code length increases, the code tends to become more heterogeneous, in which the fraction of girth-6 decreases significantly and the fraction of girth-8 increases. At the code length larger than 1,200 bits, the fraction of girth-10 begins to contribute to the girth distribution and to substitute the lower girths. This is not the case within the 3/4-rate code, in which girth-6 still plays a dominant role. Therefore, the 1/2-rate code has better girth distribution.

The comparison of both codes with code rate 1/2 and 3/4 in terms of girth distribution can be summarized in their average girth as shown in Figure 3.8. The average girth of both codes increases as their code length gets higher. The average girth of the 1/2-rate code is higher than that of the 3/4–rate code. For example, at code length of 2,400 bits, the 1/2-rate code has average girth of 7.78. It is larger than that of 3/4–rate code by 1.62. This number of average girth results from their girth distribution. The 1/2-rate code has the following girth distribution

$$\Phi_{gd}(y) = 0.19\, y^6 + 0.73\, y^8 + 0.08\, y^{10},$$

while the 3/4-rate code has the following girth distribution

$$\Phi_{gd}(y) = 0.92\, y^6 + 0.08\, y^8.$$

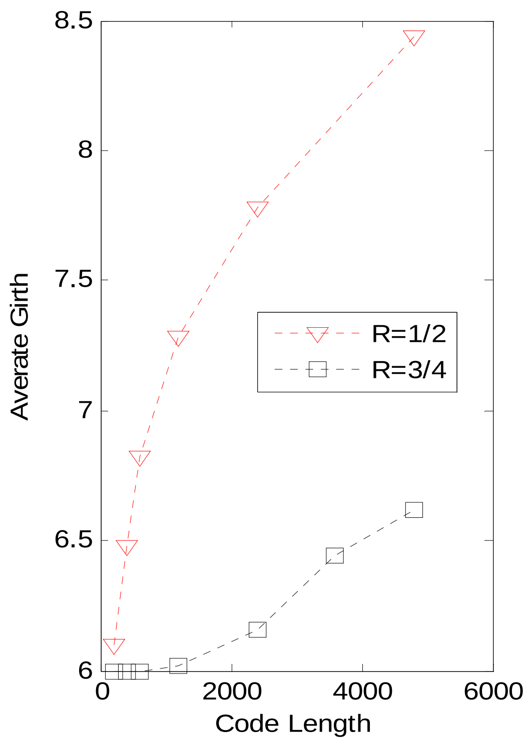


Figure 3.8 Impact of code length on the average girth at different code rate

It means that the 1/2-rate code has a smaller fraction of the smallest girth (girth-6) with 19% in comparison with that of the 3/4-rate code with 92%. In fact, in the code performance, the 1/2-rate code outperforms the 3/4-rate code. This is achieved by the better girth distribution of the first code, in which a higher average girth leads to better performance of the message-passing based decoding algorithm. The average girth and the fraction of the smallest girth can be applied to estimate the code performance.

Now, the concept of the girth degree is applied to characterize MacKay codes with code rate 1/2 and 3/4. Figure 3.9 shows the average girth degree for each local girth, i.e. girth-6, girth-8 and girth-10, for both code rates as a function of the code length up to 4,800 bits. In general, the average girth degree for all girth decreases as code length increases. It means that the number of cycles of girth passing the variable nodes decreases and asymptotically approaches one. The decrease of the girth degree allows the high independency of variable nodes leading to better performance of the message-passing based decoding.

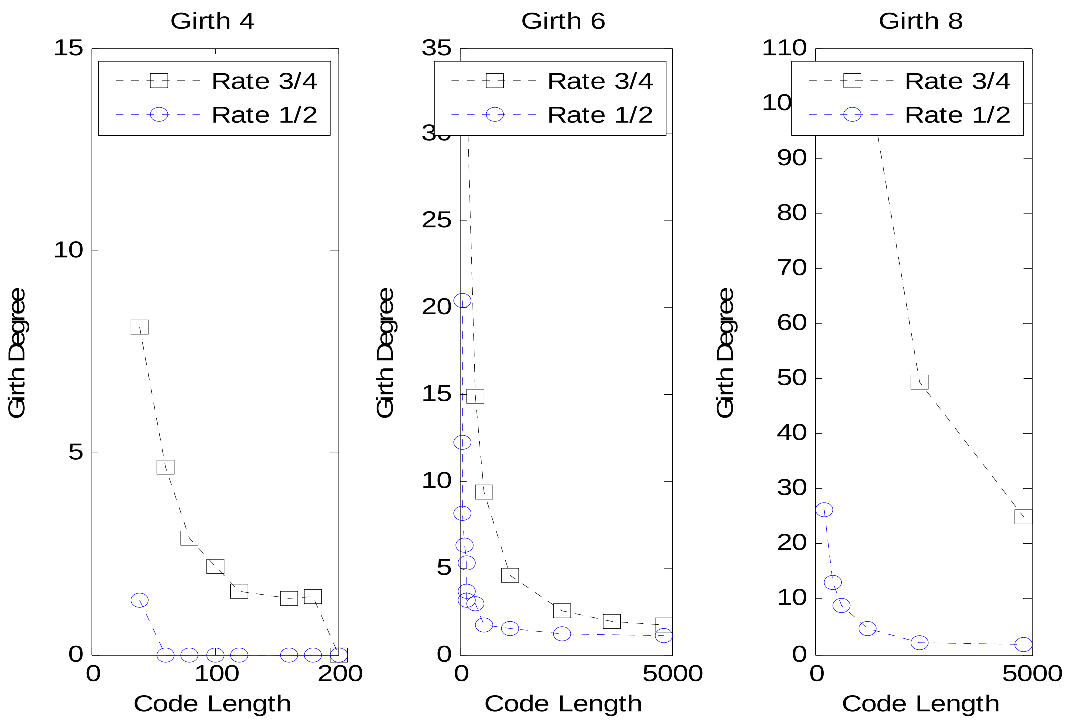


Figure 3.9 Impact of code rate on average girth degree in different code rate

Furthermore, by using the concept of girth degree, the codes can be characterized in terms of girth degree distribution. For example, the code with code length of 1,200 bits and code rate 1/2 has a heterogeneous average girth degree distribution

$$\underline{\varphi}(x,y) = 0.36\,\underline{x}^{1.72}\,y^{6} + 0.64\,\underline{x}^{4.59}\,y^{8}.$$

This polynomial means that 36% of variable nodes has girth-6, which passing them with 1.72 cycles in average and 64% of variable nodes has girth-8, which passing them with 4.59 cycles in average.

Meanwhile, the 3/4-rate code has heterogeneous average girth degree distribution

$$\underline{\varphi}(x,y) = 0.99\, \underline{x}^{4.57}\, y^{6} + 0.01\, \underline{x}^{105}\, y^{8}.$$

This polynomial means that 99% of variable nodes has girth-6, which passing them with 4.57 cycles in average and 1% of variable nodes has girth-8, which passing them with 105 cycles in average.

In comparison of both codes in terms of girth degree distribution, it is realized that the smallest girth (girth-6) of the 1/2-rate code has smaller fraction of variable nodes as well as smaller cycles (girth degree) than that of the 3/4-rate code. The smaller node fraction of the smallest girth and its corresponding girth degree can lead to better independency of variable nodes, upon which the message-passing based decoding perform better. In fact, the 1/2-rate code outperforms the 3/4-rate code as shown in Figure 3.10.

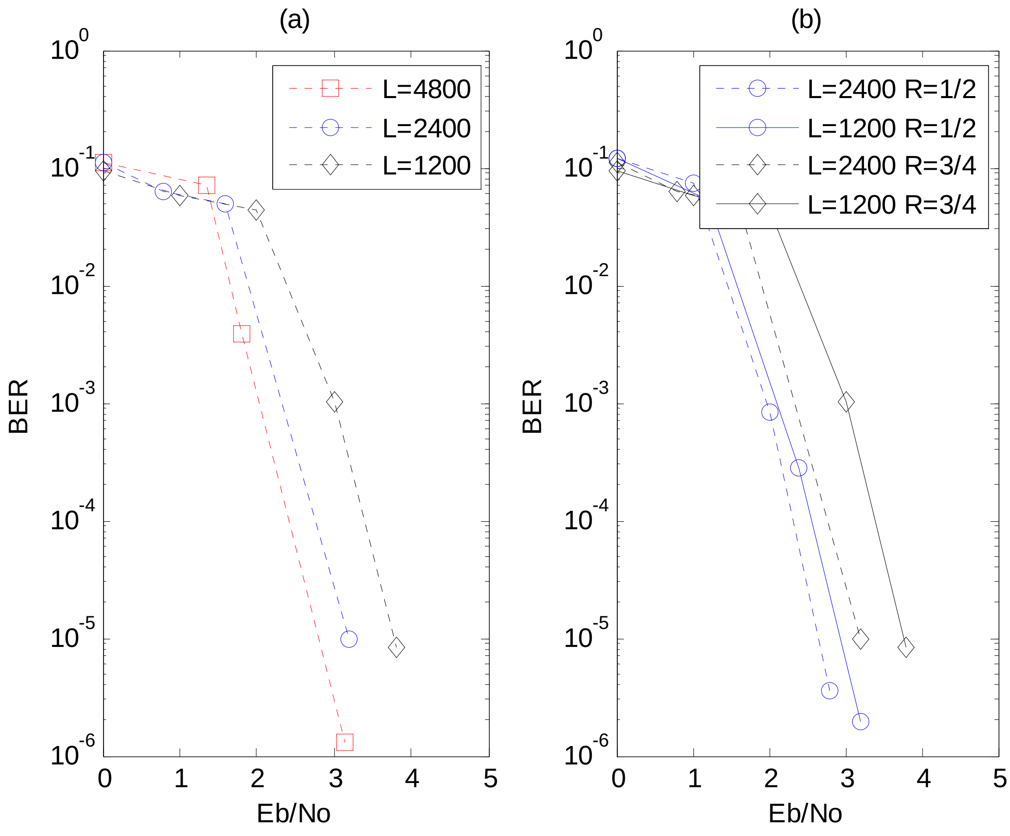


Figure 3.10 Performance of MacKay code in different code length (a) and code rate (b)

Similar to the average girth, the average girth degree of the smallest girth can be used to estimate the code performance. As shown in Figure 3.10, the 1/2-rate code has better girth degree distribution leading to a lower average girth degree than the 3/4-rate code. Smaller average girth degree means less independence of the extrinsic information exchanged in the iterative decoding leading to better decoding convergence.

Impact of code construction method on girth condition

Some aspects of code are investigated using the concept of girth degree, including girth distribution, average girth, girth degree distribution and average girth degree. Two types of LDPC codes: random MacKay code and structured array code with column weight 3 are characterized. It is of interest to observe the aspect of codes affected by their code length. Three code lengths - 600 bit, 1,200 bit, 2,400 bit - are chosen. In terms of code performance, it is of interest to observe the smaller girths, i.e. girths of six and eight, according to the folk knowledge that belief propagation works well if the graph does not contain too many short cycles.

Girth Degree Characterization of random MacKay codes

In case of random MacKay codes the girth distribution can be represented in form of polynomials in Table 3.1 and in form of histogram in Figure 3.11. The histogram shows the fraction of variable nodes based on their associated girth in random MacKay codes. It shows that the number of variable nodes with girth-6 decreases as code length increases. On the other side, the number of variable nodes with larger girth, i.e. girth-8, increases. The average girth of the codes increases asymptotically as code length increases (see Figure 3.12).

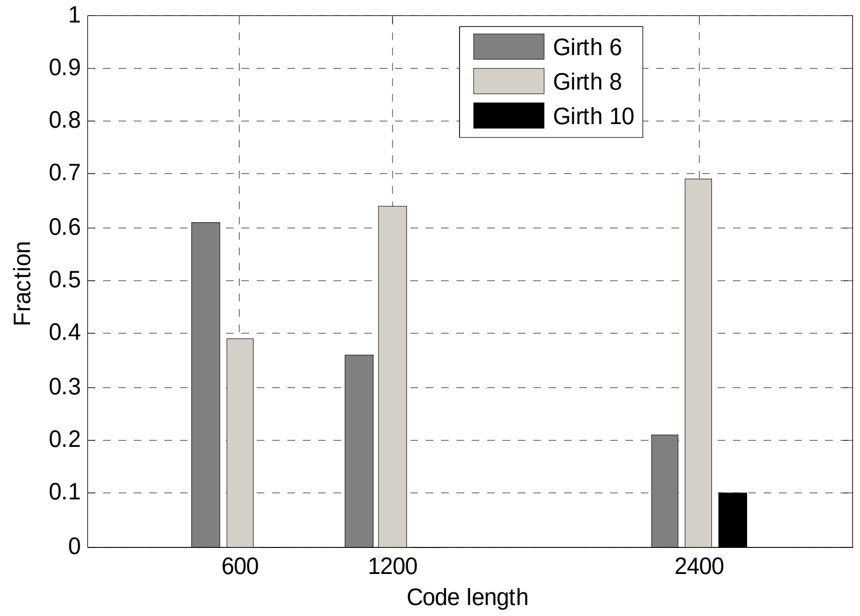


Figure 3.11 Fraction of node with girth of six and eight of MacKay code to code length

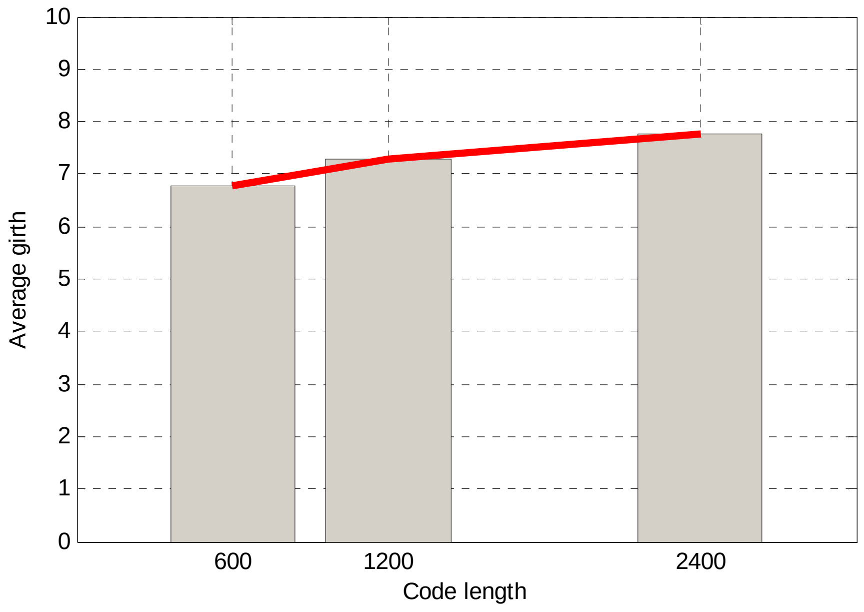


Figure 3.12 Average girth of MacKay code to code length

Table 3.1 Code length and girth distribution of MacKay code

Code length (bits)	**Girth Distribution**	**Average girth**
600	$0.61\, y^6 + 0.39\, y^8$	6.78
1,200	$0.36\, y^6 + 0.64\, y^8$	7.28
2,400	$0.21\, y^6 + 0.69\, y^8 + 0.10\, y^{10}$	7.78

Table 3.2 Girth degree distribution of MacKay code

Code length (bits)	Girth-6 (y^6)	Girth-8 (y^8)
600	$0.614\, x^1 + 0.268\, x^2 + 0.093\, x^3 + 0.025\, x^4$	$0.004\, x^2 + 0.009\, x^3 + 0.017\, x^4 + 0.051\, x^5 + 0.111\, x^6 + 0.098\, x^7 + 0.191\, x^8 + 0.085\, x^9 + 0.094\, x^{10} + 0.089\, x^{11} + 0.089\, x^{12} + 0.064\, x^{13} + 0.055\, x^{14} + 0.021\, x^{15} + 0.009\, x^{16} + 0.004\, x^{17} + 0.004\, x^{18} + 0.004\, x^{20}$
1,200	$0.794\, x^1 + 0.180\, x^2 + 0.021\, x^3 + 0.005\, x^4$	$0.052\, x^1 + 0.088\, x^2 + 0.168\, x^3 + 0.192\, x^4 + 0.182\, x^5 + 0.127\, x^6 + 0.096\, x^7 + 0.044\, x^8 + 0.031\, x^9 + 0.012\, x^{10} + 0.003\, x^{11} + 0.003\, x^{12} + 0.001\, x^{13} + 0.001\, x^{16}$
2,400	$0.890\, x^1 + 0.108\, x^2 + 0.002\, x^3$	$0.268\, x^1 + 0.319\, x^2 + 0.209\, x^3 + 0.125\, x^4 + 0.059\, x^5 + 0.012\, x^6 + 0.007\, x^7 + 0.002\, x^8$

Now, the girth degree of girth of MacKay codes is investigated. The observation is restricted to girth-6 and girth-8, which are of interests because their smaller cycles play a significant role in the decoding performance. The polynomial of girth degree distribution of those girths is presented in Table 3.2, which is illustrated in Figures 3.13-3.14 for easier analysis. The distribution of girth degree is more heterogeneously widespread and concentrated around higher girth degrees at smaller code lengths. As the code length increases, the fraction of smaller girth degree of each girth increases and therefore, the distribution of the girth degree concentrates more homogeneous at a smaller girth degree. Hence, the average girth degree decreases (see Table 3.3). It means that at larger code lengths, a larger number of variable nodes are passed through by smaller girth cycles, which reduces asymptotically to one cycle in average. It means that the node independency gets higher as the number of short cycles in the bipartite graph is reduced and hence, the message-passing based decoding performance increases.

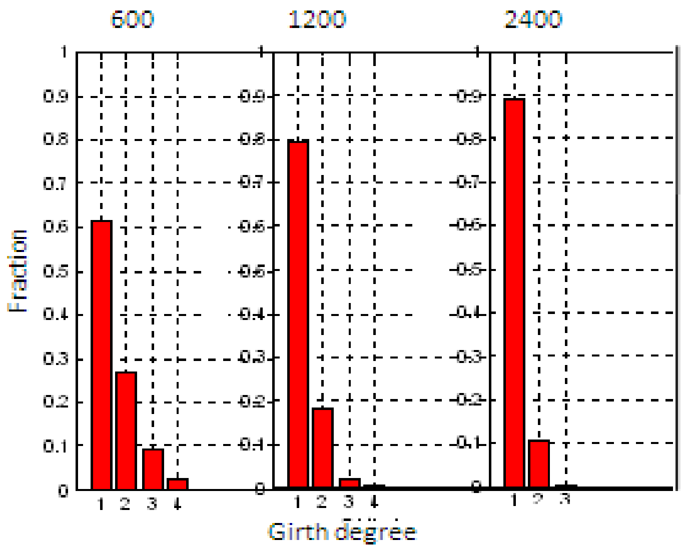


Figure 3.13 Girth degree distribution of girth 6 of MacKay code

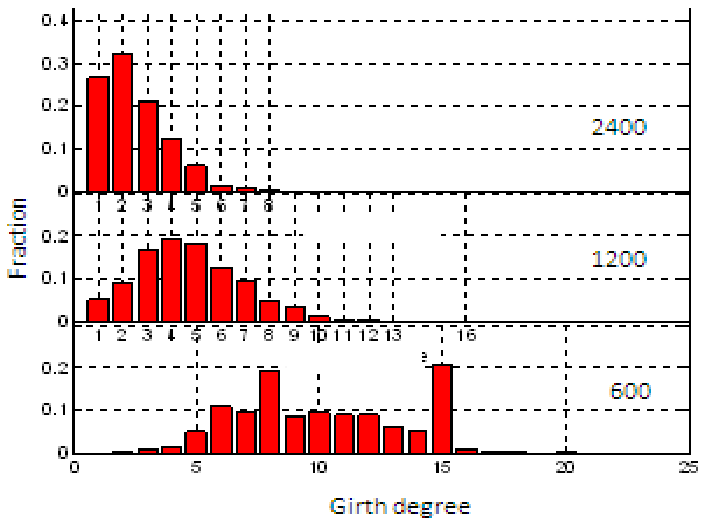


Figure 3.14 Girth degree distribution of girth-8 of MacKay code

Table 3.3 Average girth degree of the smallest girth of of MacKay code

Code length (bits)	**Average girth degree of Girth 6**
600	1.53
1,200	1.23
2,400	1.16

The trend of girth degree distribution can be also realized in terms of average girth degree as shown in Figure 3.15. Both girth-6 and girth-8 have decreasing average girth degree with different decreasing rate, in which the average girth degree of girth-8 decreases relative fast exponentially. Both approach an average girth degree of one asymptotically.

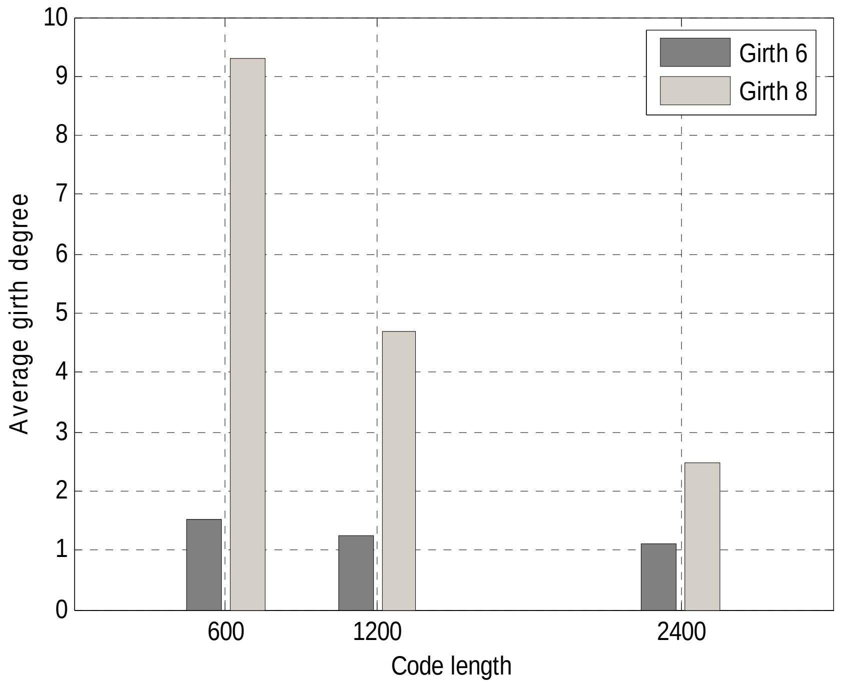


Figure 3.15 Girth degree average of MacKay code to code length

Girth Degree Characterization of Array Codes

As benchmarking to random code, array LDPC codes [Fan00, EO01] as a representative of structured LDPC codes is considered. The array LDPC codes are constructed in three different code lengths: 600, 1,200 and 2,400 bits with column weight 3. The girth condition of the code is than characterized using the concept of girth degree. The girth distribution of the code can be represented in form of polynomials as presented in Table 3.4. It is realized that array codes have homogeneous girth distribution for all code lengths examined. All variable nodes of the codes have girth-6 and no girth-8, and hence, the average girth of the codes is 6.

Table 3.4 Code length and girth distribution of Array code

Code length (bits)	Girth Distribution	Average girth
600	1 y^6	6
1,200	1 y^6	6
2,400	1 y^6	6

Now, the girth degree of girth of array codes is investigated. The polynomial of girth degree distribution of the girth-6 is presented in Table 3.5. It is shown that the girth degree distribution of girth 6 is homogeneous for all code lengths. Its girth degree 4, 6 and 8 are each distributed to one-third of the variable nodes. Therefore, the average girth degree of array code is fixed to six for all code rates (see Table 3.6).

Table 3.5 Girth degree distribution of array codes of the smallest girth of array codes

Code length (bits)	Girth 6 (y^6)
600	$0.333\,x^4 + 0.333\,x^6 + 0.333\,x^8$
1,200	$0.333\,x^4 + 0.333\,x^6 + 0.333\,x^8$
2,400	$0.333\,x^4 + 0.333\,x^6 + 0.333\,x^8$

Table 3.6 Average girth degree of the smallest girth of array codes

Code length (bits)	Average girth degree of Girth 6
600	6
1,200	6
2,400	6

From the simulation results, some aspects of codes of both random MacKay codes and structured array codes can be put in comparison in viewpoint of the concept of girth degree. As a random code, the MacKay code has a heterogeneous girth degree distribution, which tends to become more homogeneous as code length increases. Correspondingly, its average girth degree approaches asymptotically one for each girth. In contrary to that, the array codes as regular, structured codes, constitutes homogeneous girth degree distribution. Therefore, its average girth degree is constant at six for all code lengths.

In comparison with an array code, MacKay codes have larger average girth and smaller average girth degree of girth of six. In terms of code performance, MacKay codes outperform array codes at code length 1,200 bits, code rate 1/2 and column weight 3 (see Fig. 3.16). It is here to note that the significant worse BER performance of array codes is related to their low column weight established for fair comparison purpose. By higher column weight the performance of array codes is comparable with MacKay codes [EO01].

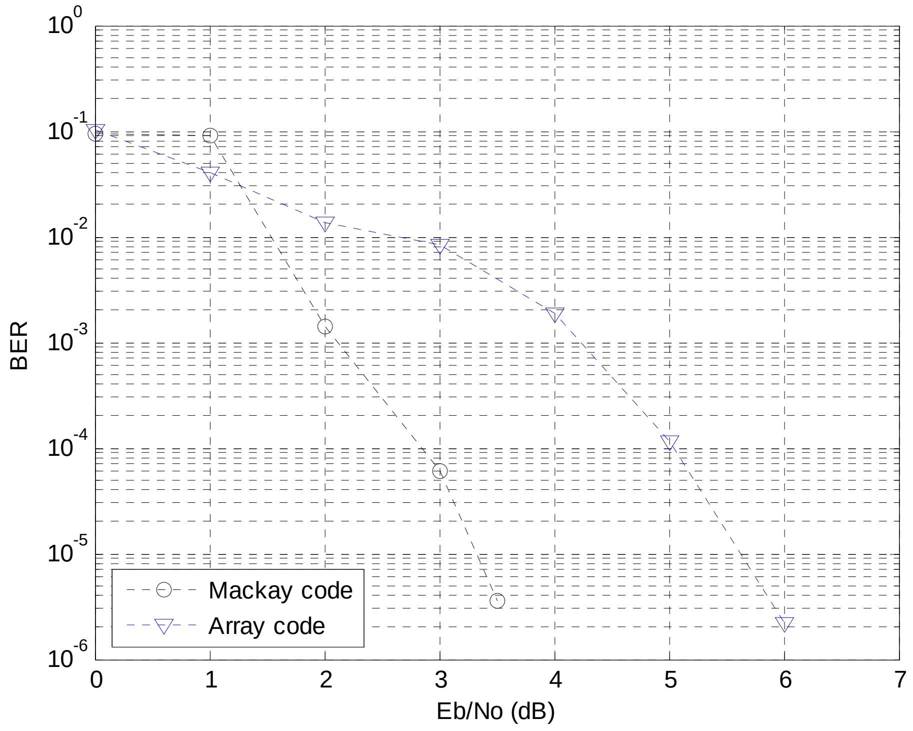


Figure 3.16 Comparison of BER performance of MacKay code and array code with column weight 3 at code length 1,200 bits and code rate 1/2

Although there are several aspects influencing the code performance, especially the number of check nodes (or column weight), the quality of girth plays also a significant role. At the same column weight, the smaller average girth degree of the smallest girth means less short cycles in the bipartite graph involved in belief propagation during iterative decoding leading to better code performance.

Girth Degree Characterization of WLAN Codes

It is of interest to characterize the standard IEEE802.11n codes [IEEE09] (hereafter, WLAN codes) in terms of their girth condition. The code characterization is applied for all code rates, i.e. 1/2, 2/3, 3/4 and 5/6 as well as all code lengths, i.e. 648 bits, 1,296 bits and 1,944 bits. The girth condition of the WLAN codes includes girth distribution and girth degree distribution.

In case of WLAN codes with code length 648 bits, girth 6 is dominantly passing through the variable nodes as shown in Table 3.7. At code rate 2/3 and 5/6 all variable nodes have girth-6, while, at code rate 1/2 the fraction of variable nodes with girth-6 reduces to 79% and girth 8 sizes 21% of variable nodes. Here, it is of interests that at code rate 3/4 the short-four-cycles (girth 4) exist at about 17% of variable nodes. Therefore, the average girth of the codes is dropped at this code rate. However, the existence of the girth 4 is harmless for code performance of the WLAN codes of code length 648 bits with code rate 3/4 as shown in Figure 3.17.

Table 3.7 Girth distribution of WLAN codes (code length = 648 bits)

Code rates	Girth distribution	Average girth
1/2	$0.79\,y^6 + 0.21\,y^8$	6.42
2/3	$1\,y^6$	6
3/4	$0.17\,y^4 + 0.83\,y^6$	5.66
5/6	$1\,y^6$	6

In terms of girth degree condition, the girth degree distribution of the girth 6 is more widespread at higher girth degree as the code rate increases (see Table 3.8). For example, at code rate 1/2 the girth degree is ranging from one cycle to 120 cycles, while at code rate 5/6 the range of the girth degree is between 32 cycles to 181 cycles. It is here to note that the girth degree of all variable nodes with girth 4 at code rate 3/4 is one cycle. In addition to that, the average girth degree of the girth 6 of the codes increases as the code rate becomes higher (see Table 3.9).

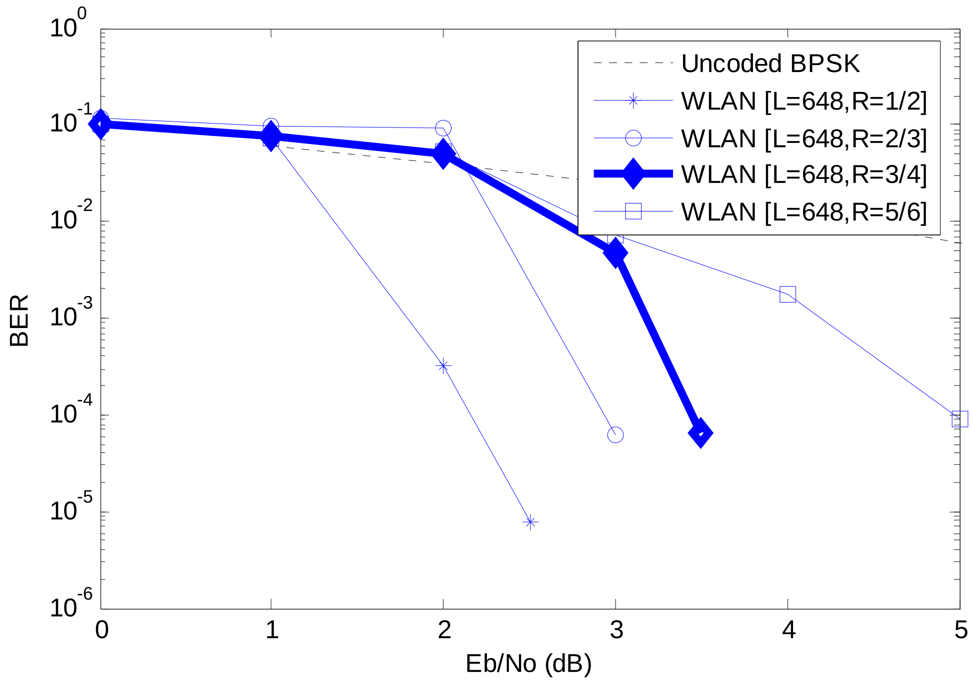


Figure 3.17 Code performance of the 648-length WLAN codes at different code rates

Table 3.8 Girth degree distribution of WLAN codes (code length = 648 bits)

Code rates	**Girth 4 (y^4)**	**Girth 6 (y^6)**	**Girth 8 (y^8)**
1/2		$0.105x^1 + 0.105x^2 + 0.105x^3 + 0.053x^4 + 0.0105x^8 + 0.105x^9 + 0.105x^{10} + 0.053x^{11} + 0.105x^{12} + 0.053x^{104} + 0.053x^{109} + 0.053x^{120}$	$0.2x^{112} + 0.2x^{118} + 0.4x^{131} + 0.2x^{352}$
2/3		$0.042x^4 + 0.042x^6 + 0.042x^7 + 0.042x^8 + 0.042x^9 + 0.042x^{10} + 0.042x^{11} + 0.042x^{14} + 0.042x^{15} + 0.083x^{18} + 0.083x^{19} + 0.083x^{23} + 0.042x^{31} + 0.083x^{32} + 0.042x^{40} + 0.042x^{46} + 0.042x^{83} + 0.042x^{133} + 0.042x^{144} + 0.042x^{149}$	
3/4	$1x^1$	$0.05x^{10} + 0.05x^{11} + 0.05x^{12} + 0.05x^{14} + 0.05x^{24} + 0.05x^{28} + 0.15x^{32} + 0.10x^{34} + 0.05x^{64} + 0.10x^{69} + 0.05x^{72} + 0.05x^{76} + 0.05x^{142} + 0.05x^{163} + 0.05x^{165} + 0.05x^{170}$	
5/6		$0.083x^{32} + 0.042x^{33} + 0.042x^{88} + 0.042x^{90} + 0.042x^{171} + 0.292x^{172} + 0.042x^{174} + 0.083x^{175} + 0.125x^{176} + 0.083x^{177} + 0.042x^{178} + 0.042x^{179} + 0.042x^{181}$	

Table 3.9 Average girth degree of WLAN codes (code length = 648 bits)

Code rates	**Girth 4**	**Girth 6**	**Girth 8**
1/2	0	23.05	168.80
2/3	0	37.25	0
3/4	1	62.65	0
5/6	0	149.75	0

A more compact girth characterization of the 648-length WLAN codes can be represented at their average girth degree distribution (see Table 3.10). For example, at code rate 1/2 about 79% of the variable nodes of the code have girth 6 with girth degree 23.05 cycles in average and about 21% of the variable nodes have girth 8 with girth degree 168.80 cycles in average. At code rate 2/3 and 5/6 all variable nodes of the WLAN codes are passed by girth 6 in 37.25 and 149.75 cycles in average. The interest lies in the WLAN codes at code rate 3/4. About 17% of the variable nodes of the codes are passed by girth 4 with 1 cycle in average and about 83% of the variable nodes have girth 6 with girth degree 62.65 cycles in average. Other code rates, 2/3 and 5/6 all variable nodes of the codes have girth 6 with girth degree 37.25 and 149.75 cycles in average, respectively.

Table 3.10 Average girth degree distribution of WLAN codes (code length = 648 bits)

Code rates	Average Girth distribution
1/2	$0.79\ \underline{x}^{23.05}\ y^{6} + 0.21\ \underline{x}^{168.80}\ y^{8}$
2/3	$1\ \underline{x}^{37.25}\ y^{6}$
3/4	$0.17\ \underline{x}^{1}\ y^{4} + 0.83\ \underline{x}^{62.65}\ y^{6}$
5/6	$1\ \underline{x}^{149.75}\ y^{6}$

In case of the WLAN codes with code length 1,296 bits, girth 6 is again becoming the most dominant girth (see Table 3.11). At code rate 1/2 girth 6 passes about 92% of variable nodes and the girth 8 passes the rest variable nodes. The WLAN codes at code rate 3/4 and 5/6 have all variable nodes with girth 6. It is of interests that at code rate 2/3 about 17% of variable nodes of the WLAN codes have the short-four-cycles (girth 4) exist. The existing girth 4 leads to decreased average girth, which is lower than those of the WLAN codes with higher code rate, 3/4 and 5/6. However, again, the existence of the girth 4 is harmless for the code performance of the 1296-length WLAN codes with code rate 2/3 4 as shown in Figure 3.18.

Table 3.11 Girth distribution of WLAN codes (code length = 1,296 bits)

Code rates	Girth distribution	Average girth
1/2	$0.92\ y^{6} + 0.08\ y^{8}$	6.16
2/3	$0.17\ y^{4} + 0.83\ y^{6}$	5.66
3/4	$1\ y^{6}$	6
5/6	$1\ y^{6}$	6

In terms of girth degree condition, the girth degree distribution of the girth 6 is again more widespread at higher girth degree as the code rate increases (see Table 3.12). For example, at code rate 1/2 the girth degree is ranging from one cycle to 38 cycles, while at code rate 5/6 the range of the girth degree is between 12 cycles to 73 cycles. It is here to note that at code rate 2/3 the girth degree of all variable nodes with girth 4 is one cycle. In general, the average girth degree of the girth 6 of the codes increases as code rate becomes higher (see Table 3.13). At code rate 1/2 the average girth degree is 6.95 cycles, while at code rate 5/6 the average girth degree is 52.87 cycles.

The girth characterization of the 1296-length WLAN codes can be represented at their average girth distribution (see Table 3.14). For example, at code rate 1/2 about 92% of the variable nodes of the code have girth 6 with girth degree 6.95 cycles in average and about 8% of the variable nodes have girth 8 with girth degree 34 cycles in average. It is of interest that at code rate 2/3 about 17% of the variable nodes of the code have girth 4 with girth degree 1 cycle in average. At other higher code rates, 3/4 and 5/6, all variable nodes of the codes have girth 6 with girth degree 32.50 and 52.87 cycles in average, respectively.

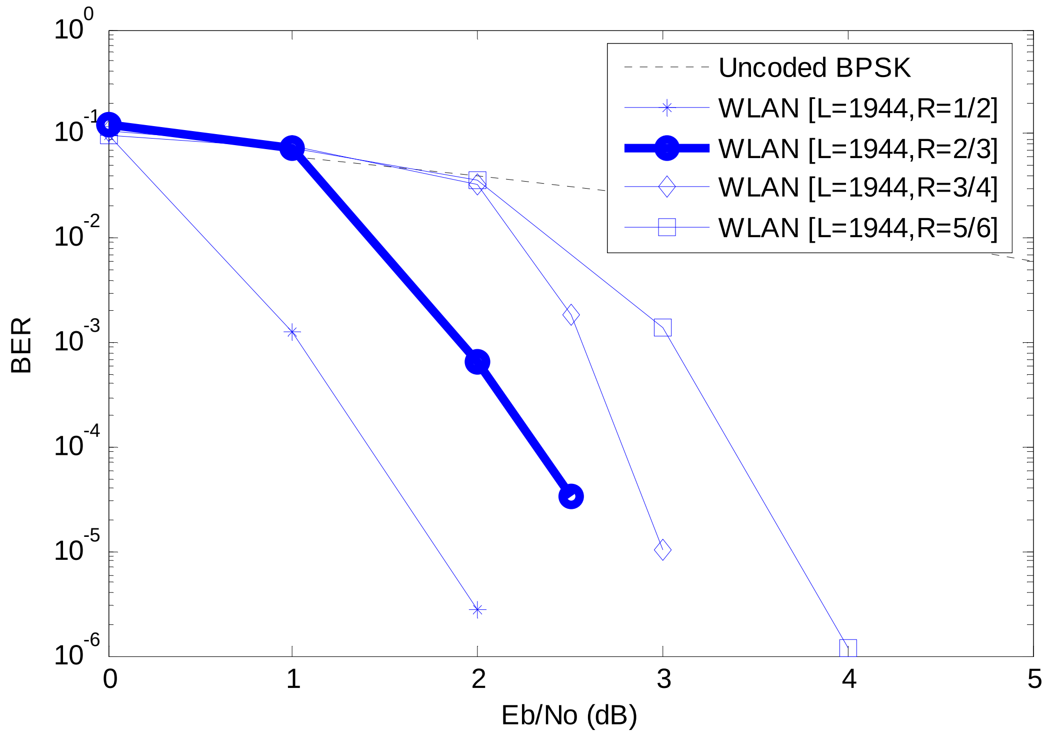


Figure 3.18 Code performance of the 1296-length WLAN codes at different code rates

Table 3.12 Girth degree distribution of WLAN codes (code length = 1,296 bits)

Code rates	**Girth 4 (y^4)**	**Girth 6 (y^6)**	**Girth 8 (y^8)**
1/2		$0.273\,x^1 + 0.318\,x^2 + 0.091\,x^3 + 0.091\,x^4 + 0.091\,x^5 + 0.045\,x^{34} + 0.045\,x^{37} + 0.045\,x^{38}$	$0.5\,x^{29} + 0.5\,x^{39}$
2/3	$1\,x^1$	$0.05\,x^1 + 0.05\,x^2 + 0.05\,x^3 + 0.15\,x^4 + 0.10\,x^5 + 0.20\,x^7 + 0.05\,x^8 + 0.10\,x^9 + 0.05\,x^{10} + 0.10\,x^{11} + 0.05\,x^{15} + 0.05\,x^{54}$	
3/4		$0.083\,x^7 + 0.083\,x^8 + 0.125\,x^{11} + 0.042\,x^{12} + 0.042\,x^{13} + 0.083\,x^{16} + 0.042\,x^{17} + 0.083\,x^{18} + 0.083\,x^{19} + 0.042\,x^{21} + 0.042\,x^{75} + 0.042\,x^{76} + 0.083\,x^{78} + 0.042\,x^{79} + 0.042\,x^{80} + 0.042\,x^{82}$	
5/6		$0.042\,x^{12} + 0.042\,x^{13} + 0.042\,x^{15} + 0.042\,x^{28} + 0.042\,x^{32} + 0.042\,x^{33} + 0.042\,x^{36} + 0.042\,x^{38} + 0.042\,x^{57} + 0.042\,x^{59} + 0.042\,x^{60} + 0.042\,x^{63} + 0.083\,x^{64} + 0.042\,x^{67} + 0.042\,x^{68} + 0.125\,x^{69} + 0.083\,x^{71} + 0.042\,x^{72} + 0.042\,x^{73}$	

Table 3.13 Average girth degree of WLAN codes (code length = 1,296 bits)

Code rates	Girth 4	Girth 6	Girth 8
1/2	0	6.95	34
2/3	1	9.15	0
3/4	0	32.50	0
5/6	0	52.87	0

Table 3.14 Average girth degree distribution of WLAN codes (code length = 1,296 bits)

Code rates	Average girth distribution
1/2	$0.92\,\underline{x}^{6.95}\,y^{6} + 0.08\,\underline{x}^{34}\,y^{8}$
2/3	$0.17\,\underline{x}^{1}\,y^{4} + 0.83\,\underline{x}^{9.15}\,y^{6}$
3/4	$1\,\underline{x}^{32.50}\,y^{6}$
5/6	$1\,\underline{x}^{52.87}y^{6}$

Table 3.15 Girth distribution of WLAN codes (code length = 1,944 bits)

Code rates	Girth distribution	Average girth
1/2	$0.87\,y^{6} + 0.13\,y^{8}$	6.32
2/3	$0.17\,y^{4} + 0.83\,y^{6}$	5.66
3/4	$1\,y^{6}$	6
5/6	$1\,y^{6}$	6

In case of the WLAN codes with code length 1,944 bits, girth 6 is still the most dominant girth (see Table 3.15). At code rate 1/2 girth 6 sizes about 87% of variable nodes of the codes, while girth 8 appears at about 13% of variable nodes. The codes at code rates 3/4 and 5/6 have all variable nodes with girth 6. It is of interests that at code rate 2/3 about 17% of variable nodes have the short-four-cycles (girth 4) exist, what leads to decreased average girth. Again, the existence of the girth 4 is however harmless for code performance of the 1944-length WLAN codes with code rate 2/3 as shown in Figure 3.19.

In terms of girth degree condition, the girth degree distribution of the girth 6 is more widespread at higher girth degrees as the code rate increases (see Table 3.16). For example, at code rate 1/2 the girth degree is ranging from one cycle to 31 cycles, while at code rate 5/6 the range of the girth degree is between 2 cycles to 39 cycles. It is here to note that at code rate 2/3 the codes have of a half of variable nodes with one-cycle-girth 4 and the other half with three-cycles-girth 4. The average girth degree of the girth 6 of the codes increases as code rate becomes higher (see Table 3.17). At code rate 1/2 the average girth degree is 5.86 cycles, while at code rate 5/6 the average girth degree is 22.25 cycles.

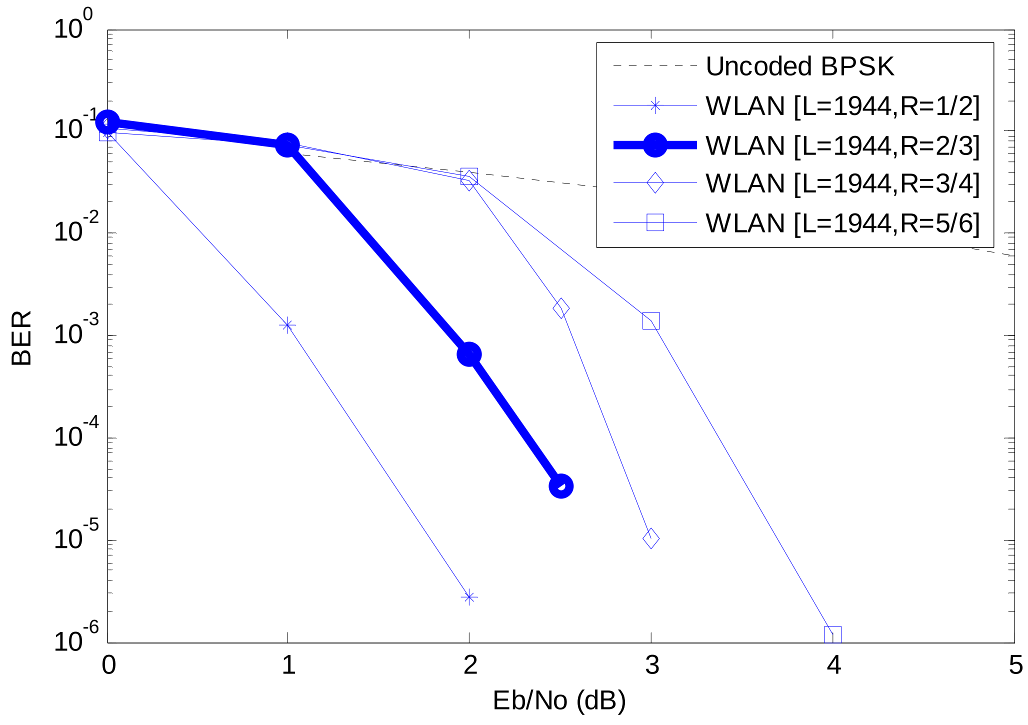


Figure 3.19 Code performance of the 1944-length WLAN codes at different code rates

Table 3.16 Girth degree distribution of WLAN codes (code length = 1,944 bits)

Code rates	Girth 4 (y^4)	Girth 6 (y^6)	Girth 8 (y^8)
1/2		$0.286\,x^1 + 0.333\,x^2 + 0.095\,x^3 + 0.143\,x^4 + 0.048\,x^{25} + 0.048\,x^{29} + 0.048\,x^{31}$	$0.333\,x^{26} + 0.333\,x^{30} + 0.333\,x^{34}$
2/3	$0.5\,x^1 + 0.5\,x^3$	$0.150\,x^1 + 0.100\,x^2 + 0.250\,x^3 + 0.150\,x^4 + 0.150\,x^5 + 0.050\,x^6 + 0.50\,x^{23} + 0.050\,x^{31} + 0.050\,x^{47}$	
3/4		$0.042\,x^2 + 0.083\,x^3 + 0.083\,x^4 + 0.167\,x^5 + 0.083\,x^8 + 0.125\,x^9 + 0.083\,x^{10} + 0.042\,x^{13} + 0.042\,x^{33} + 0.042\,x^{34} + 0.042\,x^{35} + 0.042\,x^{38} + 0.042\,x^{40} + 0.042\,x^{41}$	
5/6		$0.083\,x^2 + 0.042\,x^8 + 0.042\,x^{10} + 0.167\,x^{15} + 0.083\,x^{17} + 0.042\,x^{18} + 0.083\,x^{22} + 0.042\,x^{28} + 0.083\,x^{30} + 0.042\,x^{32} + 0.042\,x^{32} + 0.042\,x^{33} + 0.042\,x^{37} + 0.083\,x^{38} + 0.042\,x^{39}$	

Table 3.17 Average girth degree of WLAN codes (code length = 1,944 bits)

Code rates	Girth 4	Girth 6	Girth 8
1/2	0	5.86	30
2/3	2	7.80	0
3/4	0	14.38	0
5/6	0	22.25	0

The girth characterization of the 1944-length WLAN codes can be represented at their average girth distribution (see Table 3.18). For example, at code rate 1/2 about 87% of the variable nodes of the code have girth 6 with girth degree 5.86 cycles in average and about 13% of the variable nodes have girth 8 with girth degree 30 cycles in average. It is of interests that at code rate 2/3 about 17% of the variable nodes of the code have girth 4 with girth degree 2 cycles in average and about 83% of the variable nodes have girth 6 with girth degree 7.8 cycles in average. Other code rates, 3/4 and 5/6 all variable nodes of the codes have girth 6 with girth degree 14.38 and 22.25 cycles in average, respectively.

Table 3.18 Average girth degree distribution of WLAN codes (code length = 1,944 bits)

Code rates	Average girth distribution
1/2	$0.87\, \underline{x}^{5.86} y^{6} + 0.13\, \underline{x}^{30} y^{8}$
2/3	$0.17\, \underline{x}^{2} y^{4} + 0.83\, \underline{x}^{7.8} y^{6}$
3/4	$1\, \underline{x}^{14.38} y^{6}$
5/6	$1\, \underline{x}^{22.25} y^{6}$

As previously described, some WLAN codes contains short-four-cycles, i.e. the codes in code length 648 bits with code rate 3/4, code length 1,296 bits with code rate 2/3 and code length 1,944 bits with code rate 2/3 as summarized in Table 3.19. Those codes have the same girth distribution, i.e. 17% of variable nodes with girth 4 and 83% of variable nodes with girth 6. The girth degree of the girth 4 of the WLAN codes is one at code length 648 and 1,296 bits and two at code length 1,944 bits.

Table 3.19 Average girth distribution of WLAN codes having girth 4

Code length (bits)	Code rates	Average girth distribution
648	3/4	$0.17\, \underline{x}^{1} y^{4} + 0.83\, \underline{x}^{62.65} y^{6}$
1,296	2/3	$0.17\, \underline{x}^{1} y^{4} + 0.83\, \underline{x}^{9.15} y^{6}$
1,944	2/3	$0.17\, \underline{x}^{2} y^{4} + 0.83\, \underline{x}^{7.8} y^{6}$

The impact of girth and girth degree on the code performance is put in comparison as presented in Table 3.20. It is a rule of thumb that the code performance gets worse as the code rate increases. Moreover, the code performance has a strong correlation with the girth. Some constructions of LDPC codes are established with higher girth to have better code performance. In case of WLAN codes, it is difficult to make, either global girth or (average) local girth, to become the determinant factor for code performance. Only the codes with code rate 1/2 have relatively significant higher average girth. Hence, the rate-1/2 codes show the best code performance. However, at higher code rates, the average girth lies on 6, even lower than 6 in some codes because they include short-four-cycles.

Table 3.20 Girth condition of WLAN codes in comparison

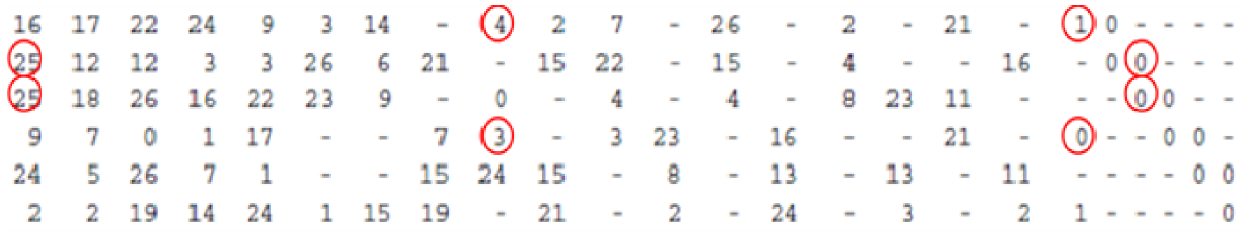
Code length/ Code rates	648 bits		1,296 bits		1,944 bits	
	Average girth	Avrg Girth degree of girth 6	Average girth	Avrg Girth degree of girth 6	Average girth	Avrg Girth degree of girth 6
1/2	6.42	23.05	6.16	6.95	6.32	5.86
2/3	6	37.25	5.66	9.15	5.66	7.80
3/4	5.66	62.65	6	32.50	6	14.38
5/6	6	149.75	6	52.87	6	22.25

On contrary to the average girth, the girth degree shows its efficiency as a measure for the code performance. The average girth degree of girth 6 increases as the code rate increases which in turn leads to a worse code performance. This fact shows that the number of cycles of girth plays more significant roles than girth in determination of code performance.

It is of great interest to investigate the impact of the existence of four-short-cycles on the performance of WLAN codes. The concept of girth degree has indicated the existence of short four cycles within some WLAN codes shown before: the 648-length, 3/4-rate code, the 1296-length, 2/3-rate code, and the 1944-length, 2/3-rate code. Their corresponding parity check matrices withdrawn from the IEEE802.11n Standard [IEEE09] are depicted in Figure 3.20.

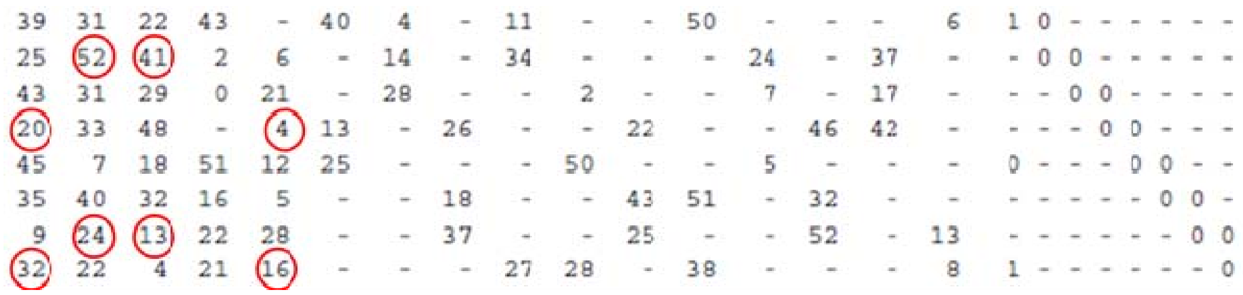
In those parity-check matrices, the slope parameter indicating non-zero positions of submatrices with girth 4 are marked with circle. The short four-cycles involves non-zero elements in four submatrices forming a quadratic cycle form, in which the modulus operation of the difference of non-zero positions between two submatrices in the same row is the same as in the other ones. This principle is used to avoid short four-cycles in the code construction method discussed in the chapter 4.

In case of the 648-length, 3/4-rate WLAN code, the short-four-cycles involves four submatrices in two pair of column, i.e. column 1 with column 21 and column 9 with column 19. In case of the 1296-length, 2/3-rate code, the short four-cycles also involves four submatrices in two pair of column, i.e. column 1 with column 5 and column 2 with column 3. In case of the 1944-length, 2/3-rate code, the short-four-cycles involves four submatrices in a pair of column, i.e. column 1 with column 3.



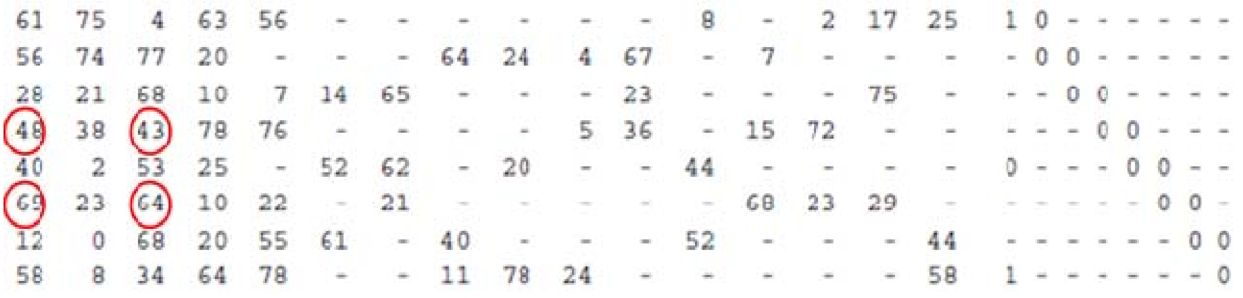
16	17	22	24	9	3	14	-	4	2	7	-	26	-	2	-	21	-	1	0	-	-	-	-
25	12	12	3	3	26	6	21	-	15	22	-	15	-	4	-	-	16	-	0	0	-	-	-
25	18	26	16	22	23	9	-	0	-	4	-	4	-	8	23	11	-	-	-	0	0	-	-
9	7	0	1	17	-	-	7	3	-	3	23	-	16	-	-	21	-	0	-	-	0	0	-
24	5	26	7	1	-	-	15	24	15	-	8	-	13	-	13	-	11	-	-	-	-	0	0
2	2	19	14	24	1	15	19	-	21	-	2	-	24	-	3	-	2	1	-	-	-	-	0

(a) Code length 648 bits and code rate 3/4



39	31	22	43	-	40	4	-	11	-	-	50	-	-	-	6	1	0	-	-	-	-	-	-
25	52	41	2	6	-	14	-	34	-	-	-	24	-	37	-	-	0	0	-	-	-	-	-
43	31	29	0	21	-	28	-	-	2	-	-	7	-	17	-	-	-	0	0	-	-	-	-
20	33	48	-	4	13	-	26	-	-	22	-	-	46	42	-	-	-	-	0	0	-	-	-
45	7	18	51	12	25	-	-	-	50	-	-	5	-	-	-	0	-	-	-	0	0	-	-
35	40	32	16	5	-	-	18	-	-	43	51	-	32	-	-	-	-	-	-	-	0	0	-
9	24	13	22	28	-	-	37	-	-	25	-	-	52	-	13	-	-	-	-	-	-	0	0
32	22	4	21	16	-	-	-	27	28	-	38	-	-	-	8	1	-	-	-	-	-	-	0

(b) Code length 1,296 bits and code rate 2/3



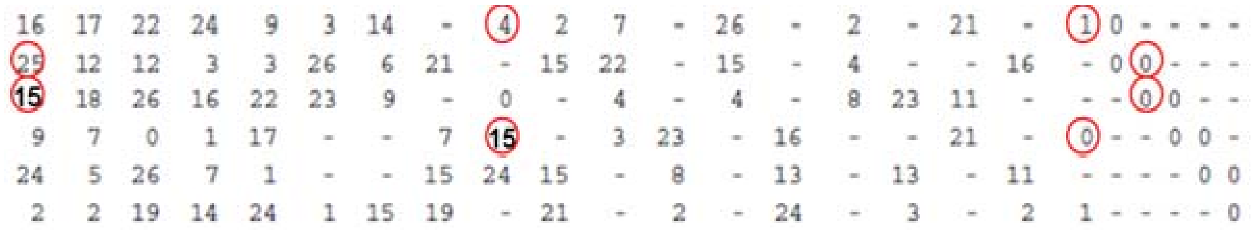
61	75	4	63	56	-	-	-	-	-	-	8	-	2	17	25	1	0	-	-	-	-	-	-
56	74	77	20	-	-	-	64	24	4	67	-	7	-	-	-	-	0	0	-	-	-	-	-
28	21	68	10	7	14	65	-	-	-	23	-	-	-	75	-	-	-	0	0	-	-	-	-
48	38	43	78	76	-	-	-	-	5	36	-	15	72	-	-	-	-	-	0	0	-	-	-
40	2	53	25	-	52	62	-	20	-	-	44	-	-	-	-	0	-	-	-	0	0	-	-
69	23	64	10	22	-	21	-	-	-	-	-	68	23	29	-	-	-	-	-	-	0	0	-
12	0	68	20	55	61	-	40	-	-	-	52	-	-	-	44	-	-	-	-	-	-	0	0
58	8	34	64	78	-	-	11	78	24	-	-	-	-	-	58	1	-	-	-	-	-	-	0

(c) Code length 1,944 bits and code rate 2/3

Figure 3.20 Parity check matrices of WLAN codes [IEEE09] containing short-four-cycles

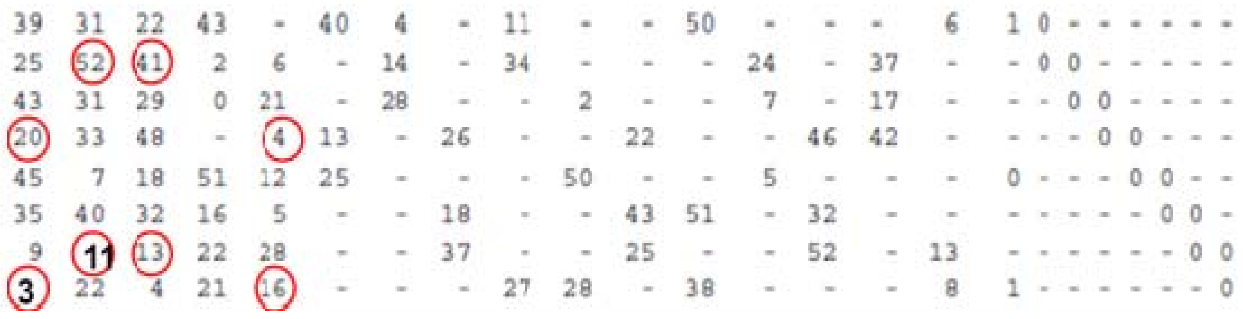
The objective is to compare the performance of the corresponding original WLAN codes with those codes without and with more cycles of four-short-cycles. Two kinds of modification of the WLAN codes containing four-short cycles are conducted. In the first modification, their four-short cycles are alleviated by changing the slope parameters of the affected submatrices. In the second modification, more four-short cycles are introduced to the parity check matrices by also changing the slope parameters of other submatrices. Either the original WLAN codes or the modified ones are simulated over AWGN channel at the maximal iteration 20.

As shown in Figure 3.21, in the first modification, the change of slope parameter takes place several submatrices (indicated by bold number). The selection of the slope parameter is arbitrarily with the condition that no short-four-cycles are included. In case of the length-648, rate-3/4 WLAN code, the modification of the slope parameter takes place in the first and ninth column. In case of the length-1296, rate-2/3 WLAN code, the modification of slope parameter takes place in the first and second column. In case of the length-1944, rate-2/3 WLAN code, the modification of slope parameter takes place in the first column.



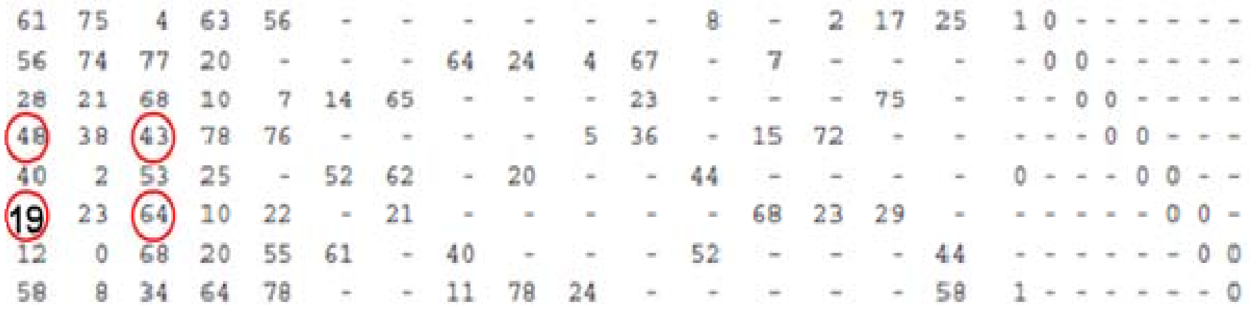
16	17	22	24	9	3	14	-	4	2	7	-	26	-	2	-	21	-	1	0	-	-	-	-
25	12	12	3	3	26	6	21	-	15	22	-	15	-	4	-	-	16	-	0	0	-	-	-
15	18	26	16	22	23	9	-	0	-	4	-	4	-	8	23	11	-	-	-	0	0	-	-
9	7	0	1	17	-	-	7	15	-	3	23	-	16	-	-	21	-	0	-	-	0	0	-
24	5	26	7	1	-	-	15	24	15	-	8	-	13	-	13	-	11	-	-	-	-	0	0
2	2	19	14	24	1	15	19	-	21	-	2	-	24	-	3	-	2	1	-	-	-	-	0

(a) Code length 648 bits and code rate 3/4 without short-four-cycles



39	31	22	43	-	40	4	-	11	-	-	50	-	-	-	6	1	0	-	-	-	-	-	-
25	52	41	2	6	-	14	-	34	-	-	-	24	-	37	-	-	0	0	-	-	-	-	-
43	31	29	0	21	-	28	-	-	2	-	-	7	-	17	-	-	-	0	0	-	-	-	-
20	33	48	-	4	13	-	26	-	-	22	-	-	46	42	-	-	-	-	0	0	-	-	-
45	7	18	51	12	25	-	-	-	50	-	-	5	-	-	-	0	-	-	-	0	0	-	-
35	40	32	16	5	-	-	18	-	-	43	51	-	32	-	-	-	-	-	-	-	0	0	-
9	11	13	22	28	-	-	37	-	-	25	-	-	52	-	13	-	-	-	-	-	-	0	0
3	22	4	21	16	-	-	-	27	28	-	38	-	-	-	8	1	-	-	-	-	-	-	0

(b) Code length 1296 bits and code rate 2/3 without short-four-cycles



61	75	4	63	56	-	-	-	-	-	-	8	-	2	17	25	1	0	-	-	-	-	-	-
56	74	77	20	-	-	-	64	24	4	67	-	7	-	-	-	-	0	0	-	-	-	-	-
28	21	68	10	7	14	65	-	-	-	23	-	-	-	75	-	-	-	0	0	-	-	-	-
48	38	43	78	76	-	-	-	-	5	36	-	15	72	-	-	-	-	-	0	0	-	-	-
40	2	53	25	-	52	62	-	20	-	-	44	-	-	-	-	0	-	-	-	0	0	-	-
19	23	64	10	22	-	21	-	-	-	-	-	68	23	29	-	-	-	-	-	-	0	0	-
12	0	68	20	55	61	-	40	-	-	-	52	-	-	-	44	-	-	-	-	-	-	0	0
58	8	34	64	78	-	-	11	78	24	-	-	-	-	-	58	1	-	-	-	-	-	-	0

(c) Code length 1944 bits and code rate 2/3 without short-four-cycles

Figure 3.21 Modified parity check matrix of IEEE802.11n codes without short-four-cycles

As shown in Figure 3.22 the alleviation of short four-cycles improves the BER performance. The significant improvement of code performance takes place in the length-1944, rate-2/3 WLAN code. At the other codes, the alleviation of short four cycles improves slightly.

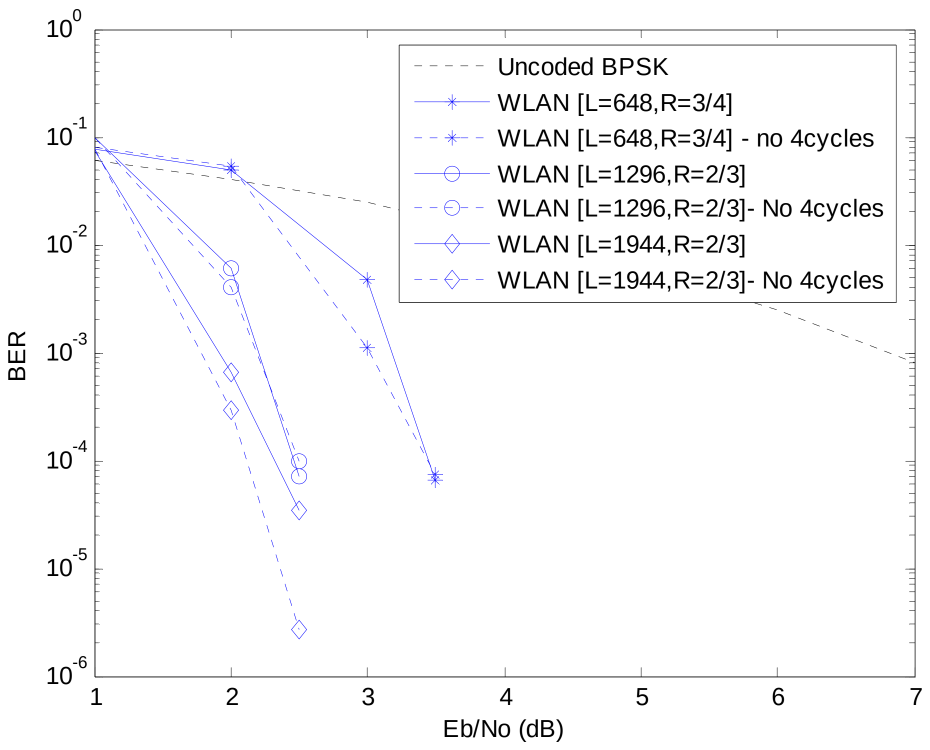
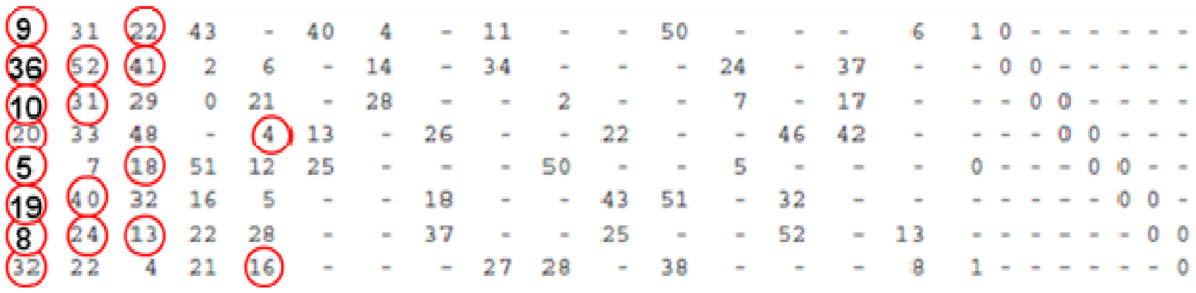


Figure 3.22 Comparison of the WLAN codes with and without the short-four-cycles

The impact of short four-cycles with higher girth degree is investigated. For this purpose, as representative, the length-1296, rate-2/3 WLAN codes are modified such as that their girth degree of short four-cycles (girth 4) becomes higher. For this purpose, the girth degree is increased to 4. The modification of the slope parameter can be realized in the Figure 3.23. As the result, the impact of girth degree on BER performance of the codes can be considered as negligible as depicted in Figure 3.24. In this case, the assigned level of girth degree can be considered as still not harmful enough to degrade the code performance. However, up to a certain extent, higher girth degree can degrade the BER performance significantly as shown in Chapter 4.



9	31	22	43	-	40	4	-	11	-	-	50	-	-	-	6	1	0	-	-	-	-	-	-
36	52	41	2	6	-	14	-	34	-	-	-	24	-	37	-	-	0	0	-	-	-	-	-
10	31	29	0	21	-	28	-	-	2	-	-	7	-	17	-	-	-	0	0	-	-	-	-
20	33	48	-	4	13	-	26	-	-	22	-	-	46	42	-	-	-	-	0	0	-	-	-
5	7	18	51	12	25	-	-	-	50	-	-	5	-	-	-	0	-	-	-	0	0	-	-
19	40	32	16	5	-	-	18	-	-	43	51	-	32	-	-	-	-	-	-	-	0	0	-
8	24	13	22	28	-	-	37	-	-	25	-	-	52	-	13	-	-	-	-	-	-	0	0
32	22	4	21	16	-	-	-	27	28	-	38	-	-	-	8	1	-	-	-	-	-	-	0

Figure 3.23 Modified parity check matrix of WLAN codes (code length 1296 bits and code rate 2/3) with higher short-four-cycles

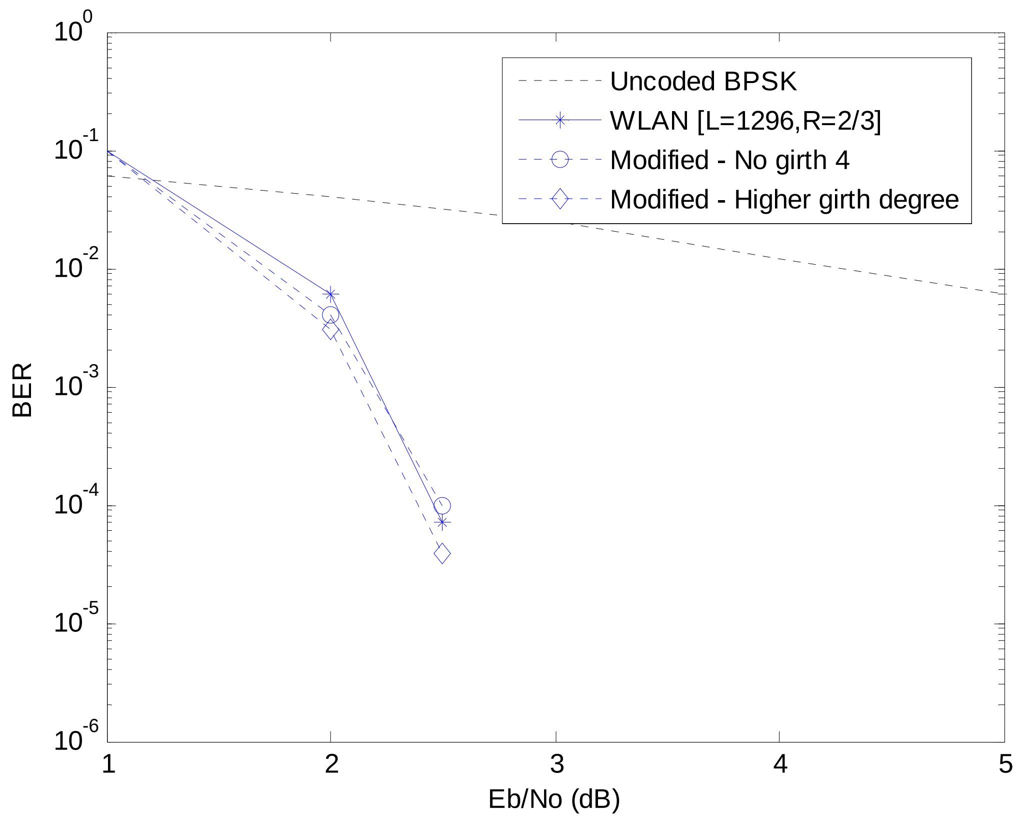


Figure 3.24 Impact of girth degree on BER performance of the length-1944, rate-2/3 codes

3.4 Conclusion

In this chapter, a novel concept for LDPC codes, the concept of girth degree, has been presented. The concept is used to evaluate LDPC codes by characterizing the codes in terms of their shortest cycles (girth) passing through the variable nodes in order to realize their impact on code performance. The concept considers the number of girth cycles (or girth degree) of all variable nodes and their distribution as well.

In this concept, two algorithms are proposed to characterize the girth degree of LDPC codes. The girth detection algorithm is used for detecting the existence of girths in the parity-check matrix or the corresponding bipartite graph, and the girth degree counting algorithm is applicable for counting the number of girth cycles passing through the variable nodes.

Using the concept of girth degree, some types of LDPC codes are characterized in terms of its girth degree distribution. In case of random MacKay codes, the concept shows that the girth increases in average as the code length gets higher. Meanwhile, the girth degree decreases in average, correspondingly. Their girth degree distribution becomes smaller and more concentrated to lower girth degree.

It is well-known in the theory of LDPC codes that the increasing code length leads to better code performance. The increasing girth as well as decreasing girth degree could be considered as to contribute to the better code performance because the increasing girth as well as the decreasing girth degree guarantees the higher bit independency in iterative decoding, which leads to a more effective and better decoding process.

In addition to that, the concept of girth degree also shows that the girth of random MacKay codes decreases in average as the code rate increases. Meanwhile, their girth degree gets higher in average, correspondingly. It is proven that LDPC codes at higher code rate have poorer code performance. The reason for this could be correlated with the decreasing average girth as well as the increasing average girth degree reducing the bit independency in iterative decoding, which leads to a worse decoding process.

The concept of girth degree is also used to characterize structured array LDPC codes. The array codes have fixed number of girth as well as girth degree in average as code length increases. Their girth degree distribution is also relative constant. This girth condition of the array codes may lead to poorer performance in comparison with random MacKay codes.

The WLAN codes are also characterized using the concept of girth degree. It is of interest that some WLAN codes contain short-four-cycles. Although those cycles exist, the corresponding WLAN codes still perform well. It can be considered as a proof, that the existence of short-four-cycles to some extent is not harmful enough to significantly degrade the code performance.

Moreover, the girth degree shows its efficiency as a measure for code performance on contrary to average girth. The average girth degree gets higher as code rate increases, which in turn, this leads to worse code performance. This fact shows that the number of cycles of girth plays a more significant role than girth in the determination of code performance.

CHAPTER 4

CODE CONSTRUCTION WITH STAIR STRUCTURE

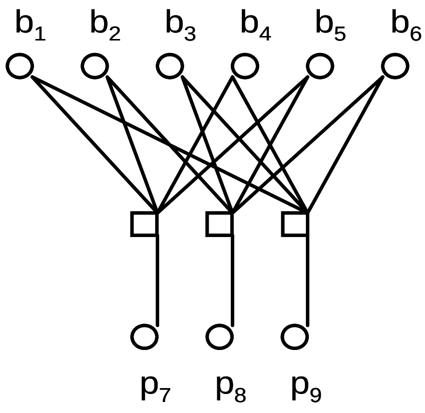
4.1 Background

In this chapter, a class of LPDC codes with very low encoding complexity and simple code construction is introduced. The codes are generated by only cyclic-shift registers in parallel processing. These codes are a trivial variant of LDPC staircase suggested in [KN95]. However, instead of dual-diagonal structure, the idea of LDGM structure that uses an identity matrix to ease encoding in linear time [OM01] is applied. Roca and Neumann [RN04] shows that the staircase structure makes encoding/decoding for applications with bandwidths of several hundreds of Mbps possible, which in turn, makes it appropriate to communications over high-speed networks. Some researchers are interested to employ the class of LDPC codes with staircase structure as Forward Error Correction (FEC) codes for reliable multicast data transport in internet [RNF06].

The parity check matrix of LDPC staircase code makes use of the identity matrix for the $\mathbf{I}_{n\text{-}k}$ matrix, in which its ones slope has staircase structure. The parity check matrix is divided into two parts, namely the left side of the matrix, which defines equations involving the source symbols and the right side of the matrix, which defines equations involving the repairing (parity) symbols. Figure 4.1 depicts an example of the parity-check matrix of LDPC staircase codes and its associated bipartite graph. The box nodes and round nodes denote the check and variable nodes, respectively. The way they are connected is defined by matrix **H**. In the bipartite graph, the connection is represented by the edges connecting variable nodes to a check node.

$$\left[\begin{array}{cccccc:ccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & p_7 & p_8 & p_9 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

(a)



(b)

Figure 4.1 LDPC staircase codes: parity-check matrix (a) and its corresponding graph (b)

The n-bit packet generated by the LDPC encoder is composed of the k-bit source packet and the (n-k)-bit parity packet. In Figure 4.1, the k source packets constitutes the k variable (information) nodes **b**, while the (n-k)-bit parity packet constitutes the (n-k) variable nodes **p**. The check nodes represent relationships between the variable nodes. The relationship constitutes a set of linear equations involving the value of the variable nodes. The staircase structure makes it possible to produce the parity packet directly from the relationship between the check node and its associated source node as follows.

$$\begin{aligned} p_7 &\equiv b_1 \oplus b_2 \oplus b_4 \oplus b_5 \\ p_8 &\equiv b_2 \oplus b_3 \oplus b_5 \oplus b_6 \\ p_9 &\equiv b_1 \oplus b_3 \oplus b_4 \oplus b_6 \end{aligned}$$

In the staircase approach, all parity nodes are linked to exactly one check node. The decoding algorithm for LDPC stair codes can be derived by applying belief propagation [Gall62]. For this special case of LDPC codes there exist n-k parity nodes with node degree 1 and k bit source nodes corresponding to the systematic bits. Therefore, the messages propagated from the degree-1 parity bit node to the corresponding check node will be always just the a priori probability of the corresponding bit. The recovery of the systematic bit is more important in this case.

The proposed codes differ from LDGM codes by the type of permutation matrices and encoding technique. Instead of multiplying the sparse generator matrix with the vector composed of source packet, the encoding of the proposed codes are done directly from the parity-check matrix. The design of permutation matrices is proposed in combination with the staircase structure, which leads to extreme low encoding complexity without performance degradation [HEK07b]. The appropriate structure of the permutation matrices can be expected to lower the error floor [KN95].

4.2 Code Structure

In the proposed LDPC stair codes two kinds of structure of the systematic part of the parity check matrix are presented. The systematic matrix consists of several permutation matrices, which are arranged in different concatenation: cascade and lattice structures. Recall that a permutation matrix is a square matrix composed of 0's and 1's, with a single 1 in each row and column. The permutation matrix is considered circulant because the i-th row of the matrix is obtained by cyclically shifting the (i-1)-th row by one position to the right.

The identity matrix acts as reference for circulant permutation matrix (hereafter, shortly referred to as submatrix). Figure 4.2 depicts an identity matrix and a circulant permutation matrix with a singular cyclic shift

$$H_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad H_i = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) (b)

Figure 4.2 An identity matrix (a) and a singular cyclic shift submatrix (b)

In Figure 4.3 an important parameter to characterize a submatrix, namely, cyclic-shift number *s* is introduced. This parameter (hereafter, referred to as slope parameter) denotes how far the stair (hereafter, referred to as slope) of identity matrix is cyclic right-shifted. In case of the identity matrix and a single cyclic shift submatrix in Figure 4.2, the cyclic-shift number $s = 0$ and $s = 1$, respectively.

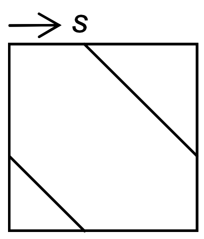
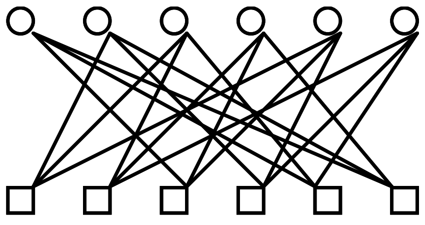


Figure 4.3 Submatrix with cyclic right-shifted slope *s*

A submatrix can consist of some slopes, which is resulting from correspondingly addition of some submatrices with one slope each. The number of slopes is associated with the number of column weight and row weight of the parity check matrix. Figure 4.4 shows the submatrix with three slopes and is characterized by its slope parameter $s = 1$, 2 and 4.

$$\begin{array}{cccccc} & s=1 & s=2 & & s=4 & \\ \hline 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{array}$$

(a)



(b)

Figure 4.4 Submatrix (a) with its corresponding graph (b)

To simplify the representation of the proposed codes, a submatrix with the series of the slope parameter of their submatrices in the systematic part of parity-check matrix can be represented in the following form

$[\, s_1 + s_2 + ... + s_m \,]$

where [...] = submatrix and s_m = the-m-th slope parameter. A submatrix containing only zero elements is represented by [-]. Serially concatenated submatrices are represented by [...][...], while parallel concatenated submatrices are separated by sign ; and represented by [...];[...]. For example, the submatrix illustrated in Figure 4.4 can be represented by its slope parameter as [1+2+4], while an identity matrix can be represented by [0]

4.2.1 Cascade Structure

The cascade structure is constructed as follows. The parity check matrix **H** is given by

$$\mathbf{H} = [\mathbf{P}\ \mathbf{I}] \tag{4.1}$$

where **I** denotes an identity matrix and **P** denotes p submatrices in cascaded format, which is given by

$$\mathbf{P} = [\mathbf{H}_1\ \mathbf{H}_2 \dots \mathbf{H}_p] \tag{4.2}$$

where $\mathbf{H}_i$ denotes the submatrix i, where $i = 1, 2 \dots p$.

Each submatrix has quadratic dimension $n/(t+1)$, where n is code length and t is the number of the submatrix. The code rate is $t/(t+1)$. The column weight and row weight correspond to the number of slopes applied in the submatrices. In the quasi-regular case, where the number of slopes in each submatrices is equal, the column weight in the left submatrices and in the stair matrix is $t \times n_s$ and 1 in the right part, respectively, and the row weight is $t \times n_s + 1$, where n_s is the number of slopes in a submatrix.

For example, a quasi-regular LDPC code is constructed with code rate 3/4 as shown in Figure 4.5. The code consists of three submatrices and an identity matrix. The code has row-weight 10 and column-weight 3 and 1 in systematic matrix and the identity matrix, respectively. The corresponding parity check matrix is depicted in Figure 4.5.

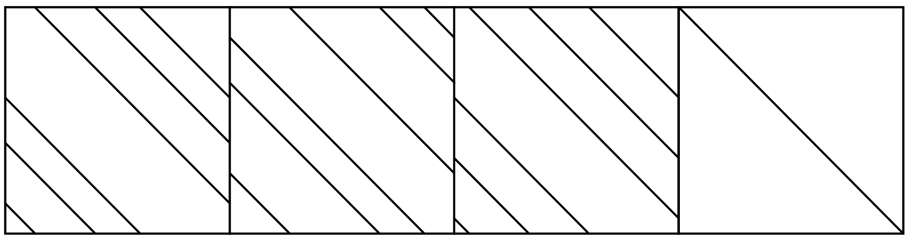


Figure 4.5 Cascade structure: parity check matrix

4.2.2 Lattice Structure

The parity check matrix **H** of the lattice structure is the same as of the cascade one, but with different construction of the matrix **P**, which denotes $p \times q$ submatrices in lattice format. The transpose of matrix **P** is given by

$$\mathbf{P}^{\mathrm{T}} = [\mathbf{H}_1\ \mathbf{H}_2\ \ldots\ \mathbf{H}_q] \tag{4.3}$$

$\mathbf{H}_i$ is then partitioned into p submatrices.

$$\mathbf{H}_i = [\mathbf{H}_{i1}\ \mathbf{H}_{i2}\ \ldots\ \mathbf{H}_{ip}] \tag{4.4}$$

where $\mathbf{H}_{ij}$ denotes the submatrix ij for $i = 1, 2 \ldots q$ and $j = 1, 2 \ldots p$.

Each submatrix has quadratic dimension $m_i \times m_i$, with $m_i = m/i$, where i is number of submatrices in vertical arrangement. The code rate is $t/(t+i)$, where t is number of submatrices in horizontal arrangement. The column weight and row weight correspond with the number of submatrices in vertical and horizontal, respectively. In the quasi-regular case, the column weight in the left submatrices is $t \times n_s$ and 1 in the right part, respectively, and the row weight is $i \times n_s + 1$, where n_s is number of slopes in a submatrix.

In this structure the parity-check consists of several smaller submatrices with an identity matrix composed of a number of identity submatrices in its diagonal. Figure 4.6 depicts the parity check matrix for a quasi-regular LDPC code with code rate 3/4. The code has row-weight 10 and column-weight 3 and 1 in the systematic matrix and the identity matrix, respectively.

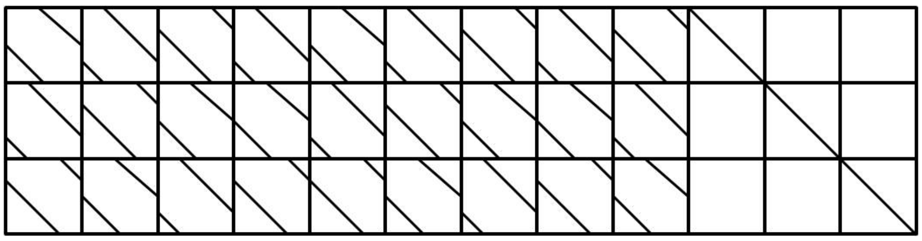


Figure 4.6 Lattice structure: parity check matrix

The lattice structure can be obtained by horizontally appending concatenated submatrices at the cost of a prolonged code length. If the code length shall be held, a submatrix is divided into smaller ones in quadratic form. Figure 4.7 shows a cascade submatrix is broken into four and nine smaller quadratic submatrices (or lattice-4 and lattice-9, respectively).

Based on this idea, the lattice structure can be seen as a derivation of the cascade structure. The concatenation way of lattice structure makes it more flexible to cope with any code length in comparison with cascade structure. However, lattice structure is more complicated in the implementation of the encoder as described later.

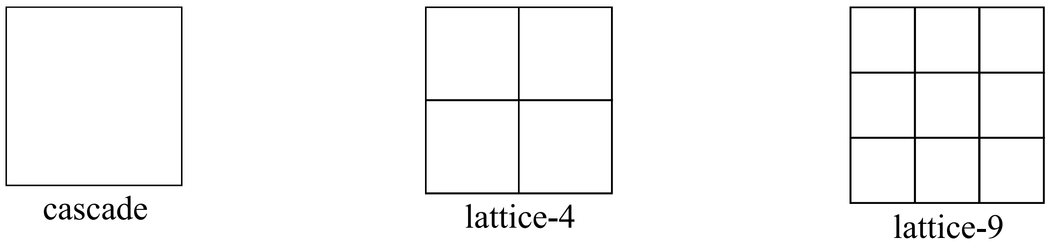


Figure 4.7 Cascade and lattice submatrices

4.3 Construction Methods

The proposed codes are constructed such that most possibly no short four-cycles exist in the bipartite graph as illustrated in Figure 4.8. The presence of such cycles has to be avoided to cope with degradation in decoding performance. However, to some extent, their existence is not harmful enough to degrade code performance [TJVW04, ZZ04, ZP01d].

The construction method can be simply done in two ways. The first step is to generate the value of the slope parameter using random permutation process. Then, the values of the slope parameter are examined using two methods, namely slopes method and girth detection method. If the requirement that no short-four-cycles found is not met, the permutation process of the slope parameter is repeated. In case of high code rate or dense column weight in systematic matrix, this requirement is hardly to meet. However, the existing of short-four-cycles does not inhibit codes to perform well as described later.

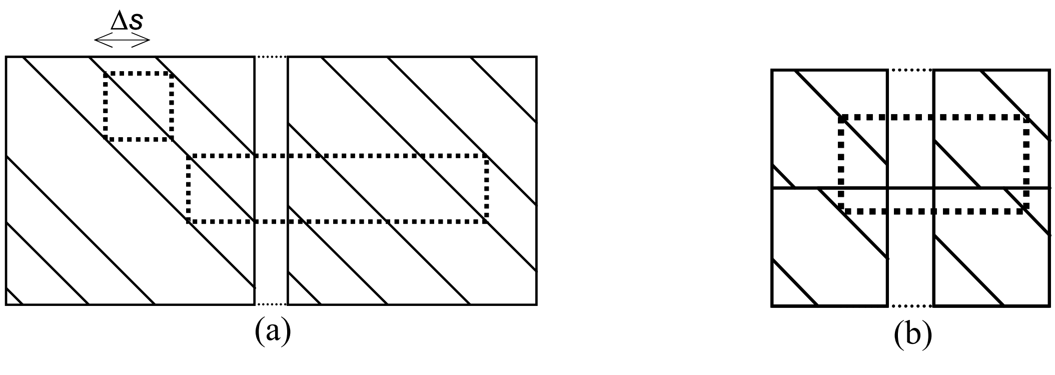


Figure 4.8 Presence of four-cycles in the cascade (a) and lattice structure (b)

4.3.1 Slopes Method

The slopes method takes two parameters into consideration, namely, the slope parameters s and so-called slope distance Δs. The slope distance is here defined as the distance between any two slopes, either in a submatrix or in any two submatrices. The slope method gives the principal rule to guarantee that no short-four-cycles exist in the systematic matrix of the parity check matrix of the proposed codes.

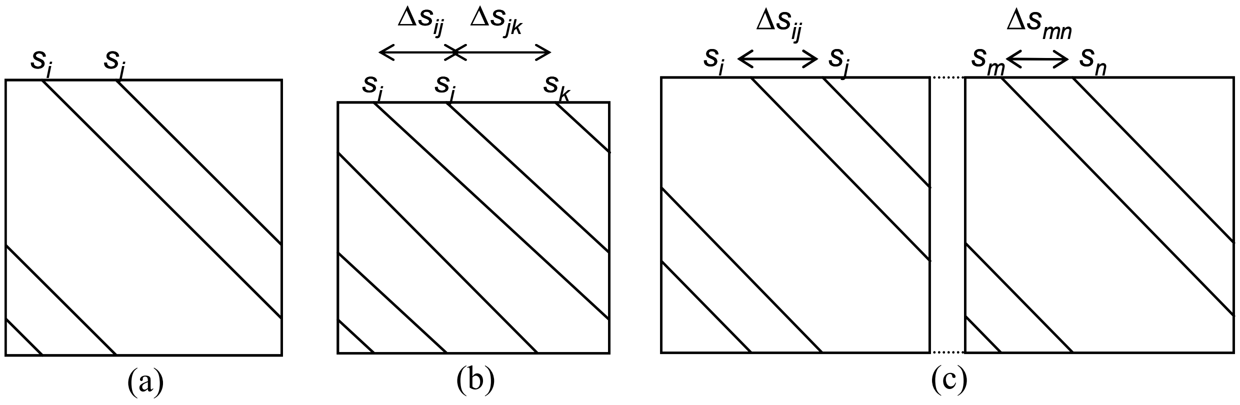


Figure 4.9 Parameters in the slope method

The principal rules of the slope method can be described as follows

1. Constructing a submatrix by determining shift parameter s
   a. between slope and submatrix dimension

$$s_i \neq p/2 \tag{4.5}$$

   b. between two slopes (see figure 4.9a):

$$s_i \neq ½\,(2.s_j \bmod p) \tag{4.6}$$

   c. between more than two slopes (see Figure 4.9b):

$$s_i \neq ½\,((s_j + s_k) \bmod p) \tag{4.7}$$
$$\Delta\, s_{ij} + \Delta s_{ik} \neq p \tag{4.8}$$

   d. between two slope distance:

$$\Delta s_{ij} \neq \Delta s_{jk} \tag{4.9}$$

   where $\Delta s_{ij} = |s_i\text{-}s_j|$, $\Delta s_{jk} = |s_j\text{-}s_k|$, $s_i < s_j < s_k$ and $p{\times}p$ is submatrix dimension

2. Putting all permutation matrices together and consider the condition between two slope distances in different matrices (see Figure 4.9c):

$$\Delta s_{ij} \neq \Delta s_{mn} \tag{4.10}$$
$$\Delta s_{ij} + \Delta s_{mn} \neq p \tag{4.11}$$

   where $\Delta s_{ij} = |s_i\text{-}s_j|$ and $\Delta s_{mn} = |s_m\text{-}s_n|$

In case of lattice structure, an additional principal rule must be taken into account. The slope method has to assure that no pair of slopes with the submatrix construction illustrated in Figure 4.10 has the same slope distance Δs. To achieve this objective, the following condition has to be held

$$\Delta s_{ik} = \mathrm{mod}(\, \Delta s_{mo}, p) \tag{4.12}$$

where $\Delta s_{ik} = |s_{ij}\text{-}s_{kl}|$ and $\Delta s_{mo} = |s_{mn}\text{-}s_{op}|$

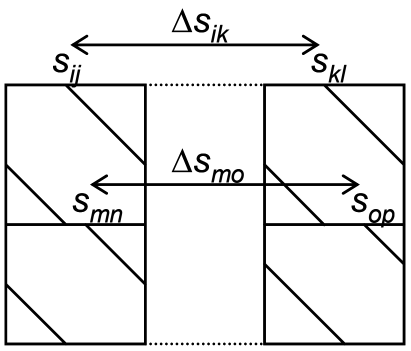


Figure 4.10 A pair of slope distance

As illustrated in Figure 4.11, the method of code construction using the slope method can be simply described as follows:

Step 1. Determine the code parameters: code rate, code length and column weight
Step 2. Determine the type of code structure type: lattice or cascade
Step 3. Determine the number of submatrices and slopes referring to the code parameters
Step 4. Determine the slope parameter in submatrices randomly using permutation process
Step 5. Check if short four-cycles exists using slope method
Step 6. If short four cycles, go back to step 4, otherwise done

4.3.2 Girth Degree Detection Method

The proposed codes can be also constructed without the presence of four-cycle by making use of the girth degree detection method as described in the previous chapter. As illustrated in Figure 4.12, the construction method can be simply described as follows:

Step 1. Determine the code parameters: code rate, code length and column weight
Step 2. Determine the type of code structure type: lattice or cascade
Step 3. Determine the number of submatrices and slopes referring to the code parameters
Step 4. Determine the slope position in submatrices randomly using permutation process

Step 5. Check if short four cycles exist using girth degree detection method
Step 6. If short four cycles, go back to step 4, otherwise done

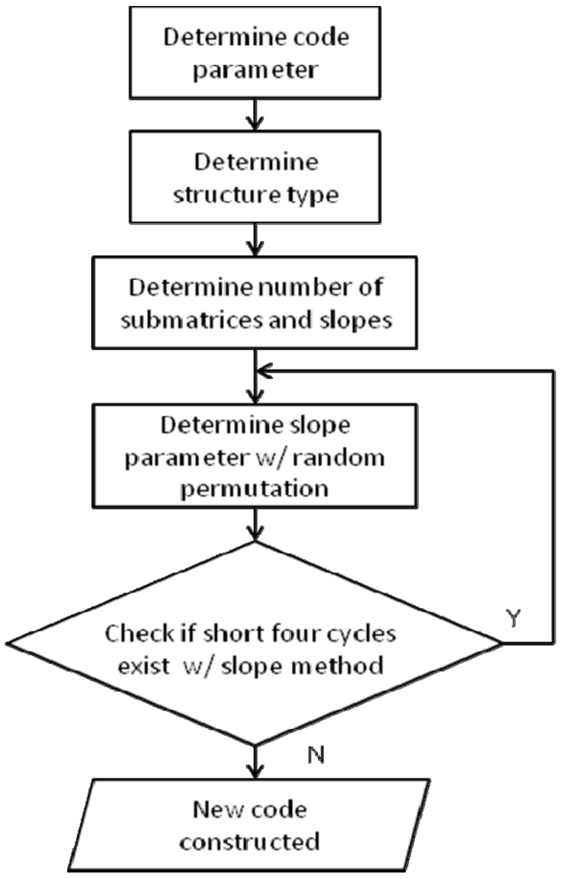

Determine code parameter
Determine structure type
Determine number of submatrices and slopes
Determine slope parameter w/ random permutation
Check if short four cycles exist w/ slope method
Y
N
New code constructed

Figure 4.11 Algorithm of code construction with slope method

4.4 Encoder/Decoder Design

4.4.1 Encoder Design

a. Cascade Structure

The structure of such parity check matrix makes its implementation very simple. The encoder can be derived analytically from the syndrome equation

$$\mathbf{H}.\mathbf{c}^T = \mathbf{0} \tag{4.12}$$

Codeword **c** is partitioned into a number of subblock, which corresponds to the partition of the parity check matrix **H**

$$\mathbf{c} = [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_p\ \mathbf{p}] \tag{4.13}$$

where $\mathbf{u}_i$ for $i = 1, 2, \ldots p$ is sub-block of information bits and p is block of parity bits.

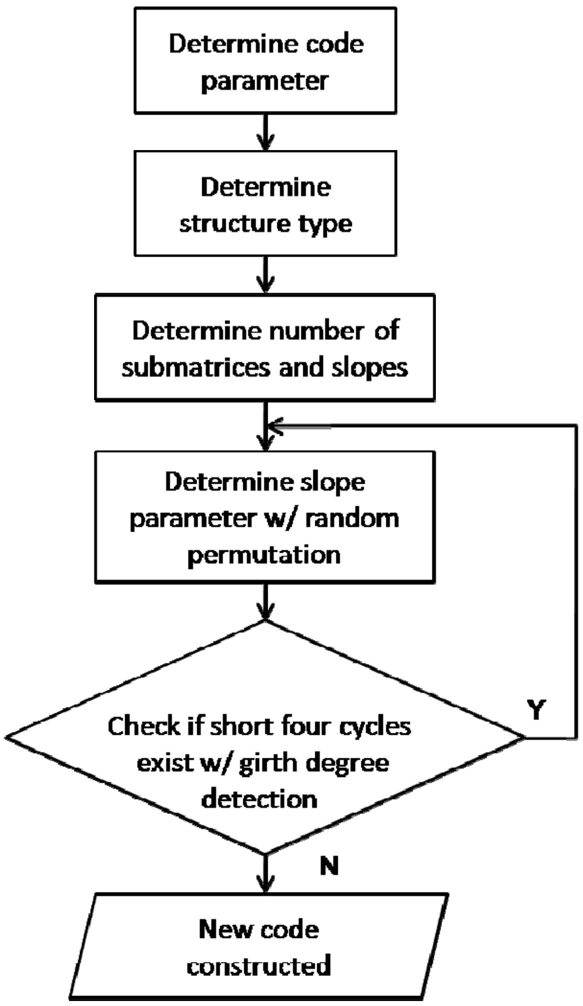

Determine code parameter
Determine structure type
Determine number of submatrices and slopes
Determine slope parameter w/ random permutation
Check if short four cycles exist w/ girth degree detection
Y
N
New code constructed

Figure 4.12 Algorithm of code construction with girth degree detection method

From the equation (4.12) and (4.13), a sum-product equation is obtained as follow

$$\mathbf{H}_1\,\mathbf{u}_1^T + \mathbf{H}_2\,\mathbf{u}_2^T + \ \ldots + \mathbf{H}_P\,\mathbf{u}_P^T\ \ldots. \ + \mathbf{I}\,\mathbf{p}^T = 0 \tag{4.14}$$

By considering $\mathbf{p} = \mathbf{I}.\mathbf{p}^T$, the vector of parity bits results in

$$\mathbf{p}^{\mathrm{T}} = \mathbf{H}_1 \mathbf{u}_1^{\mathrm{T}} + \mathbf{H}_2 \mathbf{u}_2^{\mathrm{T}} + \ldots + \mathbf{H}_p \mathbf{u}_p^{\mathrm{T}} \tag{4.15}$$

This equation can be simply implemented by parallel circuit of cyclic-shift registers. Furthermore, the work is focused on analytical operation in a submatrix. To simplify the description, only a slope containing non-zero entries in a submatrix is presented in Figure 4.13

$$H_i = \begin{bmatrix} \ldots\ldots\ldots \;\; \ldots\; h_{1(1+s)} \ldots\ldots\ldots \;. \\ \ldots\ldots\ldots \;\; \ldots\ldots\; h_{2(2+s)} \;\ldots\ldots \\ \ldots\ldots\ldots \;\; \ldots\ldots\ldots \;\; \ldots\ldots \vdots \ldots\ldots \\ \ldots\ldots\ldots \;\; \ldots\ldots\ldots \;\; \ldots\; h_{(m-s)m} \\ h_{(m-s+1)1} \;\ldots\ldots\ldots \;\; \ldots\ldots\ldots \;\; \ldots \\ ..\; h_{(m-s+2)2} \;\ldots\ldots\ldots \;\; \ldots\ldots\ldots \\ \ldots\ldots\ldots \;\; ..\vdots\ldots\ldots\ldots \;\; \ldots\ldots\ldots \;\; .. \\ \ldots\ldots\ldots \;.\; h_{ms} \;\ldots\ldots\ldots \;\; \ldots\ldots \end{bmatrix}$$

Figure 4.13 Submatrix

where s is cyclic-shift number and $m \times m$ is dimension of submatrix. By multiplying submatrix with information sub-block, the parity bit can be obtained

$$p_{\mathrm{i}} = h_{\mathrm{i}(i+s)} . u_{(i+s)} \tag{4.16}$$

Due to $h_{ij} = 1$, no multiplication is necessary.

$$p_{\mathrm{i}} = u_{(\mathrm{i}+s)} \tag{4.17}$$

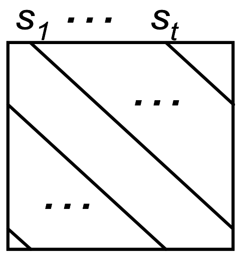
Due to cyclic operation, the parity bit is as follow

$$p_i = \begin{cases} u_{(i+s)} & , i \le m - s_j \\ u_{(i+s-m)} & otherwise \end{cases} \tag{4.18}$$

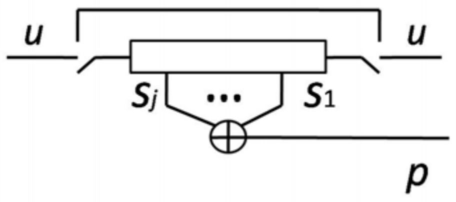
The parity bit of row i of a parity check matrix containing p submatrices with j slopes is obtained by

$$p_i = \sum_P \sum_j u_{i+s_j} \tag{4.19}$$

Figure 4.14a shows a permutation matrix containing t slopes with cyclic-shift numbers s_j, where $j = 1, 2 \ldots t$.



(a)



(b)

Figure 4.14 Cascade structure: submatrix (a) and its cyclic-shift register (b)

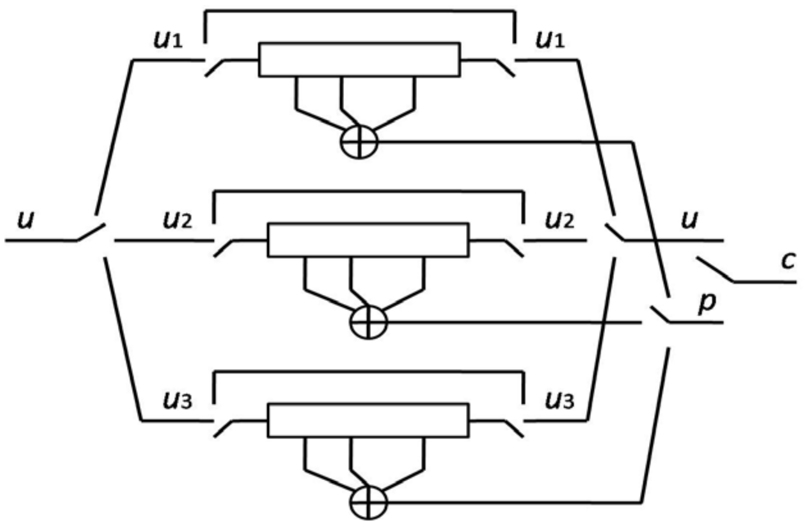


Figure 4.15 Implementation of the encoder of cascade structure

This permutation matrix is realized by a cyclic-shift register as depicted in Figure 4.14b. The cyclic-shift register of length n is initially occupied by information bits **u**. After m cycles the parity bits **p** is generated and the information bits **u** are released.

For example, for the parity check matrix of the 3/4-rate code with cascade structure and column weight 3 three cycle-shift registers connected by addition operation are necessary to calculate all parity bits. The parity bits **p** is appended to the information bits **u** = [$\mathbf{u}_1$ $\mathbf{u}_2$ $\mathbf{u}_3$] to build a valid codeword **c** as shown in Figure 4.15.

b. Lattice Structure

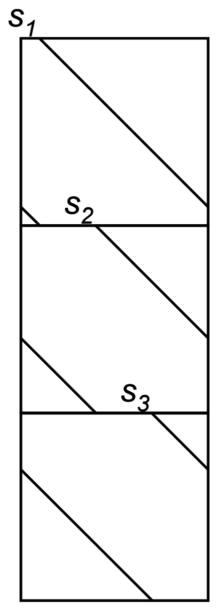
The encoder for this structure can be derived analytically from the equation (4.12)-(4.15). It differs from the cascade structure by number of permutation matrices. A lattice matrix contains of q permutation matrices: $\mathbf{H}_\mathrm{j} = [\mathbf{H}_{1j}\ \mathbf{H}_{2j}\ \ldots\ \mathbf{H}_{qj}]$. The packet is again partitioned accordingly for $\mathbf{u}_\mathrm{i}$: $\mathbf{u}_\mathrm{i} = [\mathbf{u}_{\mathrm{i}1}\ \mathbf{u}_{\mathrm{i}2}\ \ldots\ \mathbf{u}_{ip}]$ and $\mathbf{p} = [\ \mathbf{p}_1\ \mathbf{p}_2\ \ldots\ \mathbf{p}_q\ ]$.

$$\mathbf{p}_i^\mathrm{T} = \mathbf{H}_{i1}\,\mathbf{u}_{i1}^\mathrm{T} + \mathbf{H}_{i2}\,\mathbf{u}_{i2}^\mathrm{T} + \ldots + \mathbf{H}_{ip}\,\mathbf{u}_{ip}^\mathrm{T} \quad ;\ \text{for } i = 1,2,..q \tag{4.20}$$

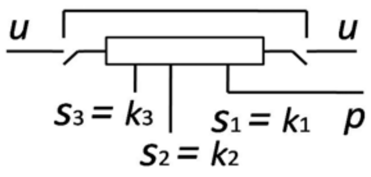
This equation can be simply implemented by parallel circuit of cyclic-shift registers, whose analytical operation is presented by the equation (4.16)-(4.19).

By this construction we break each cascade matrix into nine smaller lattice matrices or equivalently three concatenated permutation matrices. Each lattice matrix contains three permutation matrices as depicted in figure 4.16a. Each permutation matrix contains a slope with cyclic-shift numbers s_j, where j = 1, 2 and 3.

These concatenated matrices are realized by a cyclic-shift register as depicted in figure 4.16b. The cyclic-shift register of length n is initially occupied by information bits **u**. After m cycles the parity bits **p** is generated and the information bits **u** is output.



(a)



(b)

Figure 4.16 Lattice structure: submatrices (a) and its cyclic-shift register (b)

For the lattice structure it is necessary to have nine cycle-shift registers connected by addition operation to calculate all parity bits as shown in Figure 4.17. The parity bits **p** is appended to the information bits $\mathbf{u} = [\mathbf{u}_1\ \mathbf{u}_2\ \mathbf{u}_3]$ to build a valid codeword **c**.

c. Comparison of Both Code Structures

The comparison of cascade and lattice structure can be described using the slope method. Both structures are compared in terms of complexity and flexibility. Two types of complexity are introduced, that is, the complexity in design and in realization. Here, the complexity in design is meant to the number of equations involved in determination of slope parameter leading to parity check matrix. While, the complexity in realization is defined the number of shift registers needed in encoder circuit.

Basically, the complexity of the encoder depends on the number of slopes and the number of submatrices. Both correspond to column weight and code rate. In Table 4.1 the number of equation to check per a submatrix for each step of design of parity check matrix is presented. While, Table 4.2 shows number of equation needed for checking short four-cycles between submatrices.

Table 4.1 Number of equation (N_{eq}) depending on number of slopes per submatrix (N_{sl})

Step	**Number of slopes per submatrix**		
	3	**4**	**5**
1.a	3	4	5
1.b	3	6	10
1.c	2x1	2x4	2x10
1.d	1	4	10
2.a	2x3x3	2x6x6	2x10x10

Table 4.2 Number of equation (N_{eq}) for number of submatrix (N_{sm})

N_{sm}	N_{eq}
2	1
3	3
4	6
5	10

Now, both structures are compared in terms of complexity in terms of number of equation needed to determine the slope parameter and the number of shift register needed for realization. For comparison, both structures are constructed with column weight 3 at WLAN code rate, i.e. 1/2, 2/3 and 3/4. The number of equation needed by both structures is put in comparison in Table 4.3. In fact, more equations are necessary to check for constructing the lattice. Besides, the number of equation for the lattice rises in more significant rate as the code rate increases.

Table 4.3 Comparison of both structures in number of equation per code rate

Code Rate	Cascade	Lattice
1/2	9	9
2/3	36	45
3/4	54	108

In the realization, the number of submatrix for the lattice is three times larger than that for the cascade as shown in Table 4.4. This number also corresponds with the number of shift register needed for encoder circuit. In addition to that, the higher complexity in the implementation of the lattice encoder can be realized in the previous section.

Table 4.4 Comparison of both structures in number of submatrix per code rate

Code Rate	Cascade	Lattice
1/2	1	3
2/3	2	6
3/4	3	9

Here, this flexibility is meant to the feature of code to support code rate. In fact, the lattice support more types of code rate than the cascade, in which the latter only operates at code rate $t/t+1$, where t is positive integer.

4.4.2 Decoder Design

The sparseness of the parity check matrix plays an important role in decoding complexity. A non-zero entry corresponds to edge between a variable node and a check node in the bipartite-graph. The parity check matrix we are considering has column-weight 3 and 1 and row-weight 10. Therefore, this code has 3000 edges in the bipartite graph.

For decoding the code, the sum-product algorithm (SPA) is commonly used. In this algorithm, messages are exchanged iteratively between the variable and check nodes as summarized as follows:

Step 1. Initialize. The decoder calculates LLRs for received bits **y**.
Step 2. Message passing from variable nodes to check nodes
Step 3. Message passing from check nodes to variable nodes
Step 4. Make hard-decisions
Step 5. Repeat steps 2-4 until either of the following termination criteria are met; (1) if $\mathbf{H}.\mathbf{y}^T = \mathbf{0}$, then the iterations are complete. (2) Otherwise, stop after a fixed maximum number of iterations whether they are complete or not.

The sum-product algorithm enables LDPC code to be decoded parallelly. Some realization approaches for the parallel design were introduced in [HB01, KSM02]. The parallel design of the decoder [KSM02] is composed of two types of computational blocks, two interleavers, and two pipe-line registers (see Figure 4.18). The computational blocks represent the nodes that are used to compute the messages. The interleavers represent the edge structure of the bipartite graph. The pipeline registers are used to store the output of the computational blocks. The v_n blocks compute the message from variable nodes to check nodes (step 2), in which the LLRs are available as the initial value of messages. Their output are sent through *v-c* interleaver and then stored in the *v-c* pipeline registers. The c_n blocks compute the messages from check nodes to variable nodes (step 3). Their output is stored in the pipeline *c-v* registers. They are sent to the *c-v* interleaver and fed back into v_n blocks for other iteration.

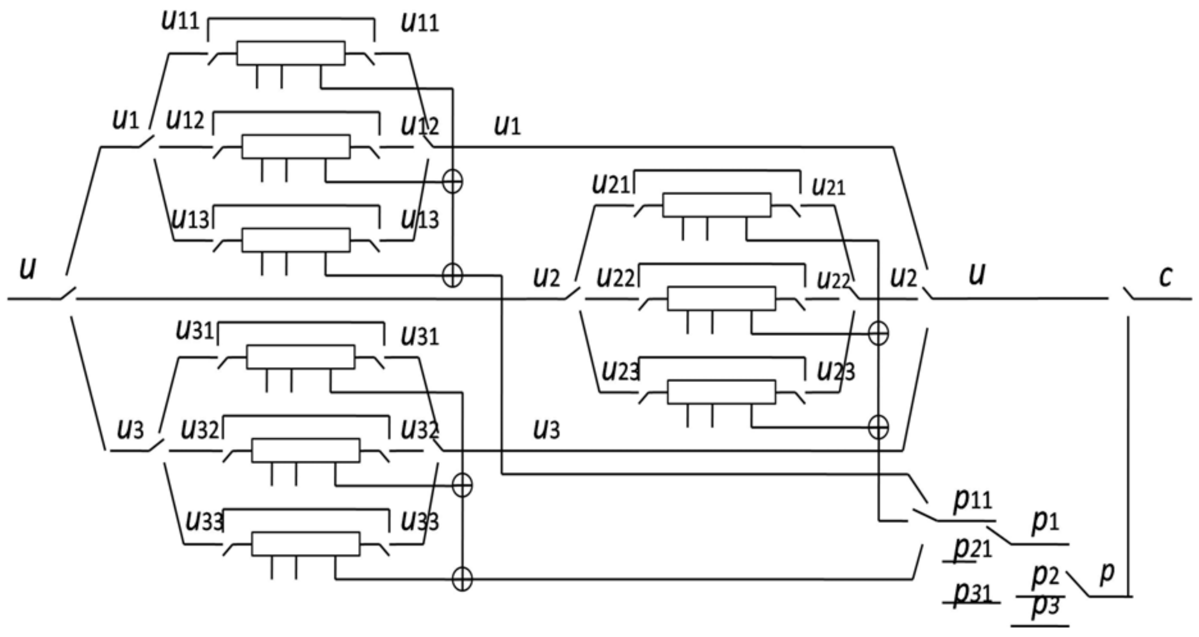


Figure 4.17 Implementation of the encoder of lattice structure

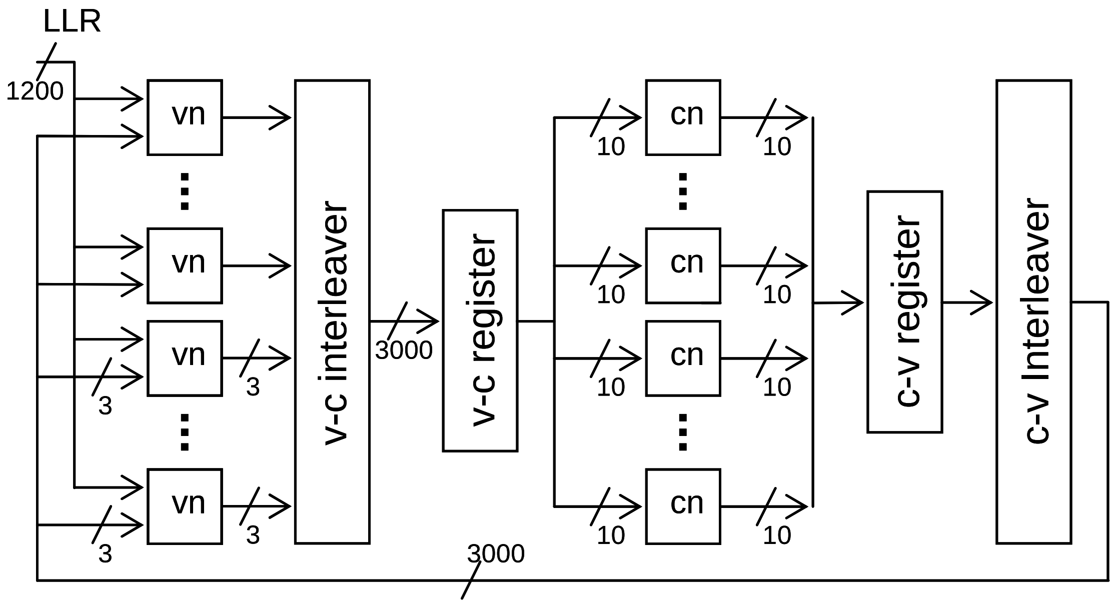


Figure 4.18 Block diagram of the decoder

4.5 Simulation Results

In this section, the proposed LDPC codes, including cascade structure and lattice structure (hereafter, denoted as Stair Cascade and Stair Lattice), are simulated, analyzed and put in comparison with some well-known benchmarking codes. For testing near-capacity performance of the codes, the simulation is conducted using BPSK modulation over additive white Gaussian noise (AWGN) channel. The sum-product decoding algorithm is applied. It is expected that the simulation can explore the characteristics of the codes and their weaknesses as well as their strength.

The simulation considers several important parameters of LDPC code and their impact on the code performance, including decoding iteration, column weight, code length, code rate. The performance of the proposed LDPC codes at high code rate and short code rate is also taken into account. Those parameters are set in sense of their practicability in wireless LAN application. Finally, the proposed LDPC codes are compared with some benchmarking codes, such as random MacKay codes [MN95] and also IEEE802.11n (WLAN) codes [IEEE09].

a. Cascade Structure

Maximal number of decoding iteration

The performance of the Stair Cascade codes at practical number of decoding iteration is firstly investigated. The maximal number of decoding iteration up to 20 is considered.

The simulated Stair Cascade code is designed with code length L = 1,200 bits, code rate R = 3/4 and column weight 6. The code has the following slope parameter

[146+203+285+193+298+1][104+121+55+281+184+124] ...
[169+154+215+247+208+165][0]

Figure 4.19 shows the impact of decoding iteration on the performance of the Stair Cascade codes. It is realized that the code still performs well in decoding with only maximal two iterations. However, better performance can still be achieved at higher decoding iteration. The simulation result shows that the code can achieve the optimal BER performance with the maximal decoding iteration 10. More number of decoding iteration brings only negligible contribution to the performance gain. Therefore, the maximal decoding iteration 10 is selected as basis for further investigation.

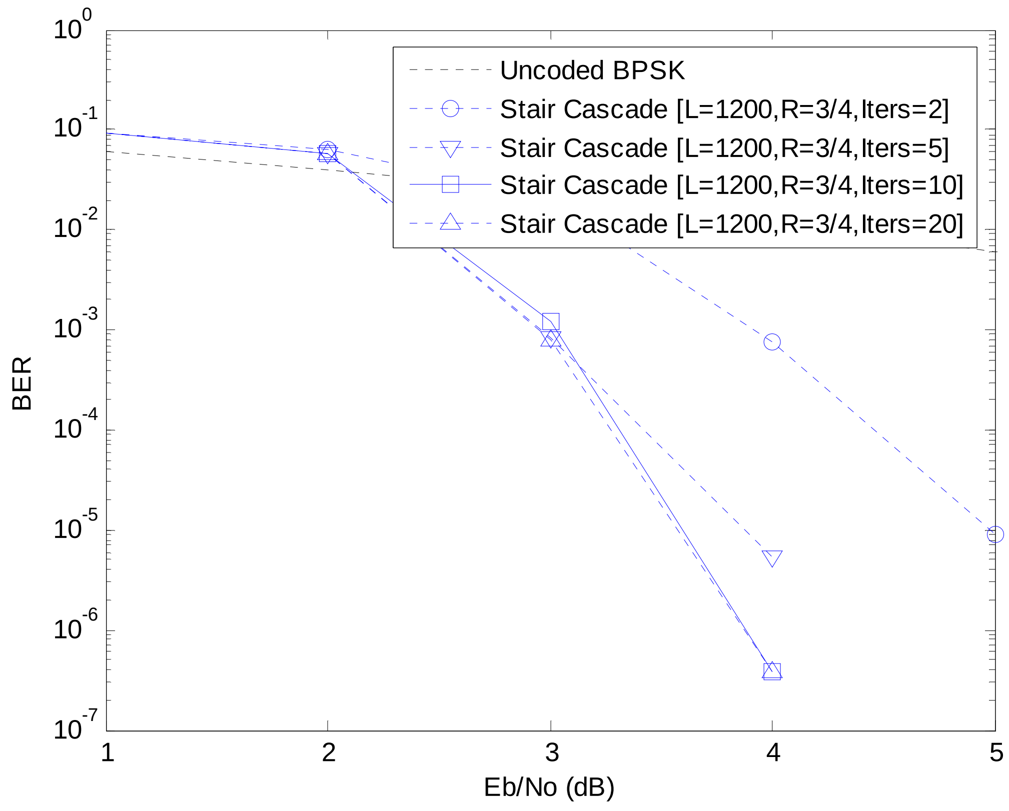


Figure 4.19 Impact of decoding iteration on the performance of the length-1200, rate-3/4 Stair Cascade code

Code rate

The performance of the Stair Cascade codes is simulated at three different code rates, i.e. 1/2, 2/3 and 3/4. The codes are constructed at code length 1,200 bits and column weight 6. Their slope parameters are as follows.

Code rate	Slope parameter
1/2	[545+216+589+518+382+94][0]
2/3	[109+311+190+87+325+38][83+377+199+157+238+282][0]
3/4	[146+203+285+193+298+1][104+121+55+281+184+124] ...
	[169+154+215+247+208+ 165][0]

Figure 4.20 shows the threshold performance of the Stair Cascade codes is subject to get worse as the code rate increases. However, at the other side, the error floor performance improves. The Stair Cascade code with code rate 2/3 can achieve a bit error rate of 10^{-5} at about 3 dB which about 0.5 dB better than the code with code rate 3/4. However, at higher signal-to-noise ratio, the Stair Cascade code with code rate 2/3 suffers from the higher error floor. Meanwhile, the performance of the Stair Cascade code with code rate 1/2 degrades significantly at higher signal-to-noise ratio due to poor error floor performance. The high number of variables being part of the identity matrix could be responsible for the high error floor at code rate 1/2.

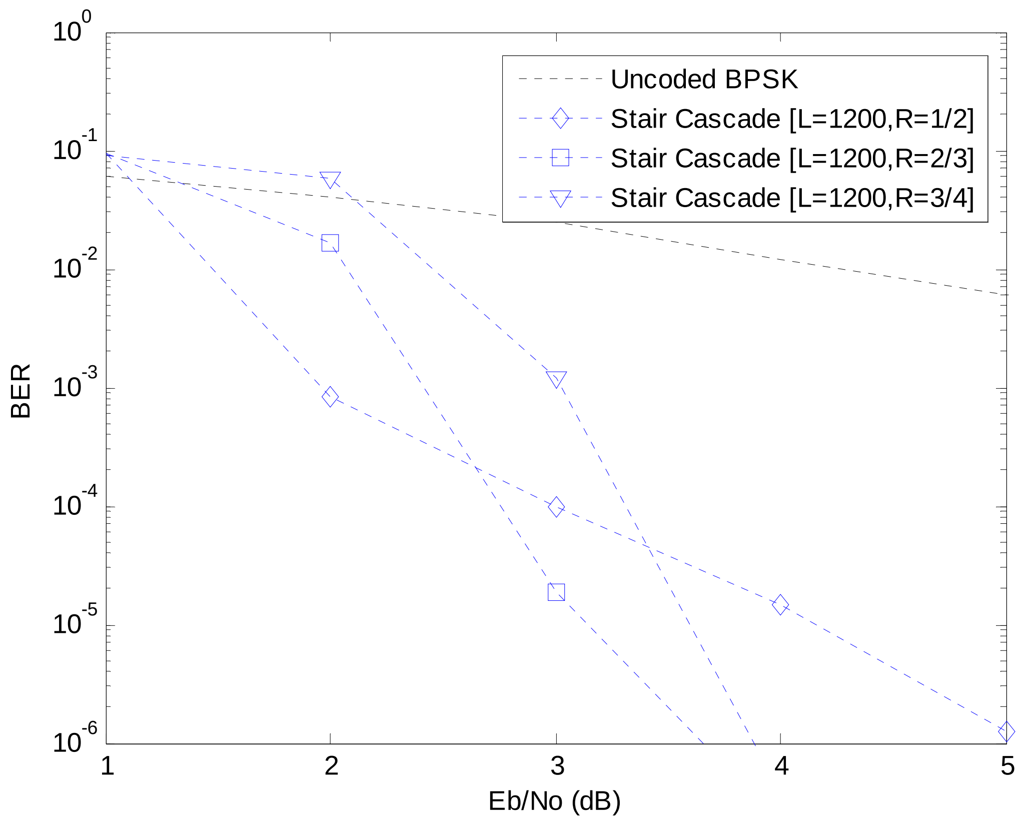


Figure 4.20 Performance of Stair Cascade codes with code length 1,200 bits at different code rates

Column weight

Concerning the poor error floor performance of the Stair Cascade codes at lower code rates, it is interesting to investigate the impact of the number of the column weight on the code performance. The Stair Cascade codes with code length 1,200 bits are constructed with different numbers of column weights, i.e. 3, 6 and 9. The codes are the simulated at different code rates, i.e. 1/2, 2/3, and 3/4.

First, the Stair Cascade codes are simulated at code rate 1/2 with the following slope parameter

Column weight	Slope parameter
3	[547+548+572][0]
6	[545+216+589+518+382+94][0]
9	[31+235+515+467+85+214+117+228+134][0]
12	[423+380+126+57+207+544+225+128+171+289+341+ 33][0]

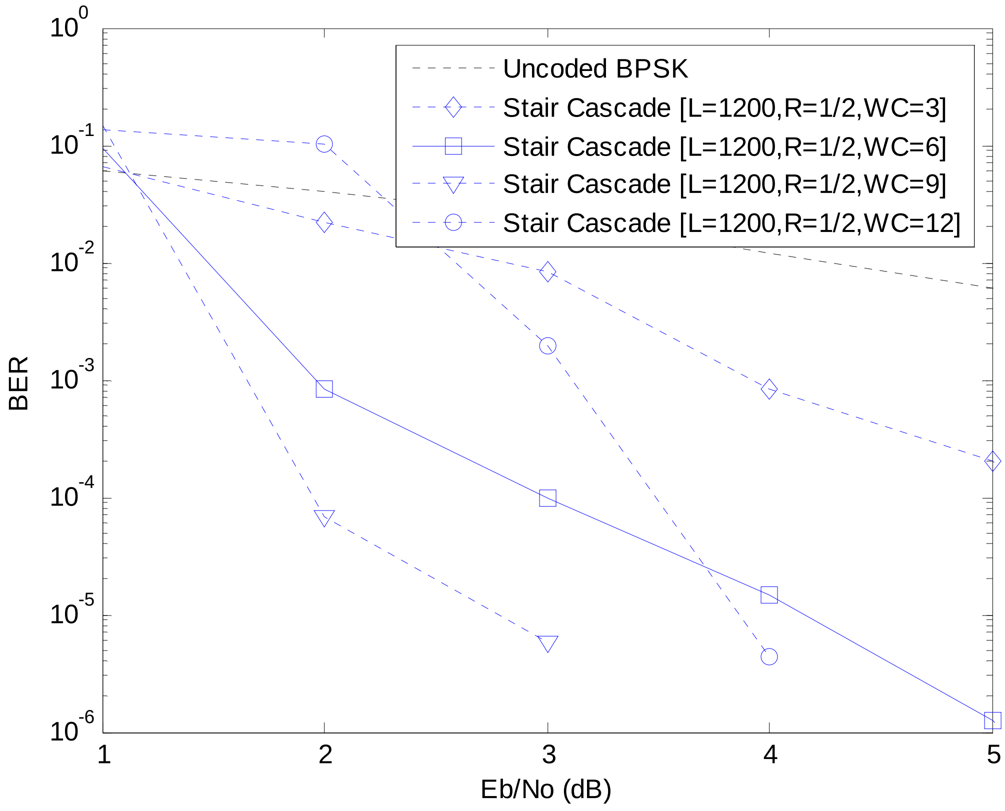


Figure 4.21 Impact of column weight on the performance of Stair Cascade codes with code length 1,200 bits at code rate 1/2

As shown in Figure 4.21 the increase of the weight of the column (WC) can improve the performance of the Stair Cascade codes to some extent. At code rate 1/2 the Stair

Cascade codes improve their performance as the column weight increases up to 9. However, more column weight cannot improve the code performance. Even, this leads to the degradation of the code performance. The excessive increased column weight contributes to higher cycles in the bipartite graph that reduce the node independence leading to degraded decoding performance.

Furthermore, the impact of the increased column weight is investigated in case of the Stair Cascade codes with code length 1,200 bits at code rate 2/3. The simulated codes are constructed with the following slope parameter

Column weight	Slope parameter
3	[195+323+100][128+345+54][0]
6	[109+311+190+87+325+38][83+377+199+157+238+282][0]
9	[346+102+173+362+190+124+84+131+234]
	[105+2 40+192+186+186+140+194+164][0]

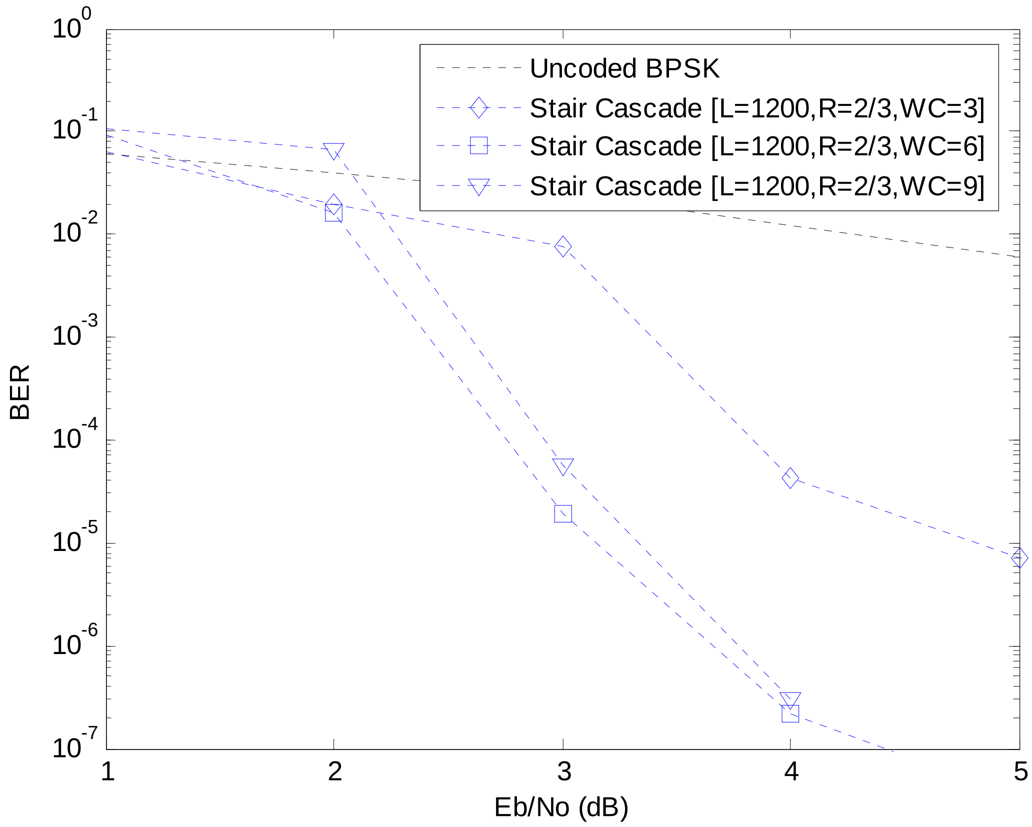


Figure 4.22 Impact of column weight on the performance of Stair Cascade codes with code length 1,200 bits at code rate 2/3

Figure 4.22 shows the impact of the increased column weight in the Stair Cascade codes at code rate 2/3. Again, the increased column weight improves the code performance to some extent. The codes perform best at column weight 6. More column weight leads to negligible code performance.

Finally, the impact of an increased column weight is investigated in the Stair Cascade code with code length 1,200 bits at a higher code rate, i.e. 3/4. The codes are constructed with the following slope parameter

Column weight	Slope parameter
3	[202+50+88][241+191+206][83+127+210][0]
6	[146+203+285+193+298+1][104+121+55+281+184+124] [169+154+215+247+208+165][0]
9	[155+51+54+219+11+133+64+227+149][273+280+275+220+297+... 89+21+151+282][230+235+186+138+124+240+152+86+229][0]

As shown in Figure 4.23 the Stair Cascade codes achieve their best performance at column weight 6. The increase of the column weight is not enough to improve the performance. Even this leads to worse performance

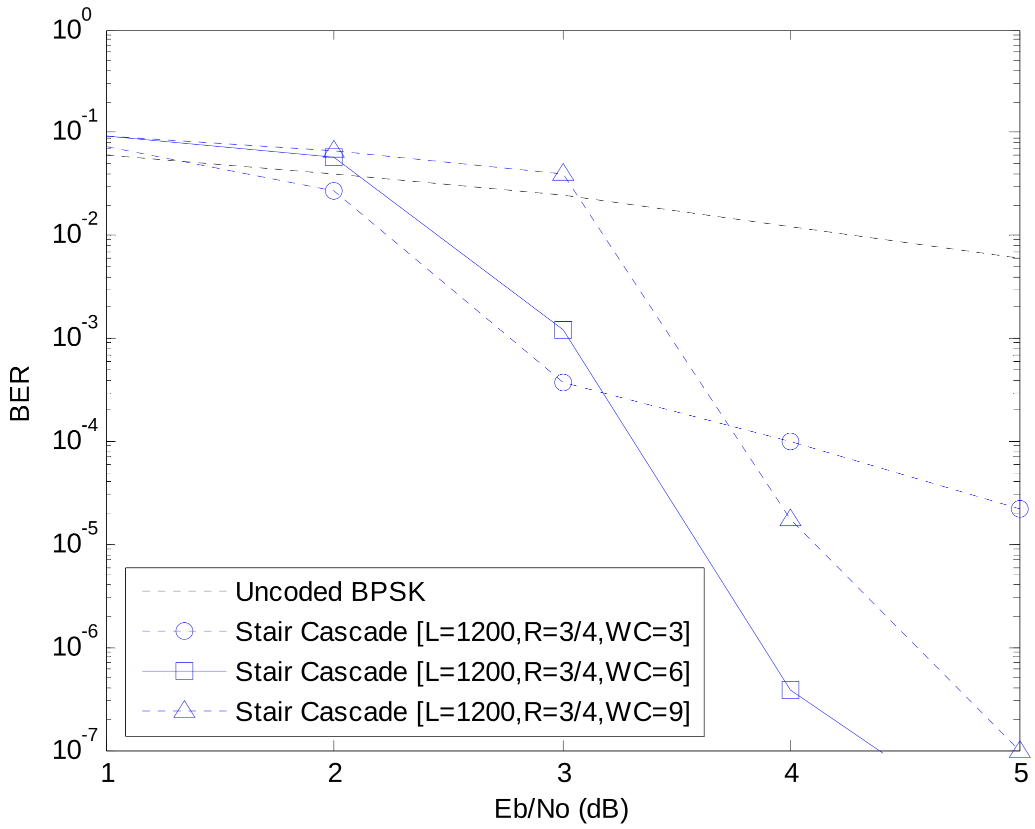


Figure 4.23 Impact of column weight on the performance of Stair Cascade codes with code length 1,200 bits at code rate 3/4

From the simulation result above, in general, the code with low column weight suffers from a higher error floor. By increasing the column weight the error floor can be effectively suppressed to some extent and therefore, the floor performance of the codes improves. However, the high column weight leads to a degraded performance. At code rate 1/2 the codes performs best with column weight 9. Meanwhile, at higher code rates, a column weight 6 would be more appropriate

To some extent, higher column weights can help the code in improving its performance because more check nodes are involved, that lead to more reliable decoding. However, the number of column weight is restricted by the rule of thumb in decoding, i.e. node independence. Higher column weights could mean lower node independence, which in turn can reduce the decoding performance. Conclusively, the choice of column weights while designing codes must concern with code rate and code length as well.

b. Lattice Structure

Maximal number of decoding iteration

The impact of the number of decoding iterations on the performance of the Stair Lattice codes is investigated. The number of maximal decoding iterations is practically set up to 20. The Stair Lattice code is set to code length 1,200 bits, code rate 3/4 and column weight 6. Each submatrix is composed by four smaller submatrices with column weight 3. Its slope parameters are as follows

[[24+92+124][31+8+42];[120+21+118][99+145+91]] …
[[25+55+85][15+50+51];[27+30+94][39+26+146]] …
[[111+142+131][106+2+88];[[138+87+29][97+68+95]][0]

As shown in Figure 4.24 the Stair Lattice codes achieve their optimal BER performance already at the maximal decoding iteration 10. More decoding iterations can be considered as negligible for the performance improvement. Therefore, again, a maximal number for the decoding iterations of 10 is selected as basis for further investigation of the Stair Lattice codes.

Code rate

The performance of the Stair Lattice codes is simulated at three different code rates, i.e. 1/2, 2/3 and 3/4. The codes are constructed at code length 1,200 bits and column weight 6. Their slope parameters are as follows

Code rate	Slope parameter
1/2	[[234+226+137][109+201+205];[291+293+60][267+180+151]][0]
2/3	[[126+57+128][171+33+89];[196+113+133][80+22+28]] ...
	[[65+118+25][85+61+164];[125+24+183][109+162+21]][0]

3/4	[[7+25+89][141+133+91];[117+101+99][76+24+43]] ... [[83+128+51][88+144+116];[18+104+16][132+52+50]] ... [[106+108+111][55+31+96];[138+118+75][70+97+39]][0]

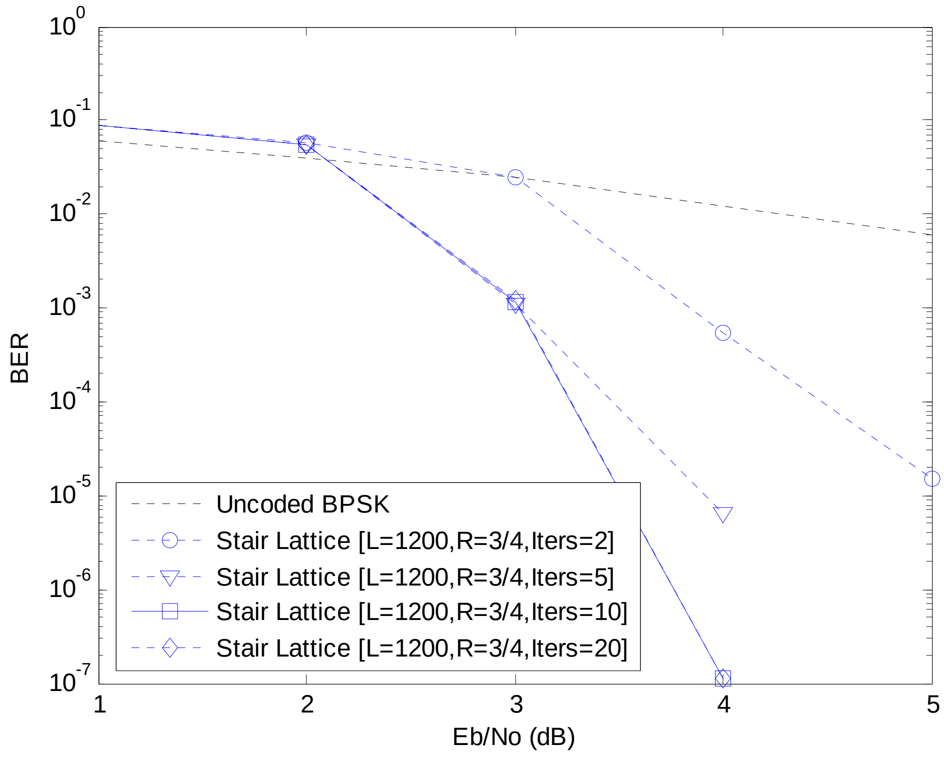


Figure 4.24 Impact of decoding iteration on the performance of the length-1200, rate-3/4 Stair Lattice code

Figure 4.25 shows that the threshold performance of the Stair Lattice codes is subject to get worse as the code rate increases. However, at the other side, the error floor performance improves. The Stair Lattice code with code rate 2/3 can achieve a bit error rate of 10^{-5} at about 3 dB which is 0.5 dB better than the code with code rate 3/4. However, at higher signal-to-noise ratio, the Stair Lattice code with code rate 2/3 suffers from a higher error floor. Meanwhile, the performance of Stair Lattice codes with code rate 1/2 degrades significantly at a higher signal-to-noise ratio due to poor error floor performance. A high number of variables being part of the identity matrix could be responsible for the high error floor at code rate 1/2.

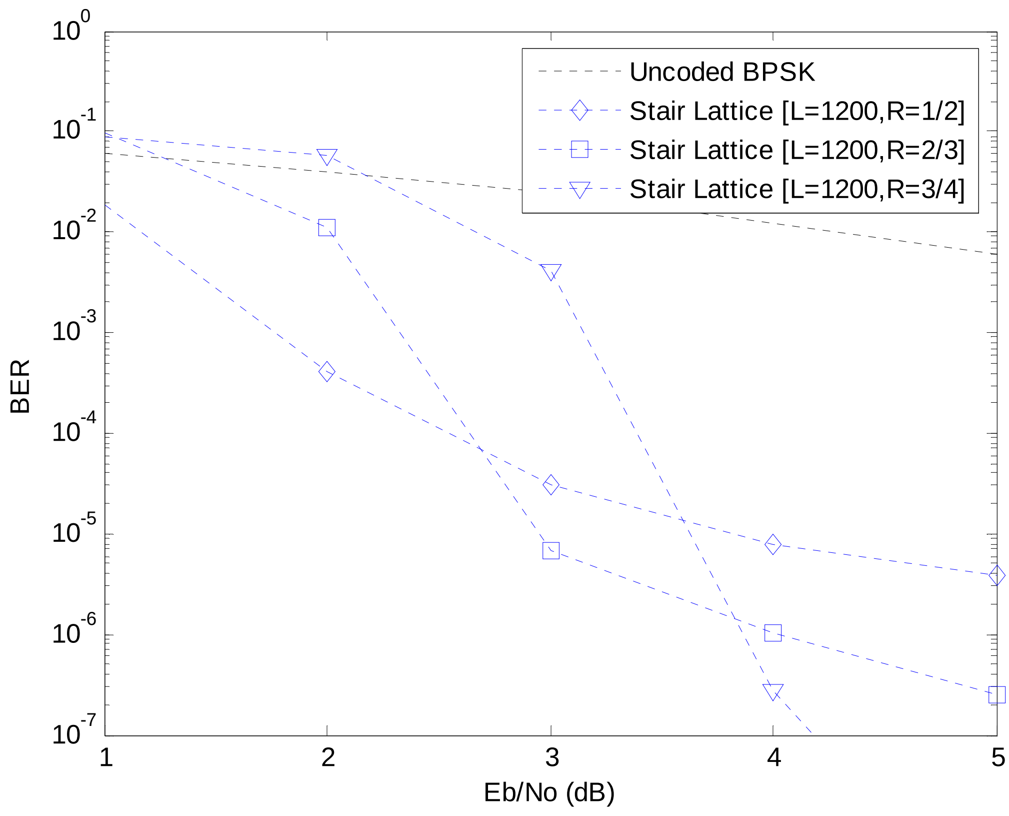


Figure 4.25 Performance of Stair Lattice codes with code length 1,200 bits at different code rates

Column weight

Concerning the poor error floor performance of the Stair Lattice codes at lower code rates, it is interesting to investigate the impact of the number of column weight on the code performance. The Stair Lattice codes with code length 1,200 bits are constructed with different numbers of column weights, i.e. 4, 6 and 8. The codes are than simulated at different code rates, i.e. 1/2, 2/3, and 3/4.

First, the Stair Lattice codes are simulated at code rate 1/2 with the following slope parameters

Column weight	Slope parameter
4	[[70+27][271+191];[266+35][117+73]][0]
6	[[234+226+137][109+201+205];[291+293+60][267+180+151]][0]
8	[[99+170+187+121][81+128+231+102]; ... [37+189+280+183][137+191+161+242]][0]

As shown in Figure 4.26 the increase of the weights of columns (WC) can improve the performance of the Stair Lattice codes to some extent. At code rate 1/2 the Stair Lattice codes show their best performance with a column weigh of 6. More column weight cannot improve significantly the code performance.

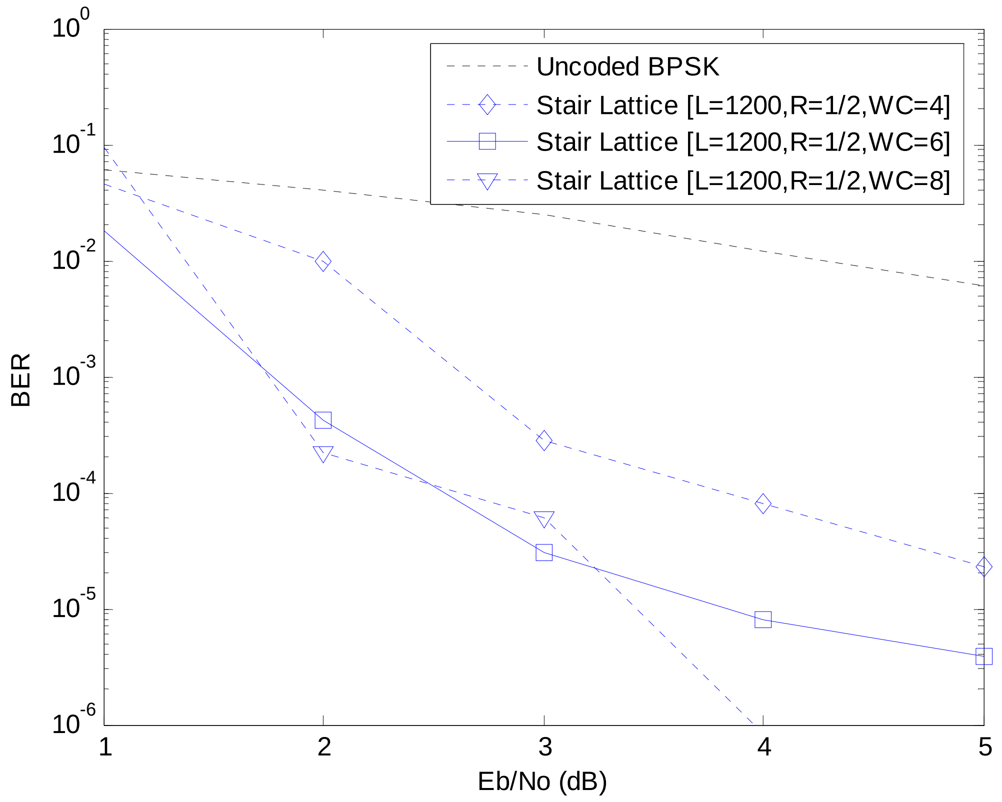


Figure 4.26 Impact of column weight on the performance of Stair Lattice codes with code length 1,200 bits at code rate 1/2

Furthermore, the impact of the increased column weight is investigated in case of the Stair Lattice codes with code length 1,200 bits at a code rate of 2/3. The simulated codes are constructed with the following slope parameters

Column weight	Slope parameter
4	[[19+123][28+89];[103+31][194+106]] ... [[159+71][42+48];[190+62][110+84]][0]
6	[[126+57+128][171+33+89];[196+113+133][80+22+28]] ... [[65+118+25][85+61+164];[125+24+183][109+162+21]][0]
8	[[43+122+158+59][198+126+108+75];[71+29+17+140] ... [155+162+193+72]][[85+66+135+134][12+116+20+150]; ... [173+52+153+113][139+5+30+22]][0]

Figure 4.27 shows the impact of the increased column weight in the Stair Lattice codes at code rate 2/3. Again, the increased column weight improves the code performance

to some extent. The codes perform best at column weight 6. More column weight leads to negligible code performance.

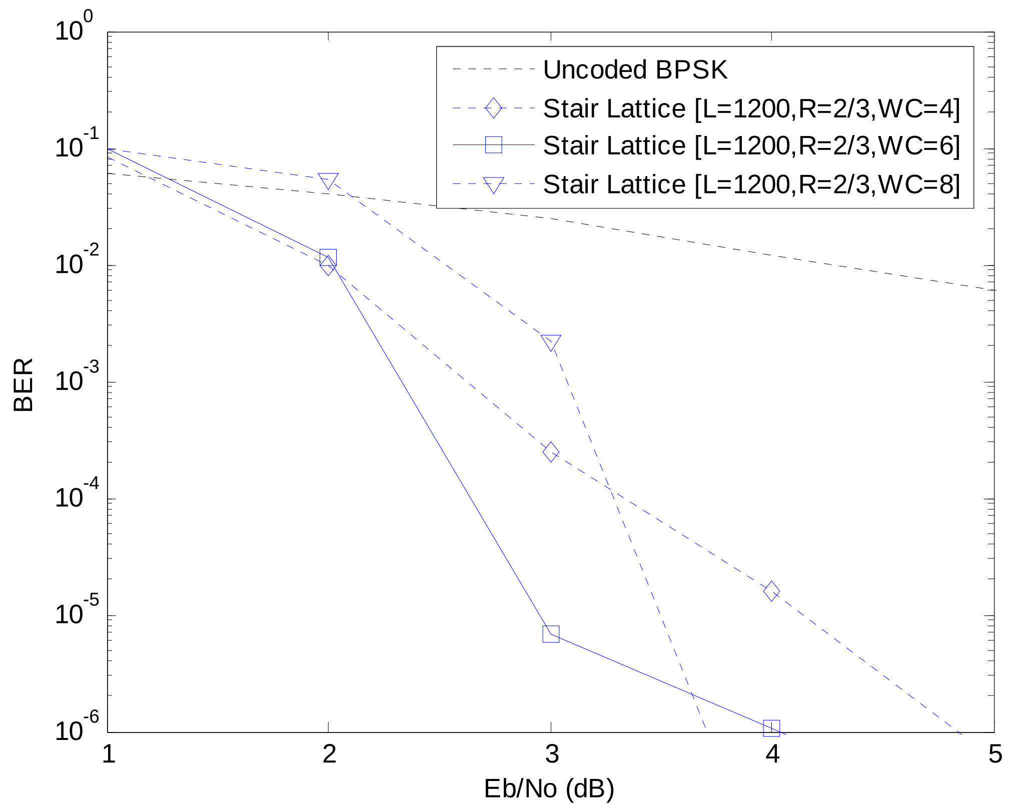


Figure 4.27 Impact of column weight on the performance of Stair Lattice codes with code length 1,200 bits at code rate 3/4

Finally, the impact of the increased column weight is investigated in the Stair Lattice code with code length 1,200 bits at higher code rates, i.e. 3/4. The codes are constructed with the following slope parameters

Column weight	Slope parameter
4	[[69+120][51+81];[25+149][56+98]] ...
	[[79+125][64+37];[124+133][131+105]] ...
	[[28+26][20+113];[135+34][40+147]][0]
6	[[7+25+89][141+133+91];[117+101+99][76+24+43]] ...
	[[83+128+51][88+144+116];[18+104+16][132+52+50]] ...
	[[106+108+111][55+31+96];[138+118+75][70+97+39]][0]
8	[[30+126+107+82][38+9+142+108];[75+81+76+94][103+19+45+1]] ...
	[[57+49+48+25][64+139+23+91];[22+6+96+87][92+58+89+125]] ...
	[[133+68+13+138][130+129+101+62];[135+33+98+41][12+31+131+140]][0]

As shown in Figure 4.28 the Stair Lattice codes achieve their best performance at column weight 6. The increased column weight contributes to a higher number of short cycles in the bipartite graph that reduce the node independency, leading to degraded decoding performance. The weight is not enough to improve the performance. Even this leads to worse performance.

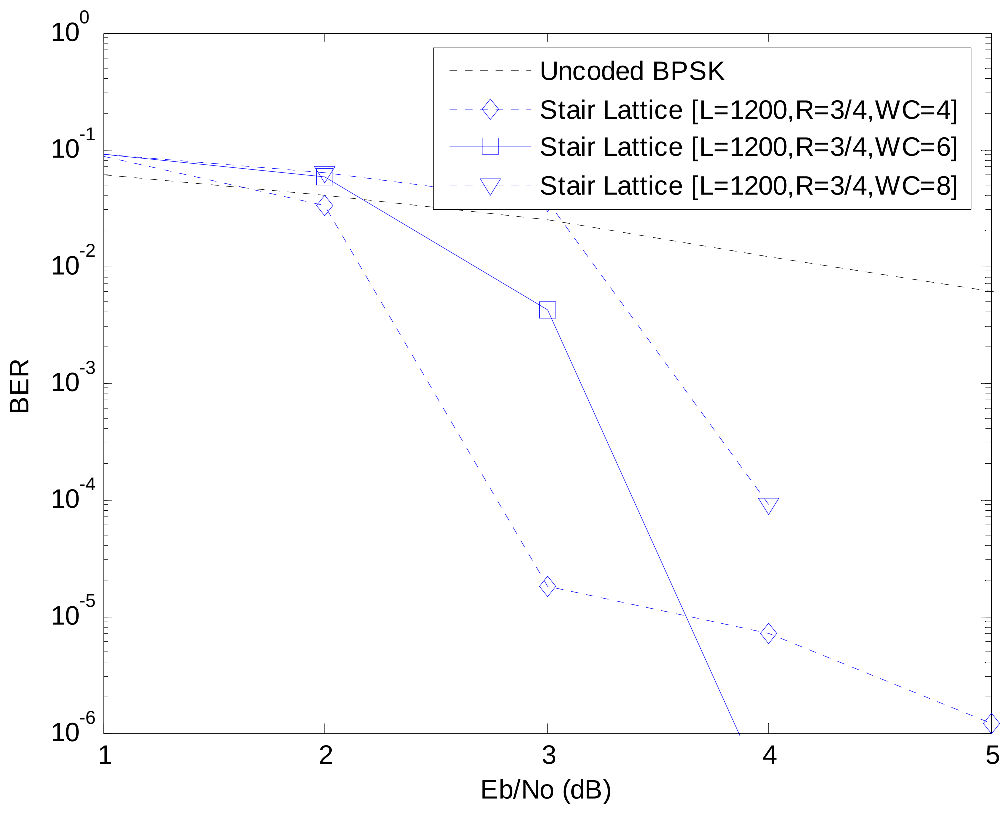


Figure 4.28 Impact of column weight on the performance of Stair Lattice codes with code length 1,200 bits at code rate 3/4

From the simulation result above, in general, the codes with low column weight suffer from higher error floors. By increasing the column weight the error floor can be effectively suppressed to some extent and therefore, the floor performance of the codes improves. However, high column weight leads to degraded performance. At all code rates, column weight 6 would be more appropriate

To some extent, a higher column weight can help the code in improving its performance because more check nodes are involved, that lead to a more reliable decoding. However, the number of column weights is restricted by the rule of thumb in decoding, i.e. node independence. Higher column weights could mean lower node independence, which in turn can reduce the decoding performance. Conclusively, the choice of column weights in designing codes must concern with code rate and code length as well.

c. Comparison of Both Stair Codes

The structure of both Stair codes is put in comparison. In the same length and code rate, the cascade structure has less complexity than the lattice structure. The lattice structure is more flexible in code rate than the cascade structure, however, at cost of more cyclic-shift register. For example, at code rate 3/4, the number of cyclic-shift register for the lattice structure is three times than that of the cascade.

In addition to that, the code rate of the cascade is confined to $R = j/k$, where $k = j+1$ and j, k are integers. Parameter j denotes the number of submatrices of the parity check matrix of the Stair cascade codes. Meanwhile, the lattice structure support more flexible code rates $R = j/k$, where $j < k$ and j, k are integers. However, more cyclic-shift registers are necessary to cope with the more flexible code rate.

The performance of both structures is put in comparison. Stair Cascade code are compared to two Stair Lattice codes, whose structure is determined by the number of smaller submatrix forming submatrix. The submatrix of the first Stair Lattice code is composed by four smaller submatrices. Meanwhile, that of the second one contains nine smaller submatrices. The Stair codes with code length 1,200 bits, code rate 3/4 and column weight 6 are simulated. The simulated structures of those Stair codes have the following slope parameters

Type	Slope parameter
Cascade	[146+203+285+193+298+1][104+121+55+281+184+124]
	[169+154+215+247+208+165][0]
Lattice-4	[[7+25+89][141+133+91];[117+101+99][76+24+43]] ...
	[[83+128+51][88+144+116];[18+104+16][132+52+50]] ...
	[[106+108+111][55+31+96];[138+118+75][70+97+39]] [[0][-];[-][0]]
Lattice-9	[[23+94][86+46][40+59];[70+92][74+48][36+24];[69+65][5+87][47+72]] ...
	[[32+91][80+42][88+43];[58+17][90+20][97+71];[62+57][49+38][61+81]] ...
	[[25+12][3+89][50+73];[15+29][93+30][26+27];[35+85][14+33][9+13]] ...
	[[0][-][-];[-][0][-];[-][-][0]]

As shown in Figure 4.29 the performance of both Stair codes is comparable. The similar results also appear by comparing the performance of both Stair codes in the previous section. In term of realization, the Stair Cascade code becomes the choice due to its more practical hardware implementation. However, this choice is valid only for certain code rates the Stair Cascade code can cope with, i.e. code rate $t/(t+1)$, where t is integer.

At specific code rates other than $t/(t+1)$, where t is integer, the Stair Cascade code can be applied. For this case, only the Stair Lattice code is applicable. To prove the performance of the Stair Lattice code at those code rates, three Stair Lattice codes with code length 1,200 bits are constructed at code rate 3/8, 4/8 and 5/8. The structure of their parity check matrices can be figured out in Figure 4.30. The permutation submatrices of the parity check matrices in the systematic parts contain one column weight each. Therefore, the column weight of the parity check matrices is proportional to the number of vertically arranged permutation submatrices. For instance, in case of the 3/8-rate Stair Lattice code, the parity check matrix with three vertically arranged permutation submatrices contains a column weight of three. Using one-column-weight permutation submatrices the 4/8-rate Stair Lattice code contains a column weight 4 and the 5/8-rate Stair Lattice code contains a column weight 5.

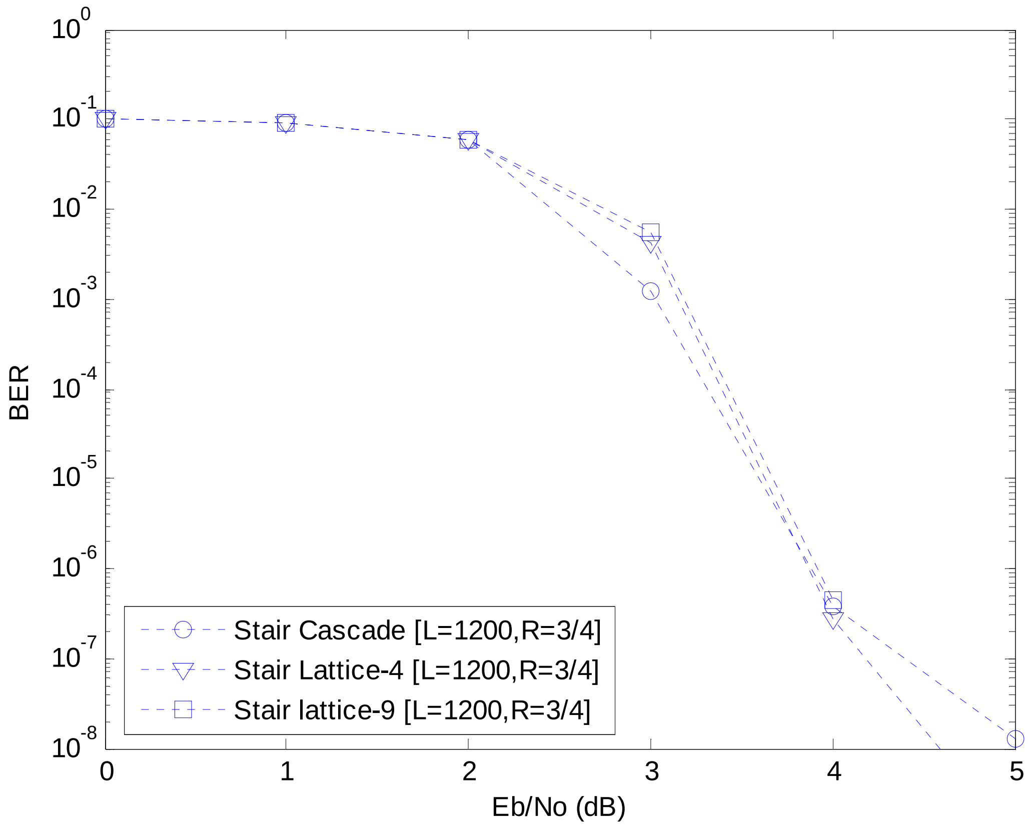
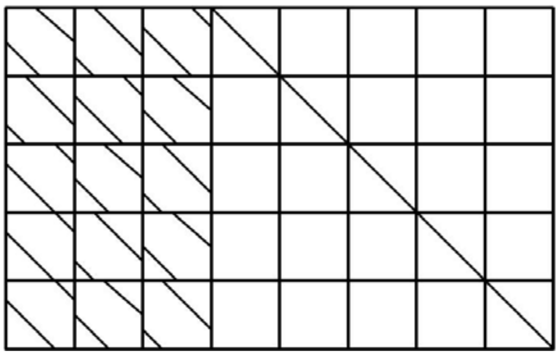
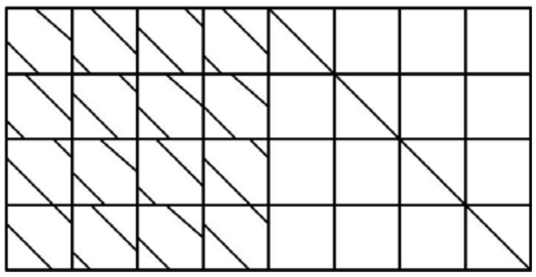


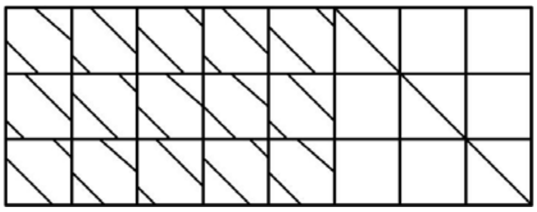
Figure 4.29 Comparison of the performance of Stair Cascade and Stair Lattice codes with code length 1,200 bits and code rate 3/4



(a) Code rate 3/8



(b) Code rate 4/8



(c) Code rate 5/8

Figure 4.30 Structure of parity check matrices of Stair Lattice codes with different code rates

Figure 4.31 shows the performance those Stair Lattice codes. The Stair Lattice code with code rate 4/8 performs best among them. However, it suffers from error floor due to its greater identity matrix, which constitutes more one-degree variable nodes. The simulated codes have slope parameters as follows.

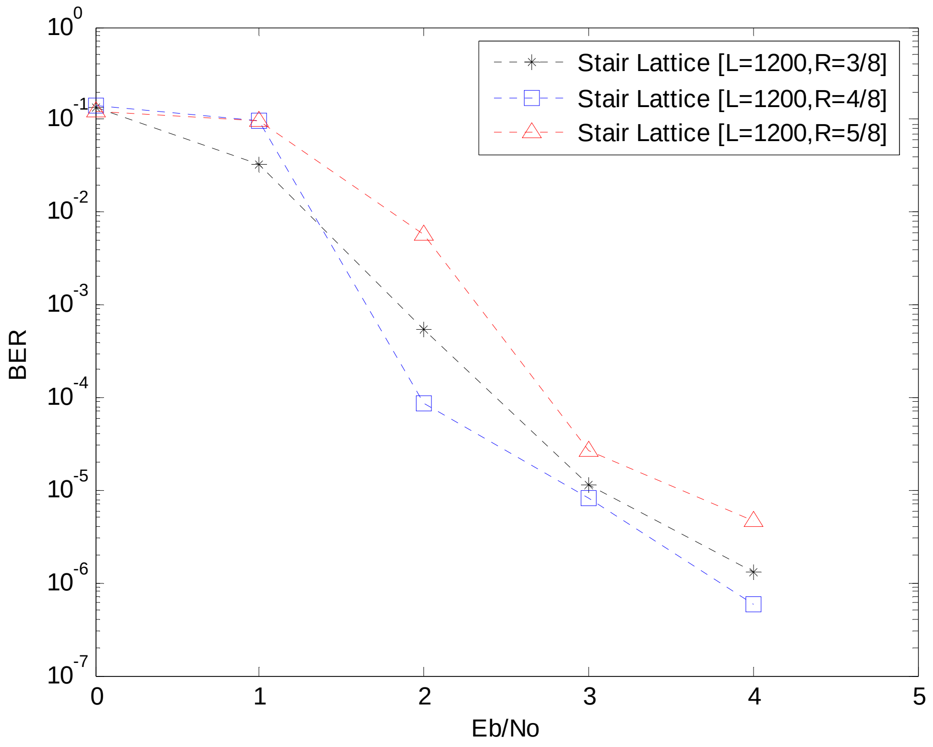


Figure 4.31 Performance of the length-1,200 Stair Lattice codes at specific code rates

Code rate	Slope parameter
3/8	[[128][124][88];[100][67][129];[99][40][6];[144][35][63];[71][62][58]] ... [[0][-][-][-][-];[-][0][-][-][-];[-][-][0][-][-];[-][-][-][0][-];[-][-][-][-][0]
4/8	[[134][109][22][93]; [39][59][27][62];[44][80][144][119]; [142][56][84][72]] ... [[0][-][-][-];[-][0][-][-];[-][-][0][-];[-][-][-][0]]
5/8	[[79][74][46][101][144];[121][89][41][20][81];[116][128][64][123][147]] ... [[0][-][-];[-][0][-];[-][-][0]]

d. Code Performance of Special Constructions

High-rate and Short-length Performances

Furthermore, the performance of the Stair codes is investigated at higher code rates. Typically, higher code rate include the case (n-1)/n, where n is integer. For this reason the Stair Cascade codes are selected. The Stair Cascade codes with code length 1,200 bits and column weight 6 are simulated at code rate 3/4, 7/8 and 19/20. The slope parameters of their parity-check matrices are presented below. As shown in Figure 4.32, in general, the performance of the codes becomes poorer as the code rate increases. However, the Stair Cascade codes still demonstrates good performance at the very high code rate 19/20.

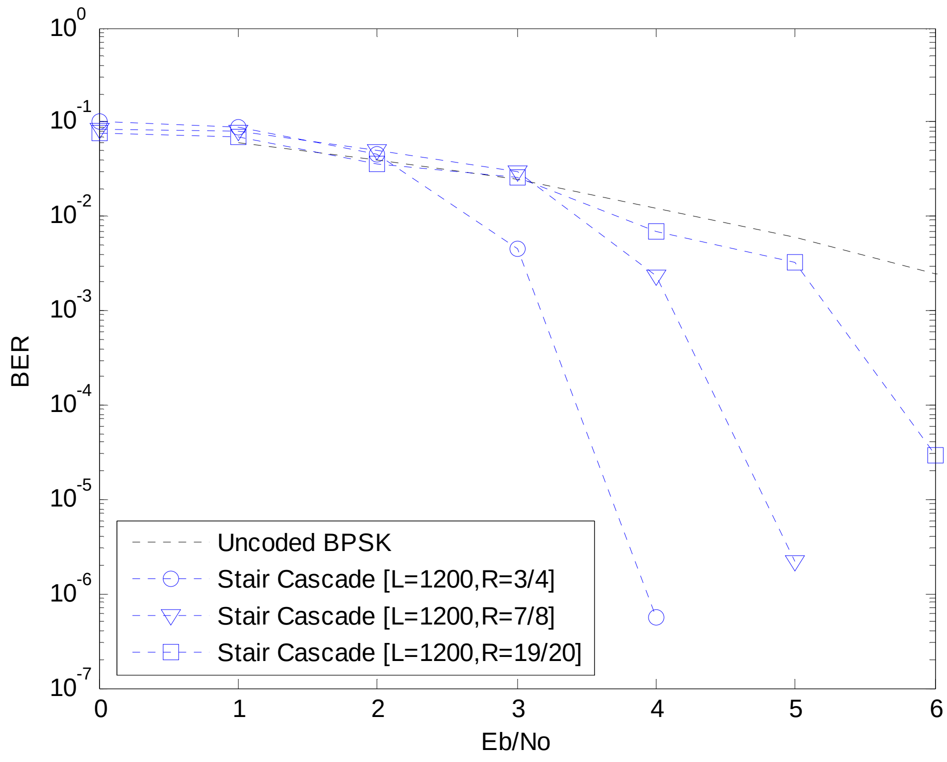


Figure 4.32 Performance of Stair Cascade codes at high code rate

Code rate	Slope parameter
3/4	[146+203+285+193+298+1][104+121+55+281+184+124] ...
	[169+154+215+247+208+165][0]
7/8	[77+49+141+145+120+91][112+129+126+26+110+103] ...
	[69+4+140+125+46+116][114+139+97+12+7+62] ...
	[88+29+21+111+101+132][81+86+127+34+31+19] ...
	[95+53+130+52+41+94][0]
19/20	[12+2+0+51+54+45][21+8+25+1+28+53][18+37+11+30+33+7] ...
	[34+13+38+50+44+42][26+6+40+55+23+39][19+9+31+34+4+35] ...
	[29+52+57+3+32+46][49+41+20+10+14+43][15+48+56+36+17+22] ...
	[16+27+59+47+5+58][0+5+30+46+24+14][47+36+56+42+18+3] ...
	[48+43+26+53+28+31][1+49+22+11+33+50][54+32+20+45+2+25] ...
	[34+6+59+52+55+4][17+35+44+12+41+29][7+21+10+9+38+39] ...
	[13+40+57+27+19+37][0]

The performance of the Stair codes at different code length, especially at very short code length is also investigated. The Stair Cascade codes at arbitrary code length i.e. 128, 576, 2304 and 6000 bits are simulated at code rate 3/4. The slope parameter of their parity-check matrix is presented below. In general, the Stair Cascade codes perform better as the code length increases as shown in Figure 4.33. Especially, at very short code length, the code still has good performance.

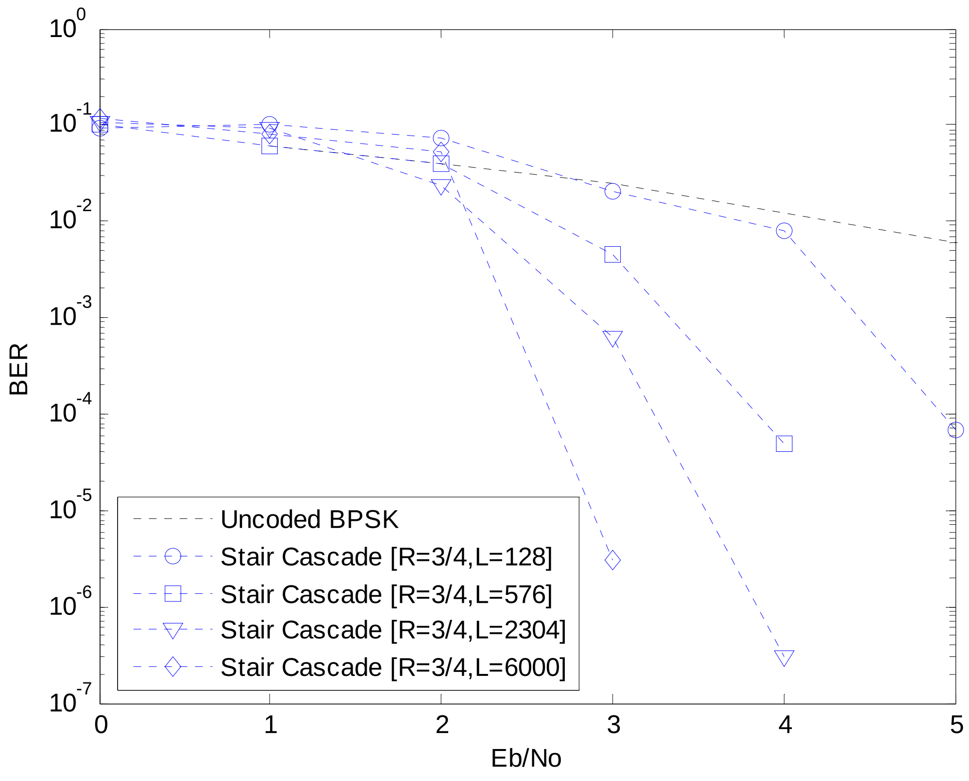


Figure 4.33 Performance of the rate-3/4 Stair codes at different code length

Code length	Slope parameter
128	[19+10+15+7+17+20][2+23+5+4+18+14][25+6+31+8+26+27][0]
576	[24+105+123+26+69+94][11+31+122+141+16+116][62+23+114+81+7+60][0]
2304	[177+134+378+298+426+43][373+95+530+87+435+496] [45+510+71+142+438+55][0]
6000	[524+581+241+663+1354+332][571+864+676+928+1216+1150] [1121+41+277+282+1122+988][0]

Irregular Codes

So far the Stair codes with "quasi regular" column weight are considered. It is also interesting to investigate the performance of the Stair codes with really irregular column weights. The irregular Stair codes contain different numbers of column weights in their submatrices.

The performance of the "regular" and irregular Stair Cascade codes with code length 1,200 bits are simulated at code rate 2/3 and 3/4. Both types of codes are compared at the same number of column weight. At code rate 2/3, the regular code is constructed with column weight 6, while the irregular code is constructed with column weight 8 and 4 in two submatrices. Those codes are constructed with the following slope parameters

[109+311+190+87+325+38][83+377+199+157+238+282][0]

and

[165+182+378+322+194+243+31+89][339+76+133+242][0]

respectively.

At code rate 3/4, the regular code is constructed with column weight 6, while the irregular code is constructed with column weights 9, 5 and 4 in three submatrices. Those codes are constructed with the following slope parameters

[277+246+53+102+84+73][190+ 208+213+146+107][4+295+200+38+199+249][0]

and

[273+144+22+270+138+228+149+31+263][200+192+283+8+7] ...
[60+110+126+111][0]

respectively.

The simulation results show that the performance of the irregular Stair Cascade codes is poorer than that of the regular ones (see Figure 4.34). The irregular codes suffer from higher error floor. Even, at lower code rates, the error floor performance is poorer in the irregular codes.

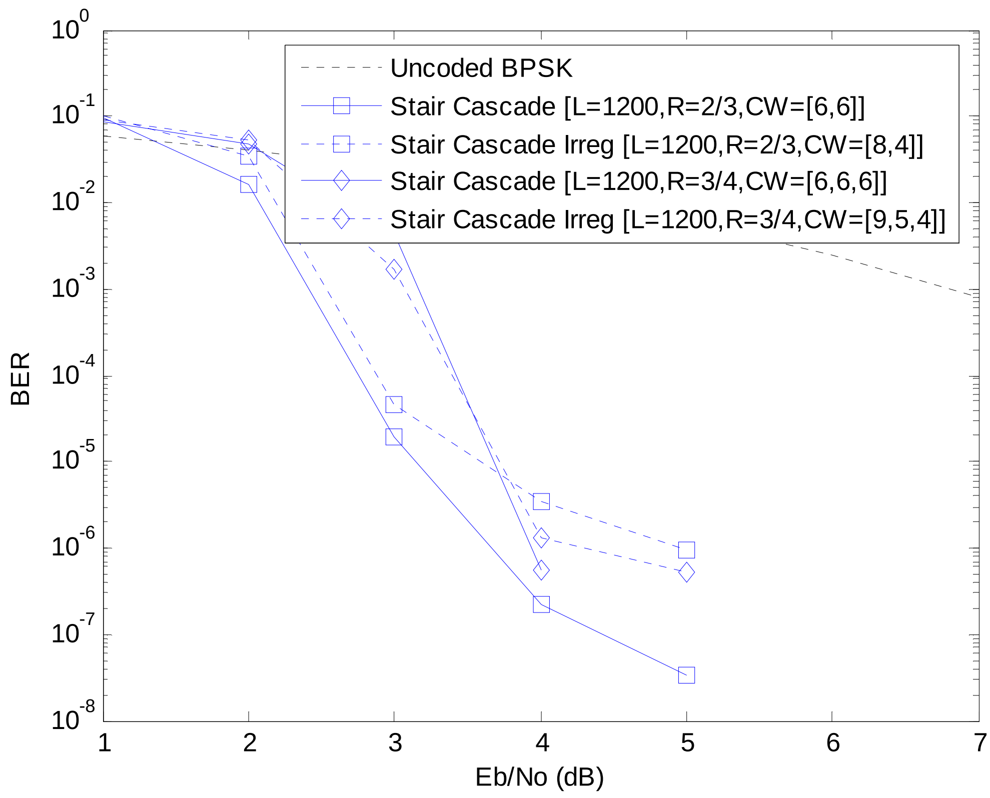


Figure 4.34 Comparison of the regular and irregular Stair Cascade codes at code length 1,200 bits and code rate 2/3 and 3/4

e. Code benchmarking

In order to measure the quality of the proposed codes, the performance of the Stair codes is put in benchmarking with several well-known LDPC benchmark codes, e.g. random MacKay code. The performance of the Stair codes at special conditions, such as high code rate or with short code is also considered and compared with some published codes. Furthermore, it is also interesting to compare the performance of the codes with that of WLAN codes. For the purpose of code benchmarking, the code rate $t/(t+1)$, where the design parameter t is integer, is of interest due to its widespread application. For their simpler implementation, again, the Stair Cascade codes are selected.

Comparison with Random codes

Random MacKay codes are well-known benchmarking codes due to their historical role in LDPC reinvention and their near-capacity BER performance. The Stair codes in Cascade and Lattice with code length 1,200 bits, code rate 3/4 and column weight 6 are simulated with the maximal number of iteration of 20 and constructed with the following slope parameters

[148+88+222+105+245+25][204+161+100+198+203+110] ...
[90+231+45+31+279+21][0]

and

[[120+146+132][143+86+119];[60+137+38][63+85+102]] ...
[[118+28+87][80+44+123];[56+116+41][122+4+88]] ...
[[14+31+17][10+90+52];[110+18+59][82+68+140]][0]

respectively.

The comparison of their BER performance and MacKay code with the same parameter and column weight 3 can be seen in Figure 4.35. Both types of Stair codes show comparable BER performance and are capable of challenging MacKay codes.

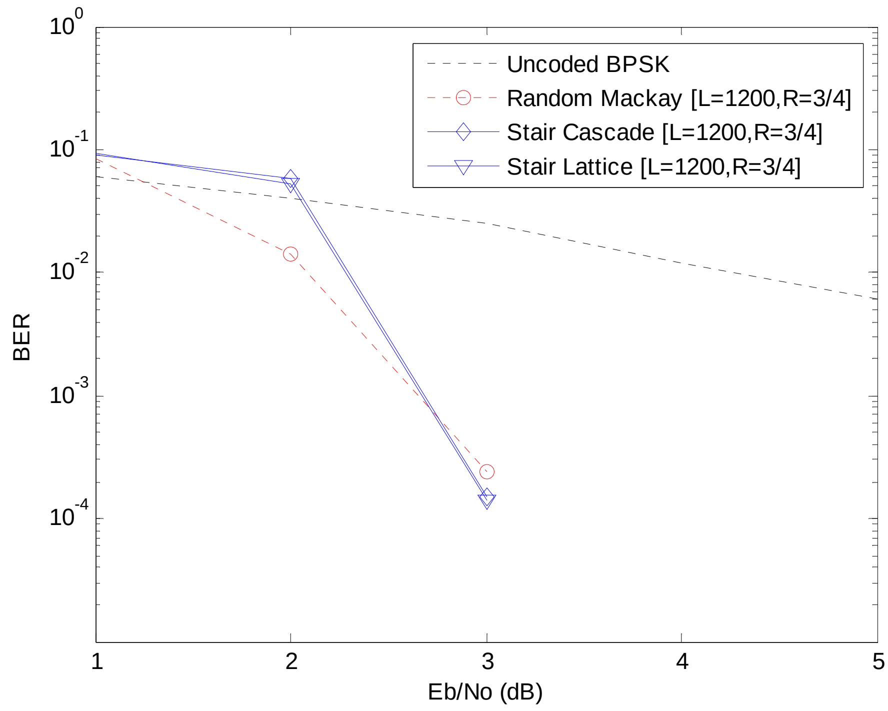


Figure 4.35 Comparison of the BER performance of Stair Codes with MacKay Codes

This achievement is of great interest because it is gained by the structured codes with low encoding complexity. However, more intensive computation is conducted by the Stair Codes in the decoding process due to more non-zero elements contained in their parity-check matrices. In this case, the Stair Codes involves 5.700 non-zero elements. It means about 58% more non-zero elements are taken into account in the decoding computations in comparison with MacKay code. But the well-structured construction of the Stair Cascade code is an advantage with respect to encoding complexity.

Furthermore, it is interesting to investigate if the Stair codes could be comparable with random MacKay code with lower column weight in order to reduce decoding complexity. The Stair Cascade codes with code length 1,200 bits are simulated at code rate 3/4 and the maximal number of iterations is 10. For benchmarking with sparse random MacKay codes, the Stair Cascade codes are constructed with different number of column weight, i.e. 3, 4 and 6. As described in the previous section, the role of the column weight for the Stair codes is important for the determination of code performance.

As shown in Figure 4.36, in spite of better threshold performance, the Stair Cascade codes with the comparable column weight, i.e. 3 and 4, cannot compete with random MacKay codes at higher SNR region due to the high error floor. Therefore, a higher number of column weights are necessary to improve the performance. Here, the performance of the Stair Cascade code with column weight 6 is comparable with that of random MacKay code with column weight 4.

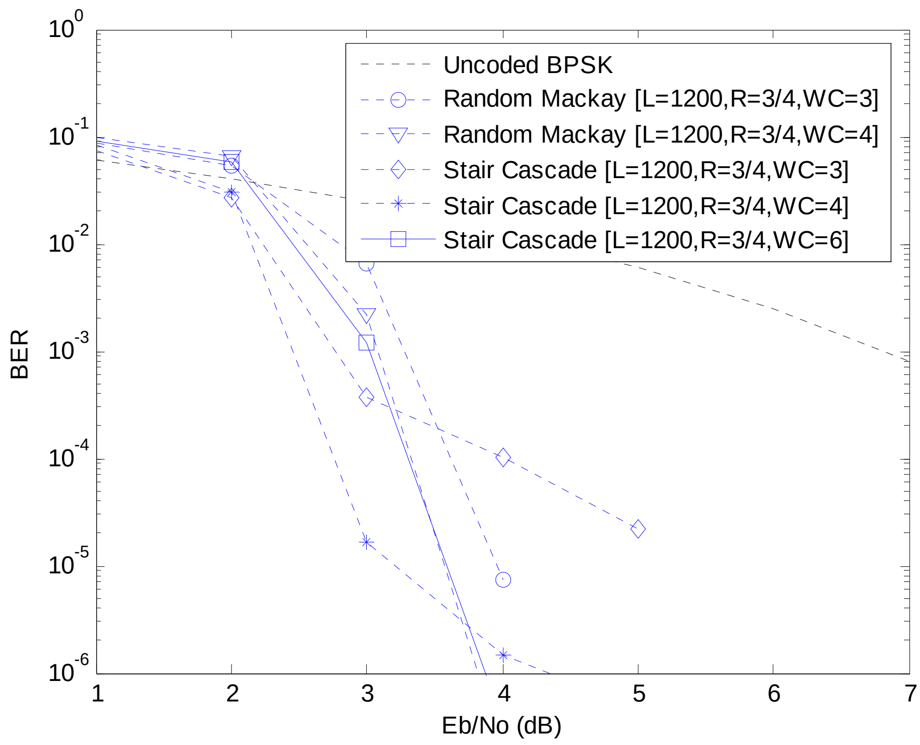


Figure 4.36 Comparison of the performance of Stair Cascade codes and random MacKay code at code length 1,200 bits, code rate 3/4.

Comparison with Short Length Code

The BER performance of the Stair codes is investigated at short code length. The codes with code length 200 bits, code rate 3/4 and column weight 6 are simulated over an AWGN channel with the maximal iteration number of 20. The code performance is then compared with that of MacKay codes. As shown in Figure 4.37 the BER performance of the Stair codes is comparable with that of MacKay codes.

The Stair Cascade code is constructed with the following slope parameters.

[36+31+18+44+4+25][3+23+26+41+19+42][35+9+20+37+7+45][0]

Meanwhile, the Stair Lattice code is constructed with the following slope parameters.

[[9+5+19][22+24+17];[0+3+6][1+11+23]] ...
[[2+20+18][8+4+7];[1+16+4][7+24+17]] ...
[[10+15+21][14+2+23];[11+19+12][18+5+0]][0]

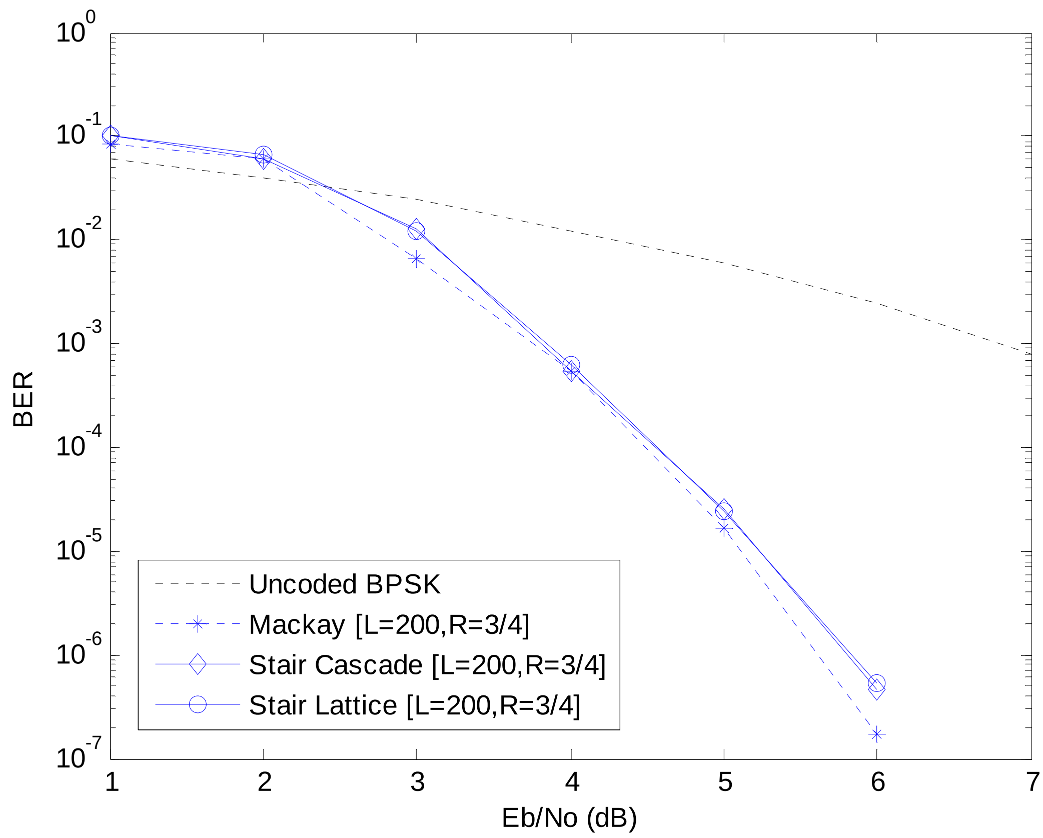


Figure 4.37 Comparison of Stair codes with MacKay code at code rate 200 and code rate 3/4

The performance of the Stair codes at the short code length 400 bits is compared with a code from difference families codes [JW02] at code rate 3/4. The difference families [404, 303] codes are constructed with four submatrices with column weight 5, 5, 3, 2, respectively. The number of column weight of the Stair Cascade codes is selected to cope with that of the difference families codes for comparable decoding complexity. Two Stair Cascade codes are constructed with column weight 3 and 5, with the following slope parameters

[74+ 21+72][18+79+7][83+9+57][0]

and

[54+4+86+ 65+ 60][99+7+97+36+72][90+61+95+27][0]

respectively.

As shown in Figure 4.38 the Stair Cascade codes outperform the difference families codes. The Stair code with lower column weight suffers from a high error floor. This deficiency can be improved by a higher column weight. With column weight 6 the decoding complexity of the Stair code is slightly higher than that of difference families. The Stair Cascade code has 1,600 non-zero-elements, meanwhile the difference families has 1,515 non-zero elements.

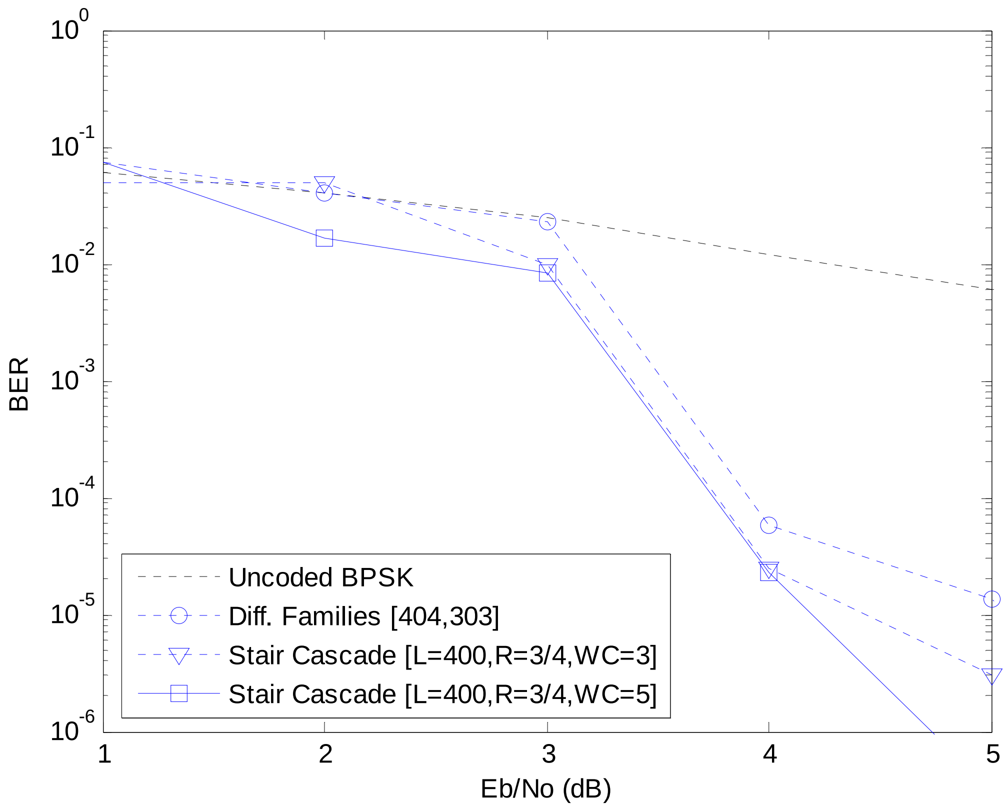


Figure 4.38 Comparison of Stair Cascade codes and difference families code

Comparison with High-Rate Code

The BER performance of the Stair Codes in high code rate is also interesting to be investigated because high code rate leads to significant performance degradation. Regarding to comparable performance but more simple implementation in code rate

$t/(t+1)$, where t is integer, the Stair Cascade is chosen as representative for testing stair codes.

Here, the Stair Cascade codes with column weight 6 are simulated on an AWGN channel at the maximal number of iteration of 10 at short code length and long code length at code rate higher than 0.85. At short code length a Stair Cascade Codes is constructed with code length 420 bits and code rate 6/7 (0.857). The Stair Cascade code is composed of 6 submatrices with column weight 6 and an identity matrix with the following slope parameters. The Stair Cascade code has the following slope parameters

[32+40+22+34+35+6][55+3+16+11+54+30][45+0+51+33+7+38] ...
[58+42+28+17+41+47][14+46+56+8+59+5][48+53+29+34+25+52][0]

It is also interesting to investigate the performance of the Stair Codes at higher code rate but with longer code length. A Stair Cascade code is constructed with code length 480 bits and code rate 7/8 (0.875). The Stair Cascade code is composed of 7 submatrices with column weight 6 and an identity matrix with the following slope parameters. The Stair Cascade code has the following slope parameters

[52+32+19+7+12+50][2+18+14+10+28+11][46+59+38+5+39+47] ...
[42+6+34+30+24+44][20+37+55+22+21+23][31+41+16+34+3+8] ...
[4+33+0+53+26+57][0]

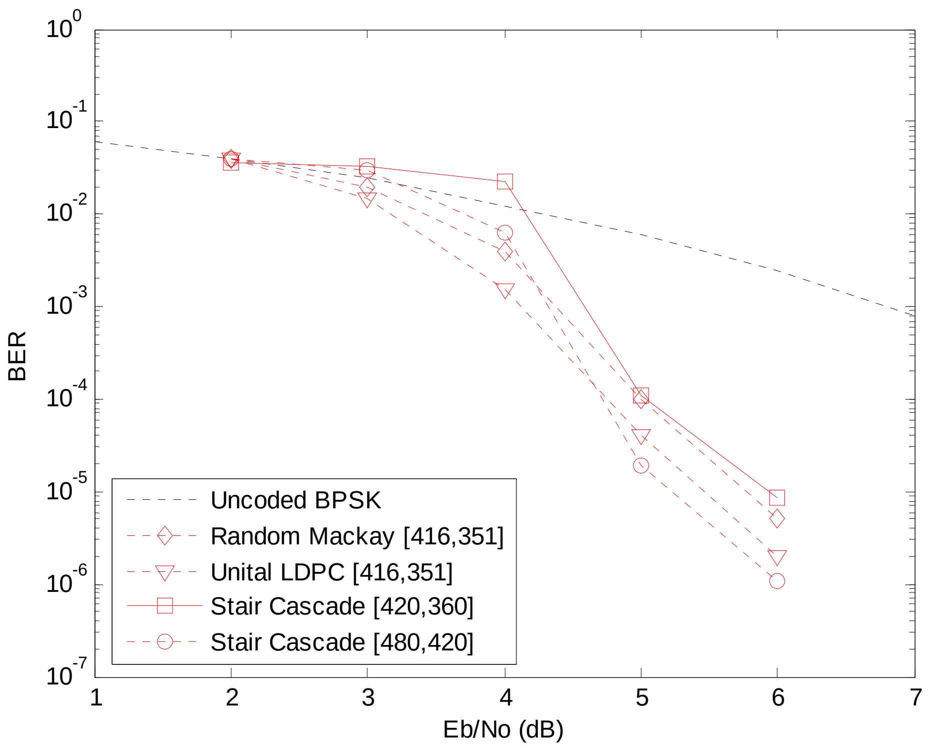


Figure 4.39 BER performance of high-rate short-length LDPC codes on AWGN channel with a maximum iteration 10

The BER performance of the Stair Cascade code is then compared to random MacKay codes with column weight 3 and a structured Unital LDPC code [JW03] with code length 416 bits and code rate 0.84375, each, as shown in Figure 4.39. The Stair Cascade code with code rate 7/8 is comparable with random MacKay codes, but is still suboptimal to the Unital LDPC code. But, it is to note that with longer code length but higher code rate, the Stair Cascade code outperforms both benchmarking codes. However, the decoding complexity of the Stair Cascade code is higher than that of both benchmarking codes due to its twice higher number of non-zero elements involved in the decoding computation. But, the Stair Cascade code offers the advantages in low encoding complexity and more simple code construction.

At long code length the Stair Cascade code is constructed at code rate 0.93 with code length 7,200 bits. The Stair Cascade code is composed of 14 submatrices with column weight 6 and an identity matrix with the following slope parameters

[240+24+252+220+92+155][437+409+124+31+162+163][8+42+153+360+159+208] ...
[152+0+451+242+305+472][193+120+330+166+21+300][118+346+99+34+324+470]
[194+463+314+171+429+347][364+158+406+336+291+476] ...
[207+181+474+457+145+91][221+25+294+372+55+175] ...
[229+213+356+228+367+422][199+448+85+198+389+15] ...
[285+381+50+164+196+230][51+261+27+248+30+94][0]

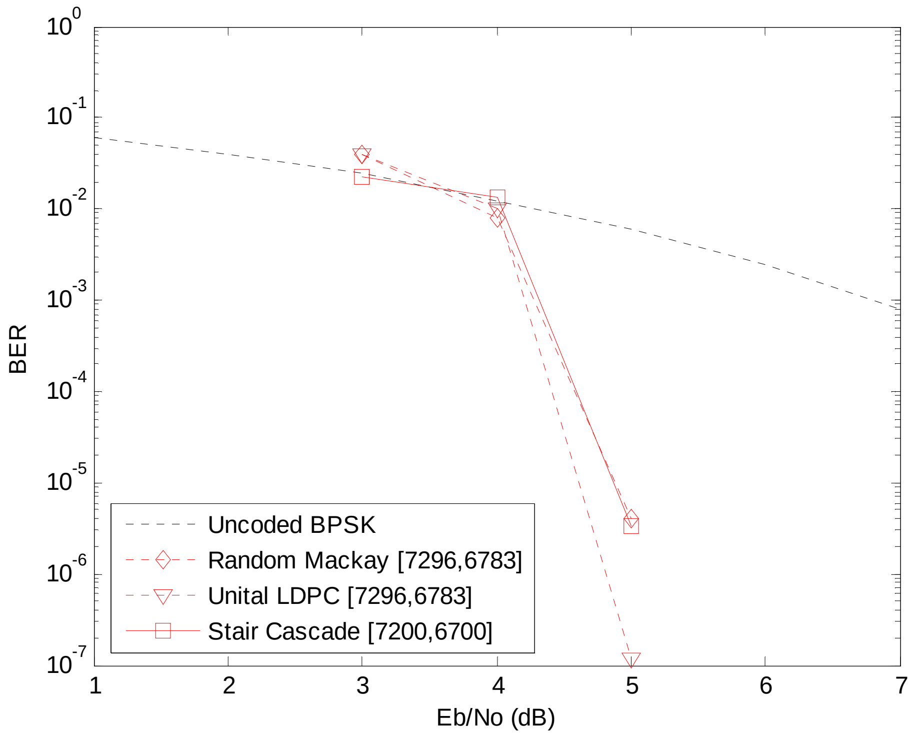


Figure 4.40 BER performance of high-rate long-length LDPC codes on AWGN channel with a maximum iteration 10

As shown in Figure 4.40, the Stair Cascade codes perform comparably with random MacKay code, but are suboptimal to Unital LDPC codes. Again, the Stair Cascade codes are higher in decoding complexity, but simpler in encoding complexity and construction method.

Comparison with WLAN Codes

The BER performance of the Stair codes is compared to the WLAN codes [IEEE09]. The comparison of the codes is done at different code rates at code length 1,296 bits over AWGN channel at the maximal number of iterations of 20. All Stair codes are constructed with column weight 6.

At code rate 1/2 the Stair Cascade codes are composed of a submatrix and an identity matrix with the following slope parameters

[366+371+608+521+275+122][0]

Meanwhile, the Stair Lattice codes are composed of a four-devided submatrix and an identity matrix with the following slope parameters

[[128+142+270][231+272+73];[101+249+16][320+293+277]][0]

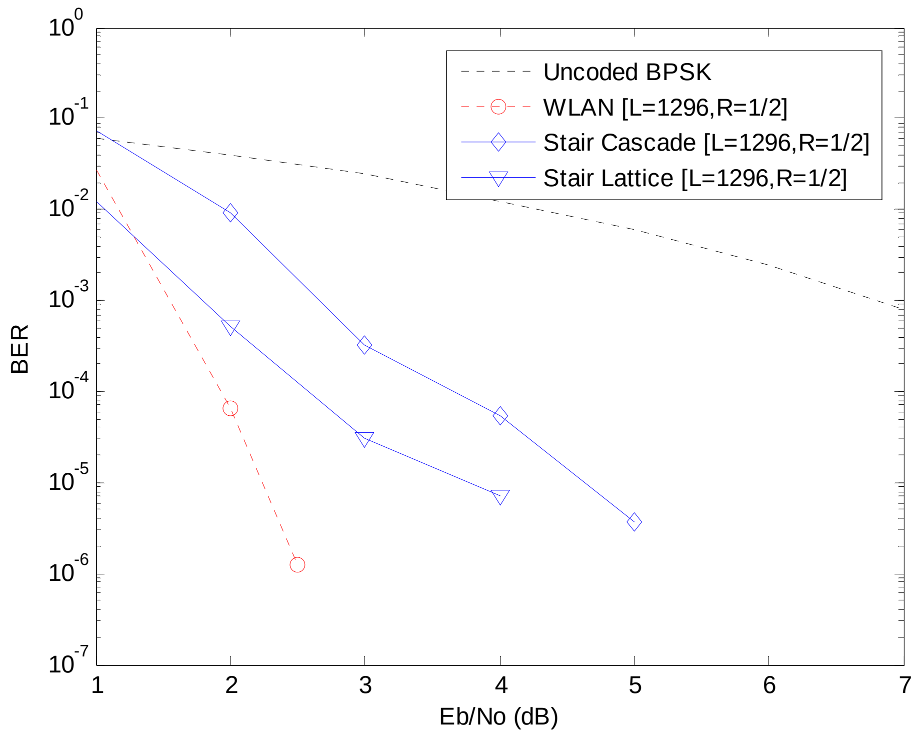


Figure 4.41 BER performance of Stair codes and WLAN code at code length 1,296 bits and code rate 1/2.

As shown in Figure 4.41 the Stair codes are suboptimal to the WLAN code. Moreover, the Stair code suffers from a high error floor.

The improvement of the BER performance of the Stair codes is done by involving more check variables or equivalently more column weigth. In case of the Stair Cascade code, the column weight is increased to 9. As shown in Figure 4.42 the BER performance of the Stair Cascade improves significantly. However, the error floor is still there. In case of the Stair Lattice, the increase of the column weight is also capable of improving the BER performance of the code, however with high error floor as shown in Figure 4.43.

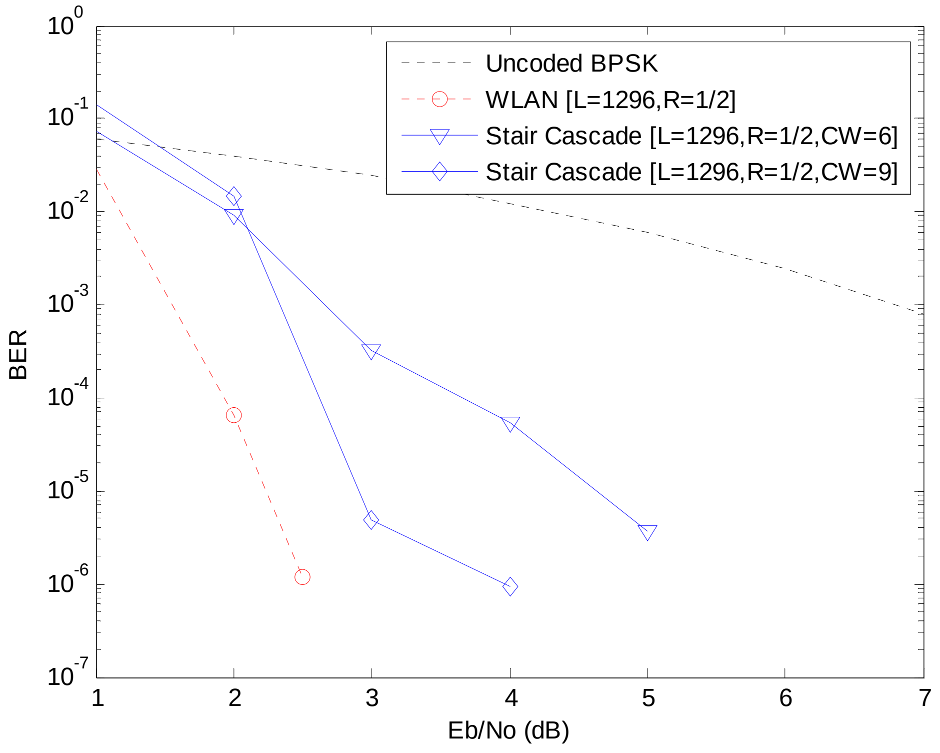


Figure 4.42 BER performance of the improved Stair Cascade codes and WLAN code at code length 1,296 bits and code rate 1/2.

At code rate 2/3 the Stair Cascade codes is composed of two submatrices and an identity matrix with the following slope parameters

[189+206+402+346+429+218][267+55+113+363+100+13][0]

Meanwhile, the Stair Lattice codes is composed of two four-divided submatrices and an identity matrix with the following slope parameters

[[103+97+153][137+126+41];[141+122+99][59+113+23]] ...
[[185+171+73][191+30+40];[214+25+206][133+197+129]][0]

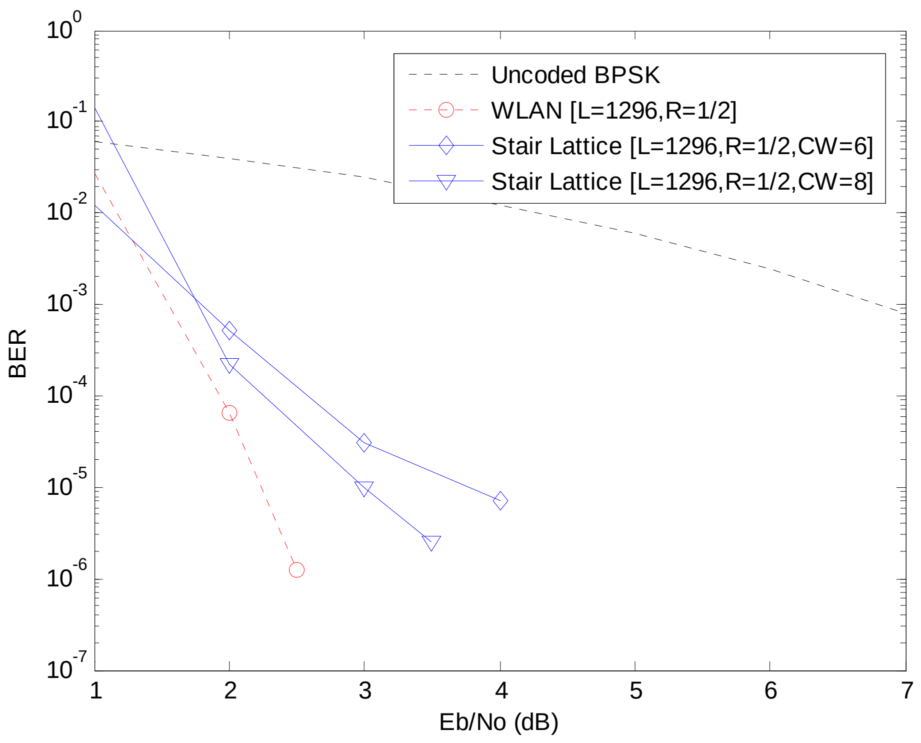


Figure 4.43 BER performance of the improved Stair Lattice codes and WLAN code at code length 1,296 bits and code rate 1/2.

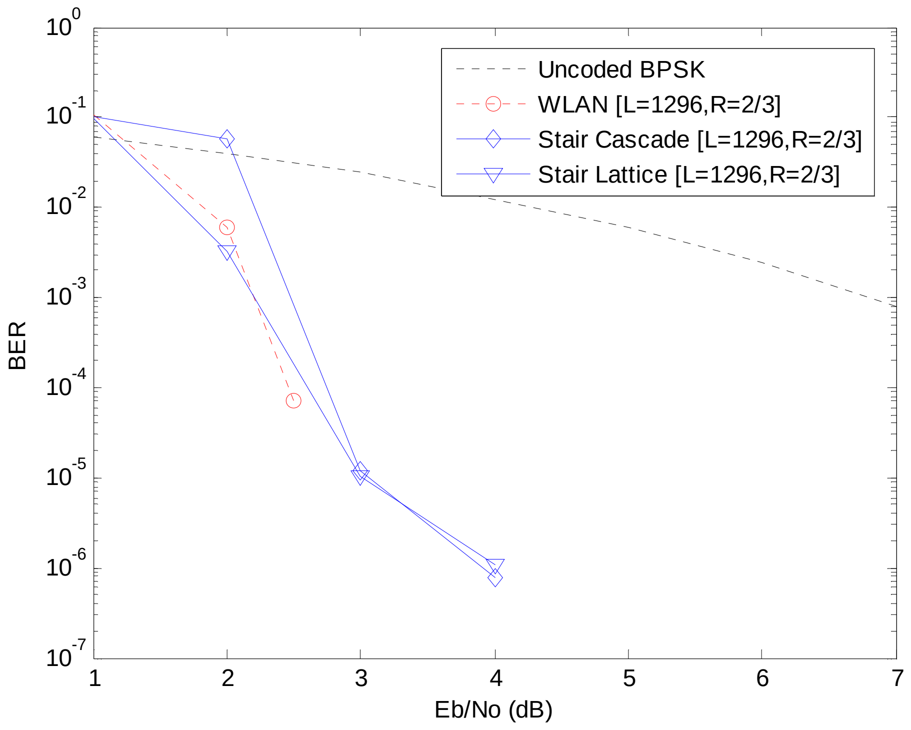


Figure 4.44 BER performance of Stair codes and WLAN code at code length 1,296 bits and code rate 2/3.

The simulation result shows that the BER performance of the Stair codes are still suboptimal to WLAN code, but closer up to about 0.5 dB at the BER 10^{-5} as shown in Figure 4.44. However, the Stair codes still suffer from a high error floor.

At code rate 3/4 the Stair Cascade codes is composed of three submatrices and an identity matrix with the following slope parameters

[45+159+276+72+161+211][50+174+221+273+81+311] ...
[87+125+41+105+261+271][0]

Meanwhile, the Stair Lattice codes is composed of three four-divided submatrices and an identity matrix with the following slope parameters

[[7+19+16][60+68+54];[106+139+59][161+102+66]] ...
[[11+21+49][46+134+89];[147+138+48][3+152+158]] ...
[[42+87+145][88+4+109];[62+140+144][119+115+104]] [0]

As shown in Figure 4.45 the simulation result shows that the BER performance of the Stair codes is still poorer to that of the WLAN code, but as close as about 0.5 dB at the BER 10^{-5}. However, the Stair codes still suffers from a high error floor.

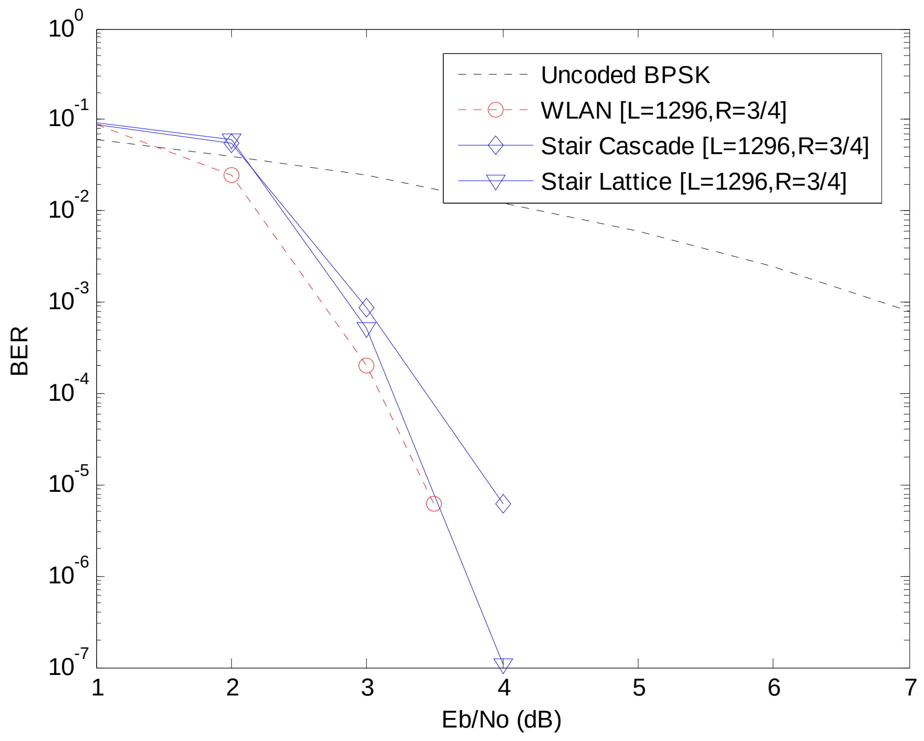


Figure 4.45 BER performance of Stair codes and WLAN code at code length 1,296 bits and code rate 3/4

At code rate 5/6 the Stair Cascade codes is composed of five submatrices and an identity matrix with the following slope parameters

[62+189+120+30+50+110][34+73+84+78+180+185][97+152+150+172+8+176] ...
[89+153+4+105+76+158][199+22+140+23+7+201] [0]

Meanwhile, the Stair Lattice codes are composed of five four-divided submatrices and an identity matrix with the following slope parameters

[[78+16+21][105+27+85];[61+100+67][71+36+10]] ...
[[48+58+98][12+52+5];[19+11+103][7+84+81]] ...
[[72+75+20][90+89+95];[31+94+34][51+66+93]] ...
[[2+40+62][37+41+53];[33+73+13][64+92+22]] ...
[[87+65+74][59+49+101];[47+30+82][15+45+102]][0]

As shown in Figure 4.46 the simulation result shows that the BER performance of the Stair codes is still poorer to that of the WLAN code, but closer as about 0.5 dB at the BER 10^{-4}. Here, the error floor performance of Stair codes is not clearly depicted.

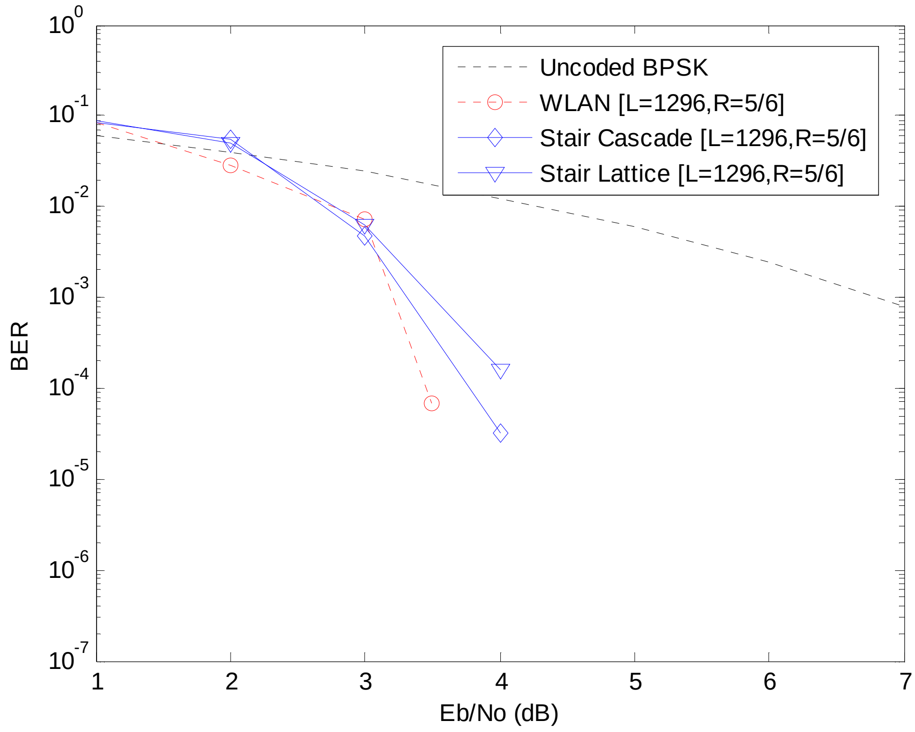


Figure 4.46 BER performance of Stair codes and WLAN code at code length 1,296 bits and code rate 5/6

Regarding to their good BER performance at high code rate, it is interesting to investigate the performance of the Stair codes at code rate 5/6. At short code length of 648 bits, the Stair Cascade code is composed of five submatrices with column weight 6 and an identity matrix with the following slope parameters

[2+14+38+39+36+107][23+10+25+73+13+37][47+58+55+20+86+98] …
[68+46+24+40+52+105][28+74+77+67+90+57][0]

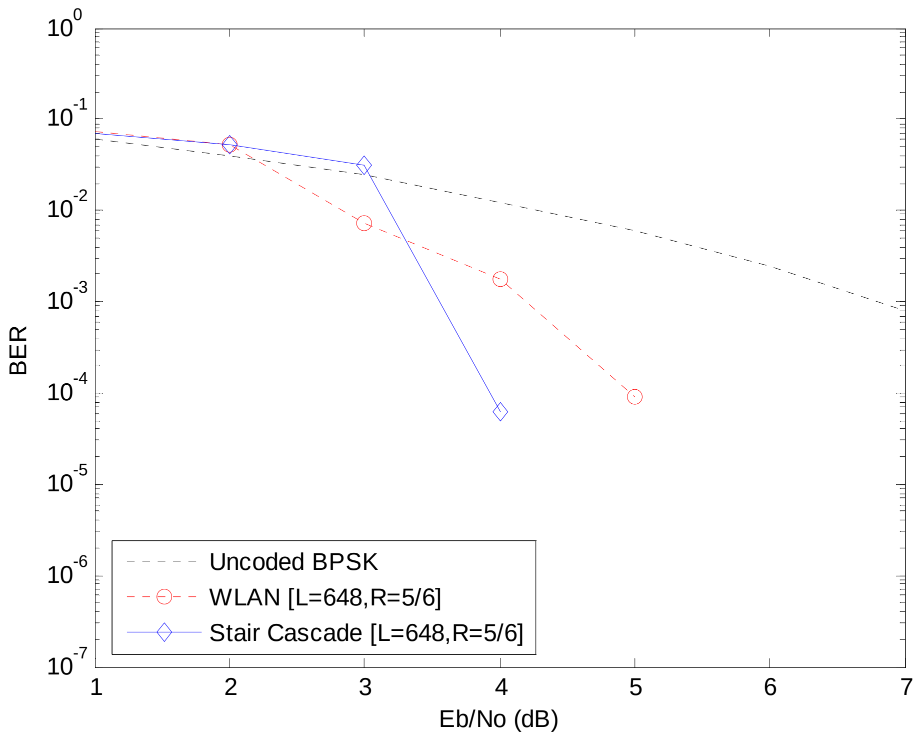


Figure 4.47 BER performance of Stair codes and WLAN code at code length 648 bits and code rate 5/6.

As shown in Figure 4.47 the Stair Cascade code outperform significantly the WLAN code by about 1 dB at the BER 10^{-4}.

At longer code lengths, the performance of the Stair codes at high rate is put in benchmarking with that of the WLAN codes. For code length 1,296 bits and 1,944 bits, the Stair Cascade code is composed of five submatrices with column weight 6 and an identity matrix with the following slope parameters

[62+189+120+30+50+110][34+73+84+78+180+185][97+152+150+172+8+176] …
[89+153+4+105+76+158][199+22+140+23+7+201][0]

and

[11+241+61+256+69+191][85+320+251+144+52+172][80+68+213+242+134+16] ...
[7+246+45+294+18+226][236+273+142+153+81+63][0]

respectively.

As shown in Figure 4.48, at longer code lengths, the performance of the Stair Cascade codes at code rate 5/6 is relatively more comparable with that of the WLAN codes. The Stair Cascade code with code length 1,944 bits performs relatively better and approaches closer the performance of WLAN codes.

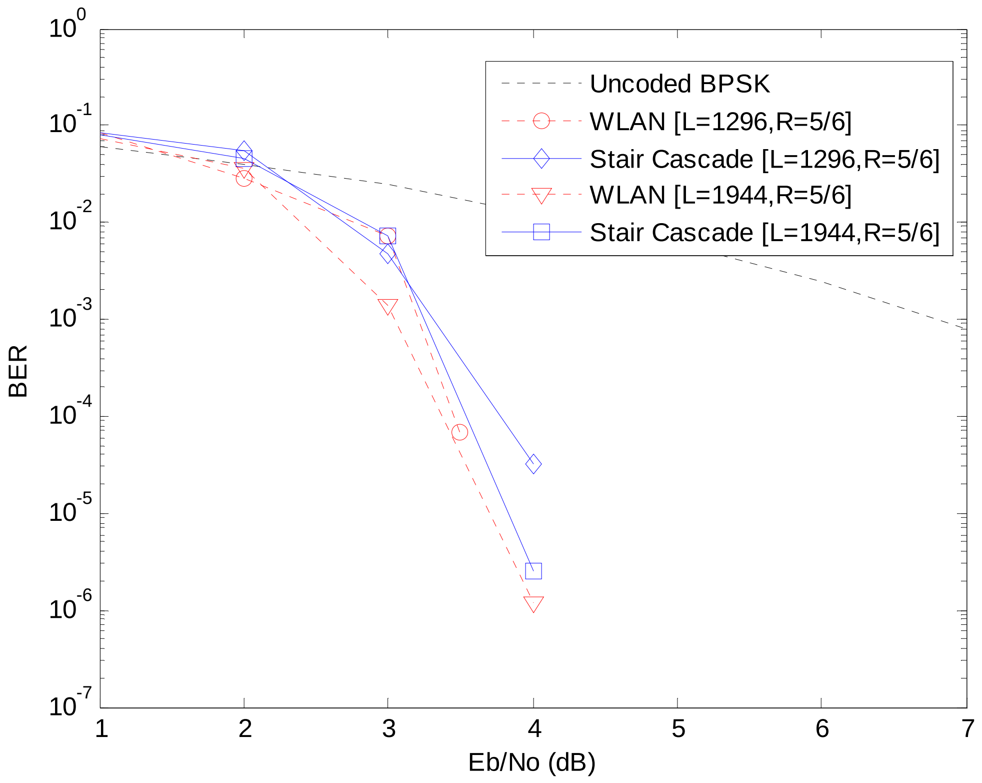


Figure 4.48 BER performance of Stair Cascade codes and WLAN code at code length 1,296 bits and 1,944 bits and code rate 5/6

In addition to their BER performance, it is also of interest to investigate their error floor performance. The Stair Lattice code with code length 648 bits and code rate 5/6 has the following parameters

[[40+43+53][6+28+20];[22+24+23][25+7+8]] …
[[49+15+14][36+33+1];[52+26+0][21+32+17]] …
[[30+39+2][9+46+5];[50+38+5][28+53+35]] …
[[11+10+48][15+4+23];[2+0+24][31+26+40]] …
[[45+20+42][30+25+16];[27+18+12][46+47+43]][0]

The Stair Cascade code with code length 1,944 bits and code rate 5/6 is constructed with the following slope parameters

[11+241+61+256+69+191][85+320+251+144+52+172][80+68+213+242+134+16] ...
[7+246+45+294+18+226][236+273+142+153+81+63][0]

At lower code rate 3/4, the error floor performance of the Stair Lattice code with code length 1,296 bits was also simulated. The code is constructed with the following slope parameters

[[7+19+16][60+68+54];[106+139+59][161+102+66]] ...
[[11+21+49][46+134+89];[147+138+48][3+152+158]] ...
[[42+87+145][88+4+109];[62+140+144][119+115+104]][0]

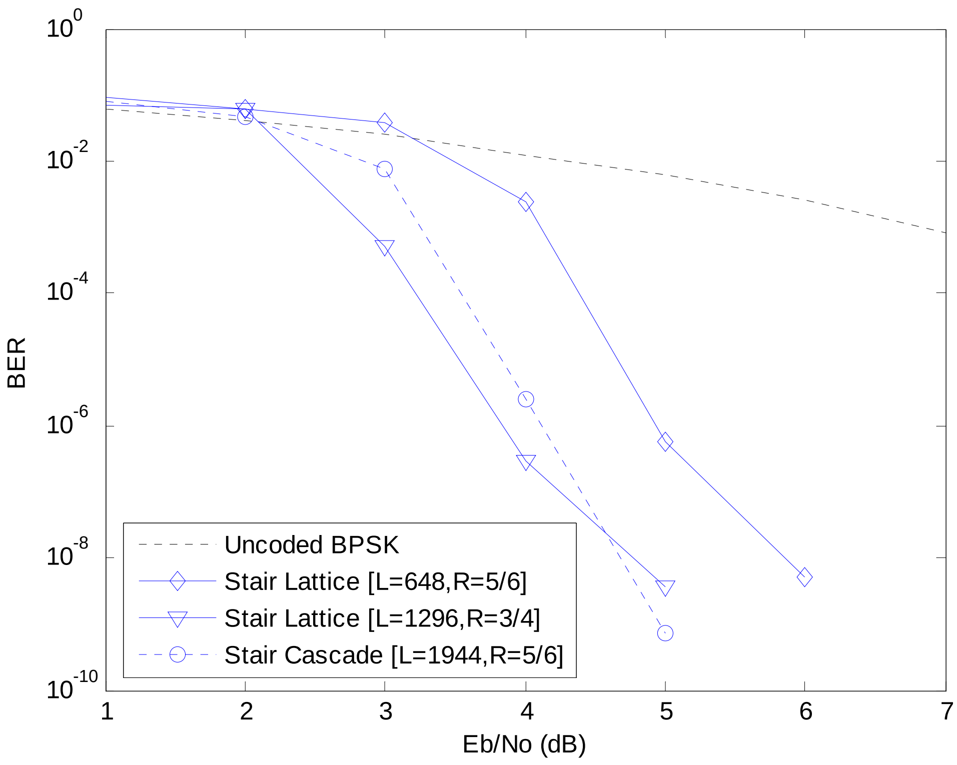


Figure 4.49 Error-floor performance of Stair codes at high code rate

As shown in Figure 4.49, at code rate 5/6 the shorter Stair Lattice code can achieve a BER of about 5x10^{-9} at E_b/N_o of 6dB, however, with the error floor tendency. While the longer Stair Cascade code can achieve a BER of about 7x10^{-10} at E_b/N_o of 5dB. At code rate 3/4, the Stair Lattice code can achieve a BER of about 3.6x10^{-10} at E_b/N_o of 5dB, however, with the error floor tendency.

Impact of Girth Degree on Code Performance

Most of Stair codes contain a number of four-cycles. It is difficult to create Stair codes for code rate 3/4 and 5/6 without four cycles. However, the existing of four cycles does not lead to significant degradation of the code performance. Even, at higher code rate the

Stair codes perform best in either BER performance or error floor performance. As described in [TJVW04, ZZ04] not all short cycles are equally harmful if the cycle conditioning is effectively done. In addition to that, the number of cycles of girth 4 is not enough to possibly degrade the decoding performance [ZP01d].

It is of great interest to understand what number of four-cycles leads to degradable code performance. Stair Cascade codes with code length 1,296 bits, code rate 1/2 and column weight 6 in different girth degree are simulated over AWGN channel at the a maximal number of iterations of 20. The Stair Cascade codes have the following slope parameters

Code	Slope parameter	Average girth degree of girth 4
1	[101+293+240+436+564+290][0]	0
2	[366+371+608+521+275+122][0]	2
3	[10+20+30+40+60+80][0]	24
4	[10+20+30+40+50+60][0]	40

The first code has no girth 4, while the others have all variable nodes with girth 4. The difference of the latter lies on average girth degree of girth 4, that is 2, 24 and 40. The codes 1 and 2 are constructed using the proposed code construction methods. Meanwhile, the other codes with some higher average girth degree of girth 4 are constructed by breaking the rule of avoiding four-cycles. Such codes with very high average girth degree of girth 4 are never resulting from the permutation of the slope parameter.

The impact of the existence of girth 4 on the code performance is investigated by comparing code 1 and 2. As shown in Figure 4.50 the performance of code 2 with girth 4 is comparable with that of code 1 with no girth 4. It is a proof that to some extent, the existence of four-cycles with only a relative low number is not harmful enough for the BER performance. On the other hand, the codes with relative high average girth degree show their poor performance.

Furthermore, Stair Lattice codes with code length 1,296 bits, code rate 3/4 and column weight 6 in different girth degree are simulated over an AWGN channel at the maximal number of iteration of 20. The Stair Lattice codes have the following slope parameters

Code	Slope parameter	Average girth degree of girth 4
1	[7+19+16 60+68+54; 106+139+59 161+102+66] [11+21+49 46+134+89; 147+138+48 3+152+158] [42+87+145 88+4+109; 62+140+144 119+115+104][0]	3
2	[10+20+30 111+67+6; 20+30+40 144+33+106] [60+80+100 23+64+39; 30+50+70 45+94+26] [50+70+90 115+52+117; 70+90+110 151+4+135][0]	24
3	[10+20+30 50+70+90; 20+30+40 60+80+100] [60+80+100 70+90+110; 30+50+70 80+100+120] [50+70+90 20+40+60; 70+90+110 50+70+90][0]	96

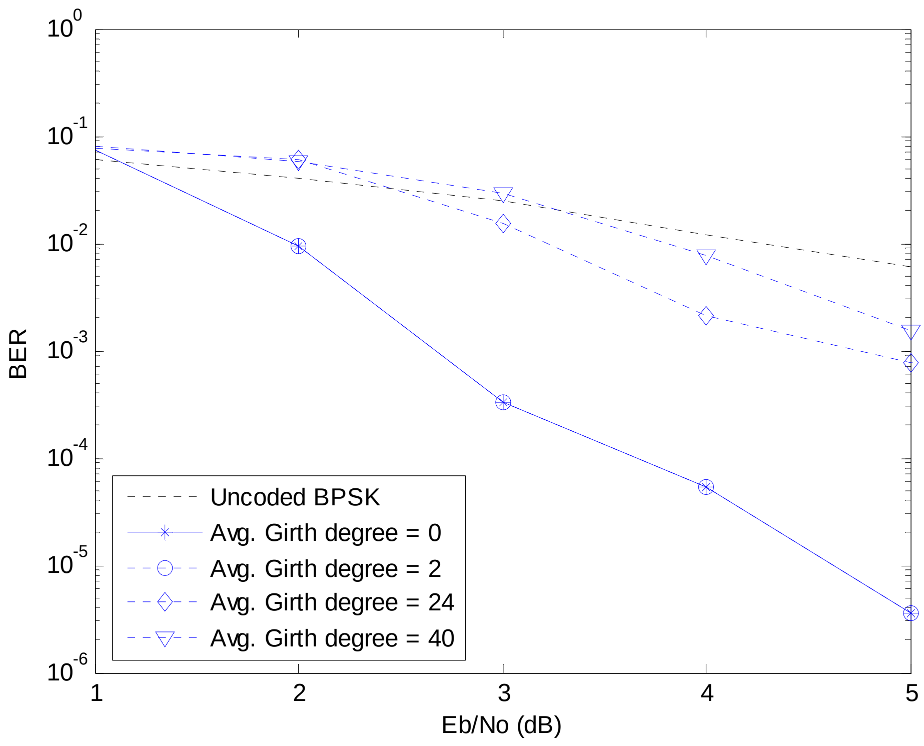


Figure 4.50 Impact of girth degree of four cycles on BER performance of the length-1,296, rate-1/2 Stair Cascade codes

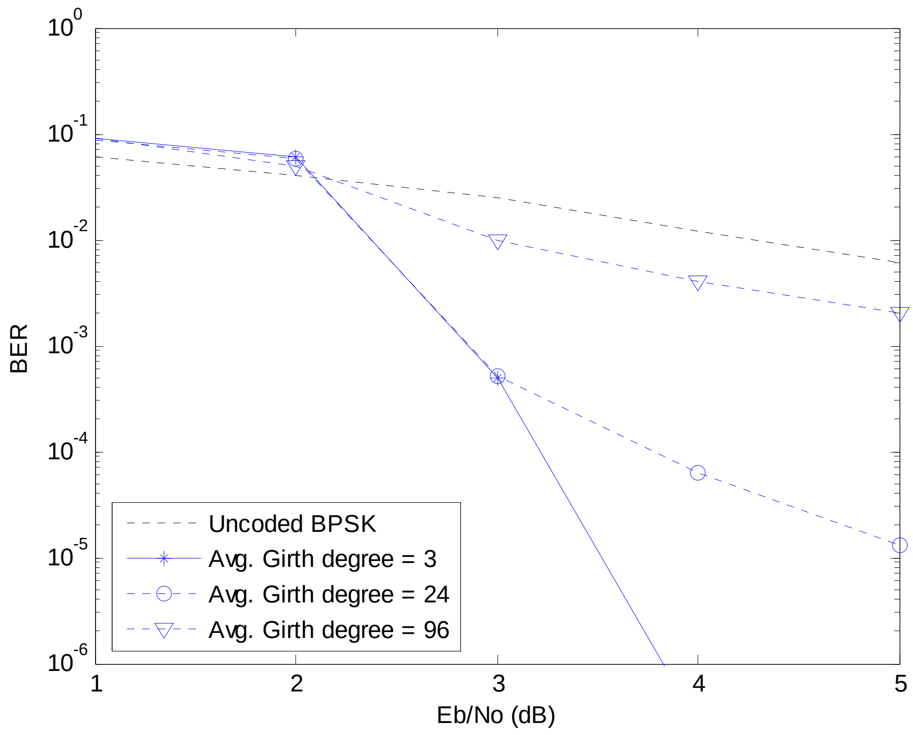


Figure 4.51 Impact of girth degree of four cycles on BER performance of the length-1,296, rate-3/4 Stair Lattice codes

The first Stair Cascade code is constructed with the proposed code construction method; meanwhile, the others are constructed by breaking the rule of avoiding four-cycles to allow the codes to have very high girth degree. The latter codes are never resulting from the permutation of slope parameter.

Again, as shown in Figure 4.51, to some extent, the existence of four-cycles is not harmful enough for the BER performance.The Stair Lattice code with girth degree 3 shows the acceptable BER performance. While, the Stair Lattice codes with very high girth degree perform poorly.

In general, the simulation results show that the girth degree gives more significant impact on code performance than the girth. With the same average girth, the average girth degree makes difference in code performance.

4.6 Conclusion

The work in this chapter presented a class of low density parity check (LDPC) codes with a very efficient encoder. Their construction methods as well as their characteristics have been discussed. The Stair codes are designed such that the codes have very low encoding complexity as a proposed solution for LDPC codes' drawback without degradation in code performance.

In principle, the proposed codes have a parity-check matrix designed by composing permutation submatrices and an identity submatrix. The way of constructing the permutation submatrices differentiate the codes into cascade and lattice structure. In comparison of both types of structures, the lattice structure can support different code rates, while the cascade structure can only cope with limited code rate. However, the lattice structure requires more shift register and addition operation at the same code rate than the cascade structure.

The identity submatrix becomes the characteristic core for the proposed codes. It plays an important role that makes the encoding process very efficient at one side and at the other side determines mainly the code performance. In the encoding process, the identity submatrix produces directly the parity check bits for the encoded messages.

The permutation submatrices are simply constructed by positioning the non-zero elements according to slope parameter generated by a random permutation process. The choice of slope parameter shall guarantee that no short-four-cycles exist in the proposed codes. Two code construction methods are proposed, namely, the slope method and the girth degree method. Both methods serve for proving the possible existing of the short-four-cycles.

In fact, the probability that the Stair codes contain the short-four-cycles with higher code rate is increased. The Stair codes with code rate 1/2 and 2/3 are relatively easier to generate without containing short-four-cycles. But, at higher code rate 3/4 and 5/6 the existence of the short-four-cycles could not be avoided. However, it is of interest to note that in spite of the existence of the short-four-cycles, the Stair codes are not suffering from performance degradation. They show a performance comparable to some benchmarking LDPC codes, such as random MacKay codes. Even, at short code length and high code rate, the Stair codes are able to outperform the difference families codes and the WLAN codes. This proves that the number of short-four-cycles in the Stair codes can be considered as still harmless for decoding performance.

However, the drawback of the Stair codes is observed at lower code rate 1/2 and 2/3. Although the codes show good threshold performance in the water-fall region, the codes are suffering from high error floor at high SNR region. This poor floor performance is correlated with the stair structure represented by the identity submatrix. The existing of degree-1 variable (parity) nodes is considered as responsible for such unacceptable performance. Their contribution to update of message passing during the iterative decoding process is relative insignificant. They only bounce the message coming from their associated single check nodes. In addition, the probability that the trapping sets considered as source of poor error-floor performance is higher than the number of variable (parity) node increases or correspondingly, the code rate decreases. To some extent, the floor performance of the Stair codes could be improved by appropriately increasing the column weight of the permutation submatrices. However, this improvement is achieved at cost of higher decoding complexity. The improvement efforts have to be made and of interests for the future work. At higher code rate, the Stair codes can achieve good floor performance at BER 10^{-9}.

In comparison with the WLAN codes, the Stair codes are still relative suboptimal, especially at lower code rate 1/2 and 2/3. But, at higher code rate 3/4 and 5/6, the performance of both codes can be considered as comparable. Even, at short code length 648 bits with high code rate 5/6, the Stair codes are capable of outperforming the WLAN codes. In spite of poorer performance in general, the encoding complexity of the Stair codes is relative lower than that of the WLAN codes. Thank to their stair structure of degree-1 variable represented by the identity submatrix in the parity-check matrix, the parity bits of the encoded message can be derived directly from the parity-check matrix. Meanwhile, with their stair structure of degree-2 variable the WLAN codes do substitution process to produce their parity bits, what leads to longer encoding latency time. However, in decoding complexity, the Stair codes require relatively more intensive computation due to their higher non-zero elements in their parity-check matrix.

In general, the proposed Stair codes show their best performance mostly at higher code rate. Their relative simpler code construction and lower encoding complexity are the main advantages of these codes and could be of the reasons to choose them for any WLAN application requiring high-rate transmission.

Again, here it is shown that the code performance is not only determined by girth as supposed in the graph theory, but also is derived by its girth degree. The latter parameter gives even more significant impact on the code performance than the first one. With the same average girth, the average girth degree can degrade the code performance when to some extent its high number is large enough to reduce the nodes interdependency so that the work of the message-passing based decoding algorithm can be not effective.

CHAPTER 5

TWO-STAGE DECODING FOR ITERATION REDUCTION

5.1 Background

In this chapter a novel simple method to enhance the performance of LPDC codes in high-order modulation is presented. In this method a two-stage decoding process making use of a feedback mechanism for improving the decoding convergence is discussed. By making use of soft-output of decoder, the method is able to improve the reliability of log-likelihood ratios (LLR) of bits in the decoding process, which leads to a reduction of the average number of decoding iterations. The method is simulated under different conditions of LDPC codes, including code length, code types, maximal iteration number, and bit constellation as well. Regarding to its possible application in WLAN environment, it is interesting to test the efficacy of the proposed method in case of short-length code, small iteration number, and high-order modulation. The simulation over AWGN channel shows that the two-stage decoding strategy contributes significantly to the reduction of decoding iterations at the waterfall region without sacrificing the BER performance.

5.2 Concept and Methods

Several publications have proposed some methods to reduce the decoding complexity by reducing the number of iterations, with or without negligible deterioration in error correction performance. In [Zimm04], the authors used a threshold rule to decide whether a variable node of LDPC decoder should update its information in subsequent iterations of the decoding process or it can be considered as converged. [GP05] proposed a method of reducing the number of iterations which is based on detecting the cycles in the soft-word sequence and stopping the decoder when a cycle is detected. The proposed stopping criterion assumes that a cycle is found, when a soft word is identical to a soft word of a previous iteration. [LK08] proposed an algorithm to eliminate unnecessary parity check computations by exploiting the structure of dual-diagonal LDPC codes. Once all data bits are successfully decoded, the decoder can stop immediately without waiting for the remaining parity bits to converge.

In this work, the two-stage decoding has the same objective, i.e. to reduce the number of decoding iterations without significant deterioration in the BER performance [HEK07c]. Inspired by the observation of the development of the LLR during decoding process as described later, the idea of the proposed method is to improve the LLR of a LDPC block by using the information of bit nodes (referred to as soft-output), from which the decisions on the codeword bits are made in the standard decoding. By applying a feedback mechanism after the first decoding stage, the decoding convergence in the second decoding stage shall be improved. For this purpose, a different maximal number of iterations are established in both decoding stages in such a way that the average number of iterations becomes minimal. In practical sense, this idea shall reduce the number of iterations needed to converge. Hypothetically, this additional information can be used to enhance the bit reliability in the second decoding and therefore improves the system performance with less decoding iterations.

The two-stage decoding can be described as following as shown in Figure 5.1. The soft-input of the received LDPC block $\mathbf{L}_i$ is fed into the decoder and iteratively decoded with the sum-product algorithm. After a maximum number of decoding iterations is achieved or the codeword is found, the soft-output vector $\mathbf{L}_\mathrm{o}$ at the decoder output is weighted with a constant weighting factor w_o and fed back as additional information for the original soft-input $\mathbf{L}_\mathrm{i}$. The new soft-input is then fed back into the decoder and iteratively decoded. After several numbers of feedback iterations, the hard-decision on the codeword is made relatively faster. The problem investigated here is to find a good weighting factor w_o and a maximal number of iteration for the second decoding, which leads to the enhancement of the system performance.

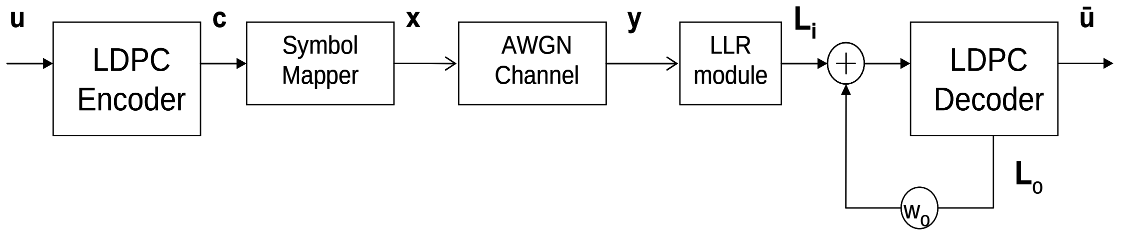


Figure 5.1 Scheme of the two-stage decoding

Mathematically, the scheme of the feedback decoding can be expressed as follows

$$\mathbf{L}_{\mathrm{i}\text{-}2} = \mathbf{L}_\mathrm{i} + w_\mathrm{o}\,\mathbf{L}_\mathrm{o} \tag{5.1}$$

where $\mathbf{L}_\mathrm{i}$ denotes the input LLR for the first decoding process, $\mathbf{L}_{\mathrm{i}\text{-}2}$ denotes the input LLR for the second decoding process, $\mathbf{L}_\mathrm{o}$ denotes the output LLR from the first decoding process, and w_o denotes weighting factor for the soft-output.

For a better description, the two-stage decoding can be modeled by two decoding stages as shown in Figure 5.2. The first decoding stage executes iteratively the decoding process for the LLR of the information received from the transmission channel, $\mathbf{L}_\mathrm{i}$. The process at the first stage can be simply modeled as follows

$$\mathbf{L}_\mathrm{o} = \mathrm{decoding}(\mathbf{L}_\mathrm{i}) \tag{5.2}$$

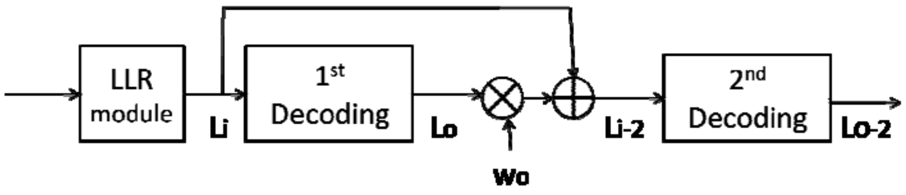


Figure 5.2 Two-stage decoding in LDPC decoder

Thereafter, the result of the first decoding stages $\mathbf{L}_o$, is weighted by a scalar weighting factor w_o and then added to the received LLR $\mathbf{L}_i$. The process at the second stage can be simply modeled as follows

$$\mathbf{L}_{o\text{-}2} = \text{decoding}\,(\mathbf{L}_i + w_o \times \mathbf{L}_o) \qquad (5.3)$$

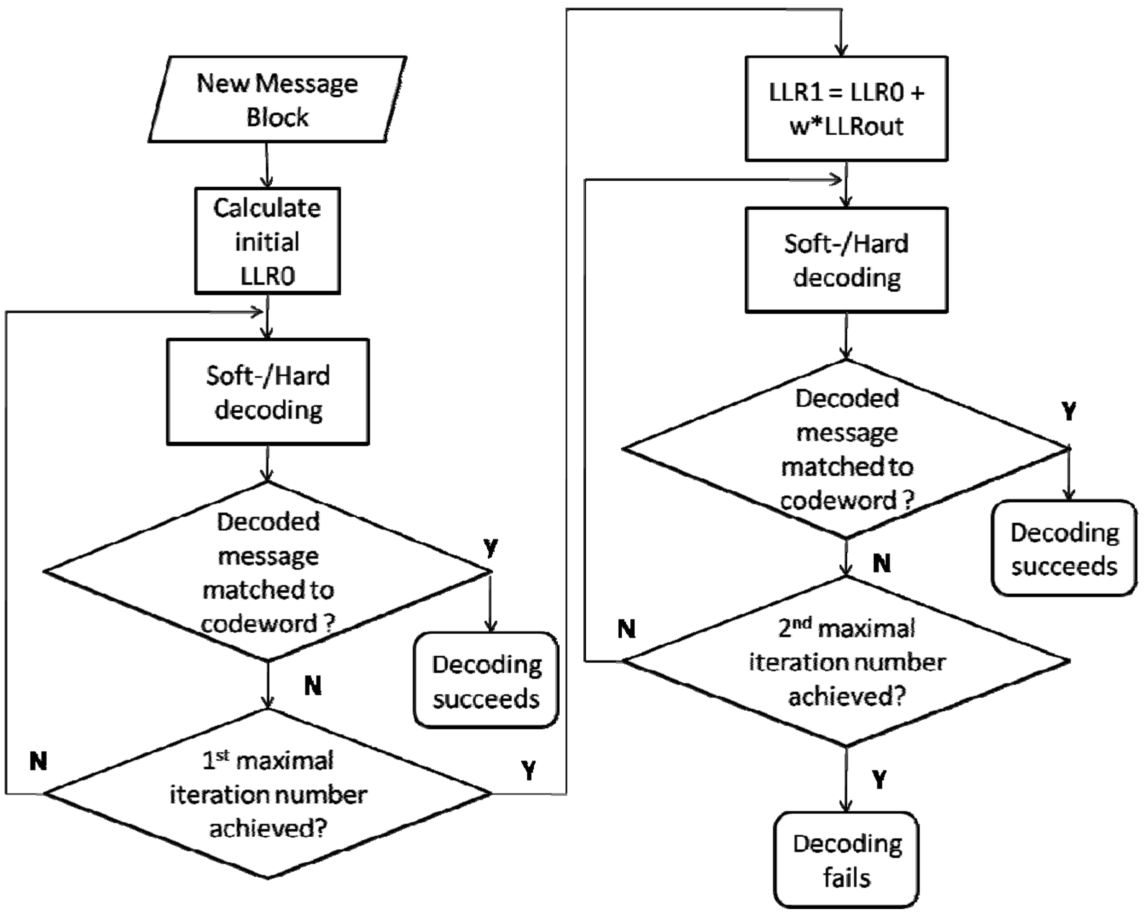


Figure 5.3 Flow diagram of two-stage decoding algorithm

The algorithm of two-stage decoding as depicted in Figure 5.3 can be summarized as follows

1. The received information bits are softly decoded by calculating each Log-likehood ratio P(0)/P(1) of each bit (respectively the LLR) and at the end, thresholding the LLRs to bit value (+1 or -1)
2. Comparing received bits with transmitted codeword bits. If both are matched, the decoding continues with new received bits. If no match is found and the first maximal iteration number is not achieved yet, the LLRs are subject to further soft decoding by going back to step 1.
3. If no match is found and the maximal iteration number is exceeded, the LLRs are scalar-weighted and added to the initial LLRs in the feedback mechanism.
4. The new LLRs are softly decoded and thresholded.
5. If decoded bits do not match with the codeword bits, the LLRs are subject to further decoding. If the maximal iteration number is exceeded, the decoding fails.

In the two-stage decoding, the number of iterations to decode a message block is the sum of iteration numbers in two decoding processes. The setting of the maximal number of iterations becomes important to achieve the significant reduction of decoding iteration.

The idea of the proposed method is coming from the observation of the development of LLR during decoding process. As shown in figure 5.4, it is realized that there are six curve types representing the development of LLR which is taken at a SNR in the waterfall region. From the observation, it is realized that typically the LLRs of bits subject to correction develop across the axis (LLR = 0) to be correctly decoded (line type 2 and 4 in Figure 5.4). If this is not the case at the last iteration, the bits remain with false LLR.

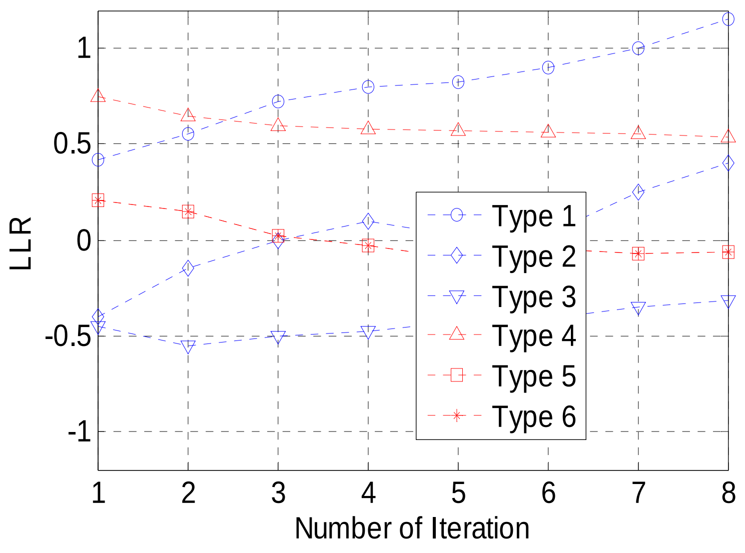


Figure 5.4 Typical development of LLRs at the waterfall region

As highlighted before, the feedback mechanism involved in two-stage decoding is used to reduce the number of iterations and therefore reduces the decoding complexity. As shown in Figure 5.5 the development of LLR is almost not changing during the iteration under conditions with low SNR leading to poor BER performance. The change of LLR begins to become significant from the beginning of the waterfall region (in this case 9dB) and gets higher as SNR increases. This change of LLR can be advantageous as additional information to reduce the decoding iteration.

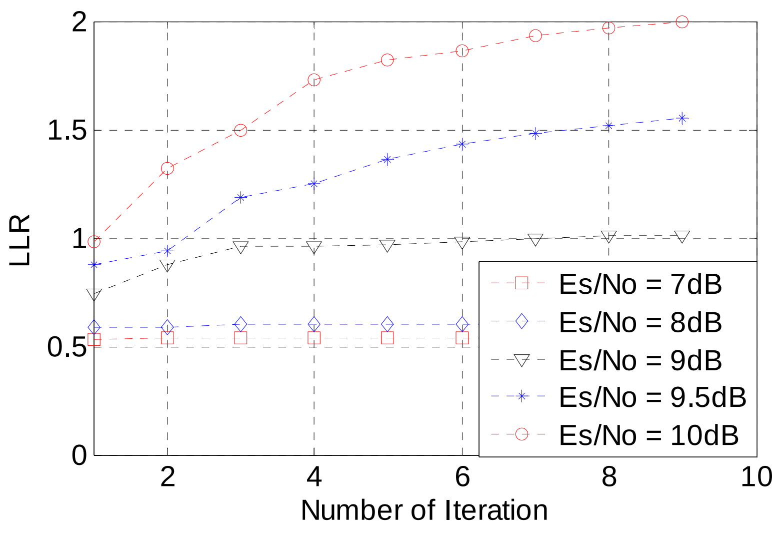


Figure 5.5 Development of LLRs depending on SNR

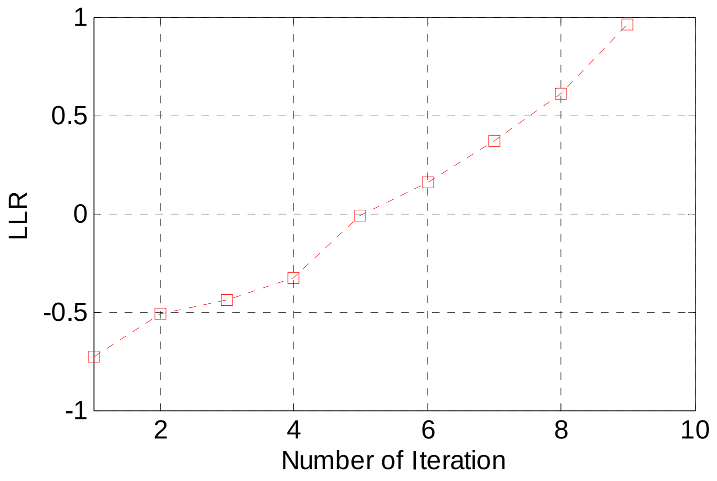
The idea of feedback decoding in the proposed decoding scheme is to accelerate the development of LLR of bits subject to false decoding into its corresponding correct decoded value. Figure 5.6b depicts the simplified model of the development of LLR of a bit subject to false decoding during the increased iteration number. Instead of following the relative slower development of the LLR in generic decoding in feedback mechanism, the first decoding stops after a certain maximal number of iterations and the resulting LLR becomes a part of the LLR leading to an accelerated decoding process in the second decoding stage.

5.3 Simulation Results

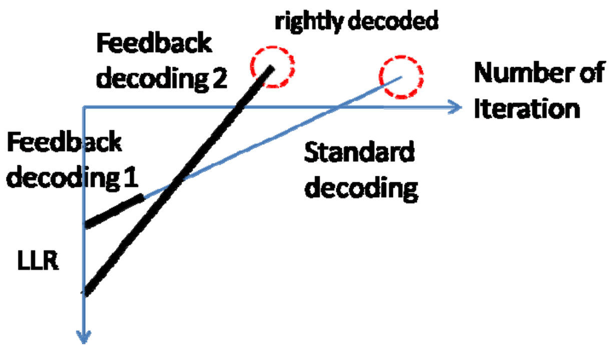
The objective of this part of research is to achieve comparable performance with less iterations for LDPC codes working at lower SNR.

To prove the efficiency of the proposed method, the simulation is conducted as follows

1. As simulation basis, the 16QAM system with irregular/random MacKay-Neal code with column weight of three and code length of 1,200 bits is established to test the efficiency of feedback decoding using an AWGN channel.
2. Two feedback parameters leading to the achievement of the objective are investigated, i.e. the weighting factor and the combination of the maximal number of iterations for two decoding stages.
3. The feedback decoding is tested on different condition, including type LDPC codes, i.e. random/irregular and structured/regular codes, code length, i.e. short and moderate ones, and on constellation number, i.e. 16QAM and 64QAM. This procedure is to accomplish to prove the generality of the application of the proposed method.



(a)



(b)

Figure 5.6 Development of LLR of bit subject to fail decoded in generic decoding (a) and its model denoting decoding improvement using feedback decoding

It is known that the all-zero codewords are adequate for assessing the performance of a linear code with a symmetrical channel and a symmetrical decoding algorithm. However, in case of a high-modulation system, due to the asymmetry of a signal set, i.e. QAM, the *i*-th, $i \in \{0, 1, \dots, L - 1\}$, where L denotes constellation bit level, binary-input component channel is not output symmetric [RU01], and thus, it cannot be assumed that the decoder errors are in the same positions, regardless of which codeword is transmitted. A paper by Hou et al. [Hou01, Lim03] developed a method to make the equivalent binary-input component symmetric channels by introducing "independent and identically (i.i.d.) channel adapters,"

An i.i.d. source generates i.i.d. random variables with $t_i \in \{0, 1\}$, $i \in \{0, 1, \dots, L - 1\}$ with $P(t_i = 0) = P(t_i = 1) = 1/2$. A modulo-2 adder adds LDPC-coded bit x_i and the random number t_i to get $d_i = x_i \oplus t_i$. The multiplier performs the following operation:, $\overline{q_0^i} = q_0^i \cdot \left(1 - 2t_i\right)$, where q_0^i is the log *a posteriori probability ratio* (LAPPR) from the channel output at coding level *i*. The new equivalent binary-input component channel satisfies the required symmetry condition given by as verified in [Hou01], and hence, it can be assumed that the all-zero codeword is transmitted when evaluating system performance [RU01]. For each coding level *i*, the signal set wit the average signal energy E_s is transmitted over AWGN channel, the noise standard deviation is fixed at σ_i^2.

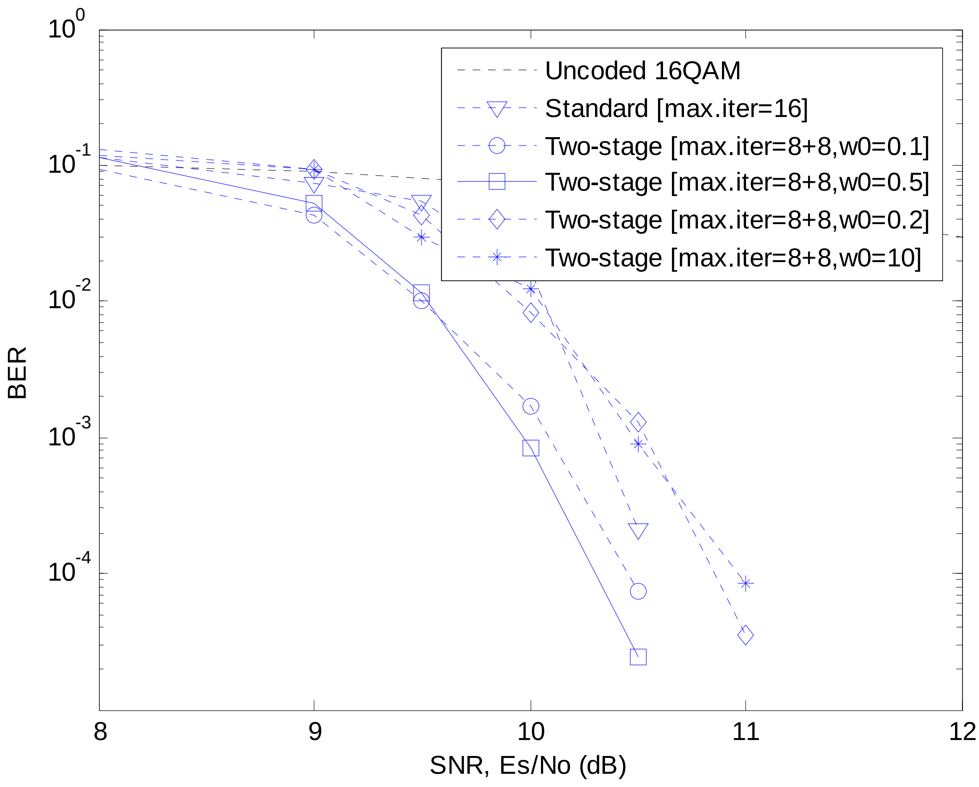
5.3.1 Two feedback parameters

Finding the best combination of two feedback parameters leading to the best performance, i.e. the weighting factor and the maximal number of iterations for two decoding stages would be an exhaustive effort. However, the finding process is simplified in such way that the best value of one parameter is explored while fixing the other parameter at certain values.

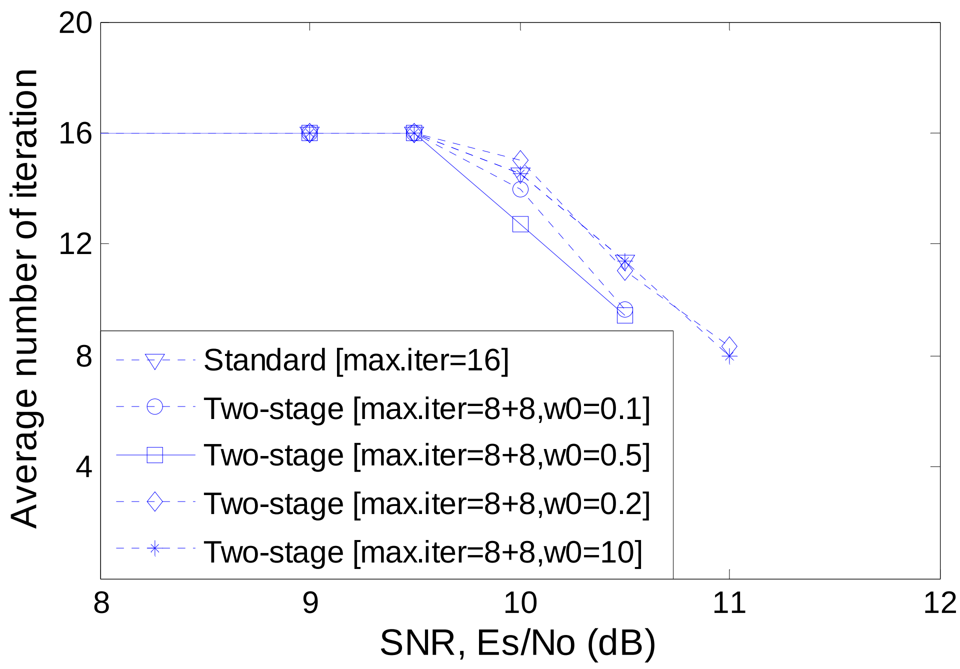
The best weighting factor is explored, while setting the maximal number of iterations as high as possible in practical computation. For this, the value of 16 is chosen because this lower number is quite challenging from a practical point of view in achieving higher decoding speed. In exploring the best combination of the maximal number of iterations, this number is divided into two maximal numbers of iteration for each decoding stage.

The best weighting factor

In order to find the best weighting factor, the maximal number of iterations for each decoding is set to 8 each. Using a heuristics mechanism, some values for the weighting factor w_0 are investigated on the basic simulation, i.e. 0.1, 0.5, 2 and 10. The simulation results show that the weighting factor of 0.5 gives the best achievement in terms of iteration reduction and BER performance [see Figure 5.7]. It is also shown that by applying appropriate weighting factors the two-stage decoding can achieve better BER performance with less iterations.



(a) BER performance



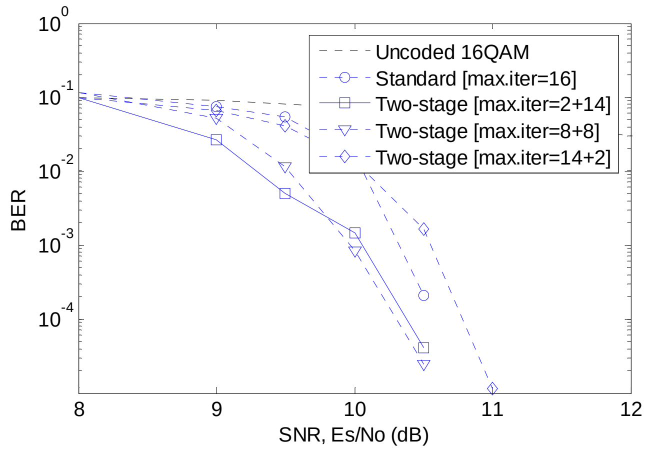
(b) Average number of iterations

Figure 5.7 Comparison of BER performance and Iteration by different weighting factor.

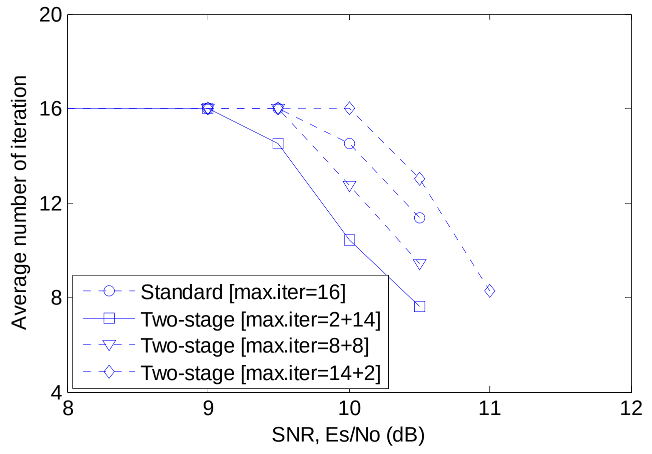
The best combination of maximal number of iteration

Here, the optimal combination of the maximal number of iteration for two decoding stages is investigated. A criterion for fixed maximal number of iterations for the first decoding stage is established. The number is chosen so that it is able to contribute to small numbers of the average number of iterations. The number is set as small as possible, but it is still capable of indicating the LLR change as additional information for the second decoding stage. On the other side, smaller fixed numbers of maximal iterations for the first decoding stage makes that for the second decoding stage greater, which ensures more bit reliability in the decision process.

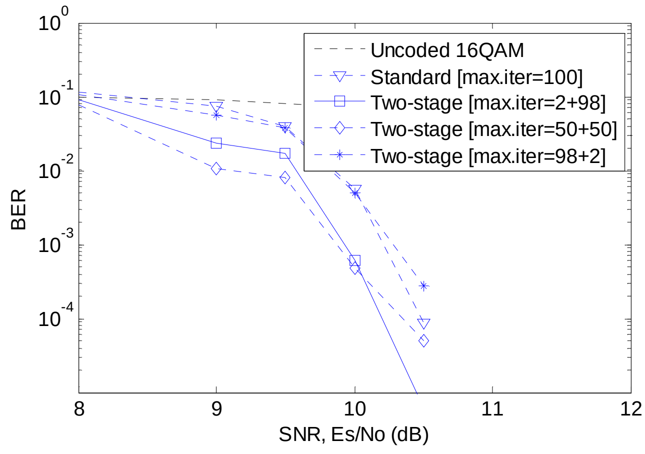
Using the best weighting factor of 0.5, the combination of the maximal number of iterations for both decoding stages is investigated by setting 2 and 14, 8 and 8 and 14 and 2 for the maximal number of iterations of 16. The simulation results show that the combined iteration number of 2 and 14 gives the largest reduction of the iteration numbers [see Figure 5.8]. The combination of smaller iteration numbers for the first decoding stage gives also the best result in the iteration reduction for higher iteration numbers, i.e. 100.



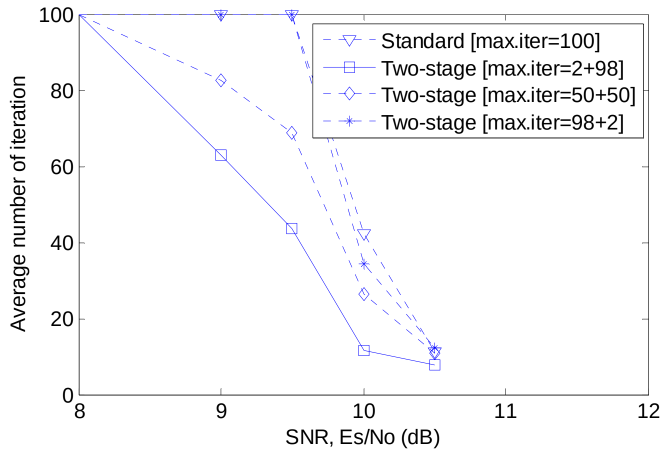
(a) BER performance (16 iteration)



(b) Iteration (16 iteration)



(c) BER performance (100 iterations)



(d) Iteration (100 iterations)

Figure 5.8 Comparison of BER performance and iteration number by different combination of iteration numbers

This fact shows that the small maximal number of iterations in the first decoding makes it possible to cut the useless decoding iteration. Instead, a higher number of iterations for the second decoding stage is set, which leads the decoding processes to better decoding convergence and in turn, less decoding iteration effort.

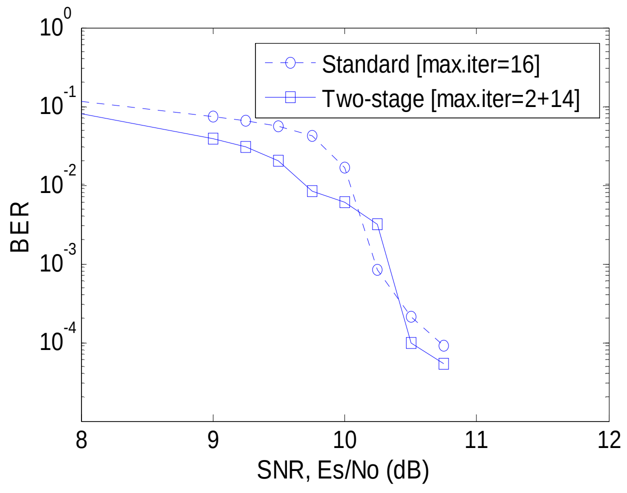
The simulation result proves that the reduction of average number of iteration can be achieved without sacrificing the performance. This resulting decoding improvement takes place in waterfall region.

5.3.2 Generality of Two-stage decoding

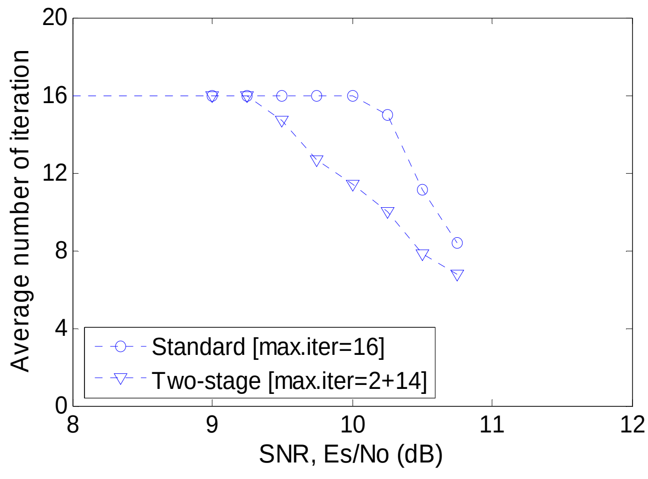
By investigating the feedback decoding on different conditions, the generality of the application of the proposed method will be proven. Firstly, the feedback decoding is tested on a higher modulation system, i.e. 64QAM. Furthermore, the impact of the code length on the efficiency of the feedback decoding is investigated by applying the codes with code length 600 bits and 2,400 bits. At the end, the efficiency of the feedback decoding is investigated for structured codes, i.e. array codes. All simulations are conducted under a different maximal number of iterations, whose reduction becomes the objective of the proposed method. Three maximal numbers of iterations are set for the simulation, i.e. 16, 100 and 1000. In all cases, the feedback decoding takes the maximal number of iterations of two for its first decoding stage, so that the simulations are controlled by three combinations of a maximal number of iterations, i.e. 2 and 14, 2 and 98, and 2 and 998.

Simulation basis

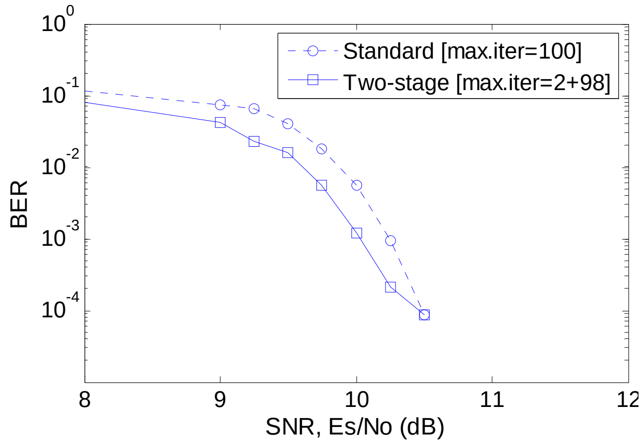
The feedback decoding is investigated on the 16QAM system. The irregular code based on the MacKay-Neal algorithm with column weight of three and code length of 1,200 bits is introduced. The simulation results show that the feedback decoding is effective to reduce the average number of iterations. As shown in Figure 5.9, at the small maximal number of iterations, i.e. 16, the iteration reduction can achieve 35%, however, with no significant improvement in BER performance. The efficiency of the feedback mechanism gets higher at higher maximal numbers of iterations, where better BER performance can be achieved by lower numbers of iterations. The feedback mechanism can reduce the iteration number up to 70% and 95% at the maximal iteration numbers of 100 and 1000, respectively.



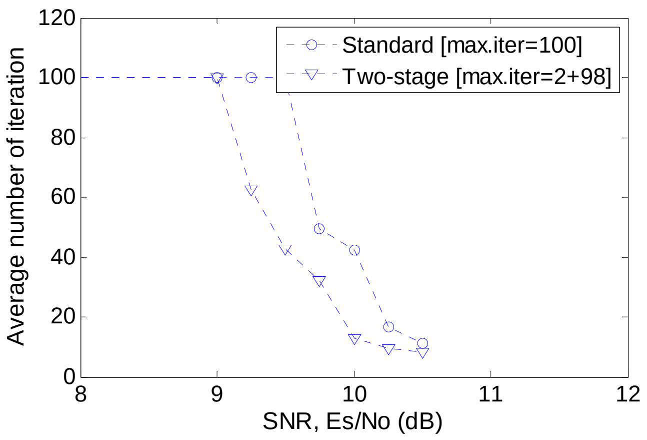
(a) BER performance (16 iterations)



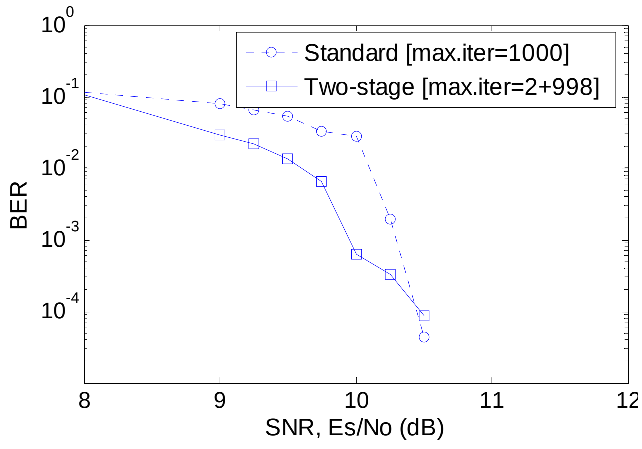
(b) Iteration (16 iterations)



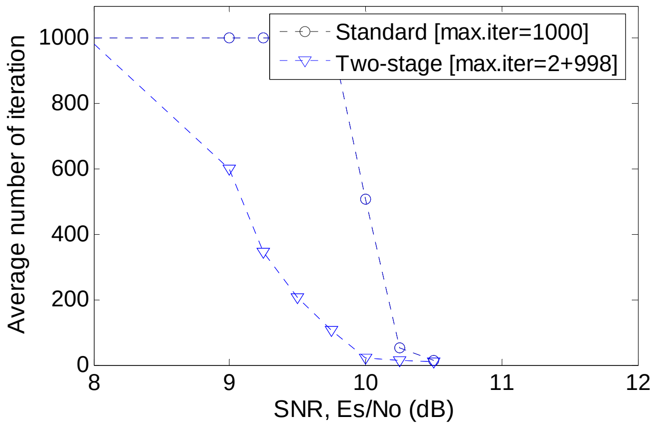
(c) BER performance (100 iterations)



(d) Iteration (100 iterations)



(e) BER performance (1000 iterations)



(f) Iteration (1000 iterations)

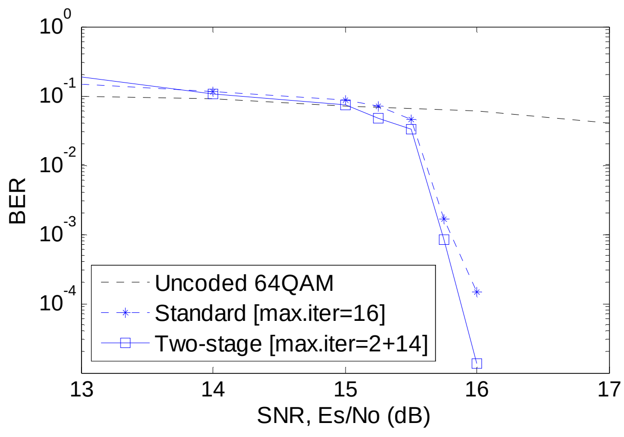
Figure 5.9 Impact of feedback decoding in the 16QAM system, irregular codes with code length 1,200 bits on BER performance and average number of iterations

Higher Bit Constellation

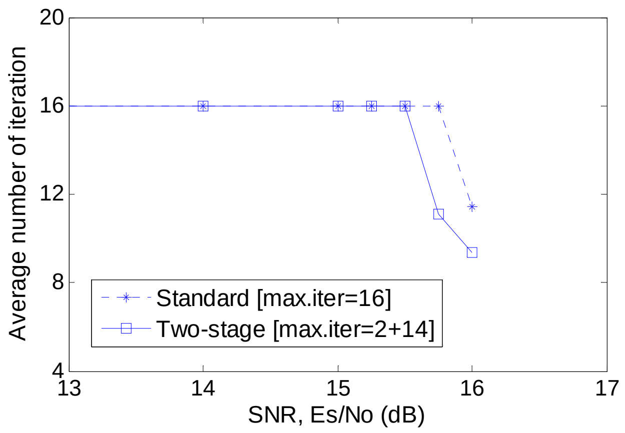
The impact of the feedback mechanism on the iteration reduction is also investigated at higher bit constellations, i.e. 64QAM. As shown in Figure 5.10, the simulation results show that the feedback mechanism can reduce the iteration number up to 80% at maximal number of iterations of 100 and 1000.

Code Length

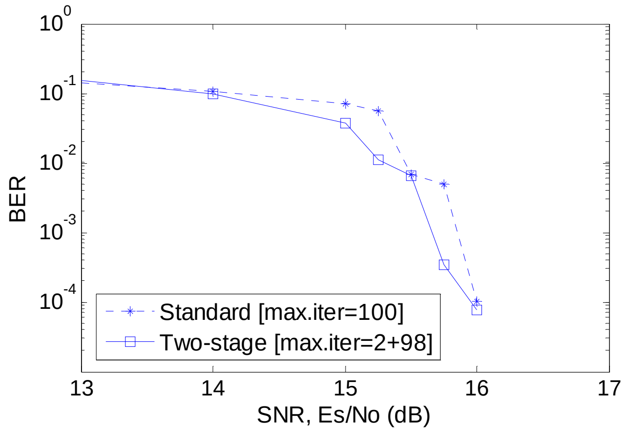
The impact of the feedback decoding on the reduction of the decoding iterations is investigated for different code lengths. The investigation started with short-length LDPC codes. By applying irregular MacKay-Neal LDPC codes with code length of 600 bits as test LDPC code, it is realized in Figure 5.11 that the feedback decoding contributes to the reduction of decoding iterations without BER performance losses for different maximal numbers of iterations.



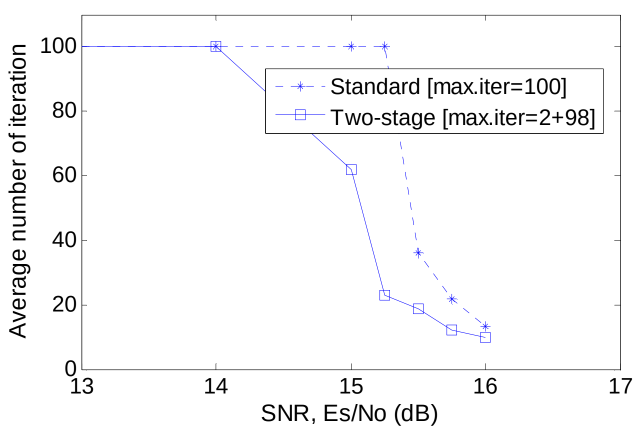
(a) BER performance (16 iterations)



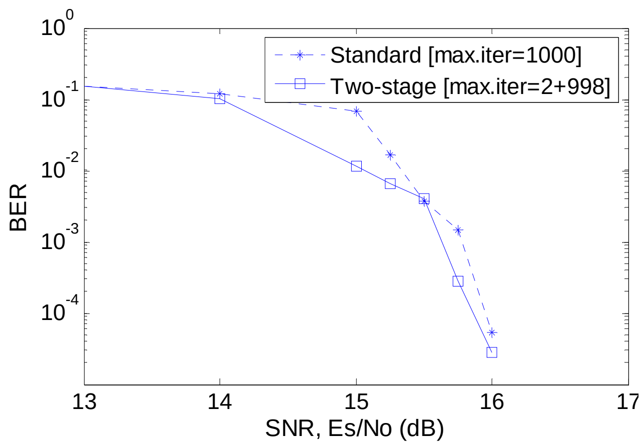
(b) Iteration (16 iterations)



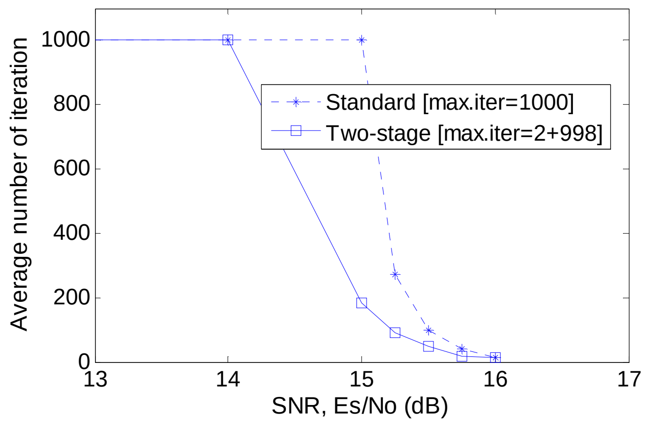
(c) BER performance (100 iterations)



(d) Iteration (100 iterations)

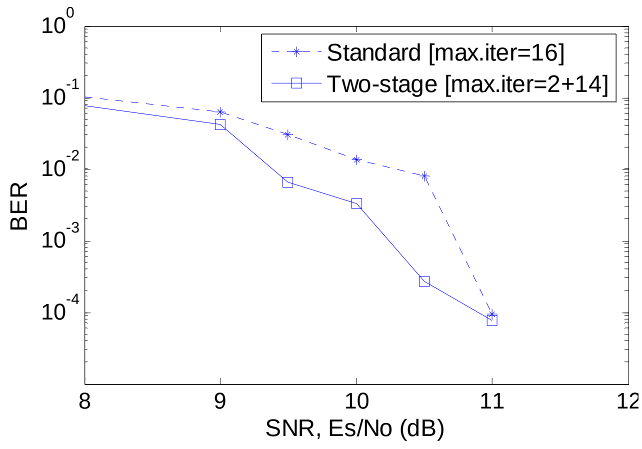


(e) BER performance (1000 iteration)

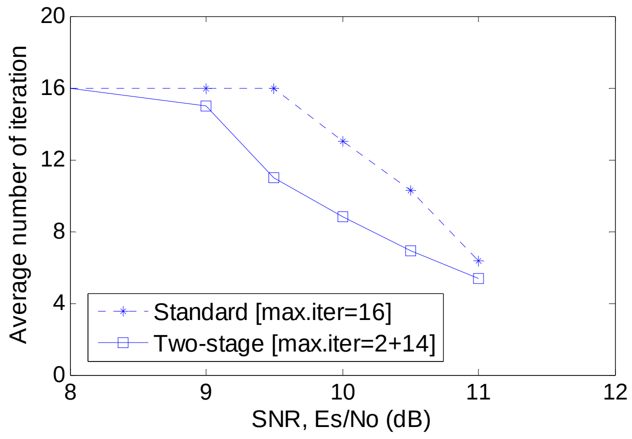


(f) Iteration (1000 iterations)

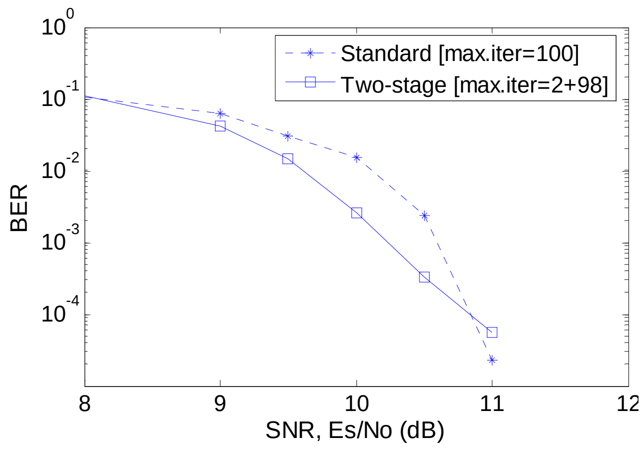
Figure 5.10 Impact of feedback decoding in the 64QAM system, irregular codes with code length 1,200 bits on BER performance and average number of iterations



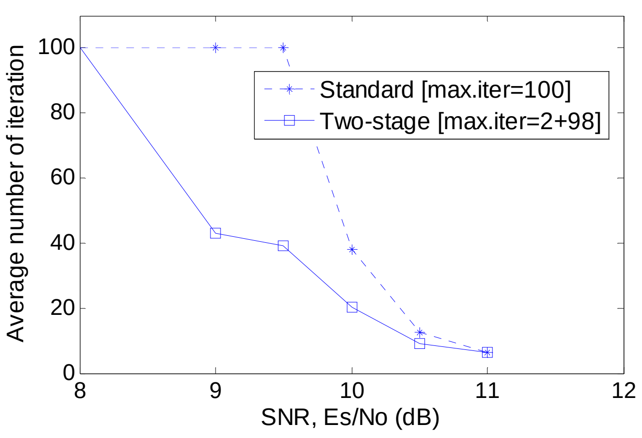
(a) BER performance (16 iterations)



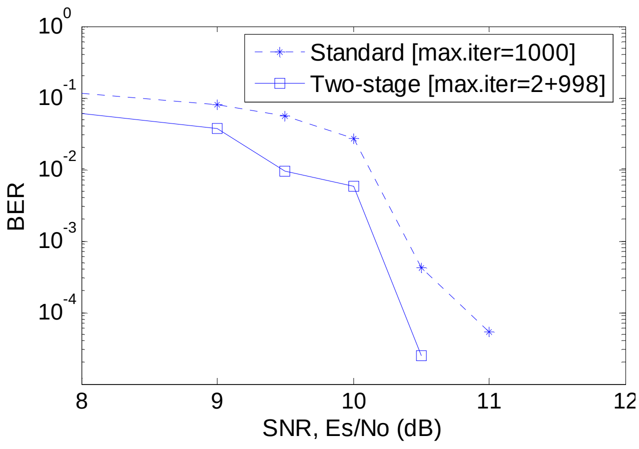
(b) Iteration (16 iterations)



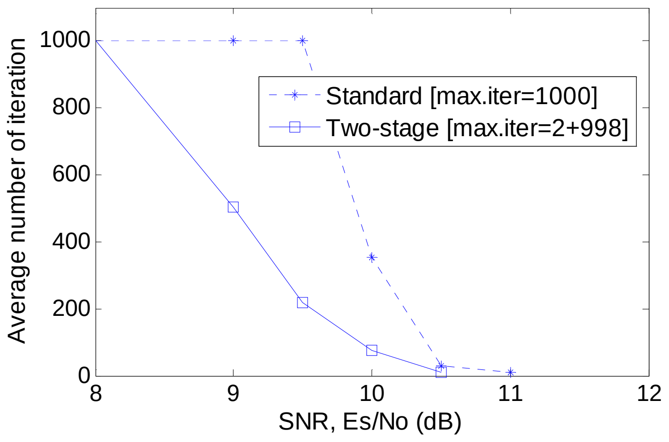
(c) BER performance (100 iterations)



(d) Iteration (100 iterations)



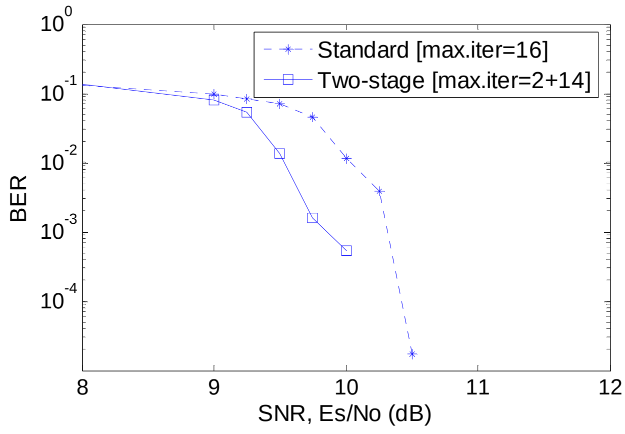
(e) BER performance (1000 iterations)



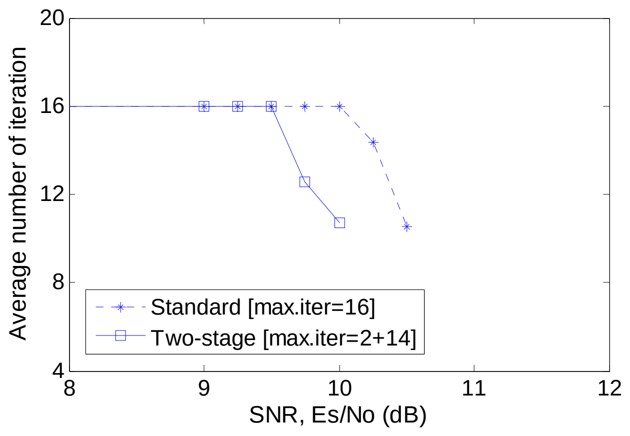
(f) Iteration (1000 iterations)

Figure 5.11 Impact of feedback decoding in the 16QAM system, irregular codes with code length 600 bits on BER performance and average number of iterations

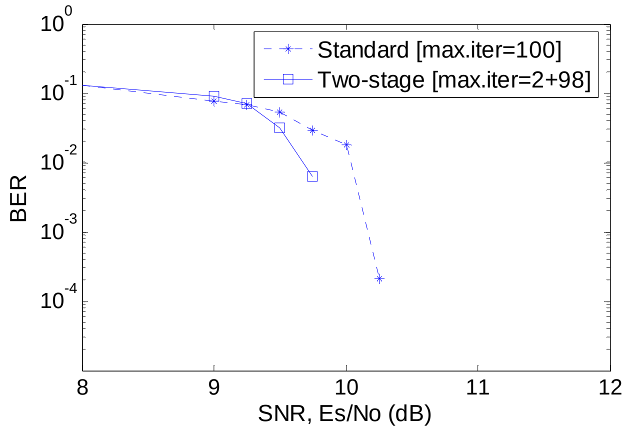
The impact of the feedback decoding on the reduction of decoding iterations for LDPC codes with higher code length is investigated. With irregular MacKay-Neal LDPC code with code length of 2,400 bits, it is shown in Figure 5.12 that the feedback decoding is effective in reducing decoding iterations without BER performance losses.



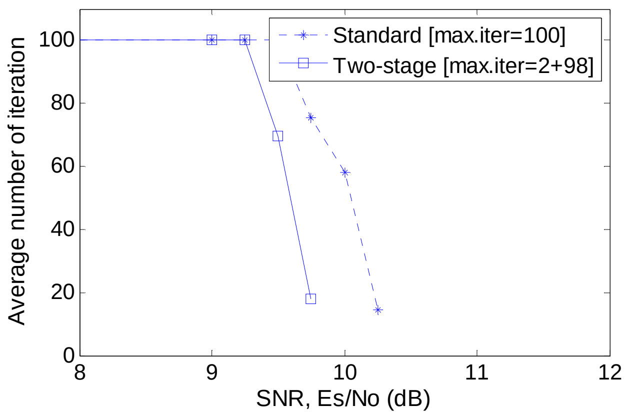
(a) BER performance (16 iterations)



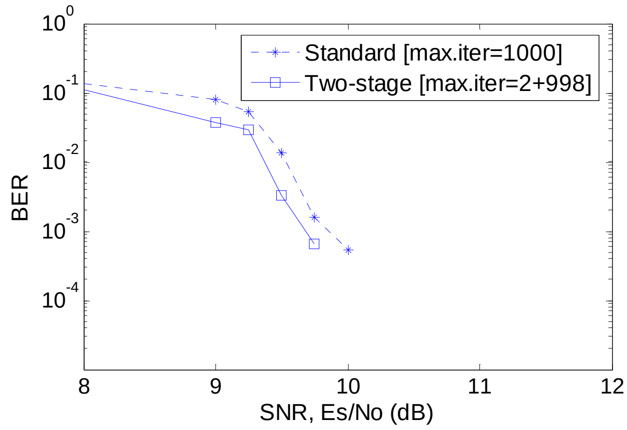
(b) Iteration (16 iterations)



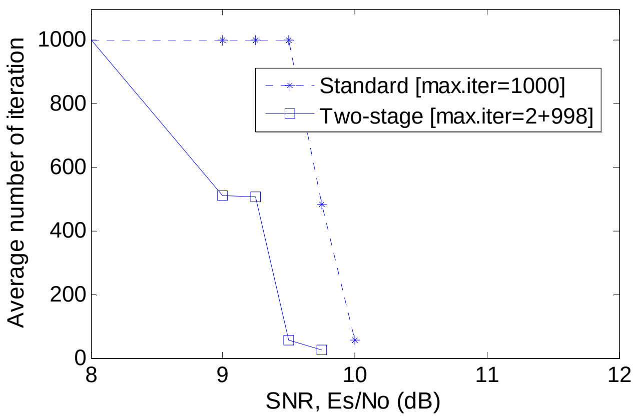
(c) BER performance (100 iterations)



(d) Iteration (100 iterations)



(e) BER performance (1000 iterations)

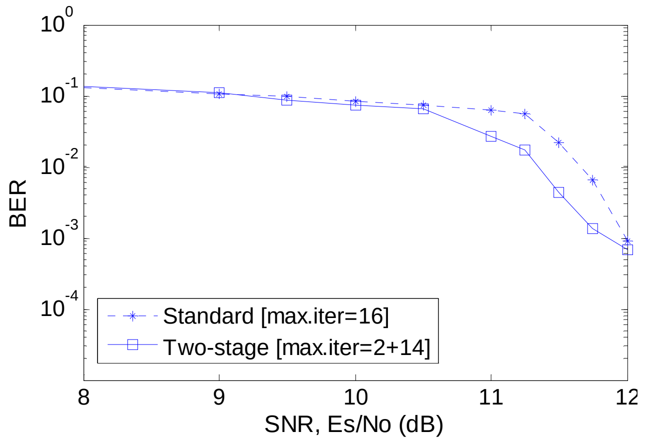


(e) Iteration (1000 iterations)

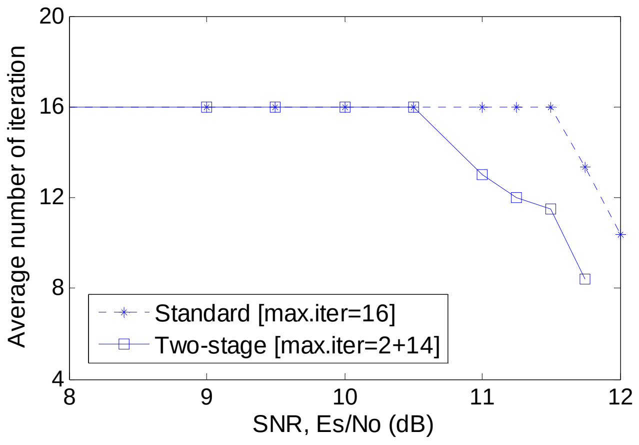
Figure 5.12 impact of feedback decoding in the 16QAM system, irregular codes with code length 2,400 bits on BER performance and average number of iterations

Structured Codes

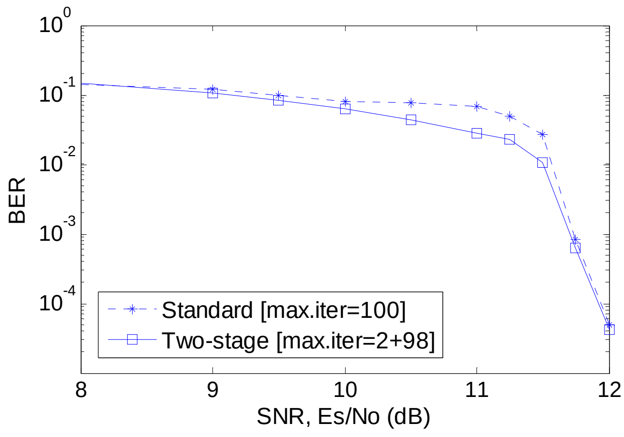
The impact of two-stage decoding on the reduction of decoding iterations in case of structured codes is also investigated. As test code, an array LDPC code [Fan00, EO01] with code length of 1,200 bits and a column weight 6 is considered. The two-stage decoding shows its impact in reducing the number of decoding iterations for different maximal numbers of iterations as shown in Figure 5.13.



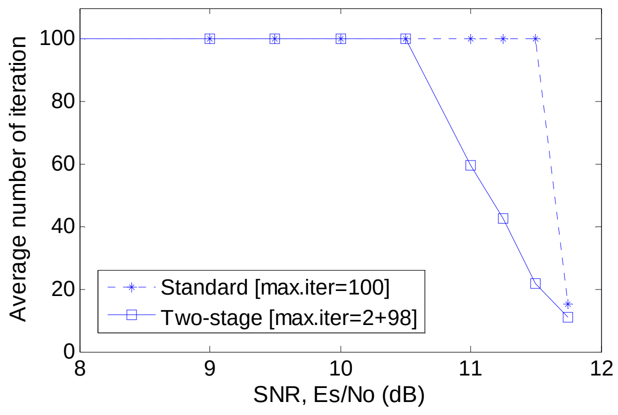
(a) BER performance (16 iterations)



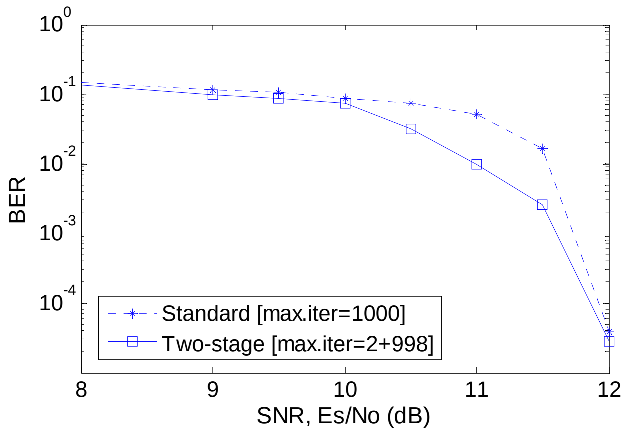
(b) Iteration (16 iterations)



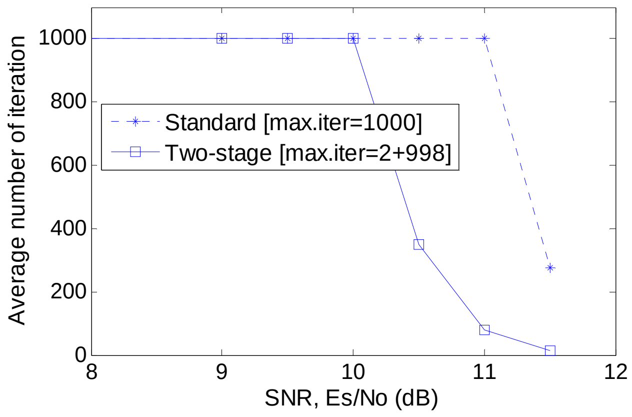
(c) BER performance (100 iterations)



(d) Iteration (100 iterations)



(e) BER performance (1000 iterations)

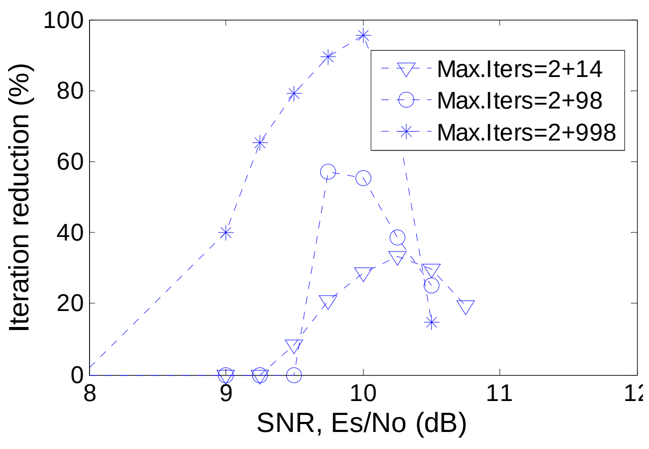


(f) Iteration (1000 iterations)

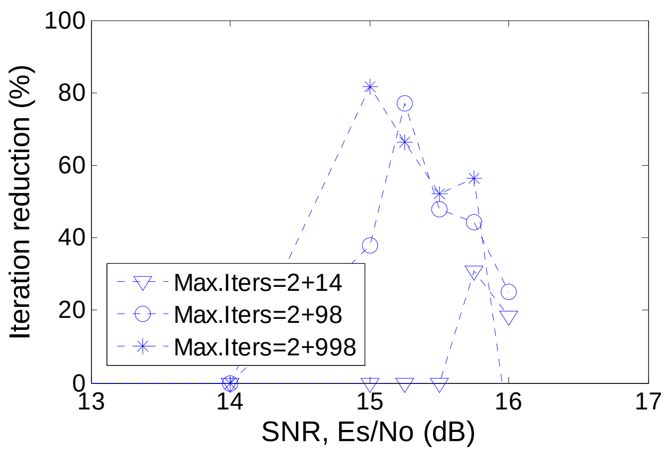
Figure 5.13 Impact of feedback decoding in the 16QAM system, array codes with code length 1,200 bits on BER performance and average number of iterations

Maximal Number of Iteration

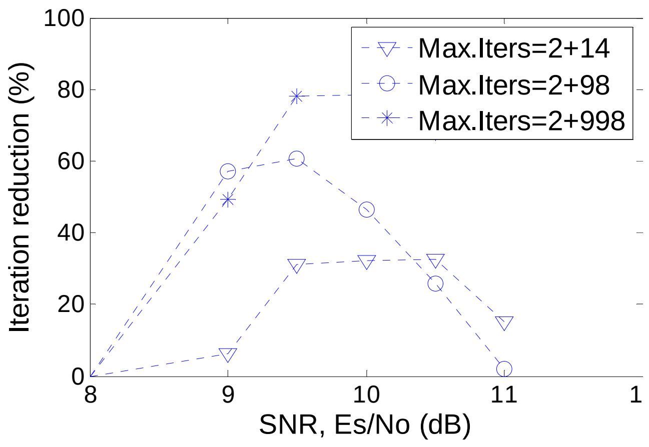
As the objective of the two-stage decoding is to reduce the number of decoding iterations, the impact of the selection of the maximal iteration numbers on the performance of the two-stage decoding becomes important. In the simulation, three values of the maximal iteration numbers of the two-stage decoding are given, i.e. 2 and 14, 2 and 98, and 2 and 998. Related to the objective, their resulting average iteration number is put in comparison with that resulting from the standard decoding with the maximal iteration numbers 16, 100 and 1000, respectively.



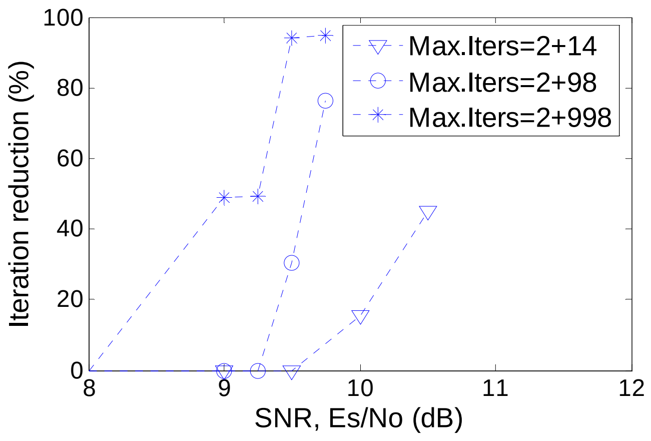
(a) MacKay code, 16QAM, 1200 bit



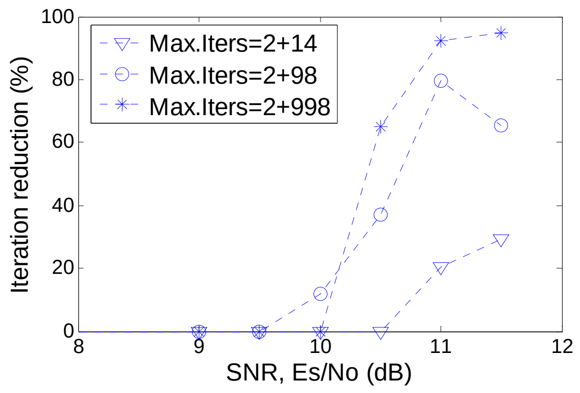
(b) MacKay code, 64QAM, 1200 bits



(c) MacKay code, 16QAM, 600 bits



(d) MacKay code, 16QAM, 2400 bits



(e) Array code, 16-QAM, 1200 bits

Figure 5.14 Impact of the two-stage decoding on the reduction of iteration numbers for different condition

As shown in Figures 5.14, in general, the reduction of average iterations occurs at SNRs within the water-fall region. Greater reduction of the average iterations is achieved at greater maximal iteration numbers. The peak of iteration reduction is obviously achieved around the middle of the water-fall region. This is not shown clearly yet in case of relative longer code lengths due to long simulation time (see Fig. 5.14.d, refer to Fig. 5.12). Furthermore, SNR values at the lower and higher part of water-fall region become critical because at those regions the improvement in decoding is not obvious yet due to relative constant development in LLR values (refer to Fig.5.5). This is the reason that in some cases in Fig. 5.14 there are cross-sections between the curve of iteration reduction within the lower and higher part of the water-fall region due to still unobvious efficiency of the proposed decoding method in the reduction of average iterations within those regions.

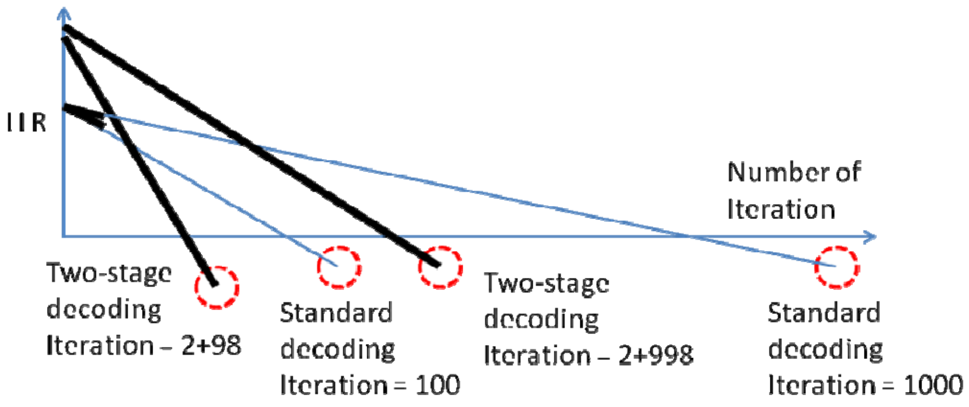

LLR
Number of Iteration
Two-stage decoding Iteration – 2+98
Standard decoding Iteration = 100
Two-stage decoding Iteration – 2+998
Standard decoding Iteration = 1000

Figure 5.15 Model of LLR development during iterations for low and high numbers of decoding iterations

The difference of the impact of the two-stage decoding depending on the maximal number of iterations can be illustrated in Figure 5.15. The larger iteration reduction is gained by the two-stage decoding at the higher number of decoding iterations because it benefits from ineffective longer iterations.

5.3.4. General Observation

According to simulation results, the two-stage decoding is effective to reduce the number of iterations necessary to decode the codeword in the water-fall region. In the region before water-fall region (lower SNR), the number of iterations in the two-stage decoding approaches the maximal number of iterations as well as that of normal decoding. It means that the feedback decoding brings no improvement in reducing the iterations during the decoding process at bad SNRs, at which no significant change in LLR values is experienced as shown in Figure 5.16. Meanwhile, in regions after the water-fall region (higher SNR), the number of iterations in the two-stage decoding is approximately similar to that of the standard decoding approaching the very small number of iterations. It means the change of the LLR value, which is excited by the two-stage decoding, gives only a subtle impact on the greater change of the LLR value due to the high SNR.

Some results show that the feedback decoding gives also subtle improvements in BER performance in the waterfall region. Such improvements are also given by [Zhe06] that proposed the time-delay feedback control method for his LDPC decoding, which controls signals formed from the difference between the current state and the state of the system delayed by some time period.

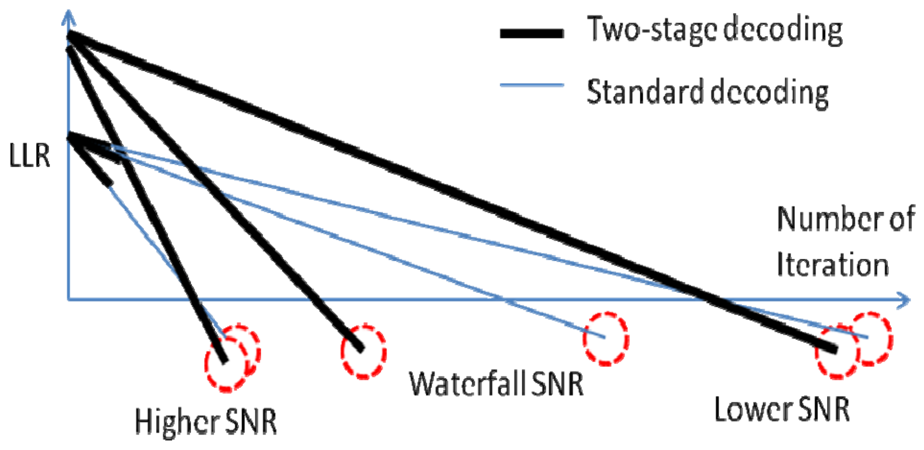


Figure 5.16 Model of the LLR development during iterations under influence of SNR

In general, the two-stage decoding is capable of improving decoding process leading to the reduction of iteration numbers. The iteration reduction means higher decoding speed, which is important for broadband applications, and also reduced decoding complexity, which saves hardware resources and cost.

Moreover, the two-stage decoding performs well at any condition of some system parameters, i.e. code type, code length, bit constellation and decoding iterations. It means that the performance of the two-stage decoding is independent of those system parameters. The only condition influencing the performance of the two-stage decoding is the SNR value, at which the system is working. As predicted in the idea, the two-stage decoding gives good impact on the iteration reduction at the SNR in the water-fall region.

5.4 Conclusion

In this chapter, the two-stage decoding for reducing the average number of iterations in case of high coded modulation has been presented. In principal, the proposed decoding scheme makes use of the output message from the first decoding stage as additional information for the second decoding stage. In order to achieve the objective, i.e. maximal iteration reduction, the weighting factor for the output message as well as the combination of the maximal number of iterations within two decoding stages must be appropriately chosen.

The simulation results show that in comparison with the standard decoding, the two-stage decoding needs less average number of iterations to achieve the comparable code performance. The level of iteration reduction becomes more significant as the maximal number of iterations is set higher. It means that in certain conditions prolonging iteration brings no potential improvement leading to rightly decoded messages.

It is of interest that the iteration reduction resulting from the two-stage decoding brings mostly no performance degradation. Even, in some cases, the proposed decoding is capable of improving code performance in the water-fall region of the SNR. This is strongly correlated with the simulation results showing the largest level of iteration reduction takes place in the water-fall region of the SNR. In the other SNR regions, no significant iteration reduction is achieved by the proposed method.

CHAPTER 6

CONCLUSION

This thesis consists of three topics. The first topic discusses theoretical aspect of LDPC codes, i.e. a novel concept for code performance measure. The other topics discuss practical aspects of LDPC codes, i.e. code construction and decoding improvement. The latter topics are strongly developed in relation with the requirement of WLAN system, while the previous one is used for analyzing the proposed codes and other codes in term of code performance.

In the first topic, the novel concept on girth degree to characterize and evaluate LDPC codes is introduced. This concept can be considered as the extension of the concept on girth. This concept adopts the concept of node degree, in which the concept of degree relies on girth. Instead of concerning edges within node degree concept, girth degree (or the number of girth cycles) is considered as the determinant component for girth in affecting the code performance. Intuitively, higher girth degree would result in higher bit dependency in decoding process. In turn, it would degrade decoding performance.

In order to support the concept of girth degree, a method for detecting girth and a method for counting the number of girth cycles are introduced. The girth-detection method is used to detect the local girth of each variable node, while the girth-degree counting method is used to count the number of girth cycles passing through each variable node resulting in girth degree. The computation involved by those methods takes longer time and runs more intensively as the code length or the column weight increase.

Some types of LDPC codes are characterized in terms of their girth degree distribution to prove the efficiency of the concept of girth degree. In case of random MacKay codes the concept of girth degree shows that as the code length increases the average girth degree decreases as well as their girth degree distribution becomes smaller and more concentrated to lower girth degree. Such a girth degree condition is also applicable as code rate decreases.

In order to investigate the correlation between code performance and girth degree, the girth condition of random MacKay codes and array LDPC codes is put in comparison. With the same column weight and code length, random MacKay codes have better average girth degree as well as girth degree distribution than those of structured array LDPC codes. This girth degree condition allows MacKay codes to outperform array LDPC codes.

The concept of girth degree is used to characterize the optional LDPC codes of WLAN. It is of interests that short-four-cycles are realized in some WLAN codes. However, the existence of those cycles does not degrade the performance of the corresponding WLAN codes. It can be considered as a proof that to some extent that the existence of short-four-cycles is not harmful enough to significantly degrade the code performance.

The simulation shows that the girth degree is more effective to be used as performance measures than the girth. The average girth degree becomes higher as code rate increases. In turn, this leads to worse code performance. This fact shows that the number of cycles of girth plays a more significant role than the girth in determination of code performance. This is quite reasonable that the girth degree indicates more precisely on how dependent the relationship between the variable nodes is. The bit dependency correlates with the number of cycles passing through the nodes.

In the second topic, a class of LDPC codes with a very efficient encoder is introduced as proposed solution for LDPC codes' drawback in encoding without degradation in code performance. The proposed codes make use of stair structure in form of an identity matrix in combination with permutation matrices, whose slope parameters are generated by a random permutation process. The identity matrix allows the encoding process to be implemented with low complexity. The resulting encoder is relative simply to realize. The permutation matrices are constructed by two proposed methods, i.e. the slope method and the girth degree method, which proves the possible existence of short-four cycles.

The construction methods introduce two structure types of the proposed codes, i.e. cascade and lattice structure. Their comparison shows that the lattice structure can support any code rate, while the cascade structure can only cope with limited code rate. However, the lattice structure requires more shift registers and additional operations at the same code rate as the cascade structure does. The performance of both codes is comparable.

In designing the codes, it is relative easier to produce the codes without short-four-cycles at lower code rate, i.e. 1/2 and 2/3. While, at higher code rate 3/4 and 5/6 the existence of short-four-cycles could not be avoided. Although short-four-cycles exist, the codes do not suffer from performance degradation. The performance of the codes is comparable with random MacKay codes. Even, at short code lengths and high code rates, the codes are able to outperform the difference families codes and some WLAN LDPC codes. This proves that the number of short-four-cycles in the codes can be considered as still harmless for decoding performance. In addition, at higher code rates, the codes can achieve good error floor performance at BER 10^{-9}.

At lower code rates of 1/2 and 2/3 the codes shows good threshold performance in the water-fall region, but are suffering from high error floor at high SNR regions. This poor floor performance is correlated with the existence of degree-1 variable (parity) nodes of the identity matrix in the stair structure. These variables do not contribute much to update of message passing during the iterative decoding process. In addition, the probability that the trapping sets considered as source of poor error-floor performance is higher as the number of variable (parity) node increases or correspondingly, the code rate decreases. To some extent, the poor performance of degree-1 variables can be compensated by appropriately increasing the column weight of permutation submatrices to suppress the error floor at cost of higher decoding complexity.

In comparison with the optional WLAN LDPC codes, the performance of the codes is relative poorer at lower code rates of 1/2 and 2/3. But, the codes show comparable performance at higher code rates of 3/4 and 5/6. At short code length of 648 bits with high code rate 5/6, the codes are capable of outperforming the optional WLAN codes. In spite of poorer performance in general, the encoding complexity of the codes is relative lower than that of the optional WLAN codes. The codes can produce the parity bits of the encoded message directly from the parity-check matrix. The optional WLAN codes do a substitution process to produce their parity bits, what leads to relative long encoding latency time. However, in decoding complexity, the codes require relatively more intensive computation due to their higher number of non-zero elements in their parity-check matrix.

In general, in addition to their relative simpler code construction and lower encoding complexity, the codes show their strength at higher code rate. These could be a reason to opt them for any WLAN application requiring high-rate/short-length transmission.

In the third topic, a feedback mechanism in a two-stage decoding process that supports the enhancement of decoding performance is presented. The two-stage decoding makes use of soft output from the first decoding stage to improve bit reliability for the decoding process in the second decoding stage. The appropriate choice of a maximal number of decoding iteration within two decoding stages will minimize average number of decoding iterations for the whole decoding process. Moreover, the weighting factor for the feedback mechanism should be chosen appropriately so that comparable BER performance gain can be achieved by less decoding iterations.

The proposed two-stage decoding shows its impact in reducing decoding iterations in the waterfall region. The magnitude of reduction of the iterations is more significant as the maximal decoding iterations increases.

6.1 Thesis contribution

This thesis gives the following contribution

- a concept of girth degree, which proves the existence of harmless short-four-cycles in some WLAN codes. This concept also proves that the girth degree plays a more significant role than the girth in determining the code performance.
- a class of LDPC Stair codes, which allows very efficient encoding.
- a two-stage decoding scheme, which is able to achieve comparable BER performance with less decoding iterations

6.2 Future Work

Some work could be proposed to do in the future in order to extend the works presented in this thesis. Those works could be done to strengthen the theoretical basis of the proposed method as well as to prove the generality of the applications of the proposed methods.

In the first topic, as proved widely in the simulation results, the existence of short-four-cycles is not harmful enough to cause degradation in code performance. Some publications as well as some WLAN codes analyzed become the proofs. However, to some extent, the number of short-four-cycles could be considered as significantly harmful for the code performance and therefore, this must be taken into account.

To best knowledge, there is still no method that can exactly figure out on the quantity of short-four-cycles that begins to give negative impact on the code performance. The proposed concept of girth degree could be potentially applicable for exactly determining the number of girth cycles as well as its distribution on the variable nodes that could lead to the performance degradation.

In addition to that, the concept of girth degree could be used to characterize LDPC codes as many as possible in terms of their girth condition in order to get more understanding on the behaviour of LDPC codes. Furthermore, it is also interesting to investigate the possibility of the concept of girth degree as the basis for searching the codes with good performance.

In the second topic, the optimization effort could be done to produce the codes with better code performance. So far, in code construction, the permutation for determining slope parameter of parity check matrices is randomly generated. The generated slope parameter performing best are then taken into account. In the future, it is quite challenging to find systematically slope parameters leading to best performance. This systematic rule of determining the slope parameter could be established on the basis of some rules for code performance optimization. In addition to that, the floor performance of the codes with high code rates is of interest for research topics in searching for channel codes dealing with high-rate application.

In the third topic, so far, the proposed two-stage decoding is able to reduce the average number of decoding iterations without (significant) code performance degradation. The development of the concept is established using a practical based approach. In the future, it is of interests to investigate the decoding process in the second stage of decoding on more theoretical basis, especially as decoding process takes place in the water-fall region, where the greater iteration reduction is mostly achieved. In addition to that, to prove its efficiency in more general sense, the proposed two-stage decoding could be tested upon other varieties of LDPC.

ABBREVIATIONS

Acronym

ARIB	Association of Radio Industries and Businesses
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hochquenghem
BER	Bit Error Rate
BISC	Binary Input Symmetric Channel
BPSK	Binary Phase-Shift Keying
BRAN	Broadband Radio Access Networks
CDMA	Code Division Multiple Access
ERC	Engineering Research Centers
ETSI	European Telecommunications Standards Institute
FEC	Forward Error Correction
H-ARQ	Hybrid Automatic Repeat-reQuest
HIPERLAN	HIgh PErformance Radio LA
HiSWANa	High Speed Wireless Access Network type a
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
ISM	Industrial, Scientific, and Medical
LDPC	Low Density Parity Check
LDGM	Low Density Generator Matrix
LLR.	Log-Likelihood Ratio
LOS	Line Of Sight
MAP	Maximum A Posteriori
Mbps	Mega bit per second
MIMO	Multiple-Input Multiple-Output
ML	Maximum Likelihood
MMAC	Multimedia Mobile Access Communication
NLOS	Non-Line-Of-Sight
OFDM	Orthogonal Frequency-Division Multiplexing
OFDMA	Orthogonal Frequency-Division Multiple Access
PDU	Protocol Data Unit
pdf	probability density function
PHY	PHYsical layer of OSI model.
PPDU	Presentation Protocol Data Unit
QAM	Quadrature Amplitude Modulation
QoS	Quality of Service
QPSK	Quadrature Phase-Shift Keying
RS	Reed and Solomon

U-NII	Unlicensed National Information Infrastructure
WLAN	Wireless Local Area Network

Notation

A	signal set
A^n	set of n-length strings over alphabet A
a	a complex symbol
$\mathbf{A}$	submatrix
$\mathbf{B}$	submatrix
$\mathbf{b}$	information vector
C	linear code
$\mathbf{c}$	codeword vector
c_i	i-th codeword bit
E_b	Energy per bit
e	connecting edge
F	field
$\mathbf{G}$	generator matrix
g	dimension of an optimized submatrix
gd	girth degree
$\mathbf{H}_i$	circulant permutation matrix i
$\mathbf{H}$	parity check matrix
$\mathbf{H}_\mathrm{S}$	parity check matrix in a systematic form
$\mathbf{I}$	identity matrix
k	dimension of the code
$\mathbf{L}_{i\text{-}n}$	vector of log-likelihood ratio of input for n-th decoding
$\mathbf{L}_o$	vector of log-likelihood ratio of output
$L(c_i)$	log-likelihood ration of bits c_i
m	redundancy
N_o	noise power spectral density
N	gaussian random variable
n	block code length, total number of variable node or code length.
n_{gd}	number of variable node with girth degree gd
$\mathbf{p}$	parity vector
p	crossover probability of the channel
$p\times p$	submatrix dimension
$\Pr(X\|Y)$	a posteriori probabilities
$p_{\mathbf{Y}\|\mathbf{X}}(\mathbf{y}\|\mathbf{x})$	conditional probability density function
$p_{\mathbf{X}}(\mathbf{x})$	probability density function (pdf)
P^T	transposed parity check matrix
Q	permutation matrices
q	message sent from the variable node to the check node along a connecting edge

r	message sent from the check node to the variable node along a connecting edge
$S_i^{(ci)}$	symbols with c_i in the signal constellation
s	cyclic-shift number
Δs_{ij}	slope distances
t	number of symbol bits
$\mathbf{U}$	input string of encoder
$\hat{\mathbf{U}}$	output string of encoder
$\mathbf{u}$	information vector
$\mathbf{u}_i$	i-th information sub-vector
w	signal point
w_i	weighting factor for LLR input
w_o	weighting factor for LLR output
X	input random variables
$\mathbf{X}$	input string of channel
$\mathbf{x}$	input vector of channel
x	value of the bit node
x_i	input of channel at that time instant i
$\hat{x}$	output of the maximum likelihood decoder
Y	output random variables
$\mathbf{Y}$	output string of channel
$\mathbf{y}$	output vector of channel
$\boldsymbol{y}$	received message vector coming from the channel and the edges connected to the bit node, other than edge e.
y_i	output of channel at that time instant i
μ	mean
σ^2	variance
φ	fraction of variable node
Φ_{gd}	average girth degree
$\gamma(i)$	distributions of row weights of row i-th
$\rho(i)$	distributions of column weights of column i-th

REFERENCES

[Burs02] D. Burshtein. Bounds on the performance of belief propagation decoding. *IEEE Trans. Inform. Theory*, 48:112–122, January 2002.

[BC60] B.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Inform. and Control*, 3(1):68-79, March 1960.

[BCB04] Y. Blankenship, B. Classon, and K. Blankenship. Motorola harmonized structured LDPC proposal. *Motorola submission to informal LDPC group*, 13 August 2004.

[BCJR74] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Inform. Theory*, IT-20:284-287, March 1974.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshirna. Near Shannon limit error correcting coding and decoding: Turbo codes. In *Proc. Int.l Conf. Comm.*, pages 1064-1070, Geneva, Switzerland, 1993.

[BHS00] J. Bond, S. Hui, and H. Schmidt. Linear-congruence construction of low-density check codes". In *Codes, Systems and Graphical Models*, IMA 123:83-100. Springer-Verlag, 2000.

[BKA04] S. ten Brink, G. Kramer, and A. Ashikhmin. Design of low-density parity-check codes for modulation and detection. *IEEE Trans. Comm.*, 52:670-678, April 2004.

[BKLM02] D. Burshtein, M. Krivelevich, M. Litsyn, and G. Miller. Upper bounds on the rate of LDPC codes. *IEEE Trans. Inform. Theory*, 48:2437-2449, September 2002.

[BLMR98] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distributor of bulk dat. In ACM SIGCOMM'98, August 1998

[BM96] S. Benedetto and G. Montorsi. Unveiling turbo codes: some results on parallel concatenated coding schemes. *IEEE Trans. Inform. Theory*, IT-42:409-428, March 1996

[BMDP96] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara. Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding. *JPL TDA Progress Report*, pages 42-126, August 1996

[BPZ99] J. Boutros, O. Pothier and G. Zemor. Generalized low density (Tanner) codes. In *Proc. IEEE Int. Conf. Comm.*, 1:441-445, 6-10 June, 1999.

[Chun00] S.Y. Chung. On the construction of some capacity-approaching coding schemes. Ph.D Thesis, MIT,USA, 2000.

[Chun01] S. Chung *et al.*. On the design of low density parity check codes within 0.0045 dB of the Shannon limit. *IEEE Comm. Letter.*, 5:58-60, February 2001.

[Clas04] B. Classon *et al*. Modified LDPC matrix providing improved performance, *IEEE C802.16e-04/102r1*, Motorola, May 18, 2004

[CFRU01] S.-Y. Chung, D.J. Forney Jr., T.J. Richardson, and R. L. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Trans. Inform. Theory*, 47:58-60, February 2001.

[CRU01] S.Y. Chung, T.J. Richardson, and R.L. Urbanke. Analysis of sum-product decoding of low density parity check codes using a Gaussian approximation. *IEEE Trans. Inform. Theory*, 47(2):657-670, February 2001.

[CT94] G. Caire and C. Taricco. Weight distribution and performance of the iterated product of single-parity-check codes. In *Proc. IEEE Global Telecom. Conf.*, pages 206-211, 1994.

[Dave99] M.C. Davey. *Error-correction using Low density parity check codes*. Ph.D thesis, University of Cambridge, UK, 1999.

[DJE98] D. Divsalar, H. Jin, and R. McEliece. Coding theorems for turbo-like codes. In *Proc. 36th Annual Allerton Conf. on Comm., Control, and Computing*, pages 201-210, September 1998.

[DK98] M.C. Davey and D.J.C MacKay. Low density parity check codes over GF(q). *IEEE Comm. Lett.*, 2:165–167, June 1998.

[DK99] M.C. Davey and D.J.C. MacKay. Low density parity check codes over GF(q). In *Proc. IEEE Inform. Theory Workshop*:70–71, June 1998.

[DP97] D. Divsalar and F. Pollara. Serial and hybrid concatenation codes with applications. In *Proc. Intl. Symp. on Turbo Codes an Related Topics*, pages 80-87, Brest, Fance, September 1997.

[DXGL03] I. Djurdjevic, J. Xu, and K.A-Ghaffar. A class of low density parity check codes constructed based on Reed-Solomon codes with two Inform. symbols. *IEEE Comm. Letter.*, 7(7), July 2003

[Elia54] P. Elias. Error-free coding. *IRE Trans. Inform. Theory*, pages 29-37, September 1954

[Elia55] P. Elias. Coding for noisy channels. *IRE Int. Convention Record*, 4:37-46, 1955

[EC01] R. Echard and S.C. Chang. The π-rotation low-density parity-check codes. In *Proc. IEEE Global Telecom. Conf.*, pages 980-984, Nov. 2001

[EKC98] R.J. McEliece, D.J.C. MacKay, and J.F. Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *IEEE Journ. Select. Areas Comm..*, 16:140-152, February 1998.

[EKP03] A.W. Eckford, F.R. Kschischang, and S. Pasupathy. Designing very good LDPC codes for the Gilbert-Elliott channel. In *Proc. 8th Canadian Workshop Inform. Theory*, Waterloo, Ontario, Canada, 2003

[EMC98] R. J. McEliece, D. J. C. MacKay, and J.-F. Chen. Turbo decoding as an instance of Pearl's belief propagation algorithm. In *IEEE Journ Select. Areas in Comm.*, 16(2): 140-152, February 1998

[EMD01] E. Eleftheriou, T. Mittelholzer, and A. Dholakia. Reduced-complexity decoding algorithm for low-density parity-check codes. *Electronics Letter*, vol. 37, pp. 102-104, Jan. 2001.

[EÖ01] E. Eleftheriou and S. Olcer. Further results on the performance of LDPC coded modulation for AWGN channels. In *ITU-Telecomm. Standard. Sector*, May 2001.

[ETSI99] ETSI. Broadband Radio Access Networks (BRAN): HIPERLAN type 2 technical specification: Physical (PHY) layer. August 1999.

[ETV99] T. Etzion, A. Trachtenberg, and A. Vardy. Which codes have cycle-free Tanner graphs. *IEEE Tans. Inform. Theory*, 45(6):2173–2181, September 1999.

[Fan00] J. Fan. Array codes as low-density parity-check codes. In *Proc. Int. Symp. on Turbo Codes and Related Topics*, pages 543-546, Brest, France, September, 2000.

[Forn66] G.D. Forney. *Concatenated codes*. MIT Press, Cambridge, MA, 1966.

[FK02] J. Feldman and D. R. Karger. Decoding turbo-like codes via linear programming. In *Proc. Annual IEEE Symp Foundations of Computer Science*, pages 251-260, November 2002.

[FL95] M.P.C. Fossorier and S. Lin. Soft-decision decoding of linear block codes based on ordered statistics. IEEE Trans. Inform. Theory, vol. 41, pp. 1379-1396, Sept. 1995

[FO01] H. Futaki and T. Ohtsuki. Low-density parity-check (LDPC) coded OFDM systems. In *Proc. IEEE 54th Vehicular Tech. Conf.*, pages 82-86, Atlanta City, New Jersey, USA, October 2001.

[FWK05] J. Feldman, M.J. Wainwright, and D.R. Karger. Using linear programming to decode binary linear codes. *IEEE Trans. Inform. Theory*, vol. 51, pp.954-972, March 2005

[FZ99] A.J. Felstrom and K.S. Zigangirov. Time-varying periodic convolutional codes with low-density parity-check matrix. *IEEE Trans. Inform. Theory*, 45:2181-2191, September 1999.

[Gall62] R. G. Gallager. Low-density parity check codes. *IRE Trans. Inform. Theory*, IT-8:21-28, January 1962.

[Gall63] R. G. Gallager, Low-density parity check codes. Ph.D Thesis, MIT Press, Cambridge, USA, 1963.

[GP05] G. Glikiotis and V. Paliouras, “A Low-Power Termination Criterion for Iterative LDPC Code Decoder,” IEEE Workshop on Signal Processing Systems Design and Implementation, pp. 122-127, Nov. 2005.

[Hamm50] R.W. Hamming. Error detecting and correction codes. *Bell Sys. Tech. Journ.*, 29:147-160, 1950.

[Hocq59] A. Hocquenhem. Codes correcteurs d’erreurs. *Chiffres*, 2:147-156, 1959.

[Hou01] J. Hou, P. H. Siegel, L. B. Milstein, and H. D. Pfister, “Multilevel coding with low-density parity-check component codes,” in *Proc. IEEE Global Telecommunications Conf.*, San Antonio, TX, Nov. 2001, pp. 1016–1020.

[HB01] C. Howland and A. Blanksby. A 220mW 1Gb/s 1024-bit rate-1/2 low density parity check code decoder. In *Proc. IEEE Custom. Integrated Circuits Conf.* 293-296, 2001

[HEA01] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold. Progressive edge-growth Tanner graphs. In *Proc. IEEE Global Telecom. Conf.*, pages 995-1001, San Antonio, Texas, USA, November 2001.

[HEK07a] D. Hamdani, W. Endemann, and R.Kays. *Measuring Performance of LDPC Codes with Girth Degree*. International Conference on Electrical Engineering and Informatics (ICEEI2007), 17-19 June 2007, Bandung, Indonesia.

[HEK07b] D. Hamdani, W. Endemann, and R.Kays. *A Class of LDPC Codes with Very Efficient Encoder*. International Conference on Electrical Engineering and Informatics (ICEEI2007), 17-19 June 2007, Bandung, Indonesia.

[HEK07c] D. Hamdani, W. Endemann, and R.Kays. *Enhancing Performance of High-Order Modulation with LDPC Codes using Feedback Mechanism*. International Conference on Electrical Engineering and Informatics (ICEEI2007), 17-19 June 2007, Bandung, Indonesia.

[HGB02] D. Haley, A. Grant, and J. Buetefuer. Iterative encoding of low-density parity-check codes. In *Proc. IEEE Global Telecomm. Conf.,* Taipei, Taiwan, November 2002.

[HGS04] S.L. Howard, V.C. Gaudet, and C.S Schlegel. Soft Bit Decoding of Low Density Parity Check. Codes”, submitted to *Comm. Letters*, May 2004.

[HH89] J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. *IEEE Trans. Inform. Theory*, IT-13:260-269, April 1989.

[HH02a] S. Hirst and B. Honary. Decoding of generalized low-density parity-check codes using weighted bit-flip voting. *IEE Proc. Comm.*, 149:1-5, 2002.

[HH02b] S. Hirst and B. Honary. Application of effcient Chase algorithm in decoding of generalized low-density parity-check codes. *IEEE Comm. Letters.*, 6:385-387, 2002.

[HSM01] J. Hou, P.H. Siegel and L.B. Milstein. Performance analysis and code optimization of low density parity-check codes on Rayleigh fading channels. *IEEE Journ. Select. Areas in Comm.*, 19:924–934, May 2001.

[HSMP03] J. Hou, P. H. Siegel, L. B. Milstein and H. D. Pfister. Capacity-approaching bandwidth-efficient coded modulation schemes based on low-density parity-check codes. *IEEE Trans. Inform. Theory*, 49:2141–2155, September 2003.

[IEEE05] IEEE 802.16 Broadband Wireless Access Working Group. LDPC coding for OFDMA PHY. *IEEE C802.16e-05/006r2*, January 2005

[IEEE09] IEEE Std 802.11n. *IEEE Standard for Information technology-Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: Wireless LAN Medium Access Control and Physical Layer (PHY) Specification*. 29 October 2009

[IEEE99] IEEE Std 802.11a/D7.0-1999. Part11:Wireless LAN MAC and PHY Specifications: High Speed Physical Layer in the GHz Band."

[IK09] M.R. Islam and J. Kim. *Linear encoding of LDPC codes using approximate lower triangulation with postprocessing*. In the Proceeding of PIMRC 2000, 13-16 Sept 2009. Tokyo

[Jush99] J. Khun-Jush *et al*.. Structure and Performance of HIPERLAN/2 Physical Layer. *IEEE Vehicle Tech. Conf.*, pages 2667–2671, 1999.

[JKM00] H. Jin, A. Khandekar, and R. McEliece. Irregular repeat-accumulate codes. In *Proc. Int. Symp. on Turbo Codes and Related Topics*, pages 1-8, Brest, France, September 2000.

[JW01] S.J. Johnson, and S.R. Weller. Regular low density parity check codes from combinatorial designs. In *Proc. Inform. Theory Workshop*, pages 90-92, Cairns, 2001.

[JW02] S.J. Johnson and S.R. Weller. *Quasi-cyclic LDPC codes from difference families*. In Proceedings 3rd AusCTW, Canberra, Australia.

[JW03] S.J. Johnson and S.R. Weller. *High-rate LDPC codes from unital designs*. In Proceedings of the IEEE Globecom Conference, San Francisco, CA, 1-5 December 2003

[Kay99] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45(2):399-431, March 1999.

[Kim04] J. Kim *et al*. Samsung's harmonized structured LDPC proposal. *Samsung submission to informal LDPC group*, 17 August 2004.

[KD00] D.J.C. MacKay and M.C. Davey. Evaluation of Gallager codes for short block length and high rate applications. In *Codes, Systems and Graphical Models*, IMA 123, pages 113-130. Springer-Verlag, 2000.

[KFL98] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, July 1998.

[KLF00] Y. Kou, S. Lin and M. Fossorier. Low density parity check codes: Construction based on finite geometries. In *IEEE Global Telecomm Conf.*, 2:825–829, November-December 2000.

[KLF01] Y. Kou, S. Lin and M. Fossorier. Low-density parity-check codes based on finite geometries: A rediscovery and new results. *IEEE Trans. Inform. Theory*, 47:2711–2736, November 2001.

[KN95] D.J.C. MacKay and R.M. Neal. Good codes based on very sparse matrices",. in *Cryptography and Coding: Proceeding of 5th IMA Conf., Lecture Notes in Computer Science 1025*:100-111 Springer-Verlag, Berlin, 1995.

[KN97] D.J.C MacKay, and R.M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 33:457–458, March 1997.

[KN99] D.J.C. MacKay and R.M. Neal. Good error-correction codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45:399–431, March 1999.

[KSM02] S. Kim, G. Sobelman and J. Moon. Parallel VLSI Architectures for a Class of LDPC codes. *IEEE Int. Symp. on Circuits and Systems*, Arizona, May 2002,

[KWD99] D.J.C. MacKay, S.T. Wilson, and M.C. Davey. Comparison of constructions of irregular Gallager codes. *IEEE Trans. Comm.*, 47(10):1449 – 1454, October 1999.

[Lent97] M. Lentmaier. Soft iterative decoding of generalized low density parity check codes based on MAP decoding of component Hamming codes. *Diploma Thesis*, University of Ulm, Germany, 1997.

[Li02]	J. Li. Low-complexity, capacity-approaching coding schemes: Design, analysis and application. *PhD Thesis*. Texas A&M University. 2002
[Lim03]	P. Limpaphayom, Multilevel coding with LDPC component codes for power and bandwidth efficiency. *Ph.D Thesis*, Univ. Michigan, Ann Arbor, 2003
[Luby02]	M. Luby. LT codes. In *43rd IEEE Symposium on Foundations in Computer Science*, November 2002
[LC04]	S. Lin and D.J. Costello, Jr. *Error Control Coding*. Prentice Hall. 2004
[LK08]	Chia-Yu Lin and Mong-Kai Ku, Early Detection of successful decoding for dual-diagonal block-based LDPC codes, *Electronics Letters*, 6 November 2008, Vol 44, Issue 23, p. 1368-1370
[LMSS98]	M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman. Improved low density parity check codes using irregular graphs and belief propagations. In *Proc. IEEE Int. Symp. Inform. Theory*, page 117, 1998.
[LMSSS97]	M.G. Luby, M.Mitzenmacher, M.A. Shokrollahi, D.A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. 29th Symp. Theory Computing*, pages 150-159, 1997.
[LNG01a]	J. Li, K.R. Narayanan, and C.N. Georghiades. A class of linear-complexity, soft-decodable, high-rate, good codes: Construction, properties and performance. In *Proc. Intl. Symp. Inform. Theory*, page 122, Washington DC, June 2001.
[LNG01b]	J. Li, E. Narayanan and C. N. Georghiades. On the performance of turbo product codes and LDPC codes over partial-response channels. In *IEEE Int. Conf. Comm.*, 7:2176-2183, June 2001.
[LRG04]	J. Li, K. R. Narayanan, and C. N. Georghiades. Product accumulate codes: A class of capacity-approaching, low complexity codes. *IEEE Transaction on Inform. Theory*, January 2004.
[LSMS01]	M.G. Luby, M.A. Shokrolloahi, M. Mizenmacher, and D.A. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Trans. Inform. Theory*, 47:585-598, February 2001.
[LWN02]	B. Lu, X.Wang and K.R. Narayanan. LDPC-based space-time coded OFDM systems over correlated fading channels: Performance analysis and receiver design. *IEEE Trans. Comm.*, 50:74-88, January 2002.

[LZW04] J.R. Lin, R.Q. Zhang and W.L. Wu. Performance of irregular LDPC Codes on Rician fading channels. *In Proc. 2nd Int. Conf. on Inform. Tech. and Applications*, Harbin, China, January 2004

[Marg82] G.A. Margulis. Explicit constructions of graphs without short cycles and low density codes. Combinatorica, 2(1):71-78, 1982.

[MaB01] Y. Mao and A.H. Banihashemi. A heuristic search for good low-density parity-check codes at short block lengths. In *Proc*. *IEEE Int. Conf. Comm.*, Helsinki, Finland, June 2001.

[MiB01] G. Miller and D. Burshtein. Bounds on the maximum-likelihood decoding error probability of low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47:2696-2710, November 2001

[MF02] N. Miladinovic and M.P.C. Fossorier. Improved bit flipping decoding of low-density parity check codes. In *Proc. IEEE Int. Symp. On Inform. Theory*, p. 229, July 2002.

[MMØ02] B. Myhre, V. Markhus and G.E. Øien. LPDC-coded adaptive modulation on slowly varying Nakagami-fading channels. In *Proc. European Wireless*, pages 822-828, Florenze, Italy, February 2002.

[MN95] D. J. C. MacKay and R. M. Neal, C. Boyd. Good codes based on very sparse matrices. *Cryptography and Coding 5th IMA Conf.*, no. 1025, pp.100 - 111, Springer, 1995.

[MS98] J. Medbo and P. Schramm. Channel Models for HIPERLAN/2. *ETSI/BRAN doc. no. 3ERI085B*, 1998.

[MS03] M.M. Mansour and N.R. Shanbhag. High-throughput LDPC decoders. *IEEE Trans. on Very Large Scale Integration Systems*, 11(6):976-996, December 2003

[Neal99] R.M. Neal. Faster encoding for low-density parity check codes using sparse matrix methods. *IMA Workshop on Codes, Systems and Graphical Models*, Minneapolis, August 1999

[Nee99] R. Van Nee *et al*. New High-Rate Wireless LAN Standards. *IEEE Comm. Mag*., 37(12):82–88, December 1999.

[NF04] H.P-Nik and F. Fekri. On Decoding of Low-Density Parity-Check Codes Over the Binary Erasure Channel. *IEEE Trans. Inform. Theory*, 50:439-454, March 2004.

[NKS01] K. Nakamura, Y. Kabashima and D. Saad. Statistical mechanics of low-density parity check error-correcting codes over Galois fields. *Europhysics Letters*, 56:610–616, November 2001.

[OM01] T. Oenning and J. Moon. A Low Density Generator Matrix Interpretation of Parallel Concatenated Single Bit Parity Codes. *IEEE Trans. Magnetics*, 37:737-741, March 2001.

[Pear88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. :Morgan Kauffmann, San Mateo, 1988.

[PBB99] O. Pothier, L. Brunel and J. Boutros. A low complexity FEC scheme based on the intersection of interleaved block codes. In *Proc. IEEE 49th Vehicular Tech. Conf.*, 1:274-278, Houston, USA, May 1999.

[PLP99] P. Li, W.K. Leung, and N. Phamdo. Low density parity check codes with semi-random parity check matrix. *Electronics Letters*, 35:38-39, January 1999

[PSJ04] N. Burns, A. Purkovic, S. Sukobok, B. Johnson. Algebraic low-density parity check codes for OFDMA PHY layer. *Nortel submission to informal LDPC group*, 13 August 2004.

[RMC03] M. Surendra Raju, A. Ramesh, and A. Chockalingam. BER Analysis of QAM with Transmit Diversity in Rayleigh Fading Channels. In *Proc. IEEE Global Telecom. Conf.,* pages 641-645, San Francisco, November-December 2003.

[RN04] V. Roca and C. Neumann. Design, Evaluation and Comparison of Four Large Blcok FEC Codecs, LDPC, LDGM, LDGM Staircase and LDGM Triangle, plus a Reed-Solomon Small Block FEC Codes. Rapport de recherche n° 5225. INRIA Rhone-Alpes, France. 9 June 2004.

[RNF06] V. Roca, C. Neumann and D. Furodet. Low density parity check staircase and triangle forward error correction schemes", Internet-Draft, 18 July 2006

[RS60] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM Journ. Applied Math.*, 8:300-304, 1960

[RSU00] T. J. Richardson, A. Shokrollahi, and R. Urbanke. Design of provably good low-density parity-check codes. In *Proc. Int. Symp. Inform. Theory*, page 199, Sorrento, Italy, 2000.

[RSU01] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. In *Proc. IEEE Trans. Inform. Theory*, 47:619-637, February 2001.

[RU01a] T. Richardson, R. Urbanke. The capacity of low density parity check codes under message-passing decoding. *IEEE Trans. Inform. Theory*, 47:599–618, February 2001.

[RU01b] T. J. Richardson and R. L. Urbanke. Efficient encoding of low density parity check codes. *IEEE Trans. Inform. Theory*, 47:638-656, Febuary 2001.

[RU03] T. Richardson and R. Urbanke. The renaissance of Gallager's low density parity check codes. *IEEE Comm. Magazine*, pages 126-131, August 2003.

[RV00] J. Rosenthal and P.O. Vontobel. Construction of LDPC codes using Ramanujan graphs and ideas from Margulis. In *Proc. of 38th Allerton Conf. on Comm., Control, and Computing*, pages 248-257, October 2000.

[Shan48] C. E. Shannon. A mathematical theory of communication. *The Bell system Tech. Journ.*, 27:379–656, July 1948.

[Shok03] A. Shokrollahi. Raptor codes. Research Report DR2003-06-001. Digital Fountain. 2003

[Spiel96] D. A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. Inform. Theory*, 42:1723–1731, November 1996.

[SC03] H. Song and J. R. Cruz. Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording. *IEEE Trans. on Magnetics*, 39:1081–1087, March 2003.

[SKP00] V. Sorokine, F.R. Kschischang, and S. Pasupathy. Gallager codes for CDMA applications - Part I: Generalizations, constructions, and performance bounds. *IEEE Trans. on Comm*. 48(10):1660-1668, October 2000.

[SM02] H. Steendam, M. Moeneclaey. ML-performance of low density parity check codes. In *Proc of 23rd Symp. on Inform. Theory in the BENELUX*, pages 75-78 Louvain-la-Neuve, Belgium, May 2002

[SNG03] A.Serener, B.Natarajan and D.Gruenbacher. LDPC coded spread OFDM in indoor environments. In *Proc. of the 3rd IEEE Int. Symp. on Turbo Codes & Related Topics*, pages 549-552, France, September 2003.

[SS96] M. Sipser and D. A. Spielman. Expander Codes. *IEEE Trans. Inform. Theory*, 42:1710–1722, November 1996.

[STC00] H. Song, R.M. Todd, J.R. Cruz. Low density parity check codes for magnetic recording channels. *IEEE Trans. Magnetics*, 36(5), September 2000.

[Tann81] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, IT-27:533-547, September 1981.

[TJVW04] T. Tian, C.R. 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, August 2004

[TXKLG04] H. Tang, J. Xu, Y. Kou, S. Lin, and K.A-Gaffar. On algebraic construction of Gallager and circulant low-density parity-check codes. *IEEE Trans. Inform. Theory*, 50(6):1269-1279, June 2004.

[Unge82] G. Ungerböck. Channel coding with multilevel/phase signals. *IEEE Trans. Inf. Theory*, 28:55-67, January 1982.

[VKK02] B. Vasic, E.M. Kurtas, and A.V. Kuznetsov. Kirkman systems and their application in perpendicular magnetic recording. *IEEE Trans. Magnetics*, 38(4), July 2002

[VM04] B. Vasic and O. Milenkovic. Combinatorial constructions of low density parity check codes for iterative decoding. *IEEE Trans. Inform. Theory*, 50(6):1156-1176, June 2004.

[Wibe94] N. Wiberg. *Approaches to neural-network decoding of error correcting codes*. Diploma Thesis, Univ. Linköoping, Sweden, 1994.

[Wibe96] N. Wiberg. *Codes and decoding on general graphs*. PhD Thesis, Univ. Linköping, Sweden, 1996.

[WKL95] N. Wiberg, R. Kötter, and H.-A. Loeliger. Codes and iterative decoding on general graphs. *European Trans. on Telecom.*, 6:513-525, September-October 1995.

[XJ04] Bo Xia and Eric Jacobsen. Intel LDPC proposed for IEEE 802.16e. *Intel submission to informal LDPC group*, 13 August 2004.

[YLR02] M. Yang, Y. Li, and W.E. Ryan. Design of efficiently-encodable moderate-length high-rate irregular LDPC codes. In *Proc. 40th Annual Allerton Conf. on Comm., Control, and Computing*, pages 1415-1424, October 2002

[YHB04] M.R. Yazdani, S. Hemati, and A.H. Banihashemi. Improving belief propagation on graphs with cycles. *IEEE Communications Letters*, vol. 8, pp. 57-59, Jan. 2004

[Zhe06] X. Zheng, F. C. M. Lau, C. K. Tse and S. C. Wong. Techniques for improving block error rate of LDPC decoders. In the proceeding *IEEE International Symposium on Circuits and Systems.* May 21-24, 2006, Island of Kos, Greece, p. 2261-2264.

[Zimm04] E. Zimmermann et al. Reduced Complexity LDPC Decoding using Forced Convergence. In *Proceedings of the 7th International Symposium on Wireless Personal Multimedia Communications* (WPMC04)

[ZB04] P. Zarrinkhat and A.H. Banihashemi. Hybrid hard-decision iterative decoding of regular low density parity check codes. *IEEE Comm.,* 8(4):250-252, April 2004

[ZM03] H. Zhang and J.M.F. Moura. Large-girth LDPC codes based on graphical models", *IEEE Int. Workshop on Signal Processing Advance for Wireless Comm.*, Rome, Italy, June 2003

[ZP75] V. Zyablov and M. S. Pinsker. Estimation of the error-correcting complexity of Gallager low-density codes. *Problems of Info. Trans.*, 11(1):23-26, 1975.

[ZP01a] T. Zhang and K.K. Parhi. A class of efficient-encoding generalized low-density parity-check codes. In *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 4:2477-2480, May 2001.

[ZP01b] T. Zhang and K.K. Parhi. High-performance, low-complexity decoding of generalized low-density parity-check codes. In *IEEE Global Telecomm. Conf.*, 1:181-185, November 2001.

[ZP01c] T. Zhang and K.K. Parhi, VLSI Implementation-Oriented (3,k)-Regular Low-Density Parity-Check Codes. In *Proc. of the 2001 IEEEWorkshop on Signal Processing Systems (SiPS)*, Antwerp, Belgium, September 26-28, 2001

[ZP01d] T. Zhang and K.K. Parhi, Joint code and decoder design for implementation-oriented (3,k)-regular LDPC codes. In *Proc. of the 35th Asilomar Conf. Signals, Systems and Computers,* Pacafic Grove, CA, Nov 2001.

[ZZ04] H. Zhong and T. Zhang. *Joint code-encoder-decoder design for LDPC coding system VLSI implementation*. In Proc. Of ISCAS2004, 23-26 May 2004

APPENDIX

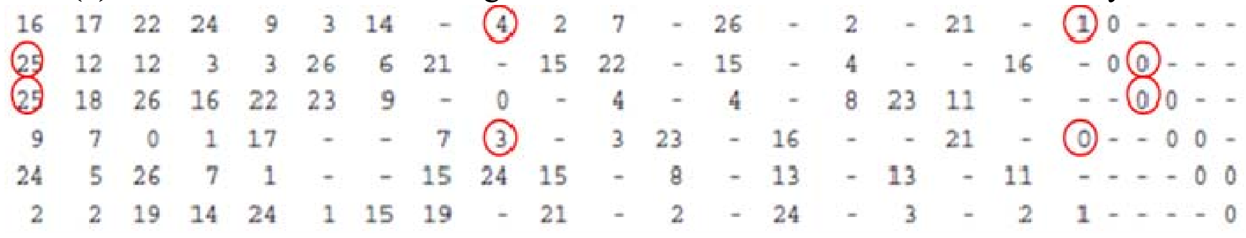
CHANNEL CODES USED

Chapter 3

MacKay code [MN95]
Array LDPC codes [Fan00, EO01]
IEEE802.11n codes [IEEE09] with code rates: 1/2, 2/3, 3/4 and 5/6 and code length: 648, 1,296 and 1,944 bits

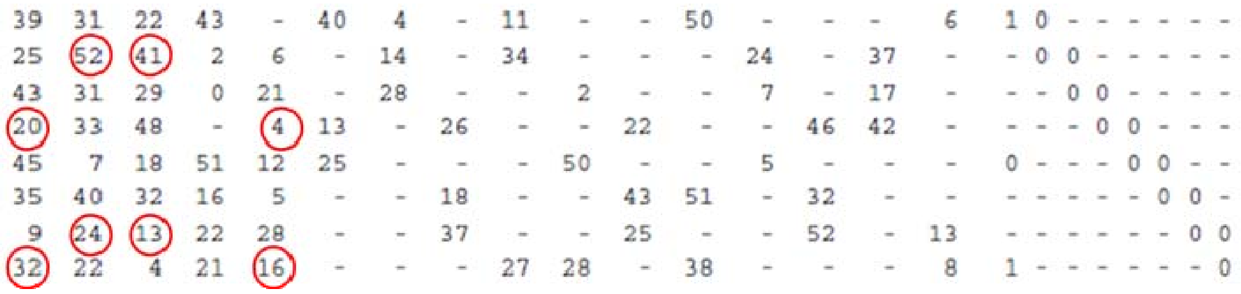
WLAN codes with short-four-cycles

(a) WLAN code with code length 648 bits and code rate ¾ with short-four-cycles



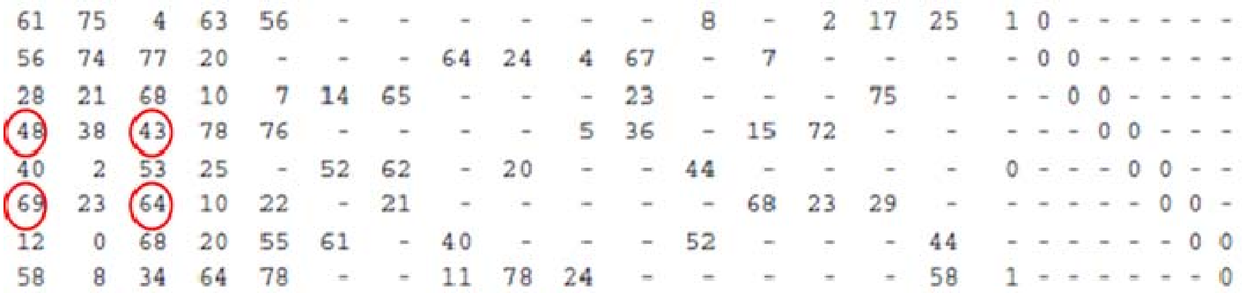
16	17	22	24	9	3	14	-	4	2	7	-	26	-	2	-	21	-	1	0	-	-	-	-
25	12	12	3	3	26	6	21	-	15	22	-	15	-	4	-	-	16	-	0	0	-	-	-
25	18	26	16	22	23	9	-	0	-	4	-	4	-	8	23	11	-	-	-	0	0	-	-
9	7	0	1	17	-	-	7	3	-	3	23	-	16	-	-	21	-	0	-	-	0	0	-
24	5	26	7	1	-	-	15	24	15	-	8	-	13	-	13	-	11	-	-	-	-	0	0
2	2	19	14	24	1	15	19	-	21	-	2	-	24	-	3	-	2	1	-	-	-	-	0

(b) WLAN code with code length 1,296 bits and code rate 2/3 with short-four-cycles



39	31	22	43	-	40	4	-	11	-	-	50	-	-	-	6	1	0	-	-	-	-	-	-
25	52	41	2	6	-	14	-	34	-	-	-	24	-	37	-	-	0	0	-	-	-	-	-
43	31	29	0	21	-	28	-	-	2	-	-	7	-	17	-	-	-	0	0	-	-	-	-
20	33	48	-	4	13	-	26	-	-	22	-	-	46	42	-	-	-	-	0	0	-	-	-
45	7	18	51	12	25	-	-	-	50	-	-	5	-	-	-	0	-	-	-	0	0	-	-
35	40	32	16	5	-	-	18	-	-	43	51	-	32	-	-	-	-	-	-	-	0	0	-
9	24	13	22	28	-	-	37	-	-	25	-	-	52	-	13	-	-	-	-	-	-	0	0
32	22	4	21	16	-	-	-	27	28	-	38	-	-	-	8	1	-	-	-	-	-	-	0

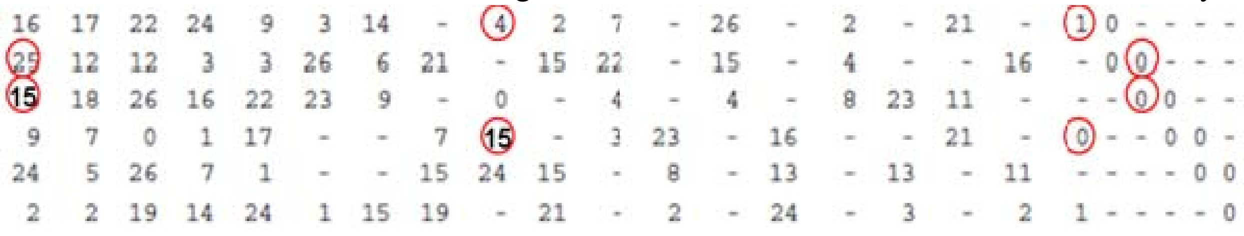
(c) WLAN code with code length 1,944 bits and code rate 2/3 with short-four-cycles



61	75	4	63	56	-	-	-	-	-	-	8	-	2	17	25	1	0	-	-	-	-	-	-
56	74	77	20	-	-	-	64	24	4	67	-	7	-	-	-	-	0	0	-	-	-	-	-
28	21	68	10	7	14	65	-	-	-	23	-	-	-	75	-	-	-	0	0	-	-	-	-
48	38	43	78	76	-	-	-	-	5	36	-	15	72	-	-	-	-	-	0	0	-	-	-
40	2	53	25	-	52	62	-	20	-	-	44	-	-	-	-	0	-	-	-	0	0	-	-
69	23	64	10	22	-	21	-	-	-	-	-	68	23	29	-	-	-	-	-	-	0	0	-
12	0	68	20	55	61	-	40	-	-	-	52	-	-	-	44	-	-	-	-	-	-	0	0
58	8	34	64	78	-	-	11	78	24	-	-	-	-	-	58	1	-	-	-	-	-	-	0

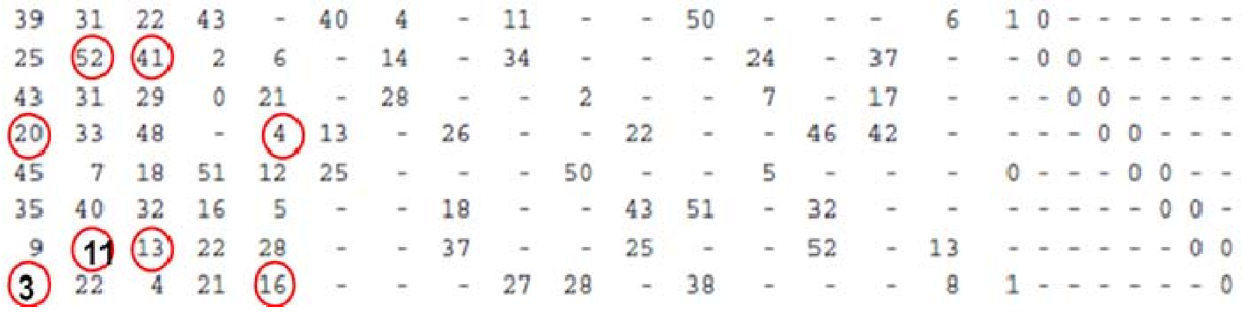
Modified WLAN codes without short-four-cycles

(a) Modified WLAN code with code length 648 bits and code rate 3/4 without short-four-cycles



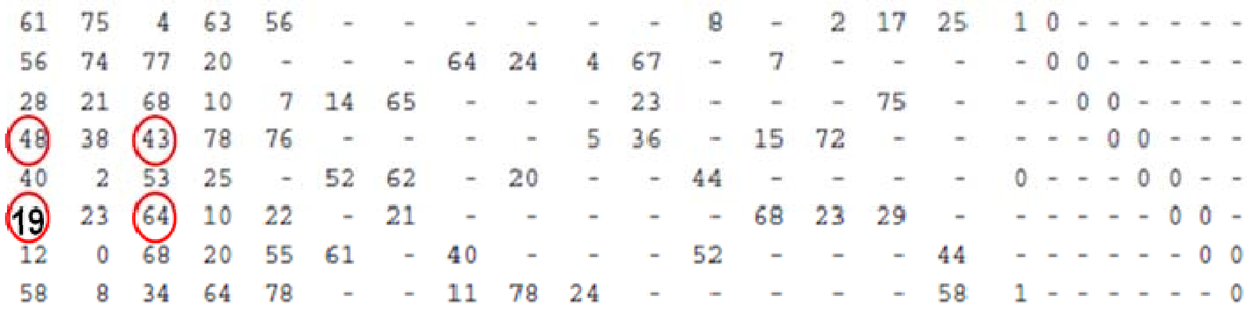
16	17	22	24	9	3	14	-	4	2	7	-	26	-	2	-	21	-	1	0	-	-	-	-
25	12	12	3	3	26	6	21	-	15	22	-	15	-	4	-	-	16	-	0	0	-	-	-
15	18	26	16	22	23	9	-	0	-	4	-	4	-	8	23	11	-	-	-	0	0	-	-
9	7	0	1	17	-	-	7	15	-	3	23	-	16	-	-	21	-	0	-	-	0	0	-
24	5	26	7	1	-	-	15	24	15	-	8	-	13	-	13	-	11	-	-	-	-	0	0
2	2	19	14	24	1	15	19	-	21	-	2	-	24	-	3	-	2	1	-	-	-	-	0

(b) Modified WLAN code with code length 1296 bits and code rate 2/3 without short-four-cycles



39	31	22	43	-	40	4	-	11	-	-	50	-	-	-	6	1	0	-	-	-	-	-	-
25	52	41	2	6	-	14	-	34	-	-	-	24	-	37	-	-	0	0	-	-	-	-	-
43	31	29	0	21	-	28	-	-	2	-	-	7	-	17	-	-	-	0	0	-	-	-	-
20	33	48	-	4	13	-	26	-	-	22	-	-	46	42	-	-	-	-	0	0	-	-	-
45	7	18	51	12	25	-	-	-	50	-	-	5	-	-	-	0	-	-	-	0	0	-	-
35	40	32	16	5	-	-	18	-	-	43	51	-	32	-	-	-	-	-	-	-	0	0	-
9	11	13	22	28	-	-	37	-	-	25	-	-	52	-	13	-	-	-	-	-	-	0	0
3	22	4	21	16	-	-	-	27	28	-	38	-	-	-	8	1	-	-	-	-	-	-	0

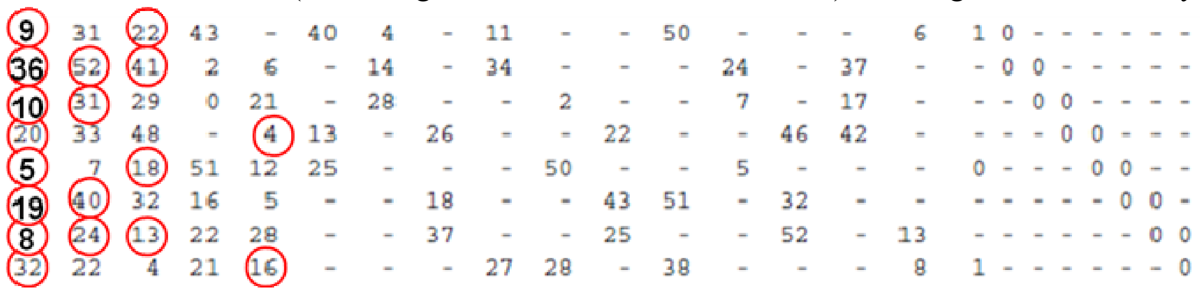
(c) Modified WLAN code with code length 1944 bits and code rate 2/3 without short-four-cycles



61	75	4	63	56	-	-	-	-	-	-	8	-	2	17	25	1	0	-	-	-	-	-	-
56	74	77	20	-	-	-	64	24	4	67	-	7	-	-	-	-	0	0	-	-	-	-	-
28	21	68	10	7	14	65	-	-	-	23	-	-	-	75	-	-	-	0	0	-	-	-	-
48	38	43	78	76	-	-	-	-	5	36	-	15	72	-	-	-	-	-	0	0	-	-	-
40	2	53	25	-	52	62	-	20	-	-	44	-	-	-	-	0	-	-	-	0	0	-	-
19	23	64	10	22	-	21	-	-	-	-	-	68	23	29	-	-	-	-	-	-	0	0	-
12	0	68	20	55	61	-	40	-	-	-	52	-	-	-	44	-	-	-	-	-	-	0	0
58	8	34	64	78	-	-	11	78	24	-	-	-	-	-	58	1	-	-	-	-	-	-	0

Modified WLAN codes with higher short-four-cycles

Modified WLAN codes (code length 1296 bits and code rate 2/3) with higher short-four-cycles



9	31	22	43	-	40	4	-	11	-	-	50	-	-	-	6	1	0	-	-	-	-	-	-
36	52	41	2	6	-	14	-	34	-	-	-	24	-	37	-	-	0	0	-	-	-	-	-
10	31	29	0	21	-	28	-	-	2	-	-	7	-	17	-	-	-	0	0	-	-	-	-
20	33	48	-	4	13	-	26	-	-	22	-	-	46	42	-	-	-	-	0	0	-	-	-
5	7	18	51	12	25	-	-	-	50	-	-	5	-	-	-	0	-	-	-	0	0	-	-
19	40	32	16	5	-	-	18	-	-	43	51	-	32	-	-	-	-	-	-	-	0	0	-
8	24	13	22	28	-	-	37	-	-	25	-	-	52	-	13	-	-	-	-	-	-	0	0
32	22	4	21	16	-	-	-	27	28	-	38	-	-	-	8	1	-	-	-	-	-	-	0

Chapter 4

Maximal number of decoding iteration
Stair Cascade code: code length 1,200 bits, code rate 3/4 and column weight 6

[146+203+285+193+298+1][104+121+55+281+184+124] ...
[169+154+215+247+208+165][0]

Code rates
Stair Cascade code: code rate. 1/2, 2/3 and 3/4, code length 1,200 bits, column weight 6

Code rate	Slope parameter
1/2	[545+216+589+518+382+94][0]
2/3	[109+311+190+87+325+38][83+377+199+157+238+282][0]
3/4	[146+203+285+193+298+1][104+121+55+281+184+124] ... [169+154+215+247+208+ 165][0]

Column weight
Stair Cascade code: code length 1,200 bits, column weight. 3, 6 and 9, code rate 1/2

Column weight	Slope parameter
3	[547+548+572][0]
6	[545+216+589+518+382+94][0]
9	[31+235+515+467+85+214+117+228+134][0]
12	[423+380+126+57+207+544+225+128+171+289+341+ 33][0]

Stair Cascade code: code length 1,200 bits, , column weight. 3, 6 and 9, code rate 2/3

Column weight	Slope parameter
3	[195+323+100][128+345+54][0]
6	[109+311+190+87+325+38][83+377+199+157+238+282][0]
9	[346+102+173+362+190+124+84+131+234] [105+2 40+192+186+186+140+194+164][0]

Stair Cascade code: code length 1,200 bits, column weight 3, 6 and 9, code rate. 3/4

Column weight	Slope parameter
3	[202+50+88][241+191+206][83+127+210][0]
6	[146+203+285+193+298+1][104+121+55+281+184+124] [169+154+215+247+208+165][0]
9	[155+51+54+219+11+133+64+227+149][273+280+275+220+297+... 89+21+151+282][230+235+186+138+124+240+152+86+229][0]

Maximal number of decoding iteration
Stair Lattice code: code length 1,200 bits, code rate ¾, column weight 6

[[24+92+124][31+8+42];[120+21+118][99+145+91]] …
[[25+55+85][15+50+51];[27+30+94][39+26+146]] …
[[111+142+131][106+2+88];[[138+87+29][97+68+95]][0]

Code rates

Stair Lattice code: code rate 1/2, 2/3 and ¾, code length 1,200 bits, column weight 6

Code rate	Slope parameter
1/2	[[234+226+137][109+201+205];[291+293+60][267+180+151]][0]
2/3	[[126+57+128][171+33+89];[196+113+133][80+22+28]] ... [[65+118+25][85+61+164];[125+24+183][109+162+21]][0]
3/4	[[7+25+89][141+133+91];[117+101+99][76+24+43]] ... [[83+128+51][88+144+116];[18+104+16][132+52+50]] ... [[106+108+111][55+31+96];[138+118+75][70+97+39]][0]

Column weight

Stair Lattice code: code length 1,200 bits, column weight 4, 6 and 8, code rate 1/2

Column weight	Slope parameter
4	[[70+27][271+191];[266+35][117+73]][0]
6	[[234+226+137][109+201+205];[291+293+60][267+180+151]][0]
8	[[99+170+187+121][81+128+231+102]; ... [37+189+280+183][137+191+161+242]][0]

Stair Lattice code: code length 1,200 bits, column weight 4, 6 and 8, code rate 2/3

Column weight	Slope parameter
4	[[19+123][28+89];[103+31][194+106]] ... [[159+71][42+48];[190+62][110+84]][0]
6	[[126+57+128][171+33+89];[196+113+133][80+22+28]] ... [[65+118+25][85+61+164];[125+24+183][109+162+21]][0]
8	[[43+122+158+59][198+126+108+75];[71+29+17+140] ... [155+162+193+72]][[85+66+135+134][12+116+20+150]; ... [173+52+153+113][139+5+30+22]][0]

Stair Lattice codes: code length 1,200 bits, column weight 4, 6 and 8, code rate 3/4.

Column weight	Slope parameter
4	[[69+120][51+81];[25+149][56+98]] ... [[79+125][64+37];[124+133][131+105]] ... [[28+26][20+113];[135+34][40+147]][0]
6	[[7+25+89][141+133+91];[117+101+99][76+24+43]] ... [[83+128+51][88+144+116];[18+104+16][132+52+50]] ... [[106+108+111][55+31+96];[138+118+75][70+97+39]][0]
8	[[30+126+107+82][38+9+142+108];[75+81+76+94][103+19+45+1]] ... [[57+49+48+25][64+139+23+91];[22+6+96+87][92+58+89+125]] ...

[[133+68+13+138][130+129+101+62];[135+33+98+41][12+31+131+140]][0]

Comparison of Both Stair Codes

Stair codes: code length 1,200 bits, code rate ¾, column weight 6

Type	Slope parameter
Cascade	[146+203+285+193+298+1][104+121+55+281+184+124]
	[169+154+215+247+208+165][0]
Lattice-4	[[7+25+89][141+133+91];[117+101+99][76+24+43]] ...
	[[83+128+51][88+144+116];[18+104+16][132+52+50]] ...
	[[106+108+111][55+31+96];[138+118+75][70+97+39]] [[0][-];[-][0]]
Lattice-9	[[23+94][86+46][40+59];[70+92][74+48][36+24];[69+65][5+87][47+72]] ...
	[[32+91][80+42][88+43];[58+17][90+20][97+71];[62+57][49+38][61+81]] ...
	[[25+12][3+89][50+73];[15+29][93+30][26+27];[35+85][14+33][9+13]] ...
	[[0][-][-];[-][0][-];[-][-][0]]

Stair Lattice code with code rate 3/8, 4/8 and 5/8.

Code rate	Slope parameter
3/8	[[128][124][88];[100][67][129];[99][40][6];[144][35][63];[71][62][58]] ...
	[[0][-][-][-][-];[-][0][-][-][-];[-][-][0][-][-];[-][-][-][0][-];[-][-][-][-][0]
4/8	[[134][109][22][93]; [39][59][27][62];[44][80][144][119]; [142][56][84][72]] ...
	[[0][-][-][-];[-][0][-][-];[-][-][0][-];[-][-][-][0]]
5/8	[[79][74][46][101][144];[121][89][41][20][81];[116][128][64][123][147]] ...
	[[0][-][-];[-][0][-];[-][-][0]]

High-rate and Short-length Performances

Stair Cascade code: code length 1,200 bits, column weight 6, code rate 3/4, 7/8 and 19/20

Code rate	Slope parameter
3/4	[146+203+285+193+298+1][104+121+55+281+184+124] ...
	[169+154+215+247+208+165][0]
7/8	[77+49+141+145+120+91][112+129+126+26+110+103] ...
	[69+4+140+125+46+116][114+139+97+12+7+62] ...
	[88+29+21+111+101+132][81+86+127+34+31+19] ...
	[95+53+130+52+41+94][0]
19/20	[12+2+0+51+54+45][21+8+25+1+28+53][18+37+11+30+33+7] ...
	[34+13+38+50+44+42][26+6+40+55+23+39][19+9+31+34+4+35] ...
	[29+52+57+3+32+46][49+41+20+10+14+43][15+48+56+36+17+22] ...
	[16+27+59+47+5+58][0+5+30+46+24+14][47+36+56+42+18+3] ...
	[48+43+26+53+28+31][1+49+22+11+33+50][54+32+20+45+2+25] ...
	[34+6+59+52+55+4][17+35+44+12+41+29][7+21+10+9+38+39] ...
	[13+40+57+27+19+37][0]

Stair Cascade code: code length 128, 576, 2304 and 6000 bits, code rate 3/4

Code length	Slope parameter
128	[19+10+15+7+17+20][2+23+5+4+18+14][25+6+31+8+26+27][0]
576	[24+105+123+26+69+94][11+31+122+141+16+116][62+23+114+81+7+60][0]
2304	[177+134+378+298+426+43][373+95+530+87+435+496] [45+510+71+142+438+55][0]
6000	[524+581+241+663+1354+332][571+864+676+928+1216+1150] [1121+41+277+282+1122+988][0]

Regular vs Irregular

Regular Stair Cascade code: code length 1,200 bits, code rate 2/3 column weight 6,
[109+311+190+87+325+38][83+377+199+157+238+282][0]

Irrgular Stair Cascade code: code length 1,200 bits, code rate 2/3 column weight 8 and 4
[165+182+378+322+194+243+31+89][339+76+133+242][0]

Regular Stair Cascade code: code length 1,200 bits, code rate 3/4 column weight 6,
[277+246+53+102+84+73][190+ 208+213+146+107][4+295+200+38+199+249][0]

Irrgular Stair Cascade code: code length 1,200 bits, code rate 3/4 column weight 9, 5, 4
[273+144+22+270+138+228+149+31+263][200+192+283+8+7] ...
[60+110+126+111][0]

Code benchmarking

Comparison with Random MacKay codes

Random MacKay code: code length 1,200 bits, code rate ¾, column weight 3

Stair Cascade code: code length 1,200 bits, code rate 3/4 and column weight 6
[148+88+222+105+245+25][204+161+100+198+203+110] ...
[90+231+45+31+279+21][0]

Stair Lattice code: code length 1,200 bits, code rate 3/4 and column weight 6
[[120+146+132][143+86+119];[60+137+38][63+85+102]] ...
[[118+28+87][80+44+123];[56+116+41][122+4+88]] ...
[[14+31+17][10+90+52];[110+18+59][82+68+140]][0]

Comparison with Short Length Differential families Code

Differential families [404, 303]: column weight 5, 5, 3, 2, code rate 3/4., code length 400 bits

Stair Cascade code with code length 200 bits, code rate 3/4 and column weight 6, maximal iteration 20.
[36+31+18+44+4+25][3+23+26+41+19+42][35+9+20+37+7+45][0]

Stair Lattice code with code length 200 bits, code rate 3/4 and column weight 6, maximal iteration 20

[[9+5+19][22+24+17];[0+3+6][1+11+23]] ...
[[2+20+18][8+4+7];[1+16+4][7+24+17]] ...
[[10+15+21][14+2+23];[11+19+12][18+5+0]][0]

Stair Cascade code: code length 400 bits, column weight 3, code rate 3/4

[74+ 21+72][18+79+7][83+9+57][0]

Stair Cascade code: code length 400 bits, column weight 5, code rate 3/4

[54+4+86+ 65+ 60][99+7+97+36+72][90+61+95+27][0]

Comparison with High-Rate Code

Unital LDPC code [JW03] with code length 416 bits and code rate 0.84375

Stair Cascade Code: code length 420 bits, code rate 6/7 (0.857)

[32+40+22+34+35+6][55+3+16+11+54+30][45+0+51+33+7+38] ...
[58+42+28+17+41+47][14+46+56+8+59+5][48+53+29+34+25+52][0]

Stair Cascade code: code length 480 bits, code rate 7/8 (0.875).

[52+32+19+7+12+50][2+18+14+10+28+11][46+59+38+5+39+47] ...
[42+6+34+30+24+44][20+37+55+22+21+23][31+41+16+34+3+8] ...
[4+33+0+53+26+57][0]

Stair Cascade code: code rate 0.93 with code length 7,200 bits

[240+24+252+220+92+155][437+409+124+31+162+163][8+42+153+360+159+208] ...
[152+0+451+242+305+472][193+120+330+166+21+300][118+346+99+34+324+470]
[194+463+314+171+429+347][364+158+406+336+291+476] ...
[207+181+474+457+145+91][221+25+294+372+55+175] ...
[229+213+356+228+367+422][199+448+85+198+389+15] ...
[285+381+50+164+196+230][51+261+27+248+30+94][0]

Comparison with WLAN codes

Stair Cascade code: code length 1,296 bits, code rate 1/2, column weight 6.

[366+371+608+521+275+122][0]

Stair Lattice code: code length 1,296 bits, code rate 1/2, , column weight 6.

[[128+142+270][231+272+73];[101+249+16][320+293+277]][0]

Stair Cascade code: code length 1,296 bits, code rate 2/3

[189+206+402+346+429+218][267+55+113+363+100+13][0]

Stair Lattice code: code length 1,296 bits, code rate 2/3

[[103+97+153][137+126+41];[141+122+99][59+113+23]] ...

[[185+171+73][191+30+40];[214+25+206][133+197+129]][0]

Stair Cascade code: code length 1,296 bits, code rate 3/4
[45+159+276+72+161+211][50+174+221+273+81+311] ...
[87+125+41+105+261+271][0]

Stair Lattice code: code length 1,296 bits, code rate 3/4
[[7+19+16][60+68+54];[106+139+59][161+102+66]] ...
[[11+21+49][46+134+89];[147+138+48][3+152+158]] ...
[[42+87+145][88+4+109];[62+140+144][119+115+104]] [0]

Stair Cascade code: code length 1,296 bits, code rate 5/6
[62+189+120+30+50+110][34+73+84+78+180+185][97+152+150+172+8+176] ...
[89+153+4+105+76+158][199+22+140+23+7+201] [0]

Stair Lattice code: code length 1,296 bits, code rate 5/6
[[78+16+21][105+27+85];[61+100+67][71+36+10]] ...
[[48+58+98][12+52+5];[19+11+103][7+84+81]] ...
[[72+75+20][90+89+95];[31+94+34][51+66+93]] ...
[[2+40+62][37+41+53];[33+73+13][64+92+22]] ...
[[87+65+74][59+49+101];[47+30+82][15+45+102]][0]

Stair Cascade code: code length 648 bits, code rate 5/6, column weight 6
[2+14+38+39+36+107][23+10+25+73+13+37][47+58+55+20+86+98] …
[68+46+24+40+52+105][28+74+77+67+90+57][0]

Stair Cascade code: code length 1,296 bits, code rate 5/6
[62+189+120+30+50+110][34+73+84+78+180+185][97+152+150+172+8+176] ...
[89+153+4+105+76+158][199+22+140+23+7+201] [0]

Stair Cascade code: code length 1,944 bits, code rate 5/6
[11+241+61+256+69+191][85+320+251+144+52+172][80+68+213+242+134+16] ...
[7+246+45+294+18+226][236+273+142+153+81+63][0]

Stair Latice code: code length 648 bits, code rate 5/6
[[40+43+53][6+28+20];[22+24+23][25+7+8]] …
[[49+15+14][36+33+1];[52+26+0][21+32+17]] …
[[30+39+2][9+46+5];[50+38+5][28+53+35]] …
[[11+10+48][15+4+23];[2+0+24][31+26+40]] …
[[45+20+42][30+25+16];[27+18+12][46+47+43]][0]

Stair Cascade code: code length 1,944 bits, code rate 5/6
[11+241+61+256+69+191][85+320+251+144+52+172][80+68+213+242+134+16] ...
[7+246+45+294+18+226][236+273+142+153+81+63][0]

Stair Lattice code: code length 1,296 bits, code rate 3/4

[[7+19+16][60+68+54];[106+139+59][161+102+66]] ...
[[11+21+49][46+134+89];[147+138+48][3+152+158]] ...
[[42+87+145][88+4+109];[62+140+144][119+115+104]][0]

Stair Cascade code: code length 1,296 bits, code rate ½, column weight 6

Code	Slope parameter	Average girth degree of girth 4
1	[101+293+240+436+564+290][0]	0
2	[366+371+608+521+275+122][0]	2
3	[10+20+30+40+60+80][0]	24
4	[10+20+30+40+50+60][0]	40

Stair Lattice code: code length 1,296 bits, code rate 3/4 and column weight 6

Code	Slope parameter	Average girth degree of girth 4
1	[7+19+16 60+68+54; 106+139+59 161+102+66] [11+21+49 46+134+89; 147+138+48 3+152+158] [42+87+145 88+4+109; 62+140+144 119+115+104][0]	3
2	[10+20+30 111+67+6; 20+30+40 144+33+106] [60+80+100 23+64+39; 30+50+70 45+94+26] [50+70+90 115+52+117; 70+90+110 151+4+135][0]	24
3	[10+20+30 50+70+90; 20+30+40 60+80+100] [60+80+100 70+90+110; 30+50+70 80+100+120] [50+70+90 20+40+60; 70+90+110 50+70+90][0]	96

Chapter 5

MacKay code [MN95]
array LDPC codes [Fan00, EÖ01]