The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

Study of Short-Length Low-Density-Parity-Check Codes

XIA ZHENG

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

March 2009

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree of diploma, except where due acknowledgement has been made in the text.

_____(Signed)

_____(Name of Student)

Abstract

Low-density parity-check (LDPC) codes can be applied to many different types of communication systems such as video broadcasting and satellite communications. However, the performance of the codes in practice may deviate a lot from the theoretical limits, achievable only by codes with infinite length. In this thesis, we will conduct a detailed investigation on the performance of short-length LDPC codes over additive white Gaussian noise (AWGN) channels.

First, we look into the rich dynamical behavior occurring at the LDPC decoders when the LDPC code has a finite length. We will report the various types of bifurcation phenomena as the signal-to-noise ratio (SNR) increases. By linearizing the iterative equations used for decoding the signals, we are able to evaluate the eigenvalues corresponding to the fixed points. Based on the eigenvalues, we can further characterize the properties of the fixed points. In addition, we make use of a simple feedback technique in an attempt to improving the convergence rate of the decoder at the waterfall (medium-SNR) region.

Then, we attempt to "optimize" the distributions of the variable-node degrees and check-node degrees such that lower error rates can be produced. Although some "optimized" variable-node and check-node degree distributions have already been reported in the literature, they are found based on the assumption that the LDPC codes having an infinite length. Therefore, the error performance of a finite-length LDPC code obeying such degree distributions may not be "optimized" and there are possibilities that codes following other degree distributions may produce better performance in terms of block/bit error rates (BLERs/BERs). Since the LDPC decoders rely on passing messages iteratively between the set of variable nodes and the set of check nodes, decreasing the short average path length (APL) will no doubt accelerate the exchange of messages among the variable nodes, thereby reducing

the number of iterations the decoder algorithm takes to converge. Inspired by the short average path length (APL) between nodes in complex networks with scale-free (SF) degree distributions, we make the first attempt in applying complex-network theories to communications engineering and build short-length LDPC codes with variable-node degrees following SF degree distributions. Not only have we evaluated the theoretical performance of our SF-LDPC codes, but also the simulated error rates. Furthermore, we compare our results with those of other well-known LDPC codes.

As we simulate the BLER of short-length LDPC codes with an increasing SNR, we further observe that the BLER is decreasing with a much lower rate when the SNR becomes large, implying that the BLER has reached an error floor. Investigations have revealed that the block errors at the high SNR region are mainly caused by trapping sets (TSs) with their induced connected subgraphs forming one or more cycles. To easily differentiate trapping sets with (i) no cycle, (ii) a single-cycle or (iii) multiple cycles, we introduce a new parameter, namely cycle indicator. Moreover, we define a "primary trapping set (PTS)" with an aim to identifying harmful TSs to the decoder. Realizing the types of PTSs that are more likely to contribute to the error floor, we propose a code-construction method that aims to avoid such harmful PTSs. Codes so constructed will be evaluated and compared with those built using other mechanisms.

Finally, as we keep increasing the SNR, a point will be reached where running Monte Carlo (MC) simulations will no longer be feasible. It is because the BLER has become so low that it will take an extremely long simulation time before an adequate number of errors are collected. In order to evaluate more effectively and efficiently the extremely low error rates of the codes at the high SNR region, we propose a simulation scheme that combines the use of importance sampling (IS) and PTS identification. Compared with the MC simulations, the proposed IS scheme produces speed-up gains of up to 3.9184×10^9.

List of Publications

Journal Papers

- **X. Zheng**, F.C.M. Lau, C.K. Tse and S.C. Wong, Study of Bifurcation Behavior of LDPC Decoders. *International Journal of Bifurcation and Chaos*, 16(11):3435–3449, 2006.
- **X. Zheng**, F.C.M. Lau, C.K. Tse and Yejun He, Application of Complex-Network Theories to the Design of Short-length LDPC Codes, *IET Communications*, accepted.

Papers submitted

- **X. Zheng**, F.C.M. Lau and C.K. Tse, Short-length LDPC codes with extremely low error rate—Part-I: Code Construction, *IEEE Trans. Commun.*, submitted.
- **X. Zheng**, F.C.M. Lau and C.K. Tse, Short-length LDPC codes with extremely low error rate—Part-II: Performance Evaluation, *IEEE Trans. Commun.*, submitted.
- **X. Zheng**, F.C.M. Lau, C.K. Tse and M.Z. Wang, Categorization of trapping sets based on their induced connected subgraph, *IEEE Commun. Lett.*, submitted.

Conference papers

- **X. Zheng**, F.C.M. Lau, C.K. Tse, Y. He and M.Z. Wang, Evaluation of the extremely low block error rate of irregular LDPC codes, In *Proc. IEEE International Conference on Communications*, Dresden, Germany, June 2009, accepted.

- **X. Zheng**, F.C.M. Lau and C.K. Tse, Construction of short-length LDPC codes with low error floor. In *Proc. IEEE Asia Pacific Conference on Circuits and Systems*, Macao, China, Dec. 2008, pages 1818–1821.

- **X. Zheng**, F.C.M. Lau and C.K. Tse, Short-length LDPC codes with power-law distributed variable-node degrees. In *Proc. Int. Symp. Nonlinear Theory and Its Applications*, Budapest, Republic of Hungary, Sep. 2008, pages 168–171.

- **X. Zheng**, F.C.M. Lau and C.K. Tse, Error performance of short-block-length LDPC codes built on scale-free networks. In *Proc. 3rd Shanghai Int. Symp. Nonlinear Science and Applications*, Shanghai, China, Jun. 2007, pages 55–57.

- **X. Zheng**, F.C.M. Lau and C.K. Tse, Study of LDPC codes built on scale-free networks. In *Proc. Int. Symp. Nonlinear Theory and Its Applications*, Bologna, Italy, Sep. 2006, pages 563–566.

- **X. Zheng**, F.C.M. Lau, C.K. Tse and S.C. Wong, Techniques for improving block error rate of LDPC decoders. In *Proc. IEEE Int. Symp. Circ. Syst.*, Kos, Greece, May 2006, pages 2261–2264.

- **X. Zheng**, F.C.M. Lau, C.K. Tse and S.C. Wong, Study of nonlinear dynamics of LDPC decoders. In *Proc. European Conf. Circuit Theory and Design*, Cork, Ireland, Aug. 2005, pages 157–160.

Acknowledgements

I would like to give my sincere gratitude to my supervisor Dr. Francis C.M. Lau, for his valuable advice, patient guidance and support which made this study possible. His research intuition and experience inspired my great interest in channel coding and guided me through the four years' PhD study. I am extremely grateful to him for his numerous proofread as well as insightful suggestions to my thesis work.

I would also like to thank Prof. C.K. Tse who devotes himself to providing a top research environment for us. His full enthusiasm to research and strong sense of responsibility promote our great interest in research.

I gratefully acknowledge the Research Committee of The Hong Kong Polytechnic University for its financial support during the entire period of my candidature.

Last, but not least, I must thank my family, and my friends for their love and care over the years.

Contents

Chapter 1

Introduction

1.1 Motivation and objectives

Turbo codes [1–3] and low-density-parity-check (LDPC) codes [4] have been widely proposed for use in coding digital messages. Their ability to provide bit-error performance close to the Shannon limit [5] has aroused much interest in the research community over the past decade. The LDPC codes were first proposed by Gallager [4] in the early 1960s. The idea was subsequently forgotten. Recently, with the rapid development of the computational techniques as well as the popularization of the personal computers, LDPC codes have been revisited [6–9] and shown to outperform the popular turbo codes with lower error floor, comparatively lower decoding complexity, and simpler implementation in hardware circuits.

The original regular LDPC codes, where viewed in a matrix format, have nearly uniform weights per row and per column. The use of bipartite graph to represent the codes [10] further brings the variation and extension of the original definition to irregular LDPC codes, of which the node degrees in the bipartite graphs are chosen according to some degree distributions. It has been shown, in general, that irregular LDPC codes outperform regular ones in terms of error rate under similar scenarios [11]. Suppose the most common algorithm, namely the sum-product iterative decoding algorithm or the belief propagation (BP) algorithm [11], is used in the decoder. The optimized degree distribution for irregular LDPC codes can

be found by maximizing the threshold in the well-known "density evolution (DE)" mechanism. The threshold in fact determines the performance of the code when the code length is infinite. For example, for the bit erasure channel (BEC) [12–14] and the additive white Gaussian noise (AWGN) channel [11, 15], the threshold can be viewed as the maximum erasure probability and maximum standard deviation of noise, respectively, below which the bit error rate (BER) of the code tends to zero. Therefore, by varying the variable-node and check-node degree distributions, researchers have been able to optimize the achievable error performance of the LDPC codes under different channel conditions [11, 13, 16–19].

Two assumptions have been made, however, during the application of DE. First, it is assumed that in the BP decoding algorithm, the updated messages passing forward and backward between the set of variable nodes and the set of check nodes are of analog nature, i.e., the messages can assume the values of any real numbers. Practically, all numbers have to be quantized or discretized for computation by hardware. To evaluate the exact behavior of the discretized BP decoder, "discretized density evolution (DDE)" has been developed [20]. It has been concluded that when the messages are quantized into discrete levels using practical quantizers with 10 or more bits, there is little discrepancy between the results obtained by DE and DDE. The other assumption made in the DE algorithm is that the messages propagated along the edges are independent of one another, implying that the code should have an infinite length. In the case of an infinite code length, the decoder will converge to a stable error free codeword if the channel parameter is below the threshold. Unfortunately, codes of infinite length are not applicable in practice. The appearance of cycles in the bipartite graphs associated with finite-length LDPC codes ruins the "independence" assumption, resulting in performance degradation to some extent. Nonetheless, the achievable error performance of an LDPC code with finite length will approach that of an infinite-length LDPC code asymptotically as the code length increases.

With DDE and code rate $1/2$, the achievable error performance of an LDPC code has been found to lie within 0.0045 dB of the Shannon limit under a binary-input AWGN channel [20]. Simulations have also shown that within 0.04 dB of the Shannon limit, a bit error rate of 10^{-6} can be achieved with a code length of 10^7 and about 800 to 1100 iterations [20]. Yet, code lengths larger than 10^6 are

not very practical for many applications because of the huge hardware complexity involved and the long time delay incurred if 1000 iterations are required to decode one codeword [21, 22]. In reality, short-length (around one thousand or less) LDPC codes will find a lot more applications, however their performance may deviate a lot from the asymptotical one.

As aforementioned, the existence of cycles has destroyed the independence assumption required by the BP algorithm. In consequence, the decoder of short-length LDPC codes may not converge even as the channel parameter is less than the threshold. Thus the degree distribution optimized by "density evolution" under an "infinite-length scenario" may not be the best choice for short-length codes. Motivated by this, we decide to perform a detailed performance study of short-length LDPC codes. In particular, we study the codes over an AWGN channel because among various channel settings, the AWGN channel produces simple and tractable mathematical models which can provide insight into the underlying behavior of communication systems [23].

Realizing that the error performance of the short-length LDPC codes will be somewhat degraded compared with that of the infinite ones, we first attempt to modify the decoder so as to produce lower error rate. Hence, we will study the nonlinear dynamical behavior of the short-length LDPC decoders. Beside fixed points, bifurcations, including fold, flip and Neimark-Sacker bifurcations [24], have been observed in the decoders when the signal-to-noise ratio (SNR) lies in the waterfall region. Based on the observations, a simple feedback control technique [25] has been proposed with an aim to improving the convergence rate of the decoder. However, the decoding systems of short-length codes with several hundreds or more bits are high dimensional nonlinear systems. With the totally unknown targets, the contribution of the proposed feedback control techniques in lowering the error rates of the codes is found to be minimal. In the second part, we will attempt to improve the error performance of the codes by "optimizing" the distributions of the variable-node degrees and check-node degrees. Although some "optimized" variable-node and check-node degree distributions have already been reported in the literature, they are found based on the assumption that the LDPC codes having an infinite length. Therefore, the error performance of a finite-length LDPC code obeying such degree distributions may not be "optimized" and there are possibili-

ties that codes following other degree distributions may produce better performance in terms of block/bit error rates (BLERs/BERs). Since the LDPC decoders rely on passing messages iteratively between the set of variable nodes and the set of check nodes, decreasing the short average path length (APL) will no doubt accelerate the exchange of messages among the variable nodes. Inspired by the APL between nodes in complex networks with scale-free (SF) degree distributions, we make the first attempt in applying complex-network theories to communications engineering and build short-length LDPC codes with variable-node degrees following SF degree distributions. Our study discloses that short-length LDPC codes with SF degree distributions can produce very low bit error rates. Moreover, they can even outperform codes with state-of-the-art degree distributions when constructed under the same constraints. We further observe that short-length LDPC codes suffer from the error-floor problem at the high SNR region [26], regardless of their degree distributions. In the third part of our thesis, we dissect the LDPC codes and investigate the root of the error-floor phenomenon. We discover that "primary trapping sets (PTSs)" with certain characteristics are the key elements causing decoding failures at the high SNR region. Based on the findings, we subsequently propose a novel algorithm to construct short-length LDPC codes with an aim to avoiding the detrimental PTSs. As we increase the SNR further, the error rates will become extremely low. The Monte Carlo (MC) method we use to evaluate the error rates becomes not feasible because of the enormous number of simulations required to obtain an adequate number of error counts. To resolve this problem, we introduce a new evaluation approach based on importance sampling and primary-trapping-set identification. Speed up gains of millions of times are then accomplished.

1.2 Nonlinear dynamics of short-length LDPC decoders

It is well known that the BP decoding algorithm can achieve very good error performance when the bipartite graph representation of an LDPC code is cycle-free. Further, a "cycle-free" LDPC code can be achieved asymptotically by increasing the block length of the code. As the block length tends to infinity, the behavior of all the individual codes will then concentrate around its expected behavior of

the ensemble. Many researchers have proposed different methods to analyze the convergence behavior of the iterative decoders as the code length becomes infinite [11, 15, 17, 27, 28]. One of the most popular analysis methods for studying LDPC decoders is *density evolution* (DE) [11], which allows the BP decoding algorithm to converge as the block length tends to infinity. As of today, the asymptotic behavior (as the block-length tends to infinity) of iterative decoding systems has been reasonably well understood. For some classes of channels, there exists a maximum channel parameter σ^* (the threshold) such that the BER tends to zero if $\sigma < \sigma^*$ and converges to a nonzero fixed point if $\sigma > \sigma^*$. In this case, bifurcation phenomena have been observed at the LDPC decoders as the SNR increases beyond the threshold.

In practical scenarios, error-correcting codes of finite-length, especially short-length, are of broader applications [29–34]. However, short codes may show much different properties, such as error performance, from their asymptotically ones due to the appearance of many short cycles [35]. In this regard, various literatures have looked into re-designing short-length LDPC decoders for better performance. In [36–39], different scheduling schemes, e.g., probabilistic schedule, edge-based schedule and node-based schedule, that pass the messages between the variable nodes and the check nodes in a certain order have been applied to LDPC decoders. The other mainstream of the study combines BP decoder and reliability-based decoding, and aims to achieve a tradeoff between the decoding complexity and error performance [40, 41].

In [42] and [2], by modeling a decoder as a discrete dynamical system, the authors have studied in detail the turbo decoding algorithm with a finite code length and under a Gaussian noise channel. They have shown that in addition to fixed points, bifurcations leading to period doubling and oscillations may be produced by the decoding algorithm. In [43], it is further discovered that chaos exists in turbo decoding and a control method has been proposed to improve the convergence rate under such a scenario. Since both the turbo decoders and the LDPC decoders are iterative systems, it is highly probable that the LDPC decoding processes are also rich in nonlinear dynamical phenomena.

In this thesis, we first attempt to study in depth the behavior of short-length LDPC decoders over AWGN channel by analyzing the phase trajectories of the

posterior probabilities as the iterative process progresses. Simulation results have shown that bifurcations, including fold, flip and Neimark-Sacker bifurcations, occur in the whole SNR region. Specifically in the waterfall region, oscillations and chaos are produced and the decoding algorithms do not converge to the fixed points. However, the exact bifurcation behavior in the waterfall region varies for different noise realizations.

Further, we exploit the properties of the two types of fixed points in the decoder: unequivocal fixed point and indecisive fixed point. The unequivocal fixed point corresponds to the case when the decoder has converged to a valid codeword. It is stable in the whole SNR. Nevertheless, the attracting basin of the unequivocal fixed point shrinks as SNR decreases, resulting in non-convergence or convergence to another fixed point, i.e., indecisive fixed point, at the low SNR and waterfall regions. The indecisive fixed point, which corresponds to a decoding failure, is unstable and will disappear as SNR increases. Based on our findings, we have applied a feedback control technique to the decoder with an aim to suppressing the oscillation and chaos in the waterfall region. The technique enables the decoder to converge more likely to the stable fixed point without any prior knowledge of the transmitted signals.

1.3 Short-length LDPC codes built on scale-free networks

In recent years, complex networks have been studied across many fields of science, including computer networks, biological networks, social networks and power networks [44–47]. Synchronization and stability of different complex dynamical networks have also gained much research interest in the engineering discipline [48–52]. A recent significant discovery in the complex network theory is that some complex networks, such as the Internet and the worldwide web, have their node degrees following power-law distributions. It implies that a small number of nodes have very large numbers of connections (degrees) while a great majority of the nodes have very few connections. Such kind of networks are called *scale-free networks* [45]. Compared with regular-coupled networks, small-world networks

and random networks, scale-free networks of the same size (number of nodes) and with the same number of connections are found to accomplish the shortest average path length [53]. While for the same average path length, among the aforementioned networks, complex networks with the scale-free property have the smallest number of connections.

In this thesis, we exploit the shortest-average-path-length property of scale-free networks and apply it to the design of short-length LDPC codes. Specifically, we will propose constructing short-length LDPC codes with variable-node degrees following power-law distributions. Here, we refer such LDPC codes to as *scale-free LDPC* (SF-LDPC) codes. We will compare the achievable error performance (threshold) and the complexity (in terms of average number of node degrees) between the proposed short-length SF-LDPC codes and other best-known LDPC codes. Moreover, we will construct SF-LDPC codes of various code rates and code lengths, and simulate their error performance under an AWGN channel environment. Finally, we will compare the error rates and the average convergence time between the constructed SF-LDPC codes and some other best-known LDPC codes.

1.4 Error floor of short-length LDPC codes

1.4.1 Code construction

Regardless of the degree distributions that have been adopted in the finite-length LDPC codes, the associated bipartite graphs will inevitably comprise a variety of cycles, which undermines the independence assumption required by the BP decoder. Mao and Banihashemi have related the error performance of LDPC codes to the distribution of lengths of the smallest cycles passing through each variable node in the associated bipartite graphs [35]. The results in the literature have concluded that too many short cycles will degrade the performance of the decoder. To construct short-length LDPC codes with large girth (cycle length), a broad class of methods have been proposed [54, 55], among which the progressive edge growth (PEG) [55] is one of the most effective algorithms. Compared with random graph codes and other known good codes, codes constructed with the PEG algorithm can produce lower error rates in the waterfall regions. However, such codes may still

suffer from the problem of error floor. The existence of error floor poses a great concern for potential applications of LDPC codes such as data storage and deep-space communications, which require BER as low as 10^{-15} [56]. Thus, constructing good short-length LDPC codes with very low error floor is an important issue.

There have been various literatures looking into the error-floor issue of short-length LDPC codes [14, 57–61]. For example, Di *et al.* have thoroughly investigated the error floor of LDPC codes over a BEC channel [14]. Moreover, they have proposed a concept known as "stopping set", which defines the set of variable nodes causing the decoder to fail if all bits corresponding to the set are lost during the transmission. The stopping set is also characterized by each of its neighboring check nodes having more than one connections to the set itself. Such a feature implies that the connected subgraph induced by the stopping set consists of one or more cycles. In addition, Tian *et al.* [57] have pointed out that some stopping sets with small sizes will degrade the performance of short-length LDPC codes over an AWGN channel at high SNR. They have further discovered that short cycles with few connections to other nodes are likely contributors to small-size stopping sets. With the use of "Approximate Cycle Extrinsic message degree (ACE)" that measures the connectivity of a cycle to all other nodes, Tian *et al.* have proposed an algorithm to generate code matrices with low error floor by avoiding short cycles with ACE less than a given threshold. However, setting the ACE threshold value is not trivial. If the threshold is set too large, there are chances that no code matrix satisfying the requirements will be found. Yet if it is too small, there will be a high probability for short cycles to combine and form small-size stopping sets. In [58], a trellis-based search algorithm has been presented to detect stopping sets. Unfortunately, the method fails to identify several typical stopping sets which are critical to code performance. Furthermore, in the aforementioned references, cycle-length distribution of the code graph has not been considered, though the existence of short-length cycles has been known to degrade the code performance in the waterfall region.

To design codes with good performance at both the waterfall region and the high SNR region, studies have been carried out on a series of PEG-based methods [62, 63]. In [62], a PEG-ACE construction algorithm has been proposed by using a combined PEG-and-ACE measure. Simulation results have confirmed that for

some particular codes, the PEG-ACE constructed codes, compared with the codes constructed by the PEG-only algorithm, possess similar error performance at the waterfall region and lower error rates at the high SNR region. In [63], the authors have also tried to avoid all the small-size stopping sets consisting of variable nodes with degrees no larger than three in the PEG construction method. However, the computation complexity would have increased exponentially if variable nodes with larger degrees are to be considered in the code construction.

In a similar time frame, another concept, known as the "trapping set (TS)", has been proposed and exploited as an effective tool to evaluate the error performance of a regular LDPC code at high SNR [26, 64]. Generally speaking, a trapping set is simply a set of variable nodes. Thus, all stopping sets can be considered as trapping sets. A trapping set with the label $[w; u]$ is further defined as a set of w variable nodes with exactly u of the neighboring check nodes having odd number of connections to the (trapping) set. The value of w determines how likely that all nodes in the $[w; u]$ TS are in error. If the value of w is smaller, the scenario that all the w variables in the $[w; u]$ TS are erroneous (an "error pattern") will become more likely to occur during the LDPC decoding process. When an error pattern occurs, the amount of valid information that can flow into the $[w; u]$ TS and help rectifying the errors is affected by the value of u. In the extreme case when all the variable nodes residing outside the TS have been decoded correctly, the check equations of the u neighboring check nodes having odd number of connections to the TS will become unsatisfied[1]. Since these u neighboring check nodes are the only instrument through which the TS can collect valid information from nodes outside the TS, a small value of u will limit the amount of valid information that flows into the TS, jeopardizing the capability of the TS to correct its own errors. As a consequence, $[w; u]$ TSs with small w and u values, named as the dominant TS, may cause the decoder to oscillate and fail to converge at the high SNR region [26]. Miloš *et al.* [65] have proposed an effective algorithm to construct codes with low error floor over BSC by eliminating the TSs based on the pre-information of dominant TSs. However, the authors have not mentioned how to identify the dominant TSs.

In our study, we also observe that trapping sets are the main error contributors

[1]We define a check node as a "*satisfied* check node" if there is an even number of neighboring variable nodes decoded as "1". Otherwise, we define the check node as an "*unsatisfied* check node".

to the rare error events of irregular LDPC codes over AWGN channel at the high SNR region. However, TSs with the same label $[w; u]$ are not identical in general and contribute differently to the error floor, particularly for *irregular* LDPC codes. In other words, not all dominant TSs are harmful to the decoder. In this thesis, we will investigate the connected subgraphs induced by the TSs and categorize them into those with (i) no cycle, (ii) a single-cycle and (iii) multiple cycles. We will also study the properties of these induced connected subgraphs and show that using $[w; u]$ alone is not adequate to describe the features of the TSs, particularly features that are crucial to determine the harmfulness of the TSs. We will introduce a new parameter, namely cycle indicator and denoted by e, that further characterizes TSs. Moreover, we define "primary trapping sets (PTSs)", the main use of which is to identify detrimental TSs. Based on our findings, we then propose a code construction algorithm that aims to avoid detrimental TSs. Finally, we will perform simulations and compare the error rates of the codes constructed by our proposed method and other codes adopting the same code rate, code length and degree distributions.

1.4.2 Evaluation of the extremely low error rate

Having constructed an LDPC with a certain length, we run simulations to evaluate its block error rate (BLER) and bit error rate (BER). Using Monte Carlo (MC) simulations, we can produce BLERs as small as 10^{-6} within a day or so. However, if we are to increase the SNR further, the resultant BLER may go down to 10^{-12}, or even lower. Under such circumstances, using the MC technique to evaluate the BLER becomes not feasible due to the prohibitive amount of simulation time needed to arrive at a meaningful BLER.

To resolve the above issue, importance sampling can be applied. Importance sampling is a very powerful tool for evaluating the low probability error events in digital communication systems [66]. The basic idea of IS is to modify the probability density function (pdf) of the input random process, making the low probability events occurring more frequently. Mean translation (MT) [66] is the most efficient IS scheme. It achieves the unbiased estimation by shifting the input density to the boundary of error region. Mean translation is implemented in a divide-and-conquer manner for multidimensional systems, e.g., the coding system. In particular, the

error region is partitioned into independent sub-regions, and then the BLER is approximated by adding together the error probability estimated for each sub-region with MT.

For regular LDPC codes, it has been reported that the error region is dominated by important TSs [26], and a two-stage scheme based on MT has been proposed to help predicting the performance of LDPC codes at the high SNR region [67–74]. The first stage is to identify as many TSs as possible using a heuristic search method. In the second stage, TSs with the same label are considered equivalent under the automorphism of the graph of regular code [26] and categorized into one class. Then the error probability associated with an individual representative selected from each class is approximated. In some special cases [70, 71, 74], the authors have circumvented the need to classify the entire important TSs. However, such methods are restricted to regular LDPC codes which possess simple structures. In [75], a [w;u] absorbing set has been defined as a [w;u] TS in which each element has strictly fewer neighboring check nodes with odd degree than those with even degree. The minimal absorbing sets are the ones with the smallest possible u among those with the smallest w. Based on the properties of the absorbing set, an analytical model can be given for array-based LDPC codes, and the maximum possible number of the minimal possible absorbing sets can be exactly calculated. But as for the generally constructed regular codes and the irregular codes, we still have to resort to the heuristic search methods to identify as many absorbing sets or trapping sets as possible. Canvus *et al.* [67] have found that trapping sets consisting of overlapping short cycles appear more frequently in the LDPC decoding failures. They have proposed a graph search technique that lists all existing short cycles for each variable node in the bipartite graph. Such a method, however, would be very time-consuming in the case of medium-length or long codes.

In [73], taking a variable node of degree 3 as the root node, a tree is obtained by expanding the root node's connection in a breadth-first way to some depth. The tree, starting from the root node in the first tier, will possess alternate tiers of variable nodes and check nodes. A four-variable-node combination is then defined as the set of four variable nodes with the root variable node, and one variable node from the set of variable nodes in the third tier associated with each of three neighboring check nodes of the root. It is proved that, for regular codes with variable node

degree 3 and check node degree 6, a dominant TS must comprise a four-variable-node combination. Subsequently, error impulses are applied to the bit positions in the four-variable-node combination as the input to the decoder. During the decoding process, if the decoder fails to find a valid codeword after reaching the maximum iteration number, the $[w; u]$ TS with the smallest u will be picked by analyzing the decoding history with hard decisions. After the dominant TSs have been found, the "average Euclidean-squared distance" of a TS to the error boundary is measured to identify TSs with large BLERs. However, if the code contains tens of thousands of different TSs, the average Euclidean-squared distance of each of these TS needs to be calculated individually, which may not be feasible. Though the authors in [73] have claimed that a similar idea, by considering d_k-variable-node combinations at variable nodes of degree d_k, could be applied to the irregular codes, a prohibitive amount of computation time may be involved. Take a degree-6 variable node with neighboring check nodes of degree 8 as an example, there will be $(8-1)^6 = 117,649$ different types of seven-variable-node combinations to consider.

Yet, in all the above discussions, TSs are classified based on $[w; u]$ and only regular LDPC codes have been studied. To the best of our knowledge, no concrete suggestions nor results have been produced for evaluating *irregular* LDPC codes identified with $[w; u; e]$ TSs. Thus, we will propose evaluating irregular LDPC code performance at the high SNR region using the importance sampling (IS) [76, 77] approach in conjunction with PTSs identification. In particular, we will propose a three-stage technique to search as many detrimental PTS as possible for a given irregular LDPC code. Next, the error region of the code will be divided into various sub-regions in terms of PTS. The sub-regions subjected to PTS with the same label $[w; u; e]$ will be classified into the same group. Then, IS is applied to a representative of each group to evaluate the BLER of each group of PTS. Based on the results, the extremely low BLER of the LDPC code at high SNR is estimated.

1.5 Outline of the thesis

Chapter 2.1 gives a brief review of LDPC codes. In Section 2.1 we present two types of representations of LDPC codes. Section 2.2 elaborates the belief propagation algorithms in the probability field, likelihood ratio field and log-likelihood

ratio field. In Section 2.3 we introduce the density evolution algorithms that analyze the performance of infinite-length LDPC codes. Section 2.4 further reviews two popular methods for constructing short-length LDPC codes.

The results of our study in short-length LDPC codes will be given in Chapter 3 to Chapter 6. Chapter 3 is devoted to investigating the dynamical behavior of short-length LDPC decoder for a wide range of SNRs. In Section 3.2, we derive the Jacobian of the decoding system and calculate the eigenvalues corresponding to the fixed points. In Section 3.3, we perform extensive simulations on both regular and irregular LDPC codes. The results illustrate the rich dynamics, including fixed points, bifurcations and chaos, at the decoder as the SNR varies. Furthermore, a feedback control method is designed in Section 3.4 to assist the decoder converging at the waterfall region.

In Chapter 4, we explore the feasibility of building LDPC codes from complex networks. In Section 4.1 we briefly present the major types of complex networks. Section 4.2 explains how to build LDPC codes of infinite length when the variable-node degree distribution follows a power law. In Section 4.3, the performance of scale-free LDPC codes with finite-length are evaluated both analytically and by simulations.

In Chapter 5 and Chapter 6, we will provide a detailed investigation of the error floor in short-length LDPC codes. Chapter 5 aims at finding the cause of the error-floor problem and proposing methods to reduce the error floor. In Section 5.1, we investigate the connected subgraphs induced by the TSs and categorize them into those with (i) no cycle, (ii) a single-cycle and (iii) multiple cycles. Section 5.2 discusses the error contributions of different types of trapping sets to the error floor in terms of the configurations of their induced connected subgraphs. In Section 5.3, we define a "primary trapping set (PTS)", the main use of which is to identify the detrimental TS. Section 5.4 elaborates our proposed PEG-ACSE code-construction method that aims at avoiding detrimental TSs and hence producing short-length irregular LDPC codes with low error floor. The codes so constructed are further evaluated over an AWGN channel using the Monte Carlo simulations, and the results are presented in Section 5.5.

Chapter 6 is devoted to the evaluation of the extremely low error rate of short-length LDPC codes at the high SNR region. We will propose an importance sam-

pling technique in conjunction with PTS identification which can efficiently and effectively estimate the BLER of the codes when the BLER is very low. A brief introduction of IS technique is given in Section 6.1. In Section 6.2, we propose a three-stage search method to identify as many detrimental PTSs as possible. Section 6.3 presents the results when we apply our proposed IS scheme to several existing LDPC codes and LDPC codes constructed by our proposed methods.

Finally, in Chapter 7, we summarize our contributions of this thesis and propose some future work.

Chapter 2

Review of LDPC codes

2.1 Representations of LDPC codes

2.1.1 Graph representation

Low-density-parity-check (LDPC) codes were first proposed by [4] in the early 1960s and have been revisited by Wiberg [6] and MacKay [7] in the 1990s. Low-density-parity-check codes are in fact linear block codes [78] which can be represented by bipartite graphs consisting of two sets of nodes, namely *variable nodes* and *check nodes.* The variable nodes represent the elements of the codeword and the check nodes represent the sets of parity-check constraints satisfied by the codewords of the code. The block length of the code, denoted by N, is the number of variable nodes; while the check length of the code, denoted by M, is the number of check nodes. The connections between the two different types of nodes are called *edges*. The number of edges emanated from a node is referred to as the *degree* of the node. The key property of LDPC codes is the sparsity of the graph. In other words, the degree of each node is very low. Also, there are two kinds of LDPC codes — regular and irregular.

For regular LDPC codes, all nodes of the same type have the same degree. For irregular LDPC codes, the degree of each set of nodes is chosen according to some

distributions. For a given distribution pair (λ, ρ) of an LDPC ensemble,

$$\lambda(x) := \sum_{k=2}^{d_v} \lambda_k x^{k-1} \quad \text{and} \quad \rho(x) := \sum_{k=2}^{d_c} \rho_k x^{k-1} \tag{2.1}$$

specify the variable node degree distribution and the check node degree distribution, respectively, where λ_k denotes the fraction of edges connected to degree-k variable nodes and ρ_k denotes the fraction of edges connected to degree-k check nodes. Moreover, d_v and d_c denote the maximum variable-node degree and maximum check-node degree, respectively. Note that regular code forms a special group among the irregular ones. For regular codes with a variable-node degree d_v and a check-node d_c, all variable nodes are connected with d_v edges, i.e.,

$$\lambda_k = \begin{cases} 1 & \text{if } k = d_v \\ 0 & \text{otherwise,} \end{cases} \tag{2.2}$$

and consequently, we have $\lambda(x) = x^{d_v - 1}$. For the same reason, we have $\rho(x) = x^{d_c - 1}$.

Based on the degree distributions, the code rate of the system, denoted by R, can be obtained using

$$R = 1 - \int_0^1 \lambda(x)dx \Big/ \int_0^1 \rho(x)dx. \tag{2.3}$$

In Fig. 2.1, an example of $(10, 5)$ irregular LDPC code is shown. The $(10, 5)$ code indicates that there are 10 variable nodes (as shown on the left hand side) and 5 check nodes (as shown on the right hand side) in the bipartite graph. Each check node represents a check equation satisfied by the codeword. Referring to the figure, it can be observed that two edges emanate from v_1 and hence the degree of node v_1 equals 2. The total number of edges is 23, among which 9 edges are connected to degree-3 variable nodes and 14 to degree-2 variable nodes. So the degree distribution of the variable nodes is expressed as $\lambda(x) = \frac{14}{23}x + \frac{9}{23}x^2$. Hence the fraction of variable nodes with degree-3 is readily shown equal to $\frac{3}{10}$.

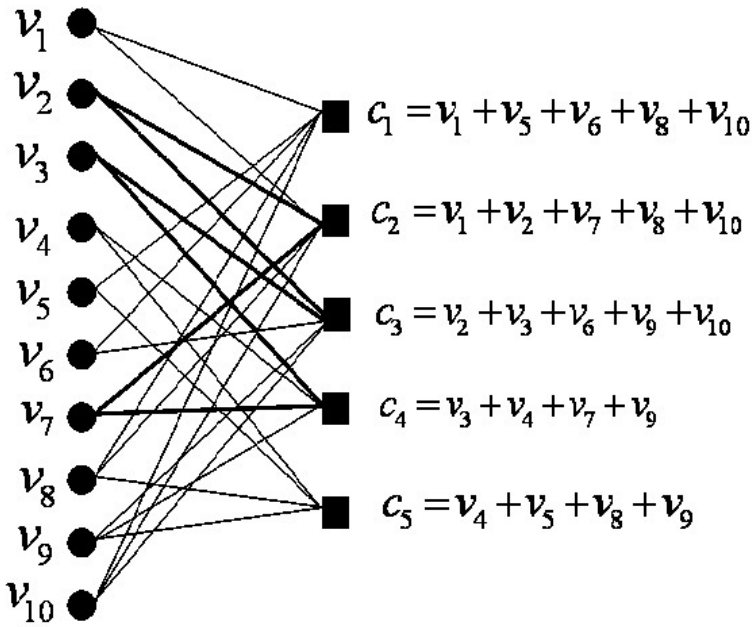


Figure 2.1: A graph representation of a $(10, 5)$ LDPC code. The path $v_1 - c_1 - v_{10} - c_2 - v_1$ forms a cycle of length 4. The variable-node set $\{v_2, v_3, v_7\}$ is a stopping set. Variable nodes are represented by filled circles and check nodes are represented by filled squares.

2.1.2 Matrix representation

Consider a linear block code with $(N-M)$ information bits and M check bits. The set of codewords Ψ can be described as a $(N-M)$-dimensional vector subspace of the space of all N-tuples over F_2^N, where F_2^N represents the binary field with N dimensions. In other words, the set of all N-bit vectors in Ψ are formed by linear combinations of $(N-M)$ linearly independent basis vectors $\{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_{(N-M)}\}$ over F_2^N, and the basis vectors can be arranged as rows of a $(N-M) \times N$ generation matrix $\mathbf{G}$ such that $\mathbf{G} = [\mathbf{g}_1^T \ \mathbf{g}_2^T \ ... \ \mathbf{g}_{(N-M)}^T]^T$ (T denotes the transpose of the vector or matrix). The null space of $\mathbf{G}$ is associated with a matrix $\mathbf{H}$, which is called the parity-check matrix. The (j, i)th element of the parity-check matrix $\mathbf{H}$, denoted by h_{ji}, is 1 if and only if the jth check node is connected to the ith variable node. As a consequence, the LDPC code can be defined as the set of vectors Ψ such that all elements $\psi \in \Psi$ satisfy $\psi \mathbf{H}^T = \mathbf{0}$. In the bipartite graph, such as the one shown in Fig. 2.1, each edge will correspond to a "1" in a particular position in the parity-check matrix. Therefore, based on the bipartite graph, the corresponding matrix

representation of the LDPC code can be derived. For example, it is readily shown that the parity-check matrix of the $(10, 5)$ code in Fig. 2.1 is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}. \tag{2.4}$$

2.2 Decoder of LDPC codes

2.2.1 Overview

Low-density-parity-check codes can be decoded with various decoding methods, including hard-decision decoding and soft-decision decoding [79, 80]. Hard decision methods such as bit flipping [81, 82] and one-step majority-logic [83] are simple for hardware implementation but are not good in error performance compared to soft-decision decoding methods. On the other hand, the maximum likelihood decoder [84] provides the lowest error rates among the various decoding algorithms, but is too complex to be implemented in practice. A widely used algorithm that represents a good compromise between complexity and error performance is the *iterative decoding* based on belief propagation (BP) [79].

The BP algorithm is one of the message-passing algorithms, which allow the exchange of information between the variable nodes and the check nodes in the graph-based model during each iteration. The message from a variable node v to a check node c is calculated based on the received messages from both the channel and the neighboring check nodes except c. Similarly, the message from a check node c to a variable node v is computed based on the incoming messages from the neighboring variable nodes except v. Without loss of generality, we consider a binary-input AWGN channel and a transmitted codeword with a block length N.

Also, we represent the number of check nodes by M. We denote the ith code bit ($i = 1, 2, ..., N$) by $\psi_i \in \{0, 1\}$. The transmitted signal corresponding to this code bit equals $(-1)^{\psi_i+1}$. The received signal, denoted by y_i, is given by $y_i = (-1)^{\psi_i+1} + z_i$, where the variables z_i are independent and identically distributed zero-mean Gaussian random variables with variance (noise power) σ^2.

Given the received signal $\mathbf{y} = [y_1, y_2, ..., y_N]$. To decode the code bit ψ_i, we can make use of either

1. a posteriori probabilities (APPs), denoted by $\Pr(\psi_i = 0|\mathbf{y})$ and $\Pr(\psi_i = 1|\mathbf{y})$ in the probability field; or
2. the APP ratio, denoted by $\frac{\Pr(\psi_i=0|\mathbf{y})}{\Pr(\psi_i=1|\mathbf{y})}$, in the likelihood ratio (LR) field; or
3. the log-APP ratio, denoted by $\log(\frac{\Pr(\psi_i=0|\mathbf{y})}{\Pr(\psi_i=1|\mathbf{y})})$, in the log-likelihood ratio (LLR) field

where $\Pr(E)$ denotes the probability that the event E occurs.

The computation of APP, APP ratio or log-APP ratio in the iterative algorithm is based on the Tanner graph of the code. For the l-th iteration ($l = 0, 1, ...$), we denote the message along the edge from check node c_j to variable node v_i as $m \uparrow_{ij}^{(l)}$ and the message along the edge from variable node v_i to check node c_j as $m \downarrow_{ij}^{(l)}$. The messages can be the APP, APP ratio or log-APP ratio. Consider the variable node v_i in Fig. 2.2 (a). The outgoing message $m \downarrow_{i0}^{(l)}$ contains the conditional probability of code bit ψ_i being b ($b = 0, 1$), which is given by

$$\Pr(\psi_i = b|y_i, m \uparrow_{i1}^{(l)}, m \uparrow_{i2}^{(l)}, ..., m \uparrow_{i(d_v-1)}^{(l)}). \tag{2.5}$$

Assume that all the incoming messages are independent. Thus the above conditional probability equals

$$\Pr(\psi_i = b|y_i, m \uparrow_{i1}^{(l)}, m \uparrow_{i2}^{(l)}, ..., m \uparrow_{i(d_v-1)}^{(l)}) = \Pr(\psi_i = b|y_i) \prod_{k=1}^{d_v-1} \Pr(\psi_i|m \uparrow_{ik}^{(l)}) \tag{2.6}$$

Furthermore, consider the check node c_j in Fig. 2.2 (b). Denote $\mathscr{P}_j$ as the check equation at check node c_j. The outgoing message $m \uparrow_{0j}^{(l)}$ contains the probability of

check equation at c_j being satisfied, which is given by

$$\Pr(\mathscr{P}_j = 0 | m \downarrow_{1j}^{(l)}, m \downarrow_{2j}^{(l)}, ..., m \downarrow_{(d_c-1)j}^{(l)}). \tag{2.7}$$

With the assumption that all the incoming messages are independent, the conditional probability can be rewritten as

$$\Pr(\mathscr{P}_j = 0 | m \downarrow_{1j}^{(l)}, m \downarrow_{2j}^{(l)}, ..., m \downarrow_{(d_c-1)j}^{(l)}) = \prod_{k=1}^{d_c-1} \Pr(\mathscr{P}_j = 0 | m \downarrow_{kj}^{(l)}). \tag{2.8}$$

In the following, we will elaborate the BP algorithm in the probability field, likelihood ratio field and log-likelihood ratio field, respectively.

2.2.2 Belief propagation in the probability field

For $b = 0, 1$, we define $q_{ij}^{(l)}(b)$ (equivalent to $m \downarrow_{ij}^{(l)}$ in Sect. 2.2.1) as the conditional posterior probability of the bit ψ_i being equal to b at iteration l; and $r_{ji}^{(l)}(b)$ (equivalent to $m \uparrow_{ij}^{(l)}$ in Sect. 2.2.1) as the conditional posterior probability of the event that the jth check equation being satisfied at iteration l for $l = 0, 1, ..., I_{\max}$, where $I_{\max}$ is the maximum iteration number. We further assume that the outgoing message $q_{ij}^{(l)}(b)$ from the variable node v_i is calculated based on messages passed from the neighboring check-node set C_i excluding the check node c_j at iteration l; and the outgoing message $r_{ji}^{(l)}(b)$ from check node c_j is calculated based on messages passed from the neighboring variable-node set V_j excluding the variable node v_i at iteration l. Assuming that the passed messages in the iterative process are independent random variables, the BP algorithm in the probability domain proceeds as follows.

1. Estimate the noise power σ^2. For $i = 1, 2, ..., N$ and $b = 0, 1$, initialize $P_i(b) := \Pr(\psi_i = b | y_i)$, where $\Pr(\psi_i = b | y_i)$ denotes the posterior probability that bit ψ_i equals b given the received signal y_i. Set $q_{ij}^{(0)}(b) = P_i(b)$ if the i-th variable node v_i and the j-th check node c_j are connected.

2. Update $\{r_{ji}^{(l)}(b)$: $i = 1, 2, ..., N$; $j = 1, 2, ..., M$; $l = 0, 1, ..., I_{\max}$ and $b =$

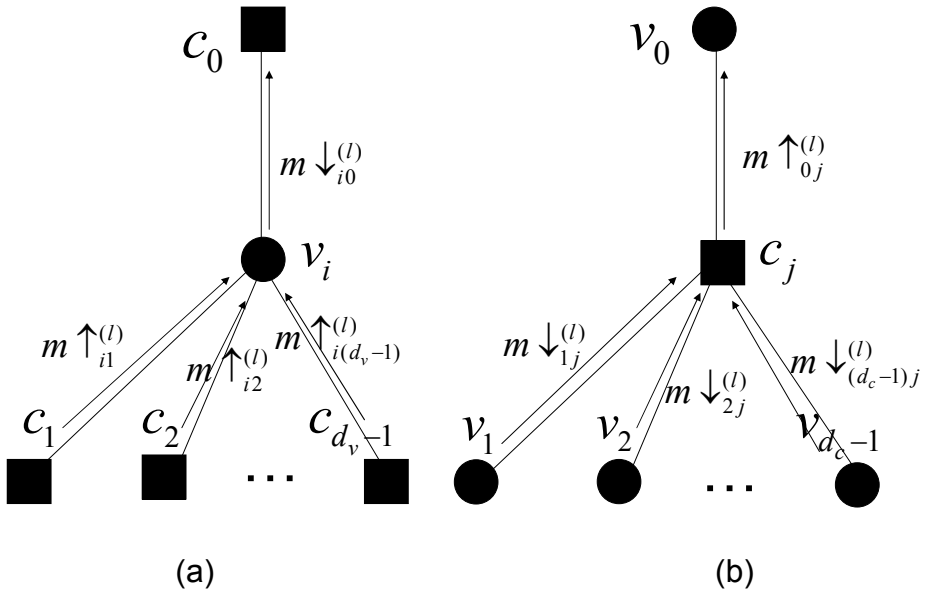


Figure 2.2: The decoding mechanism at a specific variable node and a specific check node.

$0, 1\}$ using

$$\begin{cases} r_{ji}^{(l)}(0) = \frac{1}{2} + \frac{1}{2} \prod\limits_{i' \in V_j/i} (1 - 2q_{i'j}^{(l)}(1)) \\ r_{ji}^{(l)}(1) = 1 - r_{ji}^{(l)}(0). \end{cases} \tag{2.9}$$

3. Update $\{q_{ij}^{(l+1)}(b)$: $i = 1, 2, ..., N$; $j = 1, 2, ..., M$; $l = 0, 1, ..., I_{\max}$ and $b = 0, 1\}$ using

$$q_{ij}^{(l+1)}(b) = A_{ij}^{(l)} P_i(b) \prod_{j' \in C_i/j} r_{j'i}^{(l)}(b) \tag{2.10}$$

where $A_{ij}^{(l)}$ is chosen to ensure that $q_{ij}^{(l+1)}(0) + q_{ij}^{(l+1)}(1) = 1$.

4. Compute the posteriori probability of bit $\psi_i = b$ for $b = 0, 1$ using

$$Q_i^{(l)}(b) = A_i^{(l)} P_i(b) \prod_{j \in C_i} r_{ji}^{(l)}(b) \tag{2.11}$$

where $A_i^{(l)}$ is chosen to ensure that $Q_i^{(l)}(0) + Q_i^{(l)}(1) = 1$.

5. For $i = 1, 2, ..., N$, set

$$Q_i^{(l)}(1) \underset{\hat{\psi}_i=0}{\overset{\hat{\psi}_i=1}{\gtrless}} Q_i^{(l)}(0). \tag{2.12}$$

where $\hat{\psi}_i$ is the i-th estimated codeword bit by hard decision. If $\hat{\psi}\mathbf{H}^T = \mathbf{0}$ or the number of iterations equals the maximum limit, stop; else, go to Step 2.

2.2.3 Belief propagation in the likelihood ratio field

Define lP_i as the channel message in the likelihood ratio (LR) field at i-th variable node v_i corresponding to received signal y_i. Define $lr_{ji}^{(l)}$ (equivalent to $m \uparrow_{ij}^{(l)}$ in Sect. 2.2.1) as the message in LR field passed from check node c_j to variable node v_i at iteration l and $lq_{ij}^{(l)}$ (equivalent to $m \downarrow_{ij}^{(l)}$ in Sect. 2.2.1) as the message in the LR field passed from variable node v_i to check node c_j at iteration l. Then the iterative equations of the decoder are given as follows [79].

1. Estimate the noise power σ^2. Then for $i = 1, 2, \ldots, N$, initialize $lP_i = \frac{\Pr(\psi_i=1|y_i)}{\Pr(\psi_i=0|y_i)}$. Set $lq_{ij}^{(0)} = lP_i$ if the i-th variable node v_i and the j-th check node c_j are connected.

2. Update $\{lr_{ji}^{(l)} : i = 1, 2, ..., N; j = 1, 2, ..., M; l = 0, 1, ..., I_{\max}\}$ using

$$lr_{ji}^{(l)} = \left(1 - \prod_{i' \in V_j \backslash i} \frac{1 - lq_{i'j}^{(l)}}{1 + lq_{i'j}^{(l)}}\right) \Big/ \left(1 + \prod_{i' \in V_j \backslash i} \frac{1 - lq_{i'j}^{(l)}}{1 + lq_{i'j}^{(l)}}\right). \tag{2.13}$$

3. Update $\{lq_{ij}^{(l+1)} : i = 1, 2, ..., N; j = 1, 2, ..., M; l = 0, 1, ..., I_{\max}\}$ using

$$lq_{ij}^{(l+1)} = lP_i \prod_{j' \in C_i \backslash j} lr_{j'i}^{(l)}. \tag{2.14}$$

4. Compute the LR value of the code bit ψ_i using

$$lQ_i^{(l)} = lP_i \prod_{j \in C_i} lr_{ji}^{(l)}. \tag{2.15}$$

5. For $i = 1, 2, ..., N$, set

$$lQ_i^{(l)} \underset{\hat{\psi}_i=0}{\overset{\hat{\psi}_i=1}{\gtrless}} 1. \tag{2.16}$$

If $\hat{\psi}\mathbf{H}^T = \mathbf{0}$ or the number of iterations equals the maximum limit, stop; else, go to Step 2.

2.2.4 Belief propagation in the log-likelihood ratio field

Define LP_i as the channel message in the log-likelihood ratio (LLR) field at variable node v_i corresponding to received signal y_i. Define $Lq_{ij}^{(l)}$ (equivalent to $m \downarrow_{ij}^{(l)}$ in Sect. 2.2.1) as the conditional LLR at iteration l computed based on (i) the received LLR information LP_i, and (ii) the message $Lr_{j'i}^{(l-1)}$ passed from the neighboring check-node set C_i excluding the check node c_j. Also, we define $Lr_{ji}^{(l)}$ (equivalent to $m \uparrow_{ij}^{(l)}$ in Sect. 2.2.1) as the conditional LLR at iteration l computed based on the message $Lq_{ij}^{(l)}$ passed from the neighboring variable-node set V_j excluding the variable node v_i. The message-passing algorithm then proceeds as follows.

1. Estimate the noise power σ^2. Then for $i = 1, 2, \ldots, N$, initialize $LP_i = \log\left(\frac{\Pr(\psi_i=1|y_i)}{\Pr(\psi_i=0|y_i)}\right)$. Set $Lq_{ij}^{(0)} = LP_i$ if the i-th variable node v_i and the j-th check node c_j are connected.

2. Update $\{Lr_{ji}^{(l)} : i = 1, 2, ..., N; j = 1, 2, ..., M; l = 0, 1, ..., I_{\max}\}$ using

$$Lr_{ji}^{(l)} = \left[\prod_{i' \in V_j/i} \operatorname{sign}(Lq_{i'j}^{(l)})\right] \times \phi\left(\sum_{i' \in V_j/i} \phi(Lq_{i'j}^{(l)})\right) \tag{2.17}$$

where $\phi(x) = -\log(\tanh(|x/2|))$ and the unconventional probabilistic defi-

nition of the sign function $\mathrm{sign}(x)$ is given by

$$\mathrm{sign}(x) = \begin{cases} 0 & \text{for } x > 0 \\ 0 & \text{with probability } 1/2 \text{ for } x = 0 \\ 1 & \text{with probability } 1/2 \text{ for } x = 0 \\ 1 & \text{for } x < 0. \end{cases} \tag{2.18}$$

3. Update $\{Lq_{ij}^{(l+1)} : i = 1, 2, ..., N; j = 1, 2, ..., M; l = 0, 1, ..., I_{\max}\}$ using

$$Lq_{ij}^{(l+1)} = LP_i + \sum_{j' \in C_i/j} Lr_{j'i}^{(l)}. \tag{2.19}$$

4. Compute the LLR value of the code bit ψ_i using

$$LQ_i^{(l)} = LP_i + \sum_{j \in C_i} Lr_{ji}^{(l)} \tag{2.20}$$

5. For $i = 1, 2, ..., N$, set

$$LQ_i^{(l)} \underset{\hat{\psi}_i=0}{\overset{\hat{\psi}_i=1}{\gtrless}} 0. \tag{2.21}$$

If $\hat{\psi}\mathbf{H}^T = \mathbf{0}$ or the number of iterations equals the maximum limit, stop; else, go to Step 2.

2.2.5 Summary

In all, the BP algorithm in the probability field is more readily understood from a physical point of view. The BP algorithm in the LLR field, on the other hand, converts all multiplication/division operations into addition/subtraction operations with the use of the log function, which can be realized by using a look-up table and hence more easily implemented by hardware. However, it takes a comparatively longer time to execute the log operations than the multiplication/division operations when we run the decoding algorithm on a computer. Thus, in the computer simulations, the BP algorithm in the LR field is preferred if the simulation time is a concern.

2.3 Performance analysis of infinite-length LDPC codes

Suppose the path between a variable node and the immediate neighboring check node contributes to a unit depth. Define the neighborhood of a variable (check) node v (c) to a depth d as the induced subgraph by traversing the variable (check) node v (c) along its connections to a depth d. The flow of the iterative BP decoding algorithm for the l-th iteration can be visualized on the neighborhood of each variable node to a depth $2l$ ($l = 0, 1, ...$), as shown in Fig. 2.3. This tree-like topology consists of $2l+1$ layers having alternate layers of variable nodes and check nodes. Assuming the root variable node is lying on the first layer, the $(2l' - 1)$-th layer and the $2l'$-th layer has alternatively variable nodes and check nodes for $l' = 1, 2, ..., l$. The "independence" assumption among the passing-messages implies that the neighborhood of each variable node to a depth $2l$ is cycle-free. As l increases, the assumption can only be met with code length approaching infinity. When the assumption is satisfied, however, density evolution can be utilized to track the distributions of the output messages at the variable nodes and the check nodes.

2.3.1 Density evolution

Suppose the belief propagation (BP) algorithm with regard to the log-likelihood ratio (LLR) is used as the iterative decoder [79]. Denote the density associated with the messages from the variables nodes to the check nodes during the l-th iteration by $\tilde{P}_l$ and that from the check nodes to the variable nodes by $\tilde{Q}_l$. Moreover, over the AWGN channel, the initial message density has been shown equal to [11]

$$\tilde{P}_0(y_i) = \frac{1}{\sqrt{8\pi}\sigma} \exp\left[\frac{-(y_i - \frac{2}{\sigma^2})^2\sigma^2}{8}\right] \tag{2.22}$$

where $i = 1, ..., N$.

Assuming that all the messages at each node are independent of one another, it has been shown that for $l = 1, 2, ...$ [11, 15]

$$\tilde{P}_l = \tilde{P}_0 \otimes \lambda(\tilde{Q}_l) \tag{2.23}$$

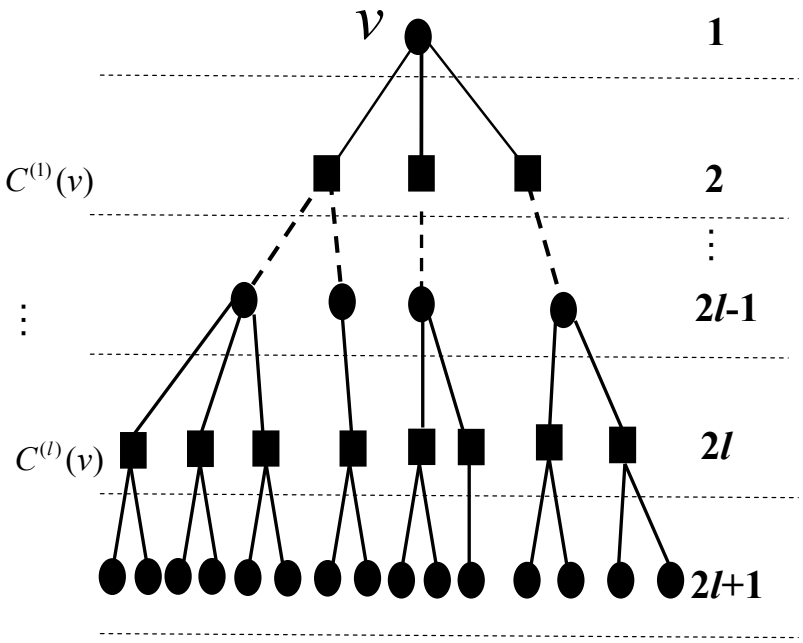


Figure 2.3: The neighborhood of a variable node to a depth $2l$. The tree-like topology of $2l+1$ layers, starting from the root variable node in the first layer, comprises variable nodes and check nodes in layer $2l'-1$ and layer $2l'$, respectively, for $l'=1,...,l$. $C^{(l)}(v)$ represents the check-node set at the $2l$-th layer of the neighborhood of the variable node v.

where $\otimes$ represents convolution and $\lambda(.)$ denotes the variable-node degree distribution, as defined in (2.1). To compute $\tilde{Q}_l$, we represent the messages $\{Lq_{ij}^{(l)}\}$ in an alternative way. We define the map $\beta : [-\infty,+\infty] \to \mathrm{GF}(2) \times [0,+\infty]$. Given a random variable $a \in [-\infty,+\infty]$ with distribution F_a and $a \neq 0$. Let

$$\beta(a) := (\beta_1(a),\beta_2(a)) := (\mathrm{sign}(a), -\ln(\tanh|a/2|))\,. \tag{2.24}$$

We also define the "distribution" of $\beta(a)$ as

$$\Gamma(F_a)(\varsigma,x) = \delta_{\{\varsigma=0\}}\Gamma_0(F_a)(x) + \delta_{\{\varsigma=1\}}\Gamma_1(F_a)(x) \tag{2.25}$$

where

$$\Gamma_0(F_a)(x) \quad = \quad \Pr(\beta_1(a)=0, \beta_2(a)\leq x)$$

$$= \Pr(a \geq -\ln\tanh(x/2)) \tag{2.26}$$

$$\Gamma_1(F_a)(x) = \Pr(\beta_1(a) = 1, \beta_2(a) \leq x)$$

$$= \Pr(a \leq \ln\tanh(x/2)) \tag{2.27}$$

and $\delta_{\{\varsigma=b\}}$ equals 1 if $\varsigma = b$ for $b = 0, 1$; and equals 0 otherwise. Thus, (2.17) can be written as

$$Lr_{ji}^{(I)} = \beta^{-1}\left(\sum_{i' \in V_j \setminus i} \beta\left(\frac{Lq_{i'j}^{(l)}}{2}\right)\right) \tag{2.28}$$

and the density of $Lr_{ji}^{(l)}$ is given by

$$\tilde{Q}_l = \Gamma^{-1}(\rho(\Gamma(\tilde{P}_{l-1}))), \tag{2.29}$$

where $\rho(.)$ denotes the check-node degree distribution, as defined in (2.1). Finally, $\tilde{P}_l$ can be expressed as

$$\tilde{P}_l = \tilde{P}_0 \otimes \lambda(\Gamma^{-1}(\rho(\Gamma(\tilde{P}_{l-1})))). \tag{2.30}$$

Having found the density $\tilde{P}_l$, the associated distribution can be computed using integration. For the symmetric linear block code and output-symmetric memoryless AWGN channel, we assume that all-zero codewords are used in the transmissions [15]. Thus, the iterative process can determine whether the expected fraction of incorrect messages, i.e., those with non-negative values with regard to decoding in the LLR field, will go to zero as iteration continues. For fixed d_v and d_c, one can then find the largest value of σ, i.e., best achievable performance or threshold of the LDPC code, by varying the distribution pair (λ, ρ) such that the expected fraction of incorrect messages will go to zero [15]. Optimization of the threshold has been carried out and the degree distributions of some good codes have already been found [85]. However, such codes provide optimal error performance only under an infinite code length and an infinite number of iterations for decoding.

2.3.2 Discrete density evolution

In the previous section, we have shown that density evolution (DE) together with belief propagation algorithm can serve as a powerful tool for finding degree

distributions that can maximize the threshold σ. However, the computation involved in DE is fairly intensive. Other techniques that are comparatively easier to visualize and less computational intensive than DE include "extrinsic information transfer (EXIT)" chart [86–88] and "Gaussian approximation density evolution (GA-DE)" [27]. These methods give suboptimal performance compared with DE though. Also, they assume that as the number of iterations increases, the probability density functions of the messages from variable nodes to check nodes are approaching Gaussian-like distributions. In order to achieve low computational complexity and to maintain the performance of DE, a different implementation of DE based on the quantization technique and fast Fourier transforms (FFT), also known as "discrete DE (DDE)" has been proposed [20] and is briefly reviewed as follows.

Let $\mathcal{Q}(m)$ be the quantized message of m, i.e.,

$$\mathcal{Q}(m) = \begin{cases} \lfloor \frac{m}{\Delta} + \frac{1}{2} \rfloor & \text{if } m \geq \frac{\Delta}{2} \\ \lceil \frac{m}{\Delta} - \frac{1}{2} \rceil & \text{if } m < -\frac{\Delta}{2} \\ 0 & \text{otherwise} \end{cases} \tag{2.31}$$

where $\mathcal{Q}$ is the quantization operator; Δ is the quantization interval; $\lfloor x \rfloor$ is the largest integer not greater than x; and $\lceil x \rceil$ is the smallest integer not less than x. In this regard, we denote by $\hat{P}_l(k)$ as the probability that the messages from variable nodes to check nodes being $k\Delta$ during the lth iteration; and we represent $\hat{Q}_l(k)$ as the probability that the messages from check nodes to variable nodes being $k\Delta$ during the lth iteration. Then the probability associated with the initial messages being $k\Delta$ is given by

$$\hat{P}_0(k) = \mathcal{Q}\left(\frac{1}{\sqrt{8\pi}\sigma} \exp\left[\frac{-(k\Delta - \frac{2}{\sigma^2})^2 \sigma^2}{8} \right] \right). \tag{2.32}$$

For the calculation of $\hat{Q}_l$, instead of involving the complicated convolution over a different domain $\mathrm{GF}(2) \times [0, +\infty]$ (see (2.24) to (2.29)), the new DE implementation employs a look-up table for the direct convolution of the discrete log-likelihood ratio probabilities.

First, define a two-input operator $\mathcal{R}$ as

$$\mathcal{R}(\hat{m}_1, \hat{m}_2) = \mathcal{Q}\left(2\tanh^{-1}\left(\tanh\frac{\hat{m}_1\Delta}{2}\tanh\frac{\hat{m}_2\Delta}{2}\right)\right) \tag{2.33}$$

where $\hat{m}_1$ and $\hat{m}_2$ are two quantized messages. Obviously, this operation can be accomplished using a pre-computed table. Let $\hat{m}_3 = \mathcal{R}(\hat{m}_1, \hat{m}_2)$ and denote the probability of $\hat{m}_n$ by $p_{\hat{m}_n}[k_n], n = 1, 2, 3$. Then

$$p_{\hat{m}_3}[k_3] = \sum_{(k_1,k_2):k_3\Delta=\mathcal{R}(k_1,k_2)} (p_{\hat{m}_1}[k_1]p_{\hat{m}_2}[k_2]) \tag{2.34}$$

where k_1, k_2 and k_3 are integer variables. Using $*^{\mathcal{R}}$ to denote the operator

$$(f *^{\mathcal{R}} g)[k_3] = \sum_{(k_1,k_2):k_3\Delta=\mathcal{R}(k_1,k_2)} (f[k_1]g[k_2]), \tag{2.35}$$

(2.34) can be re-written as

$$p_{\hat{m}_3} = p_{\hat{m}_1} *^{\mathcal{R}} p_{\hat{m}_2}. \tag{2.36}$$

Thus,

$$\hat{Q}_l = \sum_{j=2}^{j=d_c} \rho_j(\underbrace{\hat{P}_l *^{\mathcal{R}} (\hat{P}_l, ..., *^{\mathcal{R}}(\hat{P}_l *^{\mathcal{R}} \hat{P}_l))}_{j\ \hat{P}_l\text{s}}). \tag{2.37}$$

Furthermore, with reference to (2.23), the probability $\hat{P}_l(k)$ $(l = 1, 2, ...)$ can be written as

$$\hat{P}_l = \hat{P}_0 * \lambda(\hat{Q}_l) \tag{2.38}$$

where $\hat{P}_l = \{\hat{P}_l(k)\}$, $\hat{Q}_l = \{\hat{Q}_l(k)\}$ and $*$ denotes the discrete convolution. The calculation can be readily accomplished using FFT together with point-wise multiplication of the Fourier transforms of the probability distributions.

Without loss of generality, we send all-zero codewords over the channel while fixing the channel parameter at σ. Then, the probability of error is given by the probability of $\hat{Q}_l$ being negative. Afterward, the algorithm runs until the probability of error tends to zero or converges to a positive fixed point. Finally, the threshold σ^* is defined as the maximum channel parameter such that the probability of error tends to zero as the number of iterations tends to infinity.

2.3.3 Remark

Consider a binary-input AWGN channel and an LDPC code with rate $1/2$. Using the "discrete density evolution (DDE)", the threshold of the best code has been found to lie within 0.0045 dB of the Shannon limit [20]. Further, a bit error rate of 10^{-6} has been achieved within 0.04 dB of the Shannon limit using a block length of 10^7 [20]. In [89], the authors have proposed a "fast density evolution (FDE)" method. It exploits fully the symmetry of the BP algorithm and claims to give much more accurate results with much less computational complexity than DDE and other methods. Yet, when we develop a computer program for the FDE algorithm, the program is found failing to optimize the degree distributions of irregular codes. Such a failure has also been reported by other researchers [90]. Thus, in this thesis, we will make use of the DDE to design LDPC codes with large thresholds.

2.4 Construction of short-length LDPC codes

In the previous section, we have reviewed several mechanisms that can serve to evaluate the achievable error performance of infinite-length LDPC codes. The achievable performance of an LDPC code with finite length, moreover, will approach that of with infinite length asymptotically as the code length increases. Nevertheless, it is not very practical to implement such long codes in many applications due to its hardware complexity as well as the incurred time-delay problems [21]. In reality, short-length (less than several thousands) LDPC codes will find a lot more applications, but they may show very poor error performance compared with the infinite-length codes. In the following, we present two methods for constructing short-length LDPC codes with good error performance.

2.4.1 Progressive edge growth

Definition 2.1 *(Cycle) A cycle in a bipartite graph is a path, consisting of edges connecting the two sets of nodes, that originates from and terminates at the same node. Moreover, each edge can only be used once in the cycle and the length of the cycle is given by the number of edges making up the cycle. For example, the path*

$v_1 - c_1 - v_{10} - c_2 - v_1$ *in Fig. 2.1 forms a cycle of length* 4.

Define the "girth" of a variable node as the length of the smallest cycle in the bipartite graph that is originated from the variable node. The "girth average" is then the mean of the girths over all the variable nodes. In [35], the authors have related the error performance of LDPC codes to the girth average in the associated bipartite graph. The results have concluded that short-length codes with good error-correcting performance usually have a large girth average. To construct short-length LDPC codes with large girth, a broad class of methods have been proposed [35, 55]. Among the methods, progressive edge growth (PEG) [55] is one of the most effective algorithms to enlarge the girth as well as the hamming weight of the codes.

Given that there are N variable nodes and M check nodes. Moreover, the degree of the i-th variable node is denoted by d_i. Referring to Fig. 2.3, we define $C^{(l)}(v_i)$ as the check-node set at the $2l$-th layer of the neighborhood of the variable node v_i. Then, the PEG code-construction algorithm can be described as follows.

- Sort the variable nodes such that $d_i \leq d_j$ for $i \leq j$.
- For $i = 1$ to N
  - For $k = 1$ to d_i
    - If $k == 1$
      - Connect the first edge of v_i randomly with a check node, which has the lowest check-node degree under the current graph setting.
    - Else
      - Expand the neighborhood of v_i up to the depth $2l - 1$ under the current graph setting such that the check-node set at the $2l$-th layer is not empty but the check-node set at the $(2l+2)$-th layer is empty, i.e., $C^{(l)}(v_i) \neq \emptyset$ and $C^{(l+1)}(v_i) = \emptyset$.
      - Connect the kth edge randomly with a check node in the set $C^{(l)}(v_i)$ that has the lowest degree under the current setting.
    - End (if...else)
  - End ($k = 1$ to d_i)

- End ($i = 1$ to N)

Note that sorting the variable nodes in a non-decreasing order can prevent low-degree variable nodes from involving in the short cycles when the PEG algorithm is completed. It is particularly crucial for degree-2 nodes because they do not give rise to any extrinsic connections in cycles.

2.4.2 ACE-based code construction

While short cycles are found to be harmful to the code performance, not all short cycles are equally harmful. To explain the scenario, we resort to the concept of stopping set which has been first introduced when studying LDPC codes under a BEC channel [14].

Definition 2.2 *(Stopping set) A stopping set $\mathcal{S}$ of an LDPC code is a subset of variable nodes with no singly-connected check nodes attached to the subset. The size of a stopping set represents the number of variable nodes contained in the set. For example, the variable-node set $\{v_2, v_3, v_7\}$ in Fig. 2.1 is a stopping set.*

Under a BEC channel, if the bits corresponding to the variable nodes in a stopping set are erased, they can never be recovered regardless of the number of the iterations being conducted. It has also been pointed out that signals (corresponding to variable nodes) which have very poor observation reliability over an AWGN channel are analogous to the erasures over a BEC channel. Thus stopping sets may also degrade the performance of the BP decoder over the binary-input AWGN channel [57]. In particular, consider a stopping set with all the neighboring check nodes having an even number of connections. With an all-zero codeword transmitted, if the variable nodes in a stopping set have been decoded as "1" and all other variable nodes have been decoded as "0", the decoder will have converged to a non-all-zero codeword because all check equations are now satisfied. Though the non-all-zero codeword is a valid one, it is not the transmitted one (all-zero codeword). An error will therefore occur. Since the error floor of LDPC codes using a BP decoder depends on the minimum weight codewords as well as their multiplicities [26], increasing the size of the minimum stopping set in a code is beneficial to the error performance.

Moreover, consider the subgraph from by a stopping set and its neighboring check nodes. Since the check nodes are not singly-connected to the stopping set, each of the check nodes must have a minimum degree of 2 in the subgraph. In consequence, the subgraph should comprise one or more cycles.

Suppose we want to find out all stopping sets of size w that involves a particular variable node. Starting from that particular variable node, we then construct the neighborhood of the node to a depth such that the variable nodes in all possible stopping sets of size w have to be included. To satisfy the requirement, a depth of w has to be used. Hence, the neighborhood will consist of $w+1$ layers. The first layer contains the particular variable node being considered. Then the second layer will contain at most d_v check nodes (recall that d_v represents the maximum degree of the variable nodes) and consequently the third layer will have at most $d_v \times (d_c - 1)$ variable nodes (recall that d_c represents the maximum degree of the check nodes). Following this manner, in the $2l$-th layer and the $(2l+1)$-th layer, there will be at most, respectively, $d_v(d_v-1)^{l-1}(d_c-1)^{l-1}$ check nodes and $d_v(d_v-1)^{l}(d_c-1)^{l-1}$ variable nodes. Therefore, consider the $(w+1)$-th layer when w is odd and even respectively. If w is odd, the $w+1$ layer will have at most $d_v(d_v-1)^{(w+1)/2-1}(d_c-1)^{(w+1)/2-1}$ check nodes. If w is even, however, the $w+1$ layer will consist of at most $d_v(d_v-1)^{w/2}(d_c-1)^{w/2-1}$ variable nodes. In consequence, summing up the number of variable nodes at different layers, any stopping set with size w and involving the particular variable node must be a subset of the variable-node set with size

$$|\mathcal{T}| = \begin{cases} \min\left(N, 1 + d_v(d_c-1)\frac{1-((d_v-1)(d_c-1))^{w/2}}{1-(d_v-1)(d_c-1)}\right) & \text{if } w \text{ is even} \\ \min\left(N, 1 + d_v(d_c-1)\frac{1-((d_v-1)(d_c-1))^{(w-1)/2}}{1-(d_v-1)(d_c-1)}\right) & \text{if } w \text{ is odd.} \end{cases} \tag{2.39}$$

Note that as w gets large, $|\mathcal{T}| \to N$. Moreover, the number of possible combinations for the trapping set equals $\binom{w}{|\mathcal{T}|}$, where

$$\binom{w}{|\mathcal{T}|} = \frac{|\mathcal{T}|!}{(|\mathcal{T}|-w)!w!} \tag{2.40}$$

In general, to find out all the stopping sets with size no more than w in a given LDPC code, the number of possible combinations becomes $N\sum_{l=1}^{w}\binom{l}{|\mathcal{T}|}$. Even for $N = 1000, d_v = 6, d_c = 3$ and a small w, say $w = 5$, there are more than

10^{15} combinations, which is an enormous number. Instead of detecting all possible stopping sets with small w directly, Tian *et al.* search likely contributors to the small-size stopping sets based on the use of "approximate cycle extrinsic message degree (ACE)" [57] .

Definition 2.3 *(Extrinsic message degree (EMD)) An extrinsic check node of a variable node set is a check node that is singly-connected to this set. The EMD of a variable-node set is the number of extrinsic check nodes of this variable-node set.*

Definition 2.4 *(Approximate cycle EMD (ACE)) The ACE of a cycle with length* $2l$ *is* $\sum_{i=1}^{i=l}(d_i - 2)$*, where* d_i *is the degree of the* i*-th variable node in this cycle. We also say that the ACE of a degree-*d_i *variable node is* $(d_i - 2)$ *and the ACE of any check node is* 0*.*

It has been found in [57] that short cycles with small ACE are likely contributors to the small size stopping sets. Subsequently, the authors of [57] have proposed an algorithm to construct short-length LDPC codes with low error floor by avoiding short cycles with small ACE. In the following, we briefly describe the algorithm.

- Sort the variable nodes such that $d_i \leq d_k$ for $i \leq k$.
- Initialize $\mathbf{H}$ to be an empty matrix.
- For $i = 1$ to N
  - Randomly connect the variable node v_i to d_i check nodes. Record the connections by the vector $\mathbf{h}_i = [h_{1i}\ h_{2i}\ \ldots\ h_{Mi}]^T$, where h_{ji} is 1 if the variable node v_i is connected to the check node c_j, and is 0 otherwise.
  - If $(i \leq M)$ (i.e., v_i is a parity bit)
    - Perform Gaussian elimination (GE) on $\mathbf{H}$.
    - While $\mathbf{h}_i$ is linearly dependent on the space spanned by the columns of $\mathbf{H}$)
      - Randomly connect the variable node v_i to d_i check nodes. Record the connections by the vector $\mathbf{h}_i = [h_{1i}\ h_{2i}\ \ldots\ h_{Mi}]^T$, where h_{ji} is 1 if the variable node v_i is connected to the check node c_j, and is 0 otherwise.

    - End (while)
  - Add $\mathbf{h}_i$ to the i-th column of $\mathbf{H}$.
  - Use $\sum_{i=1}^{i=l}(d_i - 2)$ to calculate the ACE values of all cycles, originated from v_i, with length $2d_{\text{ACE}}$ or smaller.
  - While (The minimum ACE value at the current step is below a given threshold)
    - Remove $\mathbf{h}_i$ from $\mathbf{H}$.
    - Randomly connect the variable node v_i to d_i check nodes. Record the connections by the vector $\mathbf{h}_i = [h_{1i}\ h_{2i}\ \ldots\ h_{Mi}]^T$, where h_{ji} is 1 if the variable node v_i is connected to the check node c_j, and is 0 otherwise. Add $\mathbf{h}_i$ to the i-th column of $\mathbf{H}$.
    - Use $\sum_{i=1}^{i=l}(d_i - 2)$ to calculate the ACE values of all cycles, originated from v_i, with length $2d_{\text{ACE}}$ or smaller.
  - End (while)
- End ($i = 1$ to N)

2.4.3 Remark

In both the PEG and the ACE-based code-construction methods, the variable nodes have been arranged with degrees in a non-decreasing manner. To protect the information bits, the first M variable nodes are assigned to parity bits and the remaining $(N-M)$ nodes with relatively larger degrees are assigned to information bits. Consider the parity matrix $\mathbf{H}$. Denote the $M \times M$ sub-matrix corresponding to the first M variable nodes as $\mathbf{H}_1$, and the $M \times (N-M)$ sub-matrix corresponding to the remaining $(N-M)$ nodes as $\mathbf{H}_2$. Then the parity matrix $\mathbf{H}$ can be re-written as

$$\mathbf{H} = [\mathbf{H}_1|\mathbf{H}_2]. \tag{2.41}$$

Further, denote the number of degree-2 variable nodes by N_{v2}. In the PEG algorithm, $\min\{M, N_{v2}\}$ degree-2 nodes are actually connected in a zigzag manner, as shown in Fig. 2.4. (The connection of degree-two variable nodes in a zigzag manner maximizes the length of possible cycles involved with only degree-2 variable nodes, and prevents the degree-2 variable nodes from forming small cycles

with ACE equaling zero.) Moreover, in the figure, $N_{\text{gap}} = \max(M - N_{v2}, 0)$. It has been shown that if N_{gap} is small, the $\mathbf{H}$ in *approximate lower triangular form* will have a low encoding complexity [91]. In the special case of $N_{\text{gap}} = 0$, i.e., $N_{v2} \geq M$, the M-th degree-2 variable node can be further converted to a degree-1 variable node without resulting in much performance degradation [92]. Under such circumstances, the $M \times M$ sub-matrix $\mathbf{H}_1$, corresponding to $(M - 1)$ degree-2 variable nodes and one degree-1 variable node, becomes invertible. Moreover, $\mathbf{H}_1^{-T}$ becomes a matrix with all the upper-diagonal elements equaling 1. Subsequently, the generating matrix $\mathbf{G}$ corresponding to $\mathbf{H}$ can be written as

$$\mathbf{G} = [(\mathbf{H}_1^{-1}\mathbf{H}_2)^T | \mathbf{I}_M], \tag{2.42}$$

in which differential encoding with low complexity can be applied with the use of an accumulator. Furthermore, the matrix $\mathbf{I}_M$ in $\mathbf{G}$ denotes an $M \times M$ identity matrix.

2.5 Summary

This chapter presents a brief review of the LDPC codes. We have introduced both the bipartite graph representation and the matrix representation of the LDPC codes. With the help of the bipartite graph representations, we have elaborated the decoding algorithm and have shown that the decoder is optimal if and only if the "independence" assumption holds, i.e., the associated bipartite graph is cycle-free. Under such circumstances, the degree distributions of the codes can be optimized by "density evolution (DE)" which is described in Section 2.3.1. We have further presented a specific implementation scheme of DE, namely "discrete density evolution (DDE)". However, infinite-length codes are not realizable in practice. Short-length codes are more applicable but their performance varies as their bipartite graphs change. Furthermore, we have presented two popular methods that aim at building short-length LDPC codes with good error-correcting performance.

Regardless of the code-construction methods, short-length LDPC codes will inevitably contain cycles that will violate the "independence" assumption required by the BP decoder. In consequence, the decoder will not always converge to a fixed

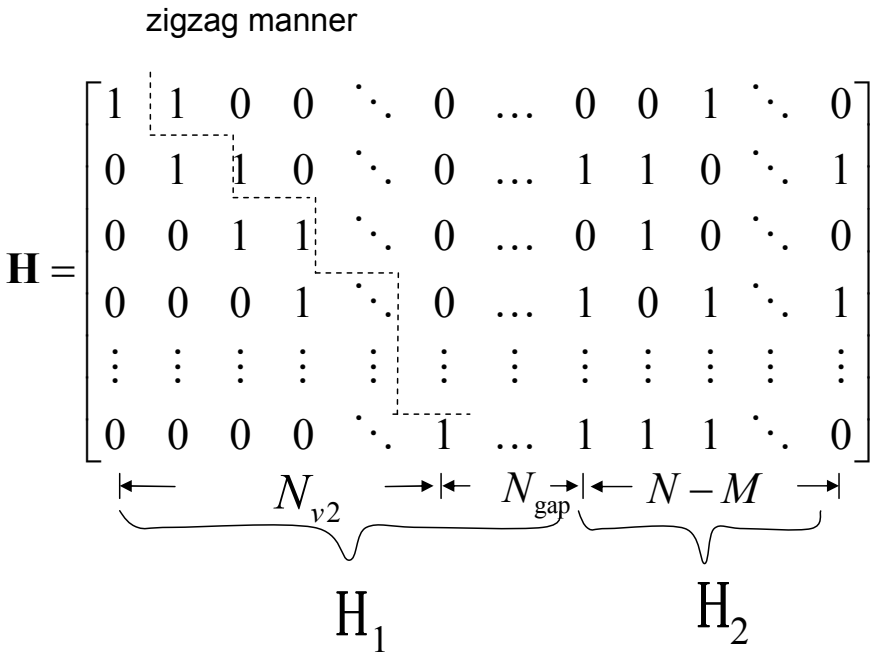


Figure 2.4: An example of matrix with degree-2 variable nodes connected in a zigzag manner.

point. Rich nonlinear dynamical phenomena should therefore be visualized at the decoder as the SNR changes. In the next chapter, we will begin our research of short-length LDPC codes by exploring the nonlinear dynamics of the BP decoders.

Chapter 3

Nonlinear dynamics of finite-length LDPC decoder

In this chapter, we begin our study into LDPC codes with an investigation of the decoding algorithm. By considering the finite-length LDPC decoders as high-dimensional nonlinear dynamical systems, we will investigate their dynamical behavior and bifurcation phenomena for a range of signal-to-noise ratios (SNRs). Moreover, we will derive the Jacobian of the system and calculate the corresponding eigenvalues at the fixed points for stability analysis. Based on the observations and findings, we propose a control scheme and evaluate its effectiveness in improving the error performance of the decoder.

3.1 Introduction to dynamical systems

The concept "dynamical system" [93] has its origins in Newtonian mechanics, and has been widely used in many natural science and engineering disciplines. In a dynamical system, the subsequent states of the system can be determined from the current states according to a given rule, which is usually described by a set of equations. A collection of the states as the equations iterate forms a trajectory or orbit. Given different initial conditions, the system may produce very different trajectories.

Moreover, the behavior of the trajectories is a function of the system parameters. As a certain parameter varies, the dynamical system may experience bifurcations, i.e., sudden "qualitative" or topological changes, in its behavior. There are several types of bifurcations, such as fold bifurcation, flip bifurcation and Neimark-Sacker bifurcation [94]. A fold bifurcation occurs when the system trajectory suddenly jumps from one fixed point to another fixed point. For a flip bifurcation, it happens whenever the period of the system doubles. Further, a Neimark-Sacker bifurcation takes place when a fixed point in the system loses stability and undergoes quasi-periodic oscillations. The types of bifurcation can also be determined by looking at the eigenvalues of the Jacobian, which is formed by the linearization of the system equations at the fixed points. If one of the eigenvalues is equal to unity, the bifurcation is a fold bifurcation. When the eigenvalue is equal to -1, it will form a flip bifurcation. If two pairs of complex eigenvalues approach the unit circle simultaneously, a Neimark-Sacker bifurcation occurs.

A dynamical system may also exhibit chaos if the system is nonlinear. Furthermore, the chaotic system is highly sensitive to the initial conditions and the system states appear to be random-like [95]. Given two different but very close initial points X_0 and $X_0 + \Delta X_0$, where $\Delta X_0 \to 0$. Then the separation between the two trajectories generated using the system equations, denoted by $\Delta X(X_0, t)$ where t denotes the time, will behave erratically. Define the mean exponential rate of divergence of the two initially close trajectories by the "Lyapunov exponent" λ_L, which is given by

$$\lambda_L = \lim_{t\to\infty} \frac{1}{t} \ln \frac{\Delta X(X_0, t)}{\Delta X_0}. \tag{3.1}$$

If $\lambda_L < 0$, the trajectory will be attracted to a stable fixed point or a stable periodic orbit. The trajectory will be an unstable fixed point if $\lambda_L = 0$. Finally, if $\lambda_L > 0$, the trajectory is unstable and chaotic.

3.2 BP decoder as a dynamical system

Consider a finite-length LDPC code with N variable nodes and M check nodes. Assume that the Belief Propagation (BP) algorithm in the probability field is used. To analyze the convergence behavior of a finite-length LDPC decoder, we

can re-write the whole iterative process in the probability field (see Sect. 2.2.2) as

$$\begin{cases} \mathbf{q}^{(l)}(0,\sigma) = f_1(\mathbf{r}^{(l)}(0,\sigma)) \\ \mathbf{r}^{(l+1)}(0,\sigma) = f_2(\mathbf{q}^{(l)}(0,\sigma)) \end{cases} \tag{3.2}$$

where both $\mathbf{r}^{(l)}(0,\sigma)$ and $\mathbf{q}^{(l)}(0,\sigma)$ are vectors parameterized by σ and of length $N\lambda(1)/\int_0^1 \lambda(x)\,dx$. (Recall that $\lambda(x)$, as defined in (2.1), specifies the variable-node degree distribution.) It can be observed that the whole process is parameterized by the probabilities $\Pr(\psi_i = 0|y_i)$ for $i = 1, 2, ..., N$, which are determined by the transmitted codeword and the noise values. Simulations have also shown that the iterative decoder is very sensitive to such parameters.

Suppose that a fixed point exists in the dynamical system. Linearizing (3.2) around the fixed point, we get

$$\begin{aligned} & \begin{cases} \mathbf{q}^{(l)}(0,\sigma') = \mathbf{J}_1\mathbf{r}^{(l)}(0,\sigma') \\ \mathbf{r}^{(l+1)}(0,\sigma') = \mathbf{J}_2\mathbf{q}^{(l)}(0,\sigma') \end{cases} \\ \Rightarrow \quad & \mathbf{r}^{(l+1)}(0,\sigma') = \mathbf{J}_2\mathbf{J}_1\mathbf{r}^{(l)}(0,\sigma')\big|_{\mathbf{r}^{(l+1)}=\mathbf{r}^{(l)}} = \mathbf{J}\mathbf{r}^{(l)}(0,\sigma')\big|_{\mathbf{r}^{(l+1)}=\mathbf{r}^{(l)}} \end{aligned} \tag{3.3}$$

where $\mathbf{J}_1$ and $\mathbf{J}_2$ are the Jacobian matrices of the functions f_1 and f_2, respectively, $\mathbf{J} = \mathbf{J}_2\mathbf{J}_1$, and σ' is the parameter at the fixed point. For specific variable nodes i and i_1 and check nodes j and j_1, the $(\varphi(i,j), \varphi(i_1,j_1))$-th element of the matrix $\mathbf{J}$ can be shown equal to

$$J_{\varphi(i,j)\varphi(i_1,j_1)} = \frac{(2r_{ji}^{(l+1)}(0) - 1) \times \mathbb{P}_{i_1} \times \prod_{j' \in V_{i_1}/j} \mathbb{R}_{j'i_1}^{(l)}}{\left[1 - \left(\mathbb{P}_{i_1} \times \prod_{j' \in V_{i_1}/j} \mathbb{R}_{j'i_1}^{(l)}\right)^2\right] \times r_{j_1 i_1}^{(l)}(0) \times r_{j_1 i_1}^{(l)}(1)} \times \delta(i,j,i_1,j_1) \tag{3.4}$$

where $\mathbb{P}_i = \frac{\Pr(\psi_i=1|y_i)}{\Pr(\psi_i=0|y_i)}$ and $\mathbb{R}_{ji}^{(l)} = \frac{r_{ji}^{(l)}(1)}{r_{ji}^{(l)}(0)}$. $\varphi(i,j)$ is an index function defined as

$$\varphi(i,j) = \sum_{i'=1}^{i-1}\sum_{j'=1}^{M} h_{j'i'} + \sum_{j'=1}^{j} h_{j'i}, \tag{3.5}$$

where h_{ji} is 1 if the j-th check node is connected to the i-th variable node; otherwise, it is 0. The output range of $\varphi(i,j)$ is from 1 to $N\frac{\lambda(1)}{\int_0^1 \lambda(x)\,dx}$. For the function

$\delta(i, j, i_1, j_1)$, its value equals 1 if the i-th and the i_1-th variable nodes are both connected to the j-th check node, with the j_1-th check node connected to i_1-th variable node; otherwise, it equals 0.

The stability of the fixed point can then be determined from the eigenvalues of the Jacobian of the iterative system evaluated at the fixed point. Perturbations grow exponentially if one of the absolute value of the eigenvalues is larger than 1 and decay if all the eigenvalues lie in the unit circle. A fixed point is said to be stable if all sufficiently small disturbances remote from it damp out in time. On the other hand, unstable equilibria, in which disturbances grow in time, are represented by unstable fixed points. If an eigenvalue approaches -1 and 1, respectively, flip bifurcation and fold bifurcation would occur. Also, Neimark-Sacker bifurcation occurs when a pair of complex conjugate eigenvalues move towards the unit circle from inside [96].

Although we can find the entire phase trajectories of $\mathbf{r}^{(l)}(0, \sigma)$ and $\mathbf{q}^{(l)}(0, \sigma)$, it is impractical to plot and study them all because both $\mathbf{r}^{(l)}(0, \sigma)$ and $\mathbf{q}^{(l)}(0, \sigma)$ are very high dimensional variables in the order of thousands. Instead, we make use of the measure $E(l)$ to investigate the dynamical behavior of the decoder, and $E(l)$ is defined as the mean-square value of the posterior probabilities of the code bits being equal to 0 at the l-th iteration, i.e,

$$E(l) := \frac{1}{N} \sum_{i=1}^{N} [Q_i^{(l)}(0)]^2 \tag{3.6}$$

where $Q_i^{(l)}(0)$ is the probability of the i-th bit being 0 at the l-th iteration. In our study, codewords with all zeros are used because it is known that the all-zero codeword is adequate for assessing the performance of a linear code with a symmetrical channel and a symmetrical decoding algorithm. Therefore, if all code bits are detected correctly after some iteration number, $Q_i^{(l)}(0) = 1$ for all i and consequently $E(l) = 1$.

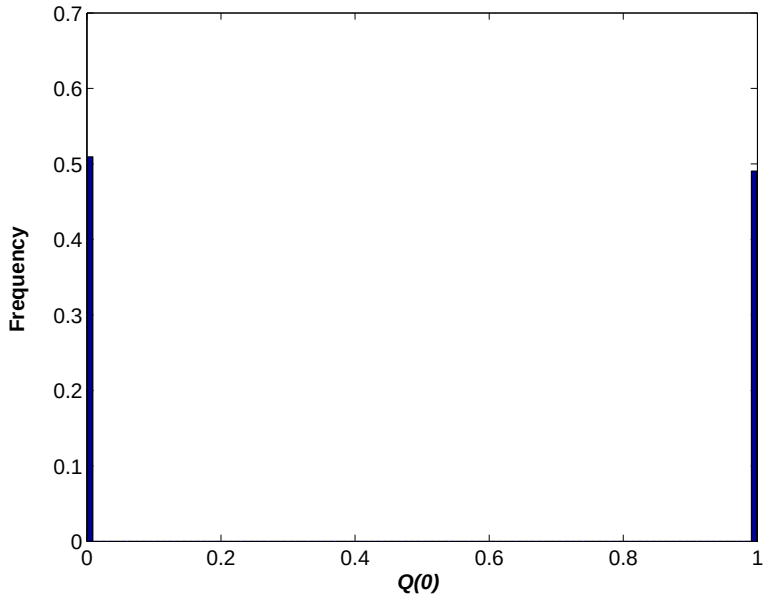
3.3 Bifurcation phenomena by simulations

Suppose the noise samples are represented by $\mathbf{z} = (z_1, z_2, ..., z_N)$. If the ratios between consecutive sample values, i.e., $(z_1/z_2, z_2/z_3, ..., z_{N-1}/z_N)$, are fixed, we refer to such noise samples as one noise realization. Different noise realizations correspond to different noise-ratios vectors. For a given noise realization, the noise vector $\mathbf{z}$ completely determines the SNR because $1/(2R{\times}\mathrm{SNR}) = \hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N} z_i^2$ where R is the code rate. Usually, N is a large integer and $\hat{\sigma}^2$ will be a good approximation of the channel noise power σ^2.

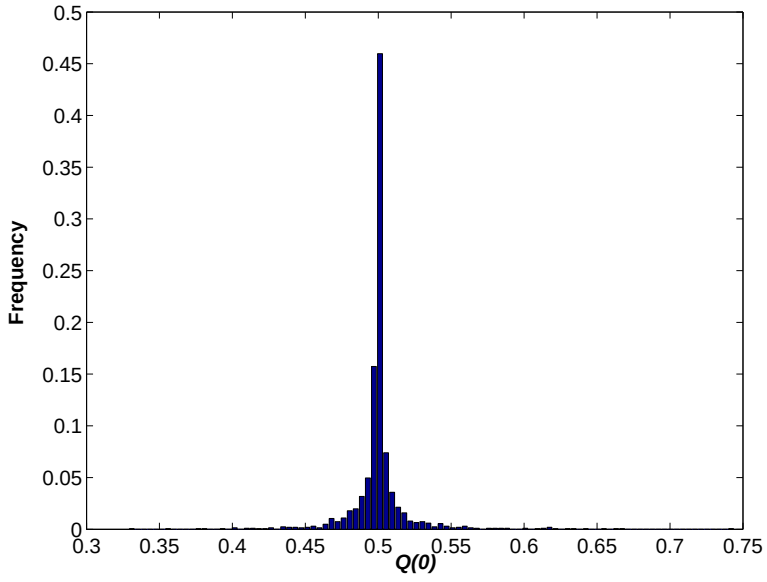
Extensive simulations have been performed to identify the relevant dynamics. In particular, it is found that fold bifurcation, flip bifurcation and Neimark-Sacker bifurcation occur within a certain range of the SNR called the "waterfall" region. For regions with high and low SNRs, two kinds of fixed points in the decoding system are observed, namely the unequivocal fixed point and indecisive fixed point. Note that the unequivocal fixed point is the desired fixed point which produces a definite decoding outcome. In Fig. 3.1, we plot the histograms of the posterior probability (based on (2.11)) at the fixed points for an arbitrary LDPC code. For an unequivocal fixed point, all the probability values converge to either 1 or 0, which is unequivocal for hard decision (see Fig. 3.1(a)). On the other hand, we refer to a fixed point as an indecisive fixed point when the LDPC decoding algorithm is relatively ambiguous regarding the values of the information bits, with posterior probability values heavily clustered around 0.5 (see Fig. 3.1(b)). It is also interesting to note that the algorithm converges to the unequivocal fixed point if and only if the decoder finds a valid codeword.

3.3.1 Irregular LDPC codes

We first consider an irregular $(1008, 504)$ LDPC code [97] and study the trajectories of the iterative decoding algorithm. Here, 1008 denotes the block length and 504 denotes the check length.



(a)



(b)

Figure 3.1: Histogram of posterior probability corresponding to (a) an unequivocal fixed point, and (b) an indecisive fixed point.

3.3.1.1 Fold Bifurcation

For some noise realizations, only fold bifurcations occur as we vary the SNR. Figure 3.2 shows the changes in the phase trajectories induced by a typical fold bifurcation for a particular noise realization. The figures on the left hand side plot the value of $E(l)$ against l, whereas those on the right plot the number of error bits against l. At an SNR value of 0.911785184 dB, the phase trajectory of the LDPC decoder converges to a stable indecisive fixed point, as indicated by the plots in Fig. 3.2(a). As the SNR is increased to 0.911785185 dB, the indecisive fixed point disappears and the phase trajectory is able to move away from this neighborhood and converges to an unequivocal fixed point. But there is a long time transient behavior of about 400 iterations before convergence finally takes place, as seen in Fig. 3.2(b). When the SNR is further increased to 0.92 dB, it takes less than 60 iterations for the trajectory to converge. In Fig. 3.3, the values of $E(l)$ at the steady state are plotted against SNR. It can be seen that in the low SNR region, the iterative algorithm converges to an indecisive fixed point, whereas in the high-SNR region, the algorithm converges to an unequivocal fixed point. Also, the two SNR regions are separated by a fold bifurcation.

3.3.1.2 Flip Bifurcation and Neimark-Sacker Bifurcation

For some other noise realizations, both flip bifurcations and Neimark-Sacker bifurcations occur. Figure 3.4 illustrates such bifurcations occurring at the LDPC decoder for another noise realization.

At an SNR value of 0.30 dB, the LDPC decoding algorithm converges to a stable indecisive fixed point (see Fig. 3.4(a)). As the SNR increases to about 0.44 dB, flip bifurcation occurs. Figure 3.4(b) shows a stable period-two cycle at the steady state at SNR = 0.55 dB. The periodic points lose their stability and eventually bifurcate at around SNR = 0.595 dB. Figure 3.4(c) shows that the trajectory converges to an indecisive fixed point at SNR = 0.601 dB. When the SNR is further increased to 0.615 dB, the phase trajectory converges to the steady state much slowly, as can be seen in Fig. 3.4(d). At around SNR=0.62 dB, the fixed point undergoes a Neimark-Sacker bifurcation and the phase trajectory goes into an invariant set. As a result, after a transient period, the phase trajectory converges to a quasi-periodic orbit.

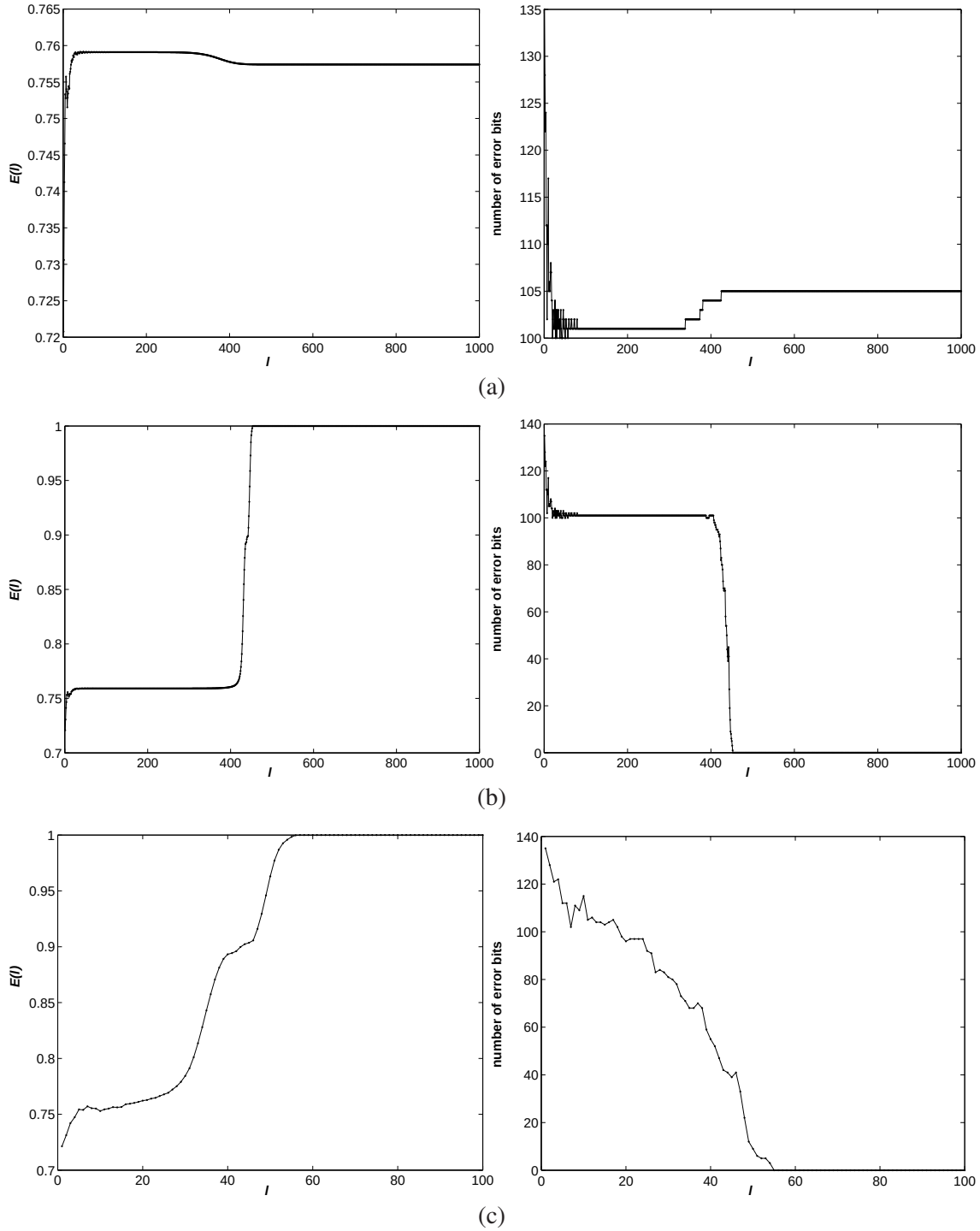


Figure 3.2: A typical fold bifurcation. Left: $E(l)$ versus l. Right: Number of error bits versus l. (a) SNR = 0.911785184 dB; (b) SNR = 0.911785185 dB; (c) SNR = 0.92 dB.

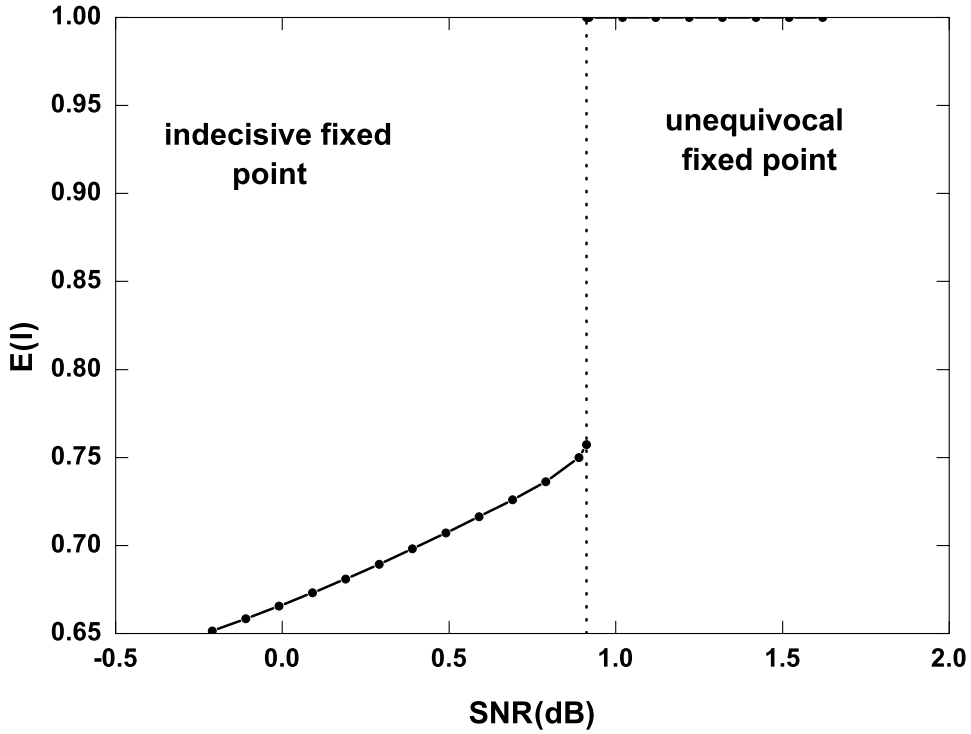


Figure 3.3: Bifurcation diagram of $E(l)$ for a particular noise realization. Dotted line corresponds to the SNR at which a fold bifurcation occurs.

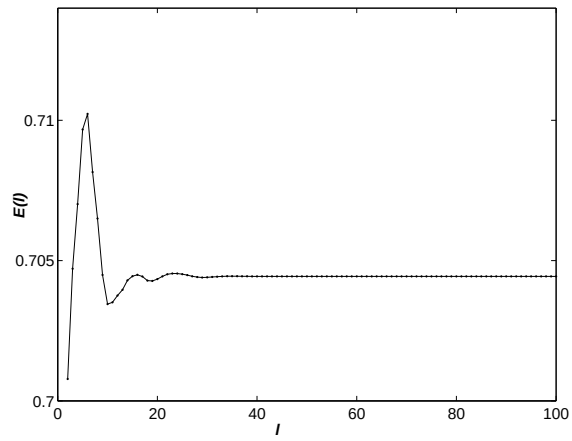
Figure 3.4(e) depicts the quasi-periodic orbit at the steady state at SNR = 0.65 dB. As the SNR increases, the trajectory finally loses its stability and chaos emerges at SNR = 0.85 dB. The chaotic trajectory at SNR = 0.94 dB is shown in Fig. 3.4(f). Finally, when the SNR is large enough, the LDPC decoding algorithm is able to find an unequivocal fixed point after a number of iterations. A trajectory corresponding to SNR = 1.004 dB is shown in Fig. 3.4(g).

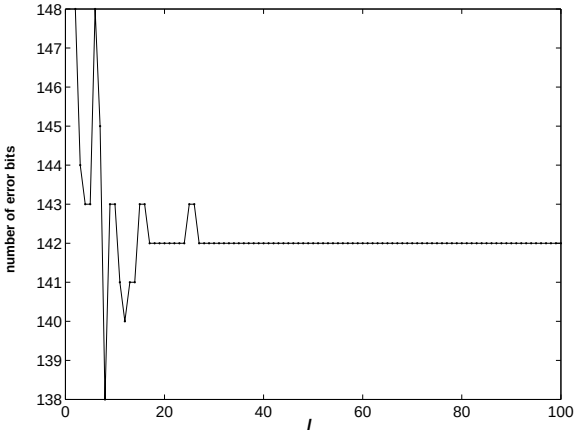
In Fig. 3.5, the values of $E(l)$ at the steady state are plotted against SNR. It can be observed that bifurcations occur in the SNR range of around 0.45 dB to 1 dB. In the SNR range of 0.45 dB to 0.595 dB, the oscillations can hardly be displayed in the figure because the amplitude of the oscillation, typically in the order of 10^{-4} as shown in Fig. 3.4(b), is too small.

Based on the values of $E(l)$ as the iteration progresses, we compute the Lyapunov exponent based on (3.1). In Fig. 3.6, we plot the Lyapunov exponent value against SNR. It is found that the Lyapunov exponent value approaches zero at SNR = 0.45 dB and at SNR = 0.62 dB, where flip bifurcation and Neimark-Sacker bifurcation occur, respectively. At SNR = 0.86 dB, the exponent turns positive, corresponding to the beginning of the chaotic region in Fig. 3.5. As the SNR is increased to around 1 dB, the Lyapunov exponent rapidly drops from a positive value to less than -30 (not shown in the figure due to its large negative value), indicating the super-stability of the unequivocal fixed point. Comparing Fig. 3.2 and Fig. 3.4, we also find that although different noise realizations produce different bifurcation diagrams, the whole SNR range can be roughly divided into three regions: (i) low-SNR region corresponding to indecisive fixed points; (ii) waterfall region where bifurcations occur; and (iii) high-SNR region corresponding to unequivocal fixed points.

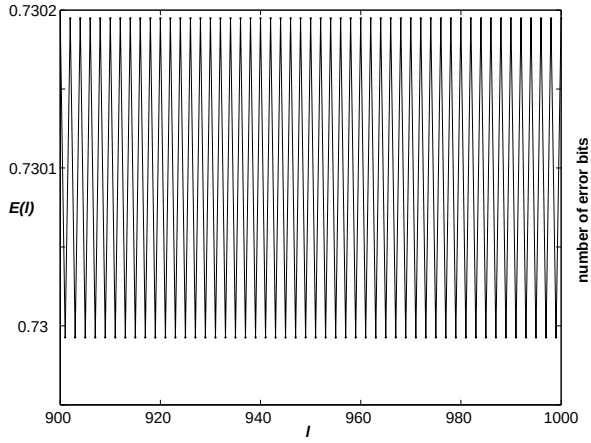
3.3.2 Regular LDPC codes

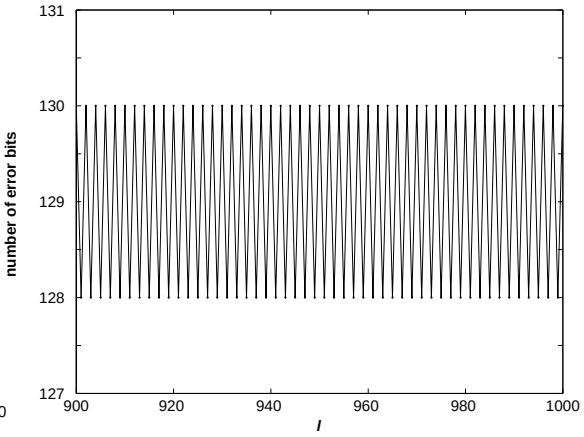
In this section, we present some results for the regular LDPC codes. From our extensive simulation results, we observe that decoders for regular LDPC codes have similar behavior as those for irregular codes. Here, we choose the regular LDPC code $(504, 252)$ (504 denotes the block length and 252 denotes the check length.) with variable-node degree being 3 and check-node degree being 6. To


0.71
0.705
0.7
E(l)
0
20
40
60
80
100
l

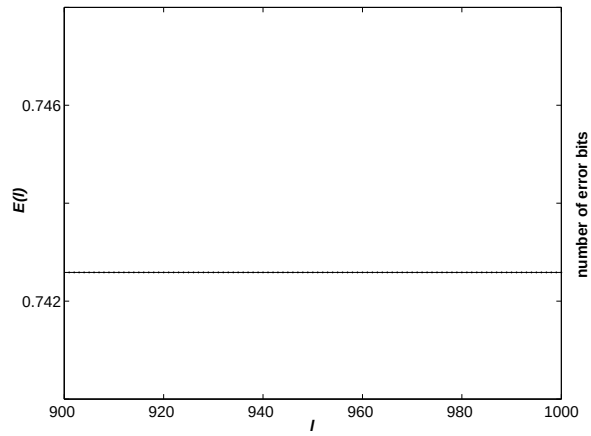

148
147
146
145
144
143
142
141
140
139
138
number of error bits
0
20
40
60
80
100
l

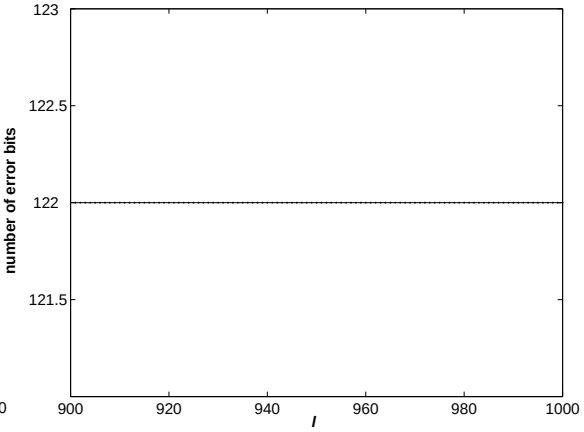
(a)


0.7302
0.7301
0.73
E(l)
900
920
940
960
980
1000
l

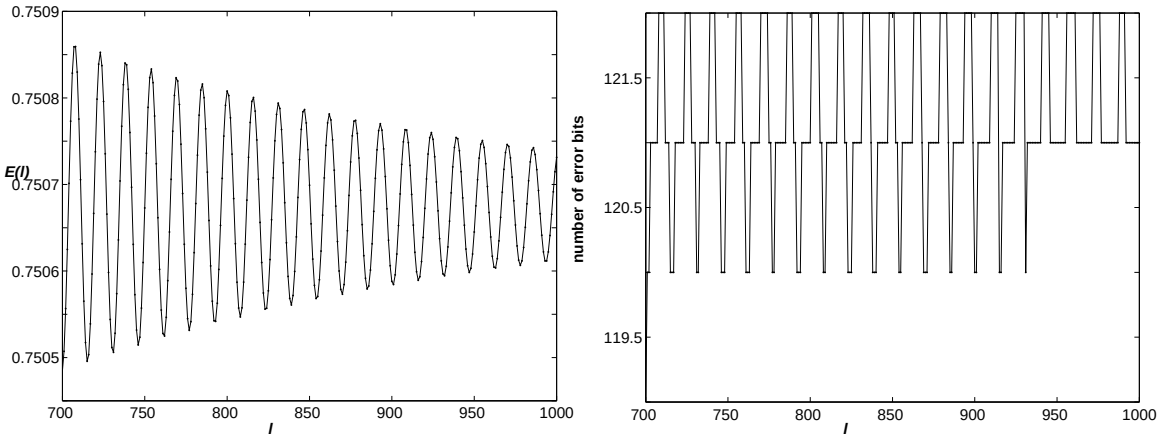

131
130
129
128
127
number of error bits
900
920
940
960
980
1000
l

(b)

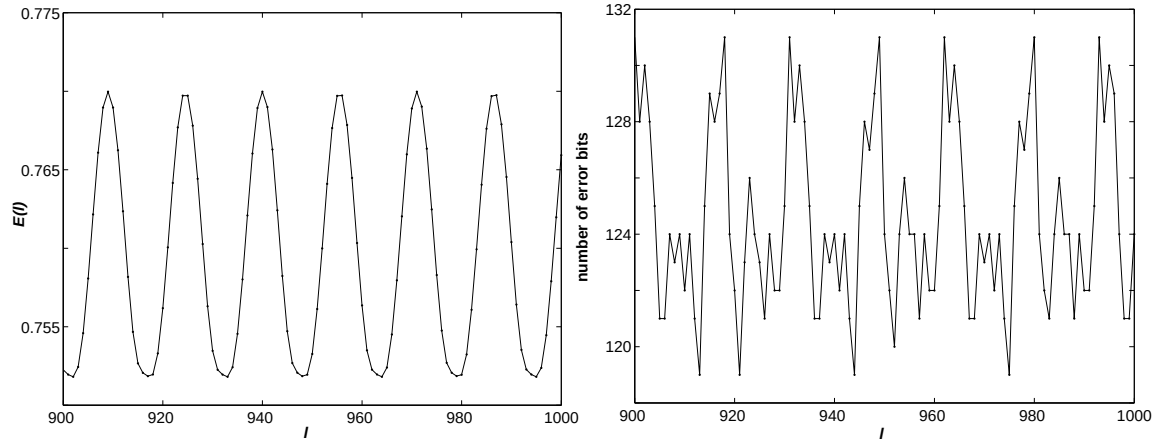

0.746
0.742
E(l)
900
920
940
960
980
1000
l


123
122.5
122
121.5
number of error bits
900
920
940
960
980
1000
l

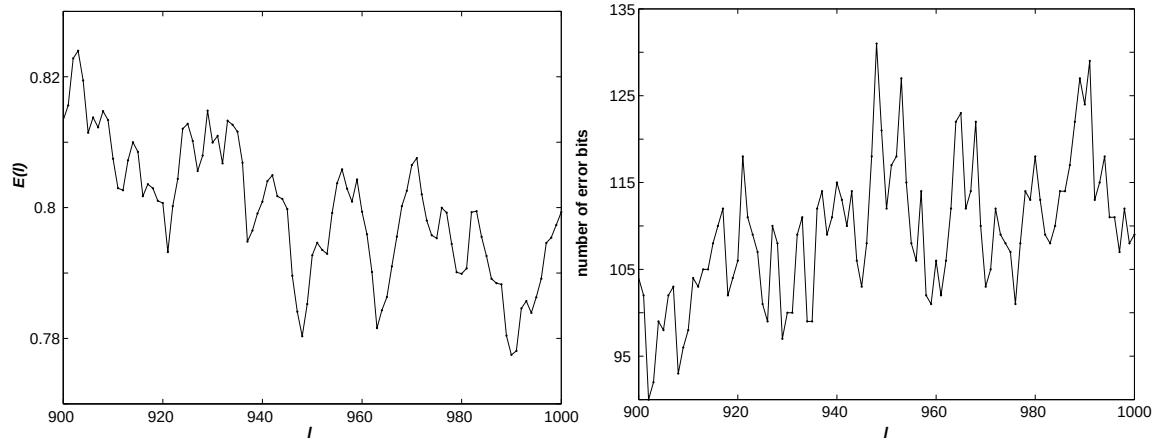
(c)


E(I)
I
number of error bits
I

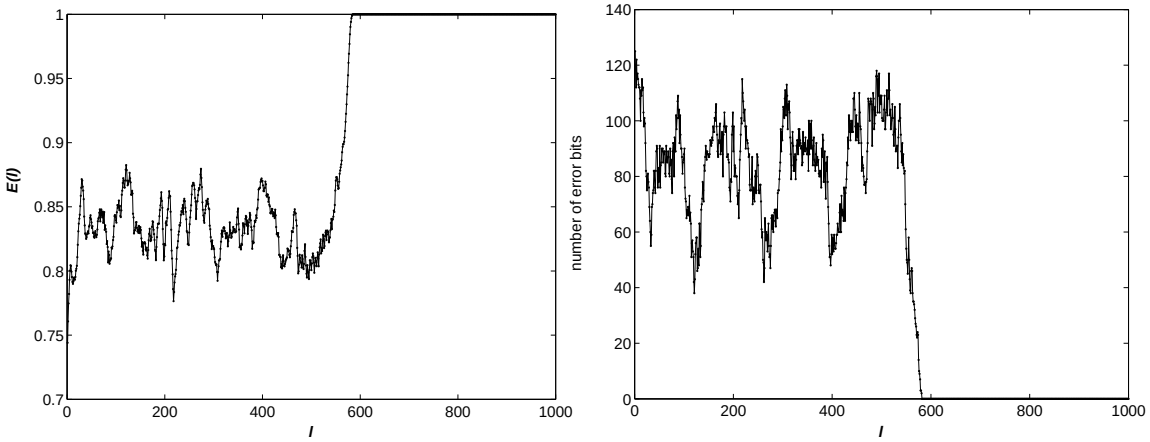
(d)


E(I)
I
number of error bits
I

(e)


E(I)
I
number of error bits
I

(f)



(g)

Figure 3.4: Typical flip bifurcation and Neimark-Sacker bifurcation. Left: $E(l)$ versus l. Right: Number of error bits versus l. (a) SNR = 0.30 dB; (b) SNR = 0.55 dB; (c) SNR = 0.601 dB; (d) SNR = 0.615 dB; (e) SNR = 0.65 dB; (f) SNR = 0.94 dB; (g) SNR = 1.004 dB.

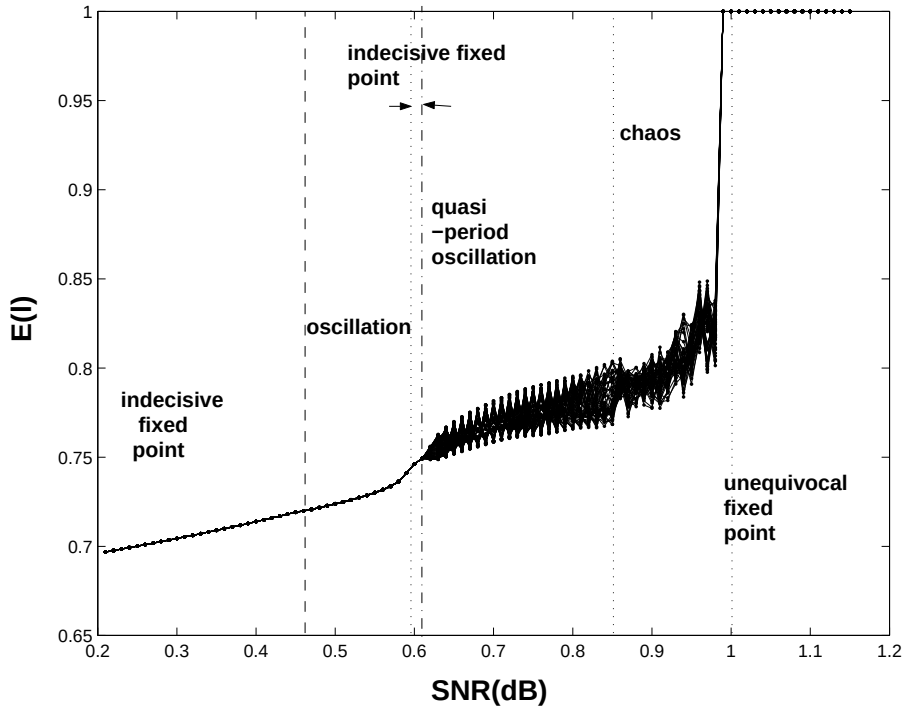


Figure 3.5: Bifurcation diagram of $E(l)$ for a particular noise realization. Dash line: flip bifurcation occurs; dash-dotted line: Neimark-Sacker bifurcation occurs; dotted line: other types of bifurcation occur.

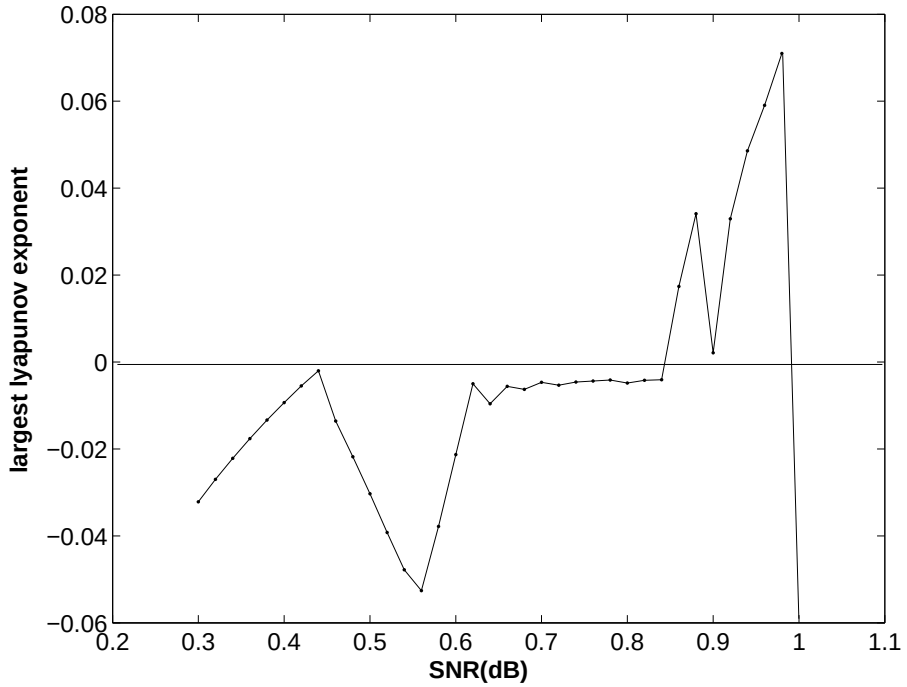


Figure 3.6: A plot of the Lyapunov exponent value against SNR with a particular noise realization.

describe the characteristics of bifurcations more clearly, we plot $E(l)$ against l and also the eigenvalues of the Jacobian for the corresponding fixed points.

In Fig. 3.7, at SNR = 1.400 dB, all the eigenvalues for the indecisive fixed points fall within the unit circle, indicating the stability of the fixed points. As SNR increases to 1.478 dB, one of the eigenvalues approaches 1, implying that fold bifurcation occurs. At SNR = 1.480 dB, the indecisive fixed point disappears, and an unequivocal fixed point appears.

To analyze the stability of the unequivocal fixed point, we derive the Jacobian as follows. As we know, when the algorithm converges to an unequivocal fixed point, all the messages passing between the variable nodes and check nodes converge to either 1 or 0. In our simulations, only the all-zero codes are transmitted, and hence the messages $\mathbf{r}^{(l)}(0,\sigma)$ and $\mathbf{r}^{(l)}(1,\sigma)$ converge to the all-one vector and all-zero vector, respectively. So, the $(\varphi(i,j),\varphi(i_1,j_1))$-th element of the Jacobian

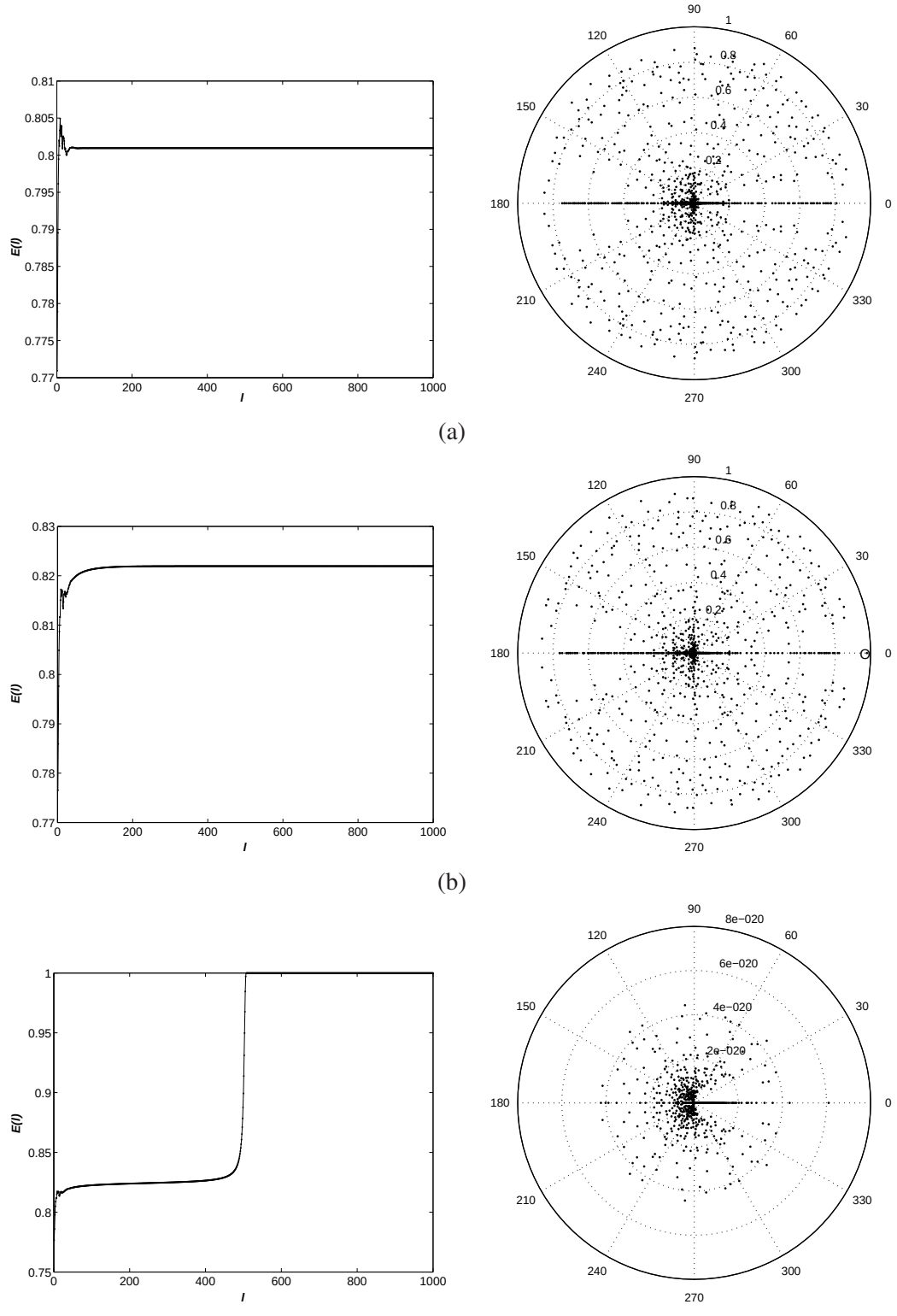


Figure 3.7: A typical fold bifurcation. Left: $E(l)$ versus l. Right: polar plots of eigenvalues. (a) SNR = 1.400 dB; (b) SNR = 1.478 dB; (c) SNR = 1.480 dB.

$\mathbf{J}$ where $\delta(i,i,i_1,j_1)=1$ can be written as

$$
\begin{aligned}
J_{\varphi(i,j)\varphi(i_1,j_1)} &= \lim_{\substack{\mathbf{r}^{(l)}(0,\sigma)\to\mathbf{1}\\ \mathbf{r}^{(l)}(1,\sigma)\to\mathbf{0}}} \frac{(2r_{ji}^{(l+1)}(0)-1)\times\mathbb{P}_{i_1}\times\prod_{j'\in V_{i_1}/j}\mathbb{R}_{j'i_1}^{(l)}}{\left[1-\left(P_{i_1}\times\prod_{j'\in V_{i_1}/j}\mathbb{R}_{j'i_1}^{(l)}\right)^2\right]\times r_{j_1i_1}^{(l)}(0)\times r_{j_1i_1}^{(l)}(1)} \\
&= \lim_{x\to 0}\frac{\mathbb{P}_{i_1}\times x^{(d_{i_1}-1)}}{x} = \begin{cases}\mathbb{P}_{i_1} & \text{if} d_{i_1}=2\\ 0 & \text{otherwise.}\end{cases}
\end{aligned}
\tag{3.7}
$$

The above equation indicates that if the degree of the variable nodes is larger than 2, i.e., $\lambda_{i_1}>2(i_1=1,2,\ldots,N)$, the eigenvalues at the unequivocal fixed point are all zeros and have nothing to do with the channel parameter. In Fig. 3.7(c), the eigenvalues all lie in the circle of diameter less than 8×10^{-20} (nonzero values caused by small computational inaccuracy).

With other noise realizations, the phenomena of Neimark-Sacker bifurcation (see Figs. 3.8 and 3.9) as well as flip bifurcation (see Figs. 3.10 and 3.11) can be observed. Note that the noise realization used to observe a Neimark-Sacker bifurcation is different from that used to observe a flip bifurcation.

3.4 Feedback control techniques

3.4.1 Methodology

In the previous section, the dynamic behavior of the iteration algorithm has been studied under different noise realizations. Fixed points, periodic oscillations, quasi-periodic oscillations and chaos have been observed. Given a noise realization, we can plot the schematic bifurcation diagram [98], as in Fig. 3.5. The whole SNR region can be divided into three parts: low SNR region, waterfall region and high SNR region. In the low SNR region, the decoder converges to a stable indecisive fixed point. As the SNR increases, the fixed point loses its stability and bifurcation occurs, leading to the phenomena of oscillation and chaos. As SNR gets higher, the algorithm finds another stable fixed point called unequivocal fixed point. Note that for different noise realizations and different codes, the waterfall region varies.

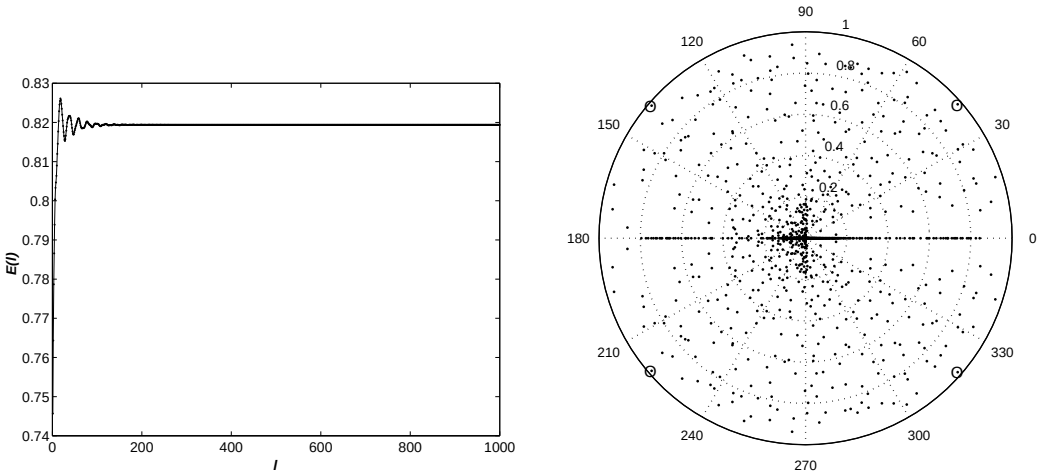


Figure 3.8: Phase trajectory of $E(l)$ before Neimark-Sacker bifurcation. Left: $E(l)$ versus l. Right: polar plots of eigenvalues. SNR = 0.80 dB. (Note that two pairs of complex eigenvalues approach the unit circle.)

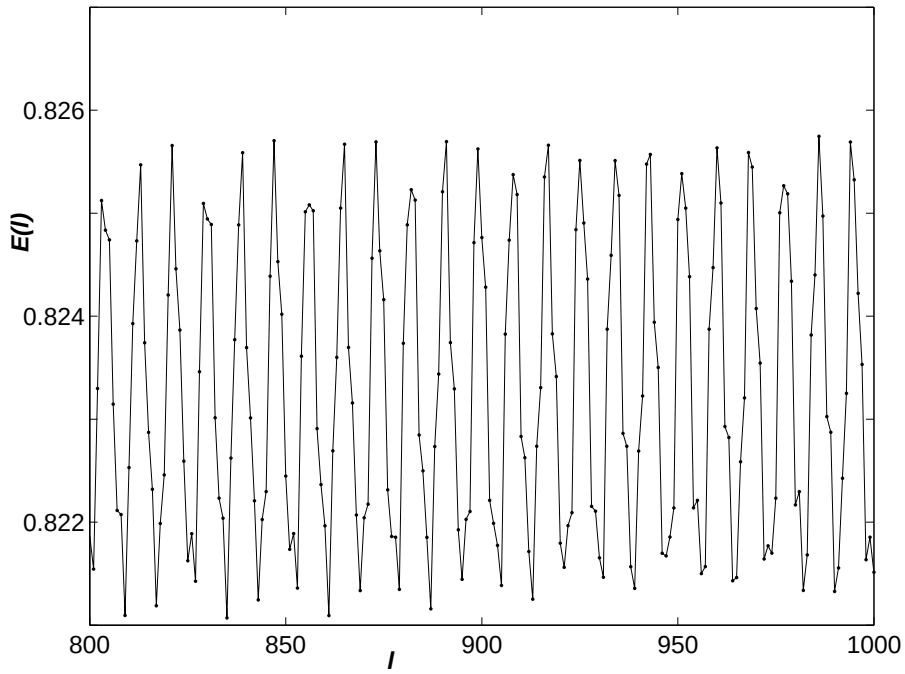


Figure 3.9: Phase trajectory of $E(l)$ after Neimark-Sacker bifurcation. SNR = 0.85 dB.

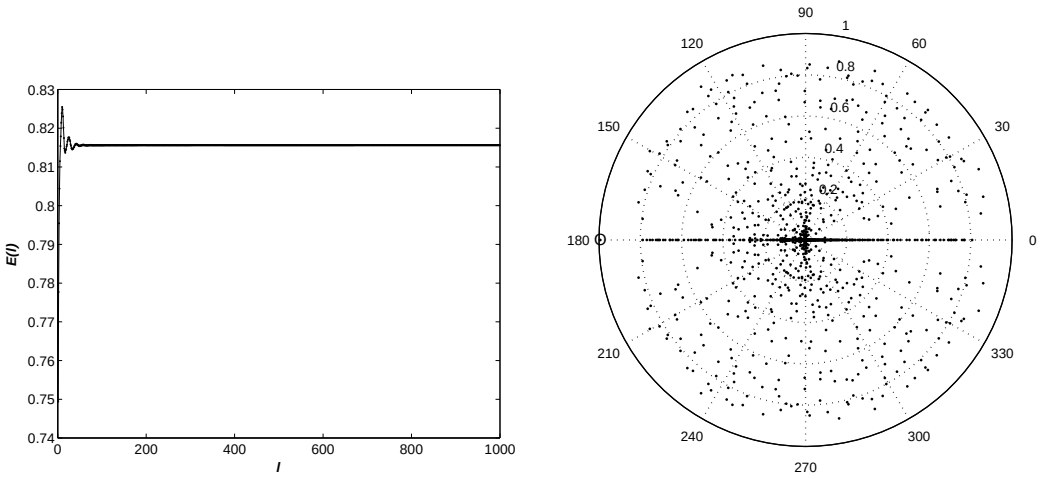


Figure 3.10: Phase trajectory of $E(l)$ before flip bifurcation. Left: $E(l)$ versus l. Right: polar plots of eigenvalues. SNR = 0.60 dB. Note that one eigenvalue approaches -1.

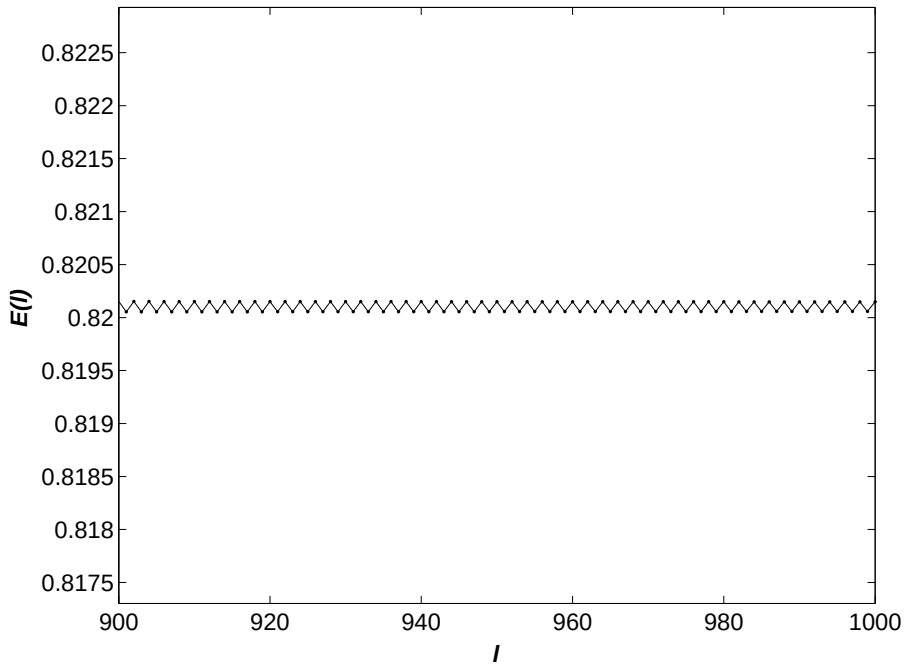


Figure 3.11: Phase trajectory of $E(l)$ after flip bifurcation. SNR = 0.65 dB.

Although the stable unequivocal fixed point exists in the whole SNR region, without any prior knowledge of the transmitted signals, it is very difficult to identify the attraction bin of the unequivocal fixed point in the high dimensional decoding system. In this section, we evaluate the effectiveness of feedback techniques that aim to facilitate the decoder to converge to the unequivocal fixed points. First, we investigate the time-delay feedback control method, which involves a control signal formed from the difference between the current state and the state of the system delayed by some time period. The technique needs no information about the target and can force the algorithm to stabilize at periodic orbits when the time-delay factor equals the period of the orbit. For a fixed point, the time-delay factor equals unity. However, this kind of feedback may also stabilize the indecisive fixed point and leads to an incorrect decoded codeword.

Suppose the iterative algorithm converges to an unequivocal fixed point. All the posterior probability values of the messages passing between variable nodes and check nodes tend to be either 1 or 0. Although we do not know which posterior probability values are 1s, the values passing from or going to the same variable nodes will be the same. In other words, if the algorithm converges to the unequivocal fixed point, the LLR messages $Lr_{ji}^{(l)}$ will be the same for any j-th check node connected to the same i-th variable node. It is a unique characteristic for unequivocal fixed points. Therefore, instead of time-delay feedback, we can use spatial-delay feedback method [99], i.e., $x_i(t) - \frac{1}{d_i}\sum_{i'=1}^{d_i} x_{i'}(t)$, where $x_i(t)$ is the current state of the i-th node of the network and d_i is the total number of nodes connected to the i-th node.

For a highly nonlinear system, it is recommended that a nonlinear feedback function $g(x_i(t) - \frac{1}{d_i}\sum_{i'=1}^{d_i} x_{i'}(t))$ where $g(0) = 0$, should be used [100]. Note that the control term will vanish as the target is realized. Since the LDPC decoder is a highly nonlinear coupled system, we choose the nonlinear spatial-delay feedback as our control method. With the control term, the system equations of the decoder in the log-likelihood ratio (LLR) field can be written as

$$\mathbf{Lr}^{(l+1)} = f(\mathbf{Lr}^{(l)}) - \eta(f(\mathbf{Lr}^{(l)}) - \bar{\mathbf{Lr}}^{(l)}) \tag{3.8}$$

where the elements of $\bar{\mathbf{Lr}}^{(l)}$, denoted by $\{\bar{Lr}_{ji}^{(l)}\}$, is given by $\frac{1}{d_i}\sum_{j'\in C_i} Lr_{j'i}^{(l)}$. Here, i and j denote, respectively, the i-th variable node and the j-th check node. Also, d_i

is the degree of the i-th variable node, and η is the feedback gain factor. Note that $f(\mathbf{Lr}^{(l)}) - \bar{\mathbf{Lr}}^{(l)}$ is the control signal vector, which is the function of the difference between the messages produced by current node and the average messages produced by other nodes connected to the current node.

3.4.2 Stability Analysis at the fixed points

Linearizing (3.8) at the fixed point, we can get:

$$\mathbf{Lr}^{(l)}(\sigma') = \mathbf{J}\mathbf{Lr}^{(l)}(\sigma') \tag{3.9}$$

where $\mathbf{J}$ is the Jacobian matrix of the function f, and σ' is the parameter at the fixed point. For specific variable nodes i and i_1 and check nodes j and j_1, we define the connection function $\delta_{1,2}(i,j,i_1,j_1)$ as follows. For the function $\delta_1(i,j,i_1,j_1)$, its value equals 1 if $i_1 = i$ and $j_1 \in C_i$; otherwise, it equals 0. For $\delta_2(i,j,i_1,j_1)$, its value equals 1 if the variable nodes i and i_1 are both connected to the check node j, with check node j_1 connected to variable node i_1; otherwise, it equals 0. So, the $(\varphi(i,j),\varphi(i_1,j_1))$-th element of the matrix $\mathbf{J}$ can be shown equal to

$$J_{\varphi(i,j)\varphi(i_1,j_1)} = (1-\eta) f'_{i,j,i_1,j_1} \times \delta_2(i,j,i_1,j_1) + \frac{\eta}{d_i}\delta_1(i,j,i_1,j_1) \tag{3.10}$$

where

$$f'_{i,j,i_1,j_1} = \frac{\exp(Lr_{ji}^{(l)}) - \exp(-Lr_{ji}^{(l)})}{\exp(LP_{i_1} + \sum_{j'\in C_{i_1}/j} Lr_{j'i_1}^{(l)}) - \exp(-LP_{i_1} - \sum_{j'\in C_{i_1}/j} Lr_{j'i_1}^{(l)})} \tag{3.11}$$

and $(\varphi(i,j)$ is the index function defined in (3.5). Here, LP_{i_1} denotes the initial messages at the i_1-th variable node as defined in Section 2.2.4.

When the algorithm converges to an unequivocal fixed point, for an all-zero transmitted codeword, the conditional posterior probability values of the messages, passing between variable nodes and check nodes, that the code bit equals 0 tend to unity. Hence, the corresponding conditional LLR messages $Lr_{ji}^{(l)} \to \infty$. Note that for codes without degree-2 variable nodes, $f'_{i,j,i_1,j_1} \to 0$ for any i,j,i_1,j_1. So the

Jacobian matrices can be given as

$$\mathbf{J} = \begin{bmatrix} \frac{\eta}{d_1}\mathbf{1}_{d_1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \frac{\eta}{d_2}\mathbf{1}_{d_2} & \mathbf{0} & \mathbf{0} \\ \dots & \dots & \dots & \dots \\ \mathbf{0} & \dots & \mathbf{0} & \frac{\eta}{d_n}\mathbf{1}_{d_n} \end{bmatrix}. \tag{3.12}$$

where $\mathbf{1}_{d_i}$ is the all ones matrix with dimension $d_i \times d_i$, and $n = N\frac{\lambda(1)}{\int_0^1 \lambda(x)\,dx}$ denotes the total number of edges. For the special Jacobian matrix, it is easy to show that the largest eigenvalue equals $\frac{\eta}{d_i}d_i = \eta$. Therefore, to maintain the stability of the unequivocal fixed point after introducing the control, η should be selected under the constraint $\eta < 1$.

3.4.3 Results and discussion

We consider the high rate $(273, 82)$ code [97]. Here, the two numbers in brackets denote the block length of the code and check length of the code respectively. Extensive simulations have been performed. Two decoders are under investigation, namely the one based on the original BP and the one with the nonlinear spatial-delay feedback introduced. The maximum iteration number is set as 50, and 5000 different noise samples are produced for evaluating the performance of the two decoders. The feedback gain factor η should be chosen carefully. If η is too small, it exerts no effect on the iteration algorithm. When η is too large, however, the magnitude of the control term will be so large that it destroys the structure of the system. Extensive simulations have been performed and results show that a proper value of η should lie between 0.05 and 0.2. We also find that 0.2 is the most appropriate value for the $(273, 82)$ code so we set $\eta = 0.2$.

Moreover, in the original BP algorithm, the LLR values of $Lr_{ji}^{(l)}$ in the decoding process will spread over a very large range. $Lr_{ji}^{(l)}$ with large values indicate that the corresponding bits are either of high reliability or seriously corrupted by noise. To avoid the use of the seriously corrupted bits as the control terms, we set a threshold η_{th} such that the control terms should be added only to those variables whose absolute values are beblow η_{th}. In our simulations, we set $\eta_{th} = 2$.

Fig. 3.12 and Fig. 3.13 shows the block error rate (BLER) and bit error rate

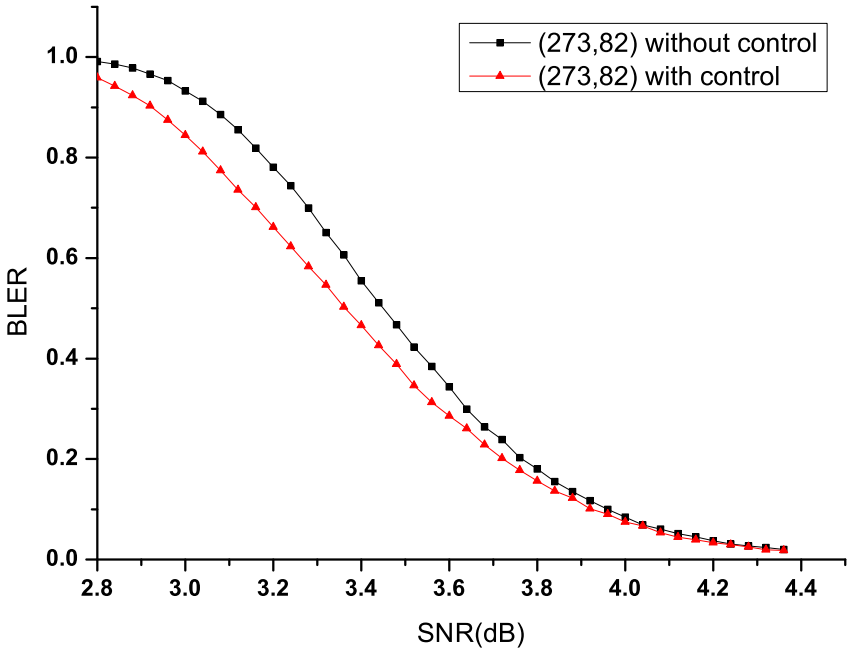


Figure 3.12: Block error rate (BLER) of the $(273, 82)$ codes with and without control.

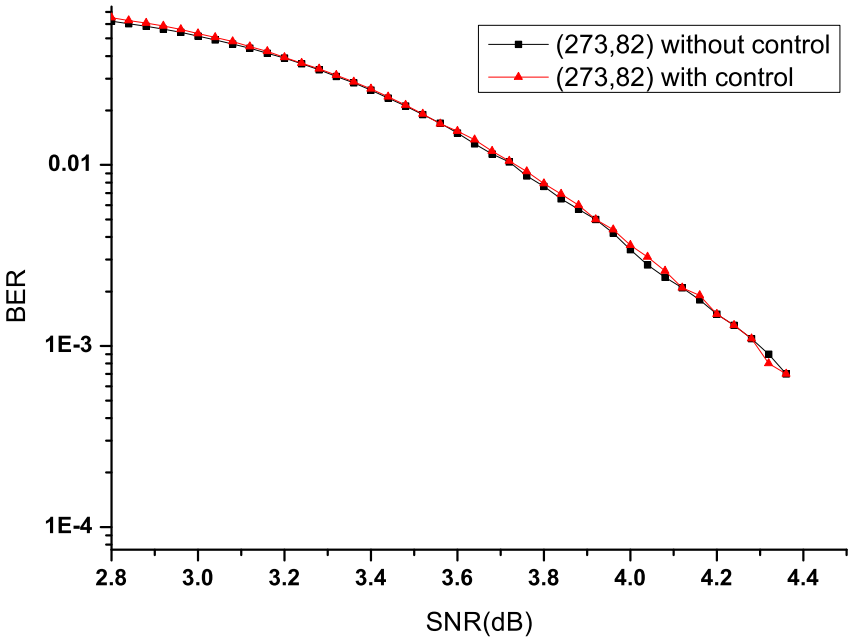


Figure 3.13: Bit error rate (BER) of the $(273, 82)$ codes with and without control.

(BER) of decoders, respectively. It can be observed that while the proposed iterative algorithm with control provides a lower BLER compared to the original BP algorithm, there is no significant difference in the BER performance between the two decoding algorithms. Fig. 3.14 show the histogram of the number of errors per block when SNR$=3.1$ dB. It is found that the proposed algorithm has increased the number of error-free blocks compared to the original BP algorithm, meaning that the success rate for the proposed algorithm to find a valid codeword is higher. Moreover, the proposed algorithm reduces the number of blocks with 1 to 15 errors. However, our method produces more blocks with errors larger than 15. When summing all the errors over all blocks, the proposed method and the BP algorithm produce comparable number of errors, as shown in Fig. 3.13. Fig. 3.15 shows the results again for SNR$=3.3$ dB, and similar observations can be found.

3.5 Summary

In this chapter, we have observed fixed points, bifurcations, oscillations and chaos in the phase trajectory of a finite-length LDPC decoder. Furthermore, we have investigated the effectiveness of a simple feedback technique that controls the transient behavior in LDPC decoding algorithm. A small BLER performance has been observed but there is no significant improvement of the BER.

Unlike infinite-length LDPC codes, LDPC codes with finite length do not always converge, as shown in the simulation results in this chapter. Thus, degree distributions optimized by "density evolution" so as to maximize the threshold will guarantee excellent performance for infinite-length LDPC code, but may not be the best choice for finite-length LDPC codes. In the next chapter, we attempt to tackle this issue by applying the complex network theory to the design of short-length LDPC codes.

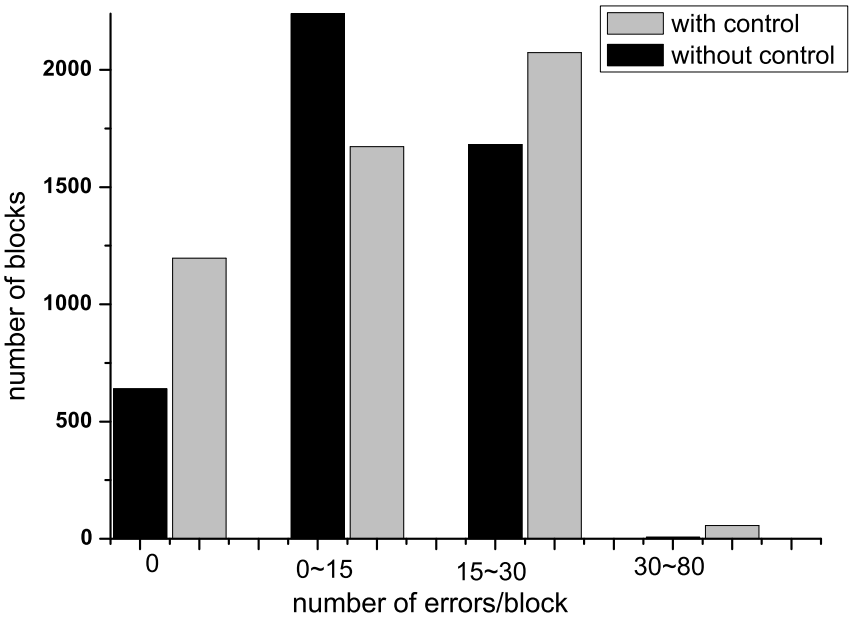


Figure 3.14: Frequency distribution of the number of errors at SNR=3.1 dB.

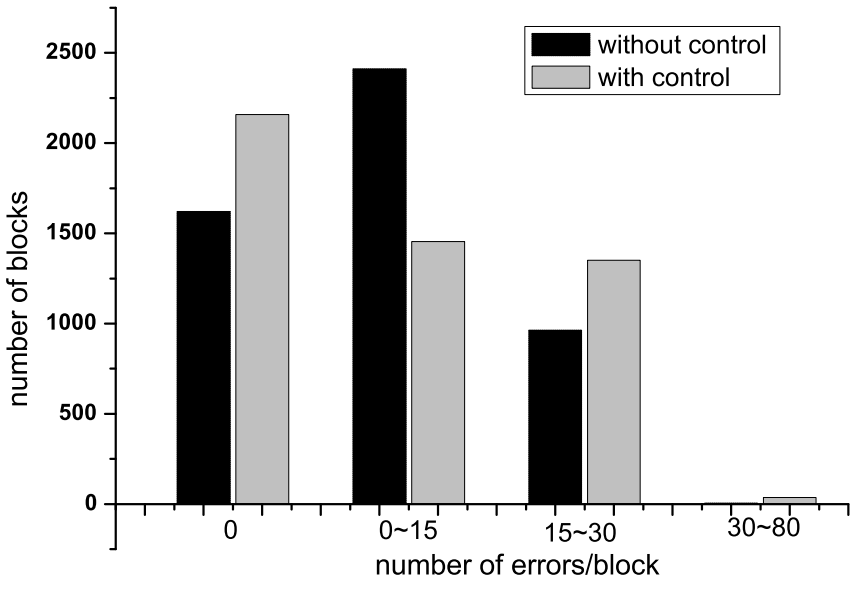


Figure 3.15: Frequency distribution of the number of errors at SNR=3.3 dB..

Chapter 4

Short-length LDPC codes built on scale-free networks

Study of complex networks has been conducted across many fields of science, including computer networks, biological networks and social networks. Characteristics of different types of complex networks such as random networks, regular-coupled networks, small-world networks, and scale-free networks, have been discovered by researchers. Application of such network properties to solve engineering problems, however, is still at the infancy stage. In this chapter, we make one of the first attempts in applying complex-network theories to communications engineering. In particular, inspired by the shortest-average-path-length property of scale-free networks, we design short-length LDPC codes with an aim to improving the convergence rate of the LDPC codes. We will also compare the error performance, both theoretically and by simulations, of the proposed codes with those of other well-known LDPC codes.

4.1 Review of complex networks

Complex networks, like graphs, consist of nodes interconnected by edges (links). In the study of networks and graphs, the *degree* of a node represents the number of edges connecting the node to other nodes. The node degrees over a network are characterized by a probability distribution $\Pr(k)$ which gives the frac-

tion of nodes having degree $k = 1, 2, 3, ...$ Consider any two nodes in a network. The two nodes are connected if there is a path composing one or several node(s) and edge(s) between them. The distance between the two nodes is defined as the number of edges along the shortest path connecting them. The average path length (APL) of the whole network is then the mean distance between any two nodes, i.e., averaging the distance over all pairs of nodes.

Consider a network with N nodes. Referring to Fig. 4.1, there are four well-known and much studied classes of complex networks: random network, lattice, small-world network and scale-free network. Random network was first defined by Erdős and Rényi in 1959 [101]. A random network is obtained by starting with N nodes and adding an edge to each pair of nodes with a given probability. The node degrees of a random network follow a Poisson distribution while the APL is proportional to $\log(N)$. In contradiction to random network, a lattice is constructed by connecting each node to a fixed number of adjacent nodes. The APL of a lattice is large and tends to infinity as N approaches infinity. However, most of the real-world networks are neither entirely regular nor entirely random. In 1998, a model for a small-world network was proposed [44]. Starting with a regular lattice containing N nodes, each link is rewired randomly with a probability of p. The rewiring process significantly shortens the APL of the network to $O(\log(N))$. The probability distribution of the node degrees over the rewired network, moreover, is determined by the rewiring probability p. In the extreme case of $p = 0$, the regular lattice remains untouched all node degrees are identical. If $p = 1$, however, all the original links will be rewired and the new network becomes a random network with node degrees being Poisson-distributed.

A recent significant discovery in the complex network theory is that some complex networks, such as the Internet and the worldwide web, have power-law degree distributions. It implies that a small number of nodes have very large numbers of connections (degrees) while a great majority of the nodes have very few connections. Such kind of networks are called *scale-free networks* [45], and their degree distributions can be written as $\Pr(k) = Ak^{\gamma}$ where γ is the characteristic exponent and A is the normalizing coefficient. Compared with regular-coupled networks, small-world networks and random networks, scale-free networks of the same size (number of nodes) and with the same number of connections are found to accom-

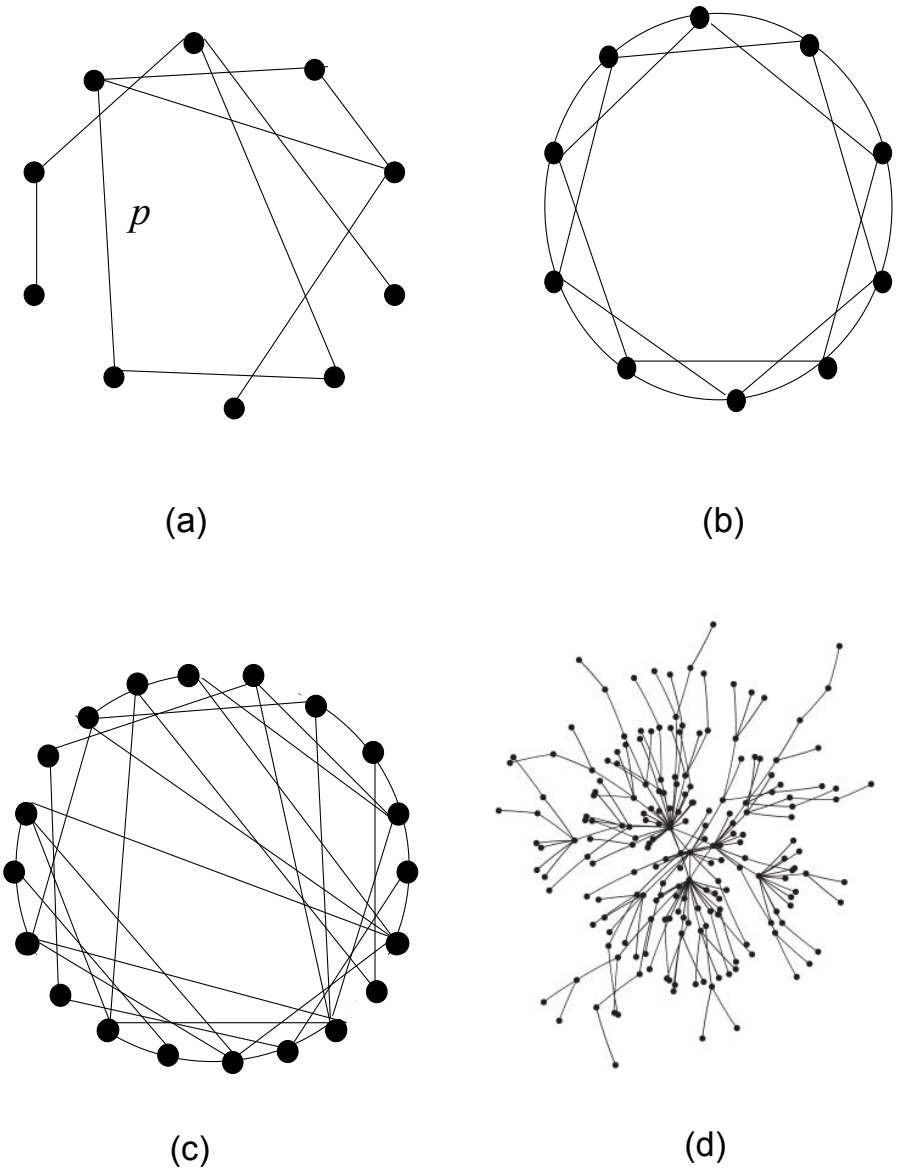


Figure 4.1: (a) Random network; (b) lattice; (c) small-world network; (d) scale-free network.

plish the shortest average path length [53]. While for the same average path length, among the aforementioned networks, complex networks with the scale-free property have the smallest number of connections. In particular, it has been shown that when (i) the value of the characteristic exponent, i.e., γ, lies between 2 and 3; and (ii) the network arrives at the natural cutoff (i.e., the maximum node degree equals the minimum node degree times $N^{1/(\gamma-1)}$), the average path length of the scale-free networks is $O(\log(\log(N)))$ [53]. Table 4.1 compares the main properties for the four network topologies.

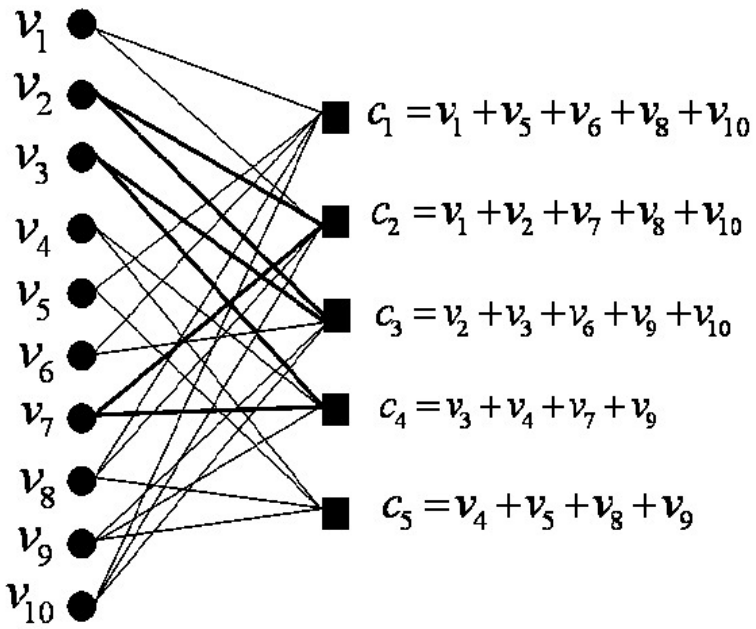
4.2 Scale-free LDPC codes

4.2.1 "Pure" scale-free LDPC codes

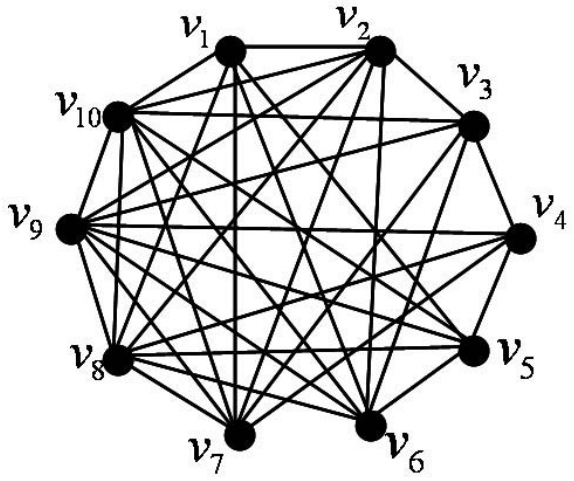
Recall that as the iterative decoding process proceeds, the information generated by each variable node will eventually be conveyed to all other variable nodes via the check nodes. To visualize the flow of messages among the variable nodes, we remove the check nodes and construct a complex network using only the variable nodes. Moreover, two variable nodes v_i and v_j are connected in the complex network only if they are connected to the same check node in the original bipartite graph. Figure 4.2(b) shows the complex network formed based on the LDPC code in Fig. 4.2(a).

For the complex network so formed, the average path length corresponds to the average number of iterations required for an updated message from one variable node to eventually pass to another variable node. Decreasing the average path length will no doubt accelerate the exchange of messages among the variable nodes, thereby reducing the number of iterations the decoding algorithm takes to converge. This is particularly useful in the high SNR region where the decoder has a much higher chance to converge. As scale-free networks have been shown to provide a very short average path length [53], it will be an advantage if LDPC codes are designed in such a way that the resultant complex network formed by the variable nodes has a scale-free property.

One approach is to start with a complex network (like Fig. 4.2(b)) with a



(a)



(b)

Figure 4.2: A $(10, 5)$ LDPC code. (a) Bipartite graph representation; (b) Complex network formed by removing the check nodes.

Table 4.1: Comparison of the main properties for random network, lattice, small-world network and scale-free network.

Network topology	**Average Path Length**	**Degree distribution**
Random network	$O(\log(N))$	Poisson
Lattice	$O(N)$	Delta
Small-world network	$O(\log(N))$	-
Scale-free network	$O(\log(\log(N)))$ [53]	Power-law

power-law degree distribution and convert it directly into a bipartite graph (like Fig. 4.2(a)) that represents the LDPC code. However, the conversion task is not a trivial one as it can be envisaged that the mapping from the complex network to the bipartite graph is not unique. An alternate way is to begin with a bipartite graph and convert it into a complex network. The challenge would then become ensuring that the complex network has a scale-free property. It is because the variable nodes are interconnected via the check nodes and hence it may not be possible to determine the properties of the resultant complex network when the check nodes are removed. To resolve the issue, we apply a theorem in [102], which states that if the degree distribution of one set of nodes in a bipartite graph follows a power-law distribution, the degree distribution of the unipartite graph (network) formed when the other set of nodes is removed also follows a power-law with the same exponent. In other words, if we can construct LDPC codes such that their variable-node degrees follow power-law distributions, the complex networks formed by the variable nodes alone will also follow power-law distributions with the same exponent. Consequently, the average path length between the variable nodes will be small which would enhance the convergence rate of the LDPC decoder.

We denote the probability that a variable node has k connections by $\Pr_\lambda(k)$. Define an LDPC code built on scale-free network as a scale-free LDPC (SF-LDPC) code. To construct a SF-LDPC code, we assign the fraction of variable nodes with degree k according to a power-law function, i.e., $\Pr_\lambda(k) \propto k^{-\gamma}$, where γ is the characteristic exponent for the variable-node degree. Since $\sum_k \Pr_\lambda(k) = 1$, the fraction of edges connecting to variable nodes with degree k can be readily shown

equal to

$$\lambda_k = \frac{k^{1-\gamma}}{\sum_{i=2}^{d_v} i^{1-\gamma}} \tag{4.1}$$

where d_v denotes the maximum variable-node degree. Then the variable-node degree distribution in (2.1) can be re-written as

$$\lambda(x) = \sum_{k=2}^{d_v} \frac{k^{1-\gamma}}{\sum_{i=2}^{d_v} i^{1-\gamma}} x^{k-1}. \tag{4.2}$$

In addition, the average variable-node degree, denoted by $< \ k_v \ >$, can be computed from

$$< k_v >= \frac{\sum_{k=2}^{d_v} k^{1-\gamma}}{\sum_{i=2}^{d_v} i^{-\gamma}}. \tag{4.3}$$

As for the check nodes, it has been well-known that the degrees of the check nodes should be kept almost the same in the design of good LDPC codes [11, 15]. Here, we restrict the check-node degrees to three consecutive integers, i.e., $d_c - 2, d_c - 1$ and d_c and that the check-node degrees are taken to follow a Poisson distribution with parameter ν. The advantage of such a model is that there are only two variables to manipulate — d_c and ν. Consolidating the above conditions, the probability that a check node has $k \in \{d_c - 2, d_c - 1, d_c\}$ connections, denoted by $\Pr_\rho(k)$, equals $\Pr_\rho(k) = \frac{\nu^k \exp(-\nu)}{k!}$. Then, the fraction of edges connecting to check nodes with degree k equals

$$\rho_k = \frac{\frac{\nu^k \exp(-\nu)}{(k-1)!}}{\sum_{j=d_c-2}^{d_c} \frac{\nu^j \exp(-\nu)}{(j-1)!}} \qquad k \in \{d_c - 2, d_c - 1, d_c\} \tag{4.4}$$

and the check-node degree distribution in (2.1) can be re-written as

$$\rho(x) = \sum_{k=d_c-2}^{d_c} \frac{\frac{\nu^k \exp(-\nu)}{(k-1)!}}{\sum_{j=d_c-2}^{d_c} \frac{\nu^j \exp(-\nu)}{(j-1)!}} x^{k-1}. \tag{4.5}$$

Combining the results in (4.2), (4.3) and (4.4), it can be readily shown that for

a given rate R,

$$\begin{aligned}\frac{< k_v >}{1-R} &= \frac{\sum_{k=d_c-2}^{d_c} \frac{\nu^k \exp(-\nu)}{(k-1)!}}{\sum_{j=d_c-2}^{d_c} \frac{\nu^j \exp(-\nu)}{j!}} \\ &= \frac{(d_c-2)(d_c-1)d_c + (d_c-1)d_c\nu + d_c\nu^2}{(d_c-1)d_c + d_c\nu + \nu^2}.\end{aligned} \tag{4.6}$$

Since d_c is an integer greater than 2, we can conclude that

$$d_c - 2 < \frac{< k_v >}{1-R} < d_c \tag{4.7}$$

and

$$d_c = \left\lceil \frac{< k_v >}{1-R} \right\rceil, \left\lceil \frac{< k_v >}{1-R} \right\rceil + 1. \tag{4.8}$$

When d_c is selected, the corresponding ν can also be found using (4.6).

4.2.2 Constrained scale-free LDPC codes

When LDPC codes are randomly constructed, there is a high probability that small cycles, say cycles of length less than 10, consisting of only degree-2 variable nodes are formed. Since these small-size cycles have ACE (approximate cycle extrinsic message degree) value of zero, they are detrimental to the decoder (more details are provided in Section 2.4.2). There are effective code construction algorithms, such as progressive edge growth (refer to Section 2.4.1), that can maximize the length of possible cycles involving only degree-2 variable nodes. However, if the number of degree-2 variable nodes, denoted by N_{v2}, is far larger than the check length M, the excess number of degree-2 nodes over the check length, i.e., $N_{v2} - M$ degree-2 variable nodes, will produce small-size cycles with ACE value of zero, giving rise to a high error rate. Therefore, in practice, the fraction of degree-2 variable nodes in any optimized degree distributions should not greatly exceed $\frac{M}{N} = 1 - R$. To overcome the aforementioned problem, an additional constraint has been proposed if the fraction of DE-optimized degree-2 variable nodes is far larger than $1 - R$ [92, 103–105]. The DE mechanism that has incorporated the degree-2 variable-node constraint is called "constrained DE", and the corresponding degree distribution obtained is called "constrained degree distribution".

Applying the above concept to the SF-LDPC codes, we can form "constrained SF-LDPC" codes. Denote the fraction of degree-2 variable nodes by $F_r(2) = \frac{N_{v2}}{N}$. For constrained SF-LDPC codes with a maximum variable-node degree of d_v, the variable-node degree distribution will be given as

$$\lambda_k = \begin{cases} \frac{2F_r(2)}{2F_r(2)+\frac{\sum_{i=3}^{d_v} i^{1-\gamma}(1-F_r(2))}{\sum_{i=3}^{d_v} i^{-\gamma}}} & \text{if } k = 2 \\ \frac{k^{1-\gamma}(1-F_r(2))}{2F_r(2)+\frac{\sum_{i=3}^{d_v} i^{1-\gamma}(1-F_r(2))}{\sum_{i=3}^{d_v} i^{-\gamma}}} & \text{otherwise.} \end{cases} \tag{4.9}$$

Using the same method as described in Section 4.2.1, the optimized check-node degree distribution, ν, d_c and $< k_v >$ can be readily found.

4.3 Performance of scale-free LDPC codes

4.3.1 Achievable Error Performance of SF-LDPC Codes

In this section, we present the analytical performance and the simulated results for SF-LDPC codes. First, we compare the achievable error-correcting capability (threshold) between SF-LDPC codes and other best-known LDPC codes [85]. We assume an AWGN channel and code rates of 0.5, 0.75 and 0.82.

4.3.1.1 Rate-0.5 SF-LDPC codes

First we consider the code of rate 0.5. Suppose the maximum variable-node degree equals 20, i.e., $d_v = 20$. We select a value for γ, say $\gamma = 2.0$. Based on (4.3), (4.8) and (4.6), we can find the values of d_c and μ, respectively. Table 4.2 shows the possible values of $< k_v >$, d_c as ν varies.

We then substitute the distributions of the variable nodes and check nodes into the DE algorithm and obtain the achievable error performance σ^* of the SF-LDPC codes. Figure 4.3 plots the value of σ^* as γ increases from 1.95 to 2.50. From the results, we observe that σ^* accomplishes a maximum value of 0.945 at $\gamma = 2.35$ and $d_c = 9$. Within the range of γ being considered, i.e., $[1.95, 2.50]$, note that $d_c = 8$ and $d_c = 10$ are, respectively, valid only in the ranges $\gamma \in [2.18, 2.50]$ and

Table 4.2: Possible values of d_c at different ranges of γ. $d_v = 20$ and $R = 0.5$.

Range of γ	Possible values of d_c
1.727–1.934	10,11
1.934–2.183	9,10
2.183–2.501	8,9
2.501–2.963	7,8
2.963–3.830	6,7

$\gamma \in [1.95, 2.18]$; whereas $d_c = 9$ is valid within the whole range (see Table 4.2 for reference).

Using the same methodology, we can evaluate the best achievable error performance (threshold) σ^* of the SF-LDPC codes with code rate $R = 0.5$ under different values of d_v. In Table 4.3, we list the highest thresholds achieved by SF-LDPC codes and the corresponding parameters used, alongside with the thresholds of other best-known LDPC codes [85]. It can be observed that in all cases, the largest threshold values σ^* for the SF-LDPC codes are comparable with those for other best-known LDPC codes (less than 2% difference). However, the average number of connections ($< k_v >$) for the SF-LDPC codes is significantly smaller (12 to 15% reduction) compared to those for other LDPC codes. Note that the fractions of degree-2 variable nodes shown in the table, i.e., $F_r(2)$, are less than or slightly larger than $1 - R$.

4.3.1.2 High rate SF-LDPC codes

We first use (4.2) to attempt optimizing the degree distributions for rate-0.75 and rate-0.82 SF-LDPC codes. However, the $F_r(2)$ obtained is far larger than $1-R$, i.e., 0.25 and 0.18. Hence, we resort to the constrained SF-LDPC codes for the rates 0.75 and 0.82. In addition, to ensure an easy implementation of the encoder, $F_r(2)$ is set equal to or slightly larger than $1 - R$ [92]. We begin with $F_r(2) = 1 - R$, and increase it with a step size of 0.001 until $(1 - R, 1 - R + 0.05)$ is reached. For each value of $F_r(2)$, the largest threshold and the corresponding constrained degree distributions of the SF-LDPC code are recorded. Among all the results, the largest

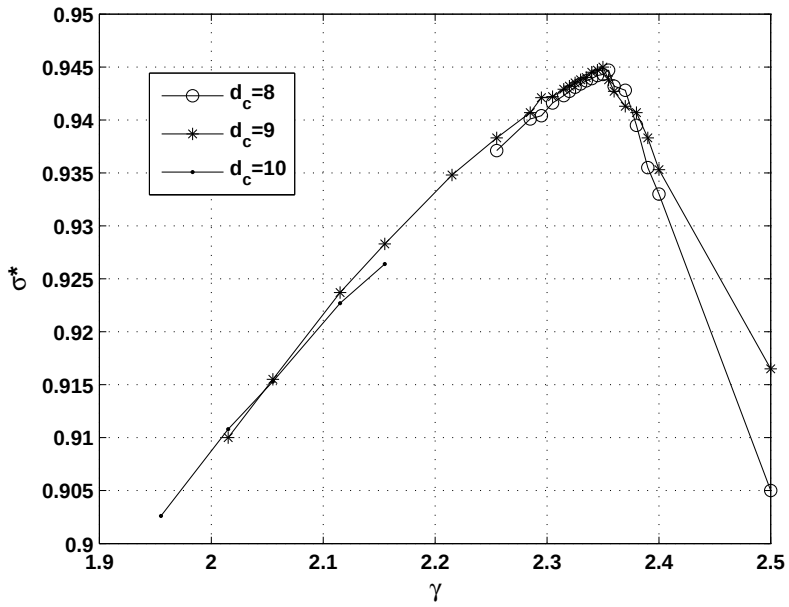


Figure 4.3: Achievable error performance (threshold) σ^* versus γ. $R = 0.5$ and $d_v = 20$.

threshold and the corresponding optimized, constrained degree distributions of the SF-LDPC code is then selected. Table 4.4 presents the highest thresholds achieved by rate-0.75 and rate-0.82 constrained SF-LDPC codes as well as other best-known high rate codes. The corresponding parameters used are also tabulated. The results indicate that the "pure" DE produces a slightly larger σ^* compared with other LDPC codes. However, "pure" DE also produces the fraction of degree-2 variable nodes almost two times of $(1-R)$, i.e., $F_r(2) \approx 2(1-R)$. We also observe that constrained DE and constrained SF-LDPC produce very similar σ^* and $< k_v >$.

4.3.2 Characteristics of Short-Length SF-LDPC Codes

Next, we form SF-LDPC codes of finite lengths using the parameters listed in Table 4.3. We select two codes randomly from the SF-LDPC code ensemble. The first one has a block length of $1,000$ (i.e., number of variable nodes $1,000$) and a maximum variable-node degree of $d_v = 15$ while the other one has a block length of $10,000$ and a maximum variable-node degree of $d_v = 20$. Then, we remove the check nodes and form complex networks with the remaining variable nodes, like the one shown in Fig. 4.2(b). Figure 4.4 shows the degree distributions of the

Table 4.3: Comparison of the threshold value and the average number of connections between SF-LDPC codes and other best-known LDPC codes. Code rate $R = 0.5$.

Common Parameters	d_v	10	15	20	30	50
Optimized Codes in [85]	Abbreviation	DE10	DE15	DE20	DE30	DE50
	σ^*	0.956	0.962	0.965	0.969	0.972
	$< k_v >$	3.663	4.009	4.164	4.496	5.077
	$F_r(2)$	0.461	0.477	0.458	0.441	0.435
SF-LDPC Codes	Abbreviation	SF10	SF15	SF20	SF30	SF50
	d_c	/	9	9	9	9
	γ	/	2.35	2.35	2.36	2.35
	ν	/	7.355	7.844	8.561	9.417
	σ^*	/	0.943	0.945	0.951	0.954
	$< k_v >$	/	3.512	3.719	3.972	4.304
	$F_r(2)$	/	0.505	0.497	0.493	0.489

complex variable-node networks of the two randomly selected codes. We observe that the degree distributions of the resultant network do follow power-laws with characteristic exponents equaling 2.35. Moreover, when the code length becomes longer, the scale-free property gets more prominent.

4.3.3 Error Performance of Short-Length SF-LDPC Codes

Further, the error performance of several short-length LDPC codes with code rates 0.5, 0.75 and 0.82 is studied. The variable nodes and the check nodes for each code are connected using the progressive-edge-growth (PEG) method, which has been shown to produce codes with both large girth and large Hamming distance [55] (see also Sect. 2.4.1).

4.3.3.1 Error Performance of Short-Length SF-LDPC Codes with Rate 0.5

We first consider the design of rate-0.5 codes. The first two code types, abbreviated by "DE15" and "DE10", are LDPC codes of which the degree distributions

Table 4.4: Comparison of the threshold value and the average number of connections between constrained SF-LDPC codes and other best-known LDPC codes. Code rate R equals 0.75 and 0.82. The letters in the code name denote the type of code, including DE, constrained DE (abbreviated by "CDE") , and constrained SF-LDPC (abbreviated by "CSF"). The digits in the code name denote the maximum variable node degree of the code.

Code Rate R	Code Name	d_v	$F_r(2)$	σ^*	$< k_v >$	d_c	γ
0.75	DE14	14	0.446	0.664	4.526	20	/
	CDE12	12	0.250	0.663	4.000	16	/
	CSF12	12	0.278	0.647	3.974	17	2.38
	CSF20	20	0.280	0.651	4.005	17	2.80
	CSF28	28	0.294	0.653	4.054	17	2.90
0.82	DE8	8	0.405	0.585	3.459	21	/
	CDE8	8	0.180	0.580	3.459	21	/
	CSF10	10	0.176	0.577	3.468	21	3.80
	CSF14	14	0.191	0.579	3.465	21	3.95

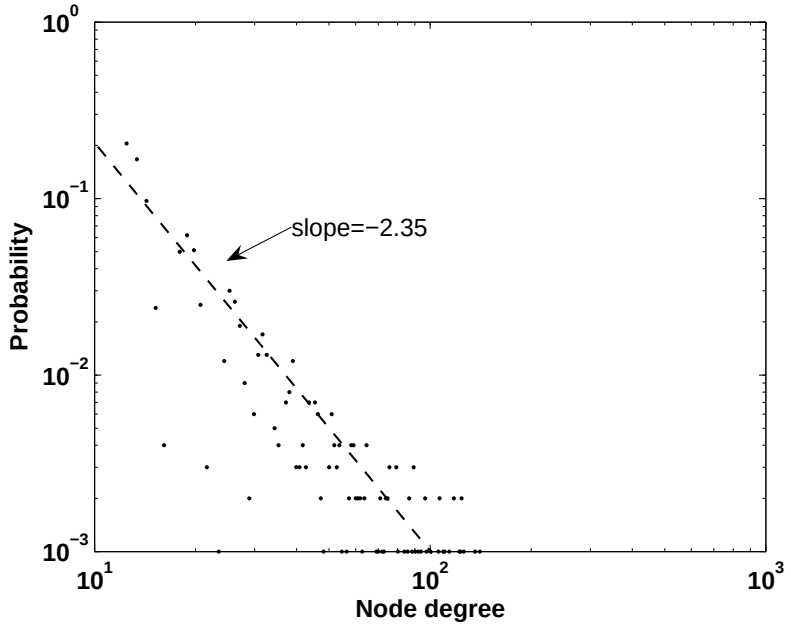
are optimized by purely the DE algorithm [85]. The corresponding variable-node degree distributions are given by

$$\begin{aligned}\lambda_1(x) &= 0.23802x + 0.20907x^2 + 0.03492x^3 + 0.12015x^4 \\ &\quad + 0.01587x^6 + 0.00480x^{13} + 0.37627x^{14} \end{aligned} \tag{4.10}$$

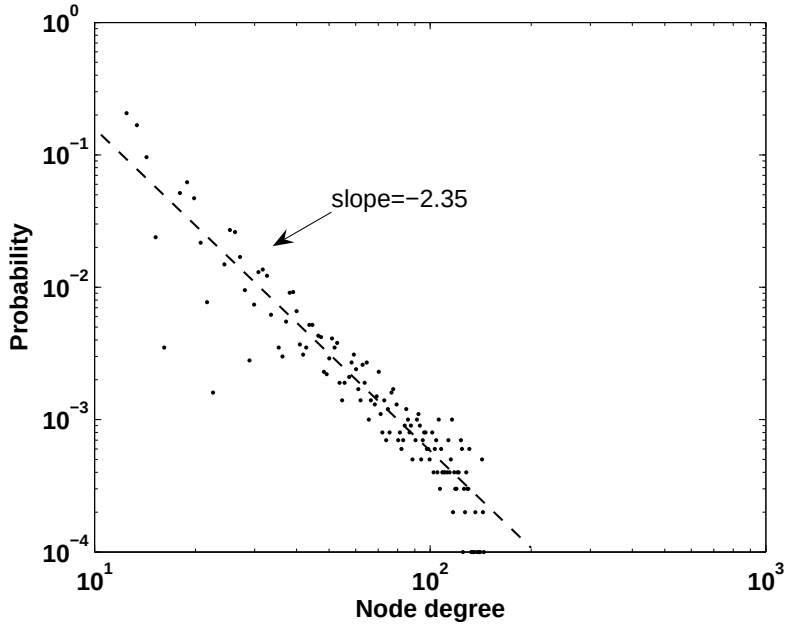
and

$$\begin{aligned}\lambda_2(x) &= 0.27165x + 0.25105x^2 + 0.30938x^3 + 0.00104x^4 \\ &\quad + 0.43853x^9, \end{aligned} \tag{4.11}$$

and they have maximum variable-node degrees 15 and 10, respectively. The other two codes, abbreviated by "SF20" and "SF30", are our proposed SF-LDPC codes. Details of all the aforementioned codes have been given in Table 4.3. The fractions of degree-2 variable nodes for the four codes are 47.71%, 46.06%, 49.74% and 49.32%, respectively, and are about $(1-0.5)$. Three different code lengths are used — 2016, 1008 and 504, while the code rate is kept at 0.5. For the codes denoted by "DE15" (except the one with code length 2016), we directly apply the codes in [97],



(a)



(b)

Figure 4.4: The degree distributions of the complex variable-node networks of the two randomly selected codes. (a) Block length $1,000$ and maximum variable-node degree 15; (b) block length $10,000$ and maximum variable-node degree 20.

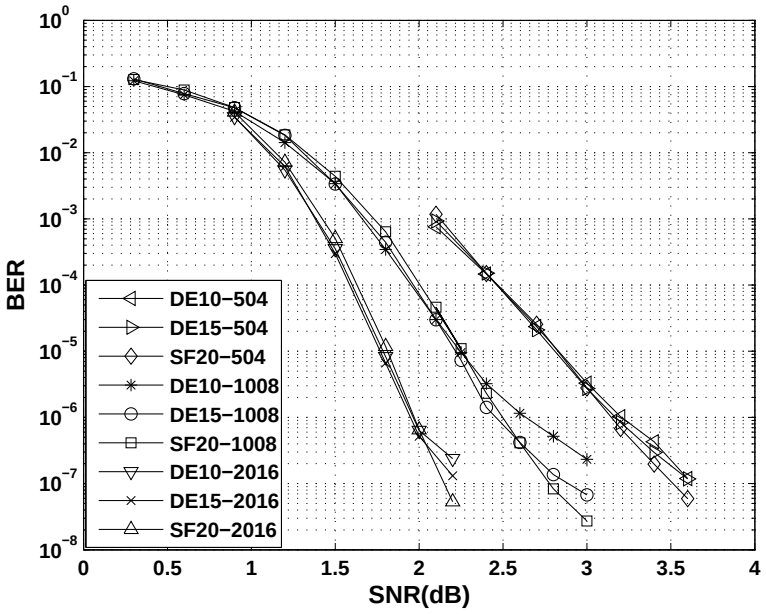
which are the best known LDPC codes in terms of error performance that possess

the properties listed in Table 4.3.

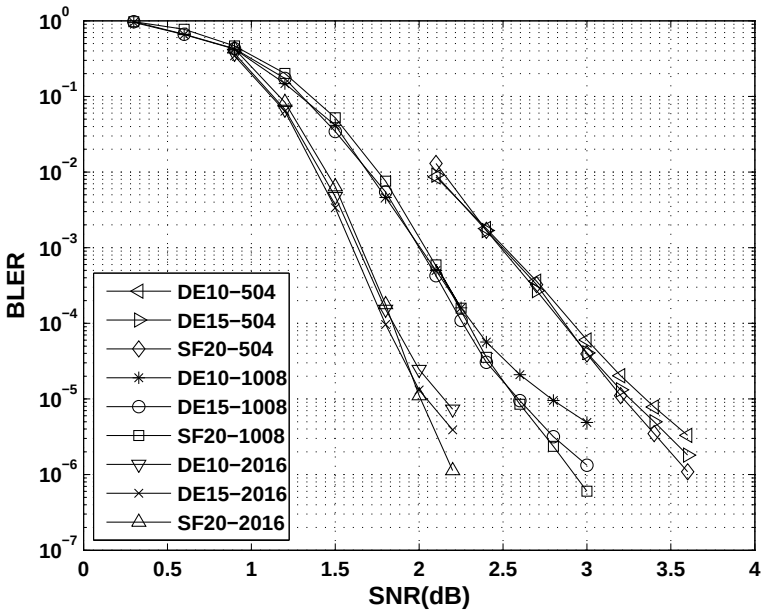
For a given code, define the average path length (APL) of the corresponding bipartite graph as the path length between any two variable nodes averaged over the whole bipartite graph. Consider codes with the same code length and the same average variable node degree. The code graphs with smaller APLs are more efficient in spreading information than those with large APLs. Similarly, under same code length and the same APL, code graphs with smaller average variable-node degree can be regarded as less complex. To measure the merit of a given code, we define the average-path-length-variable-node-degree-product (APVP) of a code as the product of the average path length of the bipartite graph and the average variable-node degree of the code. In general, codes with smaller APVPs are preferred. Table 4.5 shows the average variable-node degrees ($< k_v >$), APLs and APVPs of the four types of codes under study. It is observed that increasing the maximum variable-node degrees will decrease the APLs of SF-LDPC codes, i.e., code "SF30" has smaller APLs than code "SF20". However, the corresponding APVPs increase due to the proportionally larger increase in the average variable-node degrees. In particular, code "SF20" has lower APVPs compared to code "DE15" and similar APVPs compared to code "DE10"; whereas code "SF30" has almost identical APVPs as code "DE15".

In Figs. 4.5(a) and (b), we plot the simulated bit error rates (BERs) and block error rates (BLERs), respectively, for codes "DE10", "DE15" and "SF20". We choose code "SF20" as a representative of the SF-LDPC codes for performance comparison since it has lower APVP than code "SF30". The maximum number of iterations performed to decode one codeword is limited to 50 and the decoding process will be terminated once the maximum number is reached. It can be observed that the SF-LDPC codes "SF20" provide similar BER and BLER performance as "DE10" and "DE15" codes at low SNR, and outperform them at higher SNR values. In addition, Fig. 4.5(c) depicts that SF-LDPC codes "SF20" can be decoded with a slightly smaller number of iterations, on average, compared with the other DE-optimized codes.

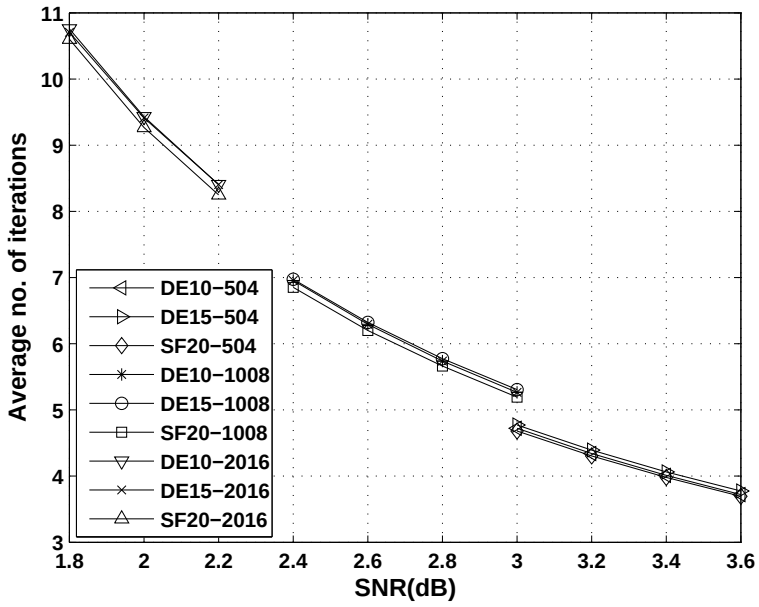
To further compare the performance of the codes, we define the metric "average convergence time", denoted by t_c, as the product of the average number of iterations to converge ($\bar{I}$) and the average variable-node degree ($< k_v >$). In gen-


BER
SNR(dB)
DE10−504
DE15−504
SF20−504
DE10−1008
DE15−1008
SF20−1008
DE10−2016
DE15−2016
SF20−2016

(a)


BLER
SNR(dB)
DE10−504
DE15−504
SF20−504
DE10−1008
DE15−1008
SF20−1008
DE10−2016
DE15−2016
SF20−2016

(b)



(c)

Figure 4.5: Performance of three different types of LDPC codes — "DE10", "DE15"and "SF20". Code lengths are 2016, 1008 and 504 while the code rate is 0.5. (a) Bit error rate; (b) block error rate; (c) average number of iterations to decode a codeword (t_c).

Table 4.5: Comparison of several rate-0.5 codes. APL: average path length; APVP: average-path-length-variable-node-degree-product.

		Code length		
Name of Code	Properties	504	1008	2016
DE10	$< k_v >$	3.651	3.658	3.662
	APL	2.256	2.495	2.693
	APVP	8.237	9.127	9.862
DE15	$< k_v >$	3.996	4.001	4.006
	APL	2.110	2.346	2.565
	APVP	8.432	9.386	10.275
SF20	$< k_v >$	3.704	3.710	3.715
	APL	2.217	2.462	2.670
	APVP	8.208	9.134	9.919
SF30	$< k_v >$	3.968	3.972	3.976
	APL	2.121	2.344	2.565
	APVP	8.416	9.310	10.198

eral, the smaller the "average convergence time", the less time the decoder takes to decode a codeword. Table 4.6 shows the typical results for the "DE10", "DE15" and "SF20" codes for the code lengths 504, 1008 and 2016. It indicates that "DE10" and "SF20" have almost identical "average convergence time" while "DE15" requires, on average, 10% more time (resources) to decode a codeword. The results are consistent with the fact that "DE10" and "SF20" have similar APVPs while "DE15" has the largest APVPs. However, note that "SF20" produces less errors than "DE10" and "DE15" at higher SNR values

4.3.3.2 Error Performance of Short-Length SF-LDPC Codes with Rate 0.75

Further, three codes of rate-0.75 are constructed using the PEG method. The first one has a DE-optimized variable-node degree distribution given by

$$\lambda_3(x) = 0.1970x + 0.0.0801x^2 + 0.2410x^3 + 0.0082x^4 + 0.4736x^{13}, \quad (4.12)$$

Table 4.6: Comparison of average convergence times for rate-0.5 codes.

SNR /Code length	3.6 dB/504	3.0 dB/1008	2.2 dB/2016
Code type	DE10/DE15/SF20	DE10/DE15/SF20	DE10/DE15/SF20
$\bar{I}$	3.72/3.78/3.69	5.27/5.31/5.19	8.41/8.41/8.25
$< k_v >$	3.65/4.00/3.70	3.66/4.00/3.71	3.66/4.01/3.71
$t_c = \bar{I} \times < k_v >$	13.59/15.09/13.68	19.26/21.23/19.26	30.78/33.67/30.64
Normalized t_c	0.99/1.10/1.00	1.00/1.10/1.00	1.00/1.10/1.00

and the second one possesses a constrained DE-optimized variable-node degree distribution given by [92, 103]

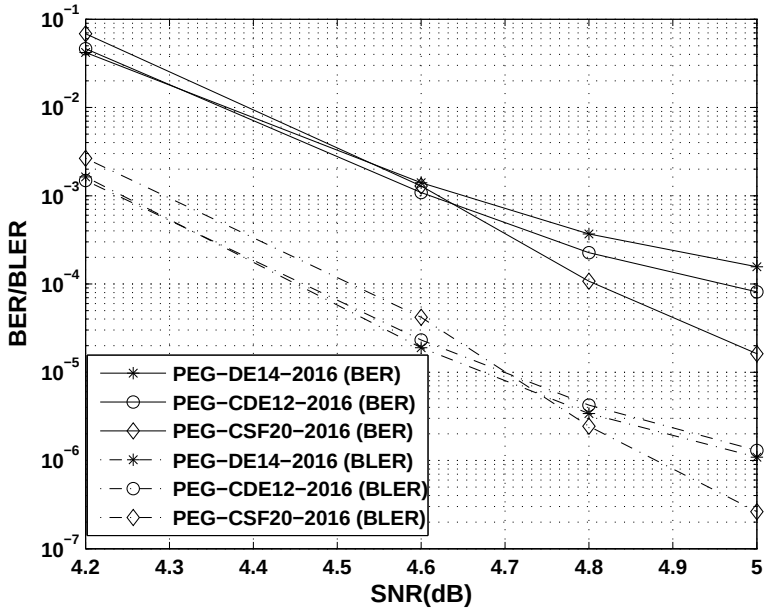
$$\lambda_4(x) = 0.1250x + 0.4460x^2 + 0.4078x^{11} + 0.0213x^{11}. \tag{4.13}$$

They are abbreviated as "DE14" and "CDE12", respectively. The third one is a constrained SF-LDPC code with a maximum variable-node degree of 20 and is denoted by "CSF20". Details of the aforementioned three codes are listed in Table 4.4. Moreover, the lengths of the three types of codes are set to be 2016.

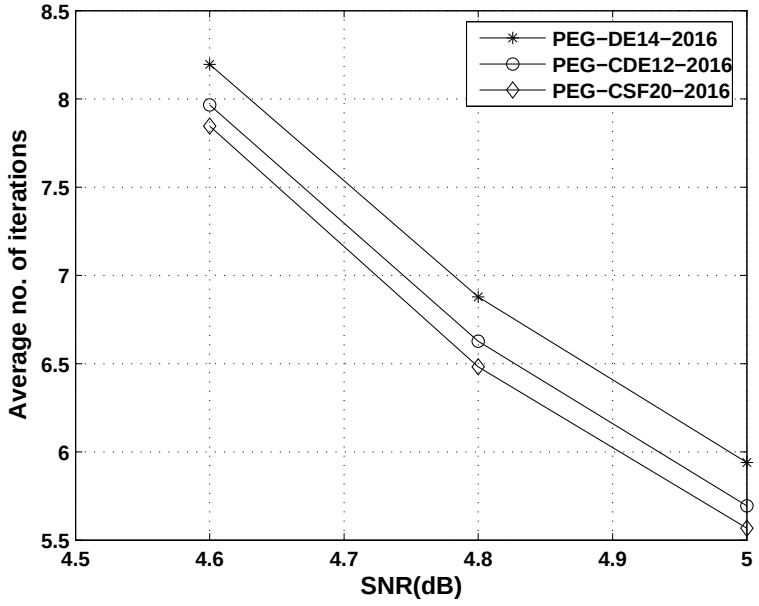
The APLs and the APVPs of the three types of codes are listed in Table 4.7. It is observed that even under the constraint in the fraction of degree-2 variable nodes, "CSF20" and "CDE12" have similar APVPs and have lower APVPs compared with "DE14".

Figure 4.6 plots the simulated BERs, BLERs as well as $\bar{I}$ for the three rate-0.75 codes. The performance curves in Fig. 4.6 (a) show that the constrained SF-LDPC code "CSF20" suffers slight degradation of BER and BLER performance compared with "DE14" and "CDE12" at low SNR, but outperform them at higher SNR values. In addition, Fig. 4.6(b) indicates that "CSF20" requires, on average, a slightly smaller number of iterations for decoding compared with the DE-optimized and constrained DE-optimized codes.

In particular, the results in Table 4.8 indicates that "CDE12" and "CSF20" have almost identical "average convergence time" (t_c) while "DE14" requires, on average, 20% more time (resources) to decode a codeword. The results are consistent with the fact that "CSF20" and "CDE12" have similar APVPs and have smaller



(a)



(b)

Figure 4.6: Performance of three different LDPC codes — "DE14", "CDE12"and "CSF20". Code lengths are 2016 and the code rate is 0.75. (a) BER and BLER; (b) average number of iterations to decode a codeword ($\bar{I}$).

Table 4.7: Comparison of APVPs of several rate-0.75 and rate-0.82 codes.

Code rate	Abbreviation	$< k_v >$	APL	APVP
0.75	DE14-2016	4.526	2.059	9.319
	CDE12-2016	4.000	2.161	8.644
	CSF20-2016	4.005	2.154	8.627
0.82	DE8-2800	3.459	2.263	7.828
	CDE8-2800	3.459	2.255	7.800
	CSF14-2800	3.465	2.250	7.796

APVPs than "DE14".

4.3.3.3 Error Performance of Short-Length SF-LDPC Codes with Rate 0.82

Finally, we study rate-0.82 codes built on (i) constrained SF-degree distribution with a maximum variable-node degree of 14 (denoted by "CSF14"); (ii) DE-optimized variable-node degree distribution given by (denoted by "DE8")

$$\lambda_5(x) = 0.2343x + 0.0.3406x^2 + 0.2967x^6 + 0.1284x^7; \tag{4.14}$$

and (iii) constrained DE-optimized variable-node degree distribution given by [92, 103] (denoted by "CDE8")

$$\lambda_6(x) = 0.1021x + 0.5895x^2 + 0.1829x^6 + 0.1262x^7. \tag{4.15}$$

Details of the codes are listed in Table 4.4. The lengths of the three types of codes are also set to be 2800.

The results in Table 4.7 have shown that "CSF14" and "CDE8" have APLs and APVPs extremely close to each other, and have APLs and APVPs slightly less than those of "DE8". Figure 4.7 further plots the BLER, BER and $\bar{I}$ of the three codes. We can observe that the constrained SF-LDPC code "CSF14" slightly underperforms compared with "DE8" and "CDE8" at low SNR but it achieves the lowest error floor at higher SNR values among the three codes. Moreover, Fig. 4.7(b) shows that "CSF14" can decode the codewords with the smallest number of itera-

Table 4.8: Comparison of the average convergence times for rate-0.75 codes at high SNR values.

SNR/Code length	5.0 dB/2016	4.8 dB/2016	4.6 dB/2016
Code type	DE14/CDE12/CSF20	DE14/CDE12/CSF20	DE14/CDE12/CSF20
$< k_v >$	4.53/4.00/4.01	4.53/4.00/4.01	4.53/4.00/4.01
$\bar{I}$	5.94/5.69/5.57	6.88/6.63/6.48	8.20/7.97/7.85
$t_c = \bar{I} \times < k_v >$	26.91/22.76/22.34	31.17/26.52/25.98	37.15/31.88/31.48
Normalized t_c	1.20/1.02/1.00	1.20/1.02/1.00	1.18/1.01/1.00

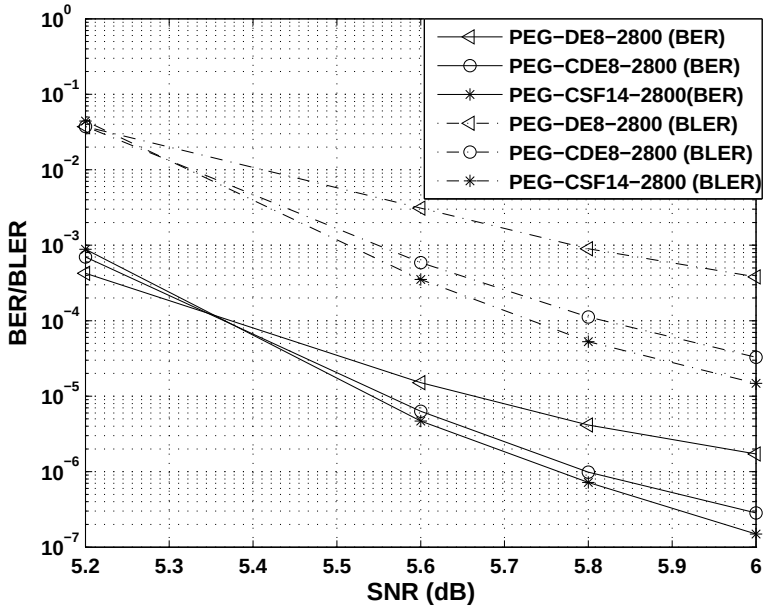
tions on average.

We also list the "average convergence time" (t_c) of the three rate-0.82 codes at high SNRs in Table 4.9. The results show that code "CDE8" and code "CSF14", compared with code "DE8", require 10% less time (resources) on average to decode a codeword.

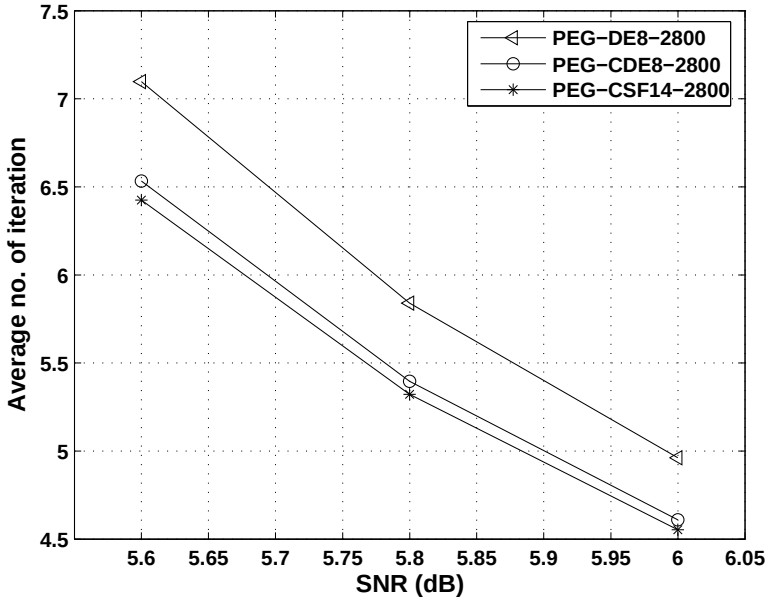
4.4 Summary

In this chapter, we have applied complex-network theories to design short-length LDPC codes with good error performance. We have shown that theoretically, our proposed scale-free LDPC (SF-LDPC) codes and constrained SF-LDPC codes can accomplish very similar achievable error performance (threshold) compared with DE-optimized and constrained DE-optimized LDPC codes. We have also built the codes and simulated their error performance under an additive white Gaussian noise channel. The results have shown that SF-LDPC codes can outperform other DE-optimized codes, producing lower block/bit error rates at high SNRs. Moreover, the average convergence times of the SF-LDPC codes are no worse than those of the DE-optimized codes. Similar conclusions can also be drawn regarding the constraint SF-LDPC codes.

In this chapter, we further observe that at the high SNR region, all the codes suffer from error-floor problem, i.e., the BLER/BER decreases at a slower rate as the SNR increases. In the next chapter, we will follow up this issue and investigate



(a)



(b)

Figure 4.7: Performance of three different LDPC codes — "DE8", "CDE8"and "CSF14". Code lengths are 2800 and the code rate is 0.82. (a) BER and BLER; (b) average number of iterations to decode a codeword ($\bar{I}$).

Table 4.9: Comparison of average convergence times for rate-0.82 codes at high SNR values.

SNR/Code length	6.0 dB/2800	5.8 dB/2800	5.6 dB/2800
Code type	DE8/CDE8/CSF14	DE8/CDE8/CSF14	DE8/CDE8/CSF14
$< k_v >$	3.46/3.46/3.46	3.46/3.46/3.46	3.46/3.46/3.46
$\bar{I}$	4.96/4.61/4.55	5.84/5.40/5.32	7.10/6.53/6.42
$t_c = \bar{I} \times < k_v >$	17.16/15.95/15.74	20.21/18.68/18.41	24.57/22.59/22.21
Normalized t_c	1.09/1.01/1.00	1.10/1.01/1.00	1.11/1.02/1.00

the main cause of the error floor. We will also propose a novel method to construct short-length LDPC codes with lower error floors.

Chapter 5

Constructing short-length LDPC codes with low error floor

In the previous chapter, our simulation results have shown that at the high signal-to-noise ratio (SNR) region, the block/bit error rate (BLER/BER) of the LDPC codes decrease at a slower rate as the SNR is further increased, indicating that the existence of the error floor. As a consequence, in this chapter, we will continue our study by looking into the main cause of the error floor and proposing construction methods to mitigate the error-floor problem.

In [26], it has been reported that $[w; u]$ trapping sets (TSs) with small w and relatively smaller u are more harmful to the decoder. To begin our investigation, we will first analyze the connected subgraphs induced by the TSs and categorize them into those with (i) no cycle, (ii) a single-cycle and (iii) multiple cycles. We will also study the properties of these induced connected subgraphs and show that using $[w; u]$ alone is not adequate to describe the features of the TSs, particularly features that are crucial to determine the harmfulness of the TSs. We will introduce a new parameter, namely cycle indicator (CI), that further characterizes TSs. Moreover, we define primary trapping sets (PTSs), the main use of which is to identify detrimental TSs. Based on our findings, we propose a code construction algorithm, namely the PEG-ACSE method, that aims to avoid detrimental TSs. We will compare the characteristics of the codes built using the proposed method with those built using other PEG-based algorithms.

5.1 Connected Subgraphs Induced by Trapping Sets

Given a TS labeled as $[w; u]$. Recall that w denotes the number of variable nodes in the TS and u represents the number of neighboring check nodes having odd number of connections to the TS. Considering the connected subgraph induced by the TS, we define

- d_i as the degree of the ith variable node;
- $N_{c,2k-1}$ and $N_{c,2k}$, respectively, as the number of check nodes having $2k-1$ and $2k$ ($k = 1, 2, \ldots$) connection(s);
- N_c as the total number of check nodes;
- $M_{v,\text{edges}}$ as the total number of connections emanating from the variable nodes;
- $M_{c,\text{edges}}$ as the total number of connections emanating from the check nodes.

Based on the aforementioned definitions, we can readily obtain the following equations.

$$
\begin{aligned}
u &= \sum_k N_{c,2k-1} && (5.1)\\
N_c &= \sum_k N_{c,2k-1} + \sum_k N_{c,2k} && (5.2)\\
&= u + \sum_k N_{c,2k} && (5.3)\\
M_{v,\text{edges}} &= \sum_{i=1}^{w} d_i && (5.4)\\
M_{c,\text{edges}} &= \sum_k (2k-1) N_{c,2k-1} + \sum_k 2k N_{c,2k} && (5.5)\\
&= u + \sum_k (2k-2) N_{c,2k-1} + \sum_k 2k N_{c,2k} && (5.6)
\end{aligned}
$$

In addition, the total number of edges emanated from the check nodes should equal the total number of edges from the variable nodes in the subgraph, giving

$$M_{v,\text{edges}} = M_{c,\text{edges}}$$

$$\Rightarrow \quad \sum_{i=1}^{w} d_i = \sum_{k} (2k-1) N_{c,2k-1} + \sum_{k} 2k N_{c,2k} \tag{5.7}$$

$$\Rightarrow \quad u = \sum_{i=1}^{w} d_i - \sum_{k} (2k-2) N_{c,2k-1} - \sum_{k} 2k N_{c,2k}. \tag{5.8}$$

We further consider all possible configurations of the connected subgraph induced by the $[w; u]$ TS. Broadly speaking, we can categorize the TSs according to the number of cycles contained in their induced connected subgraphs (ICSs). In our following discussion, we will divide TS-ICSs into those consisting of (i) no cycle, (ii) a single cycle and (iii) multiple cycles. *We do not consider cases where the TS-ICSs containing a single cycle or multiple cycles possess variable nodes not involved in the cycles. For these cases, we shall discard the variable nodes not involved in the cycle(s) and investigate only on the TS-ICSs formed by the variable nodes involved in the cycle(s).*

5.1.1 No Cycle

Suppose there is no cycle in the connected subgraph induced by a $[w; u]$ TS. We can arbitrarily select a variable node in the TS-ICS as the root node and expand its connections into various layers until all nodes are included. Figure 5.1(a) presents the TS-ICS in a tree-like form after the expansion.

Assume that there is a total of $2L$ layers in the subgraph. Denote the number of variable nodes at Layer $2l-1$ by $N_v^{(2l-1)}$ and the number of check nodes at Layer $2l$ by $N_c^{(2l)}$, where $l = 1, 2, \ldots, L$. Denote $d_j^{(2l-1)}$ as the degree of the jth variable node at Layer $(2l-1)$ for $l = 1, 2, \ldots, L$. At Layer 1, there is only one variable node (i.e., $N_v^{(1)} = 1$) and all of its edges are connected to the check nodes in Layer 2. Next, for the jth variable node in Layer $(2l-1)$ ($l = 2, 3, \ldots, L$), one of its edges is connected to a check node in Layer $(2l-2)$ while the remaining $d_j^{(2l-1)} - 1$ edges are connected to check nodes in Layer $2l$. Since each check node in Layer $2l$ is connected to one and only one variable node in Layer $(2l-1)$ ($l = 1, 2, \ldots, L$), the number of check nodes in Layer $2l$ is given by

$$N_c^{(2l)} = \begin{cases} d_1^{(1)} N_v^{(1)} = d_1^{(1)} & \text{for } \; l = 1 \\ \sum_{j=1}^{N_v^{(2l-1)}} \left(d_j^{(2l-1)} - 1 \right) & \text{for } \; l = 2, \ldots, L. \end{cases} \tag{5.9}$$

Then the total number of check nodes in the subgraph can be found by summing up the number of the check nodes at all layers, i.e.,

$$\begin{aligned} N_c &= \sum_{l=1}^{L} N_c^{(2l)} \\ &= N_c^{(2)} + \sum_{l=2}^{L} N_c^{(2l)} \\ &= d_1^{(1)} + \sum_{l=2}^{L} \sum_{j=1}^{N_v^{(2l-1)}} \left(d_j^{(2l-1)} - 1 \right) \\ &= 1 + \sum_{l=1}^{L} \sum_{j=1}^{N_v^{(2l-1)}} \left(d_j^{(2l-1)} - 1 \right) \\ &= 1 + \sum_{i=1}^{w} (d_i - 1) \,. \end{aligned} \tag{5.10}$$

In (5.10), the last equation holds because the second terms in both the last and the second last equations represent the sum of all resultant variable-node degrees when one is subtracted from the degree of each variable node in the TS-ICS.

Combining (5.3) and (5.10), we have

$$u = 1 + \sum_{i=1}^{w} (d_i - 1) - \sum_{k} N_{c,2k}. \tag{5.11}$$

Finally, multiplying (5.11) by 2 and subtracting (5.8) from it, we obtain

$$u = 2 + \sum_{i=1}^{w} (d_i - 2) + \sum_{k} (2k-2)(N_{c,2k-1} + N_{c,2k}). \tag{5.12}$$

5.1.2 Single cycle

Assuming that there is a single-cycle in the TS-ICS, all the check nodes must then have degrees no greater than two, i.e., $N_{c,2k-1} = N_{c,2k} = 0$ for $k \geq 2$, as shown in Fig. 5.1(b). Otherwise, more than one cycle will exist in the TC-ICS, which contradicts our assumption. As a result, there is no odd-degree check nodes with degree larger than or equal to three. Therefore, (5.1) is reduced to $u = N_{c,1}$. Moreover, each of the variable nodes will have consumed two edges connecting to

two different check nodes in the cycle and will connect its remaining edges to check nodes singly connected to the TS (i.e., check nodes outside the cycle). As shown in Fig. 5.1(b), the number of singly-connected check nodes equals the number of variable-node edges not included in the cycle. Therefore,

$$u = N_{c,1} = \sum_{i=1}^{w} (d_i - 2). \tag{5.13}$$

5.1.3 Multiple cycles

In a TS-ICS that contains multiple cycles, we define a node that is connected to two or more different cycles as a "T-type node". A T-type node can be a variable node or a check node, and must have a minimum of three edges. Figure 5.2 demonstrates some typical TS-ICSs that contain multiple cycles. In the following, we discuss these cases in more detail.

5.1.3.1 All T-type nodes are variable nodes

As in Fig. 5.2(a) and (b), when all the T-type nodes are variable nodes, the check nodes in the TS-ICS cannot connect to three or more variable nodes, i.e., $N_{c,2k-1} = N_{c,2k} = 0$ for $k \geq 2$. Thus, all the check nodes must have degree one or two. Further, (5.1) will be reduced to $u = N_{c,1}$. Denoting the number of edges emanating from the variable nodes and involved in the cycles as $M_{v,\text{cycles}-\text{edges}}$, we have

$$M_{v,\text{cycles}-\text{edges}} = \sum_{i=1}^{w} d_i - u. \tag{5.14}$$

Moreover, each variable node in the TS-ICS must contribute at least two edges to form the multiple cycles. Also, there must exist (i) at least one variable T-type node that have four or more edges involved in the cycles (see Fig. 5.2(a)); or (ii) at least two variable T-type nodes, each of which contributes three or more of its edges to the cycles (see Fig. 5.2(b)). Since the total number of variable nodes equals w, the total number of edges emanating from these nodes and involved in the multiple

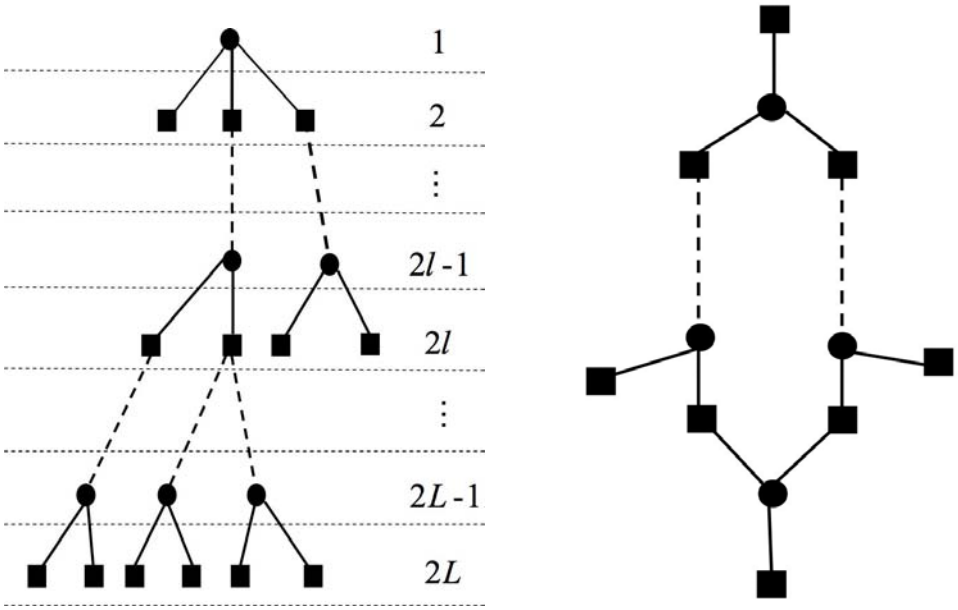


Figure 5.1: (a) A TS-induced connected subgraph with (a) no cycle; (b) a single cycle. Filled circles and filled squares represent, respectively, variables nodes and check nodes. A solid line represents a direct connection between the variable nodes and the check nodes. A dashed line depicts that there is a path between the two nodes.

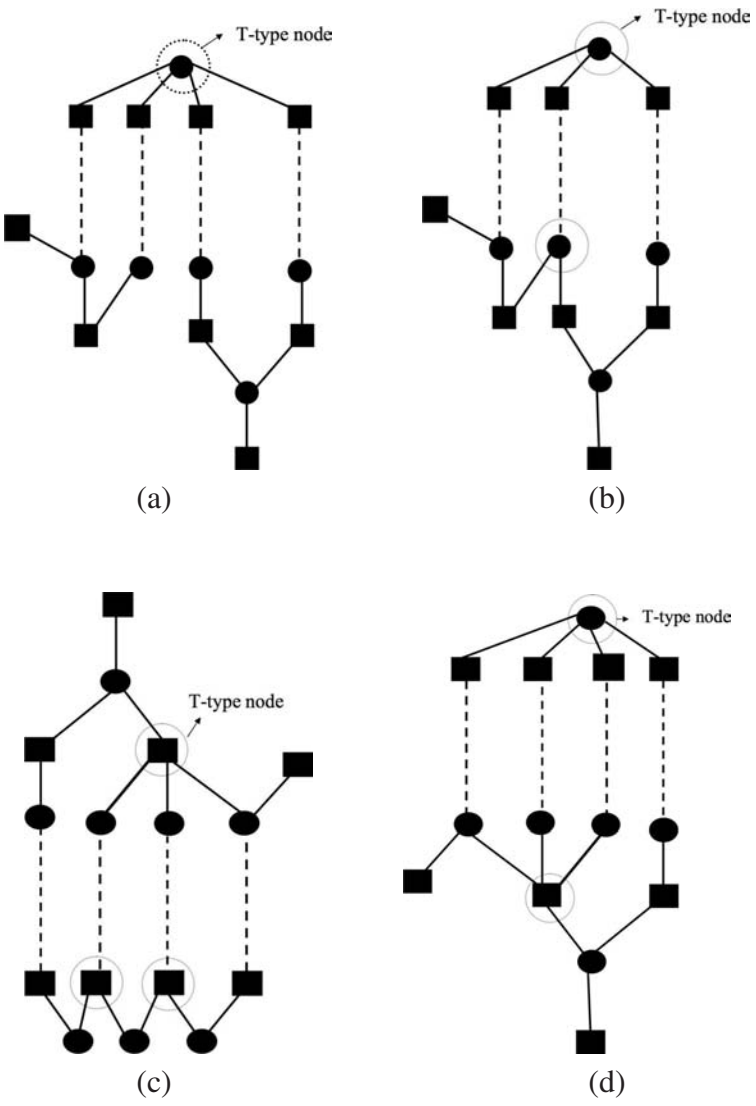


Figure 5.2: Trapping set-induced connected subgraphs containing multiple cycles. (a), (b) All T-type nodes are variable nodes; (c) all T-type nodes are check nodes; (d) T-type nodes involve both variable node(s) and check node(s).

cycles must exceed $2w$ by at least two, i.e.,

$$M_{v,\text{cycles}-\text{edges}} \geq 2w + 2. \tag{5.15}$$

Combining (5.14) and (5.15), we obtain

$$\begin{aligned} & \sum_{i=1}^{w} d_i - u \geq 2w + 2 \\ \Rightarrow \quad & \sum_{i=1}^{w} (d_i - 2) - u \geq 2. \end{aligned} \tag{5.16}$$

5.1.3.2 All T-type nodes are check nodes

In Fig. 5.2(c), we show the case when all the T-type nodes are check nodes. Under such circumstances, each variable node in the TS-ICS can contribute only two of its edges to form the multiple cycles while the remaining edges will connect to singly-connected check nodes. Hence, the number of edges emanating from the variable nodes and connecting to singly-connected check nodes equals

$$\begin{aligned} \sum_{i=1}^{w} (d_i - 2) &= N_{c,1} \\ &\leq \sum_{k} N_{c,2k-1} \\ &= u. \end{aligned} \tag{5.17}$$

5.1.3.3 T-type nodes involve both variable node(s) and check node(s)

Figure 5.2(d) illustrates the scenario when the T-type nodes involve both variable node(s) and check node(s). Here, we cannot deduce any useful information out of this configuration.

5.1.4 Regular LDPC Codes

For regular LDPC codes, all the variable-node degrees are identical and must be no less than three. Denoting the common variable-node degree by d_v, we have

$$d_i = d_v \geq 3 \quad \text{for} \quad i = 1, 2, \ldots, w. \tag{5.18}$$

Based on (5.18), we evaluate the characteristics of TS-ICSs for regular LDPC codes.

5.1.4.1 No Cycle

Substituting (5.18) into (5.12), we obtain

$$\begin{aligned} u &= 2 + w(d_v - 2) + \sum_k (2k-2)(N_{c,2k-1} + N_{c,2k}) \\ \Rightarrow u - w &= 2 + w(d_v - 3) + \sum_k (2k-2)(N_{c,2k-1} + N_{c,2k}) \\ \Rightarrow w - u &\leq -2 < 0. \end{aligned} \tag{5.19}$$

The equality $w - u = -2$ holds only when $d_v = 3$ and no check nodes with degree greater than two exists in the TS-ICS (i.e., $N_{c,2k-1} = N_{c,2k} = 0$ for $k \geq 2$).

5.1.4.2 Single Cycle

Substituting (5.18) into (5.13), we have

$$\begin{aligned} u &= w(d_v - 2) \\ \Rightarrow w - u &= -w(d_v - 3) \leq 0 \end{aligned} \tag{5.20}$$

in which equality holds only when $d_v = 3$.

5.1.4.3 Multiple cycles

- *All T-type nodes are variable nodes*

  Putting (5.18) into (5.16), we obtain

$$w(d_v - 2) - u \geq 2$$

$$\Rightarrow \qquad w - u \;\geq\; -w(d_v - 3) + 2. \tag{5.21}$$

Note that to satisfy (5.21), $w-u$ can be negative (e.g., $w = 4, u = 6, d_v = 4$), positive (e.g., $w = 4, u = 2, d_v = 3$) or even zero (e.g., $w = u = 4, d_v = 4$).

- *All T-type nodes are check nodes*

  The equation in (5.17) can be written as

$$\begin{aligned} w(d_v - 2) &\leq u \\ \Rightarrow \qquad w - u &\leq -w(d_v - 3) \leq 0. \end{aligned} \tag{5.22}$$

  Note that $w - u = 0$ occurs only when $d_v = 3$ and there are no odd-degree check nodes with degree larger than or equal to three in the TS-ICS.

- *T-type nodes involve both variable node(s) and check node(s)*

  No useful information can be obtained.

5.1.5 Irregular LDPC Codes

Unlike regular LDPC codes in which the smallest possible degree of variable nodes is three, the smallest possible degree of variable nodes in the case of irregular codes is only two, i.e., $d_i \geq 2$ for all i.

5.1.5.1 No Cycle

Subtracting w from both sides of (5.12), we obtain

$$\begin{aligned} u - w &= 2 + \sum_{i=1}^{w}(d_i - 3) + \sum_{k}(2k-2)(N_{c,2k-1} + N_{c,2k}) \\ \Rightarrow w - u &= -2 - \sum_{i=1}^{w}(d_i - 3) - \sum_{k}(2k-2)(N_{c,2k-1} + N_{c,2k}). \end{aligned} \tag{5.23}$$

Note that to satisfy (5.23), $w - u$ can be negative (e.g., $w = 3, u = 7, d_1 = 3, d_2 = d_3 = 4$ and $N_{c,2k-1} = N_{c,2k} = 0$ for $k \geq 2$, as shown in Fig. 5.3(a)), positive (e.g., $w = 3, u = 2, d_i = 2$ for $i = 1, 2, 3$ and $N_{c,2k-1} = N_{c,2k} = 0$ for $k \geq 2$, as

shown in Fig. 5.3(b)) or even zero (e.g., $w = 3, u = 3, d_1 = 3, d_2 = d_3 = 2$ and $N_{c,2k-1} = N_{c,2k} = 0$ for $k \geq 2$, as shown in Fig. 5.3(c)).

5.1.5.2 Single Cycle

Subtracting w from both sides of (5.13), we have

$$\begin{aligned} u - w &= \sum_{i=1}^{w} (d_i - 3) \\ \Rightarrow w - u &= -\sum_{i=1}^{w} (d_i - 3). \end{aligned} \tag{5.24}$$

Again, to satisfy (5.24), $w - u$ can be negative (e.g., $w = 3, u = 4, d_1 = 4$ and $d_2 = d_3 = 3$ as shown in Fig. 5.4(a)), positive (e.g., $w = 3, u = 1, d_1 = 3$ and $d_2 = d_3 = 2$ as shown in Fig. 5.4(b)) or even zero (e.g., $w = 3, u = 3$ and $d_1 = d_2 = d_3 = 3$ as shown in Fig. 5.4(c)).

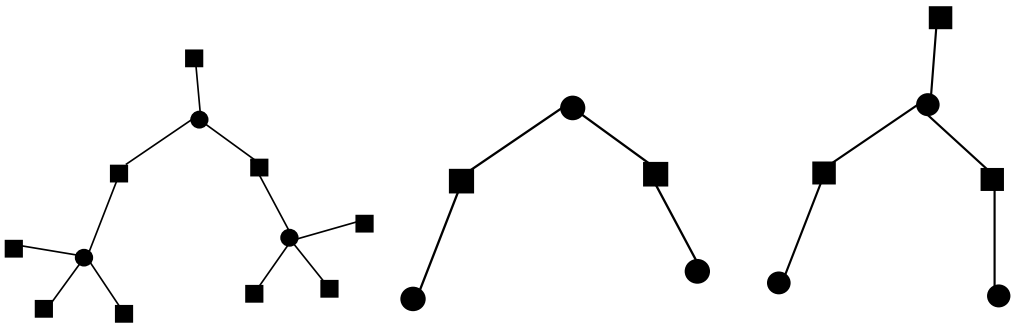
5.1.5.3 Multiple Cycle

- *All T-type nodes are variable nodes*

  The equation in (5.16) can be rewritten as

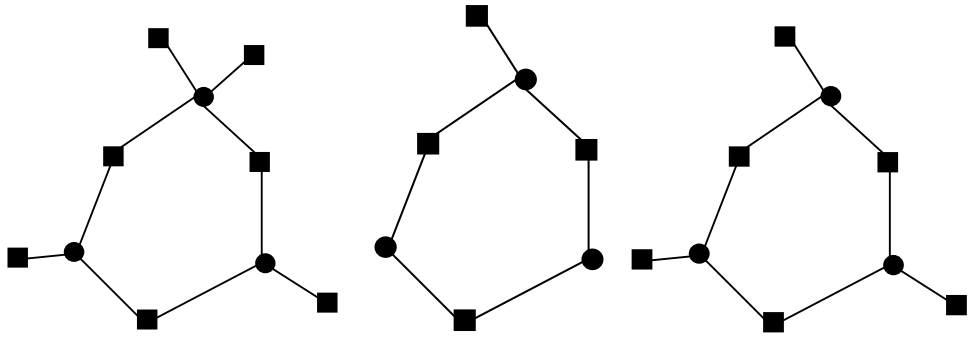
$$\begin{aligned} u - w &\leq \sum_{i=1}^{w} (d_i - 3) - 2 \\ \Rightarrow w - u &\geq -\sum_{i=1}^{w} (d_i - 3) + 2. \end{aligned} \tag{5.25}$$

  However, no useful information can be obtained from the sign of $w - u$ using (5.25). Take a TS-ICS with only two T-type variable nodes as an example. Suppose each T-type variable node contributes exactly three edges to the multiple cycles. Under such conditions, $w - u$ can be negative (e.g., $w = 4, u = 5, d_1 = d_3 = d_4 = 4$ and $d_2 = 3$ as shown in Fig. 5.5(a)); positive (e.g., $w = 4, u = 0, d_1 = d_2 = 2$ and $d_3 = d_4 = 3$ as shown in Fig. 5.5(b)) or even zero (e.g., $w = 4, u = 4, d_1 = d_2 = 3$ and $d_3 = d_4 = 4$ as shown in Fig. 5.5(c)).



(a) (b) (c)

Figure 5.3: Examples of subgraphs containing no cycle with (a) $w = 3$, $u = 7$; (b) $w = 3$, $u = 2$; and (c) $w = 3$, $u = 3$.



(a) (b) (c)

Figure 5.4: Examples of subgraphs containing a single cycle with (a) $w = 3$, $u = 4$; (b) $w = 3$, $u = 1$; and (c) $w = 3$, $u = 3$.

- *All T-type nodes are check nodes*

  The equation in (5.17) can be rewritten as

$$\begin{aligned} u - w &\geq \sum_{i=1}^{w} (d_i - 3) \\ \Rightarrow w - u &\leq -\sum_{i=1}^{w} (d_i - 3). \end{aligned} \tag{5.26}$$

  Again, no useful information can be deduced from the sign of $w - u$ in (5.26) because of the existence of degree-two variable nodes. For example, in a subgraph with $w = 4$, $d_1 = d_2 = d_3 = 2$ and two T-type check nodes with degree 3, $w - u$ is negative when $u = 5$ and $d_4 = 5$ (as shown in Fig. 5.6(a)); positive when $u = 3$ and $d_4 = 3$ (as shown in Fig. 5.6(b)); and zero when

$u = 4$ and $d_4 = 4$ (as shown in Fig. 5.6(c)).

- *T-type nodes involve both variable node(s) and check node(s)*

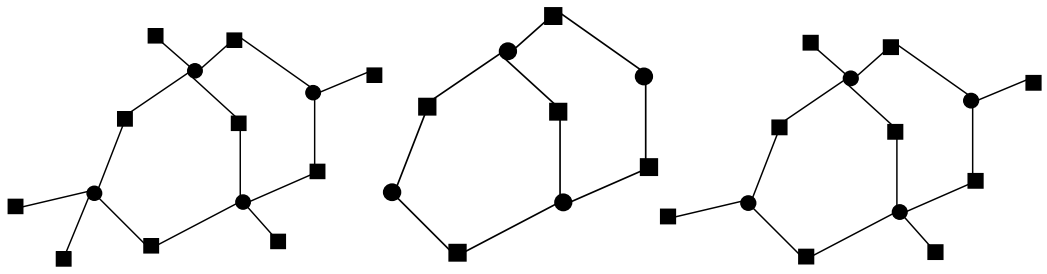
  No useful information can be obtained. Take a subgraph with $w = 4$, one T-type variable node of degree 3 and one T-type check node of degree 3 as an example. The value of $w - u$ can be negative if $u = 5, d_1 = d_2 = d_3 = 3$ and $d_4 = 4$ (as shown in Fig. 5.7(a)); or positive if $u = 1, d_1 = d_2 = d_3 = 2$ and $d_4 = 3$ (as shown in Fig. 5.7(b)); or zero if $u = 4, d_1 = 2$, $d_2 = d_3 = 3$ and $d_4 = 4$ (as shown in Fig. 5.7(c)).

Based on the observations, no useful information can be obtained regarding the configuration of the TS-ICS based on the values of w and u alone.

5.1.6 Observations

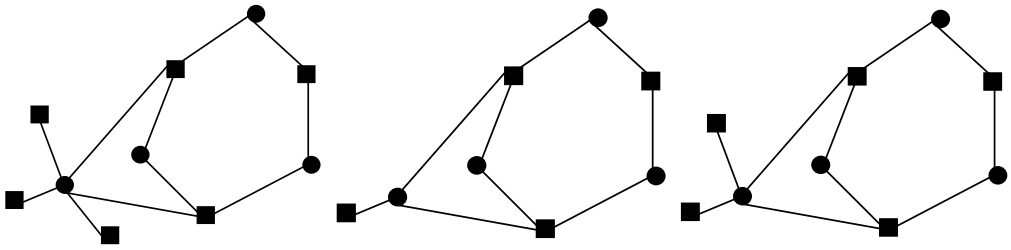
Given a $[w; u]$ TS found from a *regular* LDPC code. If $w > u$, (5.19) and (5.20) imply that it is not possible for the connected subgraph induced by this TS to contain no cycle or a single cycle. In other words, the TS-ICS must possess multiple cycles. Further, based on (5.22), we can conclude that these multiple cycles must include at least one T-type variable node. However, if it is given that $u > w$, no concrete conclusions on the configuration of the TS-ICS can be made.

As for the *irregular* LDPC codes, no useful information can be obtained regarding the configuration of the TS-ICS even given the values of w and u of the TS.



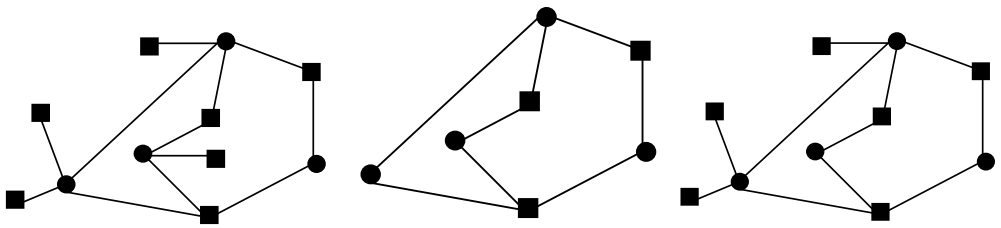
(a) (b) (c)

Figure 5.5: Examples of subgraphs containing multiple cycles involving only T-type variable nodes with (a) $w = 4$, $u = 5$; (b) $w = 4$, $u = 0$; and (c) $w = 4$, $u = 4$.



(a) (b) (c)

Figure 5.6: Examples of subgraphs containing multiple cycles involving only T-type check nodes with (a) $w = 4$, $u = 5$; (b) $w = 4$, $u = 3$; and (c) $w = 4$, $u = 4$.



(a) (b) (c)

Figure 5.7: Examples of subgraphs containing multiple cycles involving both T-type check nodes and T-type variable nodes with (a) $w = 4$, $u = 5$; (b) $w = 4$, $u = 1$; and (c) $w = 4$, $u = 4$.

5.2 Contributions of $[w; u]$ Trapping Sets to Error Floor

In the previous section, we have shown that knowing the values of w and u in a $[w; u]$ TS does not provide us much information on the configuration of the associated TS-ICS, particularly for irregular codes. Yet, $[w; u]$ TSs with different TS-ICS configurations can contribute very different error rates to the overall performance of the code at the high SNR region. For example, it has been found that short cycles with few connections to other nodes are likely contributors to small-size stopping sets, which in turn degrade the performance of short-length LDPC codes over an AWGN channel at high SNR [57]. The finding has implied that TS-ICSs with a single cycle are contributing to the error floor to a certain extent. On the other hand, to the best of our knowledge, no literature has reported that TS-ICSs with no cycles are related to the error-floor issue. Our own simulation results have also shown that TS-ICSs with one or more cycles are the main contributors to the error floor. Figure 5.8 shows three different configurations of the induced connected subgraphs of a $[9; 1]$ TS. It is obvious that each of these configurations will contribute differently to the error floor.

Consider the scenario that an all-zero codeword is received with a high SNR and assume that the likelihood-message-passing algorithm is used in the iterative decoder [11, 15]. Without loss of generality, we assume that as the decoder iterates, the message along an edge should approach zero if it carries the reliable information that the "variable-node bit" equals "0". Otherwise, the message should approach infinity when it conveys the fallacious information that the bit equals "1". Suppose that during the course of the belief propagation (BP) decoding process, the variable nodes in a $[w; u]$ TS-ICS have been mis-decoded as bits "1" while those residing outside the TS-ICS have been decoded correctly as bits "0". Recall that there are u check nodes with odd number of connections in the TS-ICS. The check equations corresponding to these check nodes are hence "unsatisfied" because each of these check nodes is connected to an odd number of variable nodes decoded with bits "1". In the following, we briefly explain the phenomena for several scenarios as the BP decoder continues to iterate.

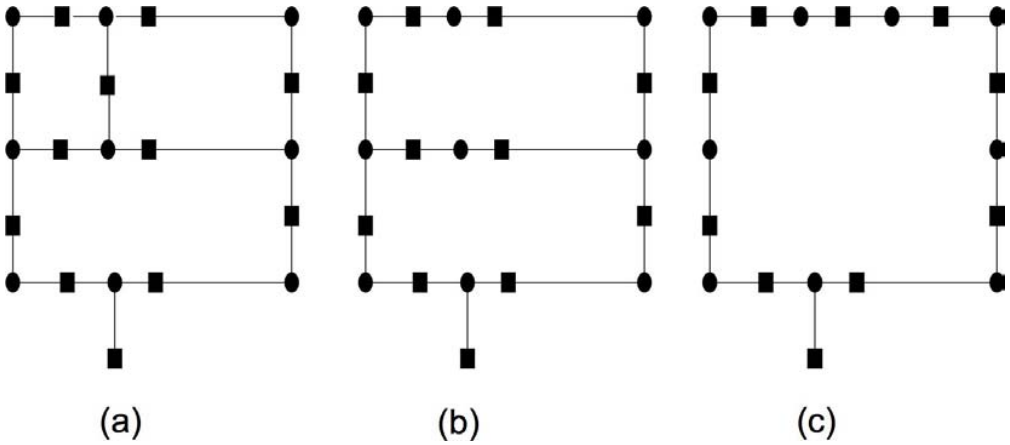


Figure 5.8: Three different induced connected subgraphs of a $[9; 1]$ trapping set. (a) A $[9; 1; 4]$ PTS-ICS contains three distinguishable cycles; (b) a $[9; 1; 2]$ PTS-ICS contains two distinguishable cycles; (c) a $[9; 1; 0]$ PTS-ICS contains a single cycle. Filled circles and filled squares represent, respectively, variables nodes and check nodes. A solid line represents a direct connection between the variable nodes and the check nodes.

5.2.1 TS-ICS With No Cycle

For a TS-ICS that contains no cycles, the erroneous message starting from a particular variable node cannot propagate back to itself via only the connections within the TS-ICS because of the non-existence of a return path. In consequence, the erroneous bits in the TS-ICS are more likely to be corrected by the valid information flowing into the TS-ICS through the unsatisfied check nodes.

5.2.2 TS-ICS with a Single Cycle

Referring to Fig. 5.9(a), suppose there is a single cycle in a $[w; u]$ TS-ICS that possesses no singly-connected check nodes, i.e., $u = 0$[1]. Then, when most, even if not all, of the variable nodes in the cycle are erroneous (decoded as "1"), the (erroneous) message starting from any of the variable nodes will be enhanced as it passes through each of the erroneous variable nodes in the cycle. After a number of iterations (equal to the number of variable nodes in the cycle), the enhanced message will return to the same variable node, further reinforcing the incorrect belief of the

[1] In this scenario, the TS is also a stopping set.

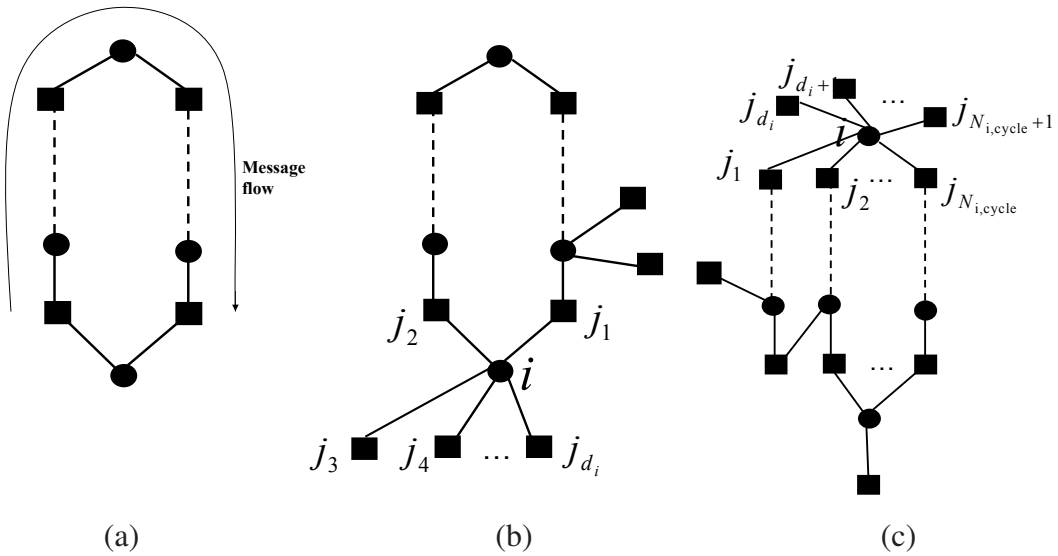


Figure 5.9: Trapping set-induced connected subgraphs. (a) A single-cycle TS-ICS without singly-connected check nodes (another message flow in an anti-clockwise direction); (b) a single-cycle TS-ICS with singly-connected check nodes; (c) a multiple-cycle TS-ICS with singly-connected check nodes.

variable node. The end result is that all variable nodes in the cycle (or TS-ICS) will become erroneous. Moreover, if all the variable nodes residing outside the TS-ICS have been decoded correctly as bits "0", all the check nodes will become satisfied and the decoder converges — to an incorrect codeword though. In consequence, $[w; 0]$ TS-ICSs with small w are very likely to give rise to errors even at the high SNR region.

On the other hand, when there are check nodes singly connected to the variable nodes in a $[w; u]$ TS-ICS with a single cycle, i.e., $u > 0$, erroneous bits in the TS-ICS may be corrected by the valid information flowing into the TS-ICS through the unsatisfied check nodes. Take the Variable Node i in Fig. 5.9(b) for example. During the iterative decoding process, the outgoing message from Variable Node i to Check Node j_1 equals the product of the following items.

1. The initial message computed for Variable Node i using the corrupted received signal.

2. The incoming message from the other neighboring check node involved in

the cycle, i.e., from Check Node j_2.

3. The incoming messages from the (unsatisfied) check nodes outside the cycle, i.e., from Check Node j_3 to Check Node j_{d_i}[2].

The initial message is of the order $O(1)$ and is relatively insignificant compared with the other items. The message from the neighboring Check Node j_2 involved in the cycle, however, carries the fallacious information "1" and bears a very large value. On the other hand, each of the incoming messages from the (unsatisfied) check nodes outside the cycle carries the reliable information "0" and bears a very small value. Thus, the reliable information "0" coming from the (unsatisfied) check nodes outside the cycle can "neutralize" or even "overcome" the fallacious information from Check Node j_2 to Variable Node i, allowing Variable Node i to produce a reasonably correct message to Check Node j_1. Recall that in a $[w; u]$ TS-ICS that contains a single cycle, there are u check nodes singly connected to the variable nodes in the cycle. In consequence, with a larger value of u, there is a larger number of "reliable messages" coming from check nodes outside the cycle, making it more likely that the errors in the cycle can be corrected. In contrast, suppose there is/are only one or two singly-connected check node(s) in the TS-ICS with a single cycle. Then, not only the incorrect messages within the cycle may not be corrected, but also be relayed to nodes outside the cycle, causing other nodes to become erroneous during the iterative decoding process. Under such circumstances, the decoding process may not converge, thus giving rise to the error floor.

5.2.3 TS-ICS with Multiple Cycles

In the case of a TS-ICS with multiple cycles, we consider the scenario where only T-type variable nodes exist. It is because multiple-cycle TS-ICS with T-type check nodes have not been found contributing much to the error floor [56]. Consider the T-type Variable Node i with degree d_i ($d_i \geq 3$) in the TS-ICS shown in Fig. 5.9(c). Among the d_i edges of the variable node, suppose $d_{i,\text{cycle}}$ of them are involved in the multiple cycles. Similar to the single-cycle case, during the iterative decoding process, the outgoing message from Variable Node i to Check Node j_1

[2]Recall that d_i represents the degree of Variable Node i.

equals the product of (i) the initial message computed for Variable Node i using the corrupted received signal; (ii) the incoming messages from the other neighboring check nodes involved in the multiple cycles, i.e., from Check Node j_2 to Check Node $j_{d_{i,\text{cycle}}}$; and (iii) the incoming messages from the (unsatisfied) check nodes outside the cycles, i.e., from Check Node $j_{d_{i,\text{cycle}}+1}$ to Check Node j_{d_i}.

The initial message is relatively insignificant. If the number of check nodes that convey the fallacious information "1" (i.e., neighboring Check Nodes j_2 to $j_{d_{i,\text{cycle}}}$) to Variable Node i is larger than the number of check nodes that pass the reliable information "0" (i.e., neighboring Check Nodes $j_{d_{i,\text{cycle}}+1}$ to j_{d_i}) to Variable Node i , there is a high chance that the message passing from Variable Node i to Check Node j_1 carries the fallacious information "1". Furthermore, if u is small, there will be a small number of singly-connected check nodes attached to the variable nodes in the cycles, indicating a high chance that the aforementioned scenario is occurring. Compared to the case of a single cycle, there are multiple paths by which the incorrect message can pass back to Variable Node i in a multiple-cycle TS-ICS. Thus, under the same set of $\{w, u\}$, a TS with multiple cycles is more detrimental than that with a single cycle. Conversely, if the number of check nodes that convey the fallacious information "1" to Variable Node i is smaller, the messages carrying the reliable information are more likely to overcome those carrying the fallacious information. Thus, the message from Variable Node i to Check Node j_1 will become more reliable. In addition, same as the single-cycle case, the augmented fallacious information within the multiple cycles can be relayed to the nodes outside the cycles through the singly-connected check nodes. Under such conditions, the decoding process may oscillate and is unable to converge, causing the error floor.

5.2.4 Summary

In this section, we have analyzed the reasons why $[w; u]$ TSs with the same w and u may produce very different effect to the error floor. Based on the findings, we can conclude that tackling the error-floor issue by avoiding $[w; u]$ TSs with small w and u during the code-construction process is not the best strategy. Instead, we should focus on avoiding $[w; u]$ TSs that are more harmful (those with one or more cycle(s) in the corresponding TS-ICSs) rather than avoiding **all** $[w; u]$ TSs with the

same w and u. However, determining whether there is a cycle and also the number of cycles in a TS-ICS can be very complicated and time consuming. It will be very useful if there is a simple way of evaluating the configuration of a TS-ICS.

5.3 Refined $[w; u; e]$ Trapping Set

5.3.1 Cycle Indicator

To determine whether there is a cycle and also the number of cycles in a TS-ICS, we introduce a parameter called "cycle indicator (CI)", which is denoted by e and is computed by

$$e = \left[\sum_{i=1}^{w}(d_i - 2)\right] - u. \tag{5.27}$$

Based on the results in Sect. 5.1, we evaluate the values of e under different scenarios and present the results in Table 5.1. The results indicate that regardless of regular or irregular codes, for a TS-ICS with

- no cycle, $e < 0$;
- a single cycle, $e = 0$;
- multiple cycles, T-type variable nodes but no T-type check nodes, $e > 0$;
- multiple cycles, T-type check nodes but no T-type variable nodes, $e \leq 0$.

Still, no useful comments can be drawn for the case of TS-ICS with multiple cycles and both T-type variable node(s) and T-type check node(s). With the introduction of the CI, we can refine the definition of a TS.

5.3.2 Trapping Set with Label $[w; u; e]$

Definition 5.1 *($[w; u; e]$ trapping set) A $[w; u; e]$ trapping set is defined as a set of variable nodes of size w, with u neighboring check nodes having odd numbers of connections to the set, and the cycle indicator e computed using (5.27) where d_i represents the degree of the ith variable node in the set.*

Table 5.1: Types of trapping set-induced connected subgraph (TS-ICS) and the corresponding cycle indicator values e. The LDPC codes can be regular or irregular.

Type of TS-ICS	**Cycle indicator** e
No cycle	$e < 0$
Single cycle	$e = 0$
Multiple cycles and all T-type nodes are variable nodes	$e > 0$
Multiple cycles and all T-type nodes are check nodes	$e \leq 0$
Multiple cycles and T-type nodes involve both variable node(s) and check node(s)	$e <=> 0$

Furthermore, research reports [56] and our own simulation results have indicated that the decoding errors at the high SNR region are caused mainly by TSs with their ICSs containing either (i) a single cycle, or (ii) multiple cycles with T-type variable nodes but no T-type check nodes. In contrast, TSs with their ICSs containing multiple cycles and T-type check nodes have not been found contributing significantly to the error floor. In order to focus our attention to TSs that are contributing to the error floor, we define a primary TS (PTS) as follows.

5.3.3 Primary trapping set

Definition 5.2 *($[w; u; e]$ primary trapping set) A $[w; u; e]$ primary trapping set (PTS) is defined as a $[w; u; e]$ TS where no check node of degree three or larger exists in the induced connected subgraph (ICS) of the TS if the TS-ICS contains two or more cycles. In other words, the induced connected subgraph of a PTS containing multiple cycles cannot have any T-type check nodes. Moreover, we define the size of a $[w; u; e]$ PTS-ICS as the number of variable nodes the subgraph contains, i.e., w.*

In particular, a $[w; 0; e]$ PTS represents a codeword with a hamming weight of w. Consider a binary vector. Let the non-zero bits in the vector correspond to all the w variable nodes in the $[w; 0; e]$ PTS. In this case all the parity check equations

of the code will be satisfied and thus the vector forms a codeword of weight w.

Note that for a $[w; u; e]$ PTS-ICS with one or more cycle(s), the CI also provides an insight into the exact number of cycles existing in the PTS-ICS. We rewrite (5.27) as

$$e = \left(\sum_{i=1}^{w} d_i \right) - 2w - u. \tag{5.28}$$

The first term on the RHS of (5.28), $\sum_{i=1}^{w} d_i$, represents the total degree of all the variable nodes in the $[w; u; e]$ PTS-ICS, which is equivalent to the total number of connections. The second term, $2w$, reflects the total number of connections that the w variable nodes need to contribute when forming a single cycle. The last term, u, denotes the number of singly-connected check nodes in the $[w; u; e]$ PTS-ICS (as all check nodes in such a PTS-ICS have degrees either one or two). Since $2w + u$ also represents the number of connections in a PTS-ICS containing a single cycle and u singly-connected check nodes, e can be viewed as the number of "extra connections" used to build multiple cycles based on an existing single-cycle PTS-ICS. It can be readily shown that for every two "extra connections", an additional "distinguishable cycle" will be formed. Here, we describe a set of cycles as "distinguishable" if each cycle in the set contains one or more variable or check node(s) that all other cycles in the set do not possess. Then, we can conclude that a $[w; u; e]$ PTS-ICS contains $e/2 + 1$ "distinguishable" cycles when $e \geq 0$ (recall that $e < 0$ when a PTS-ICS contains no cycle).

Fig. 5.8 presents examples of $[9; 1; 4]$, $[9; 1; 2]$ and $[9; 1; 0]$ PTS-ICSs. When there are more "distinguishable" cycles in a PTS-ICS, there are more chances that an erroneous message being enhanced in the PTS-ICS, making the decoder harder to converge to the correct value. Thus, if $0 \leq e_1 < e_2$, a $[w; u; e_2]$ PTS-ICS will contribute more to the error floor at high SNR than a $[w; u; e_1]$ PTS-ICS. We further define a "detrimental PTS-ICS" as a single-cycle or multiple-cycle $[w; u; e]$ PTS-ICS with small w and u. Detrimental PTS-ICSs are more harmful to the decoder. Therefore, to build codes with low error floor, an effective way is to construct codes by avoiding detrimental PTS-ICSs.

5.3.4 Observations and summary

In this section, we have refined the trapping set by the label $[w; u; e]$ where e is the newly introduced cycle-indicator (CI). We have defined a "primary trapping set (PTS)" for use in identifying the possible harmful trapping sets. By looking into the definitions of TS and PTS, we can see that PTS is just a special case of TS.

Figure. 5.10 further presents the relations among stopping set, TS, dominant TS, PTS and detrimental PTS. We use the circle to represent the set of $[w; u; e]$ trapping sets. The circle has been divided into five sub-regions, denoted by A, B, C, D and E, to represent the TSs with their ICSs having (a) no cycle; (b) single cycle; (c) multiple cycles with all T-type nodes being variable nodes; (d) multiple cycles with all T-type nodes being check nodes and (e) multiple cycles involving both T-type variable nodes and T-type check nodes, respectively. We also denote the set of $[w; u; e]$ TSs with small w and relatively smaller u, i.e., the dominant $[w; u; e]$ TSs, by the striped area in the unit cycle. Recall that a stopping set is a set of variable nodes with the neighboring check nodes having more than one connections to the set of variable nodes. The definition indicates that the ICS of a stopping set should contain (i) a single cycle or multiple cycles with check nodes of degree only two or (ii) multiple-cycles with no check nodes of degree less than two and at least one T-type check node. Based on the aforementioned features, we make use of the grey areas within B, C, D and E in the figure to represent the set of stopping sets.

Since a PTS is a TS with no check nodes of degree three or larger if the TS-ICS contains two or more cycles, the set of PTSs can be denoted by union of areas A, B and C. Subsequently, the set of detrimental PTSs can be denoted by the striped areas within B and C. Further, it is interesting to see that the grey areas within B and C represent the set of $[w; 0; e]$ PTSs. It is because stopping sets of size w with their ICSs containing no odd degree check nodes are actually $[w; 0; e]$ PTSs, which indicate the existence of codewords of weight w. Thus small-size stopping sets might give rise to low weight codewords, which are extremely harmful to the decoder.

Based on the aforementioned observations, we can conclude that not all the dominant trapping sets as well as all the small-size stopping sets are contributors to the error events of LDPC decoder at high SNR. To exactly describe such contribu-

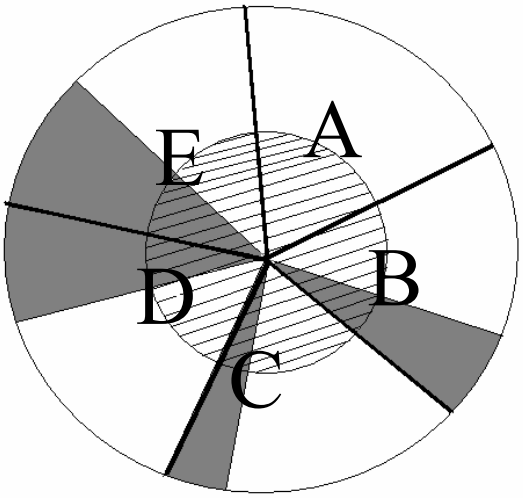


Figure 5.10: Graphic illustration of relations among TS, PTS, detrimental PTS and stopping set. The areas A, B, C, D, and E in the circle represent separately TSs with their ICSs containing (a) no cycle; (b) single cycle; (c) multiple cycles with all T-type nodes being variable nodes; (d) multiple cycles with all T-type nodes being check nodes and (e) multiple cycles involving both T-type variable nodes and T-type check nodes. Grey area: stopping sets. $A \cup B \cup C$: PTS. Grey area $\cup B \cup C$: $[w; 0; e]$ PTS.

tors, we should resort to the PTS and detrimental PTS.

5.4 Proposed code-construction method

Based on the previous findings, we will propose a code construction method, called *PEG-ACSE algorithm*, that aims to avoid $[w; u; e]$ PTS-ICS with small w, small u and $e \geq 0$.

5.4.1 Cycle set EMD (CSE)

To reduce the occurrences of detrimental PTS-ICSs, we introduce a new metric called "cycle set extrinsic message degree (CSE)" and make use of it when constructing codes.

Definition 5.3 *(Cycle set (CS)) A cycle set consists of one or more cycle(s) linked together by a common (root) variable node. The size of a cycle set, denoted by w', is defined as the total number of variable nodes it contains.*

Definition 5.4 *(Cycle set EMD (CSE)) The CSE of a cycle set, denoted by u', is defined as the number of extrinsic-message edges emanating from the cycle set to the check nodes singly connected to variable nodes in the set. It is used to measure the connectivity of a cycle set to all other nodes.*

We further denote the cycle-indicator (CI) of a cycle set by e' and define the CI of a cycle set with size w' as $e' = \sum_{i=1}^{i=w'}(d'_i - 2) - u'$ where d'_i is the degree of the ith variable node in the cycle set. We can thus label a cycle set of size w' with CSE value u' and CI value e' as a $[w'; u'; e']$ CS. Using the same arguments as in Sect. 5.3.3, we can readily show that there are $e'/2 + 1$ "distinguishable" cycles in a $[w'; u'; e']$ CS. Figure 5.11 illustrates some $[w'; u'; e']$ CS examples. The variable node with degree 6 is the common node linking up the cycles. The CSE values of the cycle sets in Fig. 5.11(b), (c) and (d) are, respectively, 4, 2 and 0.

Comparing Fig. 5.11 with Fig. 5.8 and Fig. 5.9 , it can be easily seen that PTS-ICSs can be formed by combining one or more cycle set(s) and the singly-connected check nodes. Hence, by eliminating $[w'; u'; e']$ CS with small w', relatively smaller u' and $e' \geq 0$ during the code construction process, detrimental PTS-ICSs can be effectively avoided.

5.4.2 Estimation of the minimum CSE of a cycle set

Consider a variable node, denoted by v, and expand its connections to a depth of D to form a subgraph. If all possible cycle sets, under the condition that the variable node v is the common node, have to be found and their CSE values evaluated, it will be a very complex and time-consuming process. Instead, we estimate the minimum CSE of cycle sets with different sizes as follows. Note that all cycles refer to those beginning and terminating at v.

1. Denote the degree of the variable node v by d_v. Number the edges emanated from the variable node v as $\tau_1, \tau_2, ..., \tau_{d_v}$, and form the set $Z(v) = \{\tau_1, \tau_2, ..., \tau_{d_v}\}$. Initialize another set $Z_c(v)$ to be an empty set.

2. Set $k = 1$. Search for all (single) cycles in the subgraph and select the one with the smallest ACE (please refer to Section 2.4.2 for the definition of ACE)

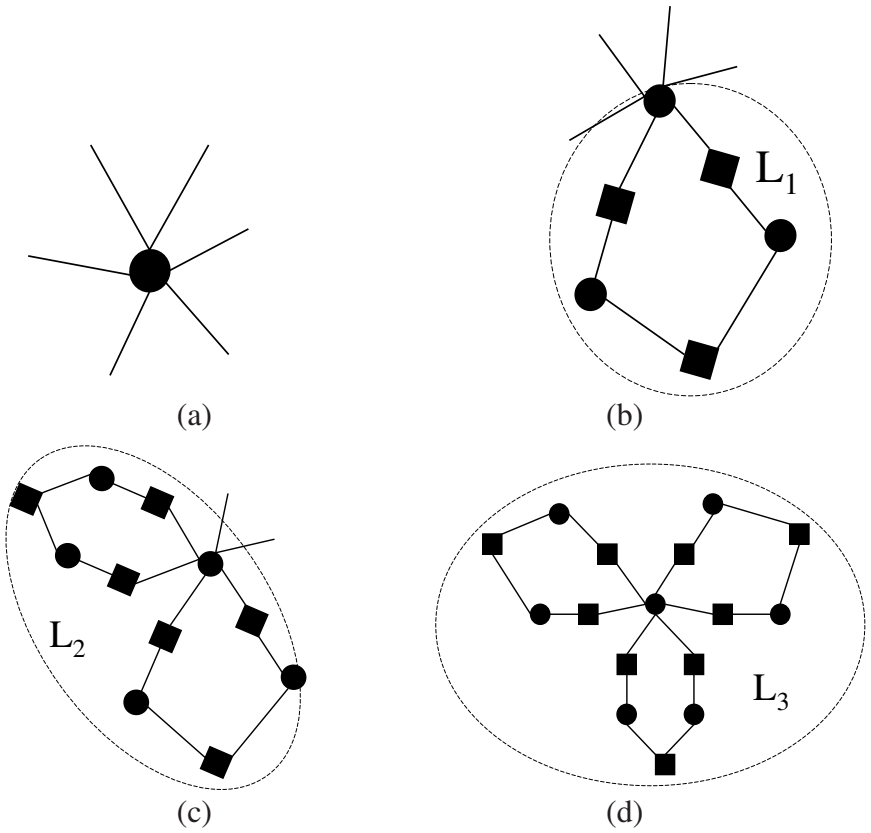


Figure 5.11: Cycle sets and CSE values. (a) A variable node with degree 6; (b) A $[3; 4; 0]$ cycle set L_1 containing one cycle and CSE of L_1 equals 4; (c) a $[5; 2; 2]$ cycle set L_2 containing two cycles and CSE of L_2 equals 2; (d) a $[7; 0; 4]$ cycle set L_3 containing three cycles and CSE of L_3 equals 0.

value. Denote the selected cycle by l_k. Suppose the two edges numbered $\tau_{k_1}, \tau_{k_2} \in Z(v)$ are contained in the cycle l_k. Then we set the cycle set $L_k = \{l_k\}$ and $Z_c(v) = \{\tau_{k_1}, \tau_{k_2}\}$. Moreover, the CSE value of L_k, denoted by Υ_k, equals the ACE value of l_k.

3. Increase k by 1. Search for all remaining cycles that begin with an edge numbered $\tau_i \in (Z(v) \backslash Z_c(v))$ in the subgraph. Add the cycle, denoted by l_k, to L_{k-1} such that the CSE of the new cycle set $L_k = \{l_1, l_2, \ldots, l_k\}$, i.e., Υ_k, is the smallest. Then add the edge(s) appearing in both the selected cycle l_k and $Z(v) \backslash Z_c(v)$ to $Z_c(v)$.

4. Repeat Step 3 until the cycles in the cycle set L_k have included all edges emanated from v, i.e. $Z_c(v) = Z(v)$.

5. The approximate minimum CSE (ACSE) value among all possible cycle sets with v as the root node is then obtained from $\min_k \Upsilon_k$.

Given an LDPC code. We can make use of the aforementioned method to obtain the ACSE value of every variable node, and hence the minimum ACSE value among all variable nodes. Note that because of the estimation, there may exist, with a small chance though, cycle sets with CSE values lower than the minimum ACSE value obtained.

5.4.3 PEG-ACSE Code Construction Algorithm

In the following, we present the PEG-ACSE code construction algorithm, in which cycle sets with small ACSE values are avoided (compared with the PEG-ACE algorithm in which single cycles with small ACE values are avoided). Suppose we are given the length of an LDPC code and also the degree distributions of the variable nodes and check nodes. We then assign each variable node with a degree randomly picked from the variable-node degree distribution. Next, we sort the variable nodes according to their degrees in a non-decreasing manner, i.e., $d_i \leq d_j$ for $i < j$. To design codes with low error floor, variable nodes with small degrees should be involved in cycles which are as large as possible because these variable nodes contribute few extrinsic connections. It is particularly crucial for degree-2 variable nodes because they do not give rise to any extrinsic connections in cycles.

Denote the block length, check length and the number of degree-2 variable nodes by, respectively, N, M and N_{v2}. Also, define $C^{(D)}(v_i)$ as the check-node set reached by a subgraph expanding from a variable node v_i at a depth D. Then we connect $\min\{M, N_{v2}\}$ degree-2 nodes in a "zigzag" manner, as in [55, 106], to guarantee that the cycles formed by the degree-2 variable nodes are the largest possible. For the remaining nodes, connections are made based on the following algorithm.

- Set the ACSE threshold value ζ with a positive integer.
- For $i = (\min\{M, N_{v2}\} + 1)$ to N
  - Set threshold_satisfied = 0;
  - While threshold_satisfied == 0
    - For $k = 1$ to d_i
      - Connect the edges of the variable node v_i to the check nodes in the set $C^{(D)}(v_i)$ using the PEG algorithm [55].
    - End ($k = 1$ to d_i)
    - With v_i as the root node, evaluate the ACSE value among all cycle sets of different sizes using the algorithm in Sect. 5.4.2.
    - If the ACSE value is smaller than the threshold ζ
      - threshold_satisfied=0;
    - Else
      - threshold_satisfied=1;
    - End (if...else)
  - End (while)
- End ($i = (\min\{M, N_{v2}\} + 1)$ to N)

(Note that in the case of the PEG-ACE algorithm [62], only single cycles instead of cycle sets will be considered.)

5.4.4 Performance analysis of PEG-only, PEG-ACE and PEG-ACSE construction algorithms

In the PEG-based construction algorithms, a bipartite graph is grown step-by-step by picking a variable node and adding connections between the variable node and the candidate check nodes. Consider the jth variable node in the construction process. We call the variable node "completed" when all its d_j connections have been connected to the check nodes. Since the variable nodes are picked in a non-decreasing manner according to their degrees, the bipartite graph obtained when the ith degree-k variable node is "completed", which we denote by $G_{k,i}$, will contain only variable nodes with degrees ranging from 2 to k.

Denote the smallest cycle length (i.e., number of edges in the cycle) incurred by any of degree-k variable nodes in the graph $G_{k,i}$ by $s_{k,i}$[3] and the corresponding (root) degree-k variable node by $R_{k,i}$. We further consider the *possible* cycle sets with $R_{k,i}$ as the common node. We denote $\mathbb{S}(R_{k,i}, t)$, where $2 \leq t \leq k$, as the *smallest possible* cycle set possessing the following two features.

- The cycle set involves t out of k edges emanating from the root node $R_{k,i}$.
- The cycle set contains T-type variable node(s) but no T-type check nodes.

Here, the size of the cycle set $\mathbb{S}(R_{k,i}, t)$, denoted by $|\mathbb{S}(R_{k,i}, t)|$, is defined as the number of variable nodes in the cycle set.

Theorem 5.1 *For $k = 2, 3, ..., d_v$ where d_v is the maximum variable-node degree and $t = 2, 3, ..., k$, the size of the cycle set $\mathbb{S}(R_{k,i}, t)$ in $G_{k,i}$ is given by*

$$|\mathbb{S}(R_{k,i}, t)| = \frac{s_{k,i}}{2} + \left(\left\lceil \frac{s_{k,i}}{4} \right\rceil - 1 \right) (t - 2) \tag{5.29}$$

where $\lceil x \rceil$ is the smallest integer larger than or equal to x.

Proof: According to the definition of $\mathbb{S}(R_{k,i}, t)$, when $t = 2$, $\mathbb{S}(R_{k,i}, 2)$ represents the smallest possible *single cycle* originating from and ending at $R_{k,i}$. Since the length of the smallest cycle has been denoted by $s_{k,i}$ and the number of variable

[3]Note that $s_{k,i}$ is always an even number.

nodes in a cycle equals half the cycle length, the size of $\mathbb{S}(R_{k,i}, 2)$ is given by

$$|\mathbb{S}(R_{k,i}, 2)| = \frac{s_{k,i}}{2} \qquad k = 2, 3, ..., d_v; \quad i = 1, 2, \ldots. \tag{5.30}$$

Thus, (5.29) is true for $t = 2$. Next, when $t = 3$, the *smallest possible* cycle set will be formed if, in addition to $\mathbb{S}(R_{k,i}, 2)$, a new path with *the shortest possible length* is created between $R_{k,i}$ and $\mathbb{S}(R_{k,i}, 2)$. Consider a new path, denoted by $l_X^{(3)}$ in Fig. 5.12, that joins $\mathbb{S}(R_{k,i}, 2)$ at the variable node X[4]. We further decompose the smallest cycle $\mathbb{S}(R_{k,i}, 2)$ into two paths — the first one from $R_{k,i}$ to variable node X, denoted by $l_X^{(1)}$; and second one from X to $R_{k,i}$, denoted by $l_X^{(2)}$. Denoting the length of a path $l_X^{(\nu)}$ by $||l_X^{(\nu)}||, \nu = 1, 2, 3$, we have

$$||l_X^{(1)}|| + ||l_X^{(2)}|| = s_{k,i} \equiv 2\tilde{s}_{k,i} \tag{5.31}$$

where

$$\tilde{s}_{k,i} = \frac{s_{k,i}}{2} \tag{5.32}$$

is equivalent to the number of variables nodes in the smallest cycle $\mathbb{S}(R_{k,i}, 2)$, i.e., $\tilde{s}_{k,i} = |\mathbb{S}(R_{k,i}, 2)|$. Without loss of generality, we also assume that the path of $l_X^{(1)}$ is longer than or equal to that of $l_X^{(2)}$, i.e.,

$$||l_X^{(1)}|| \geq ||l_X^{(2)}||. \tag{5.33}$$

Then, the new path $l_X^{(3)}$ must be longer than or equal to the length of $||l_X^{(1)}||$, i.e.,

$$||l_X^{(3)}|| \geq ||l_X^{(1)}||. \tag{5.34}$$

Otherwise, if $l_X^{(3)}$ has a smaller length than $l_X^{(1)}$, it would have combined with $l_X^{(2)}$ to form the smallest cycle $\mathbb{S}(R_{k,i}, 2)$, which contradicts our assumption. Further, we substitute (5.33) into (5.31) to obtain

$$||l_X^{(1)}|| \geq \tilde{s}_{k,i}. \tag{5.35}$$

Since all the paths being considered here start and end at variable nodes, the path

[4] X has to be a variable node because we do not consider cycle sets with T-type check nodes.

lengths must be even numbers. Thus, (5.35) can be rewritten as

$$||l_X^{(1)}|| \geq \begin{cases} \tilde{s}_{k,i} & \text{when } \tilde{s}_{k,i} \text{ is even} \\ \tilde{s}_{k,i}+1 & \text{when } \tilde{s}_{k,i} \text{ is odd.} \end{cases} \tag{5.36}$$

Combining (5.34) and (5.36), we can conclude that the shortest possible path length of $l_X^{(3)}$ equals (i) $\tilde{s}_{k,i}$ when $\tilde{s}_{k,i}$ is even; and (ii) $\tilde{s}_{k,i}+1$ when $\tilde{s}_{k,i}$ is odd. With reference to Fig. 5.12, we can observe that the path $l_X^{(3)}$ will add $\frac{||l_X^{(3)}||}{2}-1$ extra variable nodes to $\mathbb{S}(R_{k,i},2)$. Thus, when $t=3$, the size of the smallest possible cycle set, i.e., $|\mathbb{S}(R_{k,i},3)|$, can be expressed as

$$\begin{aligned} |\mathbb{S}(R_{k,i},3)| &= \begin{cases} \frac{s_{k,i}}{2}+\frac{\tilde{s}_{k,i}}{2}-1 & \text{when } \tilde{s}_{k,i} \text{ is even} \\ \frac{s_{k,i}}{2}+\frac{\tilde{s}_{k,i}+1}{2}-1 & \text{when } \tilde{s}_{k,i} \text{ is odd} \end{cases} \\ &= \frac{s_{k,i}}{2}+\left(\left\lceil\frac{s_{k,i}}{4}\right\rceil-1\right). \end{aligned} \tag{5.37}$$

Therefore, (5.29) is true for $t=3$. Using a similar argument, whenever $t \geq 3$, the *smallest possible* cycle set $\mathbb{S}(R_{k,i},t)$ will be formed if $t-2$ new paths, with length equal to the shortest possible path length of $l_X^{(3)}$ above, are created between $R_{k,i}$ and the variable node X in $\mathbb{S}(R_{k,i},2)$. Subsequently, it can be readily proven that the size of $\mathbb{S}(R_{k,i},t)$ equals that given in (5.29).

Corollary 5.1 *When all the degree-k variable nodes are "completed", we denote*

- *$\hat{G}_k$ as the graph obtained;*
- *$\hat{s}_k$ as the smallest cycle length incurred by any of degree-k variable nodes in $\hat{G}_k$;*
- *R_k as the corresponding (root) degree-k variable node;*
- *$\mathbb{S}(R_k,t),\ t=2,3,\ldots,k$, as the smallest possible cycle set involving t out of k edges emanated from R_k but containing no T-type check nodes.*

Then, for $k=2,3,\ldots,d_v$, the size of $\mathbb{S}(R_k,t)$ in $\hat{G}_k$ is given by

$$|\mathbb{S}(R_k,t)| = \frac{\hat{s}_k}{2}+\left(\left\lceil\frac{\hat{s}_k}{4}\right\rceil-1\right)(t-2). \tag{5.38}$$

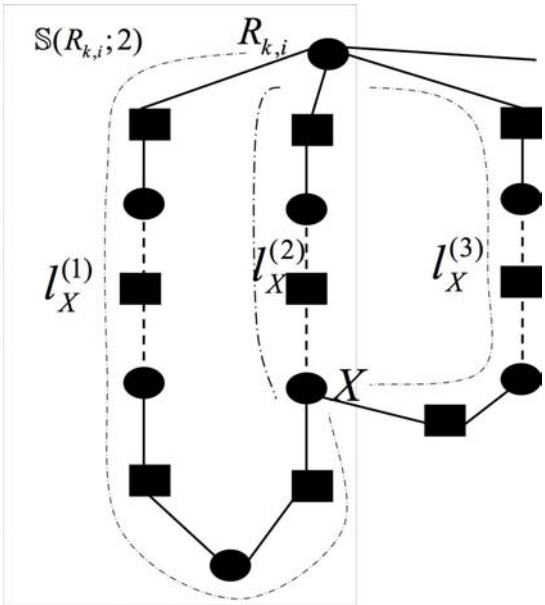


Figure 5.12: The smallest cycle $\mathbb{S}(R_{k,i}, 2)$ is decomposed into two paths — the first one from $R_{k,i}$ to variable node X, denoted by $l_X^{(1)}$; and the second one from X to $R_{k,i}$, denoted by $l_X^{(2)}$. A new path, denoted by $l_X^{(3)}$, joins $\mathbb{S}(R_{k,i}, 2)$ at X.

Proof: (5.38) is obtained when i in (5.29) equals the number of variable nodes with degree k.

Corollary 5.2 *Given the graph $\hat{G}_k$ with $k \geq 2$. For $e \geq 0$, the minimum possible size of a $[w; 0; e]$ PTS-ICS (i.e., w) equals $\min_t(|\mathbb{S}(R_t, t)|)$ where $t \in \{2, 3, \ldots, k\}$.*

Proof: When $e \geq 0$, a $[w; 0; e]$ PTS-ICS contains one or more cycles, which can be made up of one or more cycle sets (CSs). The minimum possible size of a $[w; 0; e]$ PTS-ICS, therefore, can be readily shown equivalent to the minimum possible size of a $[w'; 0; e']$ CS with no T-type check nodes and $e' \geq 0$. Moreover, the "0" in the $[w'; 0; e']$ CS indicates that there are no check nodes singly-connected to the variable nodes in the CS. In other words, all edges emanated from the variable nodes must be involved in the cycle(s). In consequence, a $[w'; 0; e']$ CS with the minimum possible size is equivalent to a $\mathbb{S}(R_t, t), t = 2, , 3, ..., k$, in which all variable nodes are connected to check nodes within $\mathbb{S}(R_t, t)$. Thus, we can conclude

that the minimum possible size of a $[w; 0; e]$ PTS-ICS equals $\min_t(|\mathbb{S}(R_t, t)|)$ where $t \in \{2, , 3, ..., k\}$.

Corollary 5.3 *Given the graph $\hat{G}_k$ with $k \geq 3$. For $e \geq 0$, the minimum possible size of a $[w; 1; e]$ PTS-ICS equals $\min_t(|\mathbb{S}(R_t, t-1)|)$ where $t \in \{3, 4, \ldots, k\}$.*

Proof: The minimum possible size of a $[w; 1; e]$ PTS-ICS is equivalent to the minimum possible size of a $[w'; 1; e']$ CS with no T-type check nodes and $e' \geq 0$. Moreover, the "1" in the $[w'; 1; e']$ CS indicates that there is only one check node singly-connected to the variable nodes in the CS. Such a CS is equivalent to (i) a $\mathbb{S}(R_t, t-1)$ if the singly-connected check node is attached to the degree-t (root) variable node; or (ii) $\mathbb{S}(R_t, t)$ if the singly-connected check node is attached to one of the other variable nodes in $\mathbb{S}(R_t, t)$ except the degree-t root node. In consequence, the minimum possible size of a $[w; 1; e]$ PTS-ICS in $\hat{G}_k$ equals $\min_t(\min\{|\mathbb{S}(R_t, t-1)|, |\mathbb{S}(R_t, t)|\}) = \min_t(|\mathbb{S}(R_t, t-1)|)$ where $t \in \{3, 4, ..., k\}$. (Note that the last equality results from (5.38).)

Corollary 5.4 *Given the graph $\hat{G}_k$ with $k \geq 3$. For $e > 0$, the minimum possible size of a $[w; 1; e]$ PTS-ICS equals $\min\{|\mathbb{S}(R_3, 3)|, \min_t(|\mathbb{S}(R_t, t-1)|)\}$ where $t \in \{4, 5, ..., k\}$.*

Proof: We use a similar procedure as in the proof of Corollary 5.3. Since $e > 0$, a $[w; 1; e]$ PTS-ICS contains more than one cycle. Using the results in Corollary 5.3, when $t = 3$, a $[w'; 1; e']$ CS with $e' > 0$ can only be formed from $|\mathbb{S}(R_3, 3)|$ if the singly-connected check node is attached to one of the other variable nodes in $|\mathbb{S}(R_3, 3)|$ except the degree-3 root node. Therefore, when $e > 0$, the minimum possible size of a $[w; 1; e]$ PTS-ICS in $\hat{G}_k$ is given by $\min\{|\mathbb{S}(R_3, 3)|, \min_t(|\mathbb{S}(R_t, t-1)|)\}$ where $t \in \{4, 5, ..., k\}$.

In the PEG-only construction algorithm, suppose the smallest cycle length incurred by a degree-3 variable node in $\hat{G}_3$ is found to be 12 during the construction process, i.e., $\hat{s}_3 = 12$. Corollary 5.1 shows that

$$|\mathbb{S}(R_3, 3)| = \frac{12}{2} + \left(\left\lceil \frac{12}{4} \right\rceil - 1\right)(3-2) = 8. \tag{5.39}$$

Based on Corollary 5.2 and Corollary 5.4, we can further conclude that $[8; 0; e]$ and

$[8; 1; e]$ PTS-ICSs with $e > 0$ (multiple cycles) may reside in the code graph. These small size PTS-ICSs will undoubtably contribute significantly to the error floor.

In the proposed PEG-ACSE construction algorithm, two parameters have to be set — the ACSE value threshold ζ and the depth threshold D. To reduce as many detrimental PTS-ICSs as possible, ζ and D should be set large. However, setting a larger D implies cycle sets with larger sizes will be searched, resulting a higher chance that cycles/cycle sets not able to meet the ACSE thresholds. Alternatively, ζ has to be reduced or else no codes fulfilling the requirements can be found. Nonetheless, if we need to evaluate cycle lengths of l_{cycle}, D should be set no less than half of l_{cycle}, i.e., $D \geq l_{\text{cycle}}/2$. For example, if we intend to remove the $[8; 0; e]$ and $[8; 1; e]$ PTS-ICSs with $e > 0$, cycle lengths up to 12 should be considered and hence we must set D no less than 6.

In addition, the value of ζ should be carefully designed to guarantee the efficient removal of detrimental PTSs. For example, consider the $[w; 0; e]$ and $[w; 1; e]$ PTS-ICSs shown in Fig. 5.13(a) and (b). In the figure, the solid line represents that the check node is directly connected to the variable node. The dash line represents that there is a path between the check node and the variable node. Assume that a degree-3 variable node giving rise to the smallest cycle length $\hat{s}_3$ is contained in each of two PTS-ICSs in the figure. Denote the degree-3 variable node by R_3 and consider it as a root node. Further, denote the three "distinguishable" cycles in the $[w; 0; e]$ and $[w; 1; e]$ PTS-ICSs by l_1, l_2 and l_3. Consider the cycle set $\mathbb{S}(R_3, 3)$ originating from R_3 and comprising l_1 and l_2. From Theorem 5.1, the size of $\mathbb{S}(R_3, 3)$ is no less than $\frac{\hat{s}_3}{2} + \left\lceil \frac{\hat{s}_3}{4} \right\rceil - 1$. When the cycle l_3 is added to the cycle set $\mathbb{S}(R_3, 3)$, at least on extra variable node will be added. Therefore, the size of the $[w; 0; e]$ or $[w; 1; e]$ PTS-ICS is governed by

$$\begin{aligned} w &\geq \frac{\hat{s}_3}{2} + \left\lceil \frac{\hat{s}_3}{4} \right\rceil - 1 + 1 \\ &= \frac{\hat{s}_3}{2} + \left\lceil \frac{\hat{s}_3}{4} \right\rceil . \end{aligned} \tag{5.40}$$

Moreover, the equality hold if all the three "distinguishable" cycles l_1, l_2 and l_3 achieve the smallest cycle length $\hat{s}_3$ in $\hat{G}_3$. For example, if it is found that $\hat{s}_3 = 8$, then $w \geq 4 + 2 = 6$.

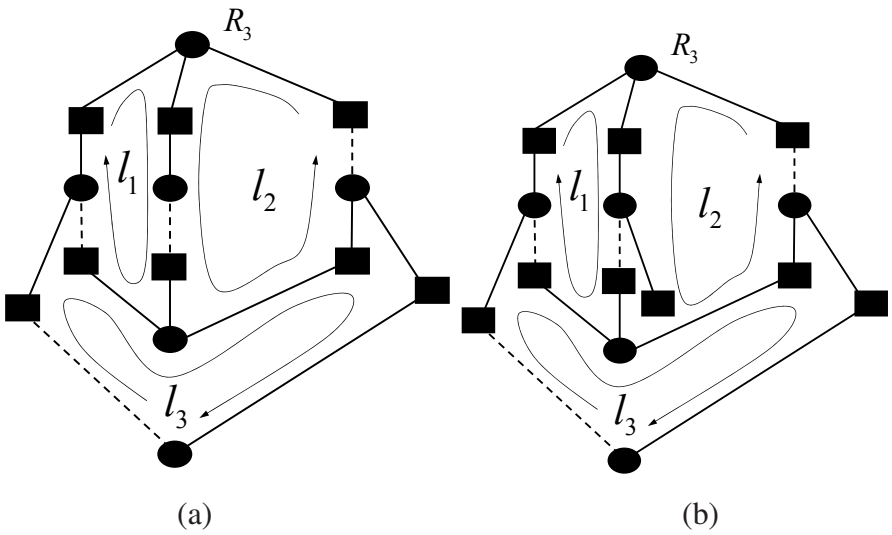


Figure 5.13: The (a) $[w; 0; 4]$ PTS-ICS and (b) $[w; 1; 4]$ PTS-ICS generated during the construction of LDPC codes by employing the PEG-ACSE algorithm with $\zeta \leq 3$. The solid line represents that the check node is directly connected to the variable node. The dash line represents that there is a path between the check node and variable node.

Further, as shown in Fig. 5.13(a), the smallest CSE of the cycle sets emanated from root node R_3 in the $[w; 0; 4]$ PTS-ICS is 2 (e.g., the CSE of $\mathbb{S}(R_3, 3)$ is 2). Also, from Fig. 5.13(b) we can see that the CSEs of all the cycle sets emanated from the root node R_3 in the $[w; 1; 4]$ PTS-ICS are larger than or equal to 3. The results show that if the ACSE threshold ζ is set to less than or equal to 3, i.e., $\zeta \leq 3$, the PEG-ACSE construction algorithm cannot avoid the $[w; 0; 4]$ and $[w; 1; 4]$ PTS-ICSs as shown in Fig. 5.13.

5.5 Results and discussions

In this section, we compare the performance of LDPC codes constructed using the PEG-only, PEG-ACE and PEG-ACSE algorithms.

5.5.1 "DE15" with code length 1008

First, we consider the DE-optimized code "DE15". Properties of the code have been listed in Table 4.3. Moreover, the corresponding variable-node degree distribution, given in (4.10), is shown as follows.

$$\begin{aligned}\lambda_1(x) &= 0.23802x + 0.20907x^2 + 0.03492x^3 + 0.12015x^4 \\ &\quad + 0.01587x^6 + 0.00480x^{13} + 0.37627x^{14}\end{aligned} \tag{5.41}$$

Suppose a code length of 1008 is used. We observe that during the PEG-only code-construction process, $\hat{s}_3 = 12$. Based on the analysis in Sect. 5.4.4, we set the depth threshold D to be larger than $\hat{s}_3/2 = 6$ in the PEG-ACE and PEG-ACSE construction algorithms, with an aim to avoiding any $[8; 0; e]$ and $[8; 1; e]$ PTS-ICSs with $e > 0$. Furthermore, given $D > 6$, our code-construction results have shown that the largest ACE/ACSE threshold that can be achieved equals 4 if the LDPC codes are to be constructed successfully. When $\zeta = 3$ and $\zeta = 4$, the largest possible depths D are, respectively, 12 and 8.

Five different codes of length 1008 are constructed based on "DE15". We denote the five codes by the abbreviations "PEG-only", "PEG-ACE-12-3", "PEG-ACE-8-4", "PEG-ACSE-12-3" and "PEG-ACSE-8-4" respectively. The code "PEG-only", which is exactly the same as the code "PEGirReg504x1008" in [97], is used a reference here for performance comparison. For the other codes, the first two blocks of alphabets in the abbreviations represent the code-construction method being used; the third block of digit(s) denotes the value of D; and the last digit gives the value of ζ. For example, the code "PEG-ACE-12-3" is constructed based on the PEG-ACE mechanism with $D = 12$ and $\zeta = 3$.

5.5.1.1 Error Rates

Assuming an AWGN channel, the belief propagation decoder will iterate a maximum of 50 times for each received codeword. For each of the codes, the simulation will continue until 100 decoding failures are collected. Figure 5.14 plots the bit-error-rate (BER) and the block-error-rate (BLER) curves of the five codes. It can be seen that, the PEG-ACSE-constructed codes have very similar error performance

as the PEG-only-constructed and the PEG-ACE-constructed codes at the waterfall region and outperform them at the high signal-to-noise ratio (SNR) region. At a BER of 10^{-7}, the code "PEG-ACSE-8-4" outperforms the codes "PEG-only" and "PEG-ACE-8-4" by about 0.1 dB and 0.2 dB, respectively. We also observe that the code "PEG-ACSE-8-4" has a lower error floor than the code "PEG-ACSE-12-3" while the code "PEG-ACE-8-4" has a lower error floor than the code "PEG-ACE-12-3". The results have shown that raising the ACE/ACSE threshold from 3 to 4 and at the same time reducing the depth threshold D from 12 to 8 can produce codes with lower error floors.

5.5.1.2 Decoding failures and $[w; u; e]$ PTSs

Also, for each of the codes and at a high SNR, we look into the 100 decoding failures to check if the failures are related to $[w; u; e]$ PTSs. We find that almost all the decoding failures are caused by single-cycle or multiple-cycle $[w; u; e]$ PTSs, i.e., $[w; u; e]$ PTSs with $e \geq 0$. In addition, failures caused by multiple-cycle $[w; u; e]$ PTSs far exceed those by single-cycle ones. In Table 5.2, we tabulate the failure statistics for the five codes when the SNR equals 2.8 dB. In the table, N_t denotes the number of transmitted blocks resulting in the 100 decoding failures. $\hat{\eta}_{e>0}$ and $\hat{\eta}_{e=0}$ represent the number of failures caused by $[w; u; e]$ PTSs with $e > 0$ (multiple-cycle) and $e = 0$ (single-cycle), respectively. The results have indicated that out of the 100 failures, 94 to 98 of them are caused by PTSs with one or more cycles. Furthermore, multiple-cycle PTSs are causing a lot more errors than single-cycle ones.

In Table 5.2, we have also shown, for each of the codes, the labels of the failure-causing $[w; u; e]$ PTSs with small w, small u and $e > 0$ (multiple-cycle). The number of failures caused by PTSs with the same label $[w; u; e]$ and the corresponding weighted block error rate are denoted by, respectively, $\hat{\eta}_{w,u,e}$ and $\hat{P}_{w,u,e}(= \hat{\eta}_{w,u,e}/N_t)$. We find that the most detrimental PTS in the code "PEG-ACSE-8-4" is $[16; 0; 2]$ PTS, giving rise to an error probability of around 4.00×10^{-8}. Moreover, the most detrimental PTS in the code "PEG-ACE-12-3" is $[9; 1; 4]$ PTS, producing an error probability of around 2.43×10^{-6}. For all the other codes "PEG-only", "PEG-ACE-8-4" and "PEG-ACSE-12-3", the most detrimental PTSs are causing error probabilities in the order of 10^{-7}. The above observations are consistent with

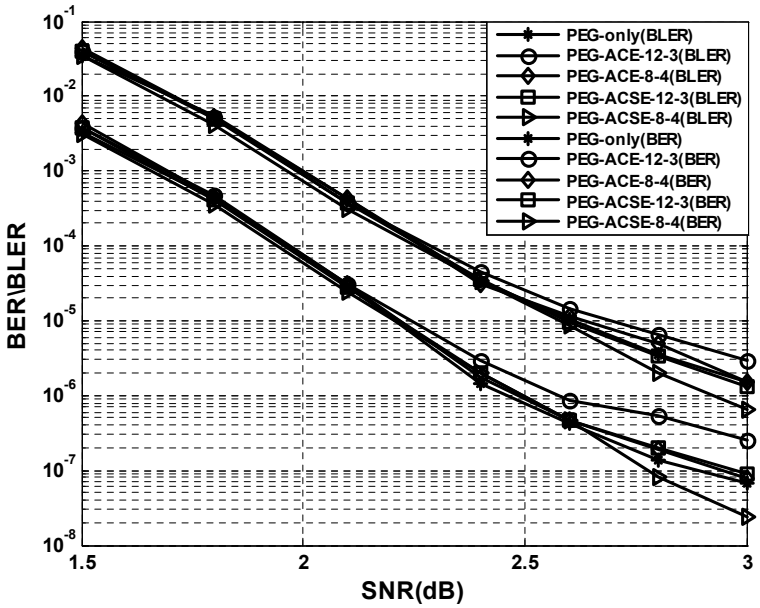


Figure 5.14: The BER and BLER performance of LDPC codes with length 1008 and degree distribution following "DE15". Codes are constructed using PEG-only, PEG-ACE, and the proposed PEG-ACSE algorithms.

the BER results in the previous section, which show that "PEG-ACSE-8-4" is the best-performing code; "PEG-ACSE-12-3", "PEG-ACE-8-4" and "PEG-only" have similar error performance; and "PEG-ACE-12-3" has the highest error floor at high SNR.

5.5.1.3 Minimum possible size of $[w; 0; e]$ and $[w; 1; e]$ PTS-ICSs with multiple cycles in the code "PEG-ACE-12-3"

We further take the code "PEG-ACE-12-3", which has a comparatively high error floor for analysis. As the variable node degrees in "PEG-ACE-12-3" follow the distribution in (5.41), the variable nodes can only attain degrees of 2, 3, 4, 5, 7, 14 and 15. In addition, during the code construction process based on the PEG-ACE algorithm, we have found that the minimum cycle lengths $\hat{s}_k$ in $\hat{G}_k$ are 12, 12, 10, 8, 6, 6, for $k = 3, 4, 5, 7, 14, 15$, respectively. (Note that the minimum cycle length at $k = 2$ needs not to be considered when evaluating multiple-cycle PTSs.) Thus, according to Corollary 5.2 and Corollary 5.4 in Sect. 5.4.4, the minimum possible

Table 5.2: Failure statistics for the codes with degree distribution "DE15". Code length is 1008. SNR equals 2.8 dB.

Code	N_t	$\hat{\eta}_{e>0}$	$\hat{\eta}_{e=0}$	$[w;u;e]$	$\hat{\eta}_{w,u,e}$	$\hat{P}_{w,u,e}$
PEG-only	28,448,370	89	9	$[13;0;2]$	8	2.81×10^{-7}
				$[9;1;2]$	8	2.81×10^{-7}
				$[10;1;2]$	5	1.76×10^{-7}
				$[8;2;2]$	1	3.52×10^{-8}
PEG-ACE-12-3	15,667,615	92	5	$[17;0;6]$	1	6.38×10^{-8}
				$[9;1;4]$	38	2.43×10^{-6}
				$[10;1;2]$	1	6.38×10^{-8}
				$[7;2;2]$	2	1.28×10^{-7}
PEG-ACE-8-4	20,150,438	84	10	$[12;0;2]$	15	7.44×10^{-7}
				$[13;0;2]$	6	2.98×10^{-7}
				$[12;1;2]$	2	9.93×10^{-8}
				$[7;2;2]$	6	2.98×10^{-7}
PEG-ACSE-12-3	29,947,195	92	4	$[14;0;4]$	8	1.34×10^{-7}
				$[16;0;4]$	1	3.34×10^{-8}
				$[10;1;4]$	12	4.01×10^{-7}
				$[10;2;2]$	1	3.34×10^{-8}
PEG-ACSE-8-4	50,017,093	91	5	$[16;0;2]$	2	4.00×10^{-8}
				$[17;0;4]$	1	2.00×10^{-8}
				$[12;1;2]$	1	2.00×10^{-8}
				$[9;2;2]$	1	2.00×10^{-8}

size of multiple-cycle $[w; 0; e]$ PTS-ICSs equals

$$\begin{aligned}\min_{k\in\{3,4,5,7,14,15\}}[|\mathbb{S}(R_k,k)|] &= \min_{k\in\{3,4,5,7,14,15\}}\left[\frac{\hat{s}_k}{2}+\left(\left\lceil\frac{\hat{s}_k}{4}\right\rceil-1\right)(k-2)\right]\\ &= \min[8,10,11,9,15,16]\\ &= 8 \end{aligned} \tag{5.42}$$

while the minimum possible size of the multiple-cycle $[w; 1; e]$ PTS-ICSs is given by

$$\begin{aligned}&\min\left[|\mathbb{S}(R_3,3)|, \min_{k'\in\{4,5,7,14,15\}}(|\mathbb{S}(R_{k'},k'-1)|)\right]\\ =\ &\min\left[\frac{\hat{s}_3}{2}+\left\lceil\frac{\hat{s}_3}{4}\right\rceil-1, \min_{k'\in\{4,5,7,14,15\}}\left(\frac{\hat{s}_{k'}}{2}+\left(\left\lceil\frac{\hat{s}_{k'}}{4}\right\rceil-1\right)(k'-3)\right)\right]\\ =\ &\min[8,8,9,8,14,15]\\ =\ &8. \end{aligned} \tag{5.43}$$

Thus, if the ACE threshold $\zeta = 3$, it is *possible* that the PEG-ACE construction method produces $[8; 0; e]$ and $[8; 1; e]$ PTS-ICSs with multiple cycles. In our study, we do not find such PTS-ICSs. The smallest size multiple-cycle $[w; 0; e]$ PTS-ICS has the label $[17; 0; 6]$ and has a much larger size than $[8; 0; e]$ PTS-ICS. In addition, multiple-cycle PTS-ICSs with labels $[9; 1; 4]$ and $[10; 1; 2]$ are found, which have size very close to the minimum possible $[8; 1; e]$ PTS-ICS with multiple cycles.

5.5.2 "DE15" with code length 2016

Next, we consider the case when a code length of 2016 is used together with the degree distribution following "DE15". The largest value of ζ we can achieve is 5 and the corresponding depth threshold is $D = 9$ for both the PEG-ACE and PEG-ACSE construction methods. Fig. 5.15 plots the BER/BLER curves of the codes constructed with different algorithms. The results have shown that at a BER of 10^{-7}, the code constructed with the PEG-ACSE method outperforms those produced with the PEG-ACE method and the PEG-only method by around 0.3 dB and 0.1 dB, respectively.

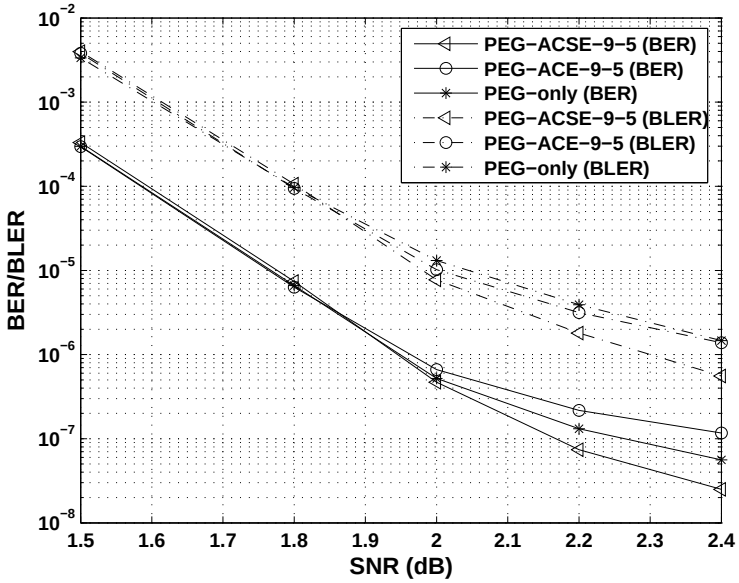


Figure 5.15: The BER and BLER performance of LDPC codes with length 2016 and degree distribution following "DE15". Codes are constructed using PEG-only, PEG-ACE, and the proposed PEG-ACSE algorithms.

5.5.3 "DE10" with code length 1008

We consider the codes with length 1008 and the degree distribution following "DE10". (Details of the code "DE10" can be found in Table 4.3 and (4.11).) The largest value of ζ we can achieve is 3 and the corresponding depth threshold is $D = 12$ for both the PEG-ACE and PEG-ACSE construction methods. At low and medium SNR regions, the codes constructed with the PEG-only, PEG-ACE and PEG-ACSE mechanisms perform very similarly. Significant differences are observed only at the high SNR region. In Table 5.3, we show the BERs and BLERs of the three codes at SNR=2.8 dB and SNR=3.0 dB. In this case, the code constructed with the PEG-ACSE method is still the best performer while that produced with the PEG-ACE algorithm outperforms the code created with the PEG-only method.

5.5.4 "DE14" and "CSF20" with code length 2016

We consider two rate-0.75 codes following degree distributions "DE14" and "CSF20". (Details of such degree distributions are provided in Table 4.4 in Chap-

Table 5.3: Bit error rate (BER) and block error rate (BLER) for the codes with degree distribution "DE10". Code length is 1008. SNR equals 2.8 dB and 3.0 dB.

Code	PEG-only	PEG-ACE-12-3	PEG-ACSE-12-3
BER (2.8 dB)	5.20×10^{-7}	2.58×10^{-7}	1.19×10^{-7}
BLER (2.8 dB)	9.57×10^{-6}	5.83×10^{-6}	4.53×10^{-6}
BER (3.0 dB)	2.32×10^{-7}	1.02×10^{-7}	8.79×10^{-8}
BLER (3.0 dB)	4.89×10^{-8}	2.20×10^{-8}	1.98×10^{-8}

ter 4.) Code lengths of 2016 are used. The corresponding $\hat{s}_3$ observed during the PEG-only code construction process for the two degree distributions are both 10. Based on the analysis in Section 5.4.4, the ACSE threshold ζ should be set larger than 3 so as to avoid the generation of $[6; 0; 4]$, $[6; 1; 4]$, $[7; 0; 4]$ and $[7; 1; 4]$ PTSs. However, we find that setting $\zeta = 4$ fails to generate any LDPC codes from the PEG-ACSE method. Even when $\zeta = 3$ is used, no LDPC codes can be successfully constructed.

We then modify the original PEG-ACSE method as follows. We set a starting threshold ζ_0 to 3, and a minimum threshold $\zeta_{\min,k}$ to 3 for $k = 2$ and 2 for $k = 4, 5, ..., d_v$. Consider the variable node with degree d_i. After all its edges have been connected to the check nodes using the PEG method, we will evaluate the ACSE value among all cycle sets of different sizes. If the ACSE value is larger than or equal to the starting threshold ζ_0, the requirement is satisfied and the next variable node in the sequence will be considered. Otherwise, the edges of the variable node will be reset and re-connected again using the PEG method. The ACSE value is then checked against the required threshold. The reset, re-connect and ACSE-check process will continue to iterate if the ACSE value fails to meet the starting threshold requirement. In the modified PEG-ACSE method, after a number of iterations, say 100 iterations, we will reduce the ACSE threshold from ζ_0 to $\zeta_{\min,k}$ such that the threshold requirement can be met with a higher chance. With such an arrangement, we might still achieve a certain performance improvement by reducing the number of small-size $[w; 0; 4]$ PTS-ICSs.

Figures 5.16 and 5.17 present the BER and BLER curves of the two types of codes constructed using PEG-only, PEG-ACE-6-3 and PEG-ACSE-6-3 methods.

The codes are abbreviated as "PEG-only", "PEG-ACSE-6-3" and "PEG-ACSE-6-3". The digit 6 is the value of D while 3 represents the value of ζ_0. We observe that for both degree distributions "DE14" and "CSF20", code "PEG-ACSE-6-3" has outperformed code "PEG-only" and code "PEG-ACE-6-3" and produces a lower error floor.

5.6 Summary

In this chapter, we have fully investigated the main cause of the error floor of LDPC codes. Given a $[w; u]$ trapping sets (TS) found from an LDPC code. We have shown that for a regular code, if $w > u$, the connected subgraph induced by this TS must possess multiple cycles. But for an irregular code, no useful information on the TS-induced connected subgraph (TS-ICS) can be deduced even given the values of w and u. Moreover, we have illustrated that TSs with the same label $[w; u]$ are not identical in general.

To distinguish different types of TSs, we have categorized TSs based on their ICSs having (i) no cycle, (ii) a single-cycle and (iii) multiple cycles. We have also proposed a cycle indicator (CI), denoted by e, as a simple metric of identifying different types of TS-ICSs. Further, we have proven that regardless of regular or irregular codes, for a TS-ICS with (i) no cycle, then $e < 0$; (ii) a single cycle, then $e = 0$; (iii) multiple cycles and T-type variable nodes but no T-type check nodes, then $e > 0$. Based on the CI, we have refined the label of a TS to $[w; u; e]$. Moreover, we have defined a $[w; u; e]$ primary TS (PTS) and have proposed that single-cycle or multiple-cycle $[w; u; e]$ PTS-ICS with small w and u are more harmful to the LDPC decoder.

Furthermore, we have analyzed the progressive-edge-growth (PEG) [55] code construction method. Based on the graph obtained when all the degree-k variable nodes have their edges connected to the check nodes, we have derived the relationship between the smallest cycle length and the smallest possible cycle set. Subsequently, we have proposed a PEG-ACSE (PEG-Approximate Cycle Set Extrinsic message degree) code construction algorithm with an aim to avoiding PTSs with single and multiple cycles. The simulation results have shown that LDPC codes constructed using the proposed PEG-ACSE method can provide the lowest error

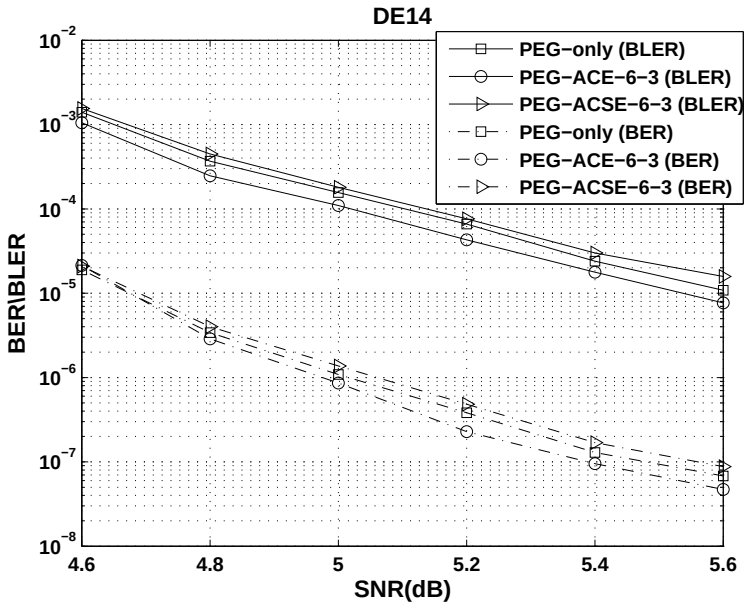


Figure 5.16: The BER and BLER performance of LDPC codes of rate-0.75 with length 2016 and degree distribution following "DE14". Codes are constructed using PEG-only, PEG-ACE-6-3 and the proposed PEG-ACSE algorithms.

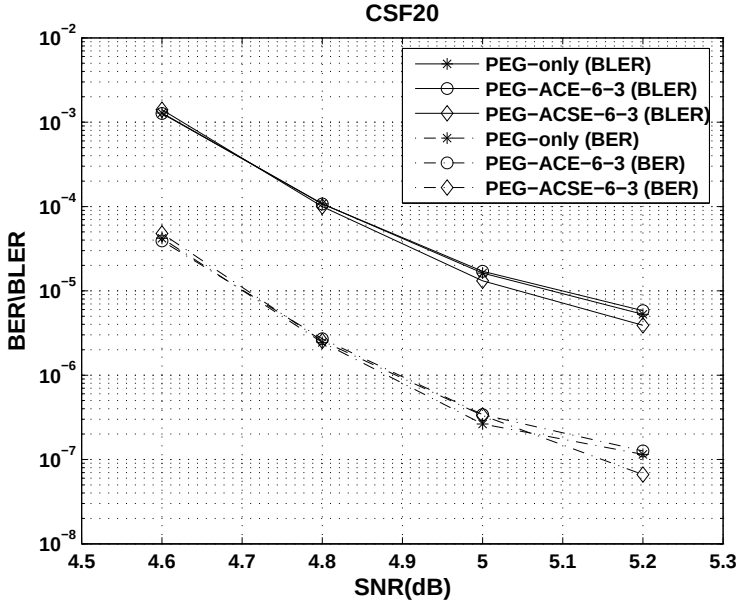


Figure 5.17: The BER and BLER performance of LDPC codes of rate-0.75 with length 2016 and degree distribution following "CSF20". Codes are constructed using PEG-only, PEG-ACE and the proposed PEG-ACSE algorithms.

floors. A further look into the decoding failures at a high SNR reviews that almost all the decoding failures are caused by single-cycle or multiple-cycle $[w; u; e]$ PTSs, i.e., $[w; u; e]$ PTSs with $e \geq 0$. In addition, failures caused by multiple-cycle $[w; u; e]$ PTSs far exceed those by single-cycle ones. We have also found that single-cycle or multiple-cycle $[w; u; e]$ PTS-ICS with small w and u can be successfully avoided by using the proposed PEG-ACSE method.

Finally, tens of millions of received codes have to be decoded in order to arrive at a reasonably reliable BER/BLER performance at the high SNR region (no larger than 3 dB). If the SNR is to be increased further, we will be not able to evaluate the code performance using Monte Carlo (MC) simulation due to the extremely low error rate and hence the prohibitive amount of simulation time needed to arrive at a meaningful error rate. To address the aforementioned issue, in the next chapter, we will propose evaluating *irregular* LDPC code performance at the high SNR region using the importance sampling (IS) approach in conjunction with PTSs identification.

Chapter 6

Performance evaluation of extremely low error rate at high SNR

In Chapter 5, we have demonstrated that trapping sets (TSs) with the same label $[w; u]$ (w denotes the number of variable nodes in the TS and u represents the number of check nodes with odd number of connections to the TS) may contribute very differently to the error floor, particularly for *irregular* LDPC codes. Subsequently, we characterize TSs with the refined label $[w; u; e]$ where e is a new parameter called "cycle indicator (CI)". Moreover, we define and make use of primary trapping sets (PTSs) to identify detrimental TSs that contribute more to the error floor. We have also constructed irregular codes and have shown their superior error performance compared with existing codes. However, at the high signal-to-noise-ratio (SNR) region, we are not able to evaluate the code performance using Monte Carlo (MC) simulation due to the extremely low error rate and hence the prohibitive amount of simulation time needed to arrive at a meaningful error rate.

In this chapter, we attempt to evaluate *irregular* LDPC code performance at the high SNR region using the importance sampling (IS) approach in conjunction with PTS identification. For any given LDPC code, we will first apply a three-step method that aims to search as many single- and mutiple-cycle PTSs within the code as possible. Then, we will classify these PTSs into different groups based on their labels, i.e., $[w; u; e]$'s. Further, by dividing the error region into various sub-regions centered by PTSs, we apply the IS simulator to evaluate the error rate of each of

the sub-regions. Based on the error rates of all the sub-regions, we can estimate the overall error rate of the LDPC code.

6.1 Importance sampling for regular LDPC codes

6.1.1 Review of MC simulator and IS simulator

For a random variable vector $\mathbf{x} = [x_1, x_2, ..., x_N]$ with joint pdf $f(\mathbf{x})$ and corresponding "error region" $\mathcal{E}$, the block error rate (BLER) of $\mathbf{x}$ equals $P_{\mathcal{E}} = \int_{\mathcal{E}} (\mathbf{x} \in \mathcal{E})$. Then the MC estimator of $P_{\mathcal{E}}$ using N_{MC} simulation runs can be expressed as

$$\hat{P}_{\mathrm{MC}} = \frac{1}{N_{\mathrm{MC}}} \sum_{i}^{N_{\mathrm{MC}}} 1_{\mathcal{E}}(\mathbf{x}), \tag{6.1}$$

where $1_{\mathcal{E}}(\mathbf{x})$ is the indicator function of the error region $\mathcal{E}$, i.e.,

$$1_{\mathcal{E}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

The MC approach becomes not viable when the error events have an extremely low probability of occurrence. It is because most of the time, the simulated events are correct and therefore of no importance.

In an IS estimator, a new random vector $\mathbf{x}^*$ sampled from the biased distribution $f^*(\mathbf{x}^*)$ is designed to increase the occurrence of rare error events. Then, using the biased distribution, the IS simulator of $P_{\mathcal{E}}$ for N_{IS} runs is given by [77]

$$\hat{P}_{\mathrm{IS}} = \frac{1}{N_{\mathrm{IS}}} \sum_{i}^{N_{\mathrm{IS}}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)], \tag{6.3}$$

where $\omega(\mathbf{x}^*)$ is the weight function given by

$$\omega(\mathbf{x}^*) = \frac{f(\mathbf{x}^*)}{f^*(\mathbf{x}^*)}. \tag{6.4}$$

Mean translation (MT) is one of the popular techniques to form the biased density. The idea of MT is to shift the mean of the original density function to the boundary

of the error region, i.e., $f^*(\mathbf{x}^*) = f(\mathbf{x}^* - \boldsymbol{\mu})$, where $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_N]$ is a point lying on the boundary. Then the weight function can be re-written as

$$\omega(\mathbf{x}^*) = \frac{f(\mathbf{x}^*)}{f(\mathbf{x}^* - \boldsymbol{\mu})}. \tag{6.5}$$

Denote $\mathrm{E}(\cdot)$ and $\mathrm{var}(\cdot)$ as the expectation operator and variance operator, respectively. Theoretically,

$$\begin{aligned} \mathrm{E}(\hat{P}_{\mathrm{IS}}) &= \int_{\mathcal{E}} \frac{f(\mathbf{x}^*)}{f^*(\mathbf{x}^*)} f^*(\mathbf{x}^*) dx \\ &= \int_{\mathcal{E}} f(\mathbf{x}^*) dx = \mathrm{E}(\hat{P}_{\mathrm{MC}}) \\ &= P_{\mathcal{E}}, \end{aligned} \tag{6.6}$$

which shows that both MC and IS are unbiased estimators of the BLER $P_{\mathcal{E}}$. Moreover, the well-known variance formulas of standard MC and IS estimators are given by [67]

$$\mathrm{var}(\hat{P}_{\mathrm{MC}}) = \frac{\hat{P}_{\mathrm{MC}} - \hat{P}_{\mathrm{MC}}^2}{N_{\mathrm{MC}}} \approx \frac{\hat{P}_{\mathrm{MC}}}{N_{\mathrm{MC}}}, \tag{6.7}$$

and

$$\mathrm{var}(\hat{P}_{\mathrm{IS}}) = \frac{\frac{1}{N_{\mathrm{IS}}} \sum_i^{N_{\mathrm{IS}}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]^2 - \hat{P}_{\mathrm{IS}}^2}{N_{\mathrm{IS}}}, \tag{6.8}$$

respectively.

In order to evaluate the quality of the estimators, the normalized error of MC and IS simulators are calculated as [67]

$$\phi_{\mathrm{MC}} = \frac{\sqrt{\mathrm{var}(\hat{P}_{\mathrm{MC}})}}{\hat{P}_{\mathrm{MC}}} \approx \frac{1}{\sqrt{\hat{P}_{\mathrm{MC}} N_{\mathrm{MC}}}}, \tag{6.9}$$

and

$$\phi_{\mathrm{IS}} = \frac{\sqrt{\mathrm{var}(\hat{P}_{\mathrm{IS}})}}{\hat{P}_{\mathrm{IS}}} = \sqrt{\frac{1}{N_{\mathrm{IS}}^2 \hat{P}_{\mathrm{IS}}^2} \sum_i^{N_{\mathrm{IS}}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]^2 - \frac{1}{N_{\mathrm{IS}}}}, \tag{6.10}$$

respectively.

To ensure that the IS simulator provides a good estimation of $P_{\mathcal{E}}$, the IS simu-

lation should be continued until $\phi_{\rm IS}^2 \leq \phi_{\rm MC}^2$ is satisfied [67], i.e.,

$$\frac{\sum_1^{N_{\rm IS}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]^2}{\{\sum_1^{N_{\rm IS}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]\}^2} - \frac{1}{N_{\rm IS}} \leq \frac{1}{\hat{P}_{\rm MC} N_{\rm MC}}. \tag{6.11}$$

Note that $\hat{P}_{\rm MC} N_{\rm MC}$ is actually the number of error events collected in the MC simulator. Moreover, a speed up gain of an IS simulator relative to MC can be calculated using [67]

$$\gamma_s = \frac{N_{\rm MC}}{N_{\rm IS}}|_{\phi_{\rm IS}=\phi_{\rm MC}}. \tag{6.12}$$

6.1.2 IS scheme for regular LDPC codes

Assume an all-zero codeword with a block length N is transmitted over a binary-input additive white Gaussian noise (AWGN) channel with mean zero and variance (noise power) σ^2. Denote the ith code bit ($i = 1, 2, ..., N$) by $\psi_i \in \{0, 1\}$. The transmitted signal corresponding to this code bit equals $(-1)^{\psi_i+1}$. With an all-zero codeword transmitted, the received vector $\mathbf{x} = [x_1, x_2, ..., x_i, ..., x_N]$ is given by $\mathbf{x} = -1 + \mathbf{z}$, where $\mathbf{z} = [z_1, z_2, ..., z_i, ..., z_N]$ is an independent and identically distributed sequence with probability density function (pdf)

$$\frac{1}{(2\pi)^{N/2}\sigma^N} \exp(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} z_i^2). \tag{6.13}$$

Suppose a message-passing algorithm [11] is used in the iterative decoder to decode the LDPC codes with input $\mathbf{x}$. To apply IS to LDPC codes at high SNR region, the error region is divided into independent sub-regions with respect to TSs. For a *regular* code, TSs with the same label $[w; u]$ are considered equivalent and categorized into one class.

The simulated error probability of a representative randomly picked from each class is obtained by employing the IS simulator. For each sub-region associated with a $[w; u]$ TS, the mean of the original density function is biased to $[\mu_1, \mu_2, ..., \mu_i, ..., \mu_N]$ based on MT. Then the weight function associated with the vector $\mathbf{x}^*$ sampled from

the biased distribution is given as

$$\omega(\mathbf{x}^*) = \frac{\exp(-\frac{1}{2\sigma^2}\sum_{i=0}^{N-1}(x_i^*+1)^2)}{\exp(-\frac{1}{2\sigma^2}\sum_{i=0}^{N-1}(x_i^*-(-1+\mu_i))^2)}. \tag{6.14}$$

Moreover, to ensure that the mean of the biased density is on the boundary of the error region, the value of μ_i will be set as follows. If the ith bit is in the TS, $\mu_i = \mu$; otherwise $\mu_i = 0$. Usually, applying the bisection method within an interval $[\mu_l, \mu_h]$, an optimal μ can be obtained by running the IS simulations until the ratio of decoding errors over the total number of trials for the specific TS is about 0.5 [70, 73]. Further, in order to achieve a robust estimation, the IS simulation for a representative TS from each class will be continued until (6.11) is satisfied.

Finally, the cumulative BLER of each class is obtained by multiplying the simulated error probability of its representative with the total number of elements in the class, and the overall BLER of the LDPC code is calculated by adding the cumulative BLERs of all classes.

Unfortunately, the IS scheme proposed for regular LDPC codes is not applicable to irregular codes. The main reason is that, as shown in the previous chapter, TSs with the same label $[w; u]$ can be configured very differently in terms of their induced connected subgraphs.

6.2 Our proposed IS scheme for LDPC codes

Our proposed scheme consists of three main stages. First, we search for the detrimental primary trapping set-induced connected subgraphs (PTS-ICSs) in the given code, i.e., those $[w; u; e]$ PTS-ICSs with small w, relatively smaller u and $e \geq 0$. Then, we classify the detrimental PTS-ICSs into groups. Finally, we apply IS to a representative from each PTS-ICS group and also to all elements in some selected groups. Based on the IS simulation results, we estimate the BLER of the code at the high SNR region.

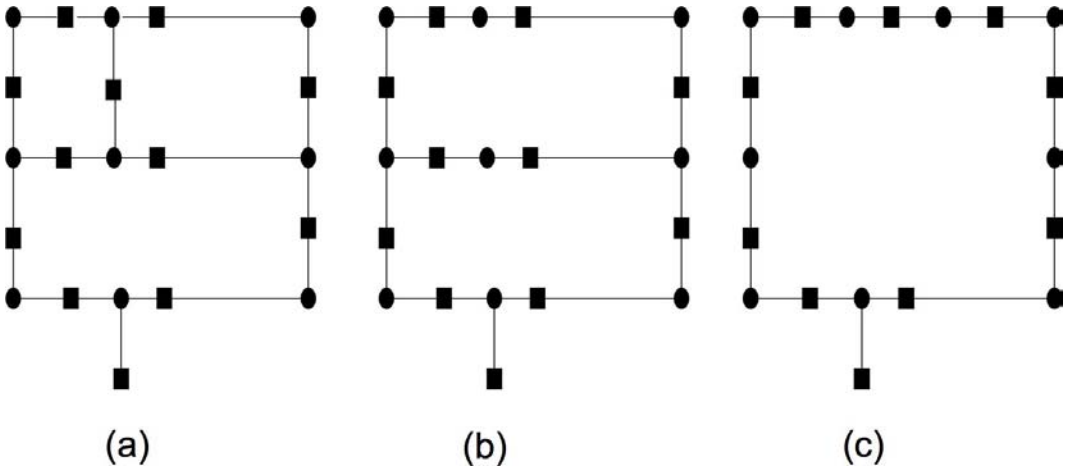


Figure 6.1: Three different induced connected subgraphs of a $[9; 1]$ trapping set. (a) A $[9; 1; 4]$ PTS-ICS contains three distinguishable cycles; (b) a $[9; 1; 2]$ PTS-ICS contains two distinguishable cycles; (c) a $[9; 1; 0]$ PTS-ICS contains a single cycle. Filled circles denote variable nodes and filled squares denote check nodes.

6.2.1 Search for PTS-ICSs

The first step of our proposed IS scheme is to search as many detrimental PTS-ICSs as possible within the code graph by using a three-stage search method.

6.2.1.1 Step one

In Sect. 2.4.2, "approximate cycle extrinsic message degree (ACE)" has been defined as a metric to measure the number of check nodes singly connected to a cycle. In particular, the ACE of a cycle with length $2l$ equals $\sum_{i=1}^{i=l}(d_i-2)$, where d_i is the degree of the ith variable node in this cycle. Moreover, detrimental PTS-ICSs contain a single cycle or multiple cycles formed by a combination of short cycles. Thus, identifying short cycles with few extrinsic edges will facilitate us searching for detrimental PTS-ICSs. In other words, we should search for short cycles with small ACE values, which can accomplished using the method in Sect. 2.4.2. In the following, we briefly outline the procedures.

- Set the depth threshold (θ) and the ACE threshold (χ).
- Sort the variable nodes according to their degrees in non-decreasing order, i.e, $d_i \leq d_j$, if $i < j$.

- Initialize $i = 1$.
- A subgraph is obtained by traversing the bipartite graph of the LDPC codes breadth-first from the ith variable node to depth θ.
- Search the subgraph and list all the cycles originating from and terminated at the ith variable node with ACE less than χ.
- Set $i = i + 1$ and repeat searching for the cycles, until all the variable nodes have been considered, i.e., $i = N$.

Note that to avoid repetitive counting, the subgraph expanded from the ith variable node excludes all the variable nodes with index less than i. At the end of the procedures, we will have recorded a number of (single) short cycles with ACE values less than χ. These cycles are indeed PTS-ICSs with a single cycle.

From the results in [15], it can be observed that all of the variable-degree distributions optimized by the density evolution algorithm contain degree-2 variable nodes. However, an irregular code containing degree-2 variable nodes, if not well constructed, may contain many detrimental $[w; u; e]$ PTS-ICSs with small w. Suppose that the single-cycle and multiple-cycle PTS-ICSs of size within $\mathcal{W}_s$ and $\mathcal{W}_m$ are relatively more harmful to the decoder. Then, in order to find out the most detrimental PTS-ICSs, we need to set the depth threshold (θ) to be large enough such that single-cycle $[w; u; e]$ PTS-ICSs with $w \leq \mathcal{W}_s$ and multiple-cycle $[w; u; e]$ PTS-ICSs with $w \leq \mathcal{W}_m$ can be found in future steps. Afterward, χ would be set to strike a balance between the computation time and accuracy in a heuristic manner. In the following, we will derive the bound of the depth threshold for given irregular LDPC codes when $\mathcal{W}_s$ and $\mathcal{W}_m$ are provided.

First, we consider the PTS-ICSs with single cycles. To ensure the identification of all possible single-cycle PTS-ICSs of size within $\mathcal{W}_s$, it can be readily shown that the depth threshold should be set larger than $\mathcal{W}_s$, i.e.,

$$\theta \geq \mathcal{W}_s + 1. \tag{6.15}$$

Next, we consider the PTS-ICSs with multiple cycles. The PTS-ICSs with multiple cycles can be linked together by one or more common variable nodes. If we intend

to identify all possible $[\mathcal{W}_m; u; e]$ PTS-ICSs, the depth threshold should be set large enough to cover at least one of the cycles in the PTS-ICSs. In other words, the depth should be larger than or equal to half of the largest achievable smallest cycle length over all possible configurations of the PTS-ICSs. Denote the smallest cycle in a $[\mathcal{W}_m; u; e]$ PTS-ICS by $l_{\min}$ and the length of the cycle $l_{\min}$ by $||l_{\min}||$. We randomly select a T-type variable node in the cycle $l_{\min}$ as the root node R. Consider a two-cycle cycle set in the $[\mathcal{W}_m; u; e]$ PTS-ICS with the node R being the common node and one cycle being $l_{\min}$ (see Fig. 6.2). The size of the cycle set is denoted by $\mathcal{W}$, where

$$\mathcal{W} \leq \mathcal{W}_m. \tag{6.16}$$

The equality holds if and only if the PTS-ICS is composed of the two-cycle cycle set and some extra edges. Consider the cycle set in Fig. 6.2(a). It consumes three edges from the root variable node R. Given the minimum cycle length $||l_{\min}||$. Based on Theorem 5.1 in Chapter 5. The size of cycle set is bounded by

$$\mathcal{W} \geq \frac{||l_{\min}||}{2} + \left(\left\lceil \frac{||l_{\min}||}{4} \right\rceil - 1 \right) \tag{6.17}$$

From (6.16) and (6.17), we have

$$\mathcal{W}_m \geq \frac{||l_{\min}||}{2} + \left(\left\lceil \frac{||l_{\min}||}{4} \right\rceil - 1 \right). \tag{6.18}$$

Denote the largest integer $||l_{\min}||$ satisfying the inequality (6.18) by $||l^{*a}_{\min}||$. We further consider the cycle set in Fig. 6.2(b), which has two cycles sharing only one variable node. Recall that $l_{\min}$ is the smallest cycle in the PTS-ICS. Denote the other cycle by l_{cycle} and its length by $||l_{\text{cycle}}||$. Hence, we have

$$||l_{\text{cycle}}|| \geq ||l_{\min}||. \tag{6.19}$$

In addition, the total number of variable nodes contained in the cycle set, i.e., the size of the cycle set, is the sum of the number of variable nodes in cycle l_{cycle} and the number of variable nodes in cycle $l_{\min}$ minus 1. Thus,

$$\frac{||l_{\text{cycle}}||}{2} + \frac{||l_{\min}||}{2} - 1 = \mathcal{W} \tag{6.20}$$

From (6.16), (6.19) and (6.20), we obtain

$$\begin{aligned} ||l_{\min}|| &\leq \mathcal{W}+1 \\ \Rightarrow ||l_{\min}|| &\leq \mathcal{W}_m+1. \end{aligned} \tag{6.21}$$

Denote the largest integer satisfying the above inequality by $||l^{*b}_{\min}||$. Then $||l^{*b}_{\min}|| = \mathcal{W}_m + 1$. Note also that if we substitute $||l_{\min}|| = \mathcal{W}_m + 1$ into (6.18), the inequality is satisfied. Thus, we can conclude that $||l^{*a}_{\min}|| \geq ||l^{*b}_{\min}||$ and that the largest achievable smallest cycle length over all possible $[\mathcal{W}_m; u; e]$ PTS-ICSs, denoted by $||l^{*}_{\min}||$, equals

$$||l^{*}_{\min}|| = \max(||l^{*a}_{\min}||, ||l^{*b}_{\min}||) = ||l^{*a}_{\min}||. \tag{6.22}$$

Finally, combining all the above results, the depth threshold θ should therefore be set to

$$\theta \geq \max(\mathcal{W}_s + 1, \frac{||l^{*}_{\min}||}{2} + 1), \tag{6.23}$$

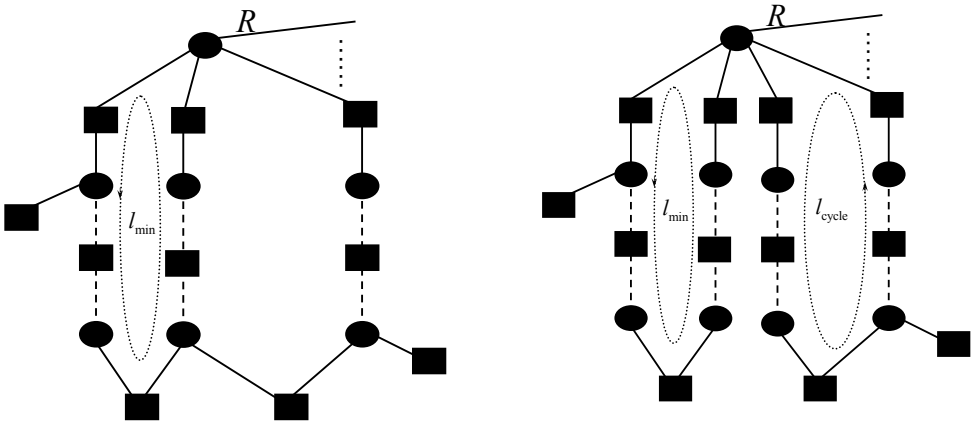
where $||l^{*}_{\min}||$ is the largest integer $||l_{\min}||$ satisfying (6.18).

6.2.1.2 Step two

Suppose W single cycles have been discovered in the previous step. To identify all the possible detrimental PTS-ICSs, one way is to consider the single cycles under all kinds of combinations, which will result a total of $\binom{W}{2} + \binom{W}{3} + ... \binom{W}{W}$ possibilities. However, this is a prohibitive number due to the very large W. Instead of using the above computation-intensive method, we apply a multi-bit error impulse technique which creates impulse errors to all the variable nodes in a single cycle. Details of the technique is described as follows.

Consider one of the single cycles. We apply some unnatural noises, called "error impulses" with amplitude ε, to all the bit positions in the cycle. We also scale the remaining bits in the codeword by a relatively smaller parameter α [73]. Without lost of generality, we assume that the first few bits of the codeword are involved in the single cycle. Then, when an all-zero codeword is transmitted, the deterministic input to the decoder can be represented by

$$\mathbf{x}_e = [\underbrace{-1+\varepsilon, -1+\varepsilon, \ldots, -1+\varepsilon}_{\text{bits in the single cycle}}, \underbrace{-\alpha, -\alpha, \ldots, -\alpha, -\alpha}_{\text{bits not in the single cycle}}]. \tag{6.24}$$



(a) (b)

Figure 6.2: Examples of two possible configurations of cycle set comprising: (a) two cycles with one common path (the path has at least one edge); (b) two cycles with one common node, i.e., root node R.

The error impulse ε has to be larger than one such that the bits in the single cycle will change state. Further, the scaling parameter α should be positive such that the bits not in the single cycle will be decoded correctly when the initial hard decisions are made. Yet, α should be smaller than one to (i) allow the fallacious information from bits in the single cycle to thoroughly propagate further out to nodes outside the cycle; and (ii) reduce the amplitude of the valid information trying to correct the variable nodes in the single cycle. Then, we run the decoder with the deterministic input $\mathbf{x}_e$. The hard decision vector $\hat{\mathbf{x}}_l$ at each iteration is recorded until the decoder reaches the maximum number of iteration and fails to find the valid codeword, where $l = 1, 2, \ldots, I_{\max}$ and $I_{\max}$ denotes the maximum iteration number.

Note that when a very small α and very large ε is selected for the deterministic input, more bits in the decoder are likely to become incorrect, leading to the discovery of a very large number of $[w; u; e]$ PTS-ICSs and a very long searching time. Thus, to achieve a balance between efficiency and effectiveness of the search method, the parameters ε and α should be selected with care.

Compared with the search method proposed by Cole *et al.* [73], our method, i.e., applying error impulses to single cycles, makes the decoder more likely to fail on detrimental $[w; u; e]$ PTS-ICSs consisting of 2 or 3 "distinguishable" cycles.

However, both our method and Cole's search method are not capable of identifying $[w; u; e]$ PTS-ICSs with (i) 5 or more "distinguished" cycles linked by several common nodes, as presented in Fig. 6.3 or (ii) a relatively large u, e.g., $u > 4$. Furthermore, we observe that when we apply our multi-bit error impulse approach to single cycles involved in the aforementioned two categories of PTS-ICSs, the decoder either succeeds in decoding or fails to decode without affecting many "non-error-impulsed" bits, regardless of the combination of ε and α. Thus, the error rate associated with those PTS-ICSs may be seriously underestimated, resulting in an underestimated BLER if such detrimental PTSs contribute significantly to the error floor. Fortunately, the PTS-ICSs in the former category (Category (i)) usually has a large w, implying that the errors are of low probability to occur in the PTSs at high SNR. For PTSs in the latter category (Category (ii)), a relatively large u means that there is adequate check-node information to correct the errors in the PTSs. Thus in both situations, each individual $[w; u; e]$ PTS-ICS contributes very little to the overall error probability. Nonetheless, if the group size of such PTS-ICSs is extremely large, the overall error contribution from all such PTS-ICSs might still be non-negligible. Finally, irregular LDPC codes built on optimized degree distributions usually have large proportions of variable nodes with degree less than 5. Under such circumstances, most of the detrimental $[w; u; e]$ PTS-ICSs consist of 2 or 3 "distinguishable" cycles with $u \leq 4$ and they will dominate the error probability at the high SNR region. For these cases, our method will be able to identify most of such PTS-ICSs.

6.2.1.3 Step three

In the previous step, we have recorded sequences of hard decision vectors when the decoder fails to converge. Consider a particular sequence. For each vector $\hat{\mathbf{x}}_I$ $(I = 1, 2, \ldots, I_{\max})$ in the sequence, a PTS or PTS-ICS can be observed by considering the set of variable nodes decoded as "1". However, to locate the PTS or PTS-ICS most detrimental to the decoder, not only the final state of the decoder but also the sequence of hard decision vectors should be taken into account. In [73], the authors have selected the target TS as the set of variable nodes corresponding to

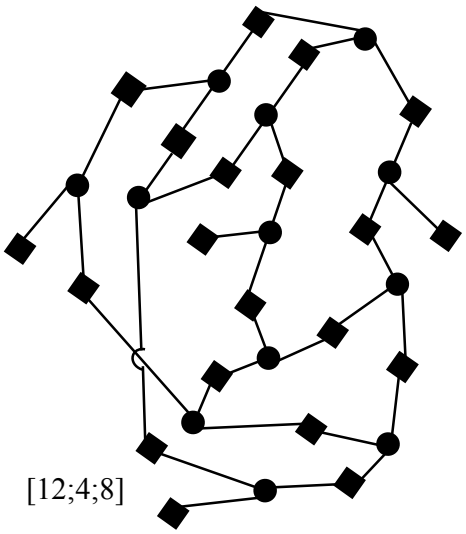


Figure 6.3: A $[12; 4; 8]$ PTS-ICS containing 5 "distinguishable" cycles.

bits decoded as "1" in $\tilde{\mathbf{x}}$, where

$$\tilde{\mathbf{x}} = \min_{l} \omega_H(\hat{\mathbf{x}}_l \boldsymbol{H}^T) \tag{6.25}$$

Here, $\boldsymbol{H}$ represents the code matrix and $\omega_H(\cdot)$ denotes the hamming weight of a vector. But then, in this manner, some redundant TSs or TS-ICSs may be recorded. For instance, suppose in two different vector sequences, two target TSs are found and their induced connected subgraphs are as shown in Fig. 6.4(a) and (b). First, we observe that in Fig. 6.4(a), there are two variable nodes lying outside the multiple-cycle TS-ICS. These nodes, being outside the multiple-cycle TS-ICS, has minimal effect in enhancing the erroneous information within the cycles and are therefore not playing any significant role in this particular $[11; 1; 2]$ PTS-ICS. When these two variables nodes and their associated connections are removed, the $[11; 1; 2]$ TS-ICS becomes exactly the same as the $[9; 1; 2]$ TS-ICS in Fig. 6.4(b). As a consequence, the $[11; 1; 2]$ TS-ICS should be removed from the list of detrimental TS-ICSs if the $[9; 1; 2]$ TS-ICS is already in the list.

In our proposed method, to ensure that redundant PTSs or PTS-ICSs will not be counted, we select PTS-ICSs as follows. First, we select $\tilde{\boldsymbol{x}}$ using (6.25). Then,

we form the PTS-ICS from the set of variable nodes decoded as "1" in $\tilde{\boldsymbol{x}}$. Next, we modify the PTS-ICS by removing all the variable nodes not in the cycles and their associated edges. Finally, if the modified PTS-ICS does not exist in our detrimental-PTS-ICS list, it is added to the list. By the end of this stage, we will have compiled a list of detrimental PTS-ICSs.

6.2.2 Classification of detrimental PTS-ICSs

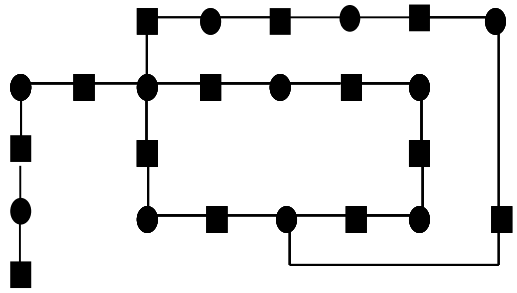
Assume that Ω detrimental PTS-ICSs have been found. We then classify the PTS-ICSs into the same group if they have the same label $[w; u; e]$. Suppose a total of $\mathbb{D}$ PTS-ICS groups are formed. We can label these groups with $[w_n; u_n; e_n]$ where $n = 1, 2, \ldots, \mathbb{D}$. Note also that there is no guarantee that PTS-ICSs with the same label $[w_n; u_n; e_n]$ (i.e., in the same PTS-ICS group) are configured equivalently under the graph of irregular codes. Fig. 6.4 (b) and (c) illustrates two $[9; 1; 2]$ PTS-ICSs with different configurations. The PTS-ICS in Fig. 6.4 (b) contains one degree-4, one degree-3 and seven degree-2 variable nodes, while the PTS-ICS in Fig. 6.4 (c) possesses three degree-3 and six degree-2 variable nodes. Further delicate classifications of the PTS-ICSs may result a better accuracy in error-floor prediction, but also give rise to higher computation complexity.

6.2.3 Implementation of IS on each PTS-ICS

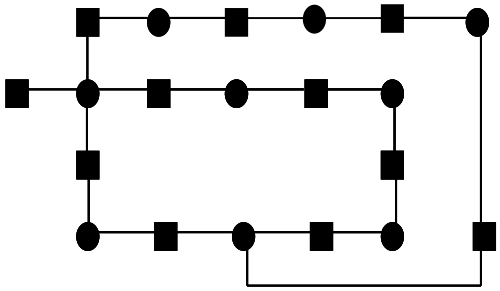
From our simulation results in Sect. 6.3.2, we find that error probabilities of PTS-ICSs in the same group, though may not be the same, lie within the same order. Based on the observation, we propose a two step approach in providing a better estimation of the error-floor without delicate classifications of PTS-ICSs.

6.2.3.1 Rough estimation

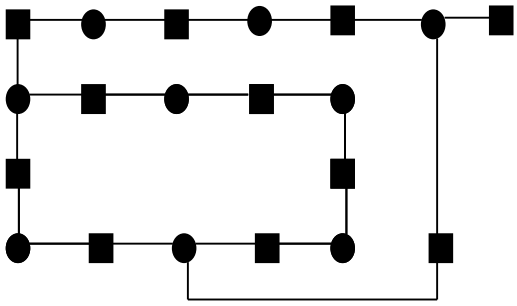
For each group of PTS-ICS, we randomly select one representative and use an IS simulator to estimate its BLER. Hence, we obtain a total of $\mathbb{D}$ different error probabilities, denoted by $\check{P}_{\text{IS}}(n), n = 1, 2, \ldots, \mathbb{D}$.



(a)



(b)



(c)

Figure 6.4: The (a) $[11;1;2]$ PTS-ICS and (b) $[9;1;2]$ PTS-ICS and (c) $[9;1;2]$ PTS-ICS.

6.2.3.2 Fine estimation

Suppose the largest value among the BLERs found in the previous step ($\check{P}_{\mathrm{IS}}(n)$, $n = 1, 2, \ldots, \mathbb{D}$) is of the order $10^{-\mathbb{I}}$, where $\mathbb{I}$ is a positive integer. We then further consider groups with representatives giving rise to error probabilities larger than $10^{-\mathbb{I}-1}$. Denote the set of the indices of such groups by $\mathbb{S}_{\mathrm{fine}}$. For Group n' where $n' \in \mathbb{S}_{\mathrm{fine}}$, we will apply IS simulator to estimate the error probabilities of all its PTS-ICS elements. Denoting the error probability of the kth element in Group n' by $\check{P}_{\mathrm{IS}}(n', k)$, the overall BLER, $\hat{P}_{\mathrm{IS}}$, of the code can then be estimated using

$$\hat{P}_{IS} = \sum_{n' \in \mathbb{S}_{\mathrm{fine}}} \sum_{k} \check{P}_{\mathrm{IS}}(n', k) + \sum_{n=1, n \notin \mathbb{S}_{\mathrm{fine}}}^{\mathbb{D}} \mathbb{M}_n \check{P}_{\mathrm{IS}}(n) \tag{6.26}$$

where $\mathbb{M}_n$ represents the multiplicities of Group n.

6.3 Results and discussion

Although the proposed importance sampling (IS) approach in conjunction with PTS identification aims at evaluating the BLER of *irregular* LDPC codes, it is readily applied to evaluate the BLER of *regular* ones. In the following, we show some of the results when the proposed method is applied to regular and irregular LDPC codes.

6.3.1 Regular codes

We first study the short-length, regular code "Mackay $(3; 6)$" [97] which has a variable-node degree d_v of 3, a check-node degree d_c of 6 and a block length N of 1008. Using the program provided in [97], we search for the smallest cycle length and the second smallest cycle length in the bipartite graph of the regular code. We then discover seven cycles with the smallest cycle length of 8 and more than 1000 cycles with the second smallest cycle length of 10.

Denoting the length of the second smallest cycle as 2ι, we can safely assume that there is a very small probability for a detrimental PTS-ICS to be formed by cycles with lengths all larger than 2ι. Thus, when we search for detrimental PTS-

ICSs using our proposed IS scheme in Sect. 6.2.1, we only need to set the depth threshold θ to $\iota + 1$ to ensure that all single cycles with length 2ι are to be found. Moreover, the corresponding ACE value of a cycle of length 2ι can be readily shown equal to $(d_v - 2)\iota$. Since the ACE threshold χ should be larger than $(d_v - 2)\iota$, we can set χ to

$$\chi = (d_v - 2)\iota + 1 = (d_v - 2)(\theta - 1) + 1. \tag{6.27}$$

In the "Mackay $(3; 6)$" regular code being studied, $2\iota = 10$ leads to a depth threshold of $\theta = \iota + 1 = 6$ and an ACE threshold of $\chi = (d_v - 2) \times 5 + 1 = 6$.

We apply the depth threshold and the ACE threshold found above to Step One in Sect. 6.2.1 to search for single cycles with ACE less than χ. Denote S_1 as the set containing sets of variable nodes forming such single cycles. The size of S_1, denoted by $|S_1|$ (also equal to W), is found to be $10,833$. Then a total number of $10,833$ trials have to be conducted in Step Two with the application of impulse errors to each element of S_1. At each decoding, the decoder will try to recover the all-zero message from the deterministic input parameterized by ε and α. Note that another parameter, namely the SNR, is required in the BP decoder for message initialization [79]. Since the SNR does not affect the BP decoding significantly, we set it arbitrarily to 6 dB here. For each input vector which is only parameterized by ε and α, the BP decoder will iterate with a maximum of 30 times. Three groups of search parameters (ε, α) used for S_1, as well as the mean number of iterations $\bar{N}_{\text{it}}$ taken at each decoding associated with each set of search parameters, are listed in Table 6.1. Moreover, the detrimental PTS-ICSs found and their numbers are also shown in the same table. At $\varepsilon = 2.6$ and $\alpha = 0.4$, relatively fewer detrimental PTS-ICSs have been discovered with a mean of 22 iterations taken to decode a message block. As ε is reduced to 2.1, a smaller α is used to trigger the decoder to fail and an increase number of detrimental PTS-ICSs is found. It takes 18 iterations on average to decode, leading to a relatively shorter running time of the program.

For comparison purpose, we also make use of the "four-variable-node" combinations [73] proposed for evaluating regular LDPC codes as the basis for searching PTS-ICSs. According to [73], there is a total of $(d_c - 1)^{d_v}$ possible "four-variable-node" combinations for every variable node in the regular code, with the variable node being considered at the center. Denoting S_2 as the set consisting of all such "four-variable-node" combinations in the regular code, the size of S_2, represented

Table 6.1: Parameters used for finding the detrimental PTS-ICSs in the "MacKay (3;6)" code and the number of detrimental PTS-ICSs returned.

	Proposed Scheme Based on Single Short Cycles S_1			**"Four-variable-node" Combinations S_2**		
Set Size $\lvert S_\kappa \rvert$	$10,833$			$126,000$		
(ε, α)	$(2.6, 0.4)$	$(2.5, 0.30)$	$(2.1, 0.20)$	$(4.2, 0.60)$	$(3.8, 0.40)$	$(2.5, 0.20)$
$\bar{N}_{\text{it}}$	22	15	18	7	7	9
$\bar{N}_{\text{it}} \times \lvert S_\kappa \rvert$	$238,326$	$162,495$	$194,994$	$882,000$	$882,000$	$1,134,000$
$[9;3;6]$ PTS-ICSs	1	1	1	1	1	1
$[11;3;8]$ PTS-ICSs	5	8	8	0	8	9
$[13;3;10]$ PTS-ICSs	0	3	12	0	2	12
$[6;4;2]$ PTS-ICSs	1	1	1	1	1	1
$[8;4;4]$ PTS-ICSs	48	44	43	29	48	47
$[10;4;6]$ PTS-ICSs	89	82	95	0	89	117
$[12;4;8]$ PTS-ICSs	4	25	52	0	4	37

by $|S_2|$, can be readily shown equal to $|S_2| = N(d_c - 1)^{d_v}$. For the regular code "Mackay $(3; 6)$" being considered, $|S_2| = 1008 \times (6-1)^3 = 126,000$. The set S_2 will then pass to Step Two, in which impulse errors will be applied to each element of S_2. The parameters (ε, α) used for S_2 and the corresponding results are listed in Table 6.1. According the results, larger multiplicities of detrimental PTS-ICSs are returned from the search when $(\varepsilon, \alpha) = (2.5, 0.20)$.

Comparing the results in Table 6.1, firstly we observe that the size of the single-short-cycle set (i.e., $S_1 = 10,833$) is much smaller than the set containing "four-variable-node" combinations (i.e., $S_2 = 126,000$). It implies that the single-short-cycle set provides a much smaller search space for dominant error events. Secondly, although the elements in the single-short-cycle set S_1 require a relatively larger average number of iterations ($\bar{N}_{\text{it}}$) to converge compared with the "four-variable-node" combinations, the total number of iterations required ($\bar{N}_{\text{it}} \times |S_\kappa|$) to search for detrimental PTS-ICSs based on the single-cycle set is still significantly lower than that based on the "four-variable-node" combinations.

We then apply the IS simulation to a representative PTS-ICS from each of the PTS-ICS groups. We assume an all-zero codeword being transmitted. Also, the ith input is from shifted $-1 + z_i$ to $-1 + \mu_i + z_i$ ($i = 1, 2, ..., N$), where z_i is a random AWGN noise sample. To ensure that the mean of the biased density is on the boundary of the error region, the value of μ_i will be set as follows. If the ith bit is in the PTS-ICS, $\mu_i = \mu$; otherwise $\mu_i = 0$. The decoder will iterate a maximum of 50 times for each input vector. Further, it has been reported that for binary AWGN channel, the optimal biased-density center should reside close to the boundary between the error region and the region that can be decoded successfully [68]. Hence, we set $\mu = 1$ for $[w; 0; e]$ detrimental PTS-ICSs. For $[w; u; e]$ detrimental PTS-ICSs with $u > 0$, μ has to be bigger than one and a bisection technique is further applied to search for the optimal μ in (μ_l, μ_u) where μ_l and μ_u are set to 1 and 3.2, respectively.

Fig. 6.5 presents the block-error-rate (BLER) performance of code "MacKay (3;6)" over an AWGN channel when evaluated using the MC method and the proposed IS technique separately. When the MC method is used, no more than 50 iterations are allowed for each decoding and the simulations ends when 100 block errors have been collected. Analyzing the block errors occurring at an SNR of 3.0 dB

shows that 79 out of the 100 error events are caused by detrimental PTS-ICSs while the other 21 are not. Moreover, for the error events not caused by detrimental PTS-ICSs, the number of bit errors has remained over 100 for every iteration during the decoding process. On the other hand, the detrimental PTS-ICSs creating the 79 error events are found to have 49 different $[w; u; e]$ labels. They include one $[11; 3; 8]$ PTS-ICS, one $[8; 4; 8]$ PTS-ICS and 47 $[w; u; e]$ PTS-ICSs with $u > 4$. The error results reflect that when the SNR is not large enough (3.0 dB for this regular LDPC code), the majority of the error events are either caused by $[w; u; e]$ PTS-ICSs with $u > 4$ or caused by other reasons not related to PTS-ICSs. In consequence, our proposed IS scheme, which focuses on predicting the error floor based on $[w; u; e]$ PTS-ICSs with $u \leq 4$ (see Table 6.1), may not produce consistent results with those predicted by the MC method. Nonetheless, the proposed IS scheme will be more accurate in predicting the error floor when the SNR increases and most error events are due to $[w; u; e]$ PTS-ICSs with $u \leq 4$.

To further validate the accuracy of our proposed IS scheme in predicting the error floor of short-length, regular LDPC codes, we investigate the regular code "96-3-963" [97] which is known to have a high error floor. The code has a variable-node degree of 3, a check-node degree of 6 and block length of 96. For this code, the values of the parameters used in our proposed IS scheme to search for the detrimental PTS-ICSs are $\theta = 6$, $\chi = 6$, $\varepsilon = 1.8$ and $\alpha = 0.8$. Block error rates obtained by both the proposed IS and MC techniques are presented in Fig. 6.5. It can be seen that the BLERs predicted by the proposed IS scheme match with those found by the MC technique. Moreover, a closer look into the error events simulated in the MC technique further reviews that the errors are mainly caused by $[w; u; e]$ PTS-ICSs with $w < 9$, $u = 0, 1$, and $e > 0$. Based on the aforementioned results, we conclude that the proposed IS scheme can accurately predict the error floor of short-length LDPC codes when most error events are due to $[w; u; e]$ PTS-ICSs with small u.

6.3.2 Irregular codes

6.3.2.1 Rate 0.5 LDPC codes

First, we consider LDPC codes with rate 0.5. We apply the proposed PTS-ICSs search methods to three irregular LDPC codes: "PEG-only-DE15" ("PEGir-

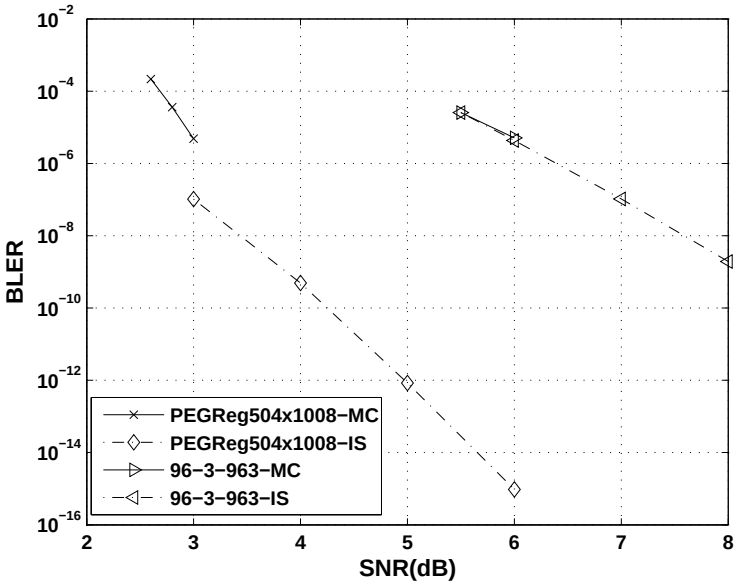


Figure 6.5: Block error rates obtained by the standard MC technique and our proposed IS technique for "MacKay $(3; 6)$" and code "96-3-963" codes.

Reg504x1008" from [97]), "PEG-ACSE-8-4-DE15" and "PEG-only-DE10". We run the MC simulations on each of the codes at SNR=2.8 dB until we have collected 100 block errors, which include the cases when the decoder fails to converge within 50 iterations. We observe that most of the block errors are due to single-cycle $[w; u; e]$ PTS-ICSs with $w < 11$ or multiple-cycle $[w; u; e]$ PTS-ICSs with $w < 17$, i.e., $\mathcal{W}_s = 10$ and $\mathcal{W}_m = 16$. According to (6.23) and (6.18) in Section 6.1.2, $||l_{\min}|| = 22$ and $\theta \geq \max(10 + 1, 11 + 1) = 12$. Moreover, among the multiple-cycle PTS-ICSs related to the block errors, most are formed by combinations of single cycles with ACE values not larger than 6. Therefore, during the search for single cycles in the codes, we set $\theta = 12$ and $\chi = 6$ to ensure that all the detrimental PTS-ICSs mentioned above can be captured. Next, we select ε and α with an aim to maximizing the number of detrimental multiple-cycle PTS-ICSs found. The mechanism of selecting parameters ε and α at Stage Two is the same as that described in the previous section. In Table 6.2, we present the parameters used and the number of single-cycles (W), the number of detrimental PTS-ICSs (Ω) and the number of PTS-ICS groups (D) found for the three irregular codes.

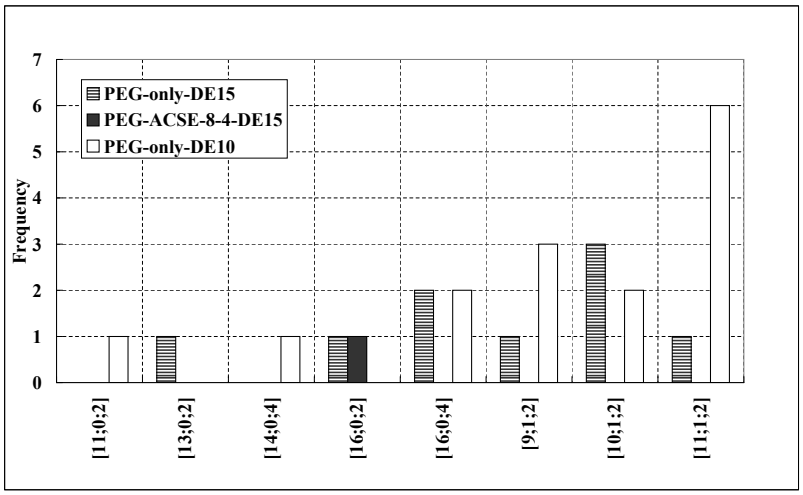
Table 6.3 further shows the multiplicities of single-cycle ($[w; u; 0]$) PTS-ICSs

Table 6.2: Parameters used for finding the detrimental PTS-ICSs in the irregular codes and the results returned.

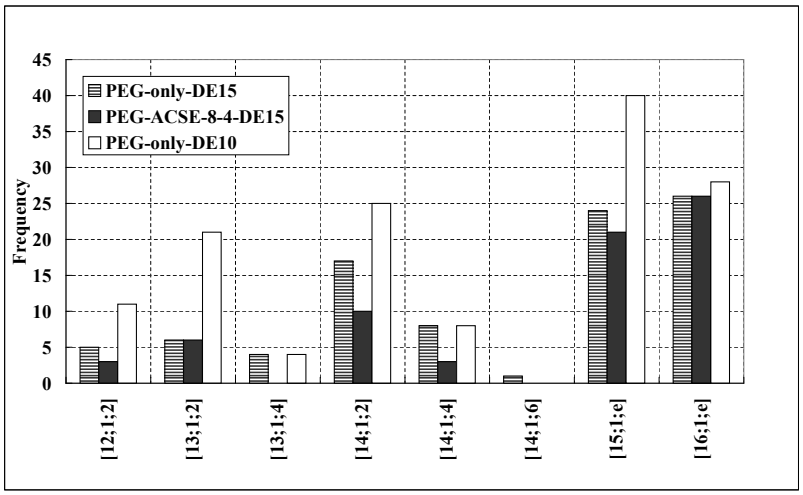
Code	**PEG-only-DE15**	**PEG-ACSE-DE15**	**PEG-only-DE10**
W	39,221	39,845	57,738
(ε, α)	$(1.7, 0.50)$	$(1.5, 0.35)$	$(1.7, 0.60)$
$\bar{N}_{\text{it}}$	13	12	12
Ω	$43,071$	$43,468$	$61,765$
D	155	153	138

with $w < 11$ and $u = 1, 2$ whereas Fig. 6.6 presents the multiplicities of multiple-cycle ($e > 0$) $[w; u; e]$ PTS-ICSs with $w < 17$ and $u = 0, 1$ found in the codes. From the results, we can see that Code "PEG-only-DE10" contains one $[11; 0; 2]$ PTS-ICS, three $[9; 1; 2]$ and two $[10; 1; 2]$ PTS-ICSs. Due to the existence of the $[11; 0; 2]$ PTS-ICS, we can conclude that the minimum hamming weight of Code "PEG-only-DE10" is no larger than 11. Furthermore, Code "PEG-only-DE15" possesses one $[13; 0; 2]$ PTS-ICS, one $[9; 1; 2]$ PTS-ICS and three $[10; 1; 2]$ PTS-ICSs. Since Code "PEG-only-DE15" does not seem to contain any $[w; 0; e]$ PTS-ICSs with $w < 13$, it might have a larger minimum hamming weight compared with Code "PEG-only-DE10". As for Code "PEG-ACSE-DE15", the minimum $[w; 0; e]$ PTS-ICS is possibly $[16; 0; 2]$, which suggests that the code might have the largest hamming weight among the three codes. Moreover, no $[w; 1; e]$ PTS-ICSs with w less than 12 have been detected in the same code. Since multiple-cycle PTS-ICSs contribute more to error floor than single-cycle PTS-ICSs, the aforementioned findings imply that at high SNR, Code "PEG-ACSE-DE15" should outperform Code "PEG-only-DE15", which in turn outperforms Code "PEG-only-DE10". The implication is also consistent with the conclusion drawn in Chapter 5.

Then, for each of the codes, we apply the IS simulation to a representative PTS-ICS from each of the PTS-ICS groups. As in Sect. 6.3.1, we set $\mu = 1$ for $[w; 0; e]$ detrimental PTS-ICSs and we use a bisection technique to search for the optimal μ for $[w; u; e]$ detrimental PTS-ICSs with $u > 0$. Moreover, for each individual PTS-ICS, the IS simulator is called "converged" if condition (6.11) is satisfied. Since our MC simulator continues until 100 block errors have been collected,



(a)



(b)

Figure 6.6: Multiplicities of multiple-cycle $[w; u; e]$ PTS-ICSs with $w < 17$ and $u = 0, 1$ ($e > 0$) in Codes "PEG-only-DE15", "PEG-ACSE-8-4-DE15" and "PEG-only-DE10".

Table 6.3: Multiplicities of single-cycle ($[w; u; 0]$) PTS-ICSs with $w < 11$ and $u = 1, 2$ in Codes "PEG-only-DE15", "PEG-ACSE-8-4-DE15" and "PEG-only-DE10".

Code	PEG-only -DE15	PEG-ACSE -8-4-DE15	PEG-only -DE10
$[7; 1; 0]$ PTS-ICS	3	0	9
$[8; 1; 0]$ PTS-ICS	5	6	2
$[9; 1; 0]$ PTS-ICS	1	3	2
$[10; 1; 0]$ PTS-ICS	13	19	9
$[6; 2; 0]$ PTS-ICS	0	0	57
$[7; 2; 0]$ PTS-ICS	89	0	207
$[8; 2; 0]$ PTS-ICS	126	58	163
$[9; 2; 0]$ PTS-ICS	63	58	105
$[10; 2; 0]$ PTS-ICS	132	121	103

i.e., $\hat{P}_{\text{MC}} N_{\text{MC}} = 100$, (6.11) can be written as

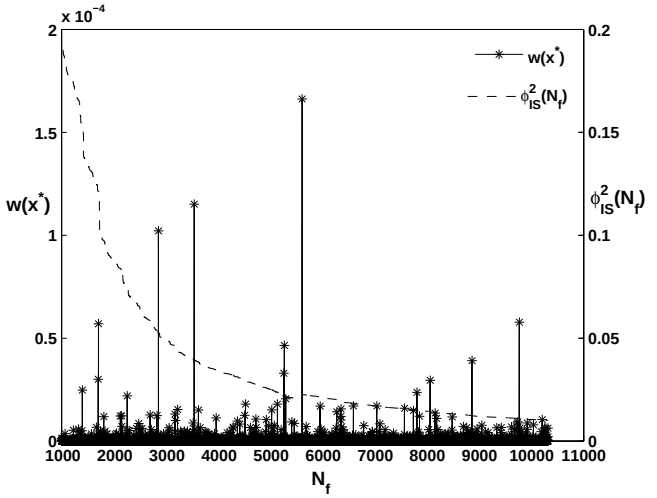
$$\phi^2_{\text{IS}}(N_{\text{IS}}) = \frac{\sum_1^{N_{\text{IS}}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]^2}{\{\sum_1^{N_{\text{IS}}} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]\}^2} - \frac{1}{N_{\text{IS}}} \leq 0.01. \tag{6.28}$$

Denote N_f as the number of decoding failures when the IS simulation progresses. Also, we define

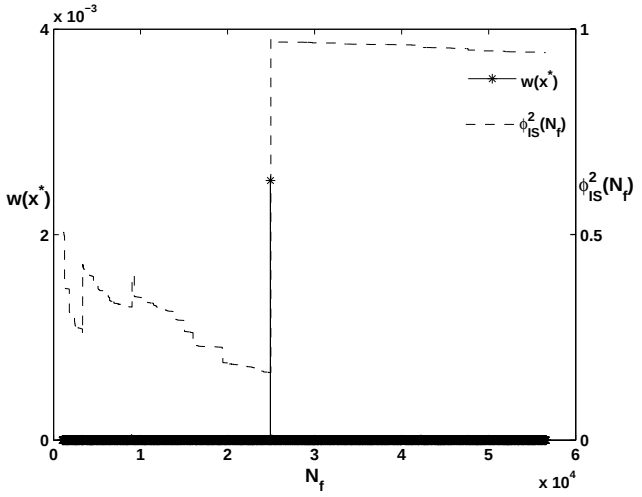
$$\phi^2_{\text{IS}}(N_f) = \frac{\sum_1^{N_f} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]^2}{\{\sum_1^{N_f} [1_{\mathcal{E}}(\mathbf{x}^*)\omega(\mathbf{x}^*)]\}^2} - \frac{1}{N_f}. \tag{6.29}$$

When the IS simulation applies to a PTS-ICS, we can observe the change of $\phi^2_{\text{IS}}(N_f)$ as N_f increases. For example, Fig. 6.7(a) plots the values of $\omega(\mathbf{x}^*)$ and $\phi^2_{\text{IS}}(N_f)$ versus N_f when IS simulation applies to a $[9; 1; 2]$ PTS-ICS in Code "PEG-only-DE15". The value of $\phi^2_{\text{IS}}(N_f)$ decreases steadily and reaches below 0.01 at $N_f \approx 10,000$.

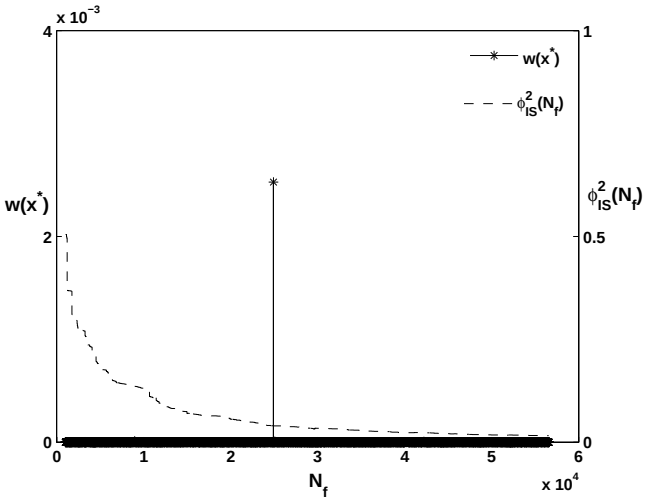
For some $[w; u; e]$ PTS-ICSs, especially $[w; u; e]$ PTS-ICSs with $u > 2$, we observe that when N_f increases, $\phi^2_{\text{IS}}(N_f)$ occasionally surges and fails to reach below 0.01 even when N_f is very large. In consequence, the estimated error rates obtained by applying IS on such PTS-ICSs keep changing as N_f increases. For



(a)



(b)



(c)

Figure 6.7: Plot of $\omega(\boldsymbol{x}^*)$ and ϕ^2_{IS} versus number of decoding failures for the Code "PEG-only-DE15" during an IS simulation. The IS simulator (a) converges to a $[9; 1; 2]$ PTS-ICS; (b) fails to converge to a $[8; 3; 2]$ PTS-ICS; (c) converges to a $[8; 3; 2]$ PTS-ICS after implementing the threshold restriction.

example, Fig. 6.7(b) plots the values of $\omega(\mathbf{x}^*)$ and $\phi^2_{\text{IS}}(N_f)$ when the IS simulation applies to a $[8;3;2]$ PTS-ICS. It is clearly seen that $\phi^2_{\text{IS}}(N_f)$ fails to converge even with 5.5×10^4 block errors. By investigating the biased inputs with noise at such instances, we find that such inputs actually fall into the domain of another PTS-ICS instead of the simulated one. In this particular case, the inputs have fallen into the domain of a nearby $[11;3;2]$ PTS-ICS instead of the intended $[8;3;2]$ PTS-ICS, as shown in Fig. 6.8. Such noise realizations should be discarded because only error events initiated by the intended PTS-ICS should be taken into account. To overcome the problem, we establish a threshold φ. If the percentage change in $\phi^2_{\text{IS}}(N_f)$ is greater than φ, i.e.,

$$\frac{\phi^2_{\text{IS}}(N_f) - \phi^2_{\text{IS}}(N_f - 1)}{\phi^2_{\text{IS}}(N_f - 1)} > \varphi, \tag{6.30}$$

$\omega(\boldsymbol{x}^*)$ at the current decoding step is discarded and another noise realization will be used. Here we set the threshold to be 10%. With the introduction of the new restriction, we run the IS simulation again for the same $[8;3;2]$ PTS-ICS. Results show that $\phi^2_{\text{IS}}(N_f)$ is capable of converging, as depicted in Fig. 6.7(c).

In Table 6.4, we present the error contributions of several PTSs in code "PEG-only-DE15" at SNR$=$ 3.0 dB. Recall that $\check{P}_{\text{IS}}(n', k)$ denotes the error probability of the k-th element in Group n'. We also denote $N_{\text{IS}}(n', k)$ as the number of IS simulation runs used to obtain the error probability. Moreover, $[w;u;e]_k$ represents the kth PTS from the same $[w;u;e]$-PTS class. The results have verified the conjecture that a $[w;u;e_2]$ PTS will contribute more to the error floor at high SNR than a $[w;u;e_1]$ PTS if $0 \leq e_1 < e_2$. For example, the error probability due to the [9;1;2] PTS is higher than that due to the [9;1;0] PTS. Furthermore, we observe that the error probabilities of PTS-ICSs in the same group may vary within the same order. For example, all $[10;1;2]_k$ PTS $(k = 1, 2, 3)$ produce errors in the order of 10^{-8}.

In Fig. 6.9, we present the BLERs of the three irregular codes estimated by both the standard MC technique and our proposed IS technique. It is clear that the Code 'PEG-ACSE-8-4-DE15" provides the best performance, while "PEG-only-DE10" is the worst. Consider further the error results for the Code 'PEG-ACSE-8-4-DE15". At an SNR of 5.5 dB, the proposed IS technique predicts an error rate of 10^{-13} for the code. Note that at SNR = 2.8 dB and 3.0 dB, there are discrepancies

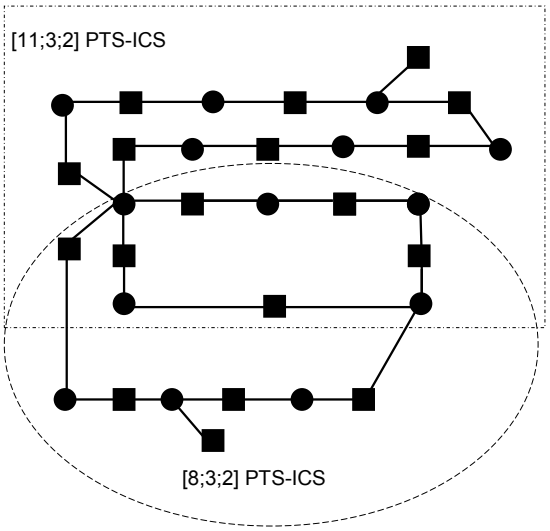


Figure 6.8: A $[8; 3; 2]$ PTS-ICS overlaps with a $[11; 3; 2]$ PTS-ICS. The filled circles denote the variable nodes and the filled squares denote the check nodes.

Table 6.4: The error contributions of several PTS-ICSs from code "PEG-only-DE15" at SNR$= 3.0$ dB. $\check{P}_{\mathrm{IS}}(n', k)$ denotes the error probability of the k-th element in Group n'. $N_{\mathrm{IS}}(n', k)$ denotes the number of IS simulation runs used to obtain the error probability.

PTS-ICS	$\check{P}_{\mathrm{IS}}(n', k)$	μ	$N_{\mathrm{IS}}(n', k)$
$[7; 1; 0]_1$	4.28×10^{-9}	1.66	$8,135$
$[8; 1; 0]_1$	1.03×10^{-9}	1.55	$10,954$
$[8; 1; 0]_2$	2.22×10^{-9}	1.55	$27,507$
$[9; 1; 0]_1$	2.21×10^{-9}	1.45	$54,307$
$[9; 1; 2]_1$	1.75×10^{-7}	1.28	$21,248$
$[10; 1; 0]_1$	2.74×10^{-10}	1.45	$38,425$
$[10; 1; 2]_1$	3.49×10^{-8}	1.28	$9,843$
$[10; 1; 2]_2$	4.45×10^{-8}	1.24	$7,522$
$[10; 1; 2]_3$	5.59×10^{-8}	1.24	$8,627$
$[11; 1; 0]_1$	2.68×10^{-10}	1.38	$15,190$
$[11; 1; 2]_1$	1.65×10^{-8}	1.26	$26,436$

between the BLERs found by the MC and IS techniques. The main reason is that at such (comparatively low) SNR levels, many of the error events are caused not by detrimental PTS-ICSs because the error floor is not yet reached. Nonetheless, the proposed IS scheme should be more accurate in predicting the error floor at higher SNR values.

Table 6.5 shows the number of simulation runs (up to 2 decimal places) for the standard MC technique and the proposed IS technique for the three codes at different SNR values. The standard MC simulator terminates until 100 block errors are collected. But for extremely low BLERs, instead of running the MC directly, the number of MC simulation runs (N_{MC}) is estimated using $100/\hat{P}_{\mathrm{IS}}$. The results show that the proposed IS technique can estimate the BLER with much less simulation runs. In particular, a speed-up gain of 3.87×10^{9} (= $3.51 \times 10^{15}/9.07 \times 10^{5}$) is achieved at SNR=$5.8$ dB for Code "PEG-ACSE-DE15".

6.3.2.2 Rate 0.75 LDPC codes

Next, we apply our proposed IS scheme to rate-0.75 LDPC codes: "PEG-only-DE14", "PEG-only-CDE12" and "PEG-only-CSF20". A close look into the decoding failures collected at 5.2 dB indicates that for such codes, most of the failures are caused by single-cycle $[w; u; e]$ PTS-ICSs with $w < 10$ and multiple-cycle $[w; u; e]$ PTS-ICSs with $w < 12$. The observations give $\mathcal{W}_s = 8$ and $\mathcal{W}_m = 11$. Substituting the values of $\mathcal{W}_s$ and $\mathcal{W}_m$ into (6.15) and (6.18), we have $||l^*_{\min}|| = 16$ and $\theta \geq \max(8+1, 16/2+1) = 9$. Moreover, among those PTS-ICSs related to the block errors, few are found containing single cycles with their ACE values larger than 4 for code "PEG-only-DE14", and 5 for codes "PEG-only-CDE12" and "PEG-only-CSF20". Therefore, we set $\theta = 9$ for all the three codes under study, $\chi = 4$ for code "PEG-only-DE14" and $\chi = 5$ for codes "PEG-only-CDE12" and "PEG-only-CSF20". The mechanism of selecting the parameters ε and α at Stage Two is the same as that used for rate-0.5 codes. In Table 6.6, we present the parameters used and the number of detrimental PTS-ICSs (Ω) and groups ($\mathbb{D}$) found for those rate-0.75 codes.

Fig. 6.10 presents the multiplicities of multiple-cycle ($e > 0$) $[w; u; e]$ PTS-ICSs with $w < 12$ and $u \in \{0, 1\}$ found in the codes. From the figure, we can see

Table 6.5: The number of simulation runs of standard MC technique and our proposed IS technique as well as the speed-up gains for codes "PEG-only-DE15", "PEG-ACSE-8-4-DE15" and "PEG-only-DE10". The symbol γ_s denotes the speed-up gain of IS compared to MC. The symbol $*$ indicates that the number of MC simulation runs (N_{MC}) is estimated using $100/\hat{P}_{\mathrm{IS}}$.

Name of code	SNR(dB)	N_{MC}	N_{IS}	γ_s
PEG-only-DE15	2.8	2.84×10^7	8.69×10^5	33
	3.0	6.50×10^7	7.58×10^5	86
	3.8	$1.83 \times 10^9*$	8.07×10^5	$2,266$
	4.8	$1.34 \times 10^{11}*$	5.97×10^5	2.24×10^5
	5.8	$2.58 \times 10^{13}*$	4.00×10^5	6.44×10^7
PEG-ACSE-8-4-DE15	2.8	5.00×10^7	1.19×10^6	42
	3.0	1.55×10^8	1.11×10^6	133
	3.8	$2.35 \times 10^{10}*$	1.03×10^6	2.29×10^4
	4.8	$4.95 \times 10^{12}*$	8.92×10^5	5.55×10^6
	5.8	$3.51 \times 10^{15}*$	9.07×10^5	3.79×10^9
PEG-only-DE10	2.8	1.01×10^7	5.49×10^5	18
	3.0	2.07×10^7	5.56×10^5	37
	3.8	$3.59 \times 10^8*$	5.38×10^5	666
	4.8	$1.32 \times 10^{10}*$	2.01×10^5	6.61×10^4
	5.8	$1.35 \times 10^{12}*$	1.68×10^5	8.03×10^6

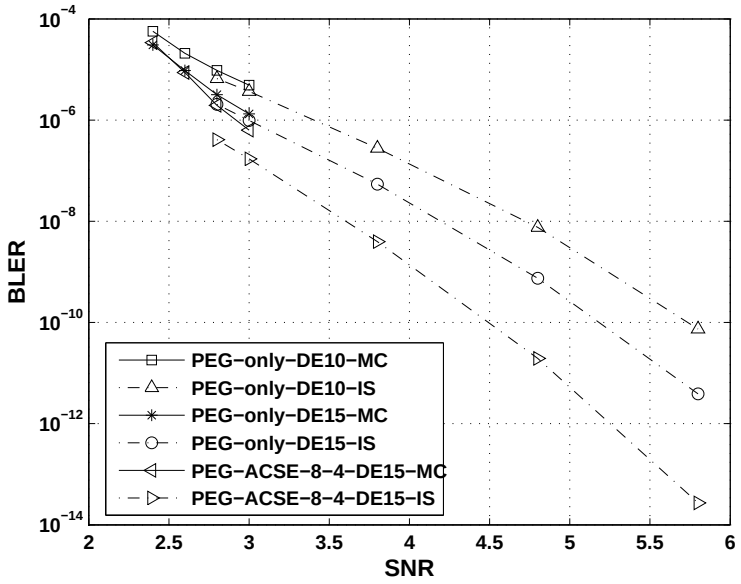
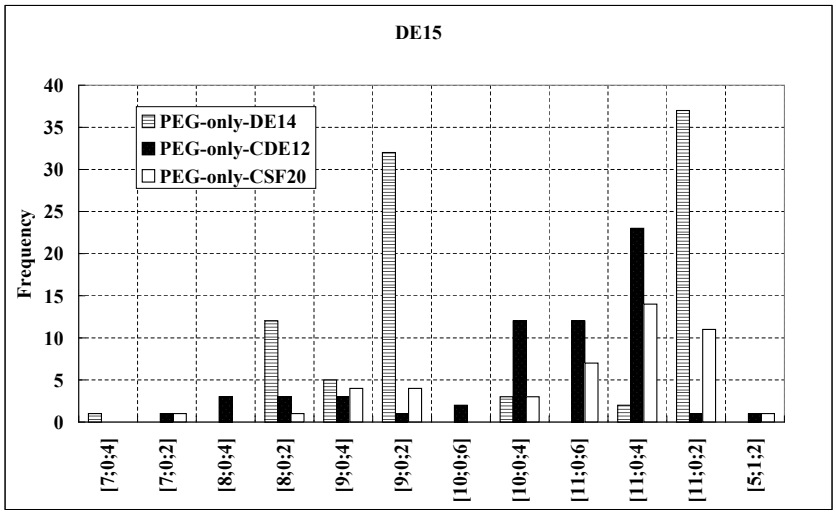


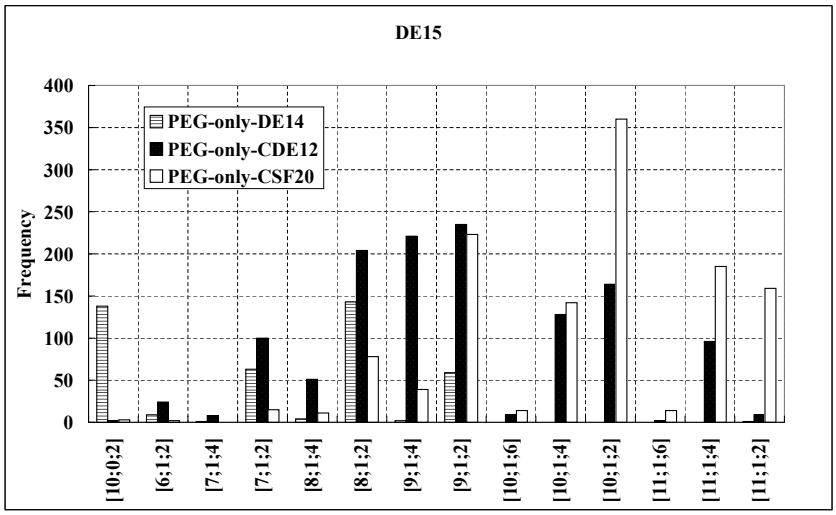
Figure 6.9: BLERs obtained by standard MC technique and our proposed IS technique for codes "PEG-only-DE15", "PEG-ACSE-8-4-DE15" and "PEG-only-DE10".

that code "PEG-only-DE14" contains one $[7; 0; 4]$ PTS-ICS, while code "PEG-only-CDE12" and code "PEG-only-CSF20" contain one $[7; 0; 2]$ PTS-ICS. The observations show that the minimum hamming weights of the three codes are no larger than 7. The figure further depicts that code "PEG-only-DE14" has the largest number of detrimental $[w; 0; e]$ PTS-ICSs among the three codes. In particular, it possesses one $[7; 0; 4]$, twelve $[8; 0; 2]$ PTS-ICSs and one hundred and thirty-eight $[10; 0; 2]$ PTS-ICSs which in total will contribute many errors to the decoder. Code "PEG-only-CDE12" has a comparable number of detrimental $[w; 0; e]$ PTS-ICSs as code "PEG-only-CSF20", but possesses more $[w; 1; e]$ PTS-ICSs. As a consequence, code "PEG-only-CSF20" should produce the best error performance at high SNR while code "PEG-only-DE14" should give the highest error floor. The inference is already verified by the results shown in Fig. 4.6 of Chapter 4.

In Fig. 6.11, we present the BLERs of the rate-0.75 codes estimated by both the standard MC technique and our proposed IS technique. It is observed that for code "PEG-only-DE14" which has a relatively higher error floor, the performance curve obtained from IS matches well with that from MC at 5.2 dB, 5.4 dB and 5.6 dB. For codes "PEG-only-CDE12" and "PEG-only-CSF20", a small gap is ob-



(a)



(b)

Figure 6.10: The multiplicities of $[w; u; e]$ PTS-ICSs with $w < 12$, $u \in \{0, 1\}$ and $e > 0$. Codes "PEG-only-DE14", "PEG-only-CDE12" and "PEG-only-CSF20" with rate 0.5 are used.

Table 6.6: Parameters Used and Results in the Search of detrimental PTS-ICSs in the irregular rate-0.75 codes.

Code	**PEG-only-DE14**	**PEG-only-CDE12**	**PEG-only-CSF20**
ε	1.28	1.40	1.37
α	0.65	0.50	0.55
Ω	345524	793275	302162
$\mathbb{D}$	58	111	97

served between the IS curve and the MC curve and it is reduced slightly as the SNR increases from 5.0 dB to 5.2 dB. Further, the performance curves show that constrained SF-LDPC code outperforms code "PEG-only-CDE12" by about 0.4 dB at BLER of 10^{-6}, which in turn outperforms code "PEG-only-DE14" by about 0.2 dB. We further analyze the estimated error contribution of each PTS at different SNR values. It is found at SNR = 8.0 dB (BLER of order 10^{-10}), the $[w; u; e]$ PTS-ICSs with $w < 8$, $u = 0, 1$ and $e \geq 0$ account for 56.34%, 83.03% and 95.83% of block errors for codes "PEG-only-DE14", "PEG-only-CDE12" and "PEG-only-CSF20", respectively. At SNR = 5.2 dB (BLER of order 10^{-5} or slightly lower than 10^{-5}), those PTS-ICSs only account for 46.96%, 49.55% and 59.55%. In other words, the error events at very high SNR, i.e., extremely lower block error rates, are more related to the detrimental $[w; u; e]$ PTS-ICSs with very small w, $u = 0, 1$ and $e \geq 0$. Under such circumstance, the proposed IS scheme should be more accurate in predicting the error floor.

The speed-up gains γ_s are listed in Table. 6.7. The table shows that compared to MC, our proposed IS scheme performs well for high rate codes by providing good estimations of BLER with much fewer simulation runs. A speed-up gain of 1.89×10^8 ($4.57 \times 10^{12}/24178$) is achieved at BLER 2.19×10^{-11} for code "PEG-ACSE-CSF20" with our proposed IS technique.

6.4 Summary

Given a particular LDPC code. We have developed in this chapter a three-step method to search for as many detrimental primary trapping set-induced connected

Table 6.7: The number of simulation runs of standard MC technique and our proposed IS technique as well as the speed-up gains for rate-0.75 codes "PEG-only-DE14", "PEG-only-CDE12" and "PEG-only-CSF20". The symbol $*$ indicates that the number of MC simulation runs (N_{MC}) is estimated using $100/\hat{P}_{\mathrm{IS}}$.

Name of code	SNR(dB)	N_{MC}	N_{IS}	γ_s
PEG-only-DE14	5.2	1.53×10^6	3.70×10^5	4
	5.4	5.51×10^6	4.13×10^5	13
	5.6	1.30×10^7	4.73×10^5	27
	6.6	$1.05 \times 10^9*$	2.15×10^5	$4,884$
	7.0	$7.30 \times 10^9*$	2.43×10^5	$32,304$
	8.0	$1.39 \times 10^{12}*$	1.35×10^5	1.03×10^7
PEG-only-CDE12	5.0	1.28×10^6	5.97×10^5	2
	5.2	2.84×10^6	9.13×10^5	3
	5.4	7.95×10^6	7.04×10^5	11
	6.6	$2.14 \times 10^9*$	3.11×10^5	$6,881$
	7.0	$1.25 \times 10^{10}*$	2.29×10^5	$54,585$
	8.0	$1.80 \times 10^{12}*$	2.59×10^5	6.98×10^6
PEG-only-CSF20	5.0	6.78×10^6	7.15×10^5	10
	5.2	1.48×10^7	7.38×10^5	20
	6.0	$6.65 \times 10^8*$	4.66×10^5	143
	7.0	$3.84 \times 10^{11}*$	7.35×10^5	8.69×10^6
	8.0	$4.57 \times 10^{12}*$	$24,178$	1.89×10^8

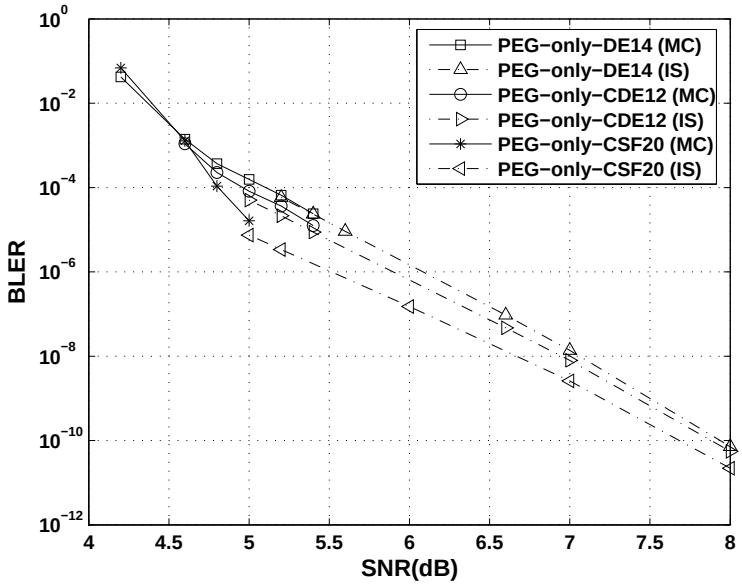


Figure 6.11: BLERs obtained by standard MC technique and our improved IS technique for rate-0.75 codes "PEG-only-DE14", "PEG-only-CDE12" and "PEG-only-CSF20".

subgraphs (PTS-ICSs) as possible in this code. Then we classify the PTS-ICSs based on the label $[w; u; e]$. Furthermore, we propose a two-step IS simulator to estimate the BLER of the code at the high signal-to-noise ratio (SNR) region. Results show that our proposed scheme can achieve a good accuracy of BLER estimation compared with the standard MC technique. In addition, speed-up gains of up to 3.87×10^9 times can be achieved.

Chapter 7

Conclusion and future work

7.1 Conclusions

In this thesis, we have presented a detailed study of short-length LDPC codes. First, we have analyzed the dynamical behavior of short-length LDPC decoders as the SNR varies. In particular, it has been found that fold bifurcations, flip bifurcations and Neimark-Sacker bifurcations occur within the waterfall region. For regions with high and low SNRs, two kinds of fixed points have been observed in the decoding system, namely the unequivocal fixed point and the indecisive fixed point. The decoder will converge to an unequivocal fixed point if it finds a valid codeword. We have also evaluated the eigenvalues from the Jacobian matrices at such fixed points for stability analysis. The indecisive fixed points are found to be unstable and will disappear as SNR increases. The unequivocal fixed points are stable in all range of SNR but their attraction bins shrink as SNR decreases. We have further analyzed the effectiveness of a spatial-delay feedback control technique in guiding the decoder to converge to the unequivocal fixed points. Results show that there is a small improvement in block error rates but no apparent improvement in terms of bit error rates.

Second, we have proposed a specific type of LDPC codes, namely scale-free LDPC (SF-LDPC) codes. Such codes are characterized by having variable-node degrees following power-law distributions. To produce codes with good performance,

the check nodes are designed to have only a few different degrees, which follow Poisson distributions. We have also evaluated the achievable error-correcting capability (threshold) of the SF-LDPC codes using density evolution. Results show that the optimized threshold values for the SF-LDPC codes are comparable to those for other best-known LDPC codes (less than 2% difference). Then several short-length SF-LDPC codes of various rates have been constructed using a popular algorithm called progressive edge growth (PEG). The simulation results show that short-length SF-LDPC codes outperform the other best-known codes by about 0.2 dB at high SNR under the same code length, same code rate and slightly less implementation complexity.

Moreover, we have observed in our simulations that short-length LDPC codes suffer from the problem of error floor. While trapping sets (TSs) are the main cause of the error floor, we have discovered that not all trapping sets are equally harmful. In addition, we classify the induced connected subgraphs (ICS) of TSs into those with (i) no cycle; (ii) single cycle and (iii) multiple cycles. By defining the node having more than two edges involved in the multiple cycles as a "T-type node", the TS-ICSs with multiple cycles are then further categorized according to the T-type nodes they contain, i.e., (i) all T-type nodes are variable nodes; (ii) all T-type nodes are check nodes; and (iii) both T-type variable nodes and T-type check nodes exist. We have found that the majority of error events at high SNR are due to the incidents that the decoder being trapped by some specific types of TSs, the ICSs of which containing single cycle or multiple cycles without T-type check nodes. Moreover, we have shown that using $[w; u]$ alone is not adequate to indicate the characteristics of a TS. (w denotes the number of variable nodes in the TS and u represents the number of check nodes with odd number of connections to the TS.) To overcome this problem, have refined the $[w; u]$ TS with the label $[w; u; e]$, where e the newly introduced parameter called "cycle-indicator". Furthermore, we define "primary TS (PTS)" and show that $[w; u; e]$ PTSs with small w, relatively smaller u and $e \geq 0$ are contributing to the error floor. Subsequently, we propose a code construction method that aims to avoid these PTSs. Then, several short-length irregular LDPC codes have been constructed based on the proposed method and they show superior performance at the high SNR region.

In order to effectively and efficiently evaluate the error rates of the codes at the

high SNR region, we have designed a novel scheme combining importance sampling (IS) and PTSs identification. The proposed IS scheme has been applied to both the regular and irregular LDPC codes. Results show that the error rate estimation is very good and a speed-up gain of 3.5184×10^9, compared with Monte Carlo simulations, is achieved at BLER 3.5151×10^{-15}.

7.2 Contributions of the thesis

- The dynamical behavior of short-length LDPC decoders has been characterized.
- A spatial-delay feedback technique has been applied to improve the error-correcting capability of short-length LDPC decoders.
- Scale-free LDPC codes have been proposed and optimized.
- Optimized scale-free LDPC codes have been evaluated analytically, constructed and further evaluated by simulations.
- Induced connection subgraphs (ICSs) of trapping sets are classified.
- Trapping sets are refined with the introduction of the new parameter "cycle indicator".
- A new method have been developed to build short-length LDPC codes with lower error floor.
- An improved importance sampling scheme has been designed to evaluated the extremely low error rates of LDPC codes at high SNR region.

7.3 Future work

Based on the findings in the thesis, we propose the following topics for future research.

- Application of complex-network theories to other error-correcting codes, such as LT codes.

- Study of the relationships between other properties of scale-free networks, e.g., clustering coefficient and betweenness, and the error-performance of short-length LDPC codes.
- Study the rate-compatible SF-LDPC codes.
- Implementation of SF-LDPC codes and rate-compatible SF-LDPC codes using hardware.
- Apply SF-LDPC codes to MIMO-OFDM system.
- Investigation of more accurate and efficient search method for detrimental PTS-ICSs.
- Study short-length Q-ary LDPC codes.
  - Optimization of the degree distributions of Q-ary LDPC codes
  - Investigation of factors determining the performance of short-length Q-ary LDPC codes
  - Study whether error floors exist in short-length Q-ary LDPC codes
- Investigate the relation between Turbo codes and LDPC codes and apply the idea of interleaver design to the design of short-length LDPC codes.

Bibliography

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

[2] D. Agrawal and A. Vardy. The turbo decoding algorithm and its phase trajectories. *IEEE Trans. Inform. Theory*, 47(2):699–722, 2001.

[3] B. Sklar. A primer on turbo code concepts. *IEEE Communications Magazine*, 35(12):94–102, 1997.

[4] R. Gallager. Low-density parity-check codes. *IEEE Trans. Inform. Theory*, 8(1):21–28, 1962.

[5] C.E. Shannon and W. Weaver. A mathematical theory of communications. *Bell System Technical Journal*, 27(2):632–656, 1948.

[6] N. Wiberg, H.A. Loeliger, and R. Kotter. Codes and iterative decoding on general graphs. In *Proc. IEEE Int. Symp. Inform. Theory*, pages 468–471. Whistler B.C, Canada, 1995.

[7] D.J.C Mackay and R.M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 33(6):457–458, 1997.

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

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

[10] M. Luby, M. Mitzenmacher, A. Shokrollah, and D. Spielman. Analysis of low density codes and improved designs using irregular graphs. In *Proc. the Thirtieth Annual ACM Symposium on Theory of computing*, pages 249–258. New York, USA, 1998.

[11] T.J. Richardson and R.L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Trans. Inform. Theory*, 47(2):599–618, 2001.

[12] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman. Efficient erasure correcting codes. *IEEE Trans. Inform. Theory*, 47(2):569–584, 2001.

[13] P. Oswald and A. Shokrollahi. Capacity-achieving sequences for the erasure channel. *IEEE Trans. Inform. Theory*, 48(12):3017–3028, 2002.

[14] C. Di, D. Proietti, I.E. Telatar, T.J. Richardson, and R. Urbanke. Finite-length analysis of low-density parity-check codes on the binary erasure channel. *IEEE Trans. Inform. Theory*, 48(6):1570–1579, 2002.

[15] T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47(2):619–637, 2001.

[16] 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 Journal on Selected Areas in Communications*, 19(5):924–934, 2001.

[17] F. Lehmann and G.M. Maggio. Analysis of the iterative decoding of LDPC and product codes using the Gaussian approximation. *IEEE Trans. Inform. Theory*, 49(11):2993–3000, 2003.

[18] R. Nuriyev and A. Anastasopoulos. Capacity-approaching code design for the noncoherent AWGN channel. In *IEEE GLOBECOM*, pages 1598–1602. San Francisco, USA, 2003.

[19] P. Berlin and D. Tuninetti. LDPC codes for fading Gaussian broadcast channels. *IEEE Trans. Inform. Theory*, 51(6):2173–2182, 2005.

[20] S.Y. Chung, G.D. Forney Jr, T.J. Richardson, R. Urbanke, A. Inc, and M.A. Chelmsford. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Commun. Letters*, 5(2):58–60, 2001.

[21] M.M. Mansour and N.P. Shanbhag. Low-power VLSI decoder architectures for LDPC codes. In *Proc. Int. Symp. Low Power Electronics and Design*, pages 284–289. New York, USA, 2002.

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

[23] T.S. Rappaport. *Wireless Communications: Principles and Practice*. IEEE Press Piscataway, NJ, USA, 1996.

[24] Y.A. Kuznetsov. *Elements of Applied Bifurcation Theory*. Springer, 2004.

[25] G.F. Franklin. *Feedback Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1993.

[26] T. Richardson. Error floors of LDPC codes. In *Proc. 41st Annu. Allerton Conf. on Communication, Control, and Computing*, pages 1426–1435. Urbana-Champaign, USA, 2003.

[27] 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, 2001.

[28] C.H. Hsu and A. Anastasopoulos. Capacity achieving LDPC codes through puncturing. *IEEE Transactions on Information Theory*, 54(10):4698–4706, 2008.

[29] J. Li, K.R. Narayanan, E. Kurtas, and C.N. Georghiades. On the performance of high-rate TPC/SPC codes and LDPC codes overpartial response channels. *IEEE Trans. Commun.*, 50(5):723–734, 2002.

[30] J. Haghighat, S.H. Jamali, H. Behroozi, and M. Nasiri-Kenari. High-performance LDPC codes for CDMA applications. In *Proc. 4th International Workshop on Mobile and Wireless Communications Network*, pages 105–109. Stockholm, Sweden, 2002.

[31] F. Kienle, T. Brack, and N. Wehn. A synthesizable IP core for DVB-S2 LDPC code decoding. In *Proc. Design, Automation and Test in Europe*, pages 100–105. Munich, Germany, 2005.

[32] T. Brack, M. Alles, F. Kienle, and N. Wehn. A synthesizable IP Core for WiMax 802.16e LDPC code decoding. In *Proc. Personal Indoor and Radio Communications Conference*, pages 1–5. Helsinki, Finland, 2006.

[33] M. Lei, I. Lakkis, T. Baykas, C.S. Sum, H. Harada, and S. Kato. Fixed point decoding performance of short-Length structured LDPC codes for SC-FDE based 60-GHz WPAN (IEEE 802.15. 3c). In *IEEE Vehicular Technology Conference*, pages 1841–1845. Marina Bay, Singapore, 2008.

[34] A. Serener, B. Natarajan, and DM Gruenbacher. Lowering the error floor of optimized short-block-length LDPC-coded OFDM via spreading. *IEEE Transactions on Vehicular Technology*, 57(3):1646–1656, 2008.

[35] Y. Mao and A.H. Banihashemi. A heuristic search for good low-density parity-check codes at short block lengths. In *Proc. Int. Conf. Commun.*, pages 41–44. Helsinki, Finland, 2001.

[36] G.D. Forney Jr. On iterative decoding and the two-way algorithm. In *Proc. Int. Symp. Turbo Codes and Related Topics*, pages 12–15. Brest, France, 1997.

[37] F.R. Kschischang and B.J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *IEEE Journal on Selected Areas in Communications*, 16(2):219–230, 1998.

[38] Y. Mao and A.H. Banihashemi. Decoding low-density parity-check codes with probabilistic scheduling. *IEEE Communications Letters*, 5(10):414–416, 2001.

[39] H. Xiao and A.H. anihashemi. Graph-based message-passing schedules for decoding LDPC codes. *IEEE Trans. Commun.*, 52(12):2098–2105, 2004.

[40] M.P.C. Fossorier and S. Lin. Soft-decision decoding of linear block codes based on ordered statistics. *IEEE Trans. Inform. Theory*, 41(5):1379–1396, 1995.

[41] M.P.C. Fossorier. Iterative reliability-based decoding of low-density parity check codes. *IEEE Journal on Selected Areas in Communications*, 19(5):908–917, 2001.

[42] T. Richardson. The geometry of turbo-decoding dynamics. *IEEE Trans. Inform. Theory*, 46(1):9–23, 2000.

[43] L. Kocarev, Z. Tasev, and A. Vardy. Improving turbo codes by control of transient chaos in turbo-decoding algorithms. *Electronics Letters*, 38(20):1184–1186, 2002.

[44] D.J. Watts and S.H. Strogatz. Collective dynamics of small-world networks. *The Structure and Dynamics of Networks*, 393:440–442, 2006.

[45] A.L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[46] L. Chen, J. Lu, and J. Lu. Synchronization of the time-varying discrete biological networks. pages 2650–2653. New Orleans, USA, 2007.

[47] B. Gou, H. Zheng, W. Wu, and X. Yu. Probability distribution of blackouts in complex power networks. In *Proc. IEEE Int. Symp. Circ. Syst.*, pages 69–72. New Orleans, USA, 2007.

[48] X.F. Wang and G. Chen. Synchronization in scale-free dynamical networks: robustness and fragility. *Arxiv preprint cond-mat/0105014*, 2001.

[49] X. Li and G. Chen. Synchronization and desynchronization of complex dynamical networks: an engineering viewpoint. *IEEE Trans. Circ. and Syst. Fund. Theor. Appl.*, 50(11):1381–1390, 2003.

[50] C.W. Wu, I.B.M.T.J.W.R. Center, and N.Y. Yorktown Heights. Synchronization in systems coupled via complex networks. In *Proc. IEEE Int. Symp. Circ. Syst.*, pages 724–727. Vancouver, Canada, 2004.

[51] Z. Li and G. Chen. Global synchronization and asymptotic stability of complex dynamical networks. *IEEE Trans. Circuits and Systems II*, 53(1):28–33, 2006.

[52] G.P. Jiang, W.K.S. Tang, and G. Chen. A state-observer-based approach for synchronization in complex dynamical networks. *IEEE Trans. on Circ. and Syst. Fund. Theor. Appl.*, 53(12):2739–2745, 2006.

[53] R. Cohen and S. Havlin. Scale-free networks are ultra small. *Physical Review Letters*, 90(5):058701.1–058701.4, 2003.

[54] J.A. McGowan and R.C. Williamson. Loop removal from LDPC codes. In *Proc. IEEE Inform. Theory Workshop*, pages 230–233. Paris, France, 2003.

[55] X.Y. Hu, E. Eleftheriou, and D.M. Arnold. Regular and irregular progressive edge-growth tanner graphs. *IEEE Trans. Inform. Theory*, 51(1):386–398, 2005.

[56] O. Milenkovic, E. Soljanin, and P. Whiting. Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles. *IEEE Trans. Inform. Theory*, 53:39–55, 2007.

[57] T. Tian, C.R. Jones, J.D. Villasenor, and R.D. Wesel. Selective avoidance of cycles in irregular LDPC code construction. *IEEE Trans. Commun.*, 52(8):1242–1247, 2004.

[58] A. Ramamoorthy and R. Wesel. Construction of short block length irregular low-density parity-check codes. In *Proc. IEEE Int. Conf. Commun.* Paris, France, 2004.

[59] G. Liva, W.E. Ryan, and M. Chiani. Quasi-cyclic generalized LDPC codes with low error floors. *IEEE Transaction on Communication*, 56(1):49–57, 2008.

[60] S. Song, D. Hwang, S. Seo, and J. Ha. Linear-time encodable rate-compatible punctured LDPC codes with low error floors. In *IEEE Vehicular Technology Conference*, pages 749–753. Marina Bay, Singapore, 2008.

[61] Y. Han and W.E. Ryan. LDPC decoder strategies for achieving low error floors. In *Information Theory and Applications Workshop*, pages 277–286. Porto, Portugal, 2008.

[62] H. Xiao and A.H. Banihashemi. Improved progressive-edge-growth (PEG) construction of irregular LDPC codes. *IEEE Communications Letters*, 8(12):715–717, 2004.

[63] G. Richter and A. Hof. On a construction method of irregular LDPC codes without small stopping sets. In *Proc. IEEE Int. Conf. Commun.*, pages 1119–1124. Istanbul, Turkey, 2006.

[64] D.J.C. MacKay and M.S. Postol. Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes. *Electronic Notes in Theoretical Computer Science*, 74:97–104, 2003.

[65] M. Ivkovic, SK Chilappagari, and B. Vasic. Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers. *IEEE Transactions on Information Theory*, 54(8):3763–3768, 2008.

[66] D. Lu and K. Yao. Improved importance sampling technique for efficient simulation of digital communication systems. *IEEE Journal on Selected Areas in Communications*, 6(1):67–75, 1988.

[67] E. Cavus, C.L. Haymes, and B. Daneshrad. An IS simulation technique for very low BER performance evaluation of LDPC code. In *Proc. IEEE Int. Conf. Commun.*, pages 1095–1100. Istanbul, Turkey, 2006.

[68] B. Xia and W.E. Ryan. On importance sampling for linear block codes. In *Proc. IEEE Int. Conf. Commun.* Anchorage, USA, 2003.

[69] R. Holzlohner, A. Mahadevan, C.R. Menyuk, J.M. Morris, and J. Zweck. Evaluation of the very low BER of FEC codes using dual adaptive importance sampling. *IEEE Commun. Letters*, 9(2):163–165, 2005.

[70] L. Dolecek, Z. Zhang, M. Wainwright, V. Anantharam, and B. Nikolic. Evaluation of the low frame error rate performance of LDPC codes using importance sampling. In *Proc. IEEE Inform. Theory Workshop*, pages 202–207. Lake Tahoe, USA, 2007.

[71] G. Li and G. Feng. Weighted IS method of estimating FER of LDPC codes in high SNR region. In *Proc. Sixth International Conference on Networking*, pages 91–95. Sainte-Luce, Martinique, 2007.

[72] X. Hu, B.V.K Kumar, Z. Li, and R. Barndt. Error floor estimation of long LDPC codes on partial response channels. In *Proc. IEEE GLOBECOM*, pages 259–264. Washington DC, USA, 2007.

[73] C.A. Cole, S.G. Wilson, E.K. Hall, and T.R. Giallorenzi. A general method for finding low error rates of LDPC codes. *Arxiv preprint cs/0605051*, 2006.

[74] P.D. Lee, L.Z. Zhang, V.B. Anantharam, and M.J. Wainwright. Error floors in LDPC codes: fast simulation, bounds and hardware emulation. In *IEEE International Symposium on Information Theory*, pages 444–448. Toronto, Canada, 2008.

[75] V. Anantharam M.J. Wainwright L.Dolecek, Z.Y. Zhang and B. Nikolić. Analysis of absorbing sets and fully absorbing sets of array-based ldpc codes. *http://www.eecs.berkeley.edu/ ananth/2008+/arrayITsub.pdf*, 2008.

[76] J.C. Chen, D. Lu, J.S. Sadowsky, and K. Yao. On importance sampling in digital communications. I. Fundamentals. 11(3):289–299, 1993.

[77] P.J. Smith and H.S. Gao. Quick simulation: a review of importance sampling techniques in communications systems. *IEEE Journal on Selected Areas in Communication*, 15(4):597–613, 1997.

[78] S. Lin and D.J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983.

[79] W.E. Ryan. An Introduction to LDPC codes. *CRC Handbook for Coding and Signal Processing for Recording Systems, (B.Vasic, ed.), CRC Press*.

[80] A. Shokrollahi. LDPC codes: an introduction. *Coding, cryptography and combinatorics*, 23:85–110.

[81] J. Zhang and M.P.C. Fossorier. A modified weighted bit-flipping decoding of low-density parity-check codes. *IEEE Communications Letters*, 8(3):165–167, 2004.

[82] N. Miladinovic and M.P.C. Fossorier. Improved bit-flipping decoding of low-density parity-check codes. *IEEE Trans. Inform. Theory*, 51(4):1594–1606, 2005.

[83] R.F. Lucas and M.P.C.Y.K.S. Lin. Iterative decoding of one-step majority logic deductible codes based on belief propagation. *IEEE Trans. Commun.*, 48(6):931–937, 2000.

[84] J.G. Proakis. *Digital communications*. Third edition, McGraw-Hill, Singapore, 1995.

[85] C. Schlegel and L. Perez. *Trellis and Turbo Coding*. Wiley-IEEE Press, Mar. 2004.

[86] A. Ashikhmin, G. Kramer, and S. ten Brink. Extrinsic information transfer functions: a model and two properties. In *Proc. Conf. Inform. Sciences and Systems*, pages 742–747. Princeton, USA, 2002.

[87] S. ten Brink. Designing iterative decoding schemes with the extrinsic information transfer chart. *AEU Int. J. Electron. Commun*, 54(6):389–398, 2000.

[88] M. Ardakani and F.R. Kschischang. Designing irregular LPDC codes using EXIT charts based on message error rate. In *Proc. IEEE Int. Symp. Inform. Theory*, page 454. Lausanne, Switzerland, 2002.

[89] H. Jin and T.J. Richardson. Fast density evolution. In *Proc. 38th Conf. Inform. Sciences and Systems*. New Jersey, USA, 2004.

[90] A. W. Eckford. Andrew eckford's software. *http://www.comm.toronto.edu/ eckford/software/index.html*, 2005.

[91] T.J. Richardson and R.L. Urbanke. Efficient encoding of low-density parity-check codes. *IEEE Trans. Inform. Theory*, 47(2):638–656, 2001.

[92] M.R. Yang and W.E.Y. Li. Design of efficiently encodable moderate-length high-rate irregular LDPC codes. *IEEE Trans. Commun.*, 52(4):564–571, 2004.

[93] G.D. Birkhoff. *Dynamical systems*. American mathematical society Providence, 1927.

[94] J. Guckenheimer and P. Holmes. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*. Springer, 1983.

[95] A.A. Tsonis. *Chaos: from theory to applications*. Plenum Publishing Corporation, 1992.

[96] S.H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Perseus Books Group, 2000.

[97] D.J.C Mackay. Encyclopedia of sparse graph codes. *http://wol.ra.phy.cam.ac.uk/mackay/codes/data.html*, 2005.

[98] X. Zheng, F.C.M. Lau, C.K. Tse, and S.C. Wong. Study of nonlinear dynamics of LDPC decoders. In *Proc. European Conference on Circuit Theory and Design*, pages 157–160. Cork, Ireland, 2005.

[99] P. Parmananda, M. Hildebrand, and M. Eiswirth. Controlling turbulence in coupled map lattice systems using feedback techniques. *Physical Review E*, 56(1):239–244, 1997.

[100] M. de Sousa Vieira and A.J. Lichtenberg. Controlling chaos using nonlinear feedback with delay. *Physical Review E*, 54(2):1200–1207, 1996.

[101] P. Erdos and A. Renyi. On Random Graphs. *Publ. Math. Debrecen*.

[102] J.L. Guillaume and M. Latapy. A realistic model for complex networks. *Arxiv preprint cond-mat/0307095*, 2003.

[103] M. Chiani and A. Ventura. Design and performance evaluation of some high-rate irregular low-density parity-check codes. In *IEEE GLOBECOM*, pages 990–994. Texas,USA, 2001.

[104] W.Y. Weng, A. Ramamoorthy, and R.D. Wesel. Lowering the error floors of irregular high-rate LDPC codes by graph conditioning. In *Proc. IEEE 60th Vehicular Technology Conference*, pages 2549–2553. Los Angeles, USA, 2004.

[105] L. Dinoi, F. Sottile, S. Benedetto, I. Superiore, M. Boella, and I. Torino. Design of variable-rate irregular LDPC codes with low error floor. In *Proc. IEEE Int. Conf. Commun.*, pages 647–651. Seoul, Korea, 2005.

[106] H. Jin, A. Khandekar, and R. McEliecet. Irregular repeat-accumulate codes. In *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, pages 1–8. Brest, France, 2002.